RAPPORT FINAL

Initiation à l'intelligence artificielle

Année universitaire 2022 - 2023

https://github.com/arcadiumstreet/projet-Titan.git

PIERROT

Réalisé par :
Choc Noé
Genevey Coemgen
Brochud Corentin
Flin Alexandre

Encadré par :
Damien PELLIER

Sommaire :

I-Introduction

Dans ce rapport, nous allons vous présenter notre projet d'Intelligence Artificielle et vous expliquer comment nous avons procédé tout au long du semestre.

L'objectif de ce projet est de programmer un robot capable d'attraper des palets en bois puis de les amener dans la zone de but adverse. Le robot doit donc être capable de repérer les palets, de les attraper et de les amener dans la zone de but. Cette zone est délimitée par une ligne blanche et les palets sont placés sur des intersections de lignes de couleurs. Le robot doit donc être capable de se repérer de façon autonome sur le terrain pour pouvoir marquer le plus de palet possible.

II-Phase de compréhension et planification du projet

Organisation et compréhension du projet

Au début du projet, nous avons regardé la librairie permettant de programmer le robot ainsi que les règles de la compétition et du projet afin de mieux comprendre ce que l'on devait rendre. Nous avons ensuite organisé la création du code et des différents livrables à travers un échéancier qui figure à la fin de cette partie ou dans le plan de développement. Enfin, nous avons réalisé un cahier des charges afin de bien identifier les besoins et objectifs de ce projet. Nous avons donc réfléchi aux premières méthodes à créer ainsi qu'à la structure du code.

Risques, contraintes et solutions possibles

Ce projet comporte principalement deux types de risques. Ceux humains regroupant les possibilités qu'une ou plusieurs personnes soient absentes ou souffrant de problèmes de santé les empêchant de travailler. Ainsi que les risques matériels comprenant : la perte ou casse d'élément du robot fournit pour le projet et incident technique inattendu lors de la compétition.

Nous avons également différentes contraintes :

- temporelle : nous disposons de 12 semaines pour réaliser ce projet, du 5 septembre au 28 novembre où la compétition aura lieu. L'ensemble des documents doivent être rendus la semaine suivante, le 5 décembre.
- matérielles : nous devons coder en langage java sous éclipse un programme exécutable par la machine Lejos EV3 pour faire fonctionner le robot. Le robot est imposé avec sa géométrie, ses composants et ses défauts. Tous les documents doivent être sur le Git du groupe.
- humaines : le groupe de travail est limité à 4 personnes il faut donc s'organiser pour travailler efficacement et respecter la contrainte temporelle.
- fonctionnelle : une fois une manche de la compétition débutée, le robot doit être autonome, respecter les règles du jeu et anticiper les situations qu'il pourrait rencontrer.

La création d'un échéancier, ainsi qu'une bonne organisation du travail et une communication via discord et GitHub permettront de limiter les risques humains et respecter les contraintes de temps et humaine. Pour ce qui est des risques matériels, à

l’exception de la brique Lejos EV3 les éléments défectueux ou abîmés du robot peuvent être remplacés si signalés. De plus, si le robot rencontre un problème durant la première minute d’une manche, un temps mort peut être demandé afin de réparer cette anomalie.

Moyens technique utilisé pour la réalisation du projet

Le robot fourni est équipé de 3 moteurs EV3 Large Regulated Motor, un pour la gestion des pinces et 2 autres pour les roues droite et gauche. Il possède également 3 capteurs différents: un capteur de distance ultrasonique EV3UltrasonicSensor, un capteur tactile EV3TouchSensor ainsi qu’un capteur de couleur EV3ColorSensor. Ces éléments sont mis en relation au travers d’une brique programmable embarqué LEGO EV3 Brick.

Le programme a été réalisé sur l’environnement de développement intégré en langage Java, Eclipse. Il a aussi été nécessaire d’y installer le plugin LeJos EV3 qui nous donne accès à la bibliothèque LeJos permettant de faire fonctionner le robot. Afin de faciliter la mise en commun du travail de chacun, le service GitHub a été utilisé, permettant ainsi de stocker en ligne le code source du programme ainsi que tout autre document nécessaire au projet, pour un accès et une mise à jour plus efficace.

III - Phase de conception et de réalisation

Définition des méthodes importantes

Robot:

public void allerjusqua(String couleur) :

Cette méthode fait avancer le robot devant lui. S'il perçoit autre chose qu'un palet, il contourne l'obstacle par la droite et continue d'avancer, s'il capte la ligne de couleur entrée en paramètre, il s'arrête.

public void catchTarget(float targetDistance) :

Cette méthode permet au robot d'attraper un palet à une distance donnée. Il commence d'abord par s'approcher du palet présupposé. Si effectivement c'est un palet, il l'attrape, s'il se rend compte que ce n'est pas un palet, le robot recule et revient à sa position initiale.

public void research() :

Le robot tourne sur lui-même jusqu'à détecter un objet à moins de 50 cm.

public void goal(boolean b) :

Le robot fonce déposer le palet qu'il tient dans ses pinces derrière la ligne blanche.

public void erreurs_boussole() :

Lors de ses déplacements, le robot dérive un peu par rapport à sa boussole et engendre alors de grosses erreurs dans les angles de rotation. Cette méthode, qui est appelée après avoir marqué un palet, permet de réduire ces erreurs en s'orientant par rapport au mur d'en face.
Le robot tourne face au mur et calcule quel angle permet de placer le capteur ultrason à la distance minimale entre le robot et le mur. Ce qui, en théorie, incline le robot perpendiculairement au mur.

public boolean estunpalet() :

Cette méthode teste si l'objet capté est un palet ou non.

public static void AllerAZone(int zone) :

Les coordonnées des zones de recherche sont prédéfinies dans le tableau *researchArea*. Toutes ces zones sont numérotées de 1 à 6. Le robot utilise la méthode *goTo(double largeurF, double longueurF)* de la classe *MotorWheels* pour se déplacer à la zone de recherche dont le numéro a été passé en paramètre.

public static void alleraupalet(int i) :

Les coordonnées des palets sont prédéfinies dans le tableau *researchArea*.
Tous les palets sont numérotés de 1 à 9. Le robot utilise la méthode *goTo(double largeurF, double longueurF)* de la classe *MotorWheels* pour se déplacer au palet dont le numéro a été passé en paramètre.

Cerveau:

public static void strategie1(Robot p, int d) :

Le robot va marquer ses deux premiers palet. Son chemin est codé en dur pour être le plus rapide et marquer le premier palet avant l’adversaire. Ce chemin est déterminé par le placement initial du robot qui dépendra du placement de l’adversaire. A la fin de cette méthode, le robot se trouve proche de la ligne blanche dans le camp de l’adversaire.

public static void strategies 1a(Robot p,int d) :

La stratégie 1a est la suite de la stratégie 1. Le robot va marquer les deux palets suivants en fonction de son placement initial dans le cas où l'adversaire commence au milieu.

public static void strategies 1b(Robot p, int d) :

La stratégie 1a est la suite de la stratégie 1. Le robot va marquer les deux palets suivants en fonction de son placement initial dans le cas où l'adversaire commence du côté opposé.

public static void strategie2(Robot p,int d,int palet1,int palet2) :

Le robot va attraper et marquer les palets entrés en paramètre de cette méthode.

public static void strategie3(Robot p,int zone) :

Le robot applique la stratégie 3 en partant de la zone entrée en paramètre.

MotorWheels:

public void maj_longueur_largeur(double distance) :

Le robot calcule sa position en longueur et en largeur par rapport à la distance entrée en paramètre et de sa boussole interne.

public void majBoussole(double i) :

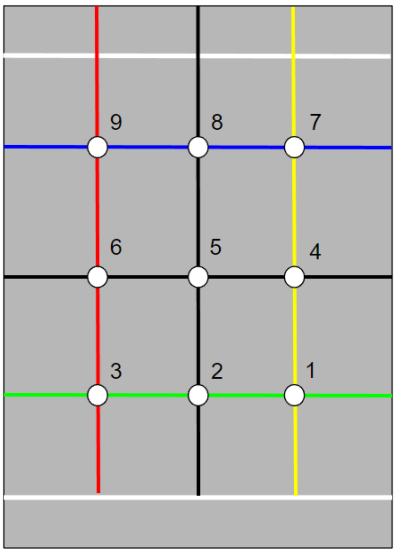
Cette méthode met à jour la boussole par rapport à l'angle avec lequel il a tourné précédemment.

public void goTo(double largeurF, double longueurF) :

Le robot se déplace à la coordonnée indiquée par la largeur et la longueur entrée en paramètre.

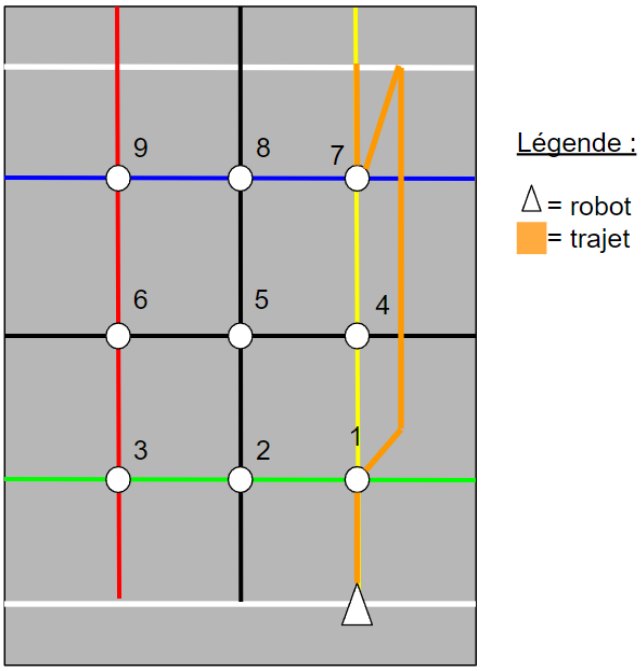
Planification des différentes stratégies

Pour le tournoi nous avons développé plusieurs stratégies qui se basent sur les positionnements des palets que nous avons numérotés de 1 à 9.



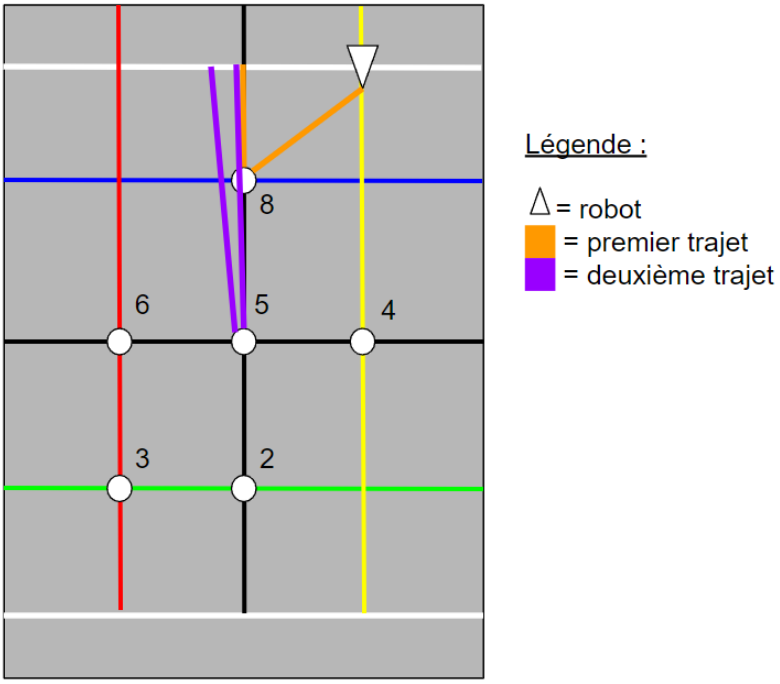
Premièrement, nous avons créé une stratégie que l'on utilise au début d'un match. Celle-ci a pour but de marquer 2 palets le plus rapidement possible. Pour cela, le robot prend en compte le côté sur lequel il part (gauche, milieu ou droite). Cette stratégie nous permet donc de potentiellement marquer le premier palet qui rapporte des points bonus et de marquer un deuxième palet assez rapidement.

Exemple : départ du robot à droite

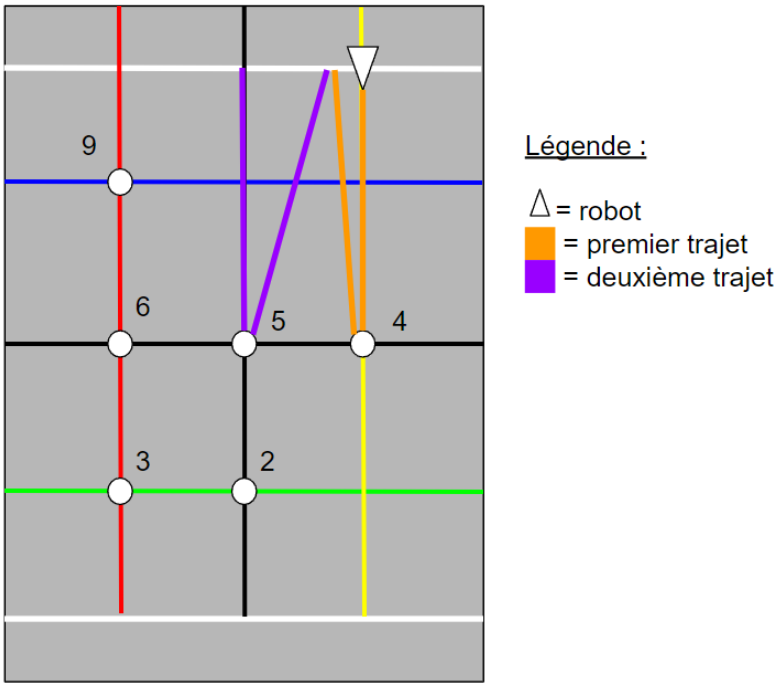


Après cette première stratégie, le robot lance une stratégie qui suit la première en fonction du positionnement de l'adversaire.

En effet, si l'adversaire part à l'opposé de notre robot, le palet n°8 sera alors encore à sa place et notre robot ira donc le chercher. Ensuite, il ira vers le palet n°5 et vérifiera à mi-chemin si celui-ci est toujours présent. Si il ne l'est pas il ira donc soit au palet n°4, soit au palet n°6 en fonction d'où il est parti à l'origine.



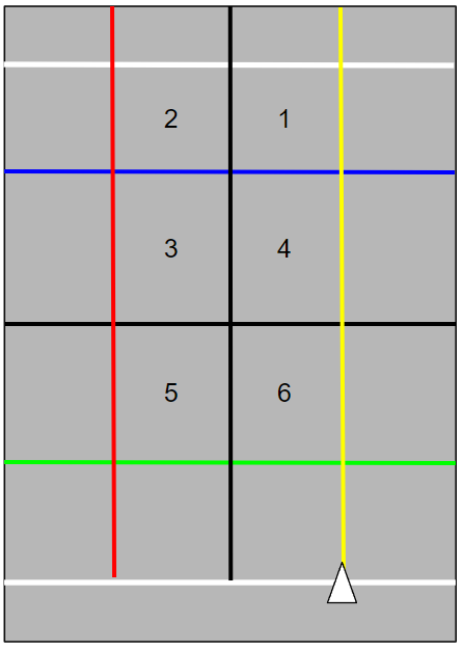
En revanche, si l'adversaire place son robot au milieu, le palet n°8 ne sera logiquement plus disponible donc notre robot ira récupérer soit le palet n°4 soit le palet n°6 encore une fois en fonction du positionnement au départ de la manche. Puis, comme décrit précédemment, il essayera de récupérer le palet n°5.



Notre deuxième stratégie est une stratégie que nous appelons après un temps mort. En effet, dans cette stratégie, le robot nous demande quels palets sont encore présents sur le

terrain. Ensuite il nous demande un premier palet qu'il ira chercher puis un deuxième qu'il marquera après le premier. Pour que cette stratégie fonctionne, il faut que les palets présents sur le terrain soient encore placés sur les intersections puisque le robot se déplacera aux coordonnées correspondantes aux palets à aller chercher.

Enfin la dernière stratégie est une stratégie automatiquement appelée après les deux stratégies précédentes ou appelée après un temps mort quand il n'y à plus de palets situés sur des intersections.
Cette stratégie divise le plateau en 6 zones de recherches.



Le robot va donc lors de cette stratégie aller sur la zone n°1 et effectuer une recherche, s' il trouve un palet alors il ira l'attraper et le marquer, sinon il ira sur la zone n°2 et ainsi de suite. Cela permet donc au robot d'être sûr de trouver les palets restants s' ils ont été déplacés de leur position initiale.

IV - Phases de test

Test des principales méthodes

D'après le plan de test, nous avons commencé par les méthodes "classiques" qui sont avancées, reculer, tourner, ouvrir et fermer les pinces.
Mais nous sommes vite arrivés à un problème, toutes les méthodes de bases ne sont pas égales à la réalité (Exemple : méthodes forward (1000)Avancer de 66 cm) donc nous avons vite conclu qu'il fallait rajouter un coefficient sur les méthodes pour avancer (Idem pour les rotations (rotate(180) Tourne de 126 degrés.).
La méthode "boussole_a_0()" permet donc d'orienter le robot face à la ligne de but adverse, nous avons dû changer le coefficient de 12 %, car cette méthode est appelée quand le robot a un palet dans les pinces donc cela augmente les frottements.
"ouvrir(boolean)" et "fermer(boolean)", nous utilisons la méthode rotate pour ouvrir et fermer les pinces (en fonction du signe du paramètre), nous avons essayé d'optimiser l'ouverture et donc le temps entre l'ouverture et la fermeture des pinces et on a conclu à une rotation optimale.
"getColor", nous avons surtout eu des problèmes au niveau du code et non au niveau des tests.
La méthode "distance()" a été testé et les données étaient cohérentes.

Test en enchaînement

Puis, nous avons testé les méthodes qui appelaient plusieurs méthodes dans leur structure. Donc tester principalement l'enchaînement de celles-ci qui sont en asynchrone ou non.
Les quatre principales méthodes utilisées sont :

- goTo(double, double)

Les erreurs étaient principalement au niveau du code.
Mais dès que les angles calculés était bon le robot allait bien au point indiqué (en asynchrone car cela réduisait l'erreur).

- catchTarget(float)

Les principales erreurs surtout sur les premiers c'était les choix sur la asynchrone ou la synchrone des méthodes appelés.
L'ajout aussi d'une méthode pour savoir si l'objet capté est un palet permet de revenir à sa position initiale ce qui réduit les erreurs lié au erreur du capteur ultrason et donc aussi d'éviter de rentrer dans les murs.

- goal(boolean)

Permet de remettre le robot face à la zone de but,avance jusqu'à atteindre la ligne blanche adverse.La méthode entre en paramètre un booléen celui sert de savoir si le robot fait une correction d'erreur de la boussole car sur les premiers déplacements, le robot ne fait aucune erreurs donc le robot ne fait pas la correction.

Puis le robot ouvre les pinces puis recule pour se libérer complètement de la prise du palet et met aussi à jour la distance car nous avons remarqué que le centre du robot n'est pas au niveau du capteur mais au niveau du des roues(différence de 7 cm).

Les problèmes rencontrés sont surtout les problèmes de synchrone car parfois après s'être arrêter le robot amène le palet car l'ouverture des pinces était en synchrone.

Donc nous avons dû faire des tests afin d'optimiser les déplacements ,les déplacements que nous devons faire en même temps et ce que nous pouvons faire indépendamment.

- research()

Permet de faire une recherche sur le terrain et s'arrête dès que le robot capte un palet et puis aussi fait un contrôle si c'est bien un palet car cela peut être un mur ou un robot adverse.Donc pouvoir avoir un mise a jour cohérente de la boussole nous avons dû tester chaque durée par rapport à la rotation donc nous avons divisé ses zones de recherche en 4 zones et fait les test sur ses zones (environ une quinze test sur chaque).Puis nous avons conclu a un coeff sur chaque zone. Cela permet de connaître l'angle de rotation du robot.

Une fois,ces test effectués ,nous avons estimé la distance maximale que le robot va capter et nous avons conclu à une distance de 50 cm et puis le capteur ultrason peut capter les palet a partir de 10 cm sur notre robot donc sa zone de recherche s'étend sur 40 cm.

V - Déroulement et résultats de la compétition

Nous avons ramassé 4 palet durant la phase de qualification.
les match de poules :
Match 1 perdant 11 à 12 contre Nono
Match 2 gagnant 11 à 3 contre Pollux
Match 3 perdant 11 à 12 contre Kermit
Les phases finales :
Demi-final gagnant 11 à 9 contre Ariane
Final gagnant 14 à 12 contre Kermit
“Victoire de la compétition”
En résumé, nous avons gagné 3 matchs et perdu 2 matchs avec une moyenne de points marqués de 11.6 points (au moins 3 palets et 100% des premiers palets sur tous les matchs).
Pour la finale, le match était très disputé avec un blocage de l'adversaire sur le dernier palet qui nous aurait enlevé la victoire.

VI - Bilan

Point a améliorer pour un futur projet

Les différents points que l’on peut améliorer seraient principalement l’utilisation des états qu’on a décidé de ne pas utiliser durant ce projet, ça nous aurait permis de simplifier le code surtout sur le main et les stratégies.
Les corrections d’erreurs surtout au niveau de la longueur et de la largeur, car nous corrigeons tout le temps la boussole, mais le robot se décalait sur la gauche donc ce qui faussait le calcul du prochain angle et de la prochaine coordonnée entré en paramètre.

Point positif de notre projet

Les points positifs sont principalement la victoire de la compétition. Cela signifie que nous sommes arrivés à tirer votre épingle du jeu. La stratégie de se concentrer sur le premier palet a été bénéfique, cela nous a permis de faire la différence notamment en final.
Notre jeu était aussi basé sur la vitesse au démarrage ce qui créait parfois des erreurs en ligne droite, car l’un des moteurs fonctionnait légèrement moins bien.
L’utilisation de la méthode “goTo” qui est une classe très développée et qui permet d’aller à un point précis sur le plan de jeu.
La méthode “research” est aussi performante avec le calcul de l’angle de recherche en fonction du temps de recherche.

VII - Annexes :

Annexe I : Cahier des charges.
    Voir le github
Annexe II : Échéancier

Liste des tâches	Semaine 1 05/09/2022	Semaine 2 12/09/2022	Semaine 3 19/09/2022	Semaine 4 26/09/2022	Semaine 5 03/10/2022	Semaine 6 10/10/2022	Semaine 7 17/10/2022	Semaine 8 24/10/2022	Semaine 9 07/11/2022	Semaine 10 14/11/2022	Semaine 11 21/11/2022	Semaine 12 28/11/2022
Préparation du projet												
Définition des objectifs												
Analyse du besoin												
Découverte de la librairie leJos												
Création du git et du projet informatique												
Rédaction du cahier des charges												
Développement du projet												
Rédaction du wiki												
Création des classes et méthodes												
Réalisation des stratégies												
Rédaction de la javadoc												
Tests des méthodes												
Tests des stratégies												
évaluation												

Annexe III : Plan de test

Voir le github

Annexe IV : Plan de développement

Voir le github