ELE795 / LOG795

Projet de fin d'étude

Apprentissage par renforcement appliqué à des robots bipèdes

PAR

Pouplier, Thierry (POUT22029207)

Brousseau-Rigaudie, Brice (BROB07039301)

Dumont, Marc Antoine(DUMM05099303)

Travail présenté

à

Tadj, Chakib

et

Alain April

MONTRÉAL, Le 12 avril 2019

Table des matières

Résumé

Ce présent rapport contient les informations techniques et théoriques pour le projet de fin étude du baccalauréat en génie à l'ETS réaliser par trois étudiants : Brice Brousseau-Rigaudie (génie électrique), Thierry Pouplier (génie électrique) et Marc Antoine Dumont (génie logiciel). Le projet est la conception d'un processus d'apprentissage machine par renforcement appliqué à des robots bipèdes jouant au soccer. Ce rapport est introduit par une mise en contexte de la problématique et des objectifs visés. Ensuite, la revue de la documentation est décrite. Elle contient les informations sur le q-learning, sur les parties de code utiles au projet et sur le simulateur utilisé et modifié pour ce projet. En troisième lieu, ce rapport détaille la méthodologie de travail à travers les éléments suivant : le choix de l'implémentation, le langage utilisé pour la programmation et la méthode de versionnement. La quatrième partie est le processus de conception dans lequel on y retrouve un résumé pour la conception du code de la partie apprentissage et la partie simulation. Pour conclure le présent document, on y retrouve un bref rappel du contexte de la problématique et des objectifs, la présentation des principaux résultats et les diverses perspectives futures.

1. Rappel du contexte, de la problématique et des objectifs visés

Naova est un club scientifique de l'ÉTS fondé en 2017 par Jonathan Fortin, Thierry Pouplier, et Alexandre Doyle. Le club scientifique Naova fut fondé dans le but de participer à une compétition internationale de robotique pendant laquelle une équipe de 5 robots joueront ensemble afin de gagner une partie de soccer contre une autre équipe de robots comme on peut voir dans la figure ci-dessous. Par ailleurs, Naova est composé d'étudiants et d'étudiantes de différentes expertises se penchant sur la programmation de robots Nao.

Figure 1 : Match de soccer de robots Nao

La robotique et l'informatisation des procédés sont devenues des enjeux majeurs pour tous les secteurs du milieu des affaires. Une des problématiques pour bien des entreprises est la pénurie de spécialistes qualifiés en informatique, en programmation, en TI et en génie logiciel.

Le club scientifique Naova de l'ÉTS participe annuellement à une compétition se nommant Robocup, une initiative scientifique internationale dont le but est de faire progresser le domaine des robots intelligents (non seulement pour le secteur industriel, mais pour tous les secteurs d'activités).

La Robocup est l'une des plus grandes compétitions internationales de robotique regroupant des futurs spécialistes passionnés de l'informatique et de l'automatisation qui, bénévolement, investissent temps et efforts pour se perfectionner. Par conséquent, Naova ÉTS et ses membres sont fiers d'être représentants du Canada dans la division « Soccer Standard Platform League », considérant que Naova est la seule équipe canadienne dans cette division de la compétition.

Les résultats du club Noava, pour 2018, ont été très positifs considérant que leur participation en 2018 était leur toute première. Malgré le terrain inconnu, le travail colossal a été hautement récompensé. Naova atteint des sommets pour l'année 2018 et rapporte les honneurs de la 2e place dans la division Challenger. Maintenant direction 2019 ! L'objectif, sachant les capacités et les talents à travers l'équipe, n'est rien de moins que la première place.

Ce club a fait appel aux étudiants de PFE pour réaliser cet objectif. De ce fait, notre équipe propose un **système d'apprentissage par renforcement** pour permettre au robot d'améliorer leurs décisions afin de remporter les matches.

Pour réaliser ce projet, il faut que notre équipe trouve un simulateur adéquat permettant d'avoir les caractéristiques de jeux qui se rapproche le plus possible de la réalité. Il faut alors adapter le simulateur pour l'apprentissage, implémenter une architecture d'apprentissage par renforcement et apprendre un aptitude. Ce rapport sera divisé par une revue de la documentation, par la méthodologie de travail, par le processus de conception et par la présentation des résultats. La revue de la documentation est décrite par l'apprentissage par renforcement et décrite par les modules du robot Nao. Par la suite, la méthodologie de travail est détaillée par l'évaluation des différents algorithmes possibles à utiliser et leurs caractéristiques et par l'aptitude que le robot doit apprendre. Pour finir, les étapes de la conception choisie seront détaillées et présentées en détaillant le processus et le résultat.

2. Revue de la documentation

La revue de la documentation est divisée en plusieurs partis subséquentes. L'apprentissage par renforcement est le premier point aborder. Cette partie rapporte la théorie sur le q-learning, le q-learning approximé et les éléments essentiels à leur conception. La deuxième partie est sur le code de l'équipe B-human présentement utilisé pour les robots Naova. Elle présente les différents modules permettant au robot de jouer au soccer dans les règles de l'art. En dernière partie, le simulateur en trois dimensions qui est également fait par l'équipe allemande B-human est sommairement détaillé.

2.1 Apprentissage par renforcement[1]

Le matériel du cours d'intelligence artificielle de l'université de Berkeley a grandement été utilisé pour ce projet. Ce cours a fourni l'information sur plusieurs concepts très importants de l'apprentissage par renforcement. Le diagramme suivant présente le fonctionnement de cet algorithme. L'environnement est le simulateur avec toutes les caractéristiques du jeu. Le «state» est l'état du jeu soit la position du ballon et des joueurs. Le «reward» est une variable qui permet de quantifier la valeur de l'état pour ce qui est bénéfique dans le jeu, c'est à dire : le ballon dans le but adverse équivaut au maximum de «reward» et le ballon dans notre but équivaut au minimum de «reward». Une variable pour lequel l'«agent» contient l'algorithme d'apprentissage par renforcement. Celui-ci aura but de maximiser la «reward» cumulative.

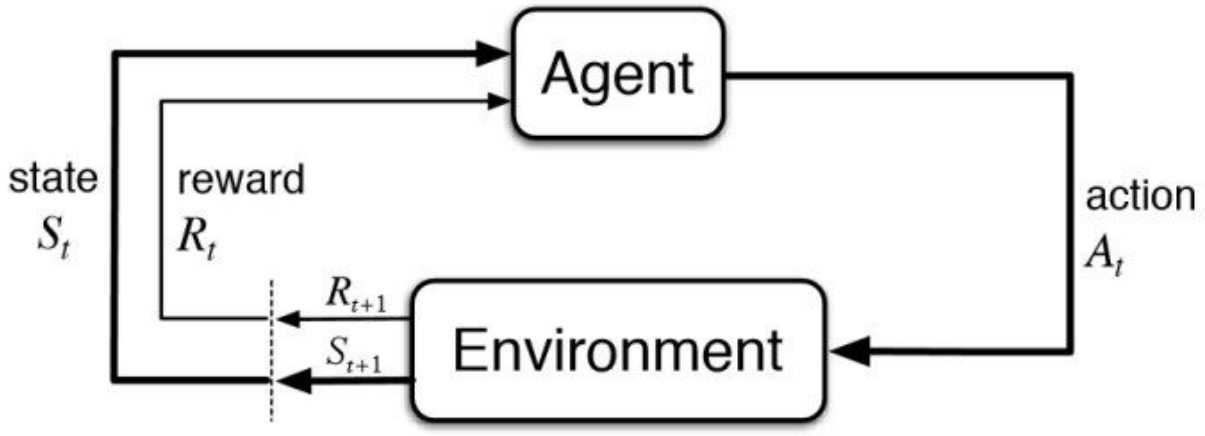
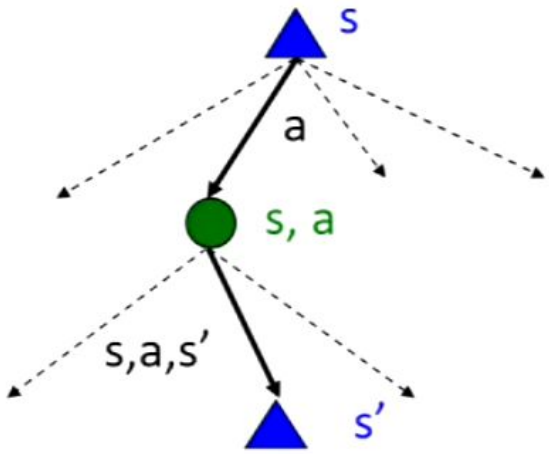


Figure 2: Apprentissage par renforcement
(https://www.kdnuggets.com/images/reinforcement-learning-fig1-700.jpg)

[1] "Reinforcement Learning - Berkeley AI Materials - UC Berkeley." 26 août. 2014, http://ai.berkeley.edu/reinforcement.html. Date d'accès : 23 mars. 2019.

Les principaux concepts utilisés pour l'amélioration de l'intelligence des robots sont les concepts de Markov, le q-learning, le concept de discounting des récompenses, les méthodes d'exploration, le taux d'apprentissage et finalement l'approximation du q-learning qui sont détaillés par la suite.

2.1.1 Processus de décision markovien



Le processus de décision markovien est utilisé dans l'algorithme de Q-learning pour permettre d'avoir l'état suivant s'. Un processus de décision est un processus de contrôle stochastique discret. À chaque étape, le processus est dans un certain état s et l'agent choisit une action a. La probabilité que le processus arrive à l'état s ′ est déterminée par l'action choisie. Plus précisément, elle est décrite par la fonction de transition d'états T (s , a , s ′). La variable s' est donc déterminée par le s actuel et l'action sélectionnée par le décideur. Cependant, pour un s et un a , le prochain état est indépendant des actions et des états précédents.

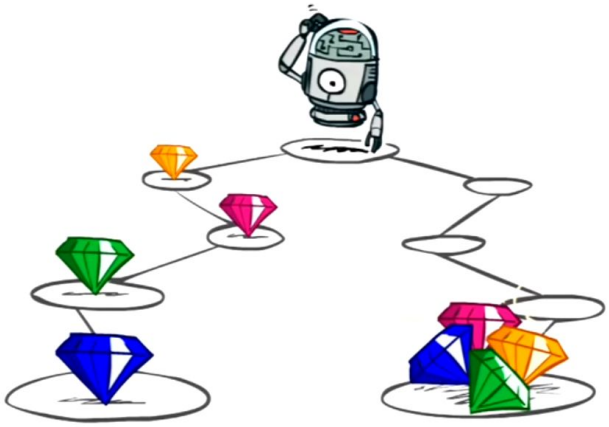
2.1.2 Le Q-Learning

L'avantage du Q Learning c'est qu'il ne nécessite aucun modèle initial de l'environnement. Cela nous permet d'appliquer ce concept d'apprentissage à la réalité du soccer qui est un sport dont la majorité des décisions sont imprévisibles. Le Q Learning est une méthode d'apprentissage qui est appliquée pour trouver une suite d'actions associées à des états (politique) d'un

processus de décision markovien quelconque. Cela fonctionne par l'apprentissage d'une fonction de valeur d'état qui permet de déterminer le potentiel bienfait (récompense) de prendre une certaine action dans un certain état en suivant une politique optimale.

$$Q(s,a) \leftarrow (1-\alpha)\, Q(s,a) + (\alpha)\left[r + \gamma\, max_{a'}(\, Q(s',a')\,)\right]$$

2.1.3 Discounting



Le «discountinge» est la variable gamma qui est multipliée à la récompense lors de l'apprentissage comme indiqué dans la figure suivante. Cela permet de réduire les valeurs futures des récompenses permettant ainsi de rendre la sommation infinie des récompenses finies et de favoriser les récompenses immédiates au détriment de ceux que l'on peut avoir plus tard. Dans l'équation suivante, U représente l'«utility», c'est-à-dire la somme de toutes les récompenses d'une suite d'action. Elle représente le meilleur chemin lors que sa valeur est maximum. Pour le q-learning approximé, on utilise Q(s,a).

$$U([r_0, r_1, r_2, \ldots]) = r_0 + \gamma r_1 + \gamma^2 r_2 \cdots$$

2.1.4 Exploration

L'exploration est un concept de l'apprentissage permettant d'essayer des actions aléatoirement. On peut modifier cet aspect aléatoire grâce à E-greedy.

2.1.4.1 E-greedy

E-greedy est la variable d'exploration concernant les actions. Quand elle est petite, elle permet à l'agent d'essayer les actions aléatoirement. Plus elle est élevée plus cela restreint l'exploration à suivre les actions de la stratégie («policy») apprises jusqu'à présent.

2.1.4.2 Fonction d'exploration

La fonction d'exploration permet d'ajouter une valeur à l'«utiliy» tel que u=u+k/n. La variable u est l'«utility», k est un chiffre suffisamment gros pour que l'exploration ait de l'importance et la variable «n» est le nombre de fois que l'agent a essayé l'état.

2.1.5 Domaine d'action

Les actions d'un jeu peuvent être simplement discrétisées et finies comme pour l'exemple du pacman qui a le choix d'aller dans 4 directions différentes. Cependant, ce n'est pas le cas pour tous les domaines de l'informatique, comme le robot NAO auquel les possibilités de bouger et de frapper la balle sont définit par des variables flottantes c'est-à-dire qu'il y a une infinité de possibilités.

Pour l'apprentissage par renforcement, ce domaine est très important et il se doit d'être restreint et fini. La possibilité qui s'offre à notre équipe est de discrétiser les actions par des valeurs physiques probables et de limiter l'action par l'augmentation ou la diminution de la vitesse ou de l'angle de marche par exemple.

2.1.6 Approximation

Le Q-Learning a malheureusement deux problèmes majeurs qui l'empêchent d'être utilisé sur des problèmes de plus en plus complexes.

La première limite est la taille du tableau de valeurs Q. Celui-ci devient exponentiellement plus gros avec chaque action et chaque état possible supplémentaire. Il devient impossible de à la fois de stocker toutes ces valeurs, mais aussi de les apprends.

Le deuxième désavantage est l'absence de transfert de l'apprentissage entre plusieurs états semblable. La figure ci-dessous montre trois états très semblables du jeu Pacman. Malgré leur ressemblance, l'algorithme devrait explorer puis apprendre de ces trois états indépendamment.

Figure 3 : Jeux pacman

Le Q-Learning approximé vient pallier à ces deux problèmes en calculant sur le coup une approximation de la valeur Q. Il n'y a donc pas de tableau de valeurs Q stocké. Aussi, l'approximation peut être faite de manière à prendre en compte les ressemblances entre les états. La méthode proposée dans le cours de Berkley est d'écrire des heuristiques ("feature") et de faire l'apprentissage des poids de ceux-ci. C'est la méthode qui sera tentée en premier. Il sera ensuite possible de pousser cette idée plus loin en faisant l'utilisation de réseaux de neurones pour à la fois faire l'apprentissage des poids, mais aussi des "feature" pour estimer Q. Il est aussi possible de rajouter un deuxième réseau de neurones pour choisir de la police à exécuter. Cet algorithme se nomme alors "actor-critic".

$$\text{difference} = \left[r + \gamma \max_{a'} Q(s', a')\right] - Q(s, a)$$

$$Q(s, a) \leftarrow Q(s, a) + \alpha\,[\text{difference}]$$

$$w_i \leftarrow w_i + \alpha\,[\text{difference}]\, f_i(s, a)$$

2.2 Rapport annuel de l'équipe B-human[2]

La consultation du rapport annuel de l'équipe Allemande B-Human a permis à l'équipe Naova d'avoir un code de base solide, puisque la base de code provient de

[2] "B-Human 2018." https://www.b-human.de/downloads/publications/2018/CodeRelease2018.pdf. Date d'accès : 23 mars. 2019.

cette équipe. B-human contient plusieurs modules qui communique parallèlement entre eux. Il y a le module de marche, le module de balle, le module du joueur et celui du **behaviorControl**. Ces modèles seront expliqués plus en détail ci-dessous.

2.2.1 Système de message et de module

B-Human participe à la RoboCup depuis 11 ans, et ont donc développé plusieurs modules très utiles pour travailler avec le Nao. Ces modules ont chacun une utilité et communique entre eu à l'aide d'un système de messages nommé "représentation". Chaque module doit s'inscrire au message qu'il veut recevoir et doit déclarer quel message il fournira. Tous les messages sont accessibles par l'entremise d'une mémoire partagée appelée le "blackboard".

2.2.1.1 Walking Engine

L'un de ces modules est le module de marche. Celui-ci s'occupe de contrôler les jambes du Nao pour le faire avancer dans une direction voulue. Plus précisément, trois modes de marche sont offertes par le module. La première demande d'entrer une cible et il essayera de se rendre à celle-ci en ligne droite, sans évitement d'obstacles. Les deux autres modes sont respectivement la marche à vitesse absolue et celle à vitesse relative. Ceux-ci demandent des vitesses en X, en Y puis une vitesse de rotation en entrée.

2.2.1.2 Modèles

Plusieurs autres modules sont utiles au projet. Une catégorie de ceux-ci est l'ensemble des modèles créé à partir des capteurs du robot. Il a été utilisé principalement deux modèles dans ce projet qui sont «Team Ball model» et «player model».

Modèle de balle de l'équipe

Le modèle de la balle est très utile, puisque la tâche à apprendre est de contrôler ce dernier. Le modèle est formé à partir des observations faites par les caméras, puis traité par un module de vision. Les détections de la balle de tous les joueurs de l'équipe sont ensuite regroupées pour former un approximé de la position et de la vitesse de balle sur le terrain.

Modèle du joueur

Le module du modèle du joueur regroupe simplement la position et la vitesse approximées du joueur. Ces informations sont estimées par la combinaison de l'odométrie de la marche et de la détection des lignes sur le terrain par les caméras.

2.2.1.3 Modules BehaviorControl

Le module **BehaviorControl** est celui qui prend les décisions de haut niveau. C'est ici, qu'avec l'aide des informations fournies par les modules mentionnés plus haut, que le robot choisisse de frapper le ballon, de marcher ou de se relever par exemple.

State machine (cabsl language)

À la base, ces prises de décision se faisaient par une machine à états. Ceux-ci sont écrites dans le langage de programmation Cabsl[3]. Ce langage est aussi une création de l'équipe B-Human. Le Cabsl permet le développement rapide et simple de machines à états.

En 2018, des MEF ont été développés par Naova permettant au joueur d'avoir un rôle pertinent avec le soccer. Ces rôles utilisent d'autres MEF représentant des habiletés spécifiques. Les MEF d'habileté vont à leur tour demander aux modules concernés d'effectuer la série d'action nécessaire pour arriver à la complétion de la tâche.

2.2.1.4 Librairie

À l'intérieur du module **BehaviorControl**, il est possible de créer des libraires. Celles-ci sont utilisées pour faire les calculs que les MEF ont besoin. C'est dans une telle librairie qui a été choisie d'implanter pour notre algorithme d'apprentissage.

[3] "B-Human 2018." https://www.b-human.de/downloads/publications/2018/CABSL.pdff. Date d'accès : 23 mars. 2019

2.3 Simulateur

Le simulateur créé par le groupe d'étudiants allemand est conçu pour simuler une à deux équipes de robot Nao. Le choix du nombre de joueurs peut être modifié dans le fichier de configuration .ros2 que l'on ouvre avant une simulation. Le simulateur simule les conditions réelles du jeu de soccer avec la gravité, la friction du terrain , le temps de jeu, etc. Ci-dessous, on y retrouve la figure du terrain de la simulation.

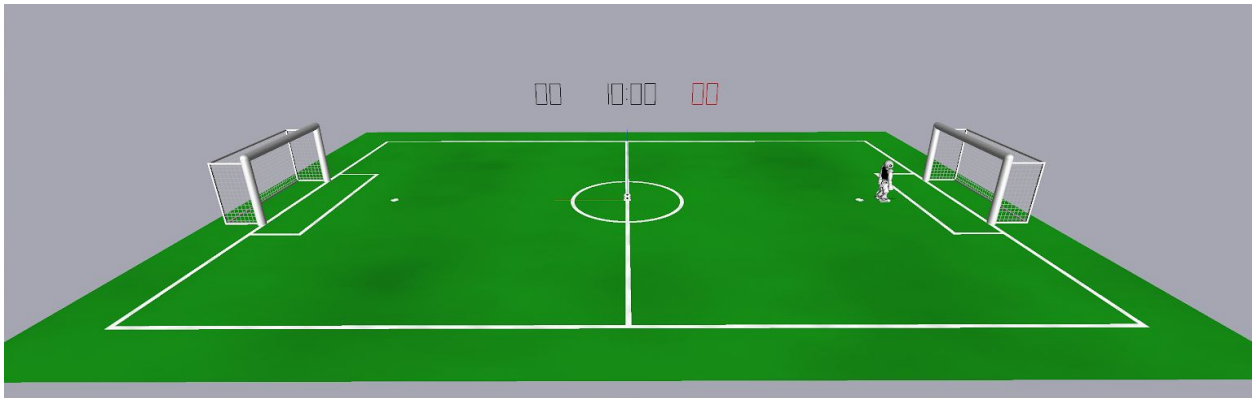


Figure 4: Terrain de soccer du simulateur NAO 3D.

3. Méthodologie de travail

Dans cette partie, il sera question d'analyser les différents algorithmes permettant l'amélioration des performances du robot Nao et par la suite de choisir l'environnement d'apprentissage selon les exigences.

Le tableau suivant présente les avantages et désavantages de plusieurs algorithmes pouvant améliorer les performances du robot. Les concepts choisis pour comparer les différents algorithmes ont été déterminé en fonction des critères requis. Par la suite, l'algorithme qui répondra le mieux au critère sera déterminer.

Algorithme	Machine à état fini	Q-learning	Q Approximé	Actor critic	Critères requis
Domaine d'action	limité par le programmeur	un ensemble fini de nombres entiers	un ensemble infini de nombres entiers	un ensemble infini de nombres entiers	un ensemble infini de nombres entiers
Nombre d'état utile	limité par le programmeur	fini	fini	infini	infini
Temps pour l'amélioration	Long	Court	Court	Court	court
Temps d'implémentation et développement	Long	Moyen	Moyen	Long	moyen/court
Dépendance à un simulateur	Aucune	Oui	Oui	Oui	Au besoin

Tableau 1: Comparaison des caractéristiques d'algorithme pour l'amélioration de la performance du robot Nao.

Comme expliqué précédemment, l'équipe Naova utilise actuellement des machines à états finis pour le choix d'actions des robots. Cette technique a une limitation sur la quantité d'états possibles dû à la complexité qui augmente exponentiellement avec l'ajout de nouveaux états. De plus, il faut implémenter manuellement le changement d'état. Cela rend l'humain complètement responsable de la stratégie des robots. Pour pallier à ces problèmes, une approche d'apprentissage par ordinateur a été choisi. Pour apprendre un comportement, l'apprentissage par renforcement est une solution efficace et facile à implémenter. Après

l'analyse des différentes options d'apprentissage par renforcement, le Q-Learning semble la plus efficace et la plus facile à implémenter. Comme dit précédemment étant donné que le robot se trouve dans un monde non fini, le Q-Learning demandent énormément de ressource. C'est pour cela que le **Q-Learning Approximé** s'avère être le meilleur choix.

Pour entraîner le module , il faut un **environnement de simulation** qui est le plus similaire à la réalité. L'équipe Naova a déjà un système de simulation. Cette simulation est très semblable à la réalité, cependant elle demande beaucoup de ressources. Ce qui fait qu'elle ne peut pas être exécutée plus rapidement qu'en temps réel et qu'une seule instance peut être exécuter en même temps. Ce qui veut dire que l'apprentissage sera long.

Pour une solution plus rapide d'exécution, il serait possible d'implémenter une simulation à deux dimensions. Cela aurait demandé beaucoup de temps de développement. De plus, comme la simulation est moins représentative de la réalité, il est incertain que les apprentissages donneraient de bons résultats dans le monde réel.

L'apprentissage du Q-Learning dans le cours de l'université Barkley a été fait en Python, car les devoirs contiennent un squelette en python. Cependant, pour ce qui est du projet de Naova, tout est en C++. Pour faciliter la compréhension et la portabilité de notre projet, il a été choisi de faire le projet aussi en C++.

Dans le cadre de ce projet, il était plus efficace de continuer avec la simulation 3 dimension déjà existante. Celle-ci fait appel à des modules appelé "contrôleur". Ces contrôleur font office d'interface entre le code du robot et la simulation.

Pour le suivi, l'attribution et la gestion des tâches, le logiciel Trello fût choisi. Le club Naova avait déjà un gestionnaire de tâche, mais celui-ci contenait tous les projets. Pour faciliter la communications avec les professeurs et les élèves du projet de fin d'étude, il fallait un gestionnaire exclusivement pour notre projet. Trello facilite grandement la compréhension de toutes les parties prenantes dû à son interface utilisateur convénient. Il est possible d'assigner chaque tâche et de donner une date de fin. Cela permet d'avoir un suivi plus précis de l'avancement des tâches.

Le versionnement du code du club de Naova est fait avec la technologie Git avec les serveur GitLab. Il a été décidé de garder ce mode de fonctionnement et de créer une branche nommée **feature/qlearning**. Un fois le module fonctionnel, nous effectuons le merge avec la branche de développement principale.

4. Processus de conception

Le processus de conception est divisé en trois partis subséquentes. La première est la remise en question des impacts que le projet a sur les aspects suivant : l'environnement, l'économie, le social et la technologie. La deuxième décrit les aspects technique de la conception de l'algorithme et la troisième détaille l'environnement d'apprentissage pour les robots Nao.

4.1 Aspect éthique,environnemental social et économique

La conception d'une intelligence artificiel pour le robot Naova amène à se poser la question suivante : Aurait-il des aboutissants négatifs ou positifs sur les parties prenantes pour les aspects environnemental, social, économique et technologique? En premier lieu, il n'y a aucun impact favorable ou défavorable pour l'aspect environnemental. En effet, l'amélioration des robots Nao qui joue au soccer n'a pas vraiment de lien avec l'environnement. Concernant l'aspect économique, si dans un cas échéant le projet permettrait au club Naova de participer au jeux de roboCup 2019 en Australie, le club pourrait obtenir des commanditaires et même l'ETS pourrait davantage commandité le club. Le code conçu serait accessible et open source pour toute la communauté de programmeurs de robot Nao.

4.2 Apprentissage par renforcement

Concernant la partie technique, nous avons commencé notre démarche de conception par faire l'apprentissage des concepts d'apprentissage machine par renforcement en regardant les cours d'intelligence artificiel du cours de Berkeley, pour ensuite compléter avec succès les exercices 4 à 8 de ce cours. Ces exercices nous ont fait bien comprendre les différents concepts de l'apprentissage par renforcement. Les algorithme développé a été testé dans différents environnements dont un jeu en grille contenant des récompenses et des pénalisées (figure 1), un robot simulé apprenant à marcher à l'aide d'un membre à deux axes (figure 2), et le jeu pacman (figure 3). C'est à travers ces divers exercices que l'équipe a bien assimilé les concepts de l'apprentissage par renforcement.

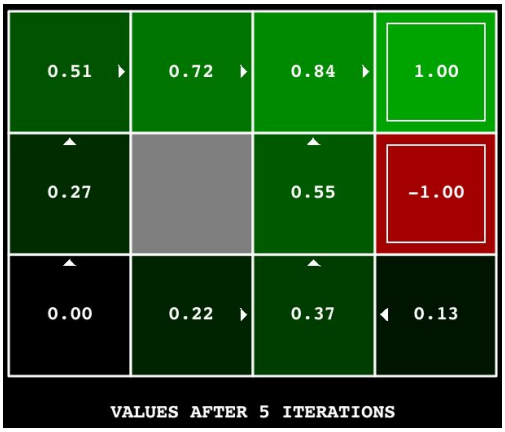


Figure 1: Gridworld

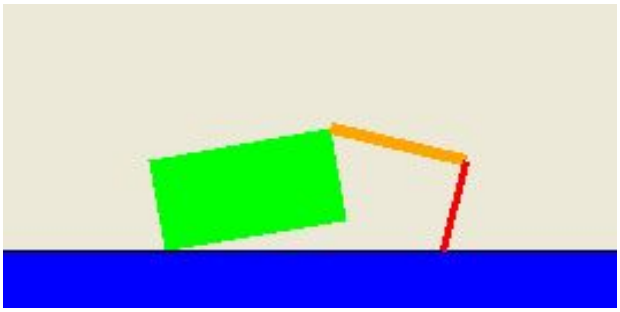


Figure 2: Crawler

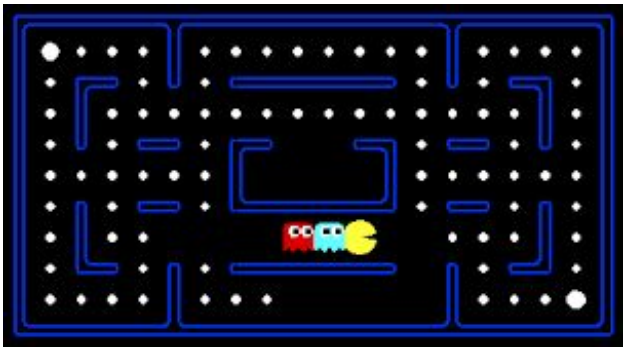


Figure 3: Pacman

4.3 Apprentissage du dribble

Pour l'apprentissage du dribble, le joueur a comme objectif d'atteindre la balle et de se diriger avec celle-ci. Il est nécessaire de modifier la simulation pour pouvoir automatiser cet apprentissage afin que le joueur reste sur le terrain. La majeur partie du code qui nous intéresse se trouve dans **Controller/GameControler.cpp.** Afin d'arriver à notre objectif le plus rapidement et limiter les modifications nécessaire au contrôleur, nous avons choisi comme cible le milieu du but. De cette manière, nous évitons d'avoir besoin d'une communication entre le contrôleur et les modules. Cette communication semblait différente et plus compliqué que ceux fait entre les modules. Une fois que le ballon arrive dans le but ennemie, le contrôleur changera aléatoirement la position du robot et du ballon pour une nouvelle séance d'apprentissage. Comme on peut voir dans la figure suivante, l'étoile représente la destination final pour le joueur qui sera positionné dans le but. Le cercle désigne le ballon et les deux rectangles sont le robot Nao. Les variables représentent les angles et les distances relatives aux positions.

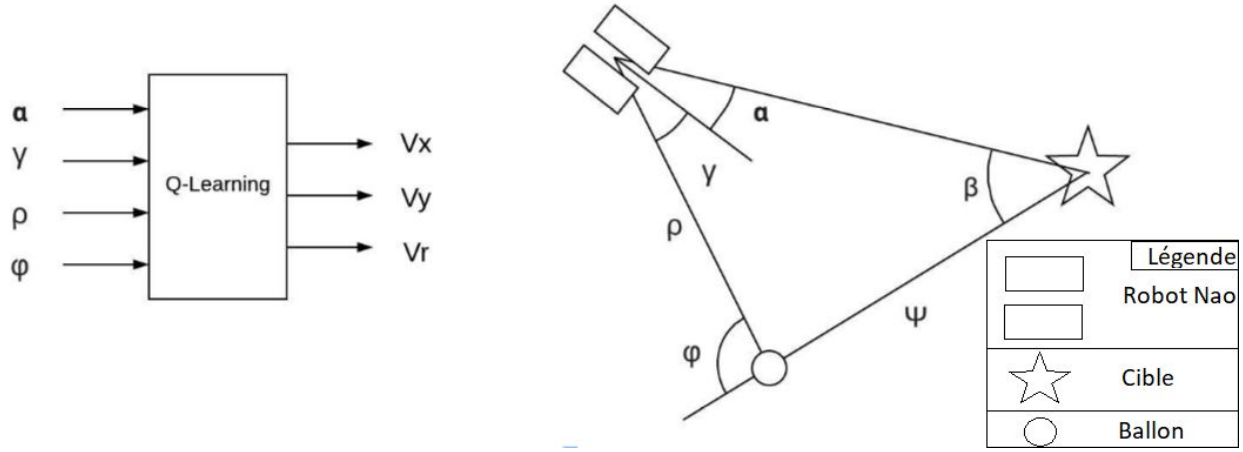


Figure 6: Schéma des entrées et des sorties du Q-Learning versus le schéma présentant les variables d'angle et de distance par rapport à la cible et le ballon.

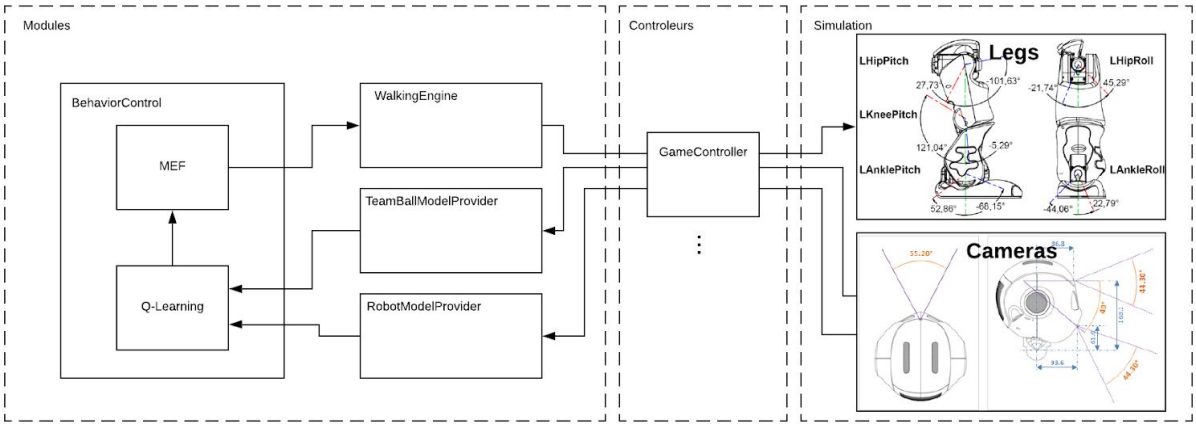


Figure 6: Schéma du système de modules

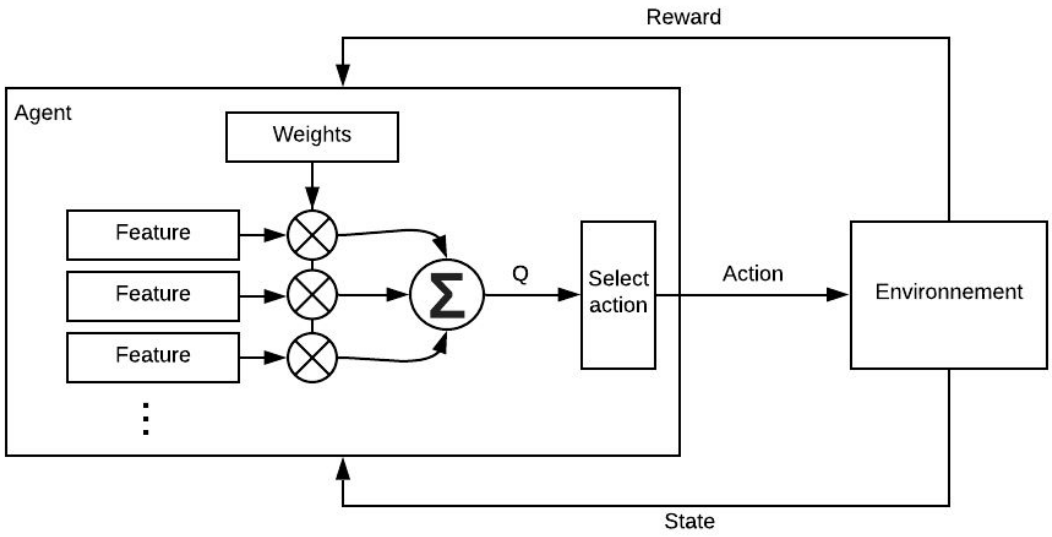


Figure 7: Schéma du Q-Learning Approximé

Afin de réussir à faire de l'apprentissage machine sur les Nao, nous avons créé une libraire à l'intérieur du module **BehaviorControl**. Cette librairie reçoit les informations qu'elle a besoin pour faire le Q-Learning par d'autres modules spécialisés. L'information est transmise par l'intermédiaire de représentation. Les représentations reçus contiennent la position du robot dans son environnement, la position du ballon et la position de la cible. La librairie calcule plusieurs heuristiques. Les heuristiques servent d'approximation de la valeur Q dans notre Q-Learning approximé. Avec chaque interaction avec l'environnement, l'algorithme améliore la valeur des poids qui est donné aux heuristiques dans le calculs de la valeur Q. Cette amélioration est faite par rapport à la récompense qui est reçu après chaque actions. La librairie sauvegarde ensuite les poids apprit dans un fichier csv pour une référence futur. La librairie

produit à son tour une représentation contenant la vitesse désirée en X, Y et en rotation. Ces vitesses sont reçues par la machine à états qui s'occupe d'effectuer le dribble. Cette machine à états est appelé au besoin par les machines à états de plus haut niveau au besoin. La machine à états remplit simplement la demande de déplacement et l'envoie au module WalkingEngine.

4.4 Création d'un module

Pour créer un nouveau module, il faut bien comprendre l'architecture du code préexistant. Celui-ci fait beaucoup appel à des macro pour simplifier la création d'un module, mais cela a le désavantage de rendre la compréhension du code plus ardu. La première étape pour créer un nouveau module est d'avoir le macro MAKE_MODULE(NomDuModule). Chaque module doit absolument avoir un fonction update(). Celle-ci est appelé à chaque cycle. De plus, la classe du module doit hériter de sa classe de base créé par le macro. La déclaration de la classe doit suivre le gabarit suivant:

```
class NomDuModule : public NomDuModuleBase
{…}
```

Comme décrit précédemment dans la revue de documentation, les modules utilise un système de message appelé représentation pour s'échanger de l'information. Dans le fichier «header» du module, les représentations auquel celui-ci s'inscrit et fournit doivent être indiqués de la manière suivante:

```
MODULE(NomDuModule,
{,
    REQUIRES(LeNomDuneRepresentation),
    USES(UneAutreRepresentation),
    PROVIDES(LeNomDuneRepresentationFournitParLeModule),
});
```

Le mot-clé REQUIRES indique au macro que ce module a besoin de la représentation mentionné entre parenthèses. Le module ne s'exécutera pas si cette représentation n'est pas publié. Le mot-clé USES indique que le module utilise la représentation, mais que si celle-ci n'est pas publié, il s'exécutera sans. Le mot-clé PROVIDES indique au autres module que cette représentation sera publié par ce module.

Si le module doit publier une représentation, celle-ci doit aussi être défini dans le dossier Src/Representations/. Tout module voulant envoyer ou recevoir cette représentation devront alors inclure sa définition comme suit:

```
#include "Representations/.../NomDeLeRepresentation.h"
```

Un exemple de la définition d'une représentation peut ressembler à cela:

```
STREAMABLE(NomDeLaRepresentation,
{,
   (float) (0.0) VariableX,
   (float) (0.0) AutreVariableY,
});
```

Le macro MAKE_MODULE permet alors d'utiliser toute représentation (inscrit ou fournit) à l'intérieur d'un module, par l'entremise d'une copie automatiquement créé avec le nom de la représentation précédé de "the". Un exemple d'une telle variable serait: theNomDeLaRepresentation.VariableX .

4.5 Simulateur

Cette partie du rapport présente en premier lieu un résumé des structures de code utiles au projet et elle présente le mode emploie pour utiliser le simulateur. Pour finir, elle détaille la conception et modification pour adapter l'environnement à l'apprentissage du dribble.

La partie du code du simulateur se trouve dans le dossier **Controller**. Le code qui gère la position des robots et le changement d'état du simulateur tel que «initial», «set» et «playing» se trouve dans **GameController.cpp**. Lorsque le programme roule, les fonctions se manifestent seulement lorsque l'on envoie une commande dans l'interface tel que «gc initial». Ces commandes sont perçues à travers le code de **RobotConsole.cpp**. Les constantes qui délimitent le terrain se trouvent dans le fichier de configuration **/Config/Location/Default/fieldDimensions.cfg** dont les valeurs sont en mm**.** À des fins pratiques, la figure suivante présente quelques principales constantes.

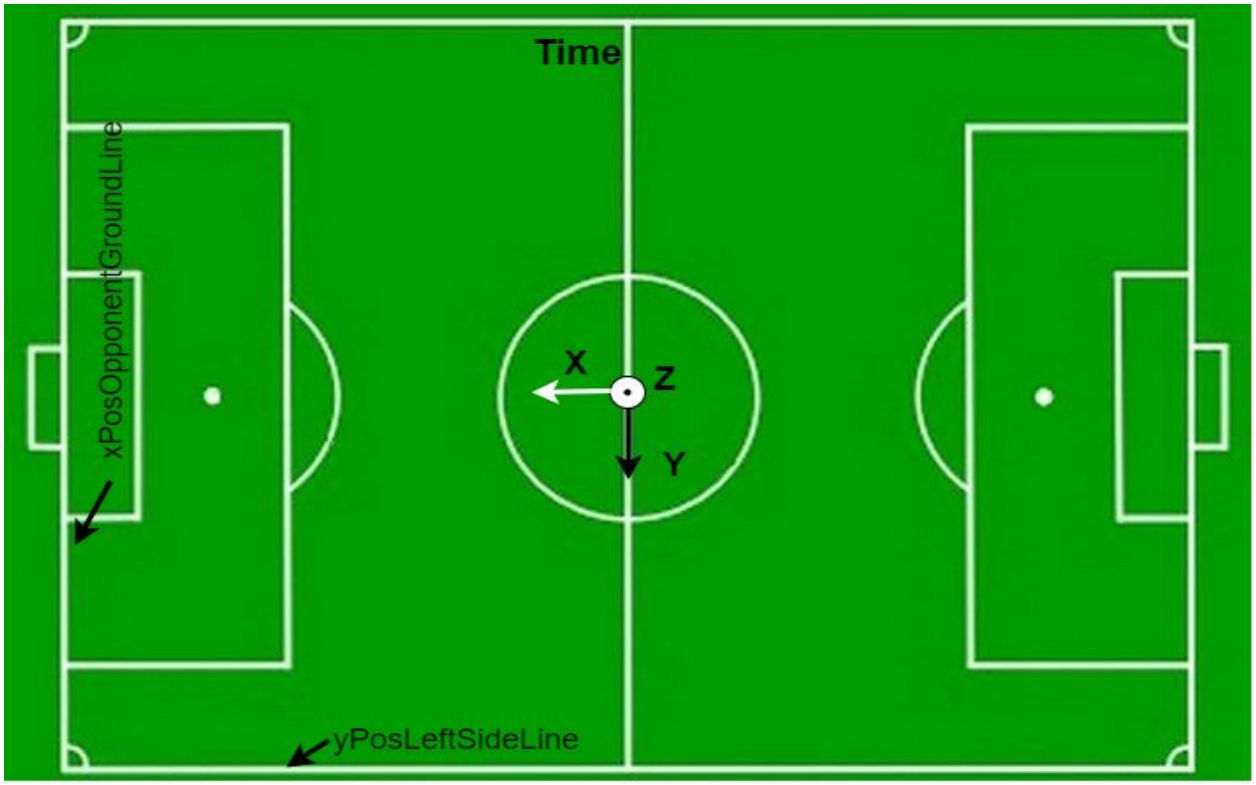


Figure 5: Terrain de soccer du simulateur NAO 3D.

Le simulateur utilisé pour le projet est un simulateur à trois dimensions adapté pour un match de soccer de robots Nao. Il permet donc d'avoir la meilleure estimation possible se rapprochant de la réalité. Comme décrit précédemment, la majorité du code se trouve dans le dossier **Controller**. Pour optimiser et automatiser l'apprentissage, l'objectif était de rendre l'environnement infaillible aux bogues et de permettre au robot d'être positionné aléatoirement lorsqu'il sort du terrain ou qu'il arrive à son objectif. Les constantes utilisées pour limiter les déplacements du joueur dans le terrain se retrouvent dans le fichier de config suivant : **/Config/Location/Default/fieldDimensions.cfg.** Pour éviter la création d'autre thread, la validation de la position du joueur se fait par l'intermédiaire de la fonction update du fichier Controller/**RobotConsole.cpp** et elle est modifié lors du non-respect de la position dans le terrain en mettant l'état à set. Le changement de cet état et la modification de la position a été réalisé dans le fichier Controller/**GameController.cpp.** Pour faciliter, la compréhension du code, les variables et les constantes créer pour le Qlearing sont écrites de la façon suivante : qLXxxxXxxx. Cet environnement permet au robot de rester dans le terrain tout au long de son apprentissage en plaçant à des positions aléatoires le ballon et le joueur lors des sortis en touche et se rapproche le plus possible de la réalité.

4.6 Mode d'emploi

1- Compiler le code en utilisant le terminal avec : /Make/<Platform>/make.

2- Ouvrir l'interface avec : Build/<Platform>/SimRobot/Develop/SimRobot.

3- Ouvrir le fichier .ros2: /Config/Scenes/XXX.ros2.

4- Écrire les commandes gc intial, gc set et gc playing dans la console.

5- Valider que le temps tourne et que le(s) joueur(s) bougent.

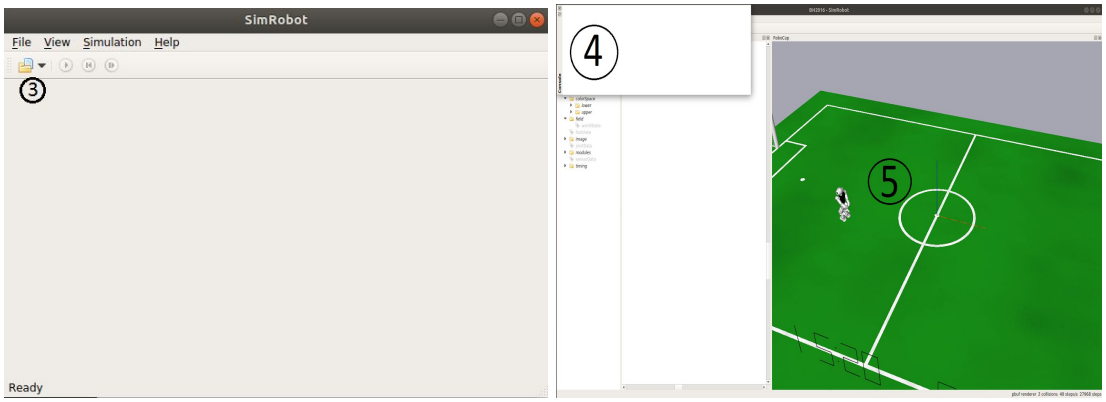


Figure 6 : Interface du simulateur

5. Résultats et discussion

Les deux objectifs principaux de ce projet étaient de mettre en place une architecture pour l'apprentissage, d'utiliser cette architecture et de l'adapter afin d'apprendre un premier comportement. De plus, le comportement doit permettre de plus grandes performances lors d'un match de soccer. Le premier objectif a été atteint. En effet, il est maintenant possible qu'un module apprenne un comportement et l'utilise en tant qu'état. Par exemple l'état de marcher vers le ballon. Ce comportement est appris grâce au Q-Learning approximé. S'il est plus efficace d'apprendre un nouveau comportement avec un autre algorithme, il faudrait commencer par l'implémenter. Par la suite, il serait facile d'utiliser le comportement. Pour ce qui est du deuxième objectif, il a été décidé d'apprendre à dribbler. Cet apprentissage n'est pas encore acquis par le robot. Les raisons de cette non-acquisition sont multiples et seront expliquées davantage par la suite. Pour la présentation des résultats, il sera d'abord question de présenter le projet de façon succincte et ensuite de discuter des erreurs et des solutions futures.

Le code se trouve dans le fichier zip avec ce présent rapport. Il se trouve également sur «gitlab» au lien suivant : https://gitlab.com/ClubNaova/NaovaCode. Pour accéder à ce code, demandez l'accès à un étudiant du club Naova de l'École de Technologie supérieure. Le système d'apprentissage par renforcement est en place et le simulateur est adapté à l'apprentissage du dribble.

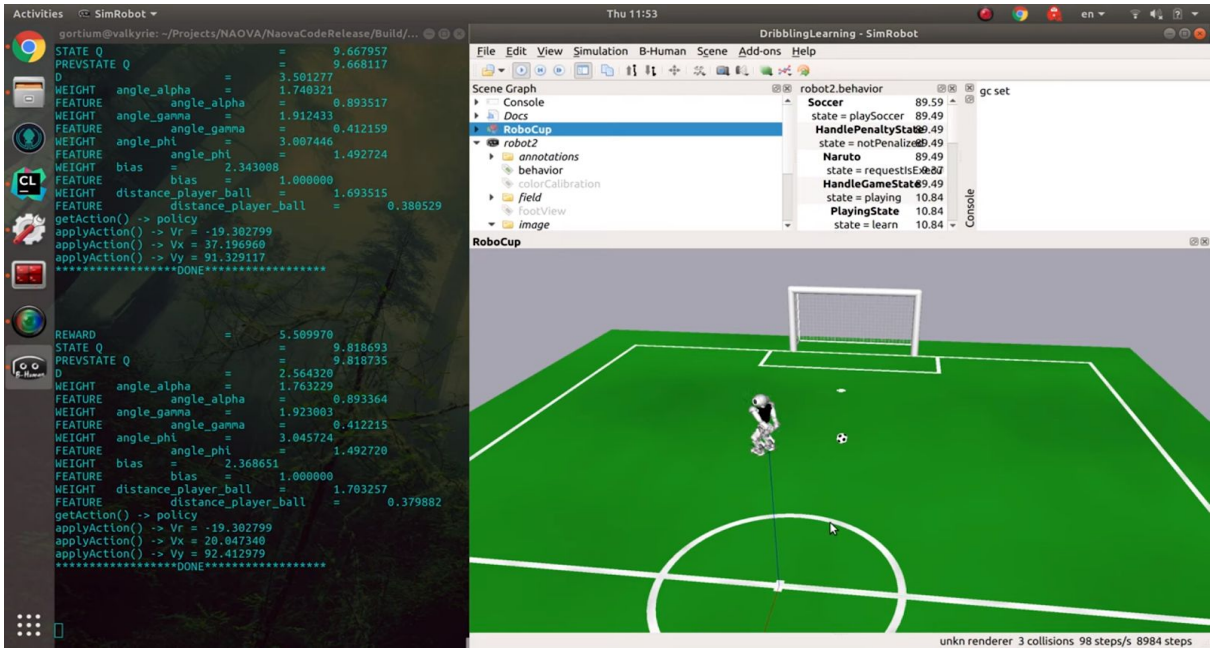


Figure 8: A gauche, les valeurs des features,des poids et des actions. A droite, la simulation du robot et du ballon sur le terrain.

Les premiers tests d'apprentissage du comportement de dribble ne convergent pas. Le robot n'apprend pas à se diriger vers le ballon. Il est incertain des raisons pour lesquelles cela ne fonctionnait pas. Il y a certaines pistes possibles. Cela est peut-être dû à l'entraînement qui était fait pendant 30 minutes ou il y avait trop de «features» pour avoir une corrélation linéaire.

Suite à nos essais non concluants, nous croyons qu'un algorithme avec plus de paramètres (avec plus de couches) aurait plus de chance de converger vers un bon comportement. Des algorithmes comme Actor-Critic ou Soft-Q-Learning seraient de bon candidat à être testé par la suite. Ceux-ci peuvent être décrits comme étant des algorithmes d'apprentissage profond. C'est-à-dire qu'il contiennent plus réseaux de neurones à couche multiple. Avec un plus grand nombre de couches, il est possible d'apprendre un comportement non linéaire. Ces algorithmes permettent aussi de ne pas avoir à écrire de feature, mais de laisser l'algorithme trouver les feature importantes et les poids associés par lui même. Les éléments manquant pour le futur seraient d'adapter la simulation pour permettre une communication entre le contrôleur, le robot et le module d'apprentissage.

Un ajout qui faciliterait l'apprentissage du robot est l'ajout d'un système permettant la modification des paramètres d'apprentissage (epsilone, discount, alpha). En modifiant ces paramètres en temps réel, il est possible de demander au robot de faire des actions plus aléatoires et augmente la vitesse d'apprentissage. En début d'apprentissage, il est recommandé d'avoir un grand epsilone, un grand discount et un grand alpha. Cela permet d'apprendre rapidement la base. Par la suite, il faut augmenter la valeur des paramètres pour peaufiner.

6. Conclusion

Ce présent rapport de fin de baccalauréat présente le projet de conception d'un processus d'apprentissage machine par renforcement appliqué à des robots bipèdes jouant au soccer. Cette conception est développée pour le club Naova afin d'améliorer les performances des robots pour leur permettre d'être compétitif lors de leur prochain tournois en 2019 se déroulant en Australie. Les principaux questionnements étaient de savoir quel algorithme d'apprentissage utilisé et quel environnement d'apprentissage il faut choisir. Après avoir étudié les aspects et les requis du problème tel que le domaine d'action, le temps d'implémentation, la vitesse d'apprentissage et l'automatisation du système, l'équipe choisit le **q-learning approximé** et le **simulateur 3D** réalisé par l'équipe allemande B-Human pour réaliser le projet. Les résultats attendus ne sont pas ceux espérés, car le robot n'arrive pas à apprendre dans les délais de ce projet et de ce fait les performances sont plus faible que ceux du code précédent. Malgrés cela, la structure est en place pour pouvoir essayer différents paramètres, d'ajouter d'autres couches de neurones et de modifier les features afin de trouver une solution performante dans un futur proche.

7. Bibliographie

C. Celemin et R. Perez et J. Ruiz-del-Solar et M. Veloso, “Interactive Machine Learning Applied to Dribble a Ball in Soccer with Biped Robots,” Electical Engineering, Universidad de Chile, Santiaga, Chile, 2016.

Dan Klein, Pieter Abbeel. *UC Berkeley CS188 Intro to AI -- Course Materials*. In University of California Berkeley., [En ligne]. http://ai.berkeley.edu/project_overview.html (Page consultée le 15 mars 2019)

"B-Human 2018." [En ligne]. https://www.b-human.de/downloads/publications/2018/CABSL.pdf. (Page consultée le 23 mars 2019)