Marques Romain 28600712
Mohamed Aliou Diallo 28604190
Besevic Ivan 3415702
Gaydon Julie 28606830
Grondin Laurent

Compte-rendu du projet 2IN013

Sommaire

I Introduction ……………………………………………………………………………………………..3

II Description de l'architecture………………………………………………………………………….4

III Capacités de la simulation………………………………………………………………………...6

IV Capacités de l'IA……………………………………………………………………………………..10

V Participation au code………………………………………………………………………………….12

VI Les initiatives entreprises………………………………………………………………………….14

VII Les tests……………………………………………………………………………………………………14

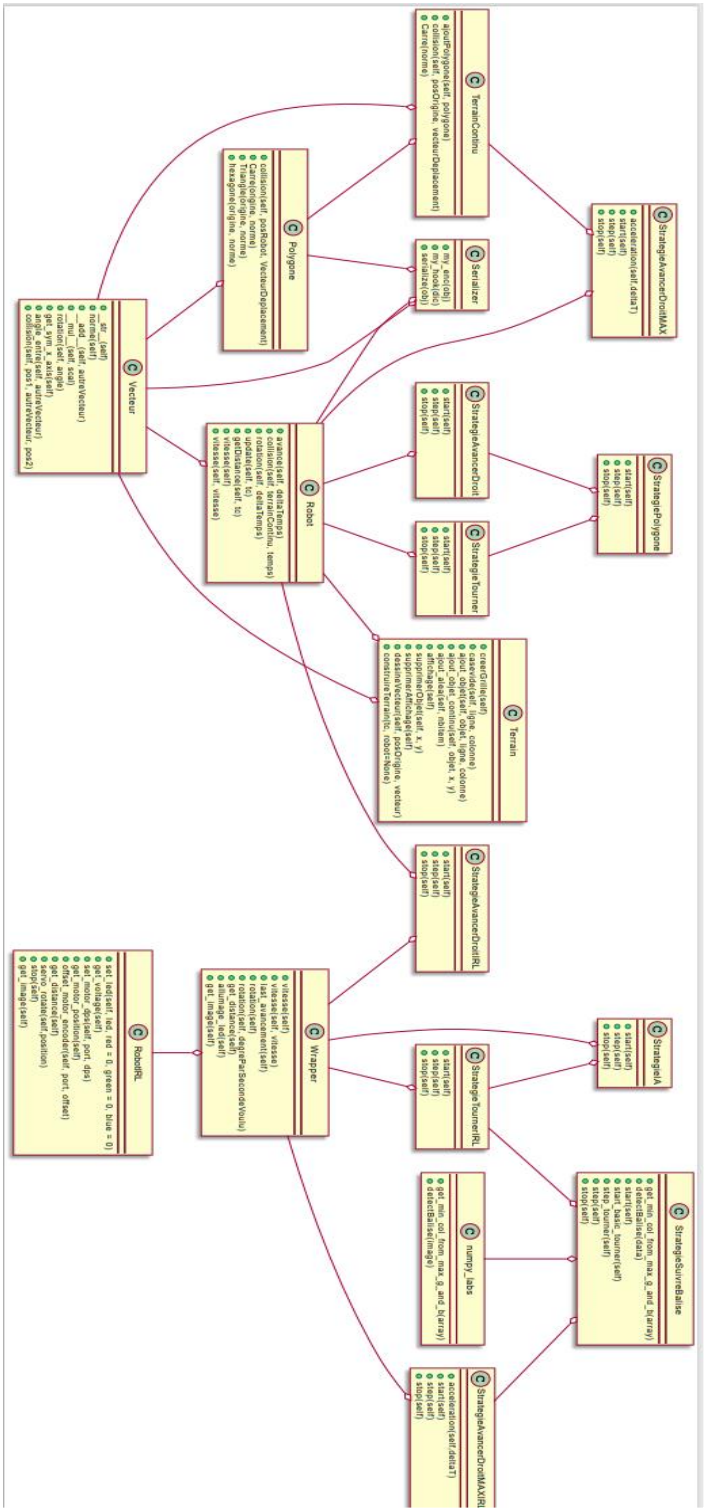
VIII Conclusion…………………………………………………………………………………………………15

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, cette unité d'enseignement a pour but de nous apprendre à faire un projet puisque pour la majorité d'entre nous c'était le premier en informatique qui soit aussi important et à faire 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 maîtriser.
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 développement 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 et de meilleures architectures sont créées. Enfin, les équipes se réunissent à intervalles réguliers pour être plus efficaces. Quant à Scrum, c'est un framework (une structure) utilisée pour implémenter la méthode Agile.
Cette matière nous a aussi permis de connaître mais aussi de maîtriser Trello et github. Le premier nous a servi 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 le 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 principal de la simulation est qu'elle n'interagit pas avec 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:



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 représentée 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, connaître 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 initialisée à 0.5cm de base. Afin de simplifier la création du code, nous avons créé 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(). De plus, en fonction de ses arguments elle peut être utilisée pour le robot physique ou pour le robot simulé. En effet, si nous lui donnons en argument les stratégies AvancerDroitIRL et TournerIRL il va alors tracer un polygone pour le robot physique tandis que si nous lui fournissons AvancerDroit et Tourner elle tracera un polygone pour le robot simulé.
La stratégie « StrategieIASimule » correspond à la StrategieIA pour le robot de la simulation.
Quant aux tests, ils sont unitaires et nous assurent que le code est fonctionnel, c'est-à-dire qu'il a le comportement souhaité et qu'il n'y ait aucun problème. Ils ont été réalisés par Ivan et Julie.

2. Robot

Pour le robot IRL, nous avons au départ créé 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, connaître 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 « StrategieAvancerDroitMaxIRL » 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

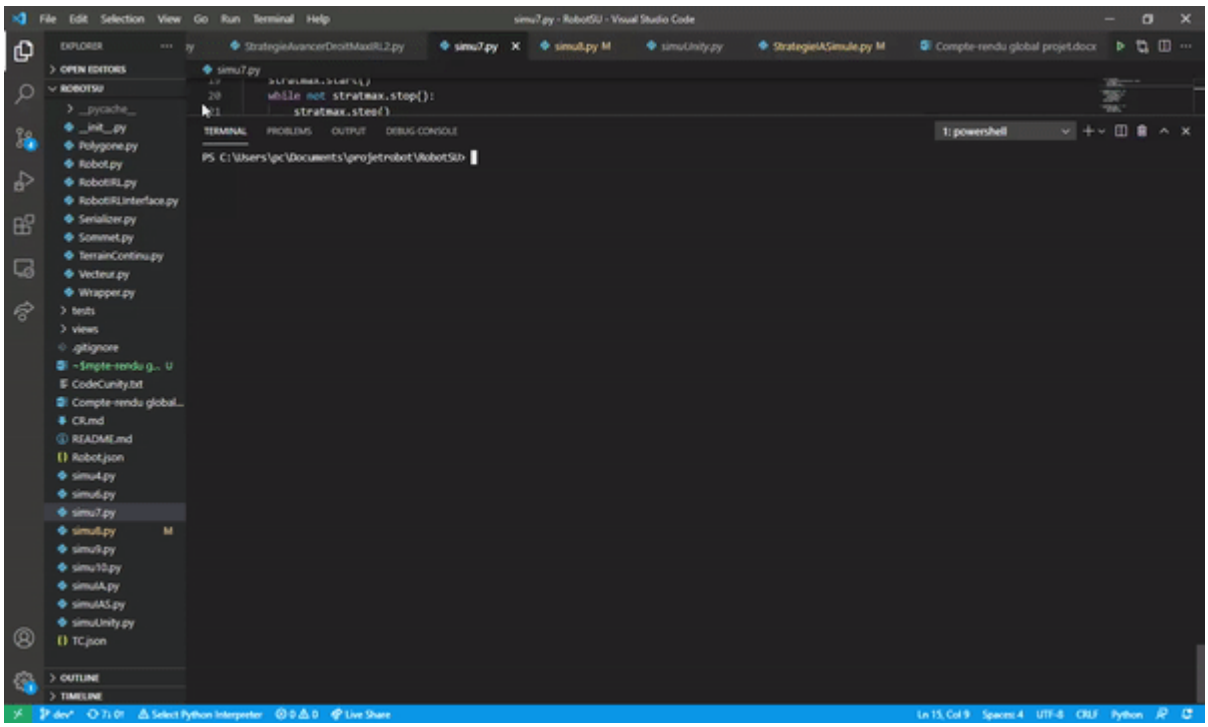
1. Simulation en Python

Après avoir codé des stratégies en python, on veut pouvoir les tester et observer le résultat : c'est le but des simulations.
Nos simulations sont réalisées à l'aide des diverses stratégies que nous avons écrites.
Les principales stratégies sont AvancerDroit et Tourner, qui permettent, comme leur nom l'indique, de faire avancer le robot et que ce dernier puisse tourner d'un certain angle. A l'aide de ces deux stratégies, nous en avons développé d'autres. La première stratégie, StrategiePolygone, permet au robot de dessiner le polygone souhaité. La deuxième, StrategieAvancerDroitMax, fait progresser le robot le plus vite possible vers un mur et l'arrête juste avant l'impact. La stratégie strategieIASimule permet, elle, au robot de contourner un objet; en effet, lorsqu'il y a un objet devant lui, le robot effectue une rotation afin d'éviter l'obstacle. Enfin, une stratégie SuivreBalise fait tourner le robot sur lui-même jusqu'à détecter la balise, et une fois qu'il l'a trouvé, le fait s'avancer vers elle.

Pour implémenter une simulation, on initialise donc les stratégies utilisées, puis, à l'aide de threads, on met à jour les informations, ce qui nous permettra de voir au fur et à mesure les instructions réalisées.
Nous avons mis en œuvre deux étapes de validation de nos stratégies.

La première étape consiste à utiliser un simulateur qui exécutera les différentes stratégies et permettra de représenter sur le terminal en 2D le déplacement du robot dans un terrain continu prédéfini. On déclare donc un robot et un terrain continu virtuels sur lequel se déplacera le robot et on peut alors vérifier le bon fonctionnement d'ensemble mais cela ne reste qu'une simulation et quand tout est opérationnel, il faut alors passer à la deuxième étape, qui est de tester notre code directement sur le robot.
Ce dernier se déplace dans le monde physique dans lequel il se trouve.
Il n'est pas possible d'utiliser le même code sur le simulateur et sur le robot. On devra donc avoir deux codes distincts en réécrivant des fonctions distinctes pour chaque stratégie pour pouvoir bénéficier des deux représentations (simulée ou IRL).
On définit donc un Wrapper qui va servir à traduire le code de la simulation pour que le robot IRL comprenne les différentes tâches à réaliser. Pour cela, le Wrapper utilisera directement les commandes du robot.

Gif de l'exécution d'une des simulations, Avancer Droit Max.



2. Représentation Graphique à l'aide d'Unity :

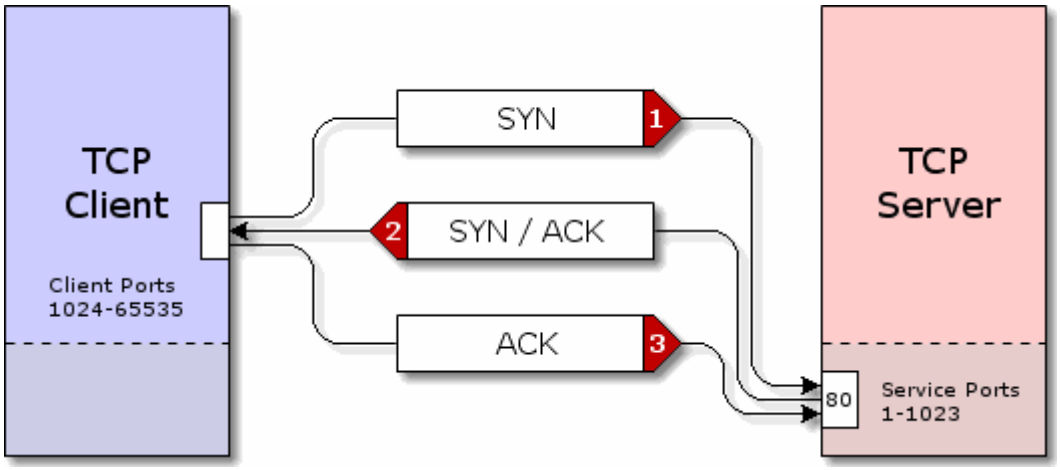
Pour obtenir une représentation virtuelle en 3 dimensions de notre robot, nous avons utilisé le logiciel Unity.
Pour pouvoir observer le déplacement du robot, on lance tout d'abord une simulation Python, dans laquelle notre robot peut réaliser diverses stratégies (ici, se déplacer de telle sorte à dessiner un carré).
Ensuite, on doit pouvoir récupérer les différentes informations de la simulation et les transmettre à Unity. Pour effectuer cela, nous utilisons un "protocole de contrôle de

transmissions“, autrement dit, une connexion TCP. Cette dernière permet à la simulation de communiquer les données nécessaires à l'exécution du script dans Unity.
On envoie régulièrement les informations de la simulation que l’on récupère dans Unity grâce à l’utilisation de threads.
Pour représenter le déroulement de la simulation, on crée les classes qui stockeront les données des différents objets. La deserialisation se fait à l’aide de Json, par le biais de la connexion TCP.
Après cette étape, on dispose de tout ce dont on a besoin pour commencer la représentation graphique.
On instancie tout d’abord le terrain, puis les différents éléments qui le composent tels que le robot ainsi que les obstacles. Les dimensions de ces objets définies au lancement de la simulation sont conservées. Le terrain est représenté par un plan, le robot par une sphère, et les obstacles par des cubes. Nous utilisons des objets simples pour la représentation. Nous n’avons pas modélisé un véritable robot qui évoluerait dans un monde avec les propriétés physiques du monde actuel.

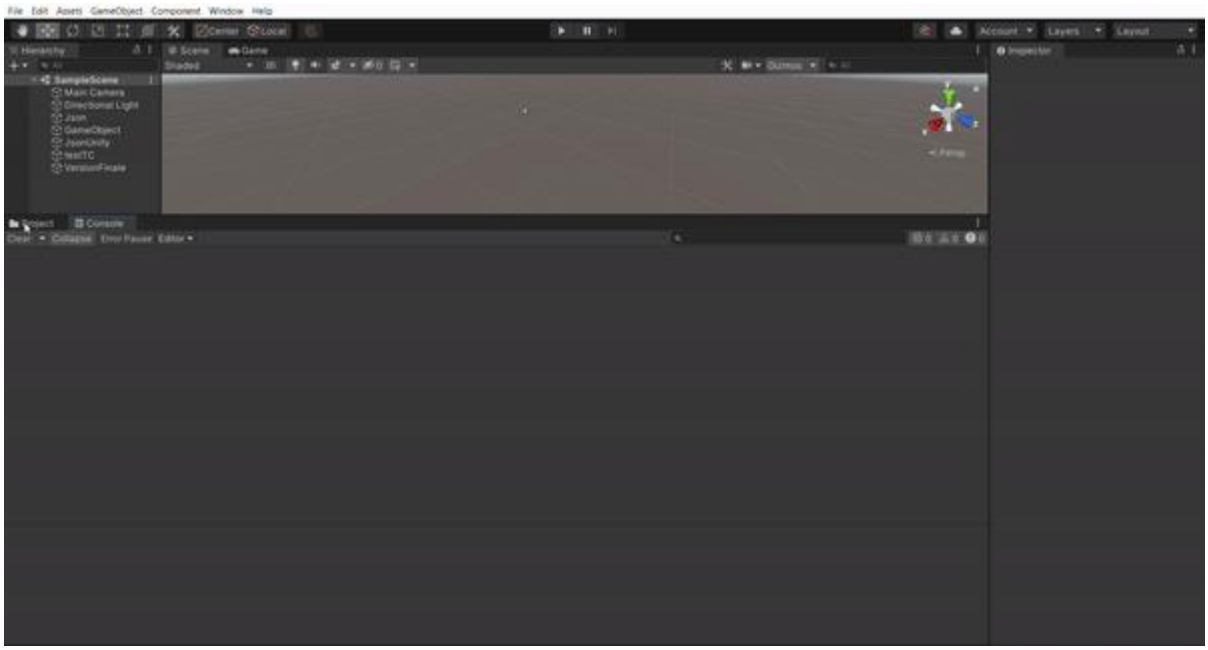
Pour que la représentation graphique se fasse, il faut dans un premier temps lancer la simulation sur python, puis passer sur Unity et y lancer le script.

Gif représentant le parcours du robot (boule blanche) effectuant un carré.
(La boule rouge correspond au point de départ)

Gif et schéma descriptif d'une connexion tcp :

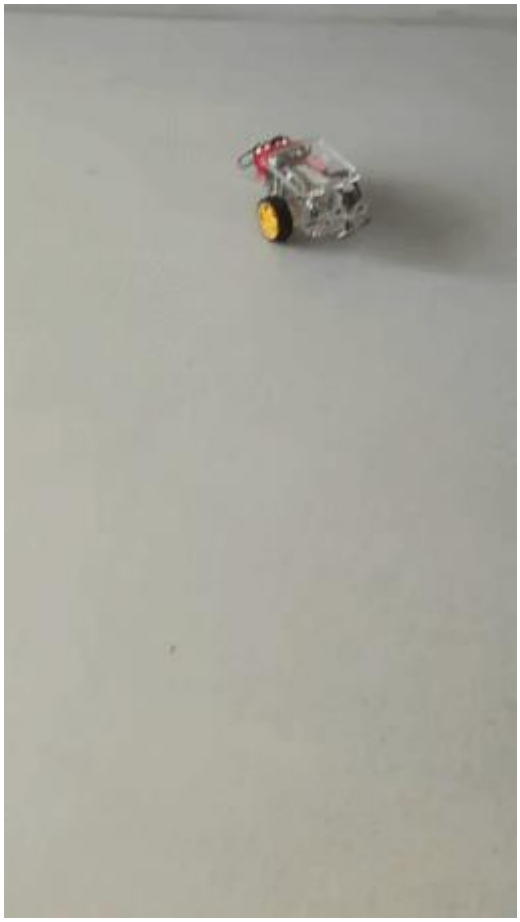


SYN = demande de synchronisation/connexion
ACK = accusé de réception



La connexion peut se faire sur un même poste, ce qui est notre cas ici.

3.représentation de la stratégie polygone sur le vrai robot (polygone effectué est un carré) :



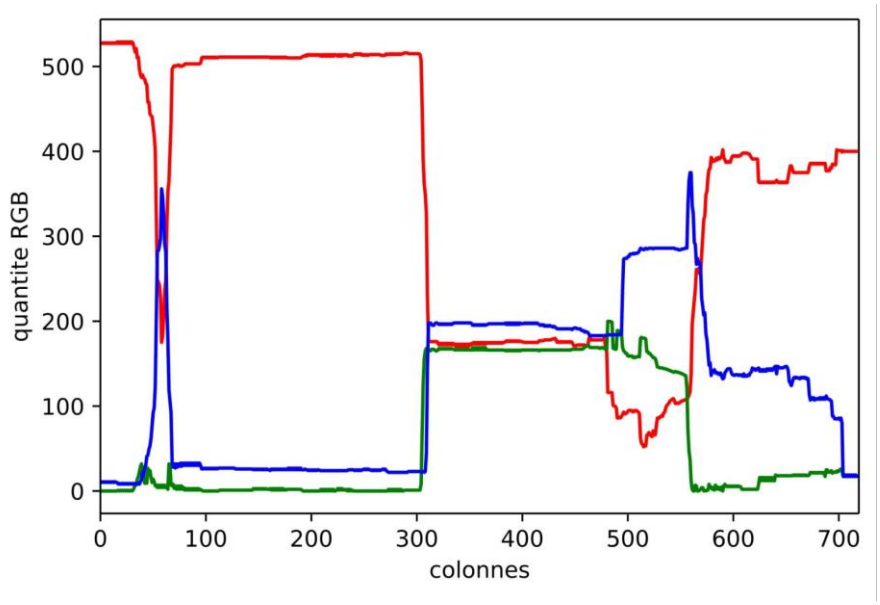
IV Capacités de l'IA

Pour l'intelligence artificielle du robot, nous voulions qu'elle réalise des actions simples. C'est dans cette optique que nous avons développé une stratégie qui déplace le robot de manière rectiligne jusqu'à ce qu'il rencontre un obstacle à une distance inférieure à 25 cm. Dans ce cas, il opère une rotation jusqu'à ce que soit il trouve une direction dans laquelle il n'y a pas d'obstacles proches de lui ou alors qu'il ait opéré une rotation complète, de 360°, sans qu'il puisse trouver de directions où se diriger. Dans le dernier cas, cela signifie que le robot est bloqué, il est entouré d'obstacles qu'il ne peut pas éviter. La stratégie s'arrête alors puisque le robot ne peut se déplacer.

Détection balise

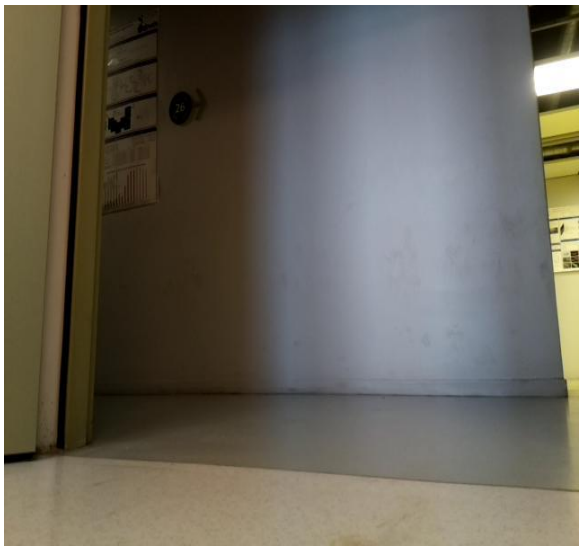
Une balise est constituée de quatre couleurs comme on peut le voir à l'image.
Nous avons fait une stratégie permettant de détecter ce format de balise et d'avancer vers celle-ci.

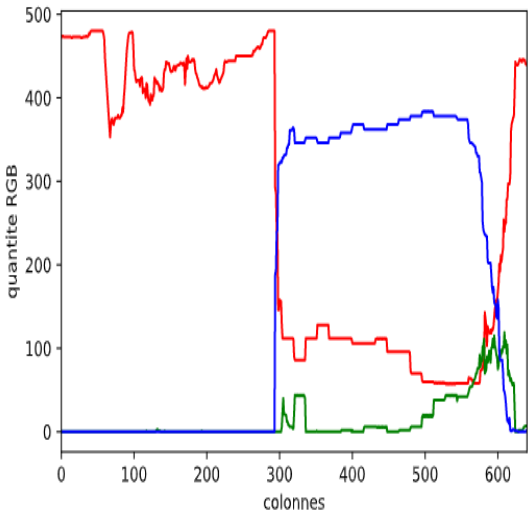
Pour ce faire, on a utilisé numpy. Avec cette librairie nous avons pu créer un histogramme de couleurs RGB en fonction des colonnes de l'image.



Pour détecter une balise la logique est la suivante:

On repère si il y a une zone suffisamment large où il y a du bleu et du vert, tous deux au-dessus d'un certain seuil de quantité GB décidé arbitrairement.
Une fois cette zone détectée, on regarde dans les colonnes antérieures; si on a plus de rouge que de bleu sur une largeur équivalente on en déduit qu'il y a une balise.





Nous somme parti du principe qu'il n'y a pas de balise si ses conditions ne sont pas respectées. Par exemple pour cette image on peut repérer une large zone de bleu et vert, en revanche le seuil arbitraire pour le vert n'est pas respecté par cette zone. On en déduit qu'il n'y a pas de balise.

Gif représentant le robot effectuant la stratégie SuivreBalise .

V Participation au code

Voici succinctement la participation au projet des différents membres du groupe.
Tout d'abord, pour le terrain représenté sous forme de grille, Romain a réalisé le constructeur de la classe ainsi que la méthode pour afficher ce dernier puis ensuite l'a modifié pour afficher le robot de manière particulière. Il a aussi créé une méthode pour supprimer l'objet d'une case particulière ainsi que d'ajouter des objets depuis des

coordonnées réelles. Mohamed a créé une méthode pour ajouter un objet particulier sur le terrain ainsi qu'une autre qui crée un certain nombre d'objets sur le terrain. Les tests ont été réalisés par Ivan et Julie.
Laurent a créé une classe Vecteur avec les méthodes norme et produit scalaire et Ivan a ajouté une méthode pour retourner un vecteur qui suivant une rotation ainsi qu'une méthode collision.
De plus, Romain a créé le constructeur de la classe Robot auquel Ivan a implémenté une méthode pour faire tourner le robot en mettant à jour son vecteur déplacement ainsi que son angle et Mohamed a ajouté une méthode qui déplace le robot pendant un certain temps. Les tests ont aussi été réalisés par Mohamed, Ivan et Julie.
Ivan a ensuite modifié le code pour le réorganiser afin de le rendre plus clair et lisible en utilisant des modules et a réécrit les tests en utilisant unitest.
Ensuite, Romain a ajouté une méthode pour supprimer l'affichage du terrain. Nous avons eu des problèmes puisque certaines solutions ne fonctionnaient pas sous Windows et d'autres sous Linux ou Mac. Nous avons ensuite réussi à trouver une solution qui permettait de supprimer l'affichage sans considération pour le système d'exploitation.
Pour développer le terrain, Ivan a développé une classe Polygone avec les méthodes de la classe vecteur.
Le constructeur de la classe Terrain Continu a été réalisé par Romain et a aussi développé la méthode pour ajouter des obstacles sous forme de polygone sur le terrain ainsi que le robot.
Par la suite, les tests seront pris en charge majoritairement par Ivan et Julie, nous ne les mentionnerons plus.
Ivan a développé une méthode de collision pour polygone ainsi que pour le terrain et pour le robot.
Mohamed a créé plusieurs méthodes pour générer des polygones simples tels que carré ou triangle.
Romain a développé la méthode pour transformer un terrain sous forme de grille en terrain continu et était aussi en charge de merge les changements dans la branche master quand le code était fonctionnel et approuvé par les membres du groupe.
Mohamed a ensuite été en charge de simuler le robot afin qu'il accélère jusqu'à rencontrer un mur et s'arrêter en face. Il a, plus tard, développé aussi la méthode pour le robot physique.
Julie a développé une stratégie pour que le robot représente un carré.
Romain et Mohamed ont développé les méthodes pour serializer et deserializer le terrain continu.
Romain a aussi développé les classes RobotIRL et Wrapper qui interagit avec le robot. Nous avons rencontré des problèmes pour la vitesse du robot à cause d'une conversion dans la taille des roues du robot.
Ivan a développé l'analyse d'image afin de pouvoir trouver la balise et d'aller à son encontre ainsi que la stratégie qui va de paire.
Julie et Mohamed ont développé la simulation en 3D sur Unity, ce qui a consisté d'abord à représenter le terrain puis a implémenté la stratégie dessiner un carré avec et sans obstacles.
Romain a développé l'intelligence artificielle pour le robot physique.
Mohamed l'a implémenté pour la simulation.
Julie, Mohamed et Romain ont contribué au compte-rendu.

VI Les initiatives entreprises

Tout d'abord, afin de faciliter le travail en équipe ainsi que les communications, nous avons créé un serveur discord. Celui-ci regroupe plusieurs salles, la première s'intitule « général » et contient nos discussions sur le développement du projet, nos difficultés rencontrées, ce que nous souhaitons faire.
La seconde se nomme « code » et contient des fonctions pour lesquelles nous avons des problèmes, alors un ou plusieurs membres du groupe aident la personne en difficulté que ce soit en l'aiguillant, en lui proposant des alternatives, en lui envoyant des liens vers des sites pouvant le débloquer ou bien en lui disant ce qui ne va pas.
Nous avons défini la troisième salle « recherches » et regroupe des liens vers des articles et des forums pouvant nous être utiles pour le projet.
Enfin, le dernier salon se nomme « git » dans lequel nous avons intégré un bot qui nous envoie un message à chaque fois qu'un membre du groupe push son code sur github. C'est très pratique pour savoir directement si quelqu'un a modifié une partie du projet et la quelle. En effet, le bot affiche aussi la description du push faite par l'utilisateur.

D'autre part, afin de représenter la simulation en 3D, nous avons choisi d'utiliser Unity comme moteur graphique. En effet, c'est une plateforme phare de la simulation 3D, très utilisée par les développeurs de jeux vidéo entre autres. En effet, les exemples les plus connus sont Temple Run, The Forest ou bien Rust. De plus, elle est simple d'utilisation et contient une version bêta supportée par python.

Enfin, pour permettre au robot physique de suivre une balise nous avons choisi d'utiliser la bibliothèque numpy qui permet de transformer une image en matrice afin de l'exploiter. Nous récupérons ainsi l'image que doit traiter numpy à l'aide de la bibliothèque Image.

VII Les tests

Nous avons testé le code, principalement la partie simulation.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Tous droits réservés.

Testez le nouveau système multiplateforme PowerShell https://aka.ms/pscore6

PS C:\Users\33637\Documents\RobotSU> python3 -m unittest discover .
..................................................
----------------------------------------------------------------------
Ran 51 tests in 86.116s

OK
PS C:\Users\33637\Documents\RobotSU>
```

Comme on peut le voir à l'image, tous nos tests sont valides. Il y a au total 51 fonctions de tests unitaire, un run de tests prend plus d'une minute dû à l'aléatoire et à la taille assez grande des terrains.

VIII Conclusion

En conclusion, ce projet nous a permis de renforcer nos connaissances en python. De plus, nous avons découvert ce qu'était de développer un projet en informatique en groupe de taille moyenne. Ainsi, nous avons pu nous familiariser avec l'utilisation de certains logiciels pour les travaux de groupe comme Trello qui nous a permis de bien nous séparer le travail et de connaître les tâches choisies par chaque membre du groupe et de leur état d'avancement ou bien github qui nous a permis de travailler tous sous la même version du projet. D'autre part, le fait que nous puissions nous rendre à l'université afin de tester nos différentes stratégies sur le robot nous a permis de rendre plus concret notre code et de garder malgré tout un lien social pendant cette pandémie. Nous en gardons tous un très bon souvenir et une très bonne expérience.