Création d'un agent pour Rocket League

Rapport délève ingénieur
Projet de 2e année
Année Universitaire 2023-2024

Présenter par :
Roubille Mathis
Dupois Thomas
Ouya Boris
Abdul-salam Sami

Responsable ISIMA :
Loic Yon

Durée du projet : 150 heures

29 mars 2024

Table des matières

Résumé **3**

Abstract **3**

1 Introduction **5**

2 Planification **6**

3 Présentation générale de Rocket League **7**

4 Apprentissage par renforcement **8**
4.1 Principes 8
4.1.1 Schéma général 8
4.1.2 Fonctions de valeur 9
4.1.3 Approximation des fonctions de valeur 10
4.1.4 Méthodes *on-policy* à gradient de politique 11
4.1.5 Algorithme PPO 12

5 Application du modèle d'apprentissage par renforcement à Rocket League **13**
5.1 Rlgym 13
5.2 Actions 15
5.3 Réprésentation de l'environnement 16
5.4 Structure du modèle 17

6 Entraînement du modèle **19**
6.1 Stable-Baselines 3 19

7 Experimentation **19**

8 Conclusion **22**

Références bibliographiques (norme IEEE) **24**

9 Annexe **25**

Table des figures

1 Diagramme de Gantt Prévisionnel 6
2 Diagramme de Gantt Réel 6
3 Logo de Rocket League 7
4 Interaction agent-environnement -[1] 9
5 Architecture *Actor-Critic* (tiré de [1] chapitre 11) 12
6 L^{CLIP} selon les valeurs de l'avantage (tiré de [1]) 13
7 Principe de l'algorithme PPO (tiré de [2]) 14
8 Logo de RLGym 14
9 Diagramme du fonctionnement de RLGym (tiré de [3] 15
10 Actions réalisables Rocket League (tiré de [4]) 16
11 Discrétisation de l'espace des actions (tiré de [4]) 16
12 Structure de SeerV1 18
13 Composantes de l'agent SeerV1 18
14 Illustration de Stable Baselines 19
15 Graphique d'évolution des récompenses et de la longueur moyenne des épisodes pour la phase 2 21
16 Graphique d'évolution des récompenses et de la longueur moyenne des épisodes pour la phase 3 22

Résumé

Le but de ce projet est de réaliser un agent pouvant jouer au jeu Rocket League en situation de 1v1. Rocket League est un jeu vidéo développé et édité par Psyonix mêlant jeu de course et football. Chaque joueur contrôle un véhicule et l'objectif est de marquer dans les cages adverses. Léquipe ayant marqué le plus de buts gagne le match. Lagent apprendra à jouer au jeu en duel et ce avec un niveau de jeu se rapprochant dun humain débutant.
La communauté autour de la création dagents Rocket League étant riche, le projet est réalisé en Python et utilise les librairies mises à disposition pour de telles entreprises, à savoir RLGym et RLGym-tools pour lentrainement et la simulation dans lenvironnement du jeu vidéo ainsi que stable-baselines3 pour la mise en place de lenvironnement dapprentissage par renforcement profond.
Après trois semaines de travail, l'agent est capable de maîtriser les bases telles que la navigation vers la balle, la frappe et marquer dans des situations simples. Bien que ses compétences correspondent à un niveau débutant, des progrès significatifs ont été réalisés. Cependant, des améliorations restent nécessaires pour rivaliser avec des joueurs plus expérimentés, notamment dans la stratégie de jeu, la défense et la prise de décision en temps réel.

Mots-clés : Rocket League, Python, RLGym, RLGym-tools, Stable-baselines3, Apprentissage par renforcement profond, On-policy, Algorithme PPO, Réseau de Neurones Récurrent, Pytorch, Agent

Abstract

The aim of this project is to create an agent that can play Rocket League in a 1v1 situation. Rocket League is a video game developed and published by Psyonix that combines racing and football. Each player controls a vehicle and the objective is to score in the opposing goal. The team with the most goals wins the match. The agent will learn to play the game in duels, at a level close to that of a human beginner.
As there is a strong community around the creation of Rocket League agents, the project is being carried out in Python and uses the libraries made available for such ventures, namely RLGym and RLGym-tools for training and simulation in the video game environment, and stable-baselines3 for setting up the deep reinforcement learning environment.
After three weeks of work, the agent is capable of mastering basics such as ball navigation, striking, and scoring in simple situations. Although its skills match those of a beginner level, significant progress has been made. However, improvements are still needed to compete with more experienced players, especially in game strategy, defense, and real-time decision-making.

Tags : Rocket League, Python, RLGym, RLGym-tools, Stable-baselines3, Deep reinforcement

learning, On-Policy,PPO, Reccurent Neural Network, Pytorch, Agent

1 Introduction

Alors que les besoins d'automatisation de procédés croissent, les domaines de l'apprentissage automatique ont le vent en poupe. Parmi ceux-ci, l'apprentissage par renforcement a au fil des ans acquis une place solide dans des domaines divers et variés, allant de l'industrie au secteur du jeu vidéo.

C'est justement dans le secteur du jeu vidéo que l'apprentissage par renforcement acquit ses premières lettres de noblesse ou du moins ses premiers principaux succès : les techniques de ce domaine constituent de puissants outils permettant la création d'agent automatique pour des tâches comprenant le jeu contre ordinateur, l'intelligence d'alliés virtuels ou encore la création d'outils d'aide aux développeurs.

Rocket League, jeu développé et édité par la société américaine Psyonix et sorti en juillet 2015, fait indubitablement partie des jeux sur lesquels la discussion sur l'apprentissage par renforcement et en particulier l'apprentissage par renforcement profond est la plus vivace. En effet, les membres de la communauté éclectique autour de ce jeu peuvent proposer leurs propres agents pouvant y jouer avec des degrés de technicité variés. Parmi ceux-ci, Nexto et Seer réussissent à jouer à un niveau s'approchant du niveau SSL (Super Sonic Legend), soit le niveau de jeu le plus élevé de Rocket League.

Notre projet vise la création d'un agent Rocket League uvrant en situation de duel. Cet agent devra atteindre le niveau d'un joueur humain debutant à Rocket League d'un point de vue mécanique, c'est-à-dire :

— Manoeuvrer vers la balle quimporte sa position relative à celle-ci
— Saisir les opportunités de marquer dans le camp adverse
— Défendre ses cages contre les attaques adverses

La communauté autour de la création d'agents Rocket League a mis à la disposition du grand public une API en python nommée RlGym basé sur Gym dOpen AI et Tensorflow de Google pour simuler l'entrainement d'agent. En conjonction avec cet environnement, nous utiliserons la bibliothèque stable-baselines3 qui implémente divers algorithmes d'apprentissage par renforcement, dont l'algorithme PPO (Proximal Policy Optimization). Pour la visualisation de nos résultats, nous utilisons Tensorboard.

Nous présentons dans un premier temps l'organisation du projet en termes de division temporelle et de responsabilités. Ensuite, nous ferons la présentation générale de Rocket League. Après cela, nous ferons un brève introduction à l'apprentissage par renforcement et expliquerons les parties de ce domaine que nous exploitons dans le cadre de notre projet. Une fois le cadre du projet posé, nous expliquerons notre méthodologie d'entrainement. Enfin, nous discuterons des résultats obtenus et des pistes que pourraient explorer de futurs projets se basant sur celui-ci.

2 Planification

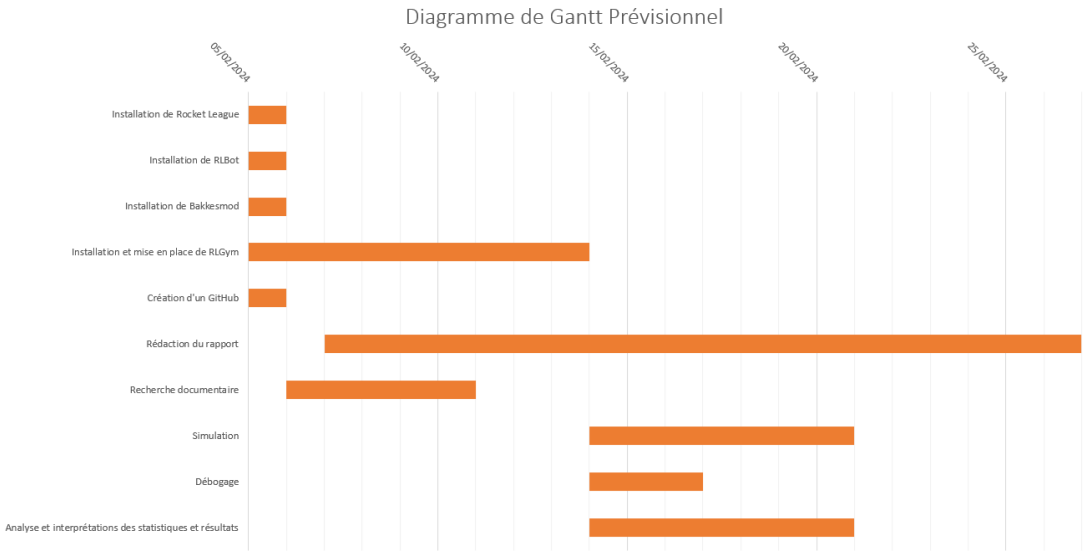

FIGURE 1 – Diagramme de Gantt Prévisionnel

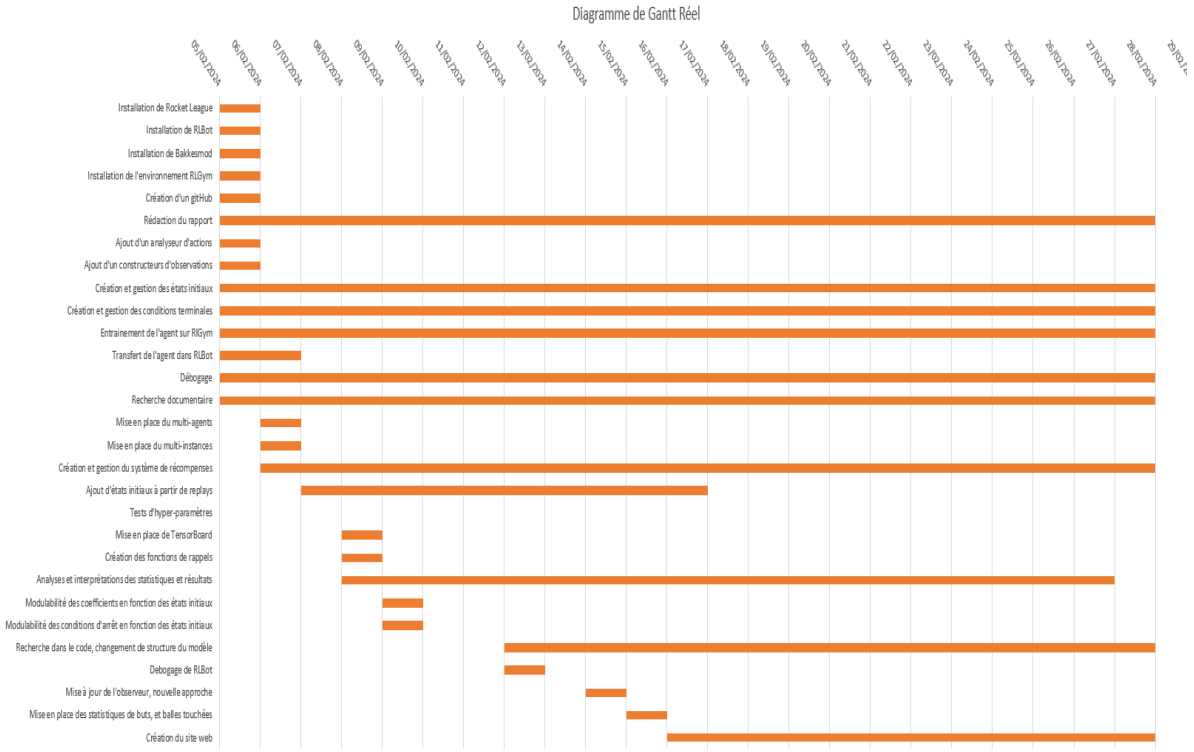

FIGURE 2 – Diagramme de Gantt Réel

3 Présentation générale de Rocket League

FIGURE 3 – Logo de Rocket League

Rocket League est un jeu vidéo développé et édité par Psyonix. Initialement publié en juillet 2015, il est disponible sur différentes plateformes telles que Playsation, Xbox, Nintendo Switch et PC. Le jeu combine des éléments de football avec des véhicules équipé de propulseurs, offrant ainsi une expérience et une mécanique de jeu uniques.
Le jeu se déroule de la manière suivante :

- Deux équipes s'affrontent, comportant entre 1 et 3 joueurs chacune.
- En début de partie, les joueurs apparaissent à des points prédéfinits de sorte à ce que les joueurs soient tous à égale distance du ballon positionné au milieu du terrain. Cette situation se nomme le coup d'envoi.
- Une fois le ballon mis en jeu, l'objectif des joueurs est de pousser le ballon dans le but adverse afin de marquer un point.
- Lorsqu'une équipe marque un but, les joueurs sont replacés en condition de coup d'envoi.
- L'équipe possédant le plus de points à la fin du temps imparti remporte la partie. En cas d'égalité, la partie se dispute en mort subite : la première équipe marquant un point remporte le match.

Ce qui rend ce jeu particulier, c'est l'aspect mécanique de celui-ci comme évoqué précédemment. En effet, les véhicules, équipés de propulseurs, ont la possibilité de se déplacer temporairement dans les airs et de pivoter selon les besoins. Ils peuvent également rouler sur les murs qui entourent le terrain en fonction de leur vitesse. Ces mouvements particuliers augmentent donc les différentes manières de traiter une situation donnée, ce qui rend le jeu bien plus complexe qu'un jeu de football classique.

4 Apprentissage par renforcement

L'apprentissage par renforcement est un domaine de l'apprentissage automatique à l'instar de l'apprentissage supervisé et de l'apprentissage non supervisé. Rappelons d'abord ce dont il est question lorsque que l'on se réfère à ces différents domaines.

L'apprentissage supervisé est le domaine de l'apprentissage automatique dans lequel les algorithmes d'apprentissage disposent d'une base de données labélisées, c'est-à-dire dans laquelle chaque observation d'entraînement (donnée prédictive) est associée à une valeur cible (prédiction) connue, émise par un expert. L'apprentissage non supervisé est, quant à lui, associé à la recherche de structures dans des données non labélisées.

L'apprentissage par renforcement s'attaque aux problèmes où un agent apprend de son environnement via une répétition d'essais et rétroactions. La connaissance de l'environnement est souvent limitée et l'agent ne peut y naviguer que par le biais de ses actions qui y induisent des changements détactables par des signaux appelées **récompenses**. Le but de l'agent est alors de maximiser les récompenses qu'il reçoit en favorisant les actions qui les augmentent et défavorisant celles qui les diminuent.

4.1 Principes

4.1.1 Schéma général

Les problèmes d'apprentissage par renforcement sont définis à l'aide des *processus de décision markoviens* [1] (chapitre 3). Ainsi, nous définissons :

- un ensemble d'états *S* modélisant l'environnement de travail. Certains d'entre eux correspondent à des *situations terminales* (par exemple, une fin de partie dans un jeu) ; nous définissons alors $S_T \subset S$ comme l'ensemble des états terminaux. Dans le cadre où l'on peut définir des états terminaux, on appelle *episode* une succession d'états terminée par un état terminal.
- un ensemble d'actions *A* que l'agent peut effectuer. A noter que certaines actions peuvent être interdites dans certains états .On peut donc définir pour un état $s \in S$ l'ensemble $A_s \subset A$ des actions possibles dans l'état s
- une fonction de récompense *R* qui fournit une *récompense numérique* à un agent lors de sa transition d'un état s à s' par le biais d'une action a.
- une politique π qui pour un état $s \in S$ et une action $a \in A_s$ donne la probabilité d'effectuer l'action a sachant que l'agent se trouve dans l'état s.

Dans ce contexte, le schéma général d'un projet d'apprentissage par renforcement se présente comme-suit :

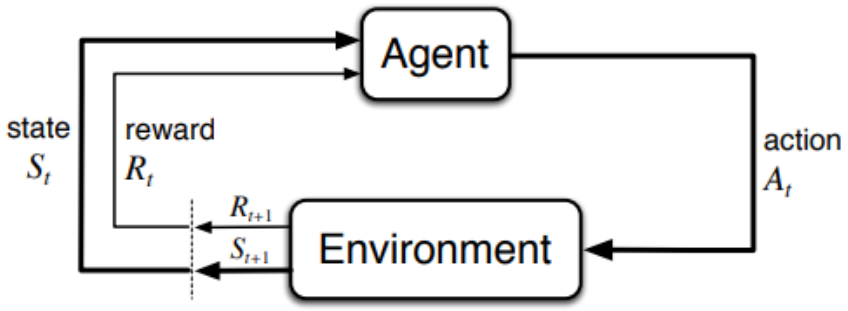

FIGURE 4 – Interaction agent-environnement -[1]

Un agent effectue des actions (A_t) choisies selon la politique π qui affectent son environnement. Celui-ci s'adapte en conséquence et renvoie une représentation de lui-même (S_{t+1}) à l'agent ainsi que la récompense générée par son action (R_{t+1}). Ces informations serviront à l'agent afin de mettre à jour sa *politique* et choisir la prochaine action à effectuer.

L'objectif de l'agent est de **maximiser l'espérance du gain cumulé actualisé** dont la formule est la suivante ([1] chapitre 3 , p60, eq. 3.2) :

$$G = \sum_{t=0}^{T} \gamma^t R_t \tag{1}$$

avec T, le pas de temps correspondant à un état terminal et $\gamma \in [0, 1]$. r

γ est appelé *facteur de dévaluation*. Comme son nom l'indique, il sert intuitivement à dévaluer les récompenses obtenues au fil du temps (cas $\gamma < 1$). L'intérêt est double. Tout d'abord, une somme infinie non dévaluée ($\gamma = 1$) pourrait ne pas être bien définie. Ensuite, γ permet de contrôler à quel point l'agent se base sur les futurs retours dans le choix de ses actions. Mettre γ à 0 nous donne un agent dont la seule motivation est le retour à l'état courant tandis qu'une valeur plus élevée de γ favorise la planification ([1] p. 60).

4.1.2 Fonctions de valeur

Afin de guider l'agent dans la poursuite de son objectif , on définit ,pour une politique π, une *fonction de valeur d'état* V^π nous donnant pour chaque état l'espérance du gain futur en partant de l'état considéré. On peut alors la définir ainsi, pour tout $s \in S$:

$$V^\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{+\infty} \gamma^{t+k+1} R_{t+k+1} | S_t = s] \tag{2}$$

avec t , un pas de temps quelconque et E_π l'espérance mathématique d'une variable aléatoire sachant que l'agent suit la politique ([1] p.70 eq 3.10).

On définit en conjonction une *fonction de valeur d'action* Q^π qui pour une politique π, donne pour chaque chaque état $s \in S$ et chaque action $a \in A_s$ l'espérance du gain juste après avoir fait l'action a à l'état s ([1] p.70 eq 3.11) :

$$Q^\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, a_t = a] \tag{3}$$

Les deux fonctions de valeur sont reliées par la relation :

$$V^\pi(s) = \sum_{a \in A_s} \pi(a|s) Q^\pi(s,a) \tag{4}$$

Nous pouvons aussi définir une fonction d'avantage A^π :

$$A^\pi(a,s) = Q^\pi(s,a) - V^\pi(s) \tag{5}$$

Intuitivement, cette fonction donne l'avantage (ou le désavantage) d'effectué l'action a à l'état s par rapport au gain moyen après visite de l'état s.

Ces trois fonctions servent de pilotes à de nombreux algorithmes d'apprentissage par renforcement.

4.1.3 Approximation des fonctions de valeur

Les fonctions sont généralement approximées par expérience ([1] p.70). Dans ce cadre, nous retrouvons deux stratégies ([1] p.70) :

— appliquer des méthodes dites de Monte-Carlo pour aproximer ces fonctions pour chaque état et chaque combinaison état-action dont le principe est de garder pour chaque point d'évaluation (état ou combinaison état-action) une moyenne des gains collectés lors d'expériences

— approximer ces fonctions par des fonctions paramétrées dont les paramètres seront mis à jour par l'expérience

.

La première stratégie souffre du fait que le coût en terme de mémoire grandit plus le nombre d'états augmente. Outre le coût en mémoire, les coûts en temps de calcul et temps de simulation peuvent être très élevées pour visiter un nombre suffisamment de fois chaque état et réaliser chaque paire état-action possible ([1] p.225).

La seconde stratégie quant à elle s'appuie sur de l'*approximation de fonction* , ensemble de problèmes étudiés dans les domaines de l'apprentissage automatique. La méthodologie est donc d'utiliser les connaissances de ce domaine afin de trouver des approximateurs qui soit à même d'approximer nos fonctions sans pour autant avoir rencontré toutes les paires d'action-valeur possible, c'est-à-dire faire preuve de *généralisation* ([1] p.225). On peut donc utiliser des *réseaux de neurones* afin de servir de fonction d'approximation.

Une autre thématique importante lors de l'approximation d'une fonction de valeur est la collecte de données. Sur ce critère, nous avons deux types d'algorithmes d'apprentissage par renforcement :

— les méthodes dites *on-policy* : la politique comportementale est la même que celle à optimiser. Les performances de l'agent sont donc directement responsables de la collecte des données sur lesquelles il s'entraîne

— les méthodes dites *off-policy* ; la polique comportementale est différenciée de la polique à optimiser. La collecte des données est donc indépendante de cette dernière.

4.1.4 Méthodes *on-policy* à gradient de politique

Rappelons l'objectif d'un agent : maximiser l'espérance de son gain cumulé actualisé

$$G = \sum_{t=0}^{+\infty} \gamma^t R_t \tag{6}$$

On se place de le cadre d'une politique π_θ paramétrée par des paramètres θ.

Les méthodes à politique de gradient ont pour principe d'utiliser un estimateur du gradient de la politique qui sera introduit dans un algorithme de montée du gradient. L'estimateur du gradient le plus courramment utilisé est celui-ci([2] p.2 eq 1) s :

$$\hat{g} = \mathbb{E}[\nabla_\theta \log \pi(a_t|s_t)\hat{A}_t] \tag{7}$$

avec $\hat{A}_t$, un estimateur de la *fonction d'avantage à l'instant t*.

Un estimateur courant de la fonction d'avantage est le GAE(λ) (*Generalized Advantage Estimator*).

Ainsi, le gradient ne renforcera la probabilité $\pi_\theta(a_t|s_t)$ que si $\hat{A}_t > 0$ et l'affaiblira quand $\hat{A}_t < 0$.

Le gradient $\hat{g}$ s'obtient en différenciant la fonction-objectif suivante ([2] eq 1) :

$$L^{PG}(\theta) = \mathbb{E}[\log \pi_\theta(a_t|s_t)\hat{A}_t] \tag{8}$$

.

Optimiser plusieurs fois sur une même *trajectoire* cette fonction peut amener à une politque assez éloignée de l'ancienne. Cela limite donc l'utilité de chaque trajectoire récoltée au cours des expérience. Pour pallier à ce problème, des méthodes qui visent à limiter la *distance entre deux politiques successives* ont été introduites. Nous y retrouvons entre autre l'algorithe TRPO (*Trust Region Policy Optimization*) et l'algorithme PPO (*Proximal Policy Optimisation*). Ces algorithmes permettent donc d'être plus efficients au niveaux des données d'apprentissage : il est possible de s'entraîner plusieurs fois sur les trajectoires données sans pour craindre de tendances destructives ([2].

L'algorithme PPO nous intéresse tout particulièrement, puisqu'il s'agit de l'algorithme que nous utilisons dans notre projet.

4.1.5 Algorithme PPO

L'algorithme PPO est une méthode *on-policy* à gradient de politique de type *Actor-Critic*.
Une méthode de type Actor-Critic est une méthode d'optimisation où coexistent deux entités :
— la politique décisionnelle d'une part ;
— la critique d'autre part dont le rôle est d'évaluer la politique de l'agent.

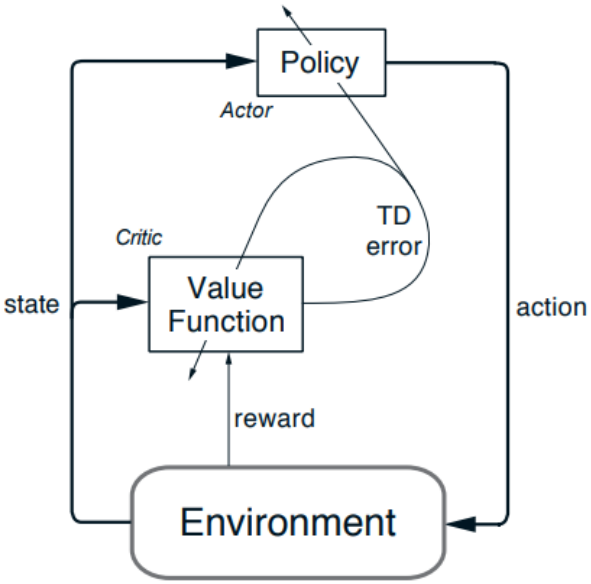

FIGURE 5 – Architecture *Actor-Critic* (tiré de [1] chapitre 11)

L'erreur de la critique sert aussi bien à sa propre amélioration qu'à celle de la politique. Cette erreur appelée *TD-error* est donnée comme suit ([1] chapitre 11) :

$$\delta_t = R_t + \gamma V(s_{t+1}) - V(s_t) \tag{9}$$

Comme mentionné précédemment, l'algorithme PPO évite l'écueil de l'explosion du gradient des méthode de gradient de politque . Ceci est rendu possible par la maximisation d'une fonction de substitution qui prend cette forme si le réseau de neurones de l'agent partage des couches entre la partie politique et la partie critique ([2]) :

$$L^{CLIP+VF+S}(\theta) = \mathbb{E}[L^{CLIP} - c_1 L^{VF} + c_2 S[\pi_\theta](s_t)] \tag{10}$$

avec c1 et c2, des réels positifs.
En posant

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{ancien}}(a_t|s_t)} \tag{11}$$

avec $\pi_{\theta ancien}$, la politique à optimiser, on définit

$$L^{CLIP} = \mathbb{E}[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t] \tag{12}$$

avec $0 < \epsilon < 1$.

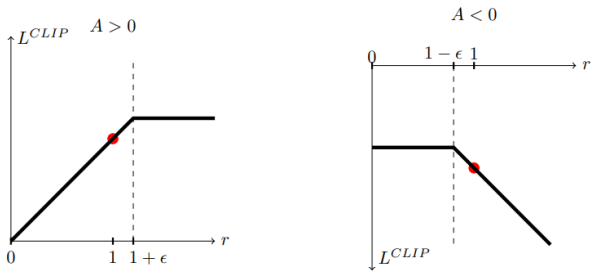

FIGURE 6 – L^{CLIP} selon les valeurs de l'avantage (tiré de [1])

Restreindre les valeurs du ratio de probabilités en deuxième terme du minimum permet d'ôter toute inclinaison au ratio de se trouver hors de l'intervalle [1-ϵ,1+ϵ].

Si l'avantage est positif, le comportement est à renforcer et donc le ratio est augmenté jusqu'à un maximum de 1 + ϵ.

Si au contraire l'avantage est négatif, le comportement est pénalisé et le ratio est diminué jusqu'à un minimum de 1 - ϵ.

$$L^{VF}(\theta) = (V_\theta^\pi(s_t) - V_t^{targ})^2 \tag{13}$$

est la fonction d'erreur de la critique.

$S[\pi_\theta](s_t)$ est quant à lui un bonus d'entropie. Il s'agit d'un terme permettant à l'agent de promouvoir l'*exploration*. Plus le coefficient c_2 est élevé, plus la tendance à l'exploration est forte [2].

Supposons que nous disposions de N environnements dans chacun desquels agit un agent suivant la politique π_{ancien}.

Si l'on prend T $\geq$ 1, une longeur de trajectoire tel que T soit bien inférieure à la longeur d'un épisode, on peut alors définir pour $t \in [0, T]$ [2]

$$\hat{A}_t = \sum_{i=0}^{T-t+1} (\lambda\gamma)^i \delta_{i+t} \tag{14}$$

5 Application du modèle d'apprentissage par renforcement à Rocket League

5.1 Rlgym

Pour entrainer un agent afin qu'il puisse résoudre un problème où effectuer une tâche précise, il nous faut avant tout lui fournir un *environnement d'entrainement cohérent avec la tâche qu'il devra effectuée*.

Algorithm 1 PPO, Actor-Critic Style

for iteration=$1, 2, \ldots$ **do**
  for actor=$1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

FIGURE 7 – Principe de l'algorithme PPO (tiré de [2])

L'une des raisons pour lesquelles le jeu vidéo se prête bien à l'apprentissage par renforcement est que le jeu lui-même peut servir d'environnement par le biais d'APIs pouvant fournir des informations sur l'état d'une partie.

OpenAI fournit de multiples environnements pour l'entrainement d'agents dans des environnements de références par le biais de l'API *Gym*. L'environnement que nous utilisons dans notre projet est directement basé sur celle-ci : Rocket League Gym.

Rocket League Gym (RLGym) est une API Python qui peut être utilisée pour traiter le jeu Rocket League comme s'il s'agissait d'un environnement de style OpenAI Gym .

FIGURE 8 – Logo de RLGym

RLGym nous permet de manipuler le jeu Rocket League comme un environnement auquel on peut entraîner un agent.

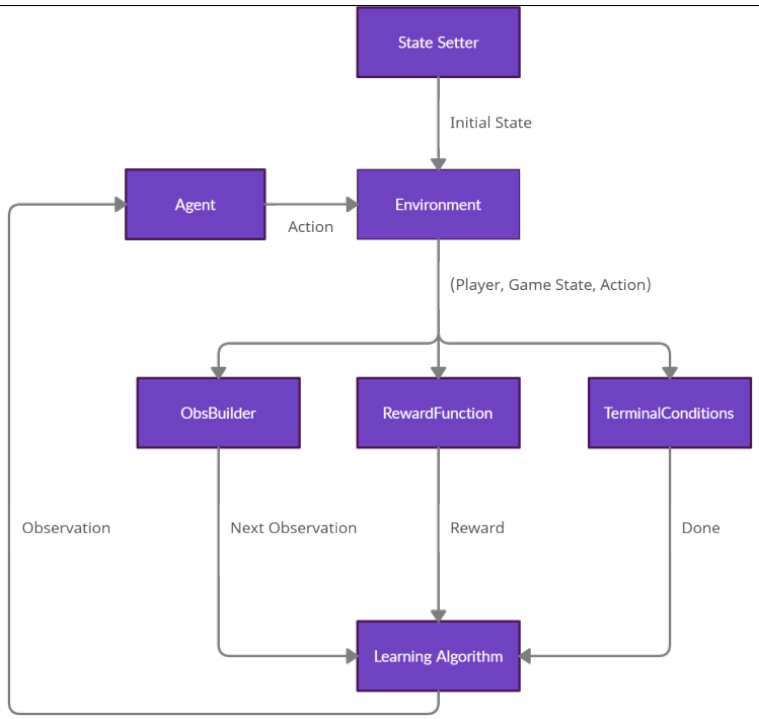

FIGURE 9 – Diagramme du fonctionnement de RLGym (tiré de [3]

L'API se prête bien au format épisodique. En effet, l'environnement de Gym nous permet de définir :

— des états initiaux : on peut configurer divers paramètres tels que la position et l'orientation de l'agent, la position du ballon ainsi que sa vitesse , la quantité de boost des propulseurs. En somme, on peut reproduire des positions intéressantes à partir desquelles un agent effectue son apprentissage. Par exemple, une position ressemblant fortement à un coup d'envoi.
— des fonctions de récompense : elles permettent de récompenser ou de punir l'agent en fonction du comportement qu'il adopte dans une situation donnée.
— des conditions terminales qui indiquent la fin d'un épisode.

5.2 Actions

Il existe 8 actions différentes dans Rocket League, dont 5 continues et 3 booléennes.

Puisque les Cependant, des résultats démontrent que travailler avec un espace d'action discrétisé améliore l'apprentissage [5]. Nous discrétisons donc l'espace des actions comme-suit :

Action	Domain
Throttle	$[-1.0, 1.0]$
Steer	$[-1.0, 1.0]$
Pitch	$[-1.0, 1.0]$
Yaw	$[-1.0, 1.0]$
Roll	$[-1.0, 1.0]$
Jump	$\{0, 1\}$
Boost	$\{0, 1\}$
Handbrake	$\{0, 1\}$

FIGURE 10 – Actions réalisables Rocket League (tiré de [4])

Action	Domain
Throttle	$\{-1, 0, 1\}$
Steer/Yaw	$\{-1, -0.5, 0, 0.5, 1\}$
Pitch	$\{-1, -0.5, 0, 0.5, 1\}$
Roll	$\{-1, 0, 1\}$
Jump	$\{0, 1\}$
Boost	$\{0, 1\}$
Handbrake	$\{0, 1\}$

FIGURE 11 – Discrétisation de l'espace des actions (tiré de [4])

5.3 Réprésentation de l'environnement

La représentation de l'environnement dans lequel évolue le agentdans Rocket League est essentielle pour prendre des décisions appropriées. Voici une liste des informations que l' agent peut observer dans cet environnement :

- Position de la balle : coordonnées spatiales de la balle sur le terrain.
- Vitesse linéaire de la balle : la vitesse à laquelle la balle se déplace dans l'espace.
- Vitesse angulaire de la balle : la vitesse de rotation de la balle.
- Norme de la vitesse de la balle : la magnitude de la vitesse de la balle.
- Action précédente : l'action que l'agent a exécutée lors du dernier pas de temps.
- Position des boosts : emplacements des boosters d'accélération sur le terrain.
- Positions des autres voitures : coordonnées spatiales des autres voitures sur le terrain.
- Position de la voiture de l'agent : coordonnées spatiales de la voiture contrôlée par l'agent sur le terrain.
- Distance de la voiture du agent par rapport aux autres voitures : la distance entre la voiture du agent et les autres voitures.
- Différence de vitesse linéaire de l'agent par rapport aux autres voitures : la différence de vitesse linéaire entre la voiture de l'agent et les autres voitures.
- Norme de la distance de la voiture de l'agent par rapport aux autres voitures : la magnitude

de la distance entre la voiture de l'agent et les autres voitures.
— Norme de la différence de vitesse linéaire de l'agent par rapport aux autres voitures : la magnitude de la différence de vitesse linéaire entre la voiture de l'agent et les autres voitures.
— Vitesse linéaire des voitures : la vitesse à laquelle les voitures se déplacent.
— Vitesse angulaire des voitures : la vitesse de rotation des voitures.
— Position relative des voitures par rapport à la balle : les coordonnées spatiales des autres voitures par rapport à la position de la balle.
— Vitesse linéaire des voitures par rapport à la balle : la vitesse à laquelle les voitures se déplacent par rapport à la balle.
— Position relative des voitures par rapport au fond du but défendu : les coordonnées spatiales des voitures par rapport au fond du but que l'agent défend.
— Position relative des voitures par rapport au fond du but attaqué : les coordonnées spatiales des voitures par rapport au fond du but que l'agent attaque.
— Vitesse vers l'avant de chaque voiture : la composante de la vitesse des voitures dans la direction avant.
— Vitesse vers le haut de chaque voiture : la composante de la vitesse des voitures dans la direction verticale.
— Niveau de boost de chaque voiture : la quantité de boost disponible pour chaque voiture.
— Si les voitures sont au sol ou non : indique si les voitures sont au sol ou en l'air.
— Si les voitures ont utilisé leur flip ou non : indique si les voitures ont utilisé leur flip.
— Si les voitures ont été détruites (demos) ou non : indique si les voitures ont été détruites ou non.
— Si les voitures se déplacent à une vitesse supersonique ou non : indique si les voitures se déplacent à une vitesse supersonique.

Toutes les observations sont mises à l'échelle entre -1 et 1.

5.4 Structure du modèle

Le modèle possède une structure similaire à celle de SeerV1 '[4].

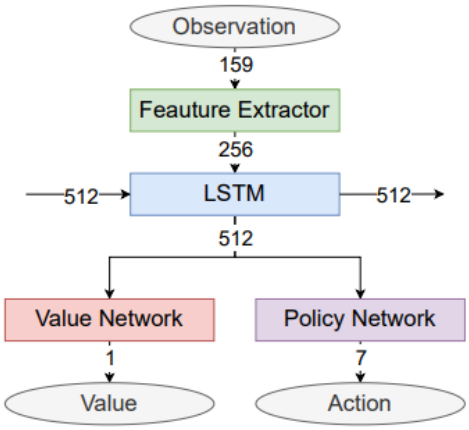

FIGURE 12 – Structure de SeerV1

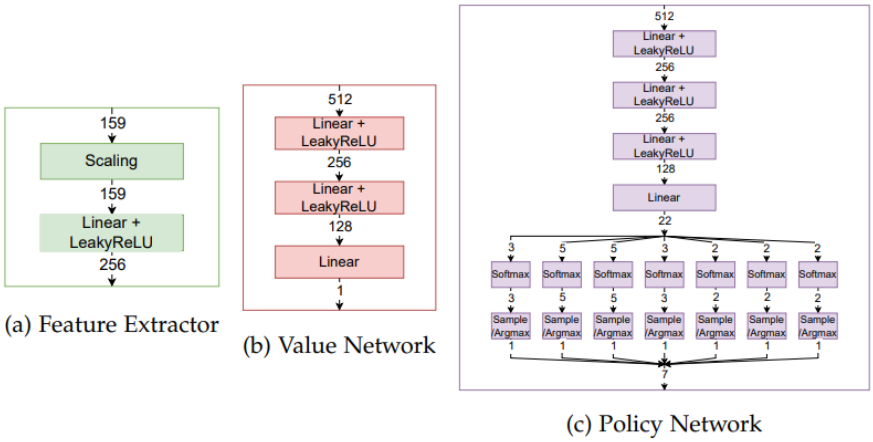

FIGURE 13 – Composantes de l'agent SeerV1

Ainsi, nous retrouvons :

1. Une couche d'extraction de caractéristiques ayant pour but de traduire les observations en données traitables par la politique
2. une couche LSTM qui permet au modèle de garder une mémoire à court termes des actions qu'il a eu à effectué
3. Deux réseaux de neurones partageant les couches précédentes :
   — le réseau de politique qui à partir des données d'observations transformées par le réseau commun fournit des probabilités d'effectuer les actions
   — La critique qui est dans notre cas un réseau approximant la fonction de valeur de notre environnement.

6 Entraînement du modèle

6.1 Stable-Baselines 3

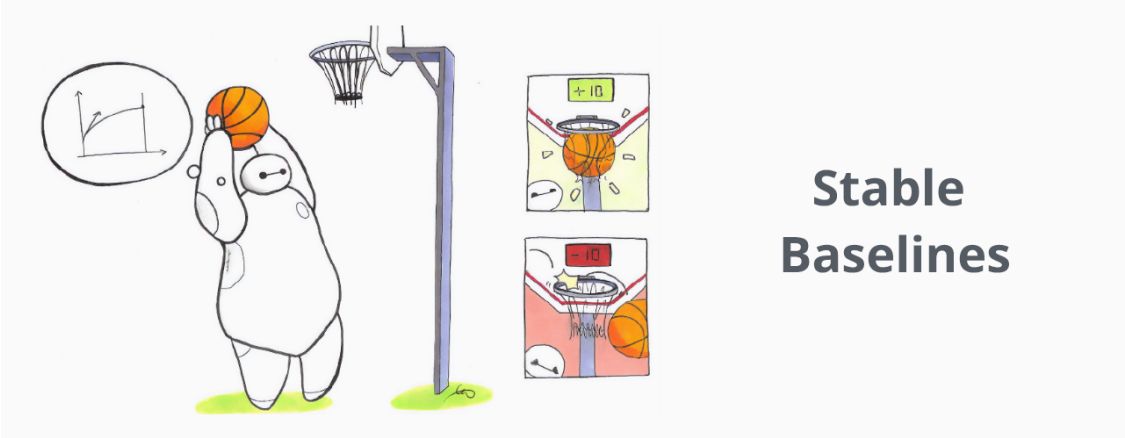

FIGURE 14 – Illustration de Stable Baselines

Stable-Baselines3 (SB3) est une bibliothèque fournissant des implémentations fiables d'algorithmes d'apprentissage par renforcement en PyTorch. Elle offre une interface propre et simple, donnant accès à ces algorithmes [6].

SB3 propose une implémentation de la PPO mais aucune implémentation supportant une politique dotée d'une LSTM. Nous utilisons donc *sb3-contrib*, un projet issu d'un branchage de Stable-baselines3. Ce dernier comprend l'algorithme *ReccurrentPPO* dont le principe sous-jacent s'approche de celui de la PPO mais supportant l'usage de politique à LSTM.

Pour l'optimisation des poids des réseaux de neuronnes, nous utilisons l'optimiseur Adam.

7 Experimentation

Dans la dernière version de l'agent, les ajustements visant à améliorer ses performances ont principalement porté sur trois aspects : les state setters (définissant les états initiaux), les fonctions de récompense et les conditions d'arrêt. Les state setters positionnent l'agent et la balle dans des configurations spécifiques pour chaque phase d'entraînement. Les fonctions de récompense déterminent les incitations pour l'agent à effectuer des actions spécifiques. Les conditions d'arrêt définissent les critères pour mettre fin à un épisode d'entraînement.

Pour les graphiques des récompenses et de la durée des épisodes, la partie "eval" correspond à l'évaluation de l'agent effectuée à intervalles réguliers sur 500 épisodes, tandis que la partie "rollout" représente les résultats sur 100 épisodes pendant l'entraînement. Il convient de noter que lors de l'entraînement, l'agent cherche à explorer l'univers des possibilités, tandis qu'en évaluation, il exécute les actions qu'il estime être les meilleures. Par conséquent, on peut s'attendre à de

meilleurs résultats en mode évaluation par rapport au mode entraînement.

Initialement, l'agent a été entraîné à effectuer des touches simples de la balle dans des situations basiques. Les state setters ont été utilisés pour placer l'agent dans des configurations similaires à des coups d'envoi, où l'agent était orienté vers la balle au centre du terrain. Durant cette phase initiale, l'agent était récompensé lorsqu'il touchait la balle, avec un coefficient de 5, et lorsqu'il était orienté vers la balle, avec un coefficient de 0.001. Cette première phase a été relativement brève, permettant simplement à l'agent d'apprendre à se diriger vers la balle. Les conditions d'arrêt utilisées étaient un temps limite fixé à 10 secondes et l'arrêt de l'épisode si l'agent ne touchait pas la balle en 5 secondes.

La deuxième phase a consisté en une extension de la première, mais avec une complexification de l'exercice. Les state setters ont positionné la balle de manière aléatoire sur le terrain, de même que les voitures, avec des orientations variées. En plus des fonctions de récompense précédentes, une nouvelle fonction de récompense a été ajoutée pour la proximité avec la balle, avec un coefficient de 0.001. Les conditions d'arrêt sont restées les mêmes mais la condition de temps a été augmentée à 20 secondes.
Au cours de cette phase, l'agent a enregistré une augmentation constante de ses récompenses ainsi que de la durée des épisodes pendant l'entraînement, ce qui correspond aux résultats attendus. En effet, la durée d'un épisode est plus longue lorsque l'agent parvient à toucher la balle. De plus, on remarque que l'agent était encore en plein progrès et ne stagnait pas, ce qui suggère qu'il aurait probablement continué à s'améliorer s'il n'avait pas été interrompu (voir figure 15).

Dans la troisième et dernière phase, l'objectif principal était d'enseigner à l'agent à tirer la balle dans les cages lorsqu'il en avait l'occasion. Pour ce faire, deux ensembles de state setters ont été utilisés : l'un plaçait la balle près des cages avec l'agent attaquant souvent plus proche de la balle que le défenseur, représentant 80% des configurations, tandis que l'autre plaçait l'agent défenseur entre la balle et les cages, bien que la balle restât toujours proche des cages. De plus, des récompenses supplémentaires ont été introduites : un coefficient de 10 pour marquer un but, un coefficient de 0.001 pour l'alignement entre l'agent, la balle et les cages, et un coefficient de 0.001 pour la position correcte de l'agent par rapport à la balle. La condition d'arrêt basée sur le temps a été montée à 30 secondes et la condition si il n'y a de but marqué en 10 secondes a remplacé celle de toucher la balle.
On observe un schéma similaire à celui de la phase 2 : les récompenses de l'agent augmentent avec la durée des épisodes, ce qui est conforme à nos attentes. En effet, si l'agent parvient à marquer, l'épisode sera plus long, et donc plus l'agent marque, plus la durée moyenne des épisodes augmente. De même, on remarque que la courbe d'évolution des récompenses de l'agent est encore en progression à la fin, et ne stagne pas, ce qui suggère que l'agent aurait probablement continué à s'améliorer s'il avait poursuivi son entraînement.(voir figure 16)

Cette stratégie a permis à l'agent de progresser de manière significative dans sa capacité à interagir avec la balle et à marquer des buts, démontrant ainsi une amélioration graduelle tout au long des différentes phases d'entraînement.

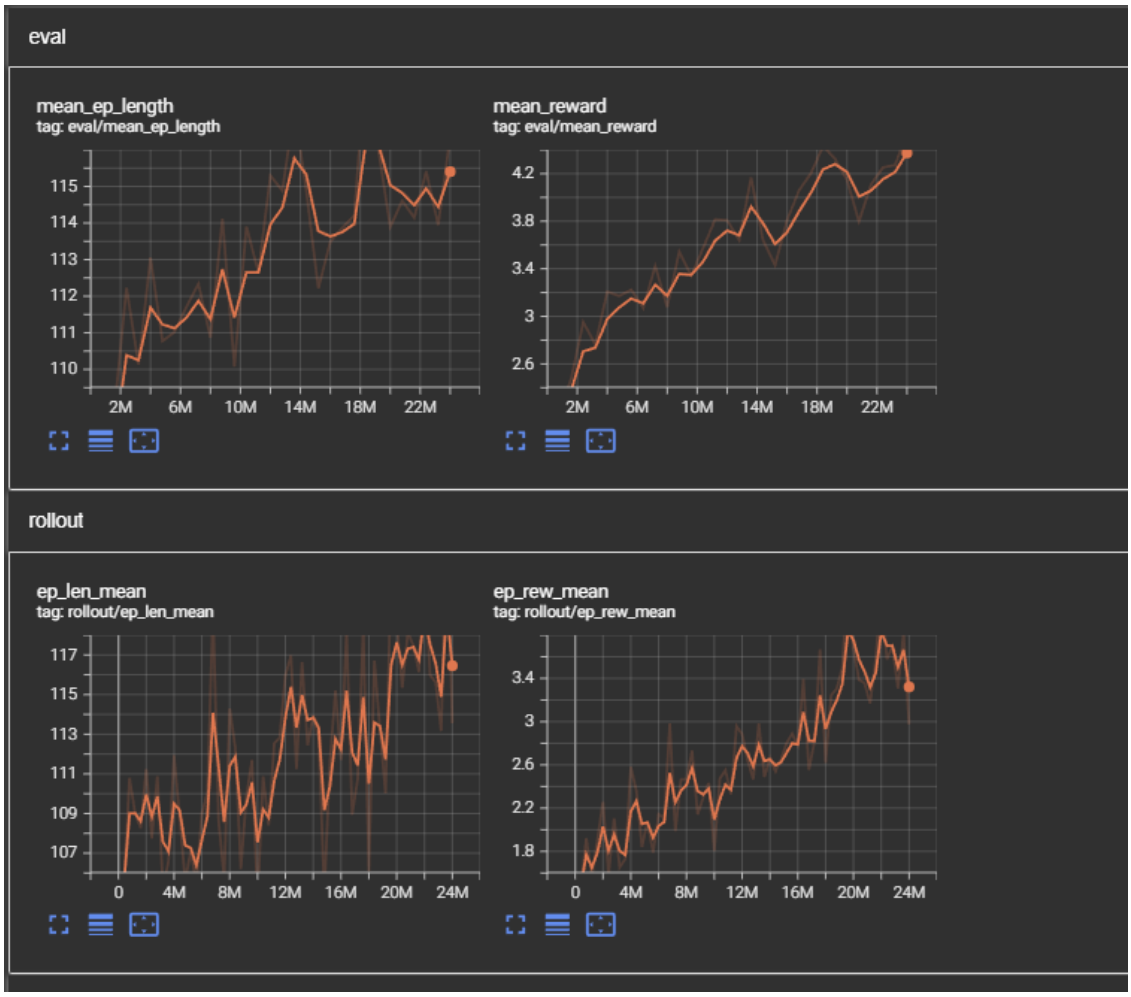

FIGURE 15 – Graphique d'évolution des récompenses et de la longueur moyenne des épisodes pour la phase 2

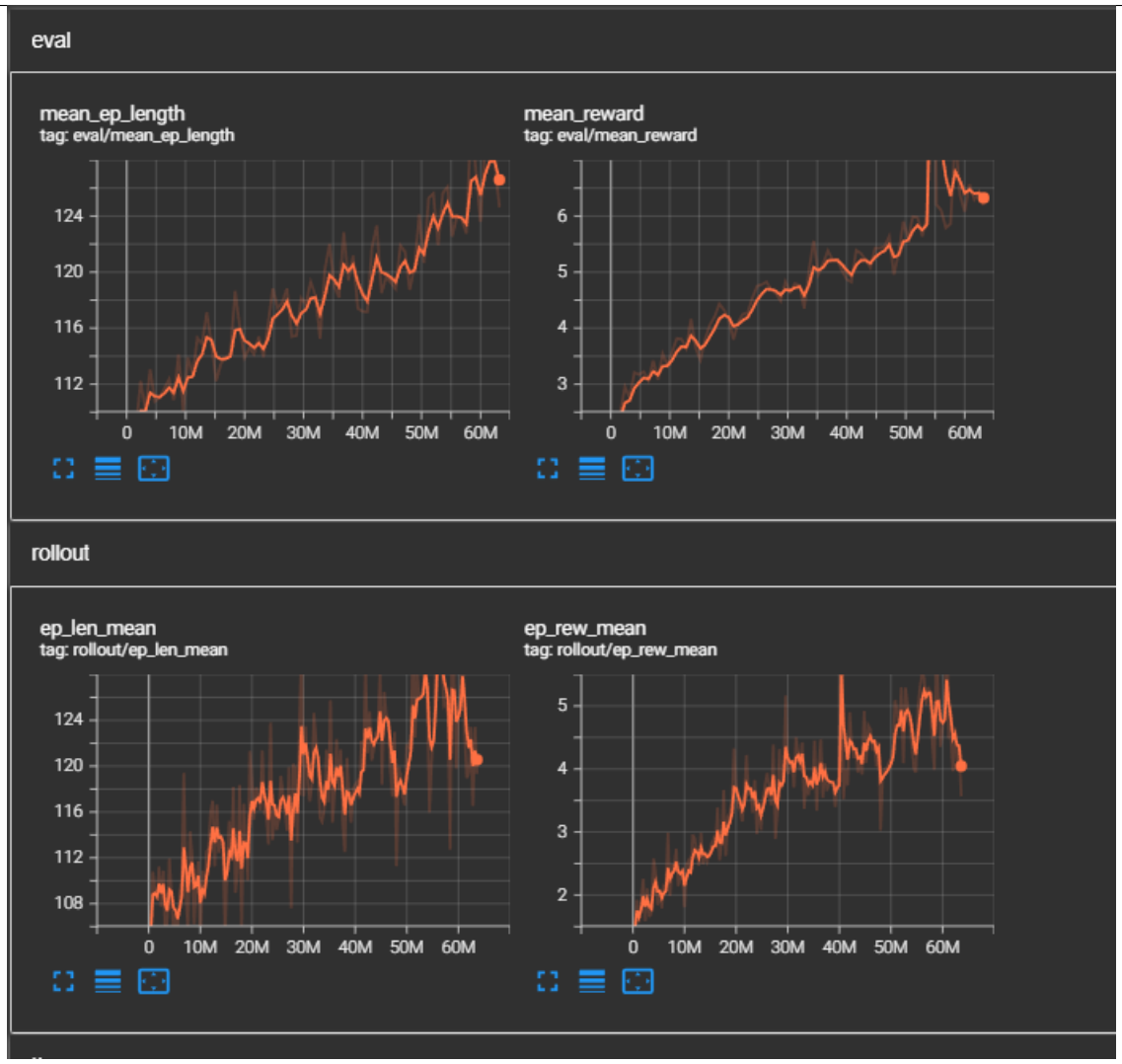

FIGURE 16 – Graphique d'évolution des récompenses et de la longueur moyenne des épisodes pour la phase 3

8 Conclusion

Ce projet de développement d'un agent Rocket League avait pour objectif de créer un agent capable d'atteindre un niveau de jeu au moins similaire à celui d'un joueur débutant. Dans le cadre de notre deuxième année à l'ISIMA, nous avons entrepris cette tâche passionnante pour explorer les applications de l'intelligence artificielle et de l'apprentissage par renforcement dans le contexte des jeux vidéo, en particulier dans Rocket League.

Au cours de ces trois semaines de travail intensif, nous avons réalisé des progrès significatifs dans la conception et le développement de notre agent. Celui-ci est désormais capable de maî-

triser les rudiments du jeu, tels que la navigation vers la balle, la frappe et le marquer dans des situations de jeu relativement simples.

Bien que notre agent actuel joue à un niveau débutant, cela correspond à notre objectif initial pour cette phase de développement. Cependant, nous sommes conscients qu'il reste encore beaucoup à faire pour qu'il puisse rivaliser avec des joueurs plus expérimentés. Des aspects tels que la stratégie de jeu, la défense efficace et la prise de décision en temps réel nécessiteront un développement ultérieur pour améliorer ses performances.

Malgré les défis rencontrés en cours de route, nous avons réalisé des avancées importantes qui témoignent du potentiel de notre approche pour atteindre des performances plus avancées à l'avenir. Notre intention est donc de continuer à affiner notre agent, en explorant de nouvelles techniques d'apprentissage par renforcement et en testant différentes approches pour améliorer ses compétences.

De plus, nous envisageons d'explorer la possibilité d'utiliser l'apprentissage à partir de replays de jeu par des humains pour enrichir l'entraînement de notre agent. Nous avons déjà développé un code permettant de décompiler les replays de Rocket League en données exploitables, mais des recherches supplémentaires seront nécessaires pour intégrer efficacement cette méthode dans notre processus d'entraînement.

Références

[1] R. S. SUTTON ET A. G. BARTO, *Reinforcement Learning : An Introduction-Second edition*. THE MIT PRESS, 2014-2015.

[2] J. SCHULMAN, F. WOLSKI, P. DHARIWAL et A. R. et O. KLIMOV, « Proximal policy optimization algorithms », 2017.

[3] ROCKET LEAGUE GYM, « A python api to treat the game rocket league as an openai gym environment ». [En ligne] Disponible sur : https://rlgym.org. [Consulté le 15/02/2024].

[4] N. WALO, « Seer : Reinforcement learning in rocket league »,

[5] C. S. e. V. H. A. KANERVISTO, « Action space shaping in deep reinforcement learning », *CoRR*, vol. abs/2004.00980, 2020.

[6] A. RAFFIN AND A. HILL AND A. GLEAVE AND A. KANERVISTO AND M. ERNESTUS AND N. DORMANN, « Stable-baselines3 : Reliable reinforcement learning implementations », *Journal of Machine Learning Research*, vol. 22, no. 268, p. 1–8, 2021.

9 Annexe

Site de présentation du processus de création d'un agent Rocket League : https://perso.isima.fr/~maroubille/ZZeer/jeu.html

GitHub : https://github.com/xerneas02/RlGym