Projet tutoré

Sujet : Robot joueur de billard

Rapport fin de semestre

Groupe : CAPAR - DESSAULX - LATH - RETTER
Tuteur : M. Pierre-André Guenego

Année scolaire : 2023-2024

SOMMAIRE :

1. Introduction

1.1. Description du sujet

Le projet consiste à développer un robot autonome qui joue au billard. Cependant, selon l'avancement du projet, d'autre objectifs et fonctionnalités pourront être définie et ajouter (cf. 6.Partie exploratoire).
Ce robot sera équipé d'un marqueur Aruco permettant d'obtenir la position du robot grâce à une caméra. Le robot sera guidé avec une caméra fixe au plafond et un serveur web. En effet, ce serveur web va gérer la communication entre le client et le robot.

2.2. Etude technique

Architecture du projet :

- Le robot sera équipé d'une carte wifi (arduino esp) permettant de communiquer avec le serveur via un routeur wifi.
- Les images de la caméra seront envoyées vers le client pour ensuite être traitées par la librairie js aruco.
- Le serveur sera sous NodeJs.
- Le serveur, le client et le robot communiquent à l'aide des websockets pour maintenir un flux continu.

- Le simulateur est une partie indépendante mais importante qui vont permettre de tester les programmes avant de les mettre en place sur le robot.

NodeJS:

NodeJS est un environnement d'exécution JavaScript open source et multiplateforme qui se concentre sur les applications côté serveur et réseau.
Dans le cadre de notre projet, nous allons l'utiliser pour créer notre serveur. Ce serveur va recevoir les différentes informations du client web, effectuer tous les calculs et piloter les robots.

Arduino :

Un Arduino représente des cartes électroniques regroupant plusieurs composants électroniques afin de réaliser des objets électroniques interactifs.

Dans notre projet, tous les robots seront équipés d'un arduino esp8266. Nous allons utiliser ce modèle car celui-ci peut se connecter à un réseau WiFi. Ils seront connectés à la borne WiFi où est déployé le serveur NodeJS.

Routeur Wifi :

En tant que routeur Wifi, nous pouvons utiliser un simple téléphone. Ainsi, chaque partie du projet (le serveur NodeJS, le client web, arduino) sera connecté au même réseau WiFi. On pourra par la suite remplacer le téléphone par un Raspberry Pi.

Websocket :

Les websockets est une technologie qui permet d'ouvrir un canal de communication bidirectionnelle entre un client et un serveur. Dans notre projet, les websockets vont permettre des intéractions en temps réel et en flux continu entre le client et le serveur.

Js Aruco :

Librairie de réalité augmentée basée sur OpenCV utilisant des marqueurs. Ces marqueurs seront positionnés sur les robots. Cela permettra d'obtenir la position et l'orientation du robot dans la réalité.

JavaFX :

Framework Java permettant de créer des interfaces graphiques. Ce sera la technologie utilisée derrière le simulateur 2D.

2.3. Principales difficultés du sujet

Modélisation des Déplacements :

Définir avec précision les mouvements du robot en tenant compte de la cinétique et de la dynamique. En effet, il est difficile de programmer par nous-mêmes et de zéro les mouvements physiques d'un robot en javafx. Il nous faut définir une échelle réaliste du plan de jeu pour y placer un robot de la même taille qu'un vrai robot de même pour la boule. Ensuite, nous pourrons définir les mouvements de ce robot.

Interaction avec les Éléments Physiques :

Programmer les réponses du robot à son environnement, comme la détection d'obstacles, en utilisant des principes physiques. Cela sera fait avec la caméra et js-Aruco. Ici la difficulté est de bien initialiser le client pour qu'il puisse obtenir les bonnes positions des différents objets pour pouvoir les envoyer au serveur pour les calculs. De plus, dans le but d'un robot joueur de billard, il faudra analyser l'interaction entre le robot et la boule.

Problème de Latence :

Gérer les défis liés à la latence dans la communication entre le serveur et l'Arduino pour assurer des réponses en temps réel. Mais aussi entre la caméra, le client et le serveur. Il est connu que la connexion wifi est souvent instable et que de nombreux éléments peuvent interférer avec celle-ci. Ainsi, si le robot a de la latence avec le serveur cela pourrait créer

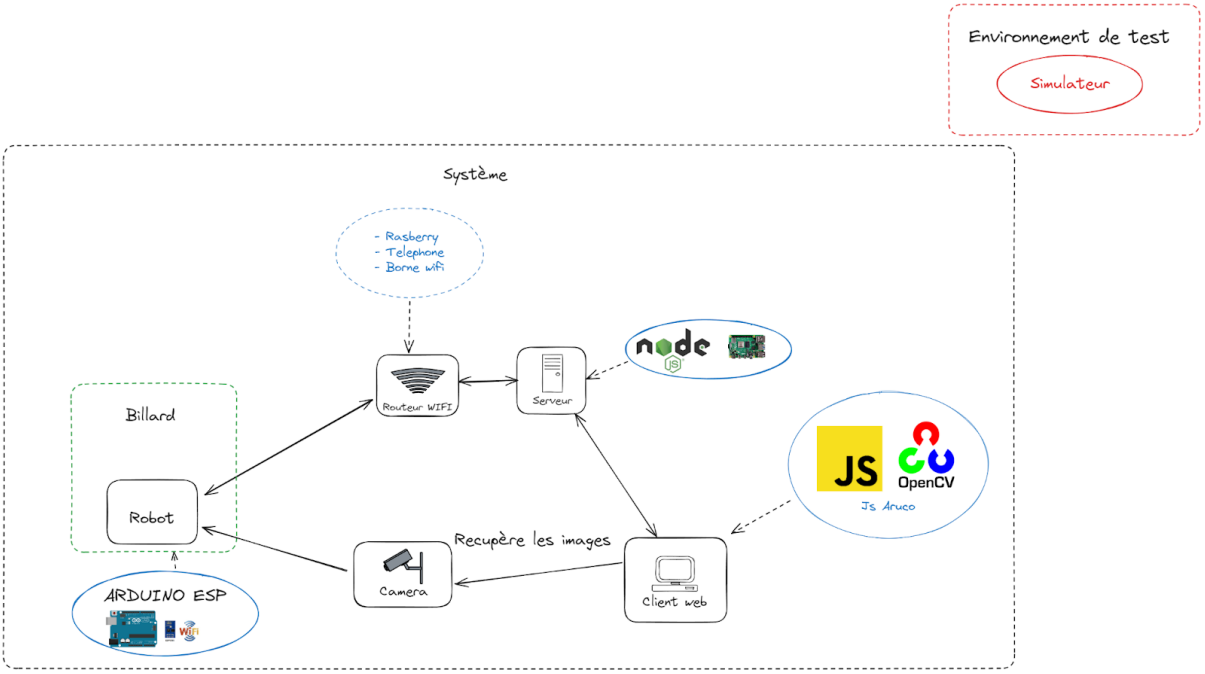
des problèmes. Par exemple, le robot pourrait recevoir l'ordre d'avancer mais qu'il ne reçoive jamais l'ordre de s'arrêter. De même avec la caméra, le client et le serveur car si le client envoie des positions qui ne sont plus bonnes/effectives alors le serveur fera de mauvais calculs et le résultat ne sera pas au rendez-vous.

Différence entre programmation et réalité :

Les ordres effectués au robot peuvent être différents de la réalité. Tout cela est relié aux problèmes de latence par exemple, mais aussi par rapport au simulateur. On sait que beaucoup d'éléments extérieurs peuvent changer l'attendu du résultat. Dans les calculs l'effet de friction ne sera pas pris en compte alors il se pourrait que le robot n'aille pas à la vitesse voulu par exemple. Aussi, il suffit d'un petit caillou sur l'espace de jeu du robot pour le faire changer de trajectoire inopinément. Donc de nombreux détails peuvent complètement changer les mouvements du robots et ils ne seront pas forcément prévisible ou pris en compte dans le projet.

2. Analyse

2.1. Architecture du projet



Notre projet se décompose en deux parties indépendantes le système et l'environnement de test.

L'environnement de test étant le simulateur :

- Le simulateur sera en premier temps une partie indépendante, et il permettra de pouvoir tester nos algorithmes de déplacement, simuler une partie de billard en prenant en compte les contraintes physiques et tester nos stratégies pour gagner au billard.

Le système est la partie qui va s'occuper des intéractions entre le robot, le serveur, la caméra et le client web:
Le robot :

- équipé d'un arduino esp étant microcontrôleur équipé d'une carte Wifi.
- sera connecté au serveur via un routeur Wifi.
- est capable de se placer dans une dimension dans l'espace de jeu (avancer et reculer).
- sera équipé d'un marqueur aruco.

Serveur :

- sera sous nodeJS, et sera sur un raspberry pi.
- sera relié au robot et un client web via des websockets.

Caméra :

- La caméra sera positionnée au-dessus de l'aire de jeu pour capturer les robots et les boules.

Client web :

- L'objectif du client web est de récupérer les informations de l'aire de jeu via la caméra.
- Le client sera sous javascript et aura des librairies comme OpenCv et JsAruco (utilisant des technologies d'OpenCv pour la détection de marqueur aruco). Ces librairies vont permettre d'analyser les images pour détecter les boules et le marqueur situé sur le robot.
- Il va permettre aussi de pouvoir déclencher le jeu et donner aux utilisateurs des intéractions.

2.2. Risques et solutions associées

Risques inévitables :

- Problèmes de connections robot - serveur
- Imprécision à la détection des robots/de la balle
- Calcul de trajectoire imparfait

Risques pour la réalisation du projet :

- Le backend java va nécessiter beaucoup de travail, notamment pour le calcul de trajectoire optimale et le fait d'emmener un robot au point désiré en évitant les obstacles sur le trajet (comme la balle) avec ses deux moteurs

Groupement de risque :

Les risque que nous ne pouvons pas éviter et pour lesquels il faut obligatoirement trouver des solutions :

- L'ESP ne capte pas le signal (Message en retard)
  => Bien écrire le programme : Effectuer un mouvement pour un temps donnée.
  partir d'exemple de références pour le C.

- Le robot ne va pas dans la direction souhaitée.
  => Faire un calibrage au démarrage de la partie (autocalibrage).

- Problème matériel, le robot n'arrive pas à jouer au billard.
  => Changement de sujet du projet.

- Une boule dérange le déplacement du robot.
  =>Décaler celui qui dérange ou bien contourner la boule.

- Le robot ne capte pas le signal wifi, coupure de réseau qu'on ne maîtrise pas. Dépassement de mémoire par exemple pour cause.
  => Reboot du robot.

- Problème de luminosité : table trop ou pas assez éclairée. Le système de reconnaissance peut confondre les boules aussi.
  => Gérer l'éclairage avec des lampes ou bien manipuler l'image pour accentuer certaines couleurs.

- Ne pas pouvoir détecter une boule avec une couleur spécifique comme une boule verte sur une table verte.
  => Utiliser des outils pour accentuer les couleurs ou bien changer la boule.

- Boule qui disparaît au cours du jeu.
  => Mettre les boules en tampon mémoire.

2.3. Fonctionnalités

Fonctionnalités serveur :

- Lancer serveur
- Mettre à jour les coordonnées des boules
- Mettre à jour les coordonnées du robot
- Calculer la trajectoire du robot
- Donner les ordres pivoter au robot (avec date)
- Donner les ordres avancer au robot (avec date)
- Manipuler robot manuellement

Fonctionnalités client web :

- Se connecter au serveur
- Détecter les boules
- Détecter le robot
- Envoyer cordonnées boules
- Envoyer cordonnées robot
- Sélectionner une boule cible
- Définir les trous du billard

- Prévisualiser les trajectoires des boules

Fonctionnalités Arduino :

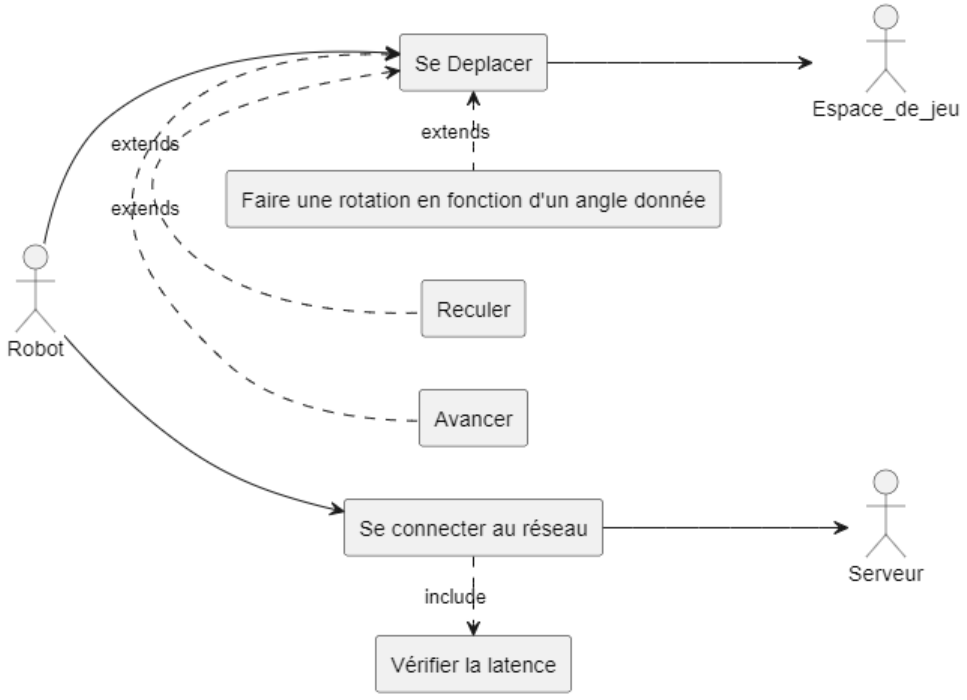
- Se connecter au serveur
- Vérifier heure du message
- Avancer
- Pivoter

Fonctionnalités Simulateur :

- Affichage des objets
- Affichage mouvements
- Calcul : trajectoire, rebond,..
- Avancer
- Pivoter
- Manipuler robot manuellement

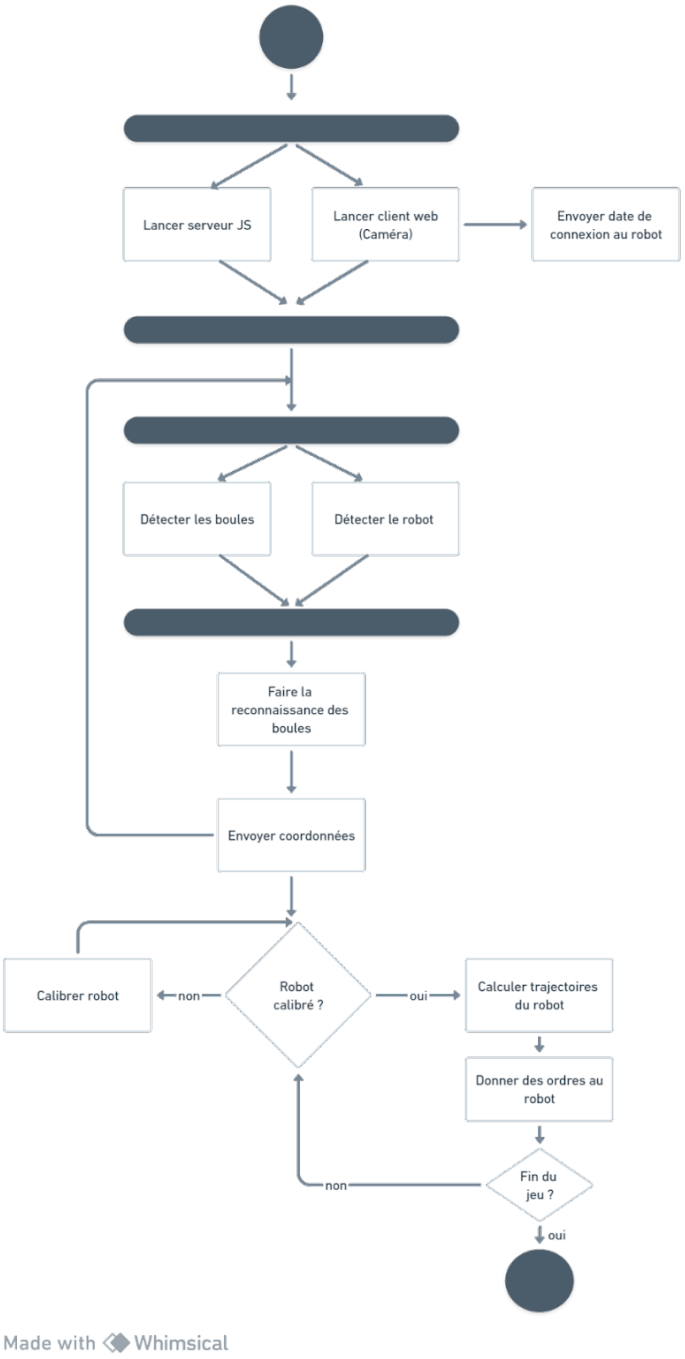
2.4. Diagrammes

2.4.1. Diagramme de Use Case du robot



2.4.2. Diagramme d'activités

Pendant l'étude préalable :

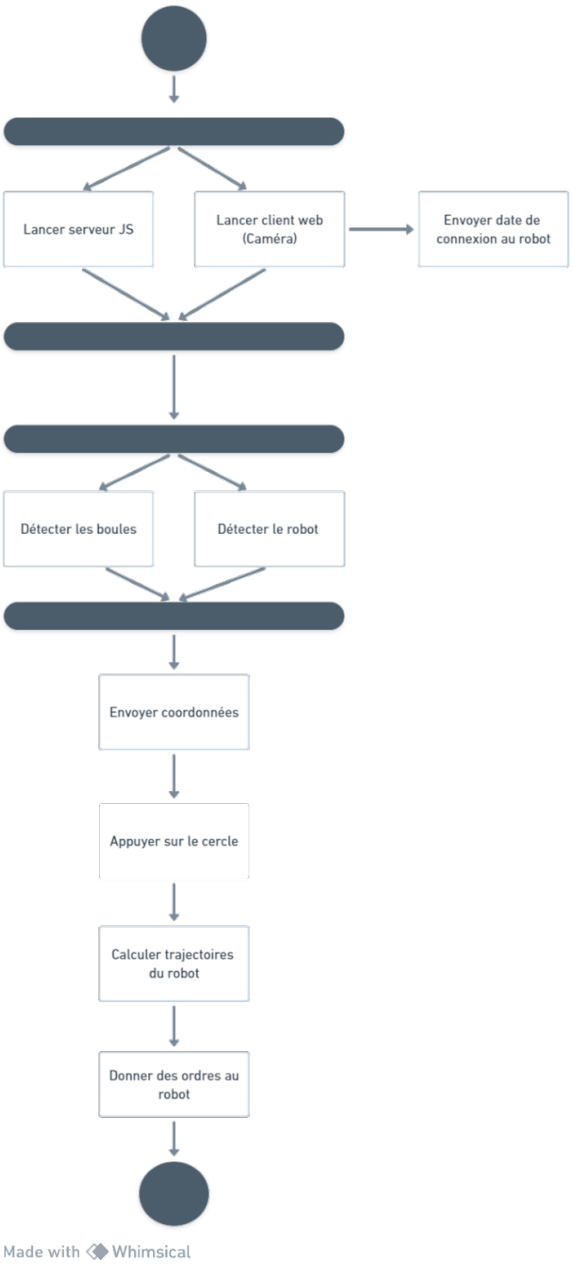


Ce diagramme représente l'activité globale du système, du lancement du client web et du serveur, puis à la détection des boules, du robot, et l'envoie de leurs coordonnées et enfin de la gestion du robot par le serveur notamment dans la gestion du calibrage et des ordres.

Pendant le projet :

Celui-ci n'est plus d'actualité car nous n'avons pas encore pu faire la reconnaissance des boules et nous n'avons pas de calibrage automatique. De plus, le programme ne sait pas encore connaître la fin de la partie.

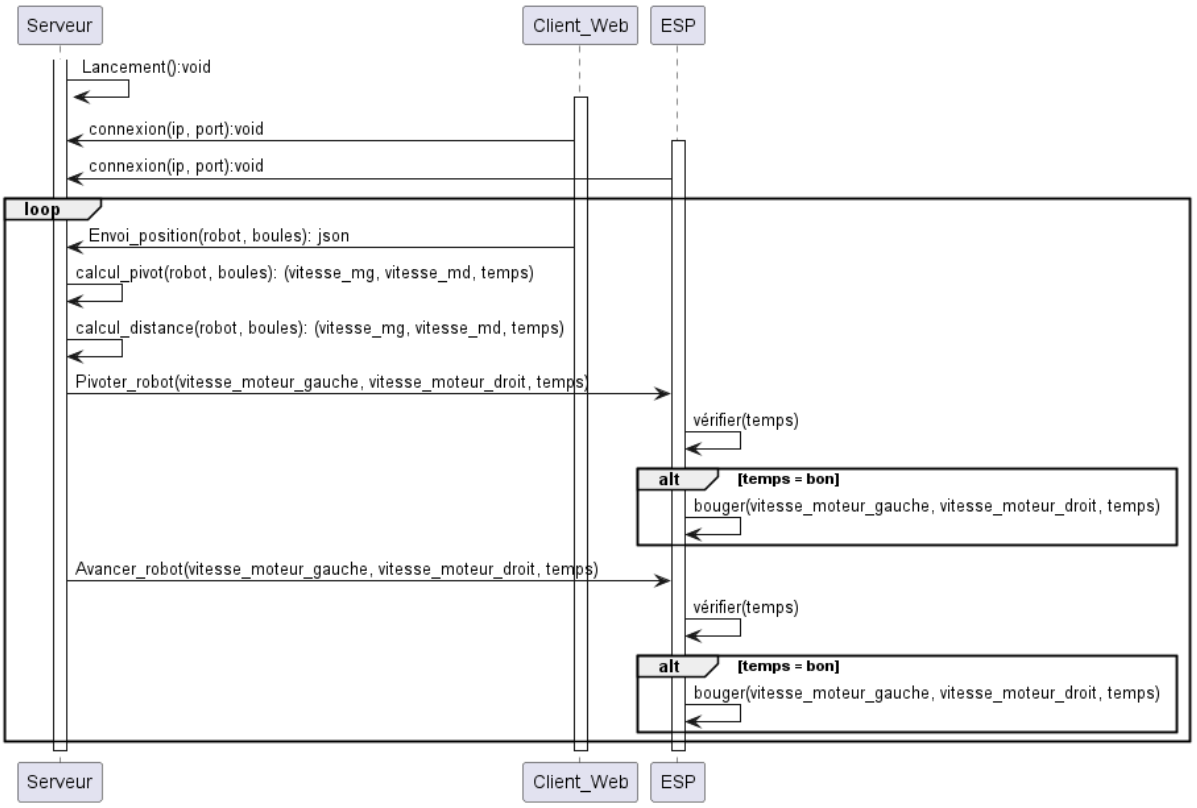
Voici le diagramme après les itérations :



Nous ne sommes pas parvenue à calibrer et à faire la reconnaissance individuelle des boules.

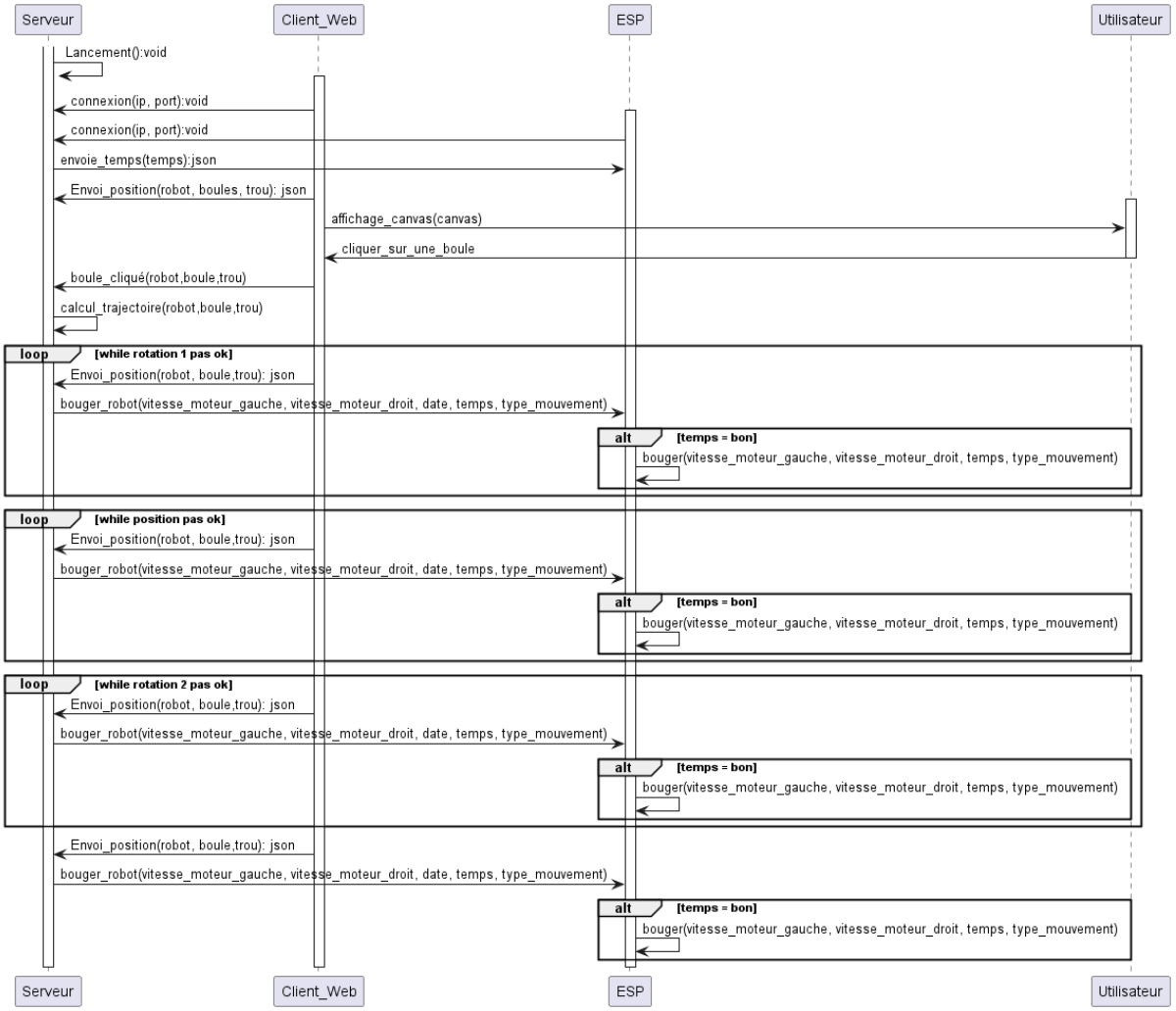
2.4.3. Diagramme de séquence

Pendant l'étude préalable :

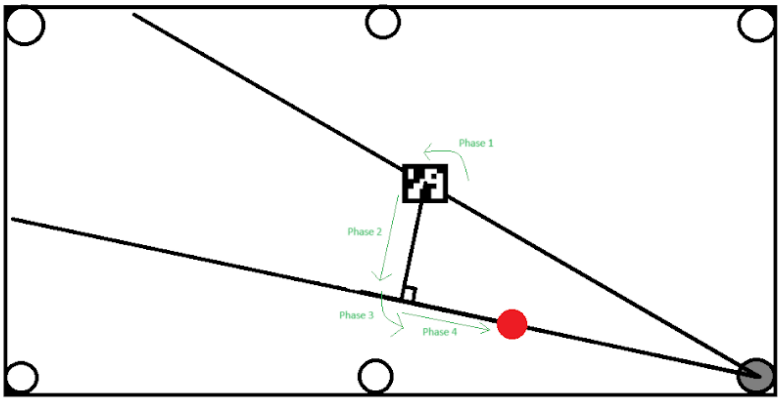


Ce diagramme représente le lancement de l'application. Tout commence par le lancement du serveur. Le client web (qui filme les objets placé sur le billard) et l'ESP (le robot) se connectent automatiquement au serveur. Le client web envoie en continue les coordonnées des objets. Le serveur effectue les calculs pour connaître le degré et la distance nécessaire pour toucher une boule. Il envoie ce message à l'ESP en lui précisant la date d'envoi. L'ESP peut alors vérifier la date. Si le message a été reçu sans trop de délais, il l'effectue sinon il l'ignore.

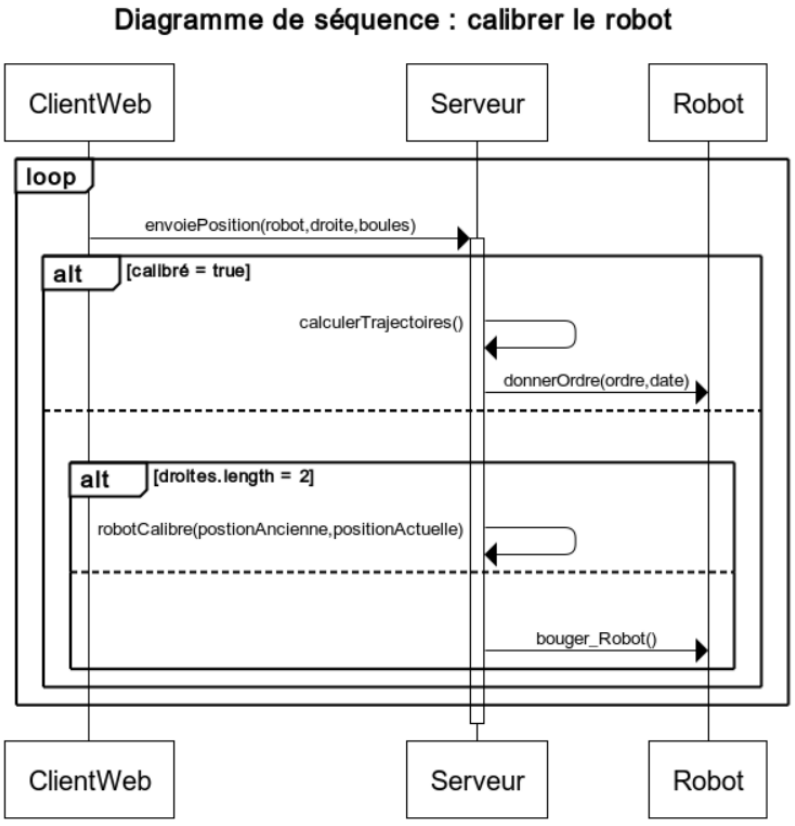
pendant le projet :



Il y a eu quelques changements au cours du projet. Tout d'abord notre robot n'est pas encore autonome et l'utilisateur doit cliquer sur la boule qu'il souhaite mettre dans le trou.
Ensuite, la manière dont sont effectués les mouvements est aussi différente. En effet, le mouvement est en 4 phases, comme vous pouvez le voir dans le schéma ci-dessus. Dans un premier temps, le robot doit être rotate pour être perpendiculaire au vecteur boule-trou. Ensuite avancer jusqu'à arriver au point intermédiaire, puis re rotate et enfin foncer sur la boule. Entre chacune des étapes, le serveur doit vérifier que le robot est bien positionné.



Pendant l'étude préalable :



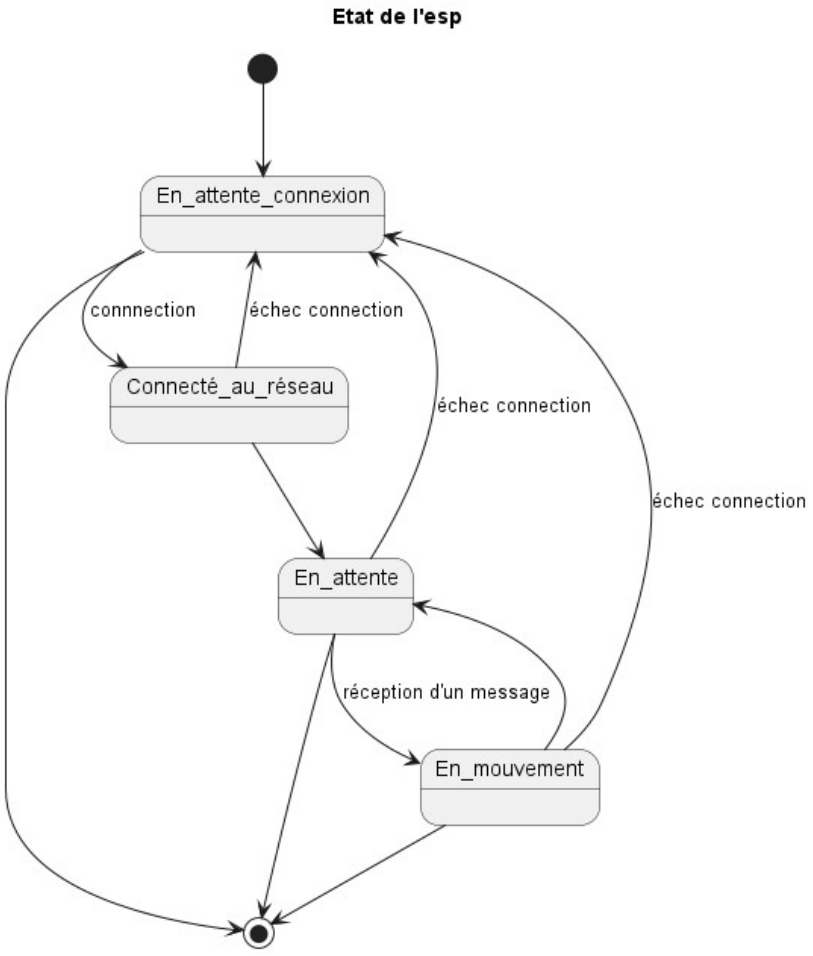
Ce diagramme fait pour l'analyse précédente n'est plus d'actualité car il a été observé que le calibrage du robot n'est pas fiable dans notre contexte et trop complexe pour l'instant. De nombreux paramètres doivent être pris en compte et certains d'entre eux évoluent au cours du temps

Pendant le projet :

Nous avons réalisé pendant le projet qu'il était difficile de calibrer le robot. En effet, la différence de vitesse entre les deux moteurs varie beaucoup et est causée par beaucoup de facteurs différents comme la puissance donnée ou bien le niveau de batterie du robot.

2.4.4. Diagramme d'état du microcontrôleur du robot (esp)

Pendant l'étude préalable :

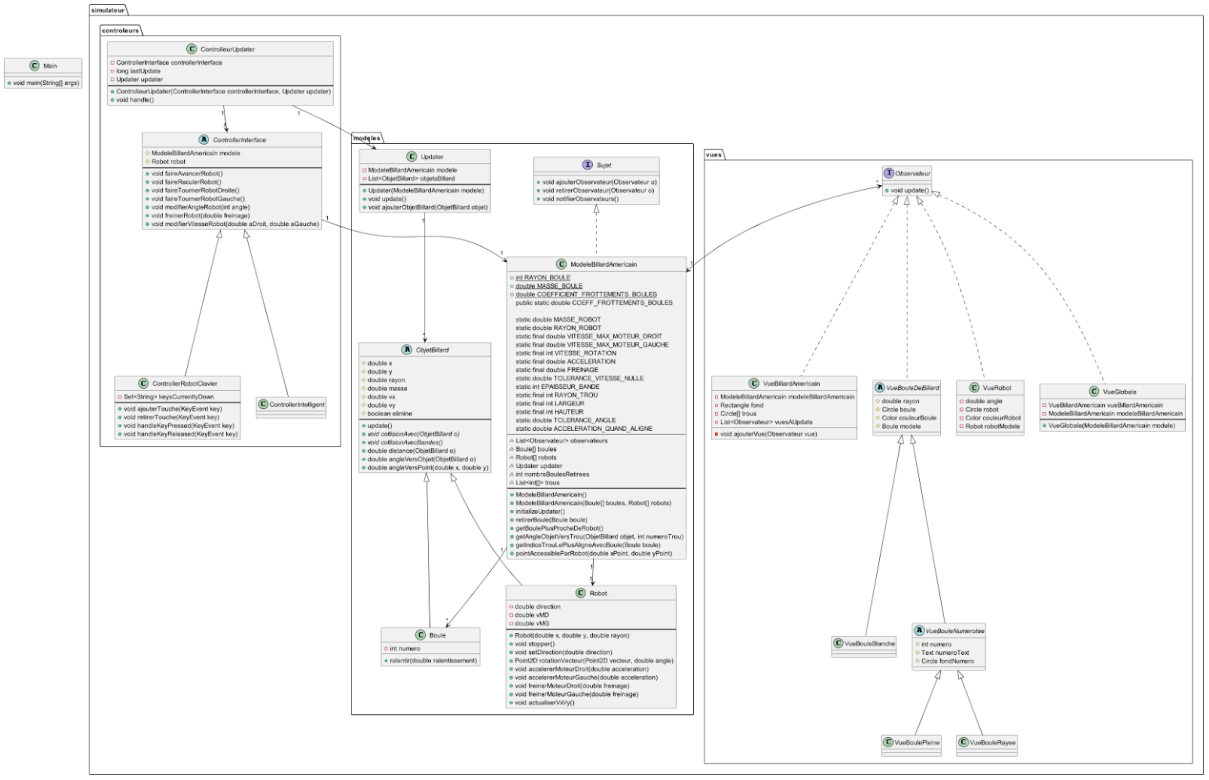


Ce diagramme représente l'état de l'ESP au cours du jeu. Dans un premier temps, il est allumé et attend que le serveur soit lancé pour pouvoir se connecter. Après être connecté, il est en attente de message de la part du serveur. A chaque fois qu'il y a un problème de connexion avec le serveur, il repasse à l'état d'attente de connexion. L'ESP peut être éteint à tout moment (par manque de batterie ou bien on l'éteint.)

Pendant le projet :

Ce diagramme reste toujours valide. Il représente bien l'état de l'arduino au cours du projet.

2.4.5. Diagramme de classe simulateur



Le projet de simulateur de billard suit une architecture modèle-vue-contrôleur (MVC), où le modèle (ModeleBillardAmericain) gère la physique du simulateur et transmet les ordres du Controlleur au Robot. Les vues représentent graphiquement les éléments du simulateur. Les contrôleurs gèrent les interactions utilisateur. L'ensemble du système permet à l'utilisateur d'interagir avec le simulateur de billard de manière interactive, en contrôlant le robot manuellement ou en laissant le ControllerAutonome effectuer des actions automatisées.
Le ControllerAutonome a pour but de contrôler le robot de sorte à ce qu'il pousse la balle la plus plus de lui vers la cible la plus proche de lui, et d'ensuite passer la la suivante.

2.5. Stratégies

Stratégie pour la gestion de la latence :

Pour gérer la latence, au début du programme, l'arduino reçoit l'heure du jour en millisecondes. Avec cette heure, l'arduino peut initialiser sa date initiale et en parallèle lancer un chrono. A chaque fois qu'il reçoit un message, l'arduino va calculer son heure interne en utilisant sa date initiale, le premier chrono et celui au moment de la réception du message. La latence max a été fixée à 300ms. En fonction de la latence du message, le temps accordé au mouvement diminue.

Stratégie du robot pour gagner au billard

Pour le développement du simulateur, nous trouvons pertinent de réfléchir sur la façon dont le robot va résoudre le problème. C'est-à-dire quelle stratégie va-t-on adopter pour que le robot puisse terminer le jeu.
Pour effectuer cette mission, nous devons prendre en compte les actions possibles du robot. Comme vu précédemment dans le diagramme du use case du robot, il peut uniquement avancer ou reculer dans l'espace de jeu. De plus, côté technique, on peut uniquement effectuer une tension au moteur pour avancer et reculer.
Donc pour l'algorithme et faciliter la programmation, nous avons décidé de prévoir des méthodes pour le déplacement prenant en compte la vitesse et l'angle.

Voici l'algorithme actuel (simulateur):

Algorithme permettant au robot de vider le billard de la trajectoire :

```
si abs(angleRobotBoule - angleBouleTrou) < TOLERANCE_ANGLE
        faireAvancerRobot(VITESSE_AVANCEMENT)
sinon
        Cible = boulePlusProche[ + DECALAGE_BOULE_CIBLE*Math.cos(angleBouleTrou);
        si cibleInnaccessibleParRobot
                Cible = Boule
        si abs(angleRobotCible - angleRobotBoule) < TOLERANCE_ANGLE
                faireAvancerRobot(VITESSE_AVANCEMENT)
        sinon
                si angleRobotCible - directionRobot > 0 ET angleRobotCible - directionRobot
< 180° OU angleRobotCible - directionRobot < -180°
                        modifierAngleRobot(VITESSE_ROTATION)
                sinon
                        modifierAngleRobot(-VITESSE_ROTATION)
```

Gestion des collisions entre le robot et le robot dans le simulateur :
Paramètres de l'équation :
vx : la vitesse de l'objet en fonction de x
vy : la vitesse de l'objet en fonction de y
m : Masse de l'objet
v : Vitesse initial de l'objet
θ : Angle du vecteur vitesse initial de l'objet
ϕ : angle de la ligne de collision
v′ : Vecteur vitesse final de l'objet après la collision
v'x : Vitesse final de l'objet en fonction de x après la collision
v'y : Vitesse final de l'objet en fonction de y après la collision

on a donc avec [variable]1 la variable indiqué de l'objet 1 et idem pour objet 2:
v = racine(vx * vx + vy * vy)
θ = arcTan²(vx, vy)
ϕ = arcTan²(y2-y, x2 - x)

$vx = v * \cos(\theta - \phi)$
$vy = v * \sin(\theta - \phi)$

$v'1 = ((m1 - m2) * v1x + 2 * m2 * v2x) / (m1 + m2);$
$v'2 = ((m2 - m1) * v2x + 2 * m1 * v1x) / (m1 + m2);$

$v'x = \cos(\phi) * v' + \cos(\phi + \pi/2) * vy$

En combinant ces étapes, l'équation permet de calculer avec précision les nouvelles vitesses des objets après la collision, en prenant en compte leurs masses, leurs vitesses initiales et l'angle de la collision.

Gestion de la reconnaissance des boules et du robot :

Comme vu précédemment, on utilise JsAruco et OpenCv pour détecter les boules et le robot mais dans la pratique la reconnaissance fonctionne mais n'est pas stable. C'est-à-dire, il est possible que sur 10 images capturées par la caméra, il y a des images où on ne détecte pas certains éléments de l'aire de jeu, ou pire on détecte des balles fantômes. En effet, dans le cas de boules fantôme, le robot peut frapper dans le vide. Pour résoudre ce problème de reconnaissance, nous envisageons plusieurs solutions:

- La première est d'utiliser le tracking, sur plusieurs images, on peut déduire la position de la boule suivante grâce aux positions obtenus précédemment. Dans les faits, on calcule la trajectoire des boules et on peut déduire sa position si OpenCv n'a pas pu la détecter.
- Deuxième solution, c'est l'utilisation de la librairie de réseaux de neurones convolutifs tel que tensforflow.js pour reconnaître les boules.

2.6. Planning avant les itérations

2.6.1. Itération 1

- Création du serveur
- Création du Client JS
- Détection des boules et l'aruco
- Piloter le robot à l'aide de l'ESP(microcontrôleur avec carte wifi)
- Développement du simulateur

2.6.2. Itération 2

- Algorithme de calcul de mouvement
- Calibrage du robot
- Différencier les boules (solutions envisagées : essayer de faire du tracking ou/et l'utilisation de réseau de neurone via tensorflow.js)

2.6.3. Itération 3

- Gestion de la latence
- Création de l’interface utilisateur au niveau du client
- Robot autonome
- Ajout de la gestion de la friction de la table de billard dans le simulateur
- Automatisation du démarrage du système

2.6.4. Itération 4

- Prévisualisation des trajectoires (dans l'interface du client js)
- Test & optimisations des algorithmes
- Déploiement du serveur sur un Raspberry pi
- Créer d'autres modes de jeu.

3. Réalisation

3.1. Tests de validations :

Les tests de validations ont été réalisés en faisant fonctionner le robot et nos algorithmes des déplacements de celui-ci sur un vrai billard et à l’aide d’une caméra fixée sur une lampe en forme de bras articulé.

3.2. Difficultés rencontrées :

Arduino :

Un des points qui nous a fait perdre beaucoup de temps est le robot. En effet, au cours des différents tests, nous avons été stoppés par des problèmes matériels. Il arrivait souvent que des câbles se détachent ou bien que les vis ne tiennent pas. Il y a aussi des problèmes de batterie.

Client JS et Serveur :

La première difficulté que l’on a rencontré au cours du projet a été la communication entre le Client JS et le Serveur. Utiliser des socket en JS a été une première pour nous. De plus, il nous est difficile, du côté du Client JS, de recevoir des informations du serveur et de les utiliser.

Ensuite, dû à l'utilisation de librairies extérieures comme openCV ou aruco, nous avons eu à comprendre leurs fonctionnements et à nous y adapter en conséquence. Par exemple, openCV à un système d'initialisation qui, lorsqu'il est mal pris en compte, bloque son utilisation dans les scripts.

De même, nous avons eu des problèmes particuliers et bien spécifiques comme un problème de chargement de chrome qui était beaucoup plus lent que le navigateur firefox pour ainsi dire.

Enfin, il a été compliqué d'implémenter parfaitement nos algorithmes. En effet, nous nous sommes confrontés à des spécificités matérielles au niveau de notre code. Nous avons dû prendre en compte la plupart des paramètres, qui dans une certaine mesure n'étaient pas fixes, du robot. Nous devions nous adapter au fur et à mesure des caprices matériels, un moteur pouvait changer de puissance en fonction de différents facteurs par exemple.

Simulateur :

Bugs de collision dans le simulateur :
Malgré le travail consacré à essayer de les résoudre, on peut toujours observer des bugs de collisions quand les boules se font pincer entre le Robot et les bandes du billard, cela résulte la plupart du temps en une boule qui part à une vitesse très élevée sur une trajectoire presque perpendiculaire au pincement.

4. Planning du projet

4.1. Itération 1 :

- Création serveur
- Piloter manuellement le robot via les touches haut bas gauche droite
- Détection de l'aruco (position et angle)
- Détection des boules
- Gestion de la latence
- Fusionner détection de l'aruco et détection des boules
- Développement du simulateur Partie 1 : Mise en place du squelette de l'application
- Connection Client JS -> Serveur

4.2. Itération 2 :

- Afficher les coordonnées des boules
- Ralentir progressivement le robot pendant le mouvement
- Réaliser un scénario simple (rotation et lancement du robot vers sa cible)
- Calibrage du robot
- Développement du simulateur Partie 2 : Faire bouger le robot avec les flèches directionnelles

4.3. Itération 3 :

- Développement du simulateur Partie 3 : Collision avec les boules
- Algorithme de calcul de mouvement : rotate puis avancer vers une boule
- Algorithme pour choisir le meilleur trou
- Configuration de l'espace de jeu (zone du billard et trous)
- Ajout de la gestion de la friction de la table de billard dans le simulateur

4.4. Itération 4 :

- Résolution bug chargement des modules sur chrome
- Modifier code arduino pour faire deux type de mouvement : constant et ralenti
- Algorithme de calcul de mouvement : positionner le robot pour frapper la boule la plus proche vers le trou le plus aligné

5. Répartition du travail

Nous nous sommes réparti les différentes parties du projet de la manière suivante :

Sila s'est occupé de la partie arduino. Comme par exemple la connexion au serveur, la gestion de la latence ou bien encore la gestion des mouvements.

Les principaux algorithmes de déplacement du robot ont été créés par Guillaume et Sila car il était important de pouvoir tester et implémenter les algorithmes en réel avec le robot. Il y a eu aussi un algorithme codé côté simulateur mais qui n'a pas encore pu être implémenter.

Victor et Guillaume se sont occupés de l'analyse des images reçues par la caméra avec l'aide de la librairie openCV. Victor s'est occupé de la détection de l'aruco, la prévisualisation des trajectoires, le fait de pouvoir sélectionner une boule et de placer les trous sur l'espace de jeu. Guillaume quant à lui s'est chargé de la détection des boules et a placé le rectangle représentant l'espace de jeu.

Mathieu a développé le simulateur en parallèle. Il a notamment créé l'espace de jeu, sa physique, son interface en JavaFX ainsi qu'un algorithme permettant au robot de pousser toutes les boules du billard dans ses trous de manière autonome.

6. Les éléments originaux

Sila Capar :

Je me suis majoritairement occupé du côté arduino donc je dirai que la partie dont je suis le plus fière est la gestion de la latence. Travailler sur l'arduino m'a permis de m'initier un peu au langage C++.

Mathieu Dessaulx :

Je suis surtout fier de l'algorithme qui permet de pousser toutes les boules dans les trous et de la propreté du code (facilité à comprendre et à modifier par exemple).

Victor Lath :

J'ai essentiellement travaillé sur la partie client connecté à la caméra pour la visualisation du jeu. Cette partie du projet m'a permis de voir brièvement ce qui se fait en matière de vision par ordinateur. L'identification et suivi des boules ainsi que la détection des arucos, m'ont permis de comprendre sommairement le fonctionnement de ces algorithmes. Même si ces fonctionnalités ne sont pas finalisées dans l'itération 4. Je pense qu'elles le seront à l'itération 5.

Guillaume Retter:

J'ai beaucoup aimé ,malgré quelques désagréments, la transmission des données entre le client et le serveur ainsi que la détection des cercles.

7. Objectifs à atteindre pour la fin du projet

7.1. Objectifs à atteindre

- La reconnaissance des boules doit être mise en place pour éviter les cercles fantômes.
- Le robot doit pouvoir gérer plusieurs boules sur le terrain.
- Une interface client présentant le jeu doit être créée.

7.2. Planning

Itération 5 :

- Implémentation de l'algorithme de recherche du meilleur trou.
- Corriger et implémenter l'algorithme ppb (Plus proche boule).
- Gérer la transmission des informations permettant de dessiner le triangle représentant les calculs de l'algorithme de positionnement du robot.
- Reconnaissance des boules.
- Création de l'interface utilisateur au niveau du client.

8. Conclusion

Au cours de ces quelques semaines et ce jusqu'à la fin de l'itération 4, nous avons pris beaucoup de plaisir à nous pencher sérieusement sur ce projet. Nous nous sommes confrontés à d'innombrables obstacles, cependant nous avons avancé en conséquence et sans fléchir. Toutes les semaines étaient remplies de nouveaux défis et tout le monde s'est donné à fond pour les vaincre. Néanmoins, nous ressentons de la frustration. En effet, nous avons pris plus de temps que prévu pour plusieurs tâches importantes et ainsi nous n'avons pas pu développer tout ce que nous voulions. On peut voir cela facilement grâce à la différence entre le planning prévu à l'analyse et le planning des itérations.

La plupart du temps, les problèmes qui nous ont majoritairement bloqué ont été les différents problèmes matériels. Le robot a été assez compliqué à cerner. Ses moteurs par exemple n'ont pas la même puissance dépendamment de la charge de la batterie par exemple. De plus, nous avons dû le réparer par intermittence car des câbles s'étaient arrachés ou des vis s'étaient desserrées. Ceci, nous avait coûté un précieux temps pendant lequel il nous était impossible de tester les différents algorithmes liés au robot et le changement des différents paramètres permettant sa manipulation dans les algorithmes.

Le deuxième problème important a été le débogage de notre code. Effectivement, pour pouvoir déboguer correctement, nous devions analyser en détail le comportement, parfois aléatoire, du robot. Ensuite, une grande partie des bugs étaient des problèmes atypiques, comme un problème de chargement uniquement du navigateur Chrome, ou les différents problèmes liés indirectement ou directement au socket dont nous n'avons jamais eu jusqu'à présent l'habitude d'utiliser.

Cependant, nous avons quand même dépassé les objectifs fixés lors de notre étude préalable du sujet. Bien que ceux-ci pouvaient paraître sans ambition (Rappel : Le but principal : faire un robot qui puisse se déplacer et toucher une boule grâce à un système intermédiaire. Objectifs intermédiaires : faire fonctionner le robot , réussir à connecter le robot au serveur, développer le client js pour visualiser l'espace de jeu avec une caméra, initialiser le serveur, développer le simulateur, intégrer les calculs de déplacement du robot dans le serveur), nous avions eu beaucoup de mal à définir les limites du projet puisque celui-ci était expérimental et nous n'avions aucune expérience dans ce type de projet.

Finalement, le projet nous a été bénéfique car nous avons tous appris ou amélioré nos compétences en informatique et en gestion de projet. Notamment dans la programmation de simulateur, la programmation de carte arduino, d'utilisation de socket JS, d'utilisation des librairies openCV et js-Aruco et dans la conception d'algorithme se servant de différents paramètres réels et de calculs mathématiques appliqués à un cas concret.

Nous souhaitons remercier chaleureusement Mr. Guenego pour ses excellents conseils et son suivi du projet. Il nous a orienté tout le long du projet, ce qui nous a aidé à mieux comprendre, analyser, appréhender et développer le projet.