Rapport de mi-parcours
Robot billard

Antoine FONTANEZ - Nathan PIERROT - Corentin FROGER

Tuteur : Pierre-André GUENEGO
Année : 2024/2025

Sommaire

Introduction

Le projet consiste en le développement d'un système de robot autonome pour jouer au billard. Les objectifs de ce projet sont de connecter le robot à un serveur que nous développerons afin qu'il puisse recevoir des ordres de celui-ci, mais également de développer une interface web pour visualiser l'état des robots et les commander en temps réel. En complément, un simulateur sera créé pour reproduire l'environnement réel du billard, facilitant ainsi les tests et la validation des fonctionnalités du système.

Après une analyse poussée du projet, comprenant la recherche des technologies à utiliser, l'analyse de l'existant, la préparation de l'architecture du projet et bien d'autres aspects, nous avons commencé la programmation.

En commençant par le minimum vital pour notre projet, à savoir la programmation des robots. Nous avons donc découvert le langage de programmation utilisé par l'arduino, le C++. Un langage bas niveau assez compliqué à utiliser. Ensuite, nous nous sommes occupés de créer une interface de contrôle et bien sûr un serveur qui permet la communication entre le robot et l'interface.

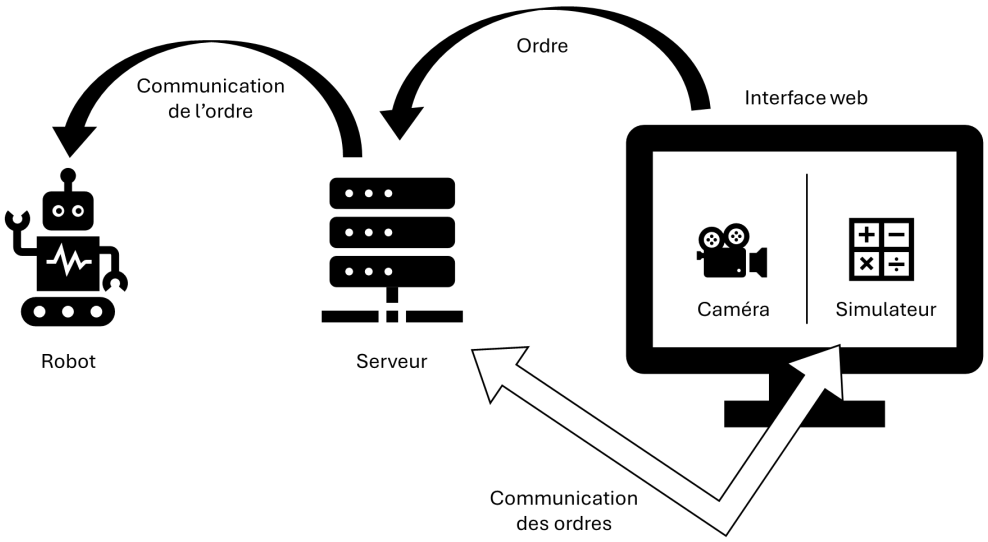
Une fois cette base réalisée, nous avons pu développer les premières fonctionnalités visuelles. Nous avons donc fait en sorte de pouvoir contrôler le robot manuellement depuis notre interface web. C'est le début d'un long parcours rempli de défis pour faire bouger notre robot selon nos désirs.

En parallèle, nous avons développé un simulateur, de plus en plus réaliste au fil des itérations, afin de réaliser des tests et pour nous permettre de continuer à travailler sur de nouvelles fonctionnalités même en l'absence de la table et du robot physique.

Analyse

Structure du projet

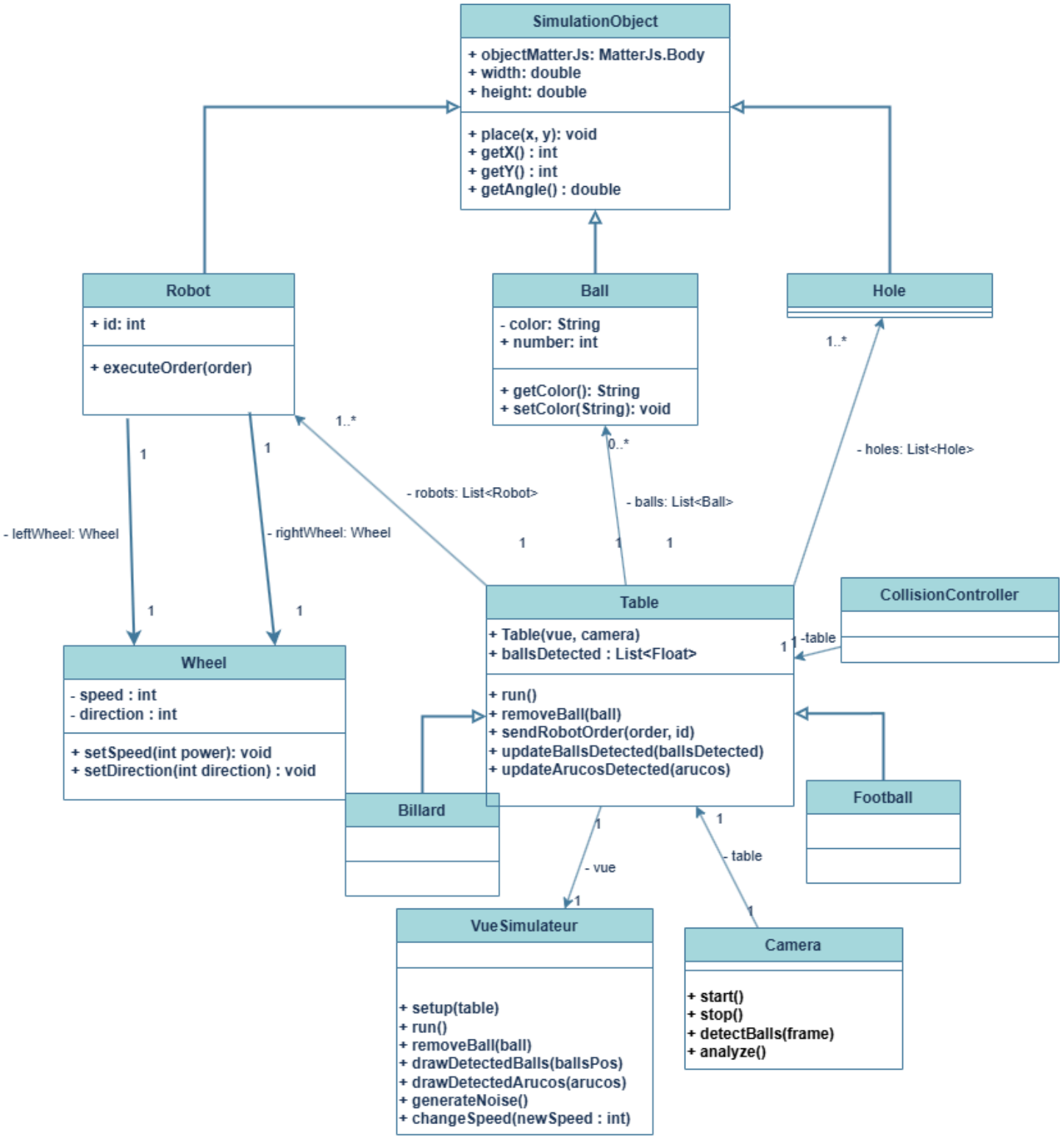
Notre projet se compose de plusieurs parties de programmation, nous avons une partie arduino pour le code interne du robot qui lui permet de recevoir des ordres et de les exécuter. Ensuite, nous avons le serveur Node.js qui permet la connexion et la communication des ordres entre le robot et l'interface web. Cette interface web est la plus grosse partie du projet et est composée de plusieurs sous parties. La première se compose d'une vue de la caméra qui permet de visualiser et de traiter le flux vidéo de la caméra branchée à l'ordinateur. Cette caméra filme l'entièreté de la table de billard et toutes ses boules. La seconde comprend la réalisation d'un simulateur réaliste programmé grâce à la librairie Matter.js qui nous permet d'obtenir une physique réaliste. Ce simulateur reçoit également des requêtes du serveur pour faire exécuter des ordres au robot. Voici ci-dessous un bref schéma récapitulatif de notre structure :



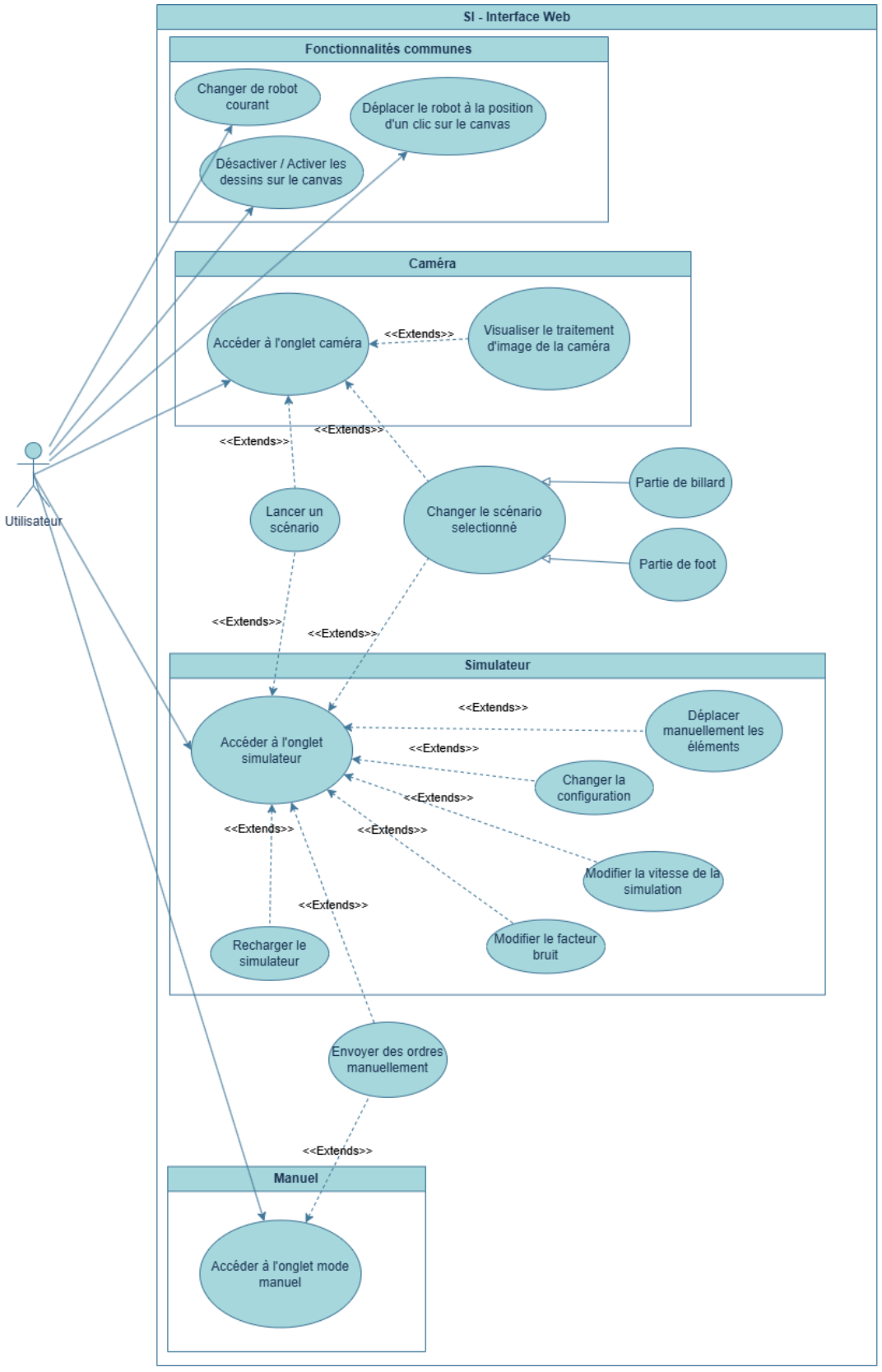
Diagrammes UML

Afin de rendre nos explications le plus claires possible, voici quelques diagrammes UML de notre structure :

- Diagramme de classe :



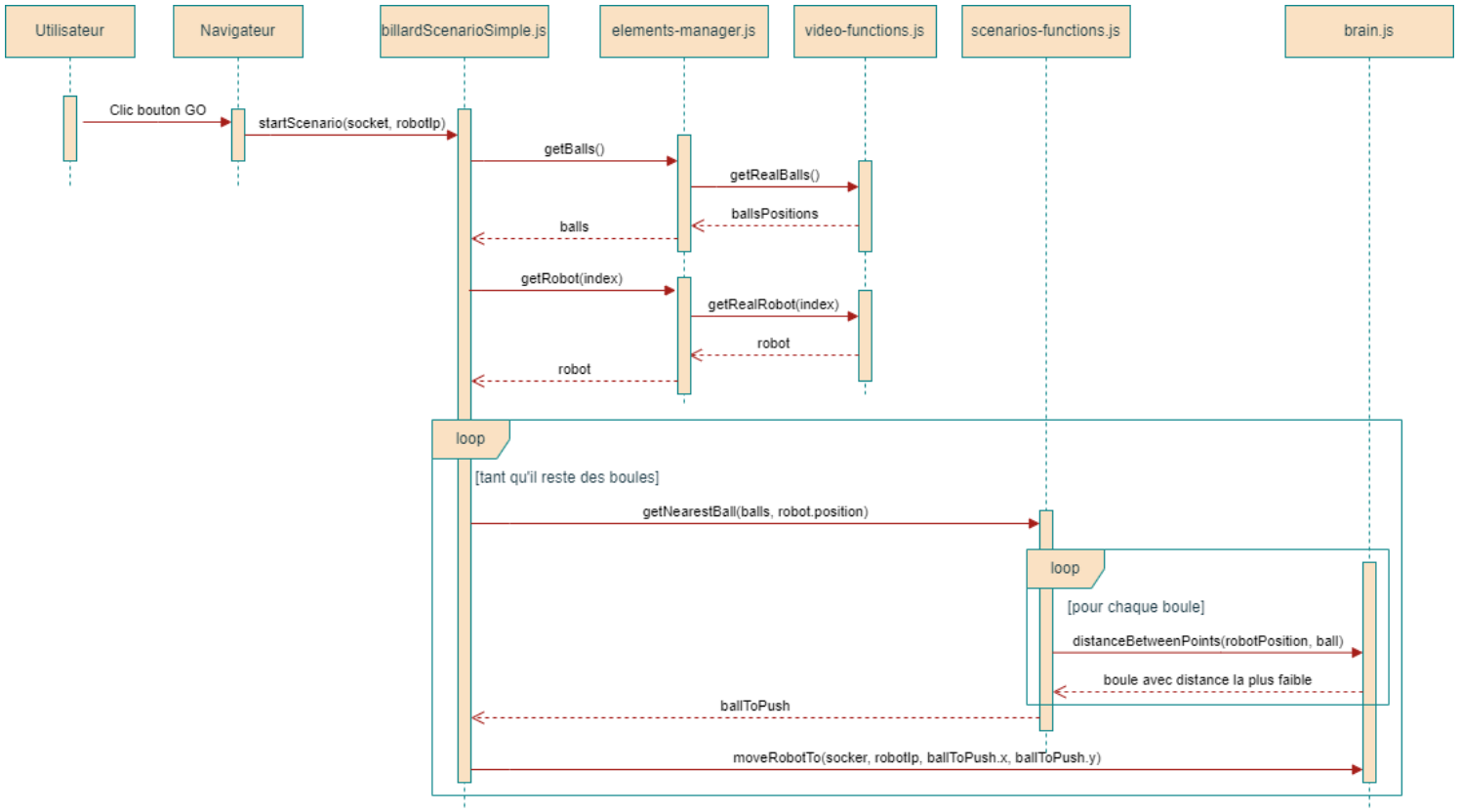
- Diagramme de CU :



Précision

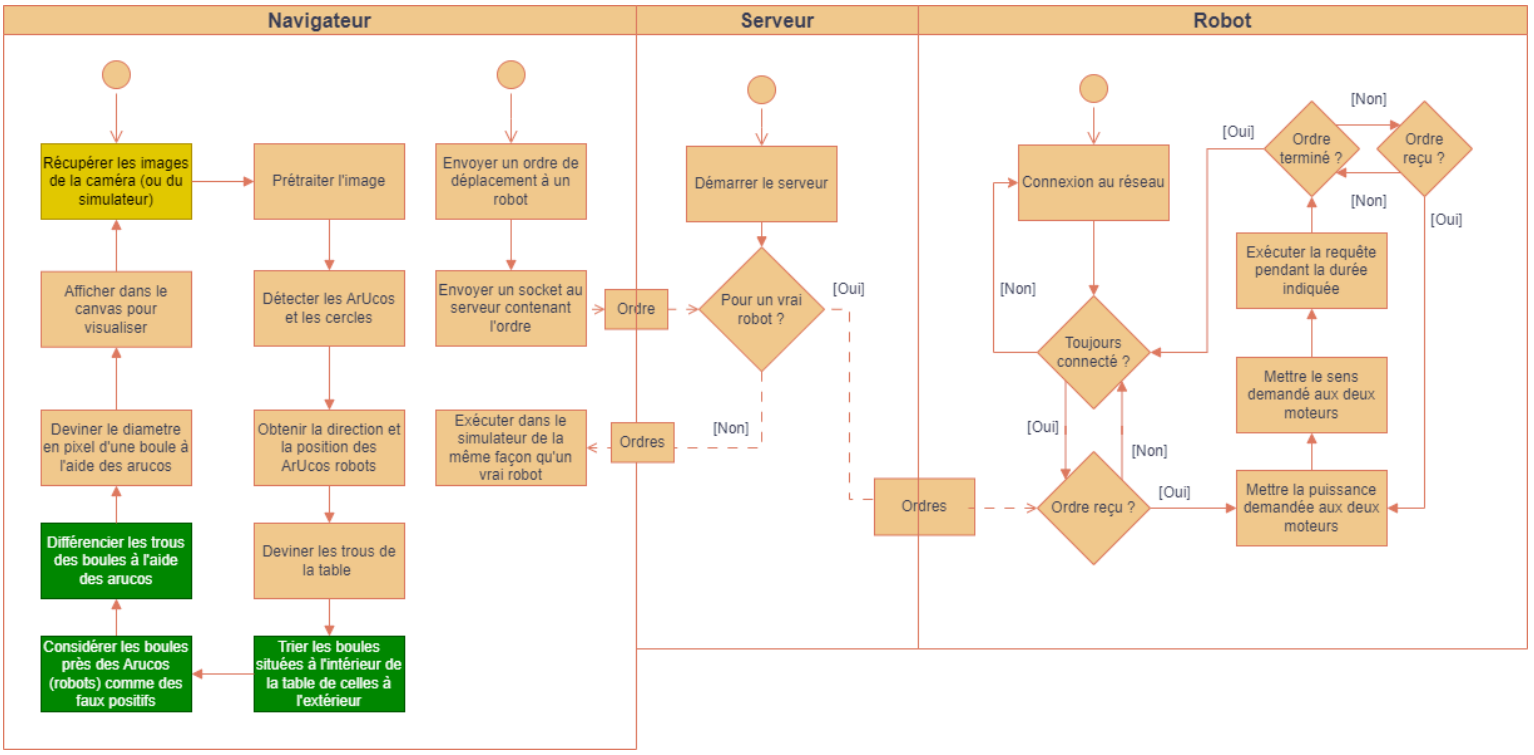
Les fonctionnalités communes sont accessibles depuis n'importe quel onglet

- Diagrammes de séquences :
  (Scénario billard simple)



Ce diagramme de séquence montre le déroulement d'une partie de billard simple (le robot se contente d'aller vers la boule la plus proche), lorsque l'utilisateur clique sur le bouton GO en ayant choisi ce scénario, le scénario se lance et cherche à obtenir la position des boules et du robot détectées par openCV, ensuite, tant qu'il reste des boules à mettre dans des trous, on cherche à savoir quelle boule est la plus proche du robot, puis on demande au robot de s'y déplacer.

- Diagramme d’activités :



Ce diagramme explique le fonctionnement général de l’application dans son état actuel. Une fois le serveur Node.js lancé, celui-ci vérifie en permanence si un ordre doit être envoyé à un robot, et s'il doit l’envoyer à un vrai ou à l’un de ceux du simulateur.

Dans le navigateur, côté client, on utilise openCV pour récupérer les images de la caméra, il faut ensuite traiter l’image pour mieux détecter les objets (boules, ArUcos, trous), on stocke ensuite leur position, et l’orientation des ArUcos robots. On peut également connaître la zone du billard à l’aide des ArUcos et donc pouvoir considérer les boules situées à l’extérieur de la table comme des faux positifs, nous utilisons également les ArUcos pour différencier les trous de table des boules.

Les robots quand à eux doivent d’abord se connecter au réseau (ils se reconnectent si besoin) et exécutent les ordres que le serveur leur envoi, après quoi ils ajustent la vitesse et le sens des moteurs concernés et exécutent la requête pendant la durée indiqué, sachant que s’ils reçoivent un nouvel ordre, ils l’éxécutent à la place de l’ancien s’il n’était pas terminé.

Evolution par rapport à l’étude préalable

Les diagrammes créés lors de l’étude préalable ont été amenés à changer plus ou moins fortement selon les cas, dans le cas du diagramme des cas d’utilisations, nous avons au final abandonné l'idée des fonctionnalités d’import et de sauvegarde de configurations pour le simulateur, car jugée inutile. Nous avons en revanche ajouté des fonctionnalitées auxquelles nous n’avions pas pensé à la base, comme pouvoir directement détecter des formes à l’aide d’openCV dans le simulateur, et de rajouter du “bruit”, toujours directement dans le simulateur pour pouvoir dégrader la détection d’openCV comme dans la réalité.

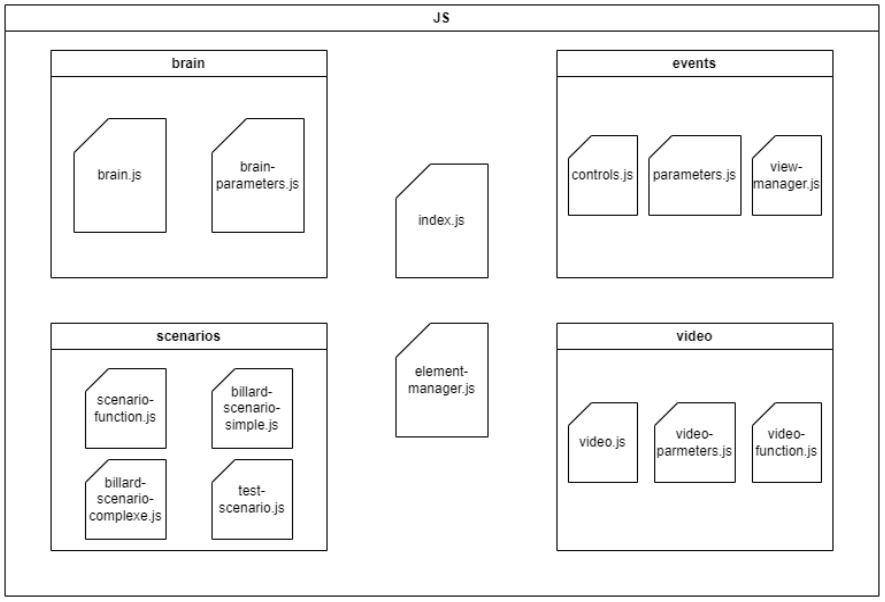
La façon dont nous avons conceptualisé les classes dans le simulateur a également été modifiée, nous avons rendu le MVC plus robuste, nous avions à la base pensé que nous devions faire une classe Serveur pour simuler le serveur dans le simulateur, mais nous avons simplement réutilisé le serveur préexistant.

Nous avons également ajouté une classe caméra gérant la détection openCV dans le simulateur, ce qui n’était pas prévu à la base, mais qui s’est avéré très utile pour pouvoir tester nos algorithmes et scénarios plus facilement.

Dans le cas du diagramme de séquence détaillant un scénario de billard, il s’est avéré assez proche de la façon dont il a été réalisé. En soit, on récupère les positions des boules et des robots détectés par openCV (réels ou ceux du simulateur), on cherche à savoir quelle est la boule la plus proche du robot concerné, puis on lui indique de s’y déplacer.

Réalisation

Architecture logicielle



Voici la structure de nos fichiers js, tout d'abord, nous retrouvons au centre le fichier *index.js* qui permet de faire le lien entre tous les fichiers js et le fichier *index.html*. Au même niveau, nous avons placé le fichier *element-manager.js* qui, lui, regroupe toutes les fonctions de récupération d'éléments communs aux différentes vues, par exemple la récupération des différentes boules de la table, que ce soit sur le billard réel ou sur le simulateur.

Ensuite, nous retrouvons en haut à gauche de l'image un dossier *brain* qui contient tous les fichiers utiles à "l'intelligence" du robot. Notamment le fichier *brain.js* qui contient entre autres les méthodes de déplacement et le fichier *brain-parameters.js* que nous utilisons pour regrouper toutes les constantes utilisées par le fichier *brain.js*. Cela nous permet de modifier plus facilement différents paramètres de déplacement sans avoir à rechercher toutes les occurrences dans le code.

Puis, en haut à droite, nous avons le dossier event, chargé de regrouper tous les fichiers en lien avec les événements et les actions réalisables par l'utilisateur :

- Pour commencer *view-manager.js* s'occupe de gérer le changement de la vue lorsque l'utilisateur décide de se déplacer.

- Le fichier *controls.js* est responsable des actions liées aux boutons de déplacements manuels (les flèches directionnelles).
- Et pour finir le fichier *parameters.js* s'occupe des boutons de configuration du canvas : activer ou non les dessins de OpenCV, le bruit, changer la vitesse de simulation…

Le troisième dossier, *scenarios*, a été créé pour accueillir les différents scénarios et les méthodes communes à l'exécution de ceux que nous avons créés. Avoir un dossier spécial nous permet d'éviter au maximum la redondance et de créer de nouveaux scénarios plus facilement en utilisant les méthodes déjà existantes.

Et en dernier, nous retrouvons un dossier conçu spécialement pour le traitement des canvas avec OpenCV, il contient tout ce dont nous avons besoin pour traiter la caméra comme le simulateur.

Tests

Etant donné que notre projet touche à des éléments physiques, il est difficile de tester nos algorithmes en faisant de simples tests unitaires. Cependant nous avons tout de même pu réaliser des tests empiriques afin de reproduire au mieux les conditions réelles dans notre simulateur. Ainsi, afin de faire nos tests nous avons utilisé le simulateur que nous avons développé à cet effet.

Difficultés rencontrées

Apprendre à programmer l'Arduino et coder en C++ nous a posé de nombreux problèmes au début du projet, par exemple faire en sorte que l'Arduino soit connecté au serveur dès qu'il est alimenté, faire en sorte que l'arduino soit capable d'utiliser les moteurs qui lui sont branchés, ou encore utiliser des JSON en C++.

Nous avons également rencontré des problèmes lorsque l'on testait le mode manuel du robot réel et celui dans le simulateur, en effet, certains des robots réels n'ont pas leurs deux moteurs dans le même sens que les autres, or, le code de l'arduino part du principe que ce n'est pas le cas, nous avons donc dû inverser le

sens des moteurs dans le code Arduino et mit de côté les robots avec les moteurs inversés.

Dans le simulateur cette fois, nous avions rencontré un bug lorsque nous voulions faire se déplacer le robot à un endroit donné, en effet, le robot avait d'abord besoin de se tourner vers sa cible avant d'avancer, or, il arrivait que le robot tourne dans le sens le moins optimal possible, il arrivait dans le pire des cas, qu'il fasse un tour complet sur lui-même plutôt que de simplement tourner un tout petit peu dans l'autre sens. Il s'est avéré que cela était dû à la librairie Matter.js qui représentait les angles de moins l'infini à plus l'infini, et non pas dans une fourchette précise (par exemple de 0 à 360°), ce bug nous a pris beaucoup de temps et de ressources à corriger, mais nous avons réussit à trouver sa cause et à le corriger.

Planning

Dans cette partie nous allons passer en revue l'organisation et l'avancement de l'application durant l'étude préalable et les 4 premières itérations.

Etude préalable

Pour commencer, nous avons réalisé des diagrammes et des maquettes qui nous ont permis de prendre conscience de la complexité du projet et de trouver par la suite des idées et des solutions aux différentes épreuves qui nous attendaient. Nous avons aussi dû faire des choix de conception, ceux qui nous semblaient les meilleurs, par exemple nous avons décidé à ce moment-là de faire notre site web sur une seule page, en changeant juste son contenu en fonction du bouton cliqué.

Itération 1

Au cours de la première itération nous nous sommes tout d'abord familiarisé avec l'IDE Arduino afin de comprendre le fonctionnement du code du robot pour ensuite pouvoir l'adapter à notre cas. Puis nous avons continué avec un prototype global de l'application, c'est-à-dire un petit serveur fonctionnel, une interface fonctionnelle et un début de simulateur. Dans cette application il était possible d'afficher le flux de la caméra en direct sur un onglet. On a également commencé à utiliser la librairie OpenCV pour détecter et afficher les boules sur le canvas. Ensuite, avec la possibilité de changer d'onglet on pouvait visionner et interagir avec le simulateur et ses objets et enfin sur le dernier onglet on pouvait faire bouger manuellement le robot réel en appuyant sur des boutons de la page. Le serveur a permis de réaliser cette fonctionnalité car il sert de passerelle entre le robot et la page web, ceci est possible car tous les éléments sont connectés au même réseau.

Itération 2

Lors de la seconde itération, nous avons déjà effectué un refactoring de notre code afin de permettre une meilleure évolution. Puis nous avons continué à utiliser la librairie OpenCV pour cette fois-ci détecter les ArUco. Nous avons fait le choix d'en mettre aux quatres coins de la table de billard pour pouvoir déterminer ses dimensions sur le canvas. De plus on a traité les clics sur le canvas en récupérant leur position, cela nous a aidé à tester une autre fonctionnalité que nous avons programmé : le calcul de distance entre 2 points du canvas. En parallèle de ce travail sur la caméra, nous avons travaillé sur le simulateur en le rendant plus réaliste visuellement (couleurs, tailles, …) et en ajoutant de nouvelles configurations pré faites (on avait déjà une configuration random et on a rajouté une configuration simple, de billard et de foot).

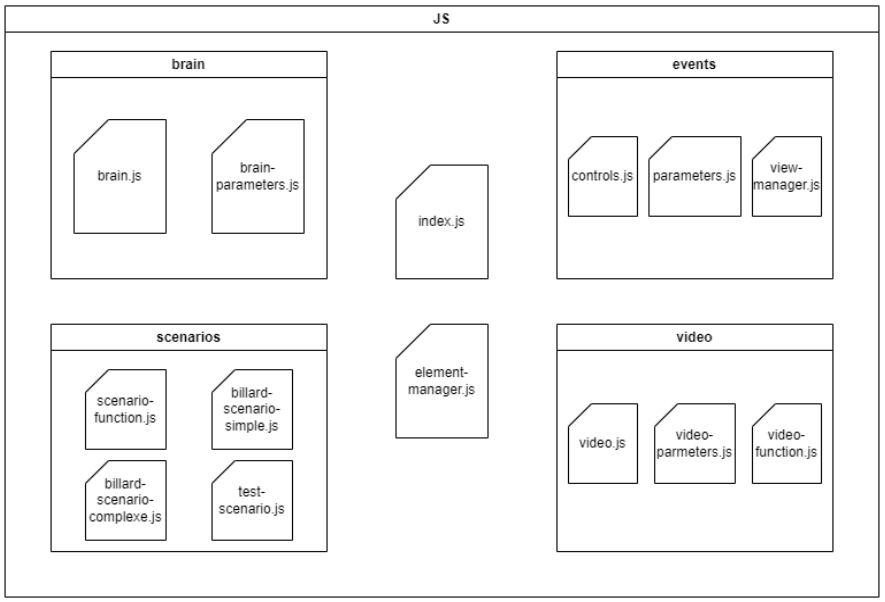
Itération 3

En ce qui concerne cette itération, nous avons grandement réutilisé les fonctionnalités programmées dans l'itération précédente pour les améliorer et les pousser plus loin. Nous avons par exemple fait en sorte qu'au clic sur le canvas, le robot réel se déplace à la position cliquée. Nous avons également amélioré la détection des ArUco pour obtenir leur orientation et ainsi connaître celle du robot. Et toujours avec les ArUco, nous avons différencié ceux des coins de la table de celui du robot. Cela nous a permis d'ignorer les boules détectées en dehors de la table (erreurs de détections, mais qui sont de toute façon inatteignables pour le robot). Puis nous avons fait en sorte de permettre à l'utilisateur de choisir le robot à contrôler. Nous avons aussi amélioré le simulateur en utilisant la détection de cercle directement dedans et nous avons pu commencer à programmer un scénario de test afin de nous préparer à en faire de plus complexes par la suite.

Itération 4

Lors de cette dernière itération, nous avons grandement amélioré notre scénario, le robot est maintenant capable de détecter la boule la plus proche et de la pousser jusqu'à ce qu'elle tombe dans un trou (pour l'instant il ne fait que la pousser sans prendre en compte la position des trous). Ce scénario est possible à la fois dans le simulateur et sur la table réelle. Concernant le simulateur, nous avons tenté de le rendre encore plus réaliste, nous avons donc ajouté du bruit pour brouiller la détection avec OpenCV et simuler les réelles interférences de la caméra. De plus, nous avons aussi ajouté la détection d'ArUco pour se rapprocher au maximum de la structure réelle du projet. Nous avons également basé la reconnaissance des trous de la table de billard grâce aux positions des ArUcos plutôt qu'en se basant sur la distance des cercles avec la bordure de la table.

Tout ce code commençait à devenir complexe à comprendre, nous avons donc refactorisé le fichier index.js et refait toute la structure des fichiers js (image explicative ci dessous). En plus de cela nous avons amélioré "l'intelligence" du robot, il est maintenant capable de se déplacer de manière plus fluide grâce à des mouvements en courbe. Et enfin, concernant l'expérience utilisateur, nous avons laissé le choix d'afficher ou non les traitements de OpenCV sur le canvas et avons développé une petite interface mobile pour permettre à n'importe qui de contrôler facilement les robots à distance.



Répartition du travail

Notre équipe est composée de 3 personnes, ce qui rend l'organisation plus facile, chacun peut facilement trouver quelque chose à faire sans empiéter sur le travail des autres. Nous avons donc décidé de diviser le projet en 3 "sous projets" et chacun peut travailler sur sa partie, nous avons donc dans les grandes lignes : Antoine FONTANEZ qui s'est principalement occupé de programmer le simulateur, Corentin FROGER qui à codé la détection d'OpenCV et l'Arduino, et Nathan PIERROT a fait le frontend en plus d'un soutien à Corentin.

Le tableau recense les principales fonctionnalités réalisées à chaque itérations et les personnes qui se sont investies dans le développement de chaque fonctionnalité :

Fonctionnalités réalisées	**Antoine FONTANEZ**	**Corentin FROGER**	**Nathan PIERROT**
Itération 1			
Prototype de l'interface web			X
Prototype du serveur Node.js	X		
Prototype du simulateur	X		
Connecter le robot au même serveur que le navigateur	X	X	X
Gérer le flux de la caméra dans le navigateur		X	X
Contrôler les robots manuellement	X	X	
Reconnaître les boules avec la librairie OpenCV		X	X
Afficher et bouger des objets dans le	X		

simulateur			
Itération 2			
Refactoring HTML/js, serveur Node.js et Arduino	X	X	X
Afficher le simulateur de façon plus réaliste	X		
Détecter la position des clics sur le canvas	X		X
Calculer la distance entre 2 points sur le canvas		X	
Reconnaître les ArUco avec OpenCV		X	
Ajout d'une page de chargement			X
Ajouter des configurations pré-faites pour le simulateur		X	X
Itération 3			
Être capable de déplacer le robot à un endroit précis (grâce à un clic sur le canvas)	X	X	X
Envoyer des ordres différents à différents robots	X	X	X
Exécuter un ordre pendant un certain temps ou jusqu'à la réception d'un nouvel ordre			X
Utiliser la détection de cercles dans le simulateur	X		
Programmer un scénario de test dans		X	

le simulateur			
Différencier les ArUco des robots et ceux des coins du billard	X		
Obtenir l'orientation du robot grâce à ArUco		X	
Ignorer les boules détectées en dehors de la table		X	
Itération 4			
Programmer un scénario de billard simple		X	X
Détection d'ArUco dans le simulateur	X		
Ajouter du "bruit" au simulateur	X		
Considérer les cercles détectés sur un ArUco comme de faux positifs	X	X	
Pouvoir modifier la vitesse de la simulation	X		
Calcul des trajectoires et de la vitesse du robot pour qu'il puisse bouger de manière fluide	X	X	X
Refactoring index.js et la structure des fichiers js	X	X	X
Faire en sorte de voir la vidéo avec et sans les dessins			X
Interface mobile simple pour contrôler les robots réels		X	

Présentation des éléments dont nous sommes fiers

Antoine FONTANEZ

La partie du projet dont je suis le plus fier est le simulateur. Le simulateur est très paramétrable, en effet on peut changer la configuration des objets, la vitesse de simulation, ajouter du bruit, etc.

Bien qu'il ne soit pas réaliste à 100%, ce qui me rend fier aussi est le fait qu'il prenne en compte une multitude d'aspects de la réalité :

- La détection OpenCV pour détecter les positions des boules et des ArUcos
- La mauvaise qualité de la caméra est simulé grâce à un système de bruit qui peut brouiller OpenCV pour tester des conditions défavorables
- Le simulateur reçoit exactement les mêmes requêtes que le vrai robot
- L'envoie d'ordres manuels grâce au même menu que le vrai robot (mode manuel)

Ainsi, il y eut plusieurs moments où nous avons utilisé le simulateur pour tester nos algorithmes de déplacement, et c'est en partie grâce à nos tests sur le simulateur que nous avons pu obtenir des déplacements plutôt satisfaisants pour le vrai robot. C'est pourquoi le simulateur est ce qui me rend le plus fier parmis tout ce dont j'ai travaillé au cours de ce projet.

Corentin FROGER

La partie *originale* dont je suis le plus fier est le fait de pouvoir contrôler manuellement les robots depuis un téléphone en se connectant au serveur. Bien que cela n'ai pas été demandé ou que cela ne soit pas spécialement utile, je me suis dit qu'avoir un moyen agréable de tester si on peut contrôler le robot serait sympa. De plus, j'ai pensé que cela pouvait être utile dans le cas où l'on créerait un scénario dans lequel l'utilisateur peut directement jouer avec le robot autonome, par exemple

un genre de chat / souris ou encore de voir lequel rentre des boules de billard le plus vite possible. En tout cas, il s'est avéré que c'était une bonne idée, car j'ai pu faire essayer le robot à d'autres étudiants, et ils ont trouvé ça pas mal.

Concrètement, il faut simplement que le téléphone se connecte au réseau du modem TP-LINK en wifi, puis de se connecter à l'adresse IP du serveur sur le port 8001 (http://192.168.137.1:8001 par exemple). Il faut malheureusement appuyer plusieurs fois sur les touches plutôt que de simplement rester appuyer dessus (pour l'instant en tout cas).

Nathan PIERROT

La partie que j'ai programmée et dont je suis le plus fier est la manière dont les robots exécutent les ordres et plus particulièrement le fait qu'ils doivent exécuter le dernier ordre qu'ils reçoivent en remplaçant le précédent.

Concrètement, la boucle infinie de l'Arduino regarde en permanence si un ordre est reçu. Quand un nouvel ordre est reçu, l'Arduino remplace le précédent et l'exécute pendant le temps indiqué dans la requête ou jusqu'à la réception d'un nouvel ordre. Pour connaître le temps d'exécution de l'ordre reçu, l'Arduino récupère l'heure de son horloge interne au moment de la réception de l'ordre ainsi il peut comparer les 2 temps à chaque exécution de sa boucle infinie afin de savoir quand s'arrêter.

La solution paraît simple mais ce n'était pas une tâche si aisée. Premièrement le code de l'arduino est du C++, c'est un langage bas niveau que nous avons vu en cours mais jamais de manière aussi pratique. Il m'a donc fallu replonger dans la syntaxe et les normes du langage avant de pouvoir réellement coder. Deuxièmement, le code de l'Arduino fonctionne d'une manière assez spéciale avec 3 fonctions principales et essentielles :

- *void socketIOEvent(...)* : cette fonction permet de gérer et de traiter les événements reçus par le réseau (les Websockets). Cela peut correspondre à la connexion et déconnexion du robot mais également aux différents types d'ordres reçus. La fonction permet alors d'initialiser et de mettre à jour les

variables utiles à l'exécution des ordres, par exemple la vitesse des moteurs ou la direction dans laquelle ils doivent tourner.

- *void setup()* : la fonction permet d'initialiser les différents paramètres de l'Arduino : les paramètres réseau et les paramètres des moteurs (à la base arrêtés) pour vérifier que tout est bien à jour et fonctionnel.
- *void loop()* : c'est cette méthode qui s'exécute en permanence et qui envoie les ordres aux moteurs, en fonction des valeurs des variables (mises à jour lors de la réception de nouvelles requêtes avec la fonction socketIOEvent). C'est aussi cette méthode qui vérifie le temps d'exécution d'un ordre.

Suite du projet

Objectifs à atteindre

L'objectif premier est d'avoir un projet fonctionnel et impressionnant pour la journée portes ouvertes du 1er mars. C'est-à-dire un robot plutôt intelligent qui réussit à pousser les boules plutôt rapidement et en peu de coups dans les trous et capable de se déplacer intelligemment. Ensuite, un second objectif est d'ajouter d'autres scénarios et d'autres jeux en mettant en concurrence plusieurs robots par exemple. Un objectif parallèle est de rendre l'application plus fonctionnelle et plus agréable à utiliser, nous pensons notamment à changer la disposition des boutons afin qu'ils prennent moins de place sur l'écran et qu'ils soient disposés en fonction de leur usage.

Planning prévisionnel des itérations suivantes

Itération 5

Pour commencer, nous comptons terminer le scénario complexe que nous avons commencé dans l'itération 4, ce scénario permet au robot de jouer plus intelligemment qu'actuellement, en ce moment il pousse juste la balle la plus proche de lui. Nous voulons donc faire en sorte qu'il pousse les boules en fonction des trous les plus proches, afin de terminer plus rapidement la partie. Ensuite nous voulons ajouter un menu permettant de regrouper tous les boutons à un seul endroit, cela permettra d'épurer la page. Et finalement nous avons pour objectif de permettre à l'utilisateur de stopper un scénario. En cliquant sur le bouton GO l'utilisateur lance le scénario, ensuite le bouton se transforme en un bouton STOP sur lequel l'utilisateur peut de nouveau appuyer pour arrêter le scénario.

Itération 6

Lors de cette itération nous avons pour objectif de réaliser un nouveau jeu, tout d'abord dans le simulateur puis ensuite le transposer dans le monde réel. En parallèle nous allons corriger les petits bugs connus et effectuer de nouveau un refactoring du code. Notre application est en constante évolution et prend des proportions toujours plus grandes, si nous voulons continuer à la faire grandir, il est obligatoire d'adapter notre structure.

Itération 7

Pour cette dernière itération nous n'avons pas prévu d'ajouter de grosses fonctionnalités. Nous voulons avoir une application la plus propre possible, donc nous allons retravailler le style de l'application et optimiser au maximum nos différentes fonctions. Et afin de rendre notre application la plus fonctionnelle possible, nous souhaitons améliorer notre version mobile pour que n'importe qui puisse utiliser notre application.

Conclusion

Ce que ce projet nous a apporté

Ce projet nous a apporté de nombreuses connaissances en robotique grâce à la programmation Arduino, nous avons également utilisé Node.js pour la première fois, de même pour les websockets. Nous avons également découvert comment utiliser, et comment fonctionne OpenCV pour la détection d'objets tels que les ArUcos et les cercles. Nous avons aussi appris à utiliser une librairie de physique réaliste avec Matter.js

Améliorations possibles

Nous pourrions par exemple créer des scénarios où plusieurs robots seraient concernés, par exemple une partie de foot, de chat et souris, où encore que l'utilisateur soit capable de directement jouer avec le robot, par exemple à l'aide d'une "télécommande" pour contrôler un robot manuellement tandis qu'un autre est géré automatiquement par l'application.