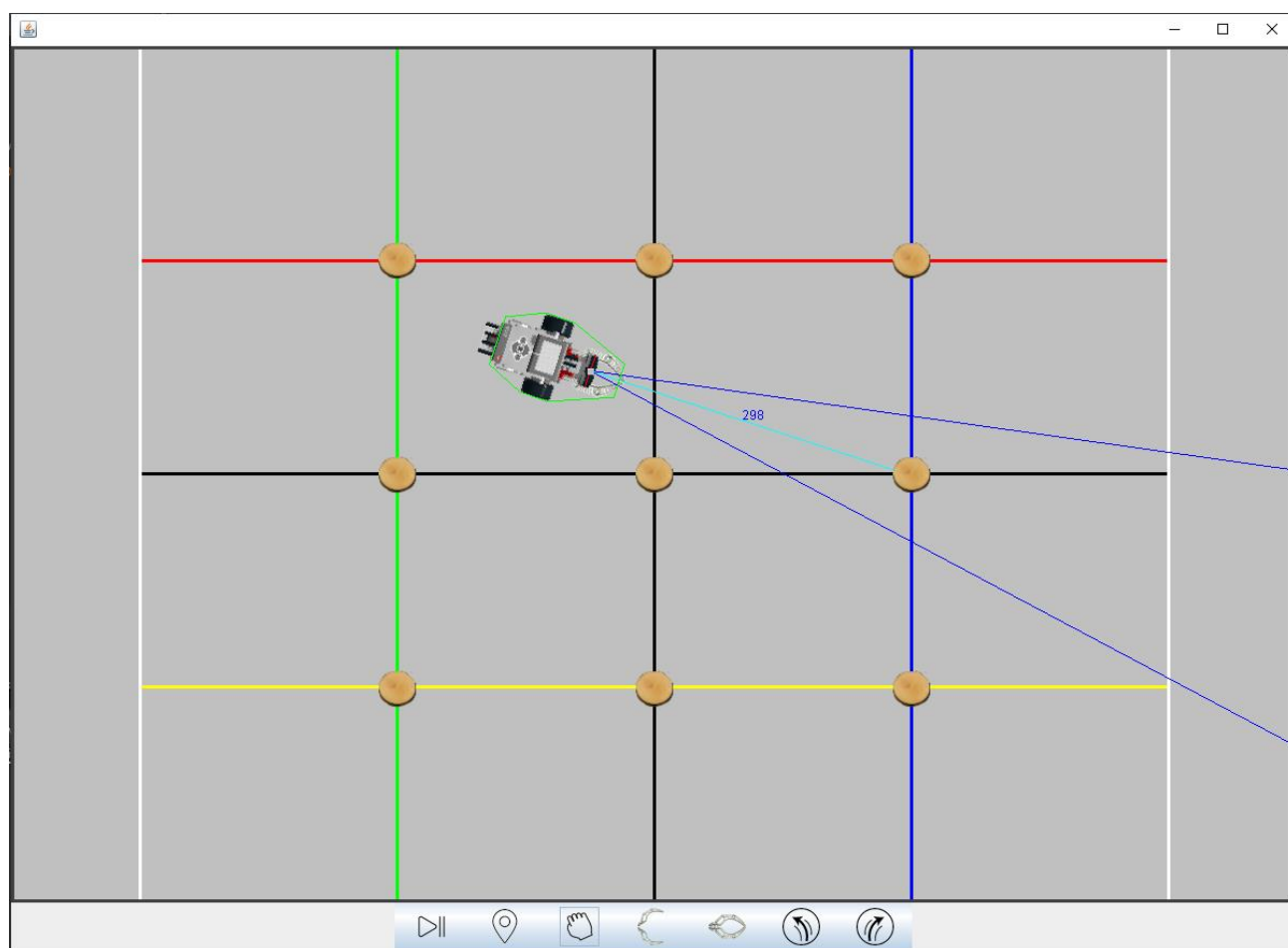


RAPPORT DE PROJET

Projet de modélisation graphique d'un robot LEGO ramasseur de palets



Capture d'écran issue de l'application

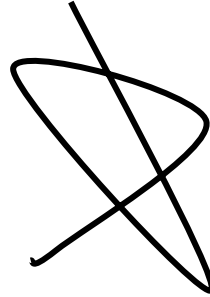
Charte anti-plagiat

Nous soussigné.e.s FLEURY Pierre et GATTACIECCA Bastien avons pris connaissance des obligations décrites dans la charte anti-plagiat, et nous nous engageons à nous y conformer strictement.

Le 21/12/2020,

A complex, stylized handwritten signature in black ink, featuring multiple loops and a long horizontal stroke extending to the left.

GATTACIECCA Bastien

A stylized handwritten signature in black ink, consisting of a large loop with a diagonal stroke crossing through it.

FLEURY Pierre

Remerciements

Nous tenons à remercier tout particulièrement notre tuteur de projet M. Benoît LEMAIRE qui a accepté de nous aiguiller tout au long du projet et nous permettre d'arriver à ce résultat. Nous remercions également chaque lecteur qui prendra le temps de s'intéresser à nos travaux, que ce soit pour la critique du code ou concernant la rédaction de nos rendus.

Table des matières

Introduction	3
Besoins & objectifs.....	3
Contexte	3
Explications sur la définition de modélisation	3
Historique de la modélisation jusqu'à "de nos jours"	3
En quoi la modélisation s'inscrit-elle comme un sujet approprié dans le cadre de la filière ?.....	4
Présentation du sujet.....	4
Motivation & enjeux	5
Objectifs et contraintes	6
Objectifs et contraintes techniques.....	6
Délais	6
Gestion de projet.....	6
Planification.....	6
Échéancier	6
Plan de développement.....	7
Gestion	7
Diagramme UML.....	7
Cahier des charges	7
Répartition des tâches	7
Outils	8
Développement.....	8
Stratégies	8
Phase itérative 1 : Gestion des collisions	9
Phase itérative 2 : Vitesse et accélération.....	10
Phase itérative 3 : Optimisation et formes géométriques	11
Phase itérative 4 : Fonctionnement des capteurs	12
Phase itérative 5 : Variabilité des capteurs	13
Phase itérative 6 : Javadoc et visibilité	14
Bilan	15
FLEURY Pierre	15
GATTACIECCA Bastien	15
Conclusion.....	16
Critique.....	16
Ouvertures.....	16
Glossaire.....	17
Annexes.....	17

Introduction

Dans le cadre de la troisième année de licence Mathématiques et Informatique Appliqués aux Sciences Humaines et Sociales (MIASHS) il est proposé aux étudiants de réaliser un projet pour appliquer des compétences en gestion agile déjà mises à l'épreuve l'année passée (en L2).

Cependant, en plus de la méthode agile qui était demandée pour la gestion de projet, le résultat du travail rentre désormais davantage dans les attentes ; que ce soit dans l'utilité du livrable une fois réalisé ou bien dans les compétences transversales acquises/utilisées pour la réalisation du livrable.

Avant de commencer le projet, les premières étapes ont été de réunir des personnes autour d'un projet concernant la programmation. Puis après avoir pris plusieurs rendez-vous avec différents professeurs concernés pour proposer nos quelques idées (et pour en avoir d'autres également), nous avons proposé à Mme Frédérique BRENET le sujet suivant : "Projet de modélisation graphique d'un robot LEGO ramasseur de palets" ; et c'est à deux que nous avons géré l'ensemble du projet.

Les mots en gras dans ce rapport sont précisés dans le glossaire en dernière partie. Ils sont écrits dans l'ordre où ils apparaissent dans le document.

Besoins & objectifs

Contexte

Explications sur la définition de modélisation

La modélisation est utilisée dans notre projet notamment pour la simplification et la prédiction. Ainsi, on cherche à établir le modèle d'un système, notre robot LEGO physique, afin d'en faciliter l'étude, on parle de modèle descriptif, et à prévoir le comportement, on parle de modèle prédictif. En somme, il s'agit d'une simplification où l'on sélectionne les variables pertinentes. Par exemple, la masse du robot est une variable que l'on a évincée. Elle aurait pu être utile lors de la gestion des collisions.

La principale difficulté lorsque l'on souhaite correctement réaliser un modèle réside dans la complexité du système que l'on cherche à modéliser. Il paraît irréalisable de modéliser avec exactitude le système c'est pourquoi nous préciserons pour chaque implémentation l'écart qu'il existe entre la réalité et notre modèle, mais aussi, pourquoi on s'accorde cet écart.

Historique de la modélisation jusqu'à "de nos jours"

Même si cela ne paraît pas évident, l'Homme essaie de faire des modèles depuis longtemps. Entre le modèle et le chercheur (du modèle) il y a une dimension technique qui permet de faire la médiation. Par exemple, l'Homme qui dessine le soleil à l'aide de son crayon, même si l'exemple n'a pas de visée scientifique ici. L'importance grandissante des mathématiques en tant que discipline scientifique au début du XXème siècle permet à

beaucoup de disciplines d'autres disciplines de s'enorgueillir de la modélisation scientifique. De plus, si on ajoute à cela la montée en puissance de l'informatique dès le milieu du XXème siècle jusqu'à aujourd'hui, il en résulte que le processus de modélisation est au cœur de la science d'aujourd'hui.

En quoi la modélisation s'inscrit-elle comme un sujet approprié dans le cadre de la filière ?

Au vu du descriptif ci-dessus, il est opportun de dire que la modélisation est un sujet adéquat dans notre filière. En utilisant le robot LEGO comme modèle on crée un projet qui nous force à démontrer les compétences acquises (et en cours d'acquisition) dans plusieurs de nos cours, mais aussi nos compétences transversales. Il s'agit d'un projet pleinement pluridisciplinaire et chaque aspect sera détaillé plus loin dans le rapport. La base même du livrable est un code, donc implique de la programmation (orientée objet). On expliquera de quelle manière on a beaucoup utilisé la trigonométrie pour la simulation graphique (orienter les objets par exemple) ou encore la géométrie pour optimiser la gestion des collisions. On peut inclure les statistiques, utilisées pour inférer **la normalité d'une distribution**, ou encore inférer la courbe d'**accélération horaire** des moteurs du robot. Aussi, de la méthodologie expérimentale, dans une moindre mesure, pour récupérer des données à partir de tests réalisés sur le robot. Excepté la trigonométrie qu'on utilise très peu dans les sciences sociales, toutes les compétences utilisées sont enseignées dans la licence. Certaines notions ont dû être approfondies voire découvertes en amont ; comme par exemple les méthodes graphiques et le **multithreading** en programmation orientée objet que l'on n'aborde pas cette année.

Présentation du sujet

Un de nos cours intitulé *Initiation à l'intelligence artificielle* nous forme à la programmation d'un **système embarqué** sur un robot LEGO. L'objectif à terme est de confronter les robots de plusieurs groupes lors d'une compétition dont le but est de placer un maximum de palet dans le camp adverse. Toutes les précisions concernant le déroulement de ce projet ainsi que le règlement de la compétition sont accessibles via [le site de Damien PELLIER](#), professeur de ce cours. Un facteur de réussite dans cette compétition est sans aucun doute la stratégie implémentée qui va rendre la récupération de palets efficace ou non. Le but de ce projet est de réaliser un outil graphique, simulant les différentes interactions entre les objets sur le plateau (image ci-dessous) pour permettre à l'utilisateur de tester et comparer différentes stratégies. L'application permet cela et plus encore d'implémenter une stratégie différente pour chaque robot ajouté sur le plateau.

Le robot LEGO possède un environnement avec des caractéristiques invariantes comme la position initiale des neufs palets, le traçage des lignes et leur couleur, les dimensions des palets et du plateau, etc. Tous les robots possèdent une structure identique, avec les mêmes moteurs, les mêmes capteurs, le même diamètre de roue, les mêmes dimensions, etc. Il s'agit des premiers éléments que nous avons intégrés à notre modèle. Leur représentation graphique est présentée dans le tableau ci-dessous.

Vue du plateau avec les palets (physique en haut, modélisé graphiquement en bas)

Vue du robot LEGO (physique en haut, modélisé graphiquement en bas)

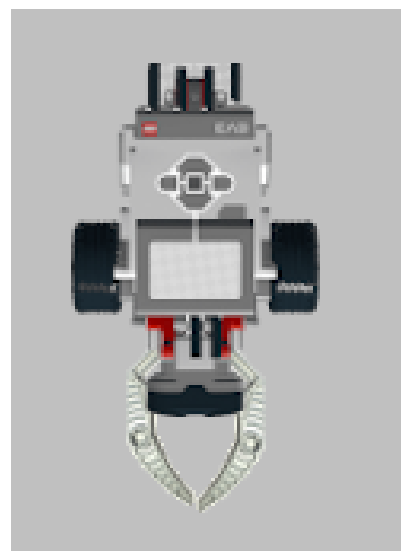
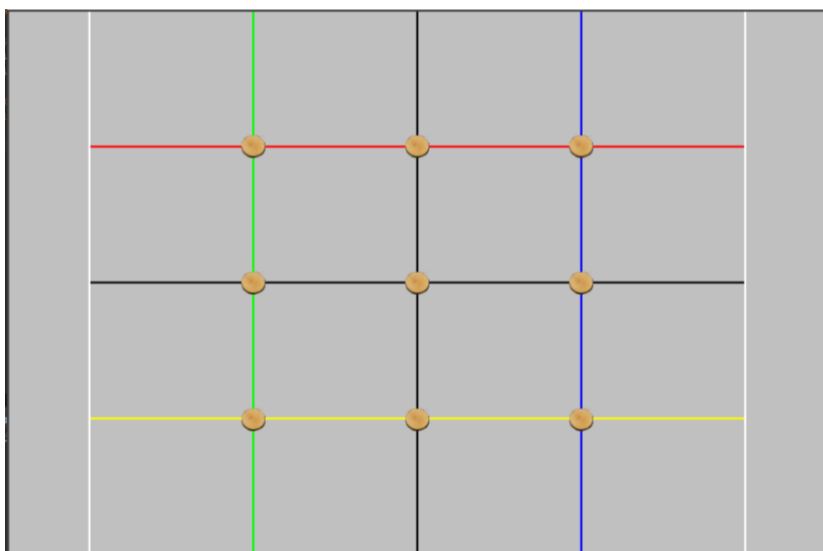
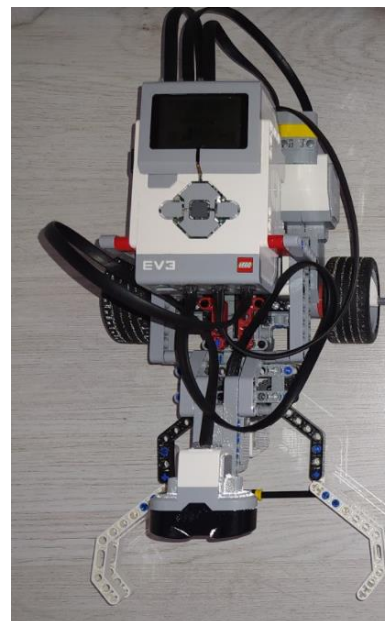


Figure 1 : Comparaison des objets graphiques et physiques

Motivation & enjeux

Comme dit en présentation, les stratégies sont non négligeables si l'on souhaite être efficace dans la récupération de palet. Ainsi, nos premières prétentions étaient de réaliser ce projet pour connaître à terme les stratégies les plus efficaces pour le robot physique. Pour information, l'application s'est avérée très utile pour illustrer la stratégie du robot à nos groupes respectifs pour le projet d'IA décrit précédemment (montrer les différentes étapes de recherches, etc...). Un autre facteur de motivation à faire ce projet est qu'il porte sur un sujet qui nous permet de montrer nos

compétences dans des domaines qui nous intéressent. Il s'agit donc aussi de faire valoir notre investissement dans ces domaines pour par exemple valoriser une candidature à un Master.

Objectifs et contraintes

Objectifs et contraintes techniques

Le but est de développer une application graphique. Son développement se fait en Java, car il s'agit d'un langage de programmation orienté objet donc pratique pour faire un code générique et maintenable. De plus, Java offre de nombreuses solutions relativement simples pour les méthodes graphiques, sujet sur lequel nous n'avions jamais été formés dans le cadre de la licence.

Pour pouvoir modéliser le comportement du robot, c'est-à-dire le comportement de tous ses composants, capteurs et moteurs, il faut avoir accès au robot physique. Cet accès est privilégié par le cours d'Intelligence Artificielle et il nous a été accordé même après la fin de ce cours.

Afin de faire des tests sur le robot, il aurait été préférable de les effectuer sur le vrai plateau (situé au Fablab derrière la maison de l'INP) plutôt que chez nous (pour cause de manque de place et de matériel parfois).

Le modèle repose sur un environnement, des objets ainsi que des règles spécifiques et les détails sont précisés sur le lien du site de M. Pellier situé en partie **Présentation du sujet**. Tous ces détails, invariants, sont stockés dans des variables dans le code. Changer un de ces détails implique de modifier le code en conséquence. Par exemple, si on vient à modifier les dimensions du plateau, il faut stocker une autre image du plateau. Ou encore si l'on remplace les moteurs des roues par des moteurs plus puissants, il faut modifier la fonction qui modélise l'accélération...

Délais

Le projet commence en date du 23 septembre 2020. Il n'y a pas de date précise pour un rendu mais dès septembre on désigne mi-janvier comme période de passation de la soutenance. Malgré qu'il y ait cette date limite, le projet doit être suivi régulièrement, ce qui le découpe en plusieurs périodes, toutes engagées et conclues par des réunions avec le tuteur. Nous allons explorer ces différentes périodes dans la suite du document.

Gestion de projet

Planification

Échéancier

La première étape du projet est de planifier son déroulement général. Définir un échéancier sur tout le semestre est la première tâche à laquelle nous nous sommes consacrés. Il permet d'ordonner les tâches et d'estimer la durée de chacune d'elles et d'en afficher les dates limites. Il est autant utile au groupe qu'à notre tuteur pour le suivi du projet. Il s'agit d'un échéancier itératif qui définit les différentes phases de travail lors d'une itération. L'avancement général du projet se définit donc sur le nombre d'itérations réalisées.

Plan de développement

Avant d'entrer dans un cycle itératif, on présente la structure de base du code dans le plan de développement. Elle décrit les attributs et les méthodes de la hiérarchie de classe de base. Ce plan de développement n'a pour but que de présenter l'organisation des différents paquetages, classes et méthodes que nous prévoyons de faire pour organiser notre code ainsi qu'une documentation succincte. Il n'a pas pour but d'être exhaustif sur l'ensemble des éléments mais simplement de documenter les principales fonctions. Ainsi, ce document n'a pas été mis à jour au fur et à mesure des modifications. Le plan de développement permet de mieux comprendre l'utilité de chacune des classes, méthodes ou autre de manière indépendante. Contrairement au diagramme UML, il ne comporte aucune documentation ; d'où leur complémentarité. Le diagramme UML est fonctionnel, d'où la nécessité de le mettre à jour à chaque itération dans le cadre de notre gestion de projet *agile*, tandis que la documentation formelle du plan de développement ne s'y prête pas vraiment. Malgré tout, la structure générale du code reste quasi inchangée au fil des itérations, car elle est conçue dès le début pour être ainsi. C'est pourquoi il est opportun de la documenter davantage dans un document.

Gestion

Diagramme UML

Le **diagramme UML** sert à représenter graphiquement les relations entre les différentes classes et interfaces du code ainsi que leurs attributs. Dans sa première version, ce diagramme représente les objets décrits dans le plan de développement. Cependant, une fois le code de base terminé et les phases itératives débutées, de nouvelles classes et de nouvelles méthodes sont ajoutées. Les relations entre les classes, ainsi que l'**encapsulation** de ses nouveaux attributs et méthodes, sont mises à jour dans le diagramme UML. Par exemple, lorsque l'interaction physique entre les objets est implémentée on rajoute la classe abstraite *Shape* qui définit les formes d'un objet abstrait. Cette classe permet de faciliter les interactions entre les différents objets du plateau.

Cahier des charges

Le cahier des charges nous permet de préciser le contexte, les objectifs et les contraintes. Il nous permet aussi d'énumérer les ressources que nous utilisons et de donner un cadre précis au projet. En bref, le cahier des charges a été un outil indispensable pour la description de nos tâches. Nous avons partagé sa première version avec M. Lemaire, pour qu'il valide la direction que nous souhaitons prendre, puis pour chacune des fonctionnalités ajoutées. Chaque mise à jour importante s'est traduite par une modification du cahier des charges et le renvoi de celui-ci. On y résume l'itération faite ainsi que les changements anticipés pour la prochaine mise à jour.

Répartition des tâches

Les aspects de gestion comme le cahier des charges et l'échéancier sont réalisés en groupe.

Le plan de développement est réalisé par Pierre

En ce qui concerne le code et le diagramme UML, ils sont réalisés par Bastien.

Les tests expérimentaux et la collection des données sont effectués par Pierre.

Les tests statistiques sont faits par Bastien.

L'évaluation et la rédaction du rapport sont effectuées en groupe.

Outils

Plusieurs ressources sont citées dans le cahier des charges. Ce sont celles que nous utilisons sur l'ensemble du projet. Le groupe n'étant composé que de deux membres, la gestion est grandement facilitée et nous privilégions une communication rapide sur *Discord*, type messagerie instantanée, pour l'exécution des tâches. Nous utilisons aussi la plateforme *Github* pour qu'il y ait régulièrement une version du code à jour visible accessible via ce lien : <https://github.com/fleuryP/ProjetJava.git>. La programmation s'est faite sur l'IDE *Eclipse* puisqu'il s'agit du logiciel de développement Java que nous utilisons dans le cadre de la licence depuis la L1. Le traitement statistique se fait sur *RStudio* pour le traitement de données, utilisé depuis la L1 aussi, et nous utilisons *Excel* pour faire les graphes. On utilise également *Diagrams.net* pour faire le diagramme UML. Il présentait l'avantage d'avoir assez peu de fonctionnalités, juste celles dont on avait besoin, et d'être éditables en ligne.

Développement

Stratégies

Réaliser les bases du code est un gros travail lui-même divisé en plusieurs étapes. Ce travail concerne essentiellement la conception générale du code, c'est-à-dire sa structure et son fonctionnement, et est documenté dans le plan de développement.

Une fois le cahier des charges défini, et les bases réalisées, le groupe fonctionne sur le principe de la conception d'une fonction après l'autre. La conception d'une fonction est équivalente à une itération divisée en phases types. Dans chacune d'elle on retrouve :

- une phase de recherche, pour implémenter une nouvelle fonctionnalité au modèle. On regarde la pertinence de la modification, puis on anticipe d'une part la complexité de l'intégration de cette fonctionnalité au modèle, et d'autre part de quelle manière on peut l'intégrer au modèle, au niveau du code essentiellement ;
- une phase de mise au point avec le tuteur, pour résumer les changements qui ont été apportés au modèle lors de l'itération précédente. Mais, également pour annoncer notre prochaine itération, et pour discuter de certains détails concernant par exemple de la méthodologie ;
- une phase de tests avec le robot LEGO physique, selon la nature de la fonctionnalité à implémenter. Par exemple pour étudier la puissance des moteurs ou la variabilité des capteurs ;
- une phase de traitement des données statistiques ou pour récupérer des invariants à partir des tests réalisés ;

- une phase d'implémentation du code. Cela peut être un simple ajout/modification de méthode ou d'attribut ou l'ajout d'une classe, voire l'ajout d'un paquetage ;
- une phase de mise à jour de deux livrables : le cahier des charges où les modifications sont faites en couleur bleue, et le diagramme UML qui est mis à jour en conséquence des modifications dans le code. Ces deux documents sont envoyés en amont pour préparer le prochain rendez-vous. Le code est mis à jour sur github.

A noter que certaines itérations n'impliquent pas de rendez-vous lorsqu'il s'agit simplement d'une amélioration d'une itération passée ou pour de l'optimisation du code, etc...

Concernant la structure générale du code, il existe trois classes principales dont *Plateau*, *Robot*, et *Palet*. Les objets qui interagissent entre eux sur le plateau (*Robot* et *Palet*) héritent d'une classe abstraite *Objects*. Un *Robot* possède une liste de *Sensor* et de *Motor*. Le premier objet est la classe abstraite mère des capteurs de distance, couleur, et tactile, et le deuxième concerne les moteurs.

Grâce au **polymorphisme** on définit des méthodes dans *Objects* pour "update" (mettre à jour) et "repaint" (re-peindre) les objets du plateau. Le code est donc générique dans la mesure où on peut très facilement ajouter de nouveaux objets sur le plateau ou modifier certaines caractéristiques d'objets existants. Cette généricité sera appréciable si certains éléments de la compétition venaient à être modifiés.

Phase itérative 1 : Gestion des collisions

La gestion des collisions est la première fonctionnalité implémentée au modèle. Il s'agit de manière plus générale de l'interaction entre les objets du plateau, incluant la prise d'un palet dans les pinces du robot. Une manière de faire est d'implémenter une classe abstraite *Shape* indiquant la forme d'un objet abstrait du plateau. Un tel objet donnerait des définitions de méthode pour savoir par exemple si la forme d'un objet croise la forme d'un autre, ou bien si la forme de l'objet contient un point, etc... Une autre manière de voir la chose est de rajouter ces méthodes directement dans la définition d'un objet abstrait ; mais cela implique de définir deux fois le même code (si un robot croise un palet ou si un palet croise un robot ; puis également palet/palet et robot/robot) mais implique également une dépendance entre des objets abstraits alors qu'une dépendance entre les formes suffit.

Notre méthode est donc de faire des tests de collisions sur les formes des objets prises deux à deux. Par exemple, on prend deux objets sur le plateau A et B ; il ne faut ni tester (A;A) ni tester (B;A) si (A;B) l'a déjà été. Pour cela, on ajoute un "handler" (gestionnaire) de collision dans le programme qui teste constamment tous les couples de formes d'objet possibles et qui effectue les décalages en fonction.

A terme, il est difficile de modéliser correctement de quelle manière un robot pousse un palet, ou un robot. Le fait qu'il soit poussé contre sa pince ou contre sa roue n'est pas discriminé par le modèle ; alors que le résultat est différent dans la réalité. De plus, le modèle intègre la vitesse de l'objet qui en percute un autre pour déterminer de quelle distance il est décalé. Il ne s'agit d'aucun calcul concernant les forces, simplement une distance de décalage proportionnelle à la vitesse de l'objet percutant.

Phase itérative 2 : Vitesse et accélération

Dans cette étape, nous pensons à modéliser l'accélération du robot, c'est-à-dire la dérivée de la vitesse, ou encore la vitesse à laquelle le robot atteint la vitesse demandée.

Pour la mesurer, on place le robot sur un sol plat et on fixe une caméra au-dessus du tout à environ un mètre du sol. A partir de là, on a testé plusieurs méthodes pour mesurer cette vitesse.

Le modèle de l'accélération se base sur un seul paramètre d'accélération. Il s'agit du paramètre par défaut du moteur, soit une accélération de 200 mm/s^2 . Cela implique que si la vitesse à atteindre est inférieure à 200 mm/s^2 , le moteur accélère plus qu'il ne faut, puis réduit la vitesse dans un second temps.

Pour précision, si la vitesse des moteurs est trop élevée, le robot a tendance à dévier et ne plus suivre une trajectoire droite. Cela peut être dû à une différence de puissance entre les deux moteurs à cause de leur détérioration. Nous avons choisi de ne pas modéliser cet aspect.

La première idée pour mesurer la variabilité de la vitesse du robot est de placer le robot perpendiculaire à un mur afin d'utiliser le capteur de distance. Il ne doit pas y avoir d'obstacles sur les côtés lorsque le robot avance vers le mur car l'angle de détection du capteur est large (20°) et il peut détecter ces objets, ce qui fausse les données. Ayant reçu un cours sur les flux et fichiers, nous savons qu'il suffit d'écrire dans un fichier textes les distances mesurées par le capteur toutes les 100 ms pour récolter les données. Après coup, il s'est avéré que cette méthode ne prend pas en compte la variabilité du capteur de distance elle-même, et est donc obsolète. Cette variabilité sera donc l'objet d'une itération future et cela nous convainc de changer de méthode.

La deuxième idée est basée sur le pointage grâce au repérage. On place sur une distance d'un mètre des feuilles graduées tous les 5 cm. Le robot est installé sur cette règle graduée. Voici une [vidéo](#) qui permet de mieux comprendre le schéma. Ici, nous n'utilisons plus le capteur de distance et la vitesse théorique du robot est de 150 mm/s . Peu importe la distance parcourue par le robot pour mesurer la vitesse, on la mesure relativement en pointant toujours le même repère, le bout d'une pince par exemple, par rapport à un point d'origine défini. On peut maintenant s'intéresser à la façon d'obtenir la vitesse de ce robot filmé.

Grâce à un logiciel de montage, on découpe la vidéo toutes les 200ms et on mesure la distance entre l'origine du robot et le repère défini. On utilise la formule de dérivation $v = OM/dt$ où v est la vitesse, OM la distance sur un intervalle de temps et dt l'intervalle de temps. Grâce à cela on récupère l'évolution de la vitesse au cours du temps et on infère plusieurs fonctions mathématiques qui semblent suivre la même évolution. On a utilisé pour cela le logiciel graphique *Geogebra*.

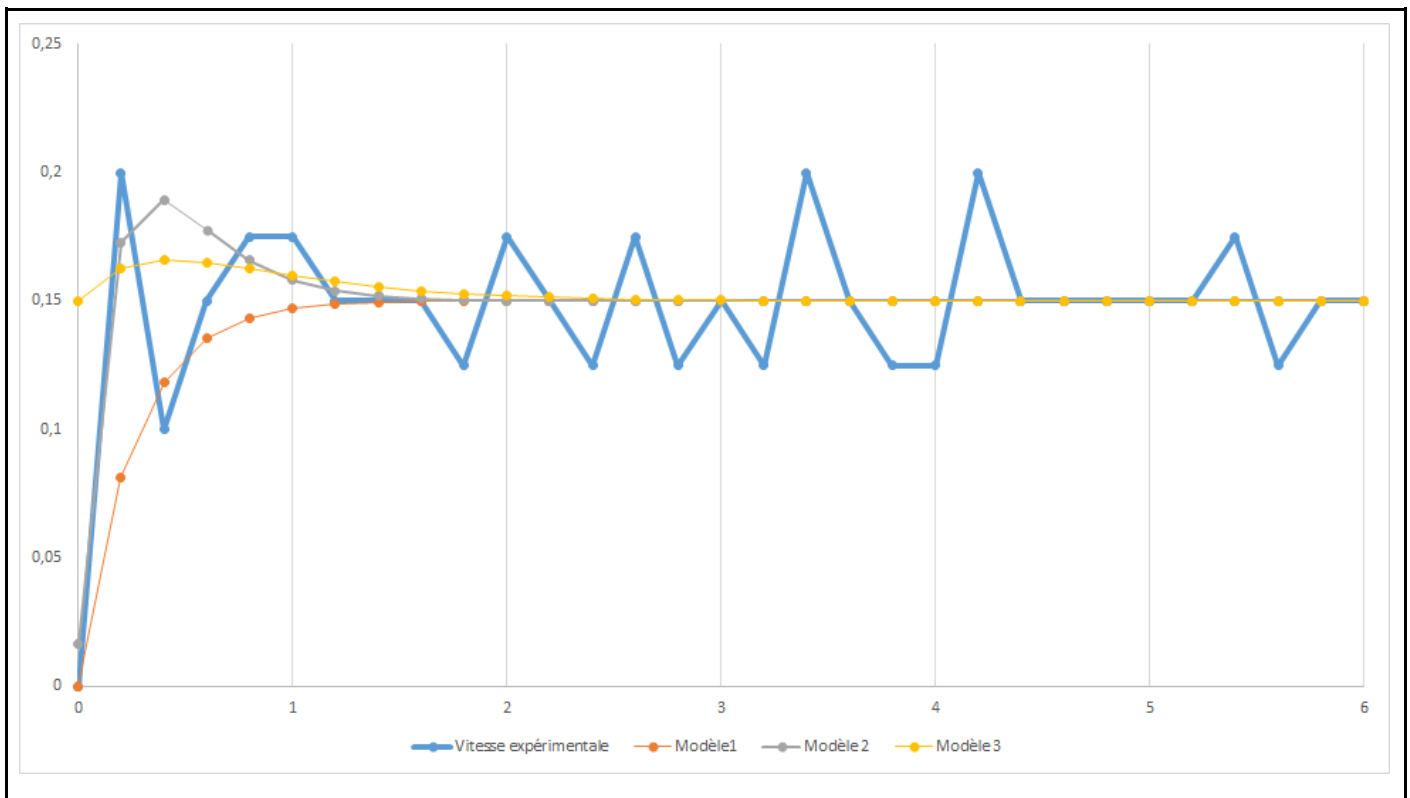


Figure 2 : Evolution de la vitesse en fonction du temps

L'unité en abscisses est en secondes et celle en ordonnées en mètres par secondes. La courbe bleue représente l'évolution de la vitesse observée, les trois autres courbes représentent chacune trois fonctions différentes. On cherche la somme des **écarts quadratiques** entre chaque valeur observée et celle du modèle. Les fonctions des modèles 1 et 2 sont de type exponentiel et celle du modèle 3 est de type sinusoïdal. Parmi les modèles, on choisit celui dont la somme des écarts quadratiques de toutes les valeurs est minimale, c'est-à-dire le modèle 2 sur le graphique précédent. Enfin, pour mettre cette fonction en application, nous avons ajouté au code une fonction *static* qui prend en paramètre un temps et une vitesse à atteindre et retourne la vitesse actuelle du robot.

Phase itérative 3 : Optimisation et formes géométriques

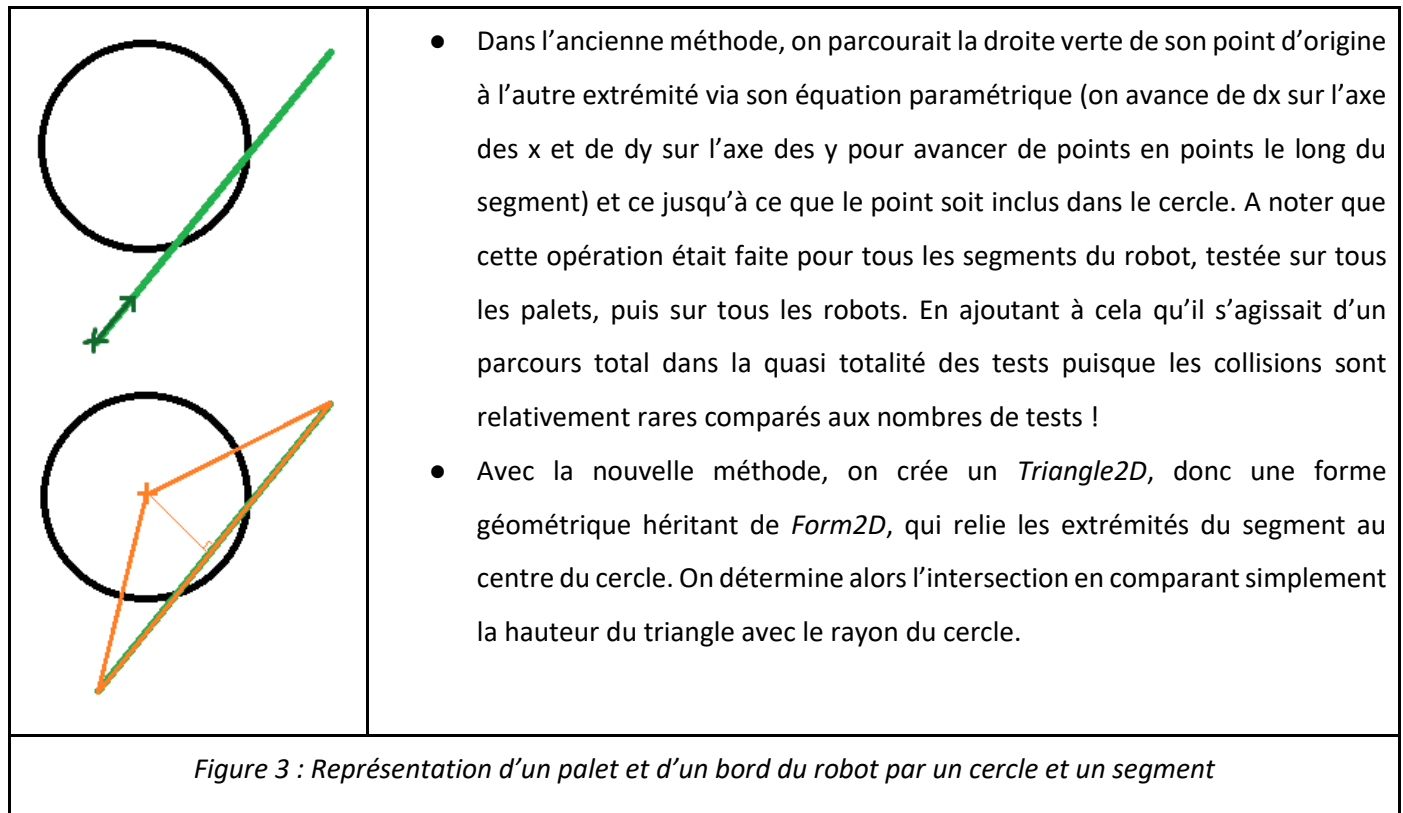
Cette étape n'a pas été anticipée. Il s'agit de résoudre un problème de performances qui devenait problématique lorsque trop d'objets étaient ajoutés sur le plateau. Le fait est qu'on ne testait nos méthodes qu'avec peu de palets et un seul robot. Voici différents vecteurs d'amélioration qui ont pallié cet imprévu.

Premièrement, des connaissances acquises du cours d'info en programmation objet concernant les *Collection* et *Map*, a permis d'utiliser le type de liste adapté là où nous utilisions auparavant uniquement des *Array*. On gagne ainsi en performance en parcours sur les *ArrayList*, plus un redimensionnement dynamique plus performant.

Au niveau de la gestion des collisions, tout a été repensé pour limiter le nombre de calculs. On a poussé plus loin le concept d'abstraction de forme d'objets en formes géométriques et donc en déléguant les tests d'intersection de la classe *Shape* à des tests d'intersection sur des *Form2D*, notre classe abstraite des formes géométriques basée sur la librairie *java.awt.Shape*. Ainsi, on ne teste plus des robots et des palets mais des polygones et des cercles. Le premier

avantage est de pouvoir récupérer directement des informations concernant une figure géométrique et non plus l'objet. Récupérer la largeur revient à récupérer le diamètre du cercle qui lui correspond, et non plus la moitié des pixels qui forment la largeur de l'image. Le deuxième est de clairement optimiser nos méthodes d'intersection ; voici un exemple plus concret ci-après.

On schématise le palet par le cercle et un des bords du robot par une ligne verte. Lorsque le gestionnaire de collision testera ces deux objets, il renverra *true* (vrai).



En réalité cette dernière condition n'est pas suffisante et notre modèle implique également la médiane, l'exemple avait pour unique but de montrer précisément de quelle manière le code avait été optimisé en déléguant les tests de collisions à des tests sur des figures géométriques.

Phase itérative 4 : Fonctionnement des capteurs

Cette itération a pour but de modéliser les capteurs de couleur, tactile et de distance du robot. Le premier permet de détecter une couleur, le deuxième permet de savoir si un objet appuie sur le balancier du robot, situé derrière les pinces. Le troisième capteur renvoie la distance de l'objet le plus proche de lui situé dans un angle de 20° face au robot. Les formes géométriques décrites dans la phase itérative 3 se sont montrées essentielles pour modéliser ces capteurs. Avant tout, précisons que le robot est construit de telle sorte que les trois capteurs sont superposés, et que par conséquent, les trois capteurs se situent aux mêmes coordonnées dans le plan (x,y) .

Le capteur tactile fonctionne sur un principe simple. Le balancier est simplement représenté par un segment (*Ligne2D*). Sachant que tous les objets possèdent une forme, si celle-ci croise la droite du capteur tactile, il renvoie *true*. Les détails techniques de cette action sont dans la documentation pour la méthode *intersects* de la classe *Geometry*.

Le capteur de couleur quant à lui récupère directement la couleur du pixel de l'image *plateau.png* situé juste en dessous de lui.

Le capteur de distance physique fonctionne sur le principe de propagation d'onde en continu. Lorsque ces ondes entrent en contact avec un objet elles reviennent jusqu'au capteur. Il calcule la distance entre les deux connaissant la célérité de l'onde propagée et le temps mis pour partir et revenir. Pour rappel, le capteur physique a une portée limitée allant de 5 cm à 255 cm et notre échelle graphique est de 1 cm pour 4 pixels. La portée du capteur graphique est donc de 5 à 1020 pixels. Pour modéliser le cône, on utilise un *Triangle2D* qui modélise la portée des ondes. En ce qui concerne la plus petite distance, on regarde d'abord l'intersection des murs sur les extrémités du triangles, et on redéfinit un "sous-triangle" par conséquent plus petit. On regarde enfin si la forme d'un objet du plateau croise ce triangle. Une fois l'objet le plus proche détecté, s'il y en a un, le capteur retourne la distance entre le robot et cet objet.

Phase itérative 5 : Variabilité des capteurs

Dans l'itération précédente on décrit le fonctionnement des capteurs du robot. Ici on cherche à décrire la variabilité de chacun des capteurs, c'est-à-dire leur marge d'erreur. En ce sens, il apparaît que nous n'auront pas besoin de modéliser la variabilité des capteurs tactile et de couleur. En effet, cette dernière dépend uniquement de l'état, usage et détérioration de son capteur, ce qui n'est pas le cas du capteur de distance. On a choisi de ne pas modéliser ces aspects.

Pour mesurer la variabilité du capteur de distance, on s'inspire fortement de la première méthode utilisée pour mesurer la vitesse et l'accélération. Cependant, nous n'avons pas besoin d'utiliser de graduation ou de filmer la scène pour récolter des données. On place juste le robot toujours au même endroit et face à un mur, à environ un mètre, et de façon à ce qu'aucun objet n'entre dans la zone du triangle décrit dans l'itération précédente. On fait avancer le robot à une vitesse constante de 150 mm/s, et on inscrit dans un fichier texte la valeur que retourne le capteur de distance toutes les 100 ms. Cela revient à faire environ 60 itérations et donc à récupérer 60 données par échantillon. Sur les bases de la psychologie expérimentale, environ 30 échantillons ont été réalisés dans un premier temps. Ce nombre étant insuffisant pour mettre en évidence la loi suivie par la variabilité du capteur, 60 échantillons de plus ont été réalisés.

Grâce au grand nombre d'échantillons, on est en mesure de centrer et réduire les valeurs pour chaque rang avec respectivement la moyenne et l'écart-type du rang. On a commencé par faire un test de normalité des données avec d'abord un test graphique. L'histogramme de notre distribution, superposé par la densité de la loi normale $N(0,1)$:

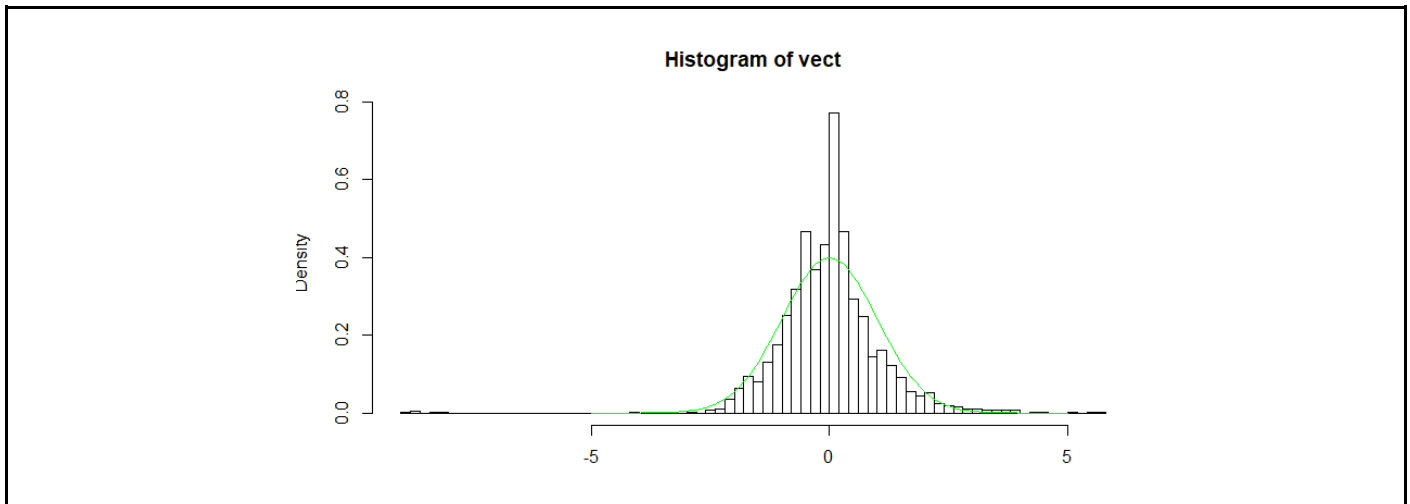


Figure 4 : Histogramme de la distribution et densité gaussienne centrée réduite

- un *qqplot* (quantile-quantile plot) qui est un graphique qui superpose les quantiles, on en a pris 200, de notre distribution avec les quantiles de la loi normale centrée-réduite

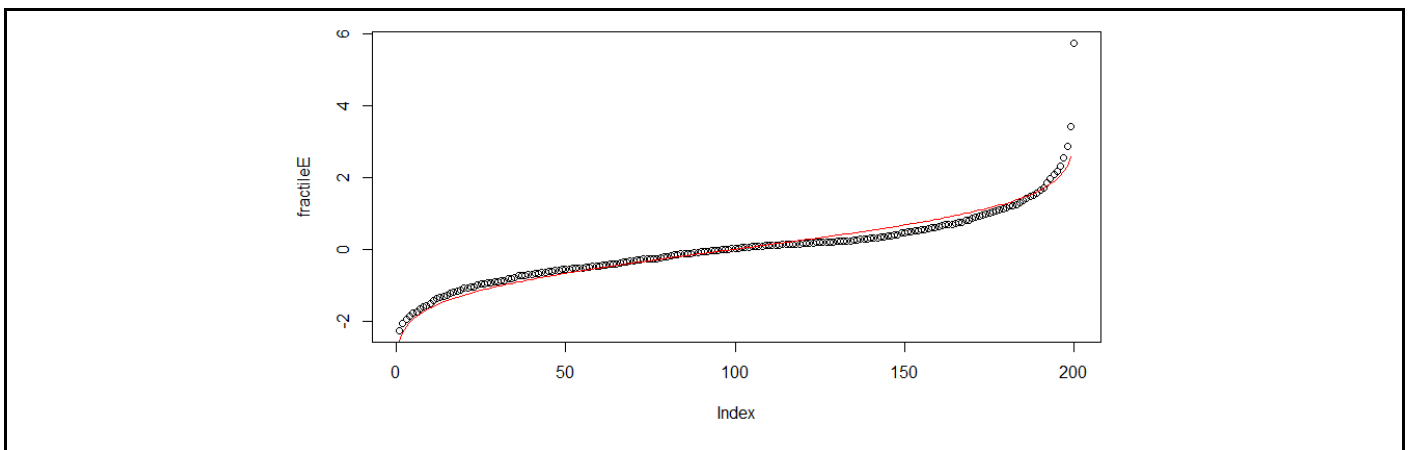


Figure 5 : Quantiles de notre distribution et de la loi normale

Ce test graphique nous convainc de monter un test statistique, le test de Shapiro, concernant la normalité des données dont l'issue corrobore notre hypothèse. On fait également un test sur les paramètres des deux lois : d'abord sur l'espérance, où on ne rejette pas H_0 avec une p-value de 0.48 puis même conclusion sur l'écart-type avec une p-value de 0.29.

Ainsi, on intègre la variabilité du capteur de distance comme ceci :

valeur affichée = [tirage aléatoire d'une $N(0,1)$] \times écart-type du rang + espérance du rang

Phase itérative 6 : Javadoc et visibilité

Étant donné qu'il y a une grande quantité de code, la Javadoc a été ajoutée au fur et à mesure de l'ajout d'objets ou de méthodes. Cependant, rappelons que la Javadoc est une documentation apparente dans le code source, et basée sur le format HTML. Par conséquent, une Javadoc efficace présente des liens hypertextes lorsqu'une méthode fait

référence à d'autres méthodes (ou classes, attributs, etc). A la fin du projet, la dernière itération est de rajouter ces liens via les tags Javadoc ("@see").

La visibilité du code quant à elle fait référence aux droits d'accès, ou de modifications, via la redéfinition, de méthodes par l'utilisateur. En effet, l'application est utilisable en l'état pour servir à simuler des stratégies. Cela implique des restrictions pour l'utilisateur. Un des objectifs est d'empêcher l'utilisateur d'avoir accès, via la visibilité ou bien l'accessibilité de certains attributs/méthodes/classes, à certaines données utilisées pour coder les bases du programme auxquelles l'utilisateur n'a pas accès dans la réalité via les capteurs de son robot. Par exemple, la position (x,y) sur le plateau. De plus, on peut parler de la redéfinition de méthodes dans l'optique de "dénaturer" le comportement de celle-ci. Il s'agit par exemple d'empêcher l'utilisateur de redéfinir la méthode *avancer()*, via l'héritage, pour faire en sorte de téléporter le robot sur le plateau.

Les seules données accessibles depuis l'environnement pour un robot sont celles perçues par l'ensemble de ses capteurs.

Bilan

FLEURY Pierre

Techniquement parlant, le projet m'a apporté des compétences, mais moins que ce que j'aurai aimé. En effet, Bastien s'est vite imposé comme le meneur du projet et le suivre a rapidement été difficile. Néanmoins, il ne fait aucun doute que j'ai aimé réaliser ce projet car à nous deux nous avons su mener technique et gestion. La liberté accordée est aussi en grande partie responsable du fait que j'ai apprécié, j'en suis convaincu.

Pour terminer je dirais que, lorsque nous avons débuté un projet de gestion l'année dernière, les frontières étaient très floues entre ce qu'on attendait de nous et ce que nous devions faire. Quand ce dernier projet s'est terminé, c'était plus précis mais toujours flou. Avec le projet de ce semestre j'ai la nette impression d'avoir gagné en netteté et en rigueur pour la réalisation de gros travaux, que ce soit pour la planification ou l'exécution. C'est selon moi ce que la réalisation du projet m'a le plus apporté.

GATTACIECCA Bastien

Le fait de lier d'une part la gestion de projet qui m'avait plu l'an dernier, sur un sujet clairement en lien avec mes cours et mes intérêts m'a largement motivé à réaliser ce projet. Le temps investi dans le code et pour trouver de nouvelles méthodes d'optimisation en est un indicateur. Hélas l'ensemble des difficultés rencontrées ne pourront pas être présentées dans ce rapport ni dans la soutenance vu leur quantité et leur diversité. Mais, enrichi de ces erreurs je serai plus rigoureux lorsque j'y serai confronté de nouveaux.

Tout au long du projet, j'ai apprécié l'efficacité de notre groupe à tous les niveaux, et le fait de ne plus être face aux mêmes problèmes que ceux rencontrés en gestion de projet de deuxième année.

C'est avec la même motivation que j'aimerais poursuivre ce projet au semestre prochain ; sans doute en orientant le sujet vers l'intelligence artificielle, domaine dans lequel j'aimerais m'investir davantage.

Conclusion

Au terme du semestre, et du projet, où en est-on ? Nous sommes en mesure de proposer un programme fonctionnel pour l'étude de l'évolution d'un robot LEGO ramasseur de palets. L'objectif de faire s'affronter différentes stratégies est rempli à la condition de comprendre le code et d'implémenter ces stratégies par soi-même.

Ce n'est pas la première fois que nous pouvons choisir le sujet qui nous inspire pour l'étudier, mais, c'est la première fois que nous pouvons choisir librement l'évolution que va prendre celui-ci. Ainsi, nous avons montré de la motivation pour rendre le modèle le plus réaliste possible. En effet, nous avons essayé d'utiliser au mieux les connaissances dont nous disposons dans plein de domaines comme l'algèbre, la trigonométrie, les statistiques, la programmation, la gestion... Nous avons aussi utilisé des compétences transversales acquises, ou apprises durant le projet comme celles durant la phase d'auto-formation.

Ce que ce travail nous apporte ? Une meilleure idée de ce qu'est un projet informatique.

Critique

Nous n'avons pas prévu de grille critériée au début du projet pour nous évaluer dans la période de finalisation et évaluation décrite par l'échéancier.

La répartition des tâches n'a pas été planifiée. Ainsi, le temps de travail des membres du groupe ne respecte pas le temps de travail théorique pour ce projet, soit 90 h.

Ouvertures

Il est possible d'améliorer ce qui existe déjà ou d'ajouter des fonctions.

Si le temps nous le permettait, on aurait essayé d'améliorer l'interface de l'application et de prendre en compte les aspects ergonomiques pour optimiser son exploitation par des utilisateurs. En ce sens, les utilisateurs n'auraient pas à comprendre le code mais simplement à rentrer des paramètres et observer les résultats.

Glossaire

Distribution (statistique) : se dit d'une fonction mathématique qui associe la fréquence d'apparition d'un caractère à un intervalle de valeurs.

La normalité d'une distribution : le fait d'approximer une distribution par une loi normale lorsque la taille de l'échantillon est "assez grande" grâce au théorème central limite (TCL).

Multithreading : en programmation, il s'agit d'un système qui permet d'exécuter plusieurs tâches en même temps

Système embarqué : il s'agit d'un système électronique programmé, qui utilise des ressources pour effectuer de manière autonome une tâche.

Diagramme UML : diagramme Unified Modeling Language qui permet de visualiser les relations entre les objets.

Encapsulation : en programmation orientée objet, l'encapsulation permet de définir les attributs et les méthodes d'un objet au sein d'un même environnement. Cela pour se protéger des effets de bord (se protéger des actions qui n'ont pas lieu dans cet environnement).

Polymorphisme : en programmation objet, permet d'appliquer une méthode sur des objets différents. Le comportement de cette méthode différera selon le type réel de l'objet sur lequel elle est appliquée.

Ecart quadratique moyen : l'écart au carré entre la valeur théorique et observée divisée par la valeur théorique

Annexes

Toutes les annexes sont disponibles sur la page Github <https://github.com/fleuryP/ProjetJava.git>. Plus précisément, le code se trouve dans le dossier *Simulation* de cette page, qui contient les différentes images nécessaires au fonctionnement de l'application.

- Cahier des charges
- Code R statistique
- Code théorique
- Diagramme UML
- Échéancier
- Plan de développement