Rapport de projet

Application du reinforcement learning à la voiture autonome

Paul FAROULT, Ba Thong NGUYEN
Tuteur : Sébastien LENGAGNE

Département IMDS
Polytech Clermont
France
2025 - 2026

Résumé

Ce projet repose sur l'application de l'apprentissage par renforcement appliqué à une voiture miniature. Le but du projet est de permettre à la voiture autonome miniature de déplacer des ballons de football dans des cages de but. Nous utilisons Gymnasium pour créer notre environnement personnalisé et l'algorithme PPO pour l'apprentissage. Enfin, à l'aide de notre tuteur, nous essayons d'implémenter le modèle appris à la voiture miniature.

Mots-clés

Voiture autonome, Apprentissage par renforcement, Récompense, Environnement, PPO, Gymnasium

Abstract

This project is an application of reinforcement learning applied to a miniaturized car. The project's goal is to make it possible for the car to score goals with footballs. Gymnasium is used to create a custom environment, and the PPO algorithm is used for training the car. Finally, with our tutor's advice, we try to implement the trained into the real vehicle.

Key words

Autonomous car, Reinforcement Learning , Reward , Environment, PPO, Gymnasium

Remerciements

Nous tenons à remercier Sébastien Lengagne pour son encadrement, sa bienveillance et sa pédagogie tout au long de ce projet. Cela a été une expérience très formatrice, et nous lui en sommes reconnaissants. Nous remercions aussi Julien Hautot pour les cours d'apprentissage par renforcement qui nous ont été dispensés, qui nous ont permis d'aborder ce projet avec sérénité.

Nous remercions aussi les étudiants en S3ER, Bathuan BULDUR et Achraf JDIDI, avec qui nous avons collaboré pour la bonne réalisation de ce projet.

Table des matières

Table des figures

Chapitre 1

Introduction

Cette section explique le contexte derrière la motivation de ce projet. De plus, il présente le problème de reinforcement learning traité.

1.1 Motivation

Le reinforcement learning, branche du machine learning récente, voit son application prendre place dans le domaine de la robotique.
En effet, il permet à des robots de s'adapter à des situations imprévues, leur confère des capacités de généralisation accrues, tout en réduisant le besoin de programmation "manuelle".

Les motivations derrière ce projet sont multiples : si le projet aboutit, cela représente une excellente activité lors des portes ouvertes de Polytech. De plus, notre tuteur, Sébastien LENGAGNE, nous a donné ce projet en préparation de son séminaire de reinforcement learning en Chine. Notre mission est alors de réaliser un travail préparatoire en amont de ce séminaire. Ce travail étant de nature prospective, il n'existe pas de cahier des charges bien défini. La finalité du projet est d'opposer notre travail à celui des étudiants S3ER,

Ainsi, dans le cadre de ce projet, nous appliquerons alors le reinforcement learning aux voitures autonomes. Ce sujet est alors parfaitement à mi-chemin entre intelligence artificielle et robotique. Ainsi, ce projet est mené en collaboration entre les départements S3ER et IMDS.

1.2 Présentation du problème

Le but du projet est de permettre à une voiture miniature autonome de pousser des ballons dans une cage de but, dans une arène de 3×3 mètres. Les figures suivantes, 1.1 et 1.2, illustrent la voiture miniature et l'arène dans laquelle le jeu prend place, respectivement.

FIGURE 1.1 : Voiture miniature utilisée.

FIGURE 1.2 : Arène d'affrontement des voitures.

Les ballons de football utilisés sont des ballons de football standard dégonflés. En effet, si les ballons sont gonflés, la trajectoire de la balle serait toujours une ligne droite. Alors, choisir des ballons dégonflés permet d'ajouter de la variance dans la trajectoire des balles.

Pendant le déroulement de ce projet, les étudiants S3ER chercheront alors à utiliser des algorithmes "traditionnels" pour permettre à la voiture de marquer des buts, tandis que nous (étudiants IMDS) utiliserons le reinforcement learning pour cette tache.

Chapitre 2

Reinforcement Learning

Dans cette section, nous précisions quelles librairies nous utilisons pour la réalisation du projet. De plus, nous détaillons le fonctionnement de l'algorithme que nous utiliserons tout au long du projet pour l'apprentissage. Nous avons acquis des compétences grâce à cette vidéo.

2.1 Contexte

La robotique a historiquement évolué autour de systèmes programmés de manière explicite, reposant sur des modèles cinématiques, dynamiques et des règles de contrôle soigneusement définies par des experts. Dans les années 1950–1960, les premiers robots industriels exécutaient des tâches répétitives dans des environnements hautement contrôlés, avec peu de capacité d'adaptation. Parallèlement, le reinforcement learning (apprentissage par renforcement) trouve ses racines dans les travaux en psychologie comportementale et en théorie du contrôle optimal, notamment avec les processus de décision markoviens formalisés dans les années 1950. Toutefois, ce n'est qu'à partir des années 1990, avec l'essor de l'informatique et des algorithmes d'apprentissage, comme *Proximal Policy Optimization*, que le reinforcement learning commence à être envisagé comme un outil pertinent pour la robotique.

L'importance croissante du reinforcement learning dans la robotique s'explique par différents facteurs. Tout d'abord, les robots sont de plus en plus amenés à évoluer dans des environnements complexes, dynamiques et partiellement inconnus, où la modélisation exacte des interactions physiques est difficile, voire impossible. Le reinforcement learning permet aux robots d'apprendre des politiques de contrôle directement à partir de l'expérience, en optimisant un comportement en fonction d'une récompense, sans nécessiter de modèle précis de l'environnement. Ensuite, les avancées récentes en deep learning ont permis de combiner réseaux de neurones profonds et reinforcement learning (deep reinforcement learning), rendant possible l'apprentissage de comportements complexes à partir de données comme les images ou les signaux numériques. Enfin, l'augmentation de la puissance de calcul, l'utilisation de simulateurs réalistes ont considérablement réduit les coûts et les risques liés à l'apprentissage par essai-erreur sur des systèmes robotiques réels, étant donné que l'apprentissage s'effectue dans un environnement virtuel.

Aujourd'hui, le reinforcement learning est appliqué à de nombreux domaines de la robotique. En manipulation, il est utilisé pour apprendre des tâches de préhension, d'assemblage ou de manipulation d'objets variés, comme le montrent les travaux sur les bras robotiques capables d'attraper des objets inconnus ou d'effectuer des tâches de dextérité fine. En locomotion, des

robots bipèdes, quadrupèdes ou humanoïdes apprennent à marcher, courir ou s'adapter à des terrains irréguliers grâce à des politiques apprises par renforcement. Le reinforcement learning est également employé en robotique mobile pour la navigation autonome, l'évitement d'obstacles et la coordination multi-robots.

2.2 Implémentation

Pour la programmation, la librairie Gymnasium sera utilisée. Son fonctionnement est décrit dans Towers u.a. (2024)

Gymnasium est une branche maintenue de la librairie Gym crée par OpenAI. Nous utiliserons son API pour définir notre propre environnement, qui sera adapté au problème que nous avons précisé dans 1.2.

De plus, la librairie StableBaselines sera utilisé pour l'apprentissage par renforcement dans l'environnement que nous avons crée. L'algorithme en particulier que nous utilisons est explicité dans Raffin u.a. (2021).

2.2.1 Environnement

Un environnement gym est une classe Python qui hérite d'un environnement générique appelé gym.Env.
Cette classe nécessite plusieurs méthodes pour fonctionner et être compatible avec les algorithmes d'apprentissage donnés par StableBaselines.

Les principales méthodes associés à un environnement sont les méthodes pour obtenir les observations et l'action des agents. Il est possible de créer des environnements personnalisés, adaptés à la tache de reinforcement learning souhaitée.

2.3 Algorithme PPO

Cette section a pour but de donner une compréhension de prime abord de l'algorithme PPO, développé par Schulman u.a. (2017).
Nous utilisons l'algorithme PPO dans notre projet, car cet algorithme est à ce jour, état de l'art en reinforcement learning, alliant performances et stabilités.

L'algorithme de PPO est le suivant :

Algorithm 1 Algorithme PPO.

for iteration = 1,2,... **do**
  for acteur = 1,2,...N **do**
    Exploiter l'ancienne politique $\pi_{\theta_{old}}$ pour T pas de temps. ▷ Collecte de la trajectoire.
    Calculer les avantages $\hat{A}_1, ..., \hat{A}_T$.
  end for
  Minimiser la fonction objectif de substitut L^{CLIP} par rapport à θ, par minibatchs de taille $M \leq NT$.
  $\theta_{old} \leftarrow \theta$
end for

Les sous-sections suivantes passent en revue les points que nous avons jugé clé pour la compréhension de l'algorithme.

2.3.1 Acteur-Critique

La **politique** est définie par un acteur. Cet acteur est un réseau de neurones, avec pour sortie μ et $log(\sigma)$. Ces paramètres calculés régissent la distribution de probabilité des actions prises, qui suivent une densité gaussienne de paramètres μ et σ.

Autrement dit :

$$p(a_i|s) = \mathcal{N}(\mu(s), \sigma(s)) \tag{2.1}$$

Ainsi, en supposant que les $(a_i|s)_i$ sont indépendantes et identiquement distribuées, un calcul de vraisemblance de routine donne :

$$\begin{aligned} log(p_\theta((a_i)_i|s)) &= log(p_\theta((a_i)_i|s)) \\ &= log(p_\theta(a_1|s)...p_\theta(a_n|s)) \\ &= log(\prod_{i=1}^{n} p_\theta(a_i|s)) \\ &= \sum_{i=1}^{n} log(p_\theta(a_i|s)) \end{aligned} \tag{2.2}$$

Avec θ les paramètres du réseau de neurones de l'acteur.

Cette quantité nécessite d'être calculée dans la partie de l'algorithme explicitée en 2.3.3
Un second réseau de neurones, appelé **Critique**, permet alors de juger l'action prise par la politique de l'acteur.

2.3.2 Avantages

Les avantages ont pour principal objectif de rendre le gradient de la fonction de cout moins difficile à calculer. Leur première apparition remonte à 2015, par Schulman u.a. (2018).

D'après labmlai (2022) et Schulman u.a. (2018) ,les avantages $\hat{A}_t, t \in [\![0; T]\!]$ sont calculés par :

$$\hat{A}_t = \delta_t + \gamma\lambda\delta_{t+1} + ... + (\gamma\lambda)^{T-t+1}\delta_{T-1}$$

Où :

- γ_t est la TD erreur définie par $\gamma_t = r_t + \lambda V(s_t) - V(s)$,
- λ est un hyperparamètre défini entre 0 et 1,
- r_t la récompense au temps t,
- $V(s)$ la valeur de l'état s.

En écrivant :

$$\gamma\lambda\hat{A_{t+1}} = \gamma\lambda\delta_{t+1} + (\gamma\lambda)^2\delta_{t+1} + ... + (\gamma\lambda)^{T-t}\delta_{T-1}$$

On s'aperçoit que :

$$\hat{A}_t = \delta_t + \gamma\lambda\hat{A_{t+1}}$$

C'est cette formule que l'on utilisera pour calculer les avantages. D'un point de vue programmation, une boucle sur la liste renversée des récompenses sera effectuée.

Ainsi, ces avantages interviennent dans le calcul de la fonction surrogate explicitée dans 2.3.3.

2.3.3 Fonction de coût : Clipped Surrogate Objective

Insister sur l'importance du choix de la reward
La fonction de cout utilisée pour l'algorithme PPO est la suivante :

$$L^{CLIP}(\theta) = \mathbb{E}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t]$$

Où :

- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$,
- ε est un hyperparamètre défini entre 0 et 1,
- $\hat{A}_t$ l'avantage au temps t.

L'idée derrière cette fonction de coût est éviter une variance trop élevée qui entraverait l'apprentissage. C'est pour cela que l'on borne entre $1-\varepsilon$ et $1+\varepsilon$.
L'intuition derrière les avantages est alors la suivante : plus l'avantage est grand (l'action prise est globalement bonne), alors on met plus à jour la politique par rapport à quand l'avantage est petit. En effet, les deux termes présents dans le minimum sont pondérés par $\hat{A}_t$.

2.3.4 Hyperparamètres de l'algorithme

L'algorithme PPO dépend de plusieurs hyperparamètres déterminants pour la qualité et la stabilité de l'apprentissage. Leur réglage influence la vitesse de convergence, la robustesse de la politique et la capacité de l'agent à généraliser. Les hyperamètres que nous avons précisés sont les suivants :

Taux d'apprentissage (`learning_rate`) : contrôle l'amplitude des mises à jour des réseaux. Un taux trop élevé peut provoquer instabilité ou divergence, tandis qu'un taux trop faible ralentit fortement l'apprentissage.

Nombre de pas collectés (`n_steps`) : définit combien de pas de temps sont utilisés pour estimer les avantages avant chaque mise à jour. Des valeurs trop faibles produisent des gradients bruités ; des valeurs trop grandes augmentent la corrélation entre échantillons et le coût computationnel.

Taille des minibatchs (`batch_size`) : fixe le nombre d'échantillons utilisés à chaque étape d'optimisation. De petits batches introduisent plus de bruit dans les gradients, favorisant l'exploration mais réduisant la stabilité. De grands batches lissent les gradients mais ralentissent parfois l'apprentissage.

Nombre d'époques (`n_epochs`) : indique combien de passages sont effectués sur les mêmes données collectées. Plus d'époques permettent d'exploiter davantage les trajectoires, mais augmentent le risque de surajustement.

Facteur d'actualisation (γ) : pondère l'importance des récompenses futures par rapport aux récompenses immédiates. Proche de 1 $\rightarrow$ stratégies à long terme ; plus faible $\rightarrow$ court terme.

Paramètre GAE (`gae_lambda`) : régule le compromis biais/variance dans le calcul des avantages. Des valeurs élevées réduisent le biais mais augmentent la variance, tandis que des valeurs faibles produisent des estimations plus stables mais légèrement biaisées.

Coefficient d'entropie (`ent_coef`) : encourage l'exploration en pénalisant les politiques trop déterministes. Trop élevé $\rightarrow$ exploration excessive et convergence lente ; trop faible $\rightarrow$ convergence prématurée vers une politique sous-optimale.

Coefficient du critique (`vf_coef`) : pondère l'importance de l'erreur du critique dans la fonction de coût globale. Un équilibre correct entre acteur et critique est crucial pour guider efficacement l'apprentissage.

Norme maximale du gradient (`max_grad_norm`) : limite l'amplitude des gradients lors de la rétropropagation. Ce mécanisme de *gradient clipping* empêche des mises à jour trop brutales et stabilise l'apprentissage, en particulier lorsque les récompenses ou les avantages ont une forte variance.

Chapitre 3

Travail effectué

Dans cette section, nous présentons le travail effectué, en commençant par l'organisation de celui-ci. Ensuite, une présentation chronologique des résultats sera effectuée.

3.1 Organisation du projet

3.1.1 WBS : Work Breakdown Structure

Cette sous-section présente le WBS du projet. Il comporte quatre grandes étapes :

- Apprendre le Reinforcement Learning.
- Créer un environnement adapté pour la tâche.
- Entrainer l'agent pour accomplir son but.
- Déployer le modèle appris à la voiture.

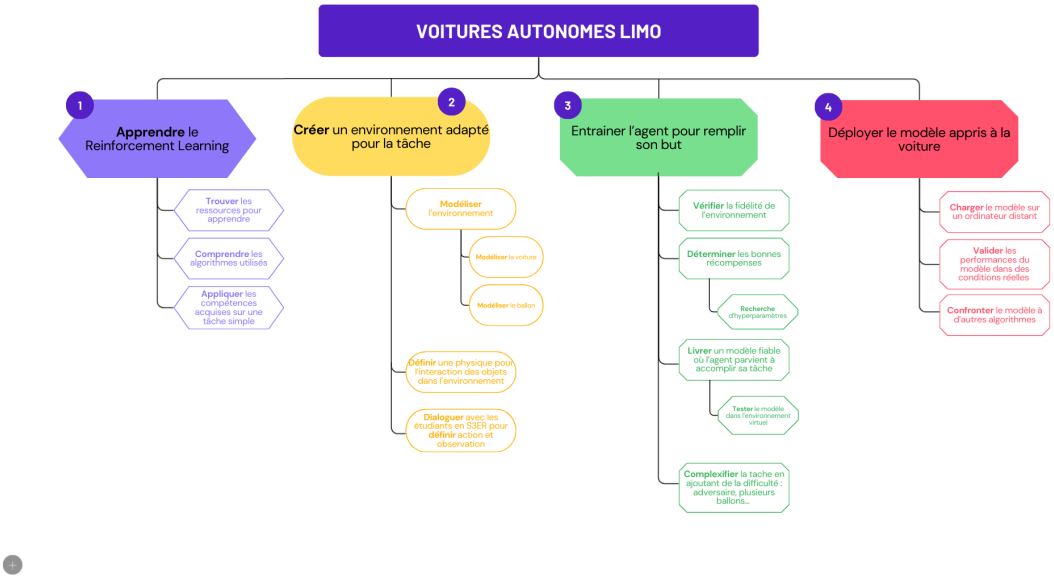

FIGURE 3.1 : Work Breakdown Structure du projet.

3.1.2 Diagramme de Gantt

Cette sous-section présente le diagramme de Gantt du projet. La figure 3.2 représente le diagramme de Gantt *prévu* du projet, tandis que la figure 3.3 décrit le déroulement *réel* du travail.

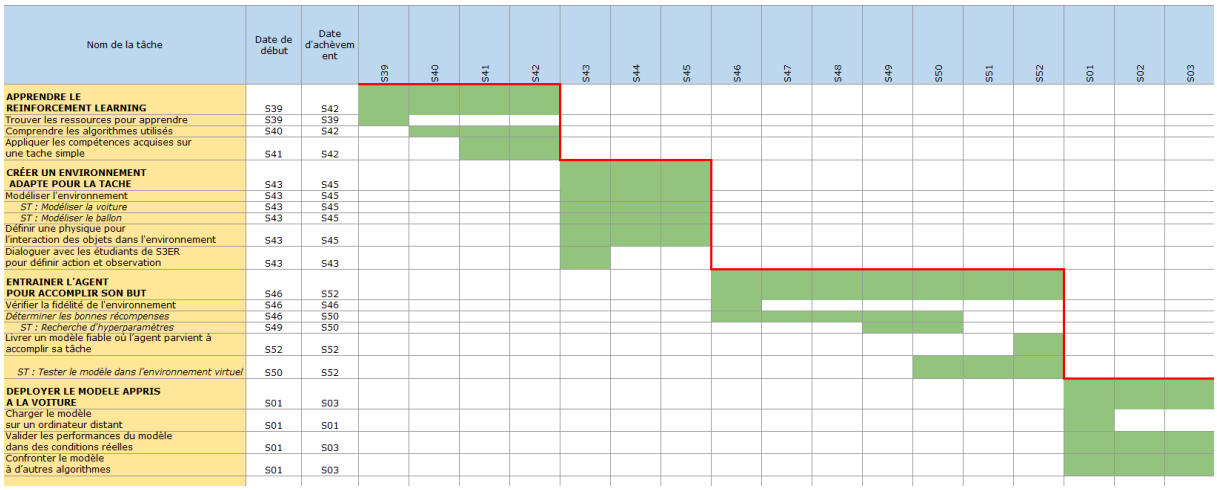

FIGURE 3.2 : Diagramme de Gantt *prévu* du projet.

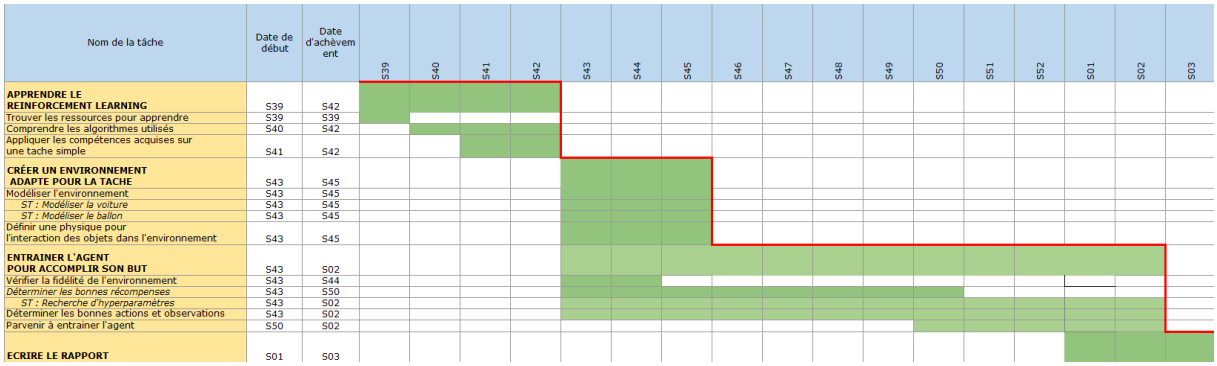

FIGURE 3.3 : Diagramme de Gantt *réel* du projet.

La différence capitale entre le projet planifié et effectué réside dans le temps de développement d'un agent fonctionnel. De plus, nous n'avons pas eu le temps de déployer le modèle appris à la voiture physique, hors de l'environnement virtuel.

3.2 Environnement

Cette section concerne l'environnement que nous utilisons pour l'apprentissage. Il est alors d'une importance capitale que l'environnement virtuel reflète avec la plus grande fidélité notre environnement réel.

3.2.1 Présentation du problème

La figure ci-dessous illustre l'environnement tel qu'il est affiché à l'utilisateur. On y trouve la voiture (rectangle bleu), avec un phare pour marquer l'avant de la voiture(cercle jaune inscrit

dans le rectangle bleu). Enfin, la balle est un cercle rouge. Quant aux cages, ce sont les parois vertes et rouges au centre des cotés latéraux du carré. La cage verte est la cage où il faut insérer le ballon. On trouve, en haut à gauche de l'écran, le vecteur des observations actuel ainsi que le pas de la simulation.

FIGURE 3.4 : Environnement développé.

3.2.2 Description de l'espace d'observation

L'espace d'observation choisi est un vecteur à cinq composantes, représentant, dans l'ordre :

- la position horizontale de la voiture (axe x),
- la position verticale de la voiture (axe y),
- l'orientation de la voiture,
- la position horizontale de la balle (axe x),
- la position verticale de la balle (axe y).

Pendant l'entrainement, les positions sont comprises entre -1 et 1 et l'orientation de la voiture est comprise entre -3.14 et 3.14.

3.2.3 Description de l'espace d'action

L'espace d'action choisi est un vecteur à deux composantes, représentant dans l'ordre :

- la vitesse du véhicule,

- la rotation du véhicule.

Pendant l'entrainement, ces deux valeurs sus-mentionnées sont comprises entre -1 et 1. Elles sont ensuite utilisées pour calculer l'accélération et la vitesse angulaire du véhicule.

3.3 Premiers résultats et difficultés rencontrées

L'un des outils les plus importants en apprentissage par renforcement est l'analyse des courbes issues de TensorBoard. Il est donc essentiel de savoir interpréter correctement ces courbes. La courbe `ep_len_mean` représente la durée moyenne des épisodes en fonction du nombre de *timesteps*. Elle permet notamment d'identifier des arrêts prématurés d'épisodes, par exemple lorsqu'une condition terminale est atteinte avant la durée maximale. Comme montré en figure 3.5, la durée d'épisode diminue ce qui montre que l'épisode s'arrête de plus en plus vite.

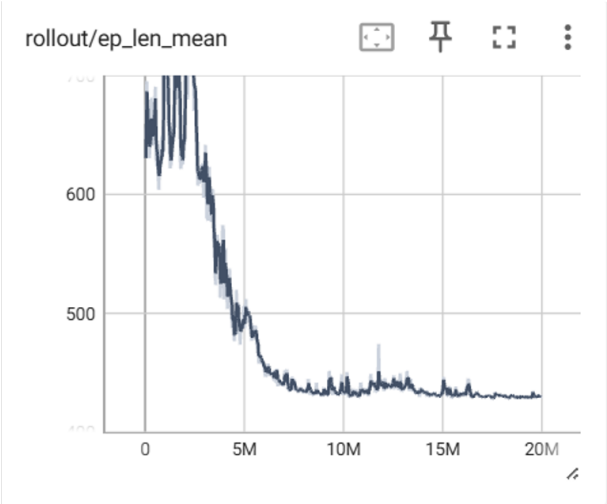

FIGURE 3.5 : Courbe tensorboard illustrative représentant la durée moyenne d'épisode.

La courbe `ep_rew_mean`, quant à elle, représente la récompense moyenne obtenue par l'agent en fonction du nombre de *timesteps*. Elle correspond à la courbe d'apprentissage principale et permet d'observer les phases de progression, de stagnation ou d'échec de l'agent. En Figure 3.6, on peut constater que la reward augmente puis stagne, ce qui montre la convergence de notre agent.

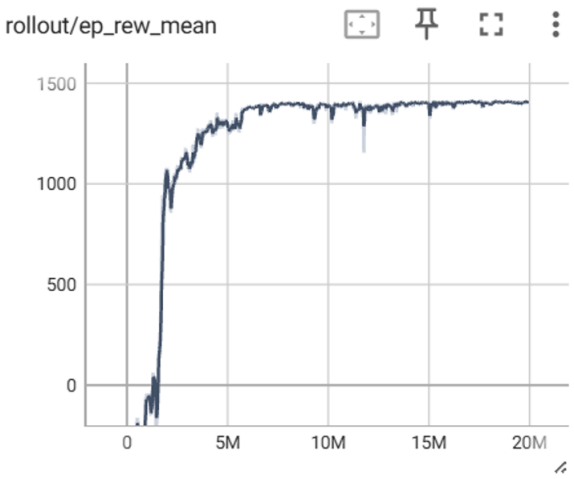

FIGURE 3.6 : Courbe tensorboard illustrative représentant la récompense moyenne par épisode.

Une fois ces éléments introduits, nous pouvons analyser les premiers résultats obtenus au cours du projet.

3.3.1 Mise en place de l'environnement et premières configurations

Les premiers résultats exploitables ont été obtenus le 28/11/2025. Dans cette configuration initiale, l'environnement contenait un seul robot et une balle, tous deux placés de manière fixe. Le robot démarrait systématiquement dans sa cage, orienté vers la balle, tandis que la balle était positionnée au centre du terrain.

À cette étape du projet, l'espace d'observation ne correspondait pas encore à celui présenté précédemment. Il était composé des informations issues du lidar, à savoir la distance à l'obstacle le plus proche à gauche, devant et à droite, ainsi que des positions (x, y) du robot et de la balle. La fonction de récompense était particulièrement complexe, intégrant un grand nombre de récompenses différentes. De plus, l'épisode se terminait immédiatement lorsqu'un but était marqué. L'intégralité du code correspondant est disponible sur GitHub, dans la branche :
`stage/but-cage-lidar`.

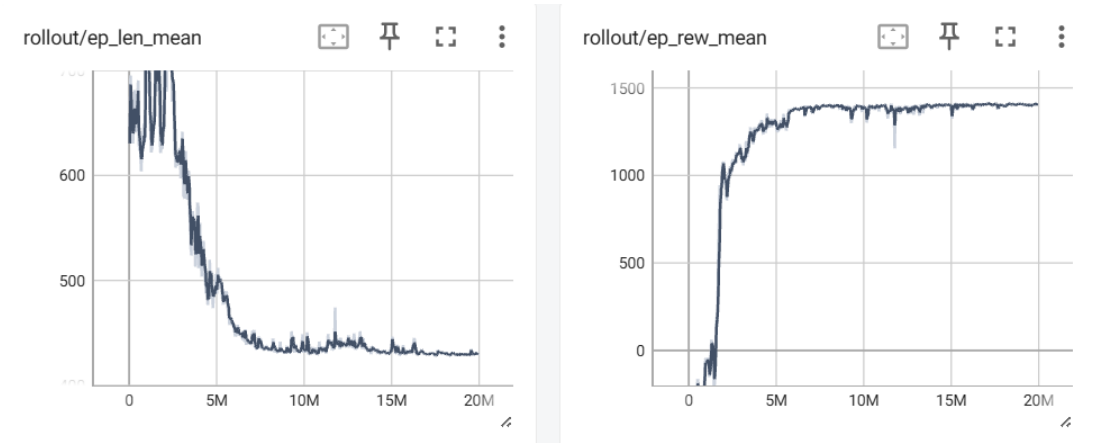

FIGURE 3.7 : Courbes TensorBoard pour la configuration initiale : durée moyenne des épisodes et récompense moyenne.

Les courbes TensorBoard présentées en Figure 3.7 montrent que les épisodes sont courts et s'arrêtent bien avant la durée maximale autorisée. Cela indique qu'un but est fréquemment marqué. La courbe de récompense moyenne montre une progression suivie d'une convergence, suggérant que l'agent apprend à marquer dans cette configuration très contrainte.
Cependant, cette configuration s'est révélée extrêmement fragile. En effet, dès que la balle était déplacée, même très légèrement, le modèle devenait incapable de marquer le moindre but. Cette incapacité à généraliser montre que l'agent avait surappris une trajectoire spécifique plutôt que d'apprendre une stratégie robuste.

Afin de résoudre ces limitations, plusieurs modifications majeures ont été apportées à l'environnement. L'espace d'observation a été redéfini afin de correspondre à celui présenté précédemment, et une normalisation systématique de l'ensemble des observations a été appliquée. La fonction de récompense a également été fortement simplifiée afin de ne conserver que le strict minimum nécessaire à l'apprentissage.

Le pas de simulation a été réduit à 10 images par seconde afin de limiter le nombre d'observations et d'actions échangées entre l'agent et l'environnement. Dans la configuration précédente, les observations successives étaient très similaires, le robot ne s'étant pas réellement déplacé entre deux décisions. Cette situation entraînait un remplissage du buffer avec des observations quasi identiques, ce qui nuisait à l'apprentissage.

Afin d'éviter tout surapprentissage lié à une configuration figée, le robot ainsi que la balle ont été placés de manière aléatoire sur le terrain au début de chaque épisode. Pour suivre l'évolution des performances de l'agent au cours de l'apprentissage, une nouvelle métrique TensorBoard, `goals_per_step`, a été ajoutée. Elle représente le nombre moyen de buts marqués par épisode en fonction du nombre de *timesteps*.

Malgré l'ensemble de ces ajustements nécessaires, le modèle ne parvenait toujours pas à marquer. Ce n'est que le 16/12/2025, avec l'aide de notre tuteur, que la source du problème a pu être identifiée. Deux erreurs majeures empêchaient l'apprentissage :

- l'espace d'observation n'avait pas été mis à jour après la normalisation, ce qui entraînait un écrêtage des valeurs négatives : toutes les valeurs appartenant à l'intervalle $[-1, 0]$ étaient ramenées à 0 ;
- l'orientation du robot n'était pas fournie dans l'observation : l'agent connaissait sa position mais ignorait la direction vers laquelle il était orienté.

Une fois ces erreurs corrigées, les résultats sont devenus concluants.

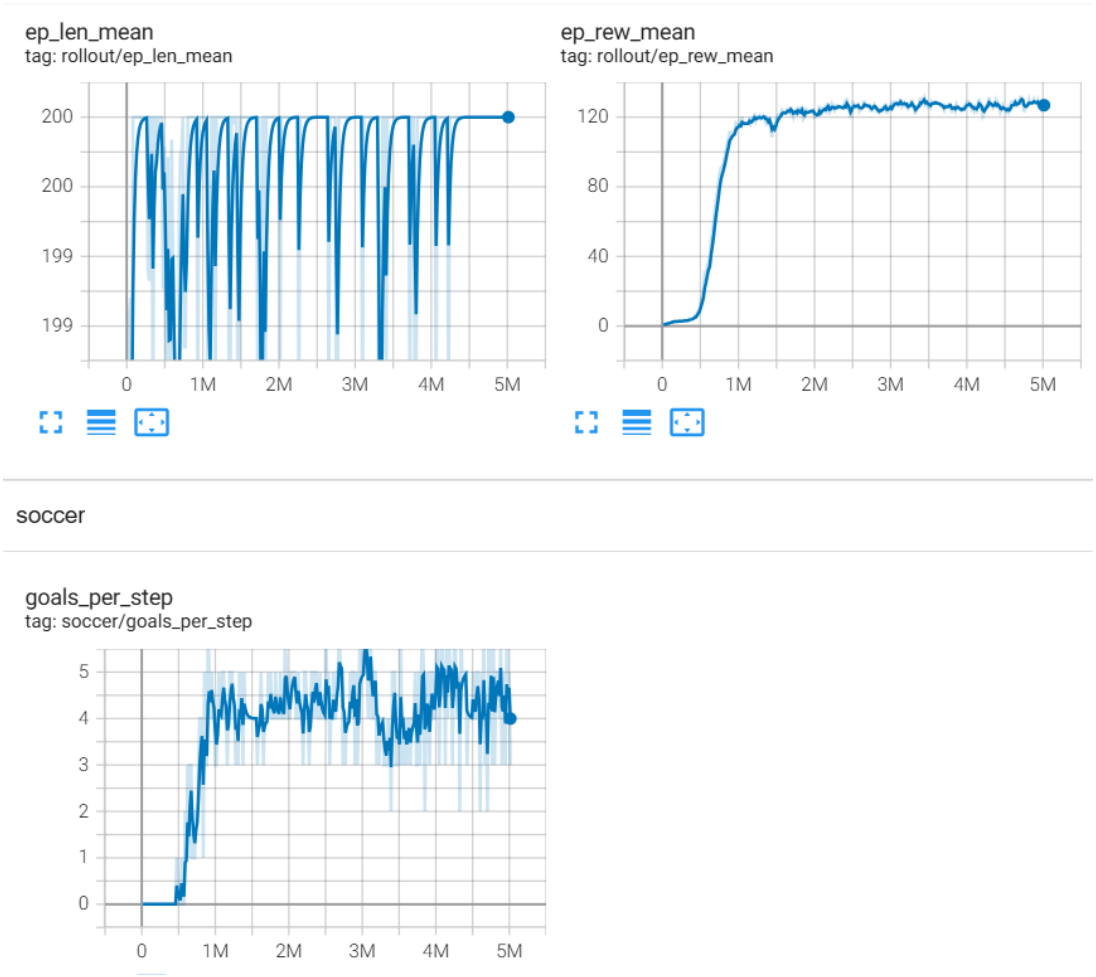

FIGURE 3.8 : Courbes TensorBoard après correction de l'espace d'observation.

Les courbes présentées en Figure 3.8 montrent que le robot apprend rapidement à marquer plusieurs buts par épisode et converge efficacement. Dans cette nouvelle configuration, l'épisode ne s'arrête plus lorsqu'un but est marqué ; la balle est replacée aléatoirement dans le carré central. Le code est disponible sur la branche : `stage/classic-rl`.

3.3.2 Analyse complémentaire

À partir de cet environnement, plusieurs études complémentaires ont été menées afin d'explorer différentes stratégies d'apprentissage. Toutefois, ces approches n'ont pas été retenues pour la suite du projet, les résultats obtenus ne montrant pas d'amélioration significative des performances.

Apprentissage hiérarchisé Dans un premier temps, une tentative d'apprentissage hiérarchisé a été implémentée dans le but de simplifier et d'accélérer la phase d'apprentissage. Le principe consistait à faire évoluer progressivement la difficulté de la tâche en modifiant la position initiale de la balle : celle-ci était d'abord placée juste devant le robot, puis progressivement éloignée, avant d'être finalement positionnée de manière totalement aléatoire au fil des *timesteps*.

Les courbes d'apprentissage associées à cette approche, issues de TensorBoard, sont présentées en Figure 3.9. Le code correspondant est disponible sur le dépôt GitHub du projet, dans la branche `experiment/hierarchical-rl`.

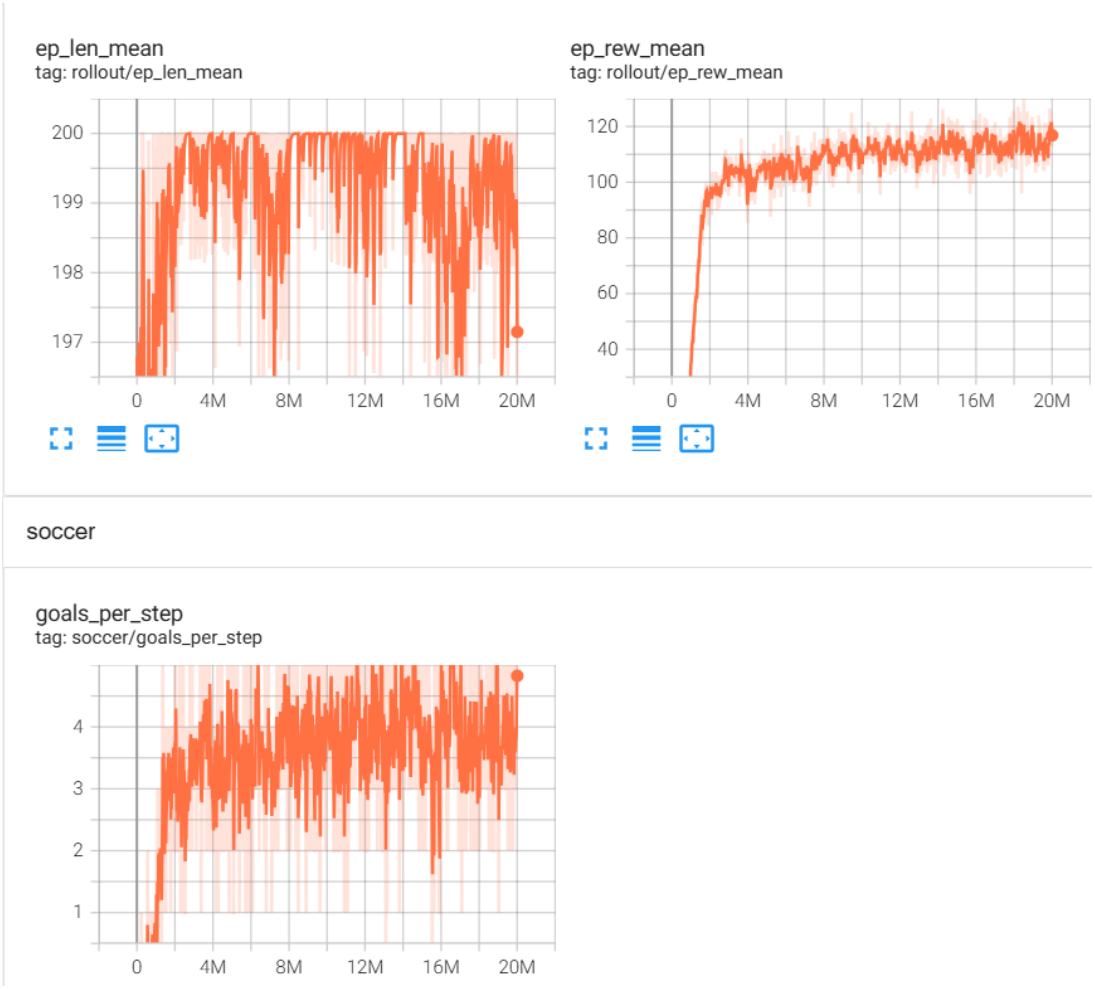

FIGURE 3.9 : Courbes TensorBoard obtenues lors de l'apprentissage hiérarchisé

Comme on peut l'observer à partir de ces courbes, cette stratégie n'apporte pas de gain notable en termes de convergence ou de performance finale. Par conséquent, cette approche n'a pas été conservée pour la suite du projet.

Influence du facteur d'actualisation γ Une autre étude a été réalisée en modifiant le facteur d'actualisation γ du modèle PPO. La branche `experiment/classic-gamma-0.995` reprend exactement la même architecture et le même environnement, mais avec une valeur de γ modifiée afin de favoriser la maximisation des récompenses à court terme.

Les résultats correspondants sont présentés en Figure 3.10.

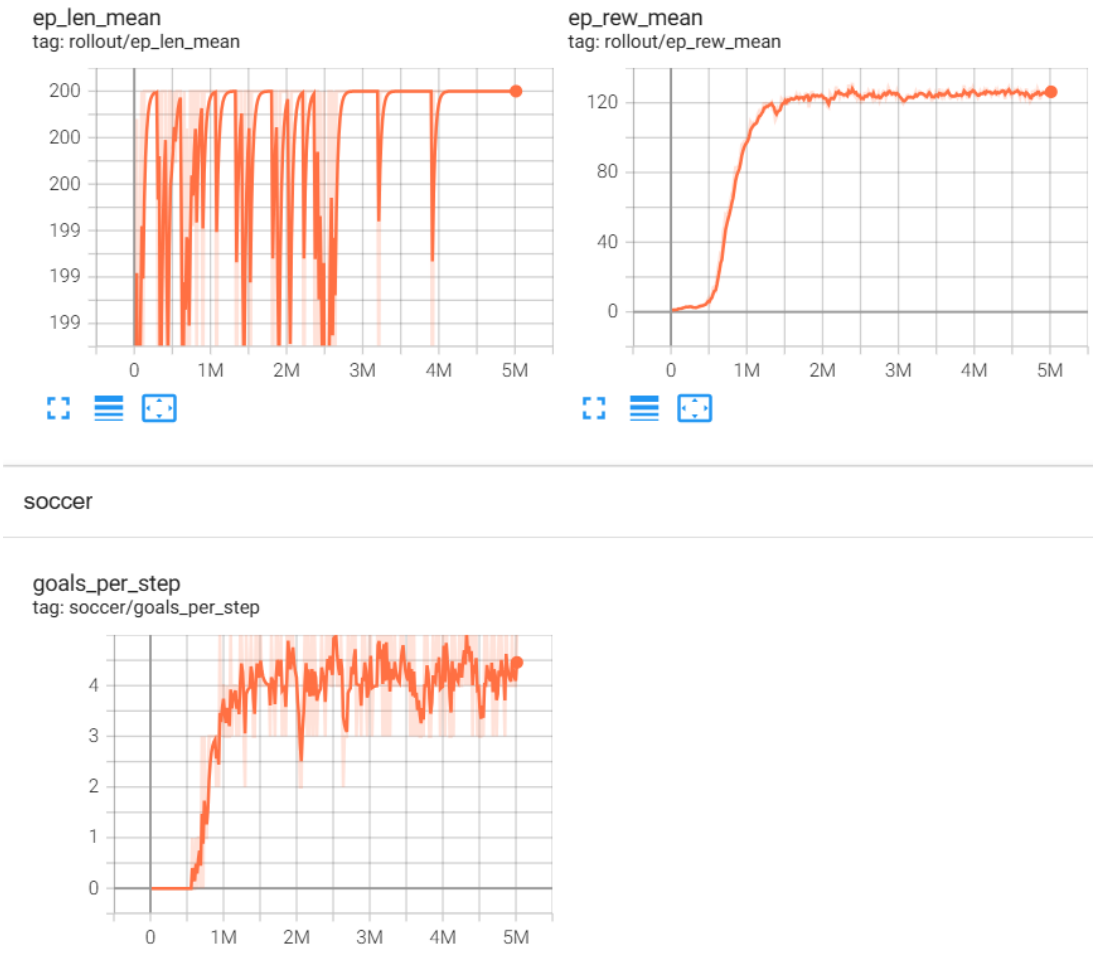

FIGURE 3.10 : Impact d'une valeur de $\gamma = 0.995$ sur la convergence de l'apprentissage

Les courbes montrent que cette modification entraîne une convergence légèrement plus lente du modèle, ce qui a conduit à l'abandon de cette configuration.

Analyse de la fonction de récompense

Enfin, une analyse approfondie de la fonction de récompense a été menée. Il est apparu que l'algorithme PPO nécessite une récompense suffisamment dense et continue afin de permettre un apprentissage efficace. Ce qui signifie qu'il faut fournir un signal informatif de manière fréquente, idéalement à chaque pas de temps. Lorsque seules des récompenses ponctuelles étaient attribuées, par exemple uniquement lors de l'atteinte d'une meilleure position jamais observée, l'agent n'apprenait pratiquement rien, les signaux de récompense étant noyés au sein de longues séquences de valeurs nulles.

PPO repose sur l'hypothèse qu'une même action, dans un état et une observation identiques, doit produire une récompense cohérente afin de permettre une optimisation stable de la politique. Ainsi, l'utilisation d'une récompense continue (calculée à chaque pas de temps), liée par exemple à la distance à l'objectif ou à la progression vers celui-ci, s'est révélée indispensable pour obtenir un apprentissage pertinent.

3.4 Affrontement en deux robots

Une fois des résultats concluants et stables obtenus avec un seul robot, un second robot a été introduit dans l'environnement afin d'étudier des situations d'affrontement. L'objectif est d'évaluer la capacité de l'agent à s'adapter à la présence d'un adversaire, d'abord passif, puis actif et capable de marquer. Avec l'ajout du second robot, notre environnement est celui présenté sur le figure 3.11. Le second robot est représenté en gris et l'espace d'observation a été modifié pour rajouter 2 valeurs :

- la position horizontale du robot adverse (axe x),
- la position verticale du robot adverse (axe y)

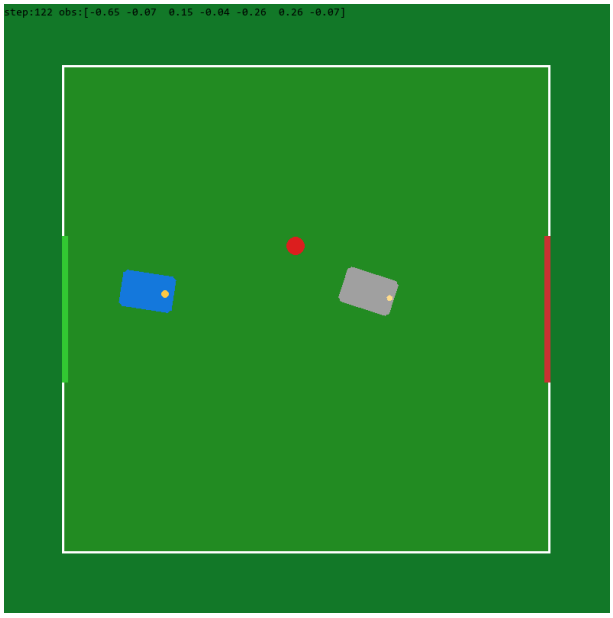

FIGURE 3.11 : Environnement avec deux robots.

3.4.1 Affrontement avec adversaire statique

Dans un premier temps, un second robot statique a été implémenté. Ce robot adverse apparaît sur le terrain sans se déplacer. L'espace d'observation a donc été enrichi afin d'inclure la position (x, y) de ce second robot.

Dans une première configuration, le robot adverse apparaissait de manière totalement aléatoire sur le terrain. Cependant, dans cette situation, l'agent ne prenait pas réellement en compte la présence de l'adversaire. En effet, dans une grande majorité des simulations, le robot adverse ne se trouvait pas sur la trajectoire empruntée par l'agent pour marquer. Il n'avait donc aucun impact sur la trajectoire suivie par l'agent. En conséquence, lorsque le robot adverse apparaissait effectivement sur cette trajectoire, l'agent ne savait pas comment réagir.

Afin de résoudre ce problème, il a été décidé de placer le robot adverse directement sur la trajectoire. Pour cela, le robot adverse est positionné soit entre le robot et la balle, soit entre la balle et la cage. Un décalage latéral aléatoire est appliqué afin d'éviter une configuration trop déterministe.

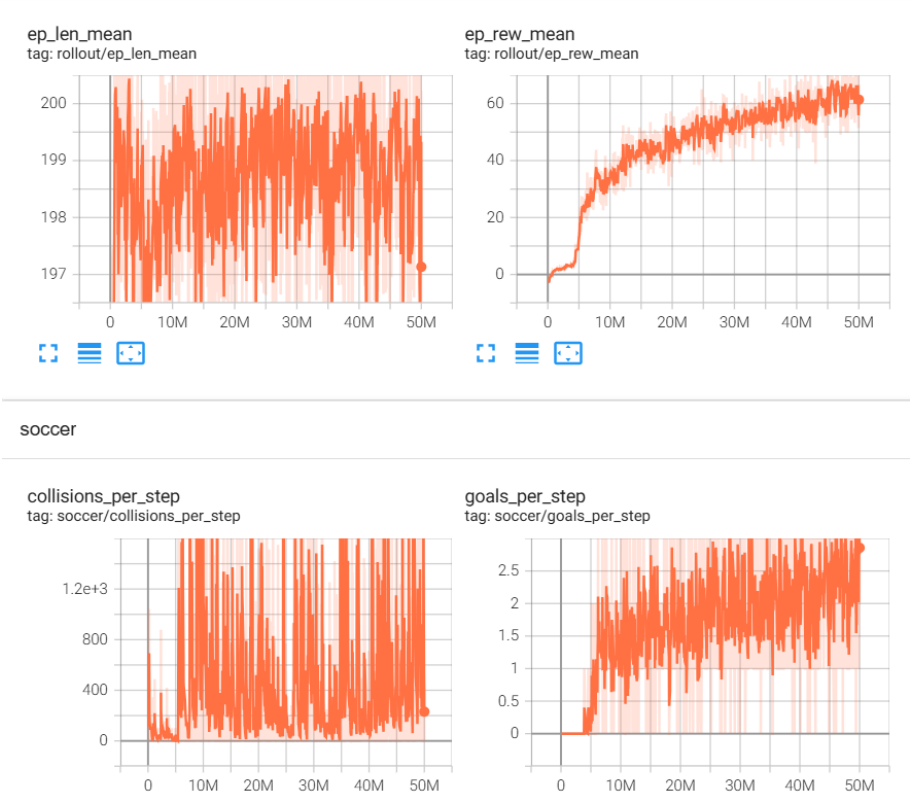

FIGURE 3.12 : Courbes TensorBoard pour l'apprentissage avec adversaire statique : récompense, buts et collisions.

Une récompense spécifique a été ajoutée afin d'inciter l'agent à éviter les collisions et à contourner l'obstacle. De plus, une nouvelle métrique TensorBoard, `collisions_per_step`, a été introduite. Elle correspond au nombre moyen de collisions par épisode.

Comme on peut l'observer en figure 3.12, le modèle converge rapidement. Il parvient à marquer efficacement tout en limitant le nombre de collisions avec l'obstacle. Le code correspondant à cette partie est disponible sur GitHub, dans la branche : `stage/static-opponent`.

3.4.2 Affrontement avec adversaire dynamique

Une fois l'étape de l'adversaire statique validée, un affrontement contre un adversaire mobile et capable de marquer a été mis en place. Pour cela, le second robot utilise un modèle précédemment entraîné contre l'adversaire statique. Les observations et les actions sont inversées afin que ce robot cherche à marquer dans la cage opposée.

Dans un premier temps, les poids de ce modèle adverse sont figés, tandis que seul le modèle contrôlant notre agent continue à être entraîné. Une fois cet entraînement terminé, le nouveau modèle obtenu est chargé dans le robot adverse, et le processus est répété. Cette approche permet d'entraîner progressivement l'agent contre des adversaires de plus en plus performants.

Ces différentes étapes sont disponibles sur GitHub dans les branches :

- `experiment/duel-vs-model-static-reward`,
- `experiment/duel-reward`.

Cependant, cette approche a conduit à l'apparition d'un comportement non souhaité. Dès que l'adversaire s'approchait trop, notre agent abandonnait systématiquement l'action offensive

afin d'éviter à tout prix la collision. Ce comportement rendait l'agent non compétitif, puisqu'il renonçait à marquer même lorsque cela était nécessaire.

Pour corriger ce problème, les récompenses liées aux collisions ont été supprimées. Les collisions provoquant naturellement un ralentissement important, voire un arrêt du robot, il n'était pas nécessaire d'ajouter une pénalisation explicite. Même sans récompense spécifique, l'agent apprend à éviter son adversaire de manière naturelle, sauf lorsque la collision est nécessaire, par exemple pour récupérer la balle.

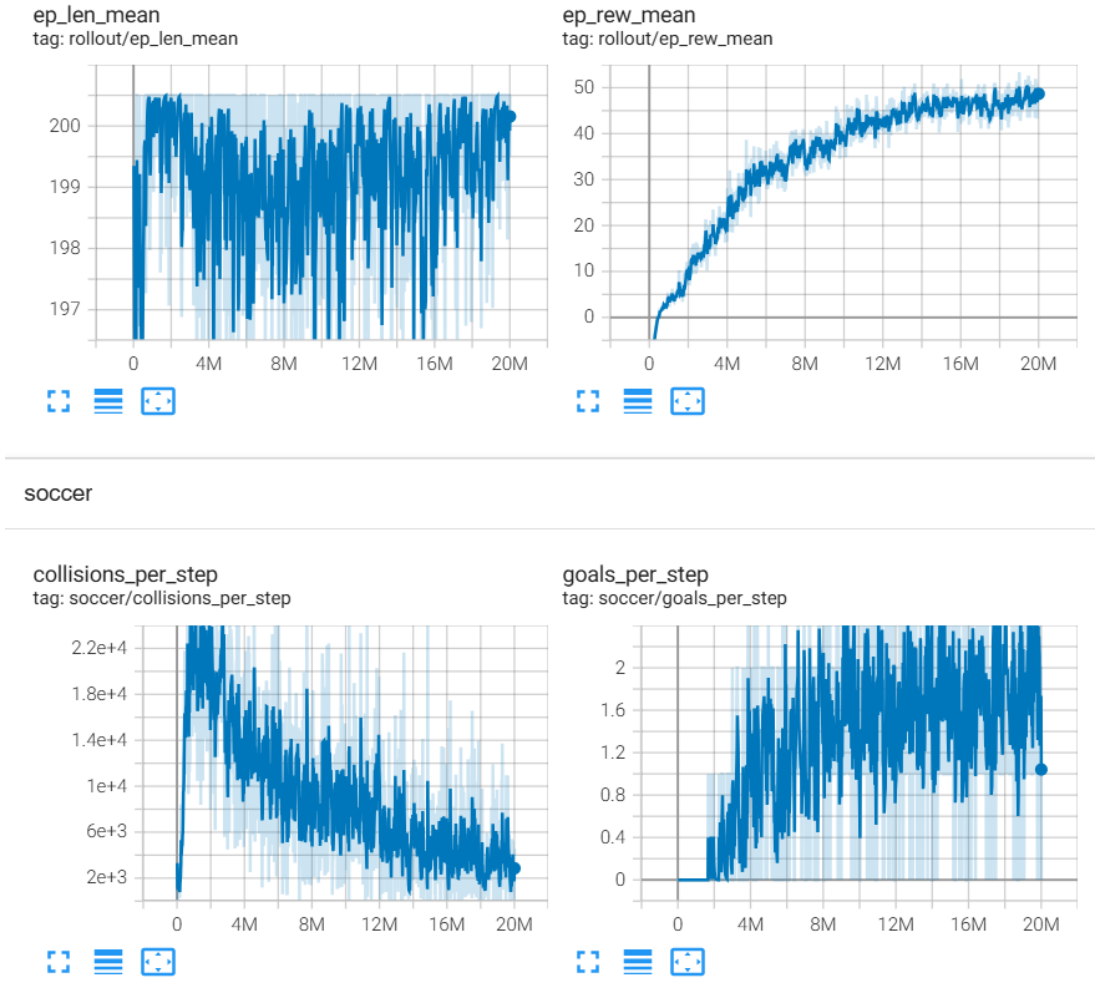

FIGURE 3.13 : Évolution de la récompense, du nombre de buts et des collisions lors d'un duel contre un adversaire statique entraîné.

Comme illustré en Figure 3.13, la récompense moyenne et le nombre de buts augmentent au cours de l'apprentissage, tandis que le nombre de collisions diminue. Le code correspondant est disponible sur la branche : `stage/duel-sans-reward-collisions-static`.

3.4.3 Évaluation et analyse complémentaire

Lorsque le modèle entraîné contre l'adversaire statique est chargé dans le robot adverse, l'agent est ensuite entraîné contre ce nouvel adversaire plus performant. Les résultats correspondants sont présentés en Figure 3.14.

FIGURE 3.14 : Apprentissage contre un adversaire dynamique performant : récompense, buts et collisions.

On observe que la récompense moyenne ainsi que le nombre de buts augmentent au cours de l'apprentissage. En revanche, le nombre de collisions ne diminue pas. Cela s'explique par le fait que l'adversaire est déjà très performant. Dans ce contexte, les collisions deviennent nécessaires afin de récupérer la balle ou d'empêcher l'adversaire de marquer.

Un comportement particulièrement intéressant a été observé lors des tests. L'épisode ayant une durée fixe et la balle étant replacée aléatoirement dans le carré central après chaque but, l'agent adopte une stratégie d'anticipation. Lorsque l'adversaire est proche de sa cage et qu'un but semble inévitable, l'agent préfère attendre au centre du terrain plutôt que de poursuivre l'adversaire. Cette stratégie maximise la probabilité de récupérer la balle en premier lors de sa réapparition et illustre parfaitement le principe fondamental de l'apprentissage par renforcement : la maximisation de la récompense cumulée.

L'ensemble du code correspondant à cette partie est disponible sur GitHub, dans la branche : `stage/duel-sans-reward-collisions`.

3.4.4 Évaluation comparative des modèles

Lors des étapes précédentes, à chaque changement de modèle pour l'adversaire, notre agent était réentraîné depuis zéro. Nous avons donc décidé de charger le même modèle pour les deux robots, puis de poursuivre l'entraînement uniquement sur notre agent afin d'évaluer l'impact d'un temps d'apprentissage supplémentaire.

Une fois cet entraînement terminé, les deux modèles ont été opposés afin d'évaluer leurs performances respectives. Le code de cette partie est disponible sur la branche `stage/training`. La Figure 3.15 présente le nombre moyen de buts marqués et le nombre moyen de buts concédés par match pour le modèle de base et le modèle affiné.

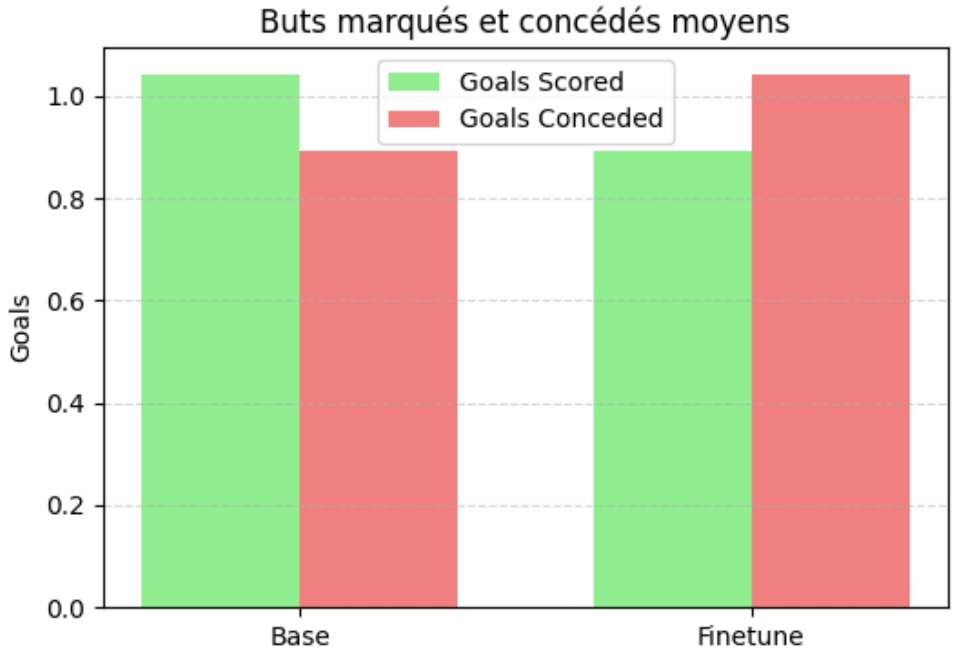

FIGURE 3.15 : Comparaison des performances entre le modèle de base et le modèle affiné.

De manière surprenante, le modèle de base obtient de meilleures performances. Son taux de victoire atteint 56.99%, contre 43.01% pour le modèle affiné. On aurait pu s'attendre à ce que le modèle affiné soit plus performant, ou au minimum à un équilibre proche de 50–50.

Afin de mieux comprendre ces résultats, une expérience supplémentaire a été menée. L'environnement a été légèrement modifié afin que les épisodes ne se terminent plus par une limite de temps, mais par un nombre total de buts. Pour éviter des simulations trop longues, l'épisode est interrompu lorsqu'un total de cinq buts est atteint. Cette expérience est disponible sur la branche : `experiment/no_time_limit`.

Les résultats en figure 3.16 montrent, une nouvelle fois, que le modèle de base reste plus performant. Nous avons alors émis l'hypothèse d'un biais potentiel dans la méthode d'évaluation. Pour la vérifier, la même analyse a été réalisée en opposant deux instances strictement identiques du même modèle.

Dans ce cas, les résultats obtenus sont proches de l'équilibre attendu : un taux de victoire de 44.46% pour le premier modèle contre 46.14% pour le second, avec un nombre moyen de buts très similaire. Les légères différences observées s'expliquent probablement par le nombre limité de matchs simulés.

Ces résultats confirment que la méthode d'évaluation n'est pas biaisée et que le modèle de base est effectivement plus performant. Plusieurs explications peuvent être avancées. La position initiale aléatoire des robots et de la balle favorise parfois mécaniquement l'un des agents. De plus, aucune *seed* n'a été fixée, ni pour l'environnement ni pour l'apprentissage, ce qui implique que deux entraînements successifs ne produisent pas exactement les mêmes performances. Il est donc possible que le modèle de base corresponde à une configuration particulièrement optimale.

Dans la branche `main` du dépôt GitHub, les points clés du projet sont récapitulés, accompagnés des instructions nécessaires à l'exécution des différents programmes.

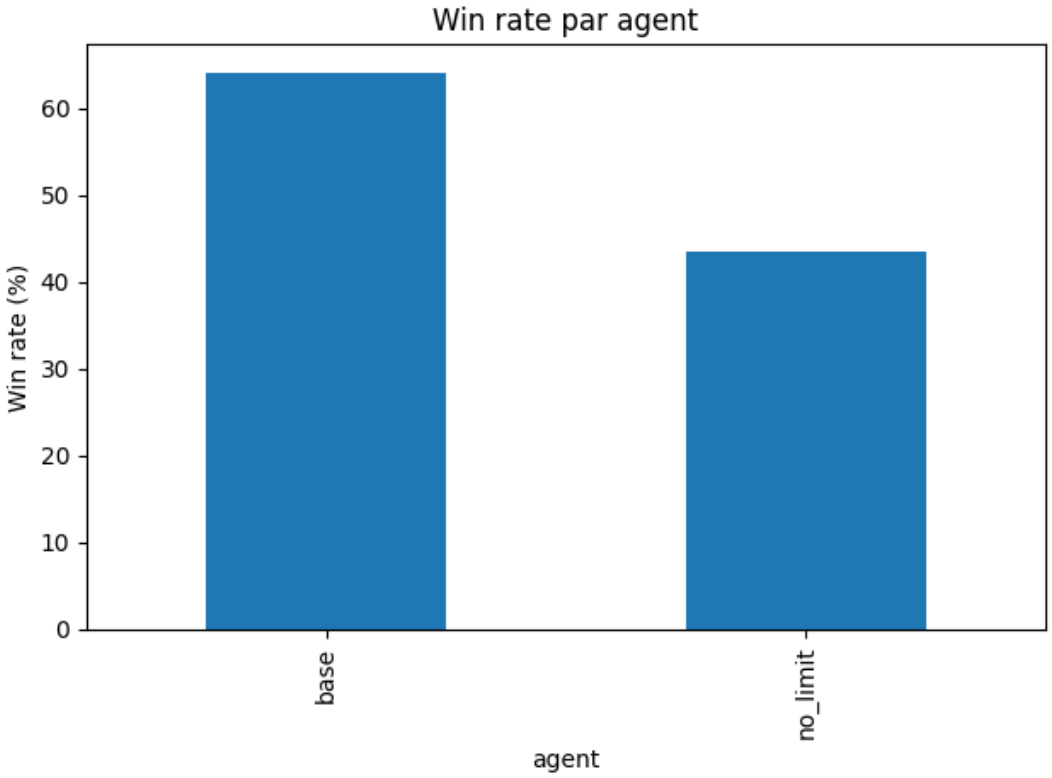

FIGURE 3.16 : Évaluation sans limite de temps : comparaison du taux de victoire.

3.5 Intégration au système embarqué de la voiture LIMO

Par manque de temps, nous ne sommes pas parvenus avec notre tuteur d'intégrer notre modèle sur le simulateur Gazebo, spécialisé dans la robotique. En effet, notre agent semblait avoir du mal à tourner et changer de direction dans l'environnement Gazebo développé par notre tuteur. Nous supposons que cela est lié au fait que la physique "artisanale" de notre environnement est différente de celle de Gazebo, qui est plus poussée et réaliste.

Chapitre 4

Conclusion

Dans ce projet, nous avons mis en pratique un exemple concret d'application de reinforcement learning dans le domaine de la robotique.

Nous avons acquéri des compétences dans l'utilisation de la librairie Gymnasium. De plus, nous avons vu à quel point il est délicat de créer un environnement personnalisé pour notre tache en particulier. Enfin, nous avons pu appliquer l'algorithme PPO à l'environnement que nous avons crée pour faire apprendre à la voiture de marquer son but.

Cependant, nous ne sommes pas parvenus à faire fonctionner la voiture dans de conditions réelles, avec un autre joueur. Ainsi, notre encadrant, Sébastien LENGAGNE et Cédric CHAUVIERE ont obtenu 4500 euros de financement pour la poursuite de ce projet. Effectivement, de nombreuses taches demandent à être réalisées pour la finalisation du projet, notamment :

- Permettre à la voiture de gérer le cas de plusieurs ballons sans adversaire.
- Faire apprendre à la voiture à jouer une réelle partie de LIMOBALL, c'est à dire avec plusieurs ballons, et un adversaire.

Bibliographie

Schulman, John / Wolski, Filip / Dhariwal, Prafulla / Radford, Alec / Klimov, Oleg(2017) : *Proximal policy optimization algorithms*.

Raffin, Antonin / Hill, Ashley / Gleave, Adam / Kanervisto, Anssi / Ernestus, Maximilian / Dormann, Noah(2021) : *Stable-Baselines3 : Reliable Reinforcement Learning Implementations*, 268 : 1-8.

Towers, Mark u.a.(2024) : *Gymnasium : A standard interface for reinforcement learning environments*.

labmlai (2022) : *Generalized Advantage Estimation Implementation* `https://nn.labml.ai/rl/ppo/gae.html`, 10/12/2025.

Renotte, Nicholas (2021) : *Reinforcement Learning in 3 Hours — Full Course using Python* .

Schulman, John / Moritz, Philipp / Levine, Sergey / Jordan, Michael / Abbeel, Pieter (2018) : *High-Dimensional Continuous Control Using Generalized Advantage Estimation* .