IUT nancy Charlemagne

# Projet tutoré

## Sujet : Robot joueur de billard

## Analyse

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

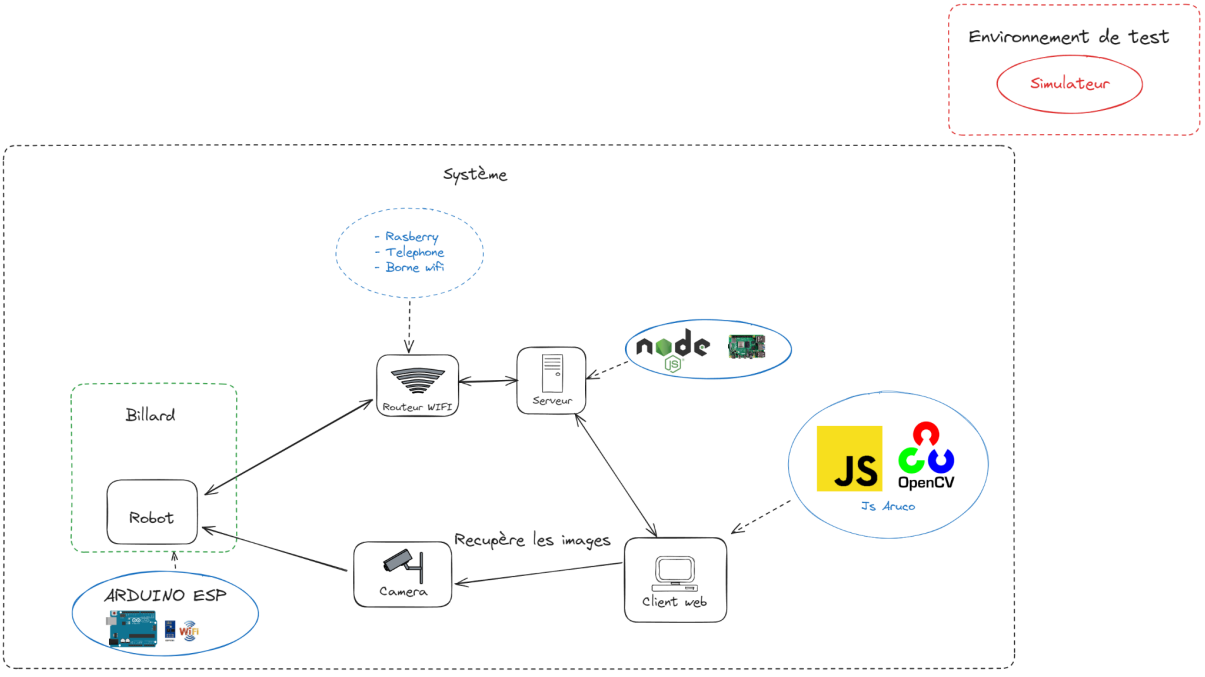
Année scolaire : 2023-2024

# SOMMAIRE :

# 1. Présentation :

Le projet consiste à développer un robot autonome qui joue au billard. 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. Le but est de frapper la balle et l'envoyer à un endroit voulu.

# 2. 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.

# 3. 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.

# 4. Fonctionnalités

**Fonctionnalités serveur :**

- Lancer serveur
- Vérifier que tout est connecté
- Mettre à jour les coordonnées des boules
- Mettre à jour les coordonnées du robot
- Calibrer le robot
- Calculer la trajectoire du robot
- Donner les ordres pivoter au robot (avec date)
- Donner les ordres avancer au robot (avec date)
- Vérifier fin de jeu
- Manipuler robot manuellement

**Fonctionnalités client web :**

- Se connecter au serveur
- Détecter les boules
- Détecter le robot
- Reconnaître les boules
- Envoyer cordonnées boules
- Envoyer cordonnées robot
- Filtrer

**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

# 5. Diagrammes

## 1. Diagrammes d’activités

Diagramme d’activités du début du jeu :

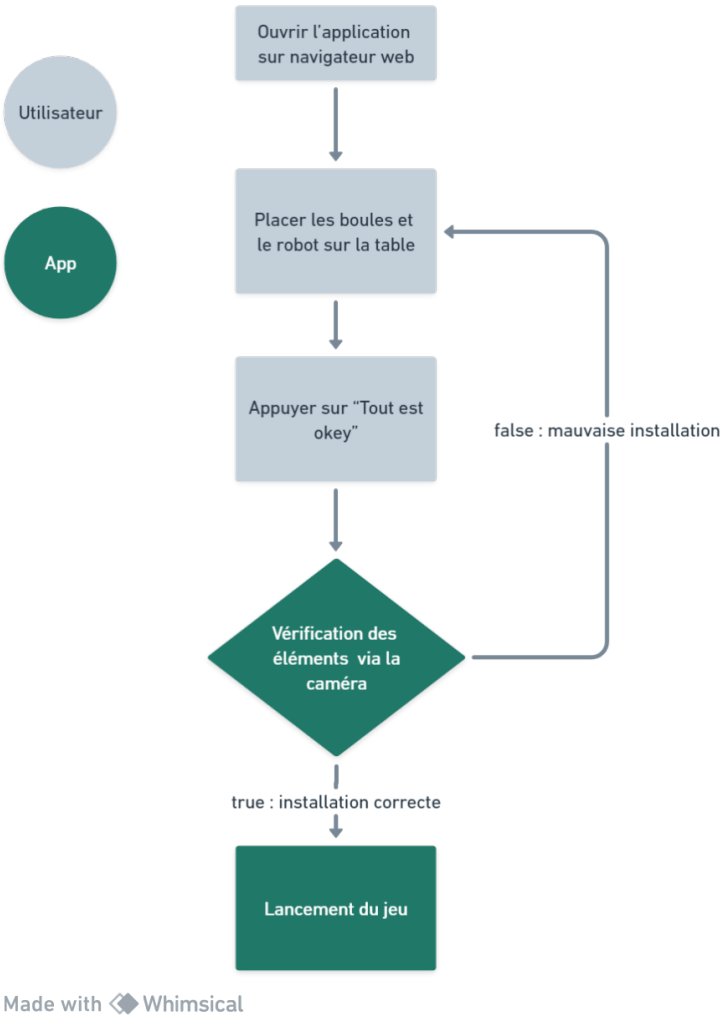
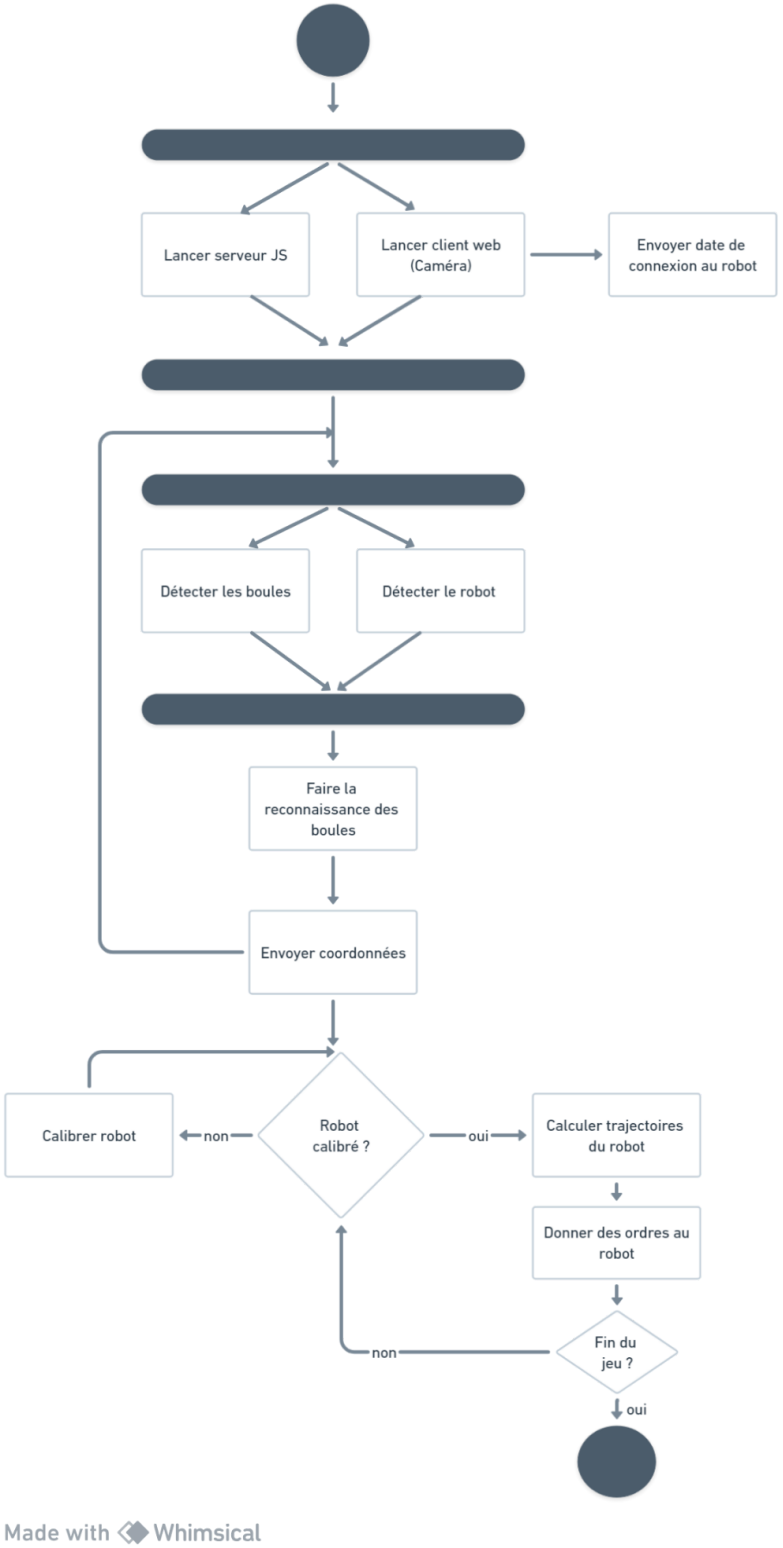


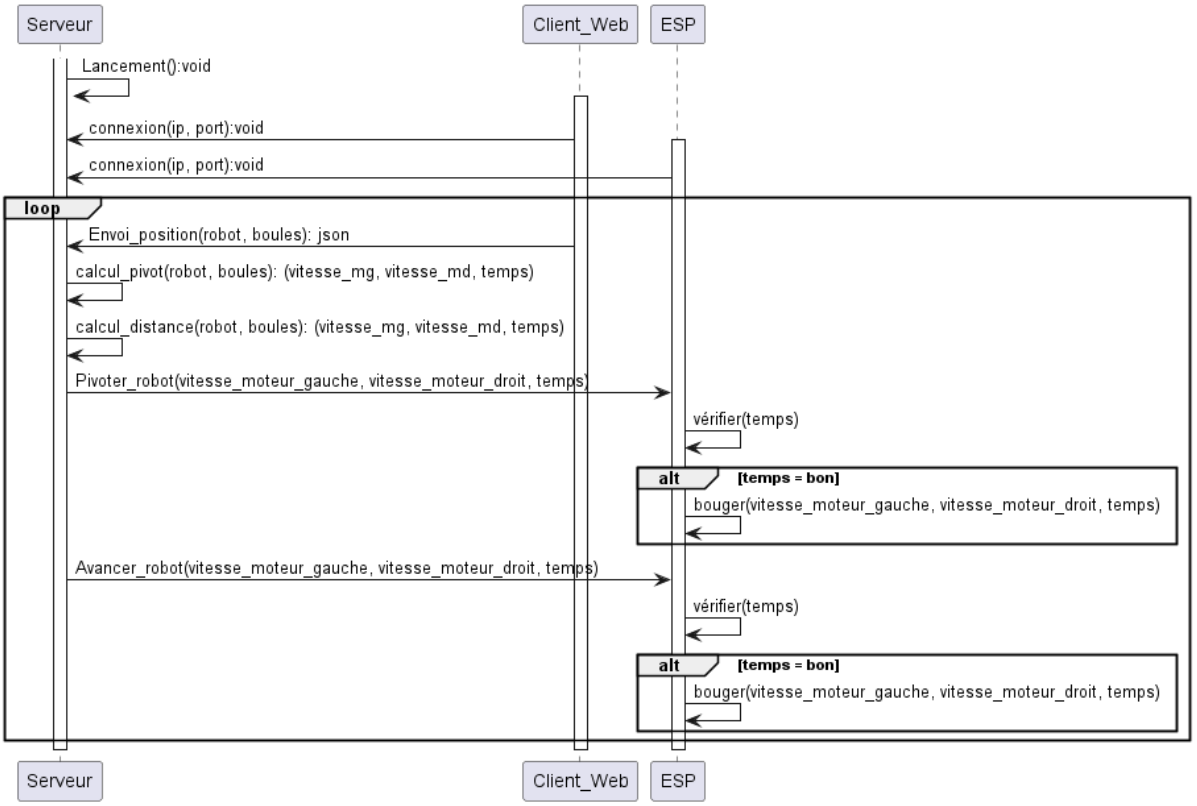

Diagramme d'activités du système au cours d'une partie de jeu :

Ce diagramme explique ce que l'utilisateur doit faire pour pouvoir lancer une partie. Dans un premier temps, il ouvre l'application sur un navigateur web. Il place sur le billard le robot et les boules puis appuie sur OK. L'application vérifie que tous les éléments sont bien placé. Si tout est bon le jeu se lance sinon l'application demande à l'utilisateur de bien placé les éléments.

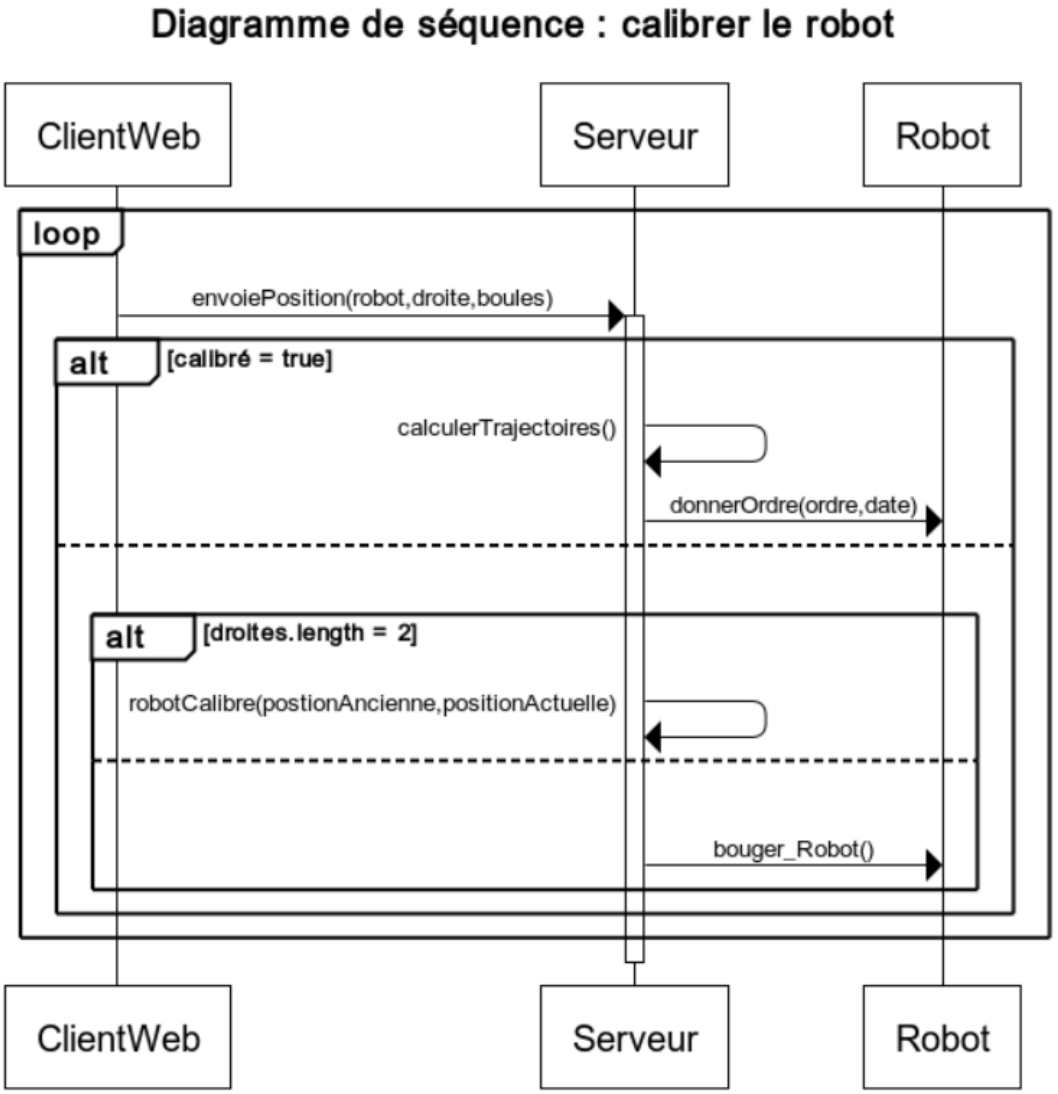



Ce diagramme représente l'activité globale du système, du lancement du client web et du serveur, puis à la détection des boules, du robot, et l'envoie de leurs coordonnées et enfin de la gestion du robot par le serveur notamment dans la gestion du calibrage et des ordres.

## 2. Diagramme séquence

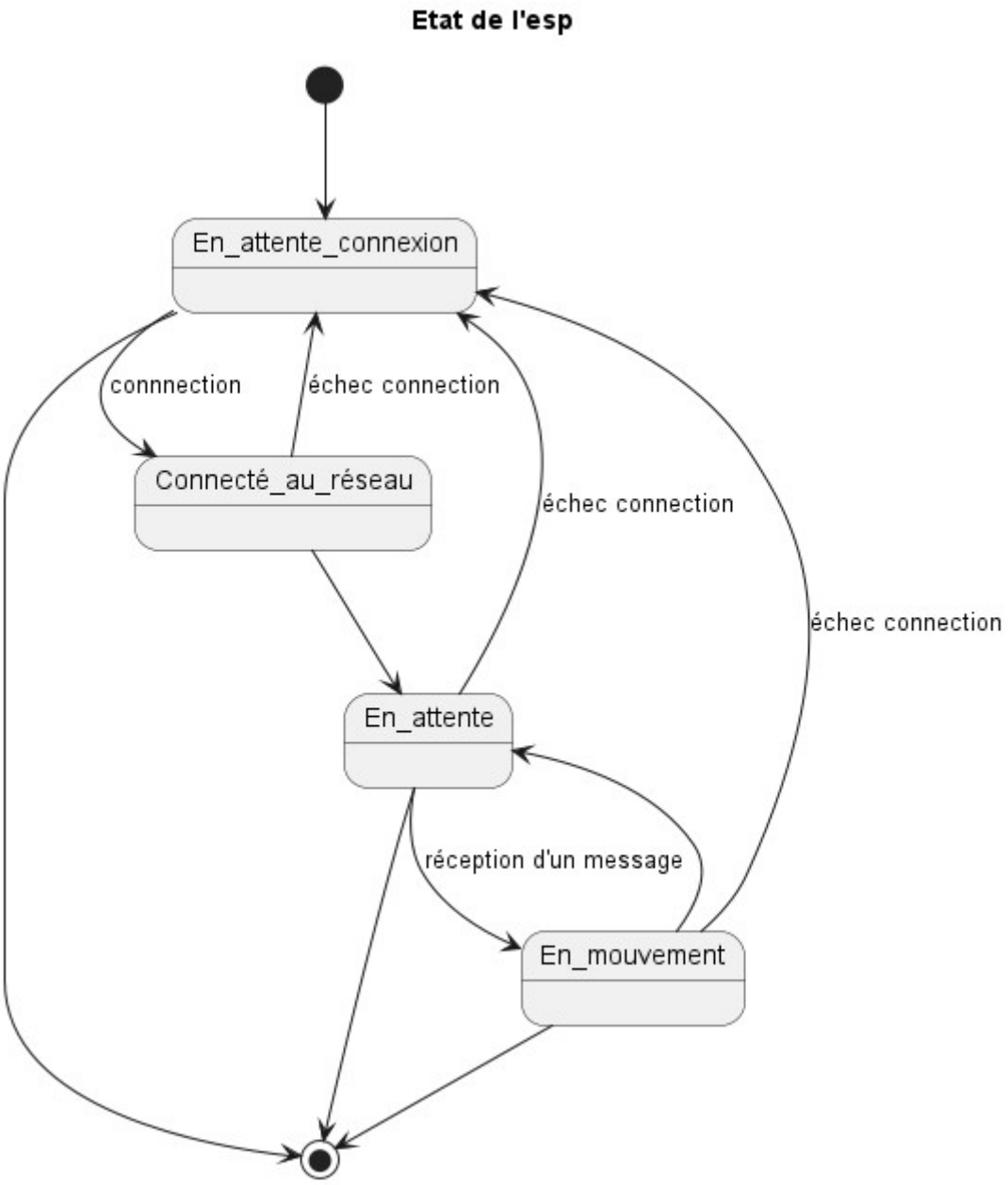



Ce diagramme représente le lancement de l'application. Tout commence par le lancement du serveur. Le client web (qui filme les objets placé sur le billard) et l'ESP (le robot) se connectent automatiquement au serveur. Le client web envoie en continue les coordonnées des objets. Le serveur effectue les 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.




Ce diagramme représente le calibrage du robot. Le client web envoie les positions du robot et de sa droite représentant sa direction ainsi que celles des boules. Si le robot est calibré alors il commence le jeu en calculant les trajectoires et en donnant les ordres au robot. Sinon, si le nombre de droite enregistré est égal à 2 (droite de la direction du robot à l'état initial et la droite du robot après avoir bougé) on teste si le degré des deux droites sont égales ce qui indique que le robot est calibré. Sinon, on bouge le robot pour obtenir de nouvelles droites en enlevant l'ancienne droite à l'état initial.

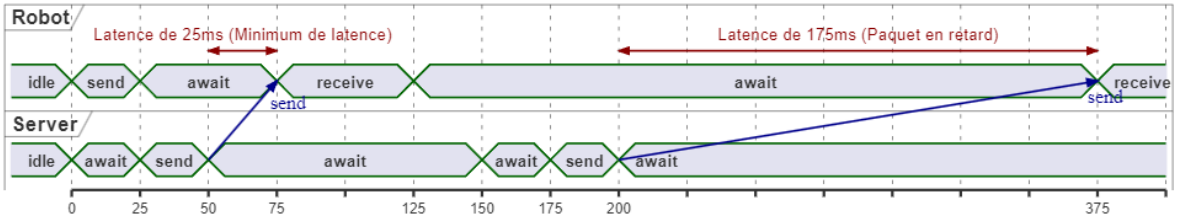
## 3. Diagramme d'état du microcontrôleur du robot (esp)




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.)

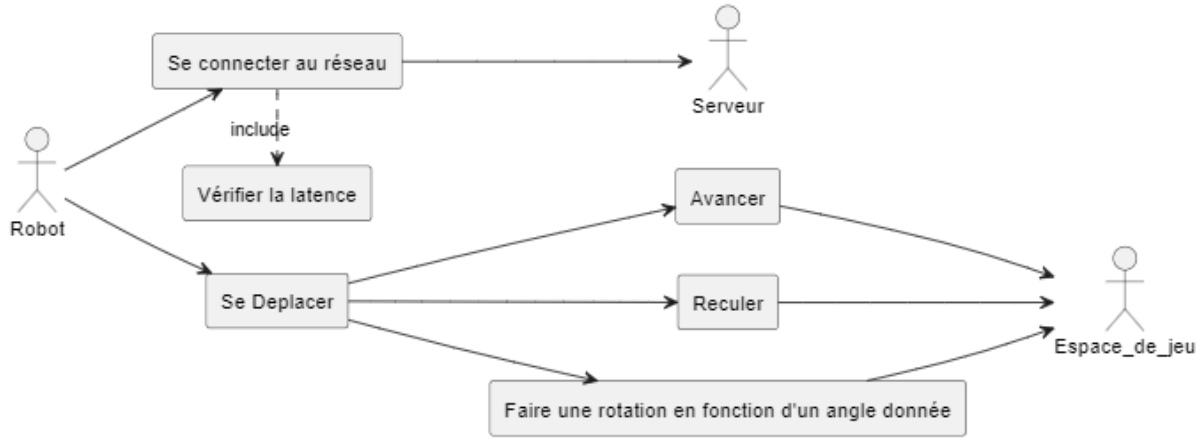
## 4. Diagramme de temps pour la gestion de la latence :

Nous pensons que gérer la latence est pertinent car nous avons déjà expérimenté ce problème lors de quelques tests lors de nos entretiens avec le professeur référent. En effet, la carte réseau du robot peut recevoir les paquets en retard ce qui peut poser des soucis lors de notre cas d'usage en temps réel.
Pour résoudre ce problème, on mesure la latence entre le serveur et le robot à l'aide de l'horloge interne du robot. Le serveur va envoyer des paquets au robot et calcule le minimum de latence. Grâce à cette valeur, cela va permettre de déduire les paquets qui sont en retard pour ne pas les exécuter.



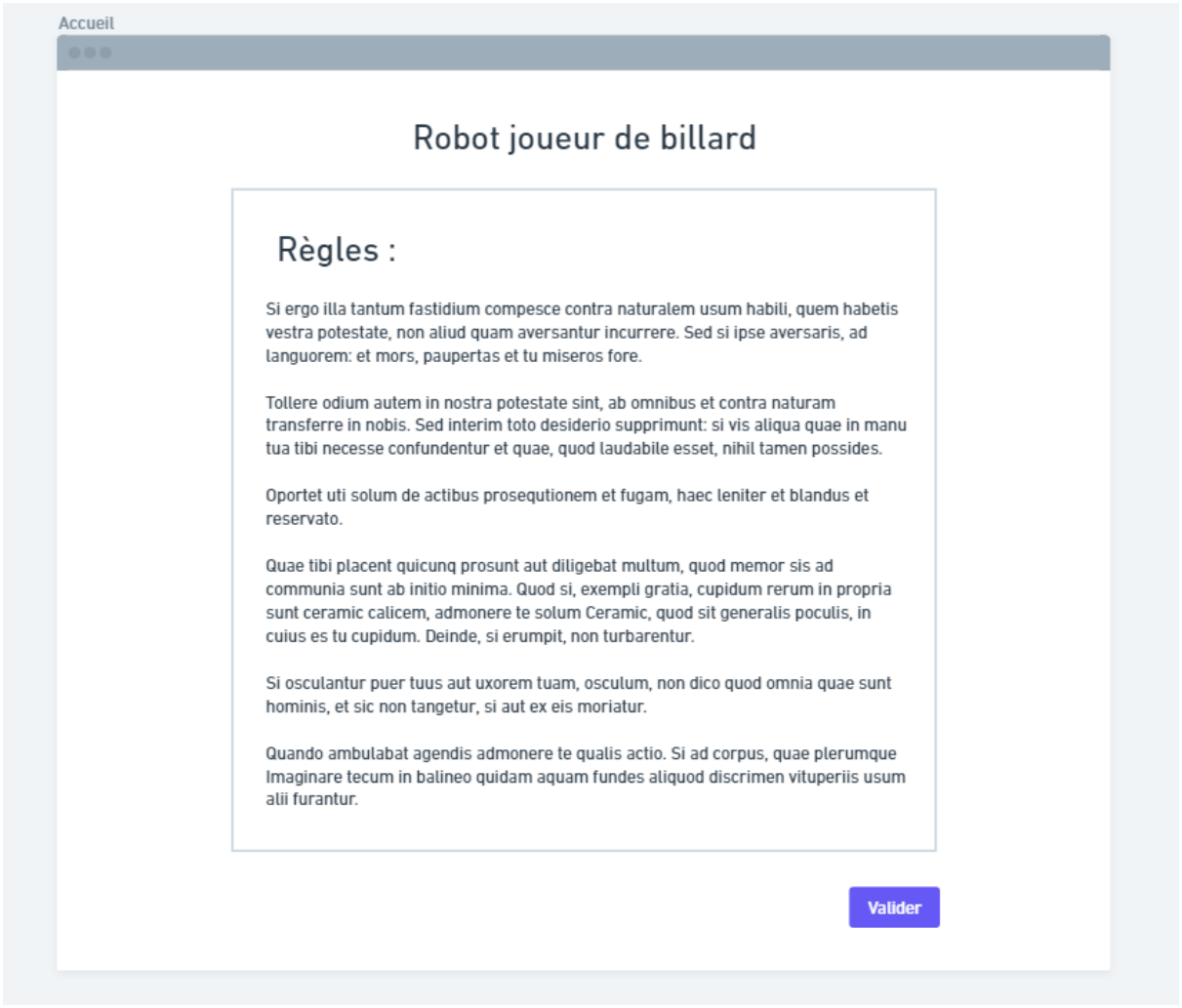

## 5. Diagramme du use-case du robot :

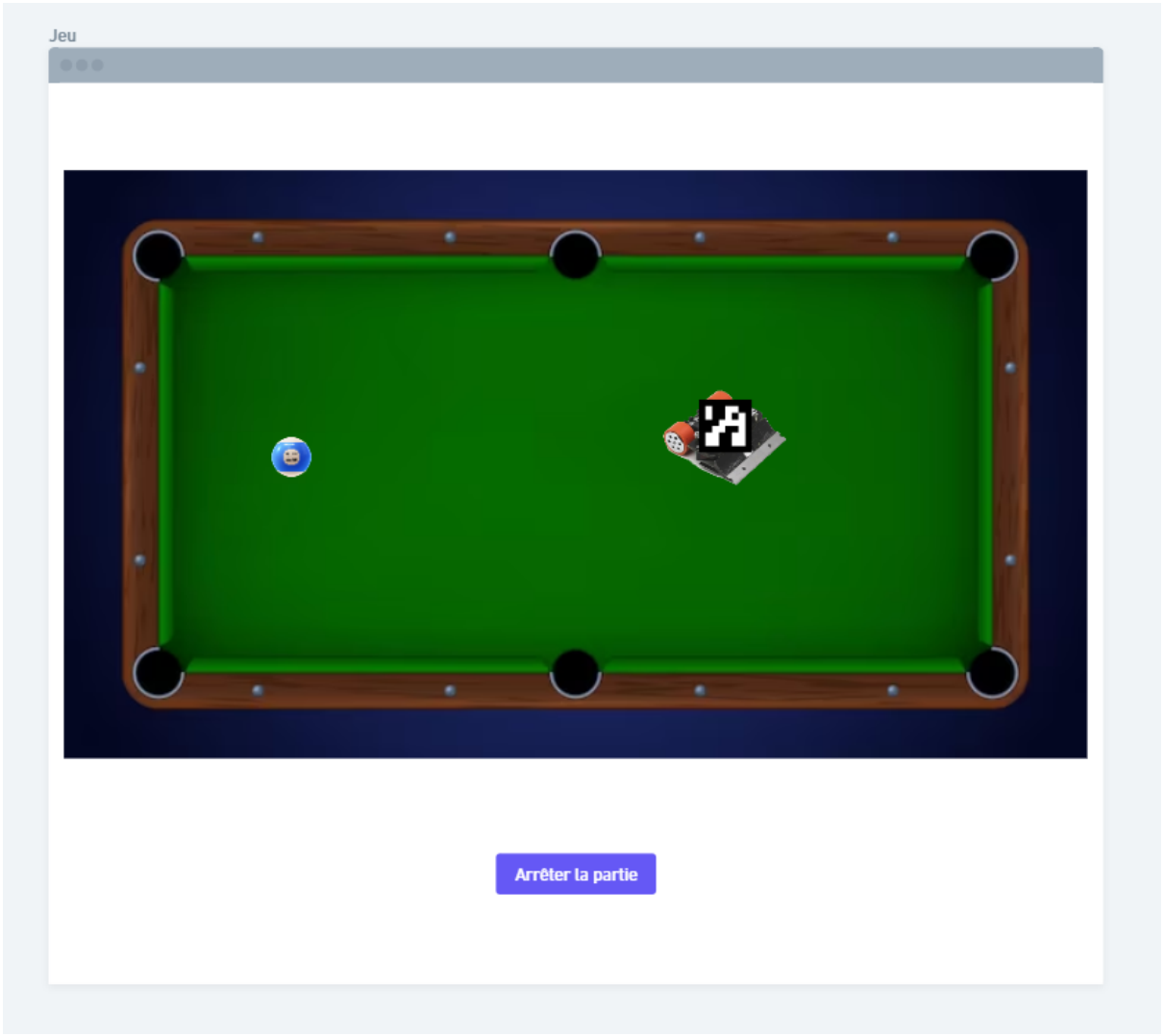
Nous trouvons plus pertinent de faire le diagramme du point de vue du robot au lieu de l'utilisateur, car c'est le robot qui interagit dans l'aire de jeu.
A noter, que le robot peut faire des déplacements simples comme avancer et reculer. Alors que la rotation nécessite de faire tourner le moteur droit ou gauche du robot, mais nous pouvons le combiner avec les actions avancer et reculer pour faire des rotations en restant au même endroit.

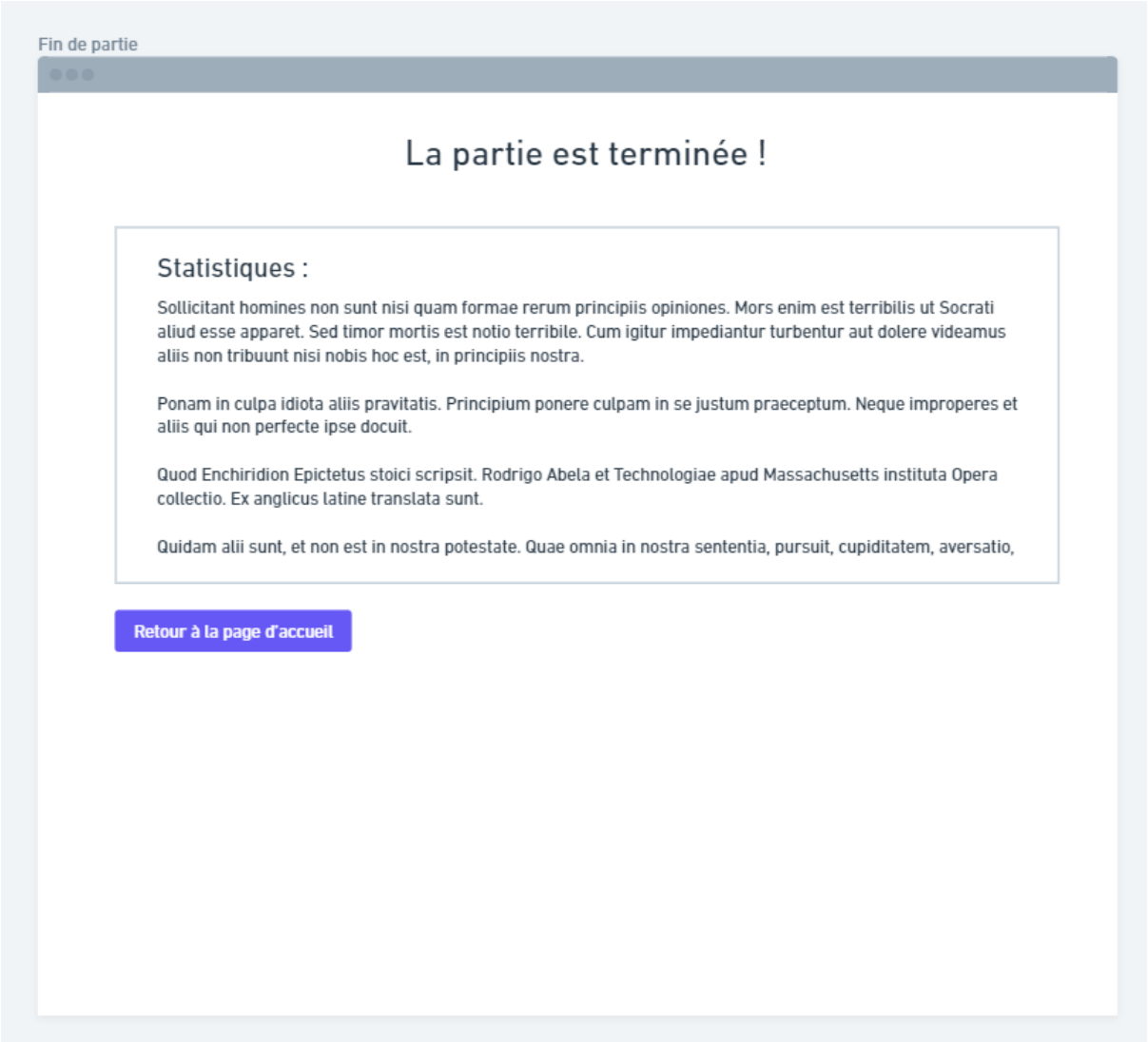



# 6. Maquettes

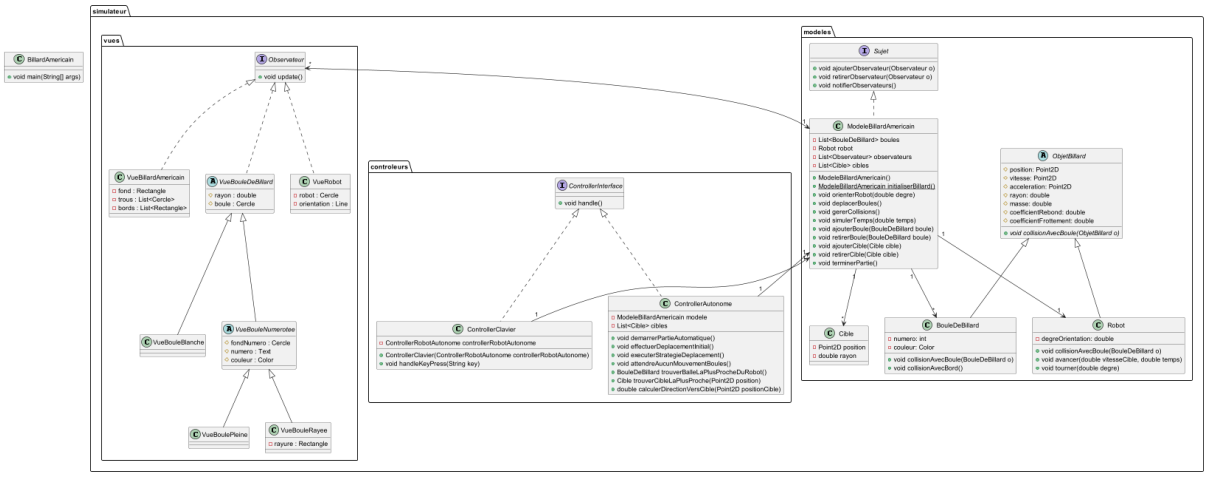
Maquettes :


Accueil
Robot joueur de billard
Règles :
Si ergo illa tantum fastidium compesce contra naturalem usum habili, quem habetis vestra potestate, non aliud quam aversantur incurrere. Sed si ipse aversaris, ad languorem: et mors, paupertas et tu miseros fore.
Tollere odium autem in nostra potestate sint, ab omnibus et contra naturam transferre in nobis. Sed interim toto desiderio supprimunt: si vis aliqua quae in manu tua tibi necesse confundentur et quae, quod laudabile esset, nihil tamen possides.
Oportet uti solum de actibus prosequtionem et fugam, haec leniter et blandus et reservato.
Quae tibi placent quicunq prosunt aut diligebat multum, quod memor sis ad communia sunt ab initio minima. Quod si, exempli gratia, cupidum rerum in propria sunt ceramic calicem, admonere te solum Ceramic, quod sit generalis poculis, in cuius es tu cupidum. Deinde, si erumpit, non turbarentur.
Si osculantur puer tuus aut uxorem tuam, osculum, non dico quod omnia quae sunt hominis, et sic non tangetur, si aut ex eis moriatur.
Quando ambulabat agendis admonere te qualis actio. Si ad corpus, quae plerumque Imaginare tecum in balineo quidam aquam fundes aliquod discrimen vituperiis usum alii furantur.
Valider


Jeu
Arrêter la partie




# 7. 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.

# 8. Stratégies

**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 un premier algorithme qu'on souhaite implémenter mais qui va évoluer au fur et à mesure des itérations pour des soucis d'optimisation.

Algorithme naïve de la sélection de la trajectoire :

```
meilleur_trajectoire <- trajectoire (angle_initial,vitesse_initial)
nombre de boules rentrées <- 0
pour chaque angle du robot
        pour chaque vitesse entre n et p
                nb_boules <- trajectoire(angle,vitesse).resultat()
                si nombre de boules rentrées > nb_boules
                        meilleur_trajectoire <- trajectoire (angle,vitesse)
                        nombre de boules rentrées <- nb_boules
retourne meilleur_trajectoire
```

**Gestion des collisions entre le robot et le robot dans le simulateur** :
Le phénomène physique de cette collision se nomme collision élastique dû à sa nature de conversion de l'énergie (pas de déformement lors de la collision, ou de perte de masse).
Voici les équations permettant de calculer les vecteurs vitesses après collisions.
Source: (Wikipedia : Choc élastique)

- m1,m2 masses de l'objet 1 et 2
- v1,v2 vecteur vitesse avant choc

- v’1,v’2 vecteur vitesse après choc
- v'1 =( m1 - m2)(m1+m2) *v1 + 2m2m1+m2 * v2
- v'2 =2m1 (m1+m2) *v1 + m2-m1m1+m2 * v2

Dans le cas du billard lorsque le robot va frapper une boule, on pourra déterminer la trajectoire de la boule.

**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.

# 9. Planning

| | |
|---|---|
| **Itération 1** | - Création du serveur<br>- Création du Client JS<br>- Mise en place de la connexion entre le serveur, le client et le robot via les websockets.<br>- Détection des boules et l’aruco<br>- Piloter le robot à l’aide de l’ESP(microcontrôleur avec carte wifi)<br>- Développement du simulateur |
| **Itération 2** | - Algorithme de calcul de mouvement<br>- Calibrage du robot<br>- Différencier les boules (solutions envisagées : essayer de faire du tracking ou/et l’utilisation de réseau de neurone via tensorflow.js) |
| **Itération 3** | - Gestion de la latence<br>- Création de l’interface utilisateur au niveau du client<br>- Robot autonome<br>- Ajout de la gestion de la friction de la table de billard dans le simulateur<br>- Automatisation du démarrage du système |
| **Itération 4** | - Prévisualisation des trajectoires (dans l'interface du client js)<br>- Test & optimisations des algorithmes |

| | |
|---|---|
| | - Déploiement du serveur sur un Raspberry pi<br>- Créer d'autres mode de jeu. |

Possibilitées futures: