Marques Romain 28600712

Mohamed Aliou Diallo 28604190

Besevic Ivan 3415702

Gaydon Julie 28606830

Grondin Laurent

**Compte-rendu du projet 2IN013**



**Sommaire**

I Introduction

II Description de l’architecture

III Capacités de la simulation

IV Capacités de l’IA

V Les initiatives entreprises

VI Conclusion

**I Introduction**

Dans le cadre du projet de 2IN013, nous étions amenés à développer plusieurs algorithmes afin de pouvoir non seulement pouvoir déplacer un robot type gopigo 3 mais aussi implémenter différentes stratégies.

Tout d’abord, ce projet a pour but de nous apprendre à faire un projet puisque pour la majorité d’entre nous c’était notre premier projet important en quasi autonomie.

De plus, il nous a permis de développer nos connaissances en python. Ce point est primordial puisqu’aujourd’hui python représente le langage de programmation open source le plus employé par les informaticiens. Il est donc fondamental de le maitriser.

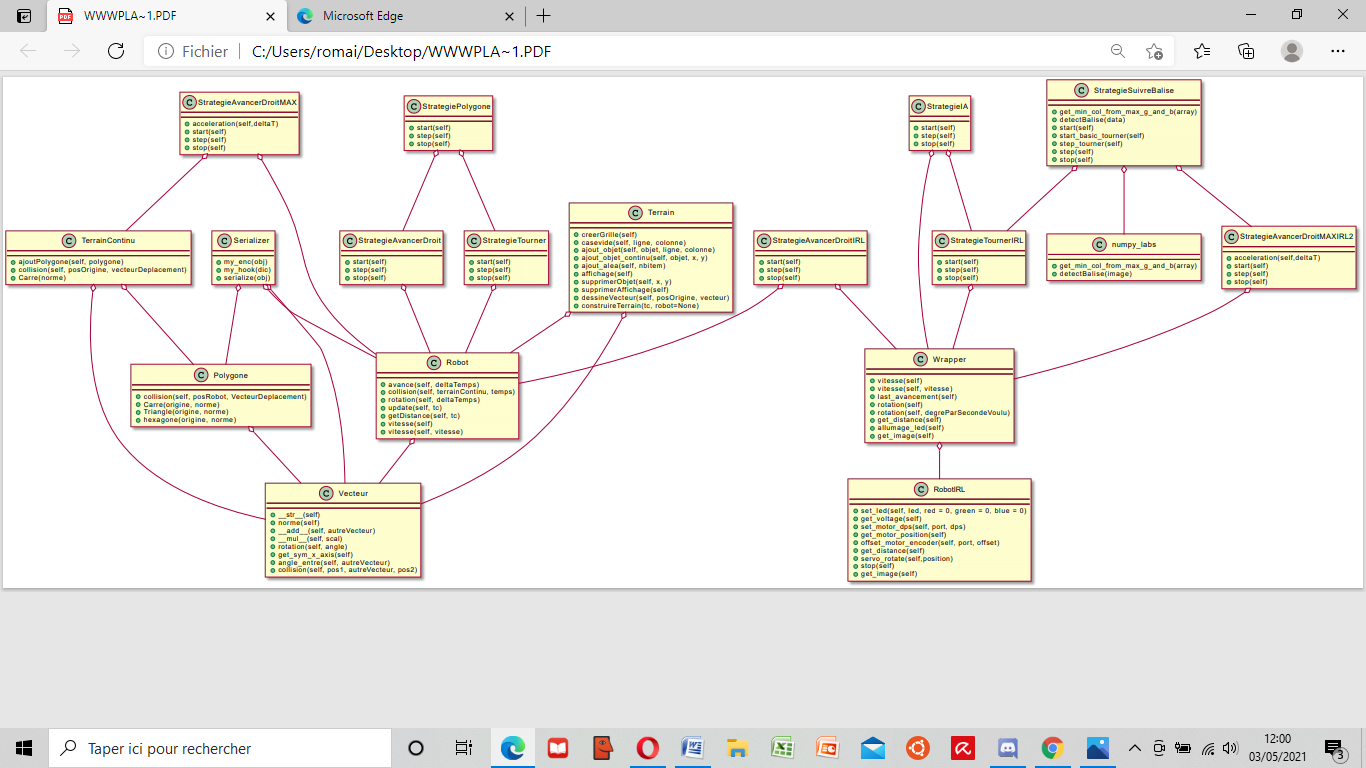
D’autre part, tout au long de ce semestre nous avons pu nous familiariser avec les méthodes Agile et Scrum qui sont deux méthodes de projet. Agile représente un ensemble de méthodes et pratiques qui repose sur la collaboration, l’autonomie et des équipes pluridisciplinaires. Il permet de livrer rapidement et régulièrement des fonctionnalités à grandes valeurs ajoutées au client et ainsi permettre d’opérer des changements sur un projet sans difficultés puisque les développeurs et les clients travaillent ensemble. Ainsi, le client reçoit fréquemment un logiciel opérationnel avec de nouvelles possibilités. Enfin, en utilisant cette méthode nous observons que le travail inutile est minimisé, les meilleures architectures se créent grâce à ce type de développement. Enfin, les équipes se réunissent à intervalles réguliers pour devenir plus efficace. Quant à Scrum, c’est un framework (une structure) utilisée pour implémenter la méthode Agile.

Le projet nous a aussi permis de connaitre mais aussi de maitriser trello et github. Le premier nous a servis à planifier et à répartir les tâches que nous souhaitions faire. Le deuxième nous a permis de tous travailler sous la même version du projet et de nous familiariser avec github qui est logiciel incontournable aujourd’hui que ce soit dans le monde du travail ou bien pour les particuliers.

Pour ce projet, nous avions plusieurs objectifs en rapport avec le robot, il devait pouvoir tracer un carré, s’approcher le plus vite et le plus près d’un mur sans le toucher mais aussi pouvoir suivre une balise. Pour se faire, nous l’avons divisé en deux parties, développer, d’un côté une simulation fidèle à la réalité, et de l’autre une implémentation afin que le robot puisse imiter le comportement de la simulation. Nous avons décidé de commencer par utiliser une simulation au lieu de seulement utiliser le robot puisque les essais réels coûtent chers que ce soit en temps ou en ressources. En effet, avec une simulation il n’y a aucun risque de casser le matériel. De plus, elle nous a aussi permis de mieux comprendre le comportement du robot. Cependant, le défaut de la simulation est qu’elle n’utilise pas le code du robot, elle est totalement indépendante et donc représente en soit une perte de temps de s’éterniser dessus.

**II Description de l’architecture**

Comme dit précédemment dans l’introduction, notre projet s’est divisé en deux parties, d’un côté la simulation et de l’autre l’utilisation du robot. Avant de commencer à rentrer dans les détails de l’architecture, voici un diagramme UML de notre projet (sans les tests ni les main de simulation ni les attributs de classe pour le simplifier) :



*1. La simulation*

Pour la simulation, nous avons commencé par développer un terrain simplifié sous forme de grille (tableau à deux dimensions) dans lequel nous pouvons ajouter dans chaque case soit un objet quelconque soit le robot. Ainsi, nous avons aussi développé un robot dont nous avons intégré certaines spécificités tout au long du semestre. Désormais, il possède une position représenté par les valeurs x et y, un vecteur déplacement, un angle pour savoir vers quelle direction il se déplace, et un booléen pour savoir s’il a rencontré un obstacle dans son parcours. Il contient des méthodes pour avancer et tourner, connaitre la distance qui le sépare d’un obstacle et savoir s’il en rencontrera un. La classe vecteur contient une position x et y ainsi que les méthodes standards des vecteurs.

Afin de pouvoir être au plus proche de la réalité, nous avons implémenté un nouveau terrain (cf TerrainContinu.py) qui est modélisé par un polygone, il contient une liste de polygones qui peuvent être soit le robot soit des obstacles. Enfin, il contient une échelle qui est initialisé à 0.5cm de base. Afin de simplifier la création du code, nous avons crée une méthode carrée qui initialise un terrain carré.

Les polygones possèdent une liste de sommets et une liste de vecteurs.

La classe Serializer permet de transformer une variable en json. C’est très utile pour la partie simulation avec Unity.

Ensuite, nous avons développé des stratégies afin de répondre à différentes problématiques.

La stratégie « StrategieAvancerDroit » permet au robot de parcourir une certaine distance passée en paramètres. Un défaut de cette stratégie est qu’elle ne gère pas les collisions puisqu’on ne lui passe pas le terrain continu en paramètres. C’est pourquoi il est préférable de s’assurer que le robot ne rencontre pas d’obstacles sur cette distance avant d’appeler la méthode. On pourrait appeler la méthode getDistance de robot pour s’en assurer.

La stratégie « StrategieTourner » permet de donner un angle en paramètres et que le robot tourne jusqu’à atteindre cet angle.

Les stratégies tourner et avancer sont les deux stratégies fondamentales du robot.

La stratégie « StrategieAvancerDroitMax » permet de foncer dans un mur le plus vite possible sans le toucher. Il répond à une des tâches données en début de semestre par notre enseignant. Il prend alors en paramètres le terrain continu, la vitesse maximale du robot et ce dernier.

La stratégie « StrategieCarre » a été renommé en « StrategiePolygone » pour être plus flexible. Elle prend en paramètres les stratégies avancer et tourner ainsi que le nombre de côtés du polygone souhaité (pour un carré n=4) et dessine la figure progressivement à chaque appel de step().

*2. Robot*

Pour le robot IRL, nous avons au départ crée une classe RobotIRL.py qui contient les méthodes fournies par le robot. Afin de pouvoir communiquer avec le robot à travers les simulations nous avons décidé d’implémenter une classe Wrapper qui se chargera de donner les instructions au robot. Ainsi, elle pourra modifier la vitesse du robot, connaitre la distance parcourue par ce dernier, le déplacer ou encore obtenir la distance avec le prochain obstacle ou obtenir les images lues par sa caméra.

Afin d’alléger le code et d’éviter de devoir écrire les mêmes lignes encore et encore, nous avons décidé d’implémenter certaines stratégies pour le robot IRL. Elles prennent donc toutes le Wrapper en paramètres.

Les stratégies « StrategieAvancerDroitIRL » , « StrategieTournerIRL » et « StrategieAvancerDroitMaxIRL2 » ont un comportement équivalent aux stratégies définies dans la simulation.

La stratégie « StrategieIA » a pour comportement d’avancer droit tant qu’elle ne rencontre pas d’obstacles puis tourner jusqu’à ce que soit il n’y est plus d’obstacles devant le robot à moins de 50cm ou si elle opéré une rotation de plus de 360° alors la simulation s’arrête. Il faut arrêter la simulation à la main si le robot n’est pas bloqué.

**III Capacités de la simulation**