



# LU2IN013

CesBogoss

- SADI Amayas 28717408
- KOLLI Hamid 28717594
- CARAUX Théophile 28602627
- DROUARD François-Xavier 3800028
- MBOURANGA Winn 287178594

## Rapport du projet

Le but du projet est de déplacer un robot GOPIGO et de lui permettre de reconnaître des obstacles afin d'éviter des collisions ou de suivre un chemin ou une balise.

## Objectifs

1. Développer une simulation pour nos tests.
2. Tester nos fonctions sur notre robot en réel.
3. Travailler en équipe sur un projet.

## Caractéristiques

Apprendre à travailler en équipe sur un projet afin de développer une simulation pour éviter tout endommagement sur notre robot réel. Pouvoir implémenter de nombreuses lignes de code pour créer une interface de simulation qui se rapproche le plus de notre modèle réel. Savoir s'organiser avec les autres pour ne pas perdre de temps.

## Grandes étapes

### I. Savoir où on va

Choix du langage de programmation, établir une stratégie de travail, un fil conducteur à suivre pour savoir où l'on va, se mettre à jour sur le code choisi et choix de la future interface.

### II. Développement de notre simulation

Écrire le code en fonction des idées de notre simulation, création de nos objets simu, implémentation des stratégies, tests sur l'interface.

### III. Passage au réel

Passage du code en mode réel, vérification du code sur notre robot, tester le robot dans de vraies conditions, réfléchir à de nouvelles idées de stratégies.

## I. Savoir où l'on va

- Langage de programmation : Python
- Outils d'interactions : GitHub, Replit, Discord, Trello
- Traitement d'image : OpenCv
- Interface : pygame

Nous avons choisi Python car c'était le langage le plus utile pour ce projet, il a donc d'abord fallu tous se mettre à jour sur la POO en Python. Ensuite, il fallait savoir comment on allait s'organiser, vers quelles idées nous allions.

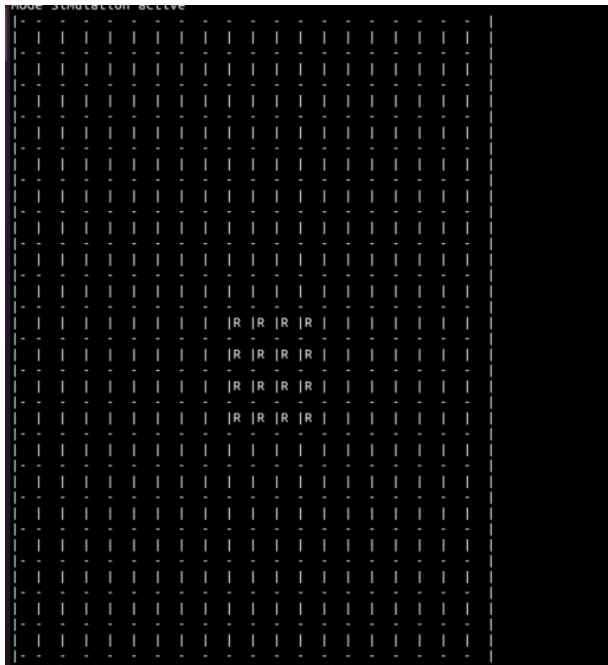
*Sous quelle forme implémenter notre simulation ? A quoi va-t-elle ressembler ? Comment s'organiser ?*

Nous étions donc partis sur l'idée de faire plusieurs classes pour chacun de nos objets :

1 classe pour implémenter l'arène

1 classe pour implémenter les obstacles

1 classe pour implémenter le robot

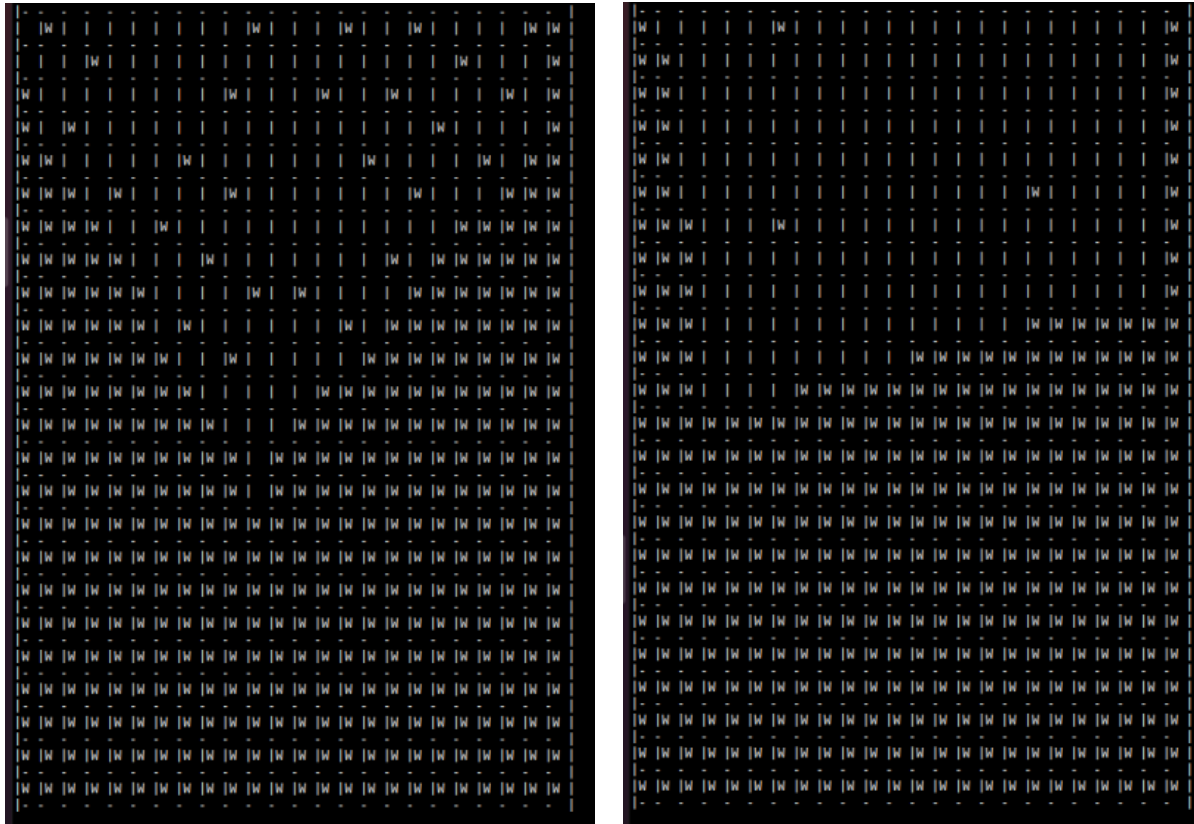


Nous avons donc pour idées de départ :

- 1) Représenter l'arène sous forme de grille et de positionner les objets (robot/obstacle) à des positions (i , j) de l'arène.
- 2) Créer une Vision qui représentait la vue de la caméra du robot en fonction de la position du robot sur l'arène et de l'angle du robot.
- 3) Un robot avec ses méthodes de déplacement

Nous avons donc une simulation avec une grille et un robot qui s'y déplace tout en remarquant les obstacles grâce à sa vision.

Voici deux exemples de la vision du robot avec notre grille(la tête du robot tout en haut) :



Le robot avait donc une bonne vision en fonction de son orientation sur l'arène. On voit ici qu'il avait trouvé un mur(représenté par des W) devant lui, mais certains murs apparaissent sur des cases de la grille par soucis de précisions.

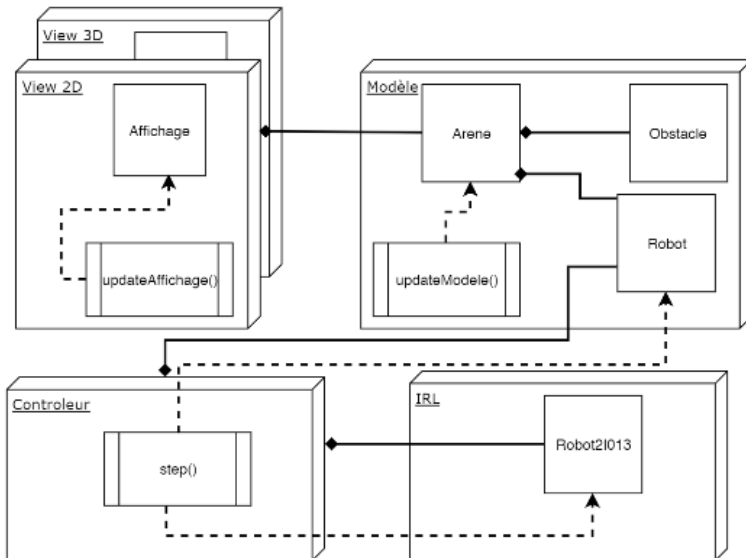
**Mais alors sont arrivés 3 problèmes :**

1. Notre code n'était pas asynchrone.
2. On travaille sur un monde discret alors que le monde réel est continu.
3. Nous nous sommes aperçu qu'avec la grille, nous avions des soucis de précisions, qui pouvaient s'avérer problématique dans certaines conditions réelles.

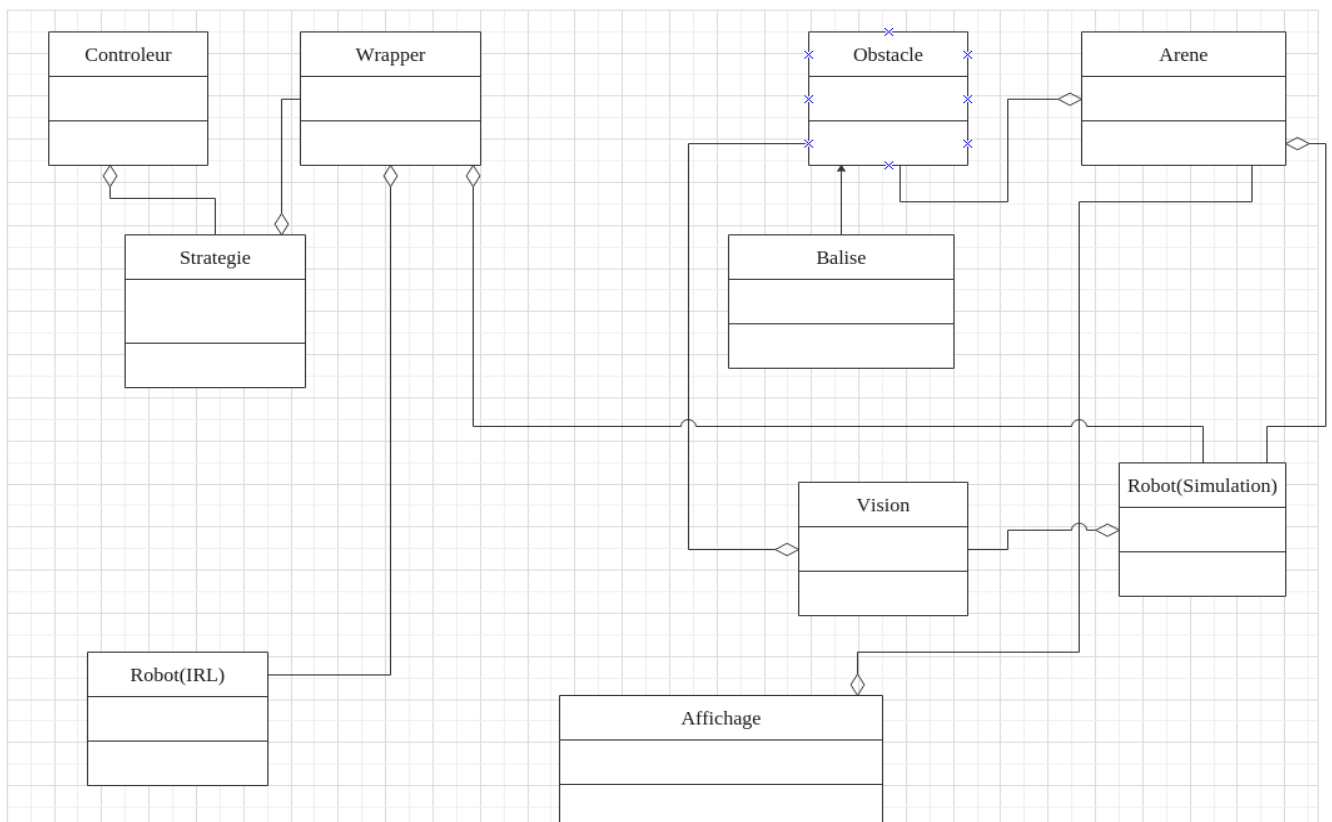
Nous avons donc recommencé tout notre code tout en gardant nos idées de base. Nous l'avons ainsi factorisé et avons repensé toutes nos classes. Nous sommes donc passés avec des coordonnées sous forme de liste, plus utiles pour notre interface pygame et plus précises que la grille.

## II. Développement de notre simulation

On utilise le pattern MVC, en suivant le modèle du cours (ci-dessous).



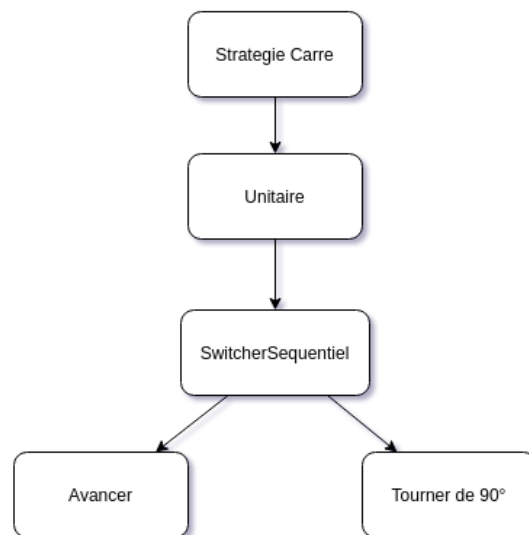
Que nous avons adapté à nos besoins. Ci-dessous notre structure globale du projet.



## A. Explication de l'architecture

Le **contrôleur** sert à contrôler le robot, il contient toutes les stratégies. Les stratégies se partagent en plusieurs types :

- *Élémentaires*, stratégies de bas niveau, qui font des choses simples comme avancer d'une distance et tourner d'un angle
- *Méta-stratégies*, stratégies qui servent à créer d'autres stratégies à partir des stratégies élémentaires. On trouve l'unitaire qui enveloppe une autre stratégie par une condition d'arrêt (amélioration de la stratégie). Ou encore le switcher qui alterne entre deux stratégies suivant une fonction de condition. On a aussi le switcher séquentiel qui est un switcher qui alterne les deux stratégies n fois.
- *Autre*, elles sont construites à partir des autres stratégies en les reliant par les méta-stratégies. Ainsi, pour former un arbre de stratégies comme l'explique le schéma suivant avec la stratégie carré :



Les stratégies agissent sur le robot avec un wrapper qui est une classe qui sert de classe commune entre le robot irl (robot2I013) et le robot de la simulation pour but d'unifier les appels et les ordres aux robots et aussi pour envelopper les fonctions bas niveau du robot et les rendre plus haut niveau.

Le **modèle** est composé par l'arène et le robot. Le robot de la simulation est un miroir du robot réel, il contient les mêmes attributs et méthodes essentiels que celui-ci. Il possède lui aussi deux roues et un servo. L'arène se charge de calculer les nouvelles positions du robot et donc son déplacement dans l'arène en fonction du temps et de la distance parcourue par ses roues, elle représente donc la physique du projet. Elle permet aussi de stocker les obstacles, qui sont représentés par des segments ( deux points src/dst ).

Pour la détection des obstacles, le robot se base sur une vision qui représente une partie de l'arène vu par le robot suivant son servo. En calculant la distance entre le servo du robot et l'obstacle le plus proche, on peut avoir exactement la même implémentation que le réel de la fonction `get_distance`. A chaque pas de temps, la vision se synchronise avec l'arène en récupérant les obstacles qui sont en face du robot.

Pour **l'affichage**, en utilisant pygame on a réussi à faire un affichage 2D qui affiche de manière fluide les interactions et les mouvements fait dans l'arène, on représentant les obstacles par des segments en jaune et le robot par un carré de segments bleu et sa tête en rouge. A chaque pas de temps (1/FPS) on actualise les dessins. L'affichage sert aussi à tracer le parcours du robot dans l'arène en utilisant un crayon défini dans le robot.

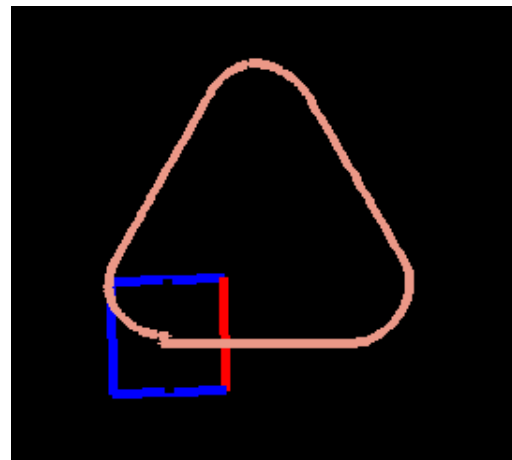
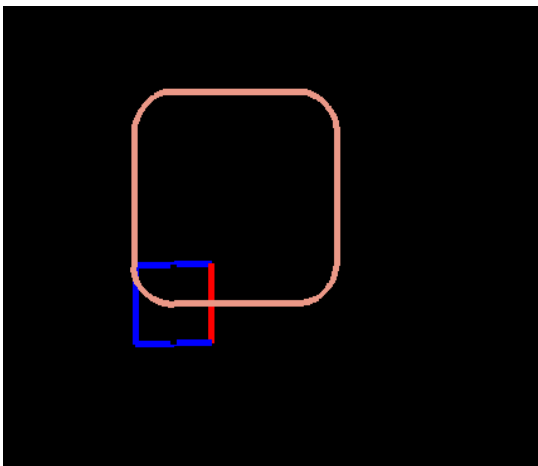
## B. Tests et Résultats

- **AvancerAuMur :**

Le robot s'arrête bien lorsqu'un obstacle lui bloque le chemin, on peut réduire la distance de sécurité pour qu'il s'approche le plus proche possible des obstacles avant de s'arrêter.

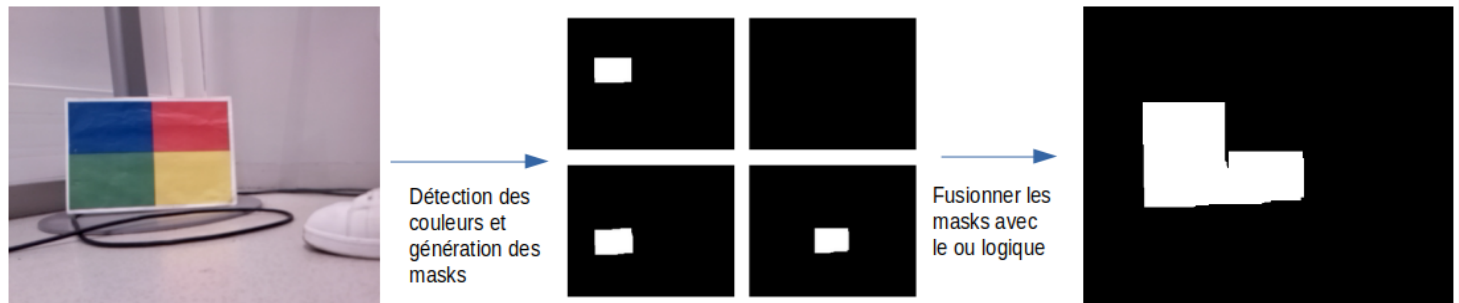


- **Formes géométriques**



Notre robot avance bien dans notre simulation, il peut dessiner des formes géométriques comme un carré, un triangle et des polygones. Tous ses déplacements sont bons peu importe sa vitesse.

- Traitement d'image



Pour le traitement d'image, nous avons décidé d'utiliser opencv qui est un module de python un peu puissant pour ce genre de traitement. Tout d'abord, on récupère les endroits où la couleur recherchée est présente et on crée une nouvelle image appelée masque qui est un fond blanc (où se trouve la couleur) sur un fond noir. Ensuite, on supprime les petits détails (les petits points de couleur) en utilisant deux méthodes (dilate et erode).

En utilisant cette méthode sur les 4 couleurs de la balise on aura 4 masques différents, en utilisant un "ou logique" entre les 4 masques on aura un masque qui englobe les 4 couleurs, ce qui représente la balise.

Pour savoir si la balise est dans l'image, en récupérant les masques on calcule le nombre de mask où le blanc est présent. Si on a 3 couleurs présentes parmi les 4, la balise est donc bien présente. La fonction 'get\_angle\_orientation\_balise' du wrapper sert à détecter la balise et à renvoyer l'angle et le côté où la balise se trouve dans l'image (si elle s'y trouve bien sûr).

Pour la stratégie 'SuivreBalise' on fait tourner le robot à 360 degrés jusqu'à ce qu'il trouve la balise et avance tout droit vers elle. Vu que le 'get\_image' du robot IRL prend un peu de temps pour récupérer l'image (il crée un petit bloque pour le thread du contrôleur et tarde un peu les autres traitements) on a préféré l'isoler dans un autre thread (une autre classe) qui prend à chaque fois une photo (get\_image du robot) et on récupère directement la dernière photo prise par le robot pour la traiter.

Le robot suit bien l'image de la simulation et tourne à l'infini tant qu'il ne l'a pas trouvée, s'il trouve l'image alors il se dirige directement vers elle.



- **Stratégie personnelle**

Concernant la stratégie personnelle, on a décidé d'implémenter une stratégie qui récupère une image (si c'est en simulation c'est une image setter dans le main, sinon on récupère une image en utilisant le `get_image`) qui contient une forme géométrique (un polygone), et si le robot arrive à trouver le nombre de côté du polygone, il le dessine en allumant les led en vert sinon il attend que quelqu'un lui mette une image à la caméra qui contient une forme en allumant les led en rouge.

### C. Conclusion de la simulation

Notre simulation fonctionne comme on le souhaitait, notre robot peut se déplacer librement dans notre arène et est capable d'éviter les obstacles de l'arène, il peut aussi suivre nos stratégies afin de dessiner des formes géométriques ou même de suivre une image.

Notre simulation est bien asynchrone comme on le cherchait et toutes nos fonctions sont correctes. Petit malus : notre simulation est en 2D.

## III. Passage au réel

Le robot est capable de réaliser quasiment toutes les mêmes fonctionnalités qu'en simulation. Il se déplace comme souhaité, détecte les obstacles et évite les collisions. Il est capable d'effectuer les stratégies des formes géométriques. On a juste un petit problème de précision lors de la rotation du robot c'est que vu que l'on utilise les thread et on calcule à chaque fois la distance qu'il nous reste à parcourir pour tourner on a un manque de précision à la fin de l'opération (soit il tourne un peu plus ou un peu moins).

Le problème est sur le traitement d'image, le robot repère bien notre balise malgré quelques fois des problèmes de luminosité. Cependant, est apparu un problème, le `get_image` du robot prend trop de temps à renvoyer l'image de la balise.

C'est à dire que, le robot tourne sur lui même comme souhaité jusqu'à trouver la balise, sauf que quand le robot trouve l'image devant lui, le temps que le `get_image` finisse, le robot a continué de tourner sur lui même et a donc dévié de plusieurs degrés de la balise. Le robot avance donc après avoir continué de tourner puis recherche la balise car elle n'est donc pas devant lui.

## IV. Conclusion


Nous avons donc une simulation qui fonctionne bien comme on l'avait souhaité, notre robot exécute bien toutes nos stratégies dans l'arène.


Le robot arrive donc à bien exécuter les fonctions et trouve bien notre balise mais sa fonction `get_image` fait perdre du temps et avec les thread qui tournent en fond le robot dévie de sa trajectoire vers la balise, mais il trouve bien la balise


## V. Participation

**Hamid :** Dans la grille j'ai fait le `forward` qui est la fonction qui fait bouger le robot (en synchrone) , et `libresur` qui permet de retourner si le robot peut passer sur une certaine distance, après le passage au continue, j'ai fait le `update` de l'arène (ce qui permet de bouger le robot), le `debug` dans l'affichage, la stratégie `triangle` et `touner`. La stratégie `EviterObstacle` avec `amayas`, j'ai aussi aidé `amayas` dans le `get_distance`, dans la classe `robot` de simulation, la stratégie `SuivreBalise` (et détection de la balise avec `opencv`) et on a aussi développé le système des méta-stratégie et le `wrapper`. La stratégie `personnel` aussi on l'a fait à 4 : moi, `Amayas`, `Théophane` et `François-Xavier`. Pour les tutoriels, j'ai regardé la POO de python et les cours du prof, pour le `pygame` je n'est pas eu le temps de regarder les tutoriels youtube mais j'ai tout compris en lisant le code de l'affichage.

**Amayas :** Dans la grille j'ai travaillé sur le système de synchronisation de la vision qui sert à synchroniser la vision du robot avec l'arène. Et lors du passage au continue, j'ai recoder la fonction `sync_vision` pour l'adapter au continue, et avec l'aide de `hamid` on a fait la fonction de `get_distance`, ensuite pour les stratégies j'ai coder la première stratégie élémentaire `Avancer` et ensuite la stratégie `Carré` et `PolygoneRegulier` (qui est une stratégie du TME Solo que j'ai copié) et avec `Hamid` on a développé le système des méta-stratégies (`Unitaires` et les `swicher`) et aussi on a fusionner nos idées moi et `hamid` pour concevoir l'idée du `EviterObstacle` et aussi on a codé ensemble la détection de la balise donc la stratégie de `Suivi de balise` qu'on a testé après sur le robot de `François-Xavier`, et enfin pour la dernière stratégie on étais avec `Hamid`, `Théophane` et `François-Xavier` on a codé notre stratégie perso. Ensuite le fichier `tools` j'ai codé les différentes fonctions de manipulations des points, vecteur et droite avec l'intervention de `hamid` et `théophane` dans quelques fonctions. Et aussi pour le contrôleur avec `hamid` implémenté le fonctionnement de ce dernier, et aussi j'ai ajouté un système de dessin du parcours avec le `crayon` (c'est une autre fonctionnalité du tme solo). Après le développement de `SuivreBalise` on a trouvé une erreur avec le groupe c'est les attentes de `get_image` donc avec `hamid` on a changé le `get_image` en un thread qui capture continuellement, et j'ai aidé `hamid` dans la création de la structure de l'arene et le robot.

 Théophane : Dans la grille j'avais fait l'affichage de l'arène, les fonctions d'ajout d'obstacles ainsi que les classes d'obstacles et je m'occupais du trello et des compte rendu de la semaine. Puis est venu le passage en continue et il a fallu factoriser tout le code, j'ai aidé Amayas pour recréer les classes du modèle et les fonctions du tools que l'on utilisait déjà dans la grille. Je faisais quelques tests unitaires pour ensuite vérifier les calculs des nouvelles fonctions de déplacement en attendant de pouvoir les vérifier visuellement sur l'interface graphique. J'ai aidé Hamid et Amayas pour certaines stratégies du contrôleur comme Carré/Triangle car j'ai compris le fonctionnement des stratégies élémentaires. J'ai créé la classe d'affichage avec pygame( crée les segments des obstacles grâce à leurs coordonnées et afficher le robot) pour avoir une interface graphique asynchrone avec notre projet. J'ai aidé dans les debug des tests IRL sur le robot et j'ai aidé Amayas sur les vérifications du traitement d'image. J'ai appuyé Hamid, Amayas et François-Xavier lors de la dernière stratégie personnelle. Et enfin, j'ai fait le rapport du projet, très vite aidé par Amayas et Hamid.

 François-Xavier : Ayant principalement mis en place la structure du Projet au début, je me suis arrêté à la création du système de la vision et des manipulations sur le Robot IRL. J'ai aussi surtout aidé à coder Amayas et Théophane lors de nos multiples réunions pendant les moments de recherches et autres. Le système de config à été créé suite à l'envie général de refactoriser le code et j'ai choisie les .cfg afin de rendre la config plus lisible a l'œil. J'ai aussi été à l'origine de la stratégie personnelle, en ayant l'idée d'une stratégie qui détecte une figure devant le robot, l'analyse et la réplique. Énormément de problèmes se sont avérés sur le robot IRL qui m'a pris énormément de temps ces dernière semaine : plusieurs jours complets de travaille pour juste rendre compatible une librairie indispensable au fonctionnement du robot.

 Winn : En ce qui me concerne pour le projet, je me suis d'abord occupé des fonctions liées avec la vision du robot comme « Libresur » par exemple avec Hamid qui est devenu membre à part entière de la fonction « sync\_vision ». Ensuite, lors de la refactorisation du code pour le travail sur les listes, j'ai réalisé des fonctions avec l'aide d'autres membres du groupes portant principalement sur les déplacements du robot (fonction forward notamment) avec l'aide d'Hamid. J'ai notamment pris part aux réunions de recherche que nous avons eu avec le groupe. Ensuite je me suis occupé des tests unitaires des fonctions de calculs qui sont essentiellement dans tools.py mais que nous avons décidés d'enlever. Enfin, j'ai voulu prendre comme tâche la partie 3D mais par manque de temps et de bases solides sur Panda3d, je n'ai pas pu honorer cette tâche.