The Super Tux Kart piloting project

Dô Amaïlys, Lorenté Matéo, Solar Alexis

Février 2025

Contents

1 Introduction

SuperTuxKart est un jeu de course de karts en 3D open source, qui met en scène les mascottes de divers projets open-source. Le jeu propose des circuits plus ou moins complexes, ce qui en fait un bon environnement pour tester des agents entrainés par renforcement.
Le framework utilisé pour l'implémentation est BBRL, comme dans les TPs. Dans le cadre de ce compte rendu, nous allons faire le tour des méthodes expérimentées et des conclusions que nous pouvons en tirer.

2 Détails sur l'environnement

2.1 Observations

L'environnement donne accès à 101 observations dont 92 continues et 9 discretes qui serviront aux modèles pour prendre des décisions:

Observations continues:

attachment time left: Box(0.0, inf, (1,), float32)
aux ticks: Box(0.0, inf, (1,), float32)
center path: Box(-inf, inf, (3,), float32)
distance down track: Box(-inf, inf, (1,), float32)
energy: Box(0.0, inf, (1,), float32)
front: Box(-inf, inf, (3,), float32)
items position: Box(-inf, inf, (5, 3), float32)
karts position: Box(-inf, inf, (5, 3), float32)
max steer angle: Box(-1.0, 1.0, (1,), float32)
paths distance: Box(0.0, inf, (5, 2), float32)
paths end: Box(-inf, inf, (5, 3), float32)
paths start: Box(-inf, inf, (5, 3), float32)
paths width: Box(0.0, inf, (5, 1), float32)
shield time: Box(0.0, inf, (1,), float32)
skeed factor: Box(0.0, inf, (1,), float32)
velocity: Box(-inf, inf, (3,), float32)

Observations discrètes:

attachment: Discrete(10)
items type: MultiDiscrete([7 7 7 7 7])
jumping: Discrete(2)
phase: Discrete(4)
powerup: Discrete(11)

Ce qui fait bien 101 observations.

2.2 Actions continues vs actions discrètes

Selon les algorithmes utilisés, l'acteur va donner des valeurs d'action discrètes ou continues.

Pour ce qui est des actions discrètes, les environnements "flattened multidiscrete-v0" et "flattened discrete-v0" sont utilisés. Dans les 2 cas, les actions à faire sont:

"acceleration": spaces.Box(0, 1, shape=(1,))
"steer": spaces.Box(-1, 1, shape=(1,))
"brake": spaces.Discrete(2)
"drift": spaces.Discrete(2)
"fire": spaces.Discrete(2)
"nitro": spaces.Discrete(2)
"rescue": spaces.Discrete(2)

où l'acceleration est discrétisée en 5 valeurs possibles et la direction en 7 valeurs possibles. La seule différence entre ces deux environnements étant que l'un est multidiscret (5, 7, 2, 2, 2, 2, 2) et l'autre discret 1120 ce qui correspond aux 1120 possibilités du multidiscret.

Pour les actions continues, l'environnement utilisé est "flattened continuous actions-v0". L'acteur renvoie une valeur dans $[0, 1]$ pour l'acceleration et une valeur dans $[-1, 1]$ pour la direction.

2.3 Wrappers

Nous avons utilisé 2 types de wrappers:

- Un wrapper sur les actions dans le cas continu afin de donner des actions dans $[-1, 1]^2$ au lieu de $[0, 1] \times [-1, 1]$. Cela est plus pratique pour appliquer une activation Tanh sur le vecteur d'actions en sortie du modèle.

- Des wrappers sur les observations pour limiter la quantité d'information et éviter d'avoir des observations inutiles. Par exemple, dans l'environnement continu, les seules actions à faire sont *avancer* et *tourner*, par conséquent, savoir quel objet on a, pour combien de temps, ou si l'on dérape, n'a aucune importance et bruite les observations inutilement.

2.4 Reward

La reward r_t, qui est l'élément central dans l'apprentissage des bons ou mauvais comportements est définie par:

$$r_t = \frac{1}{10}(d_t - d_{t-1}) + (1 - \frac{pos_t}{K})(3 + 7f_t) - 0.1 + 10f_t$$

où d_t est la distance parcourue au temps t, pos_t la position de l'agent parmi les K karts au temps t et f_t qui vaut 1 lorsque l'agent finit la course.

Elle se décompose en 4 parties. Une première, recompensant l'avancée de l'agent sur le circuit. Une deuxième partie, récompensant l'agent d'être dans la meilleure position possible durant la course et à l'arrivée. Une troisième récompensant l'agent pour avoir fini la course, et enfin, une petite partie négative pénalisant l'agent quand il ne reçoit pas de reward.

Remarque : Certaines caractéristiques de la récompense rendent l'entraînement complexe. Tant que l'agent n'est pas suffisamment entraîné, il ne recevra aucune récompense liée à sa position ou au franchissement de l'arrivée. Il cherchera alors naïvement à maximiser la distance parcourue, ce qui conduit à des comportements très simples, comme avancer tout droit le plus longtemps possible. De plus, la grande récompense associée au franchissement de l'arrivée peut également entraîner des comportements étranges, comme nous le verrons par la suite.

3 Expérimentations initiales

La première approche a été d'adapter les différents algorithmes d'entrainement par renforcement vus en cours. En TP, ils ont été testés sur des environnements très simples, tels que Cart Pole ou Pendulum, avec peu d'observations et d'actions à fournir. Cela a permis d'obtenir rapidement de très bons résultats mais la tâche n'est pas aussi simple dans un environnement complexe comme SuperTuxKart. Parmi ces expérimentations, il y a également eu beaucoup de réglages sur les hyper-paramètres.

3.1 DQN

Nous avons d'abord implémenté DQN sur l'environnement discret. Les résultats ont été très décevants, probablement en raison du nombre excessif d'actions possibles, ce qui a conduit l'acteur à converger vers une solution triviale, comme avancer en tournant constamment, quel que soit l'état, afin de maximiser la récompense à court terme.
Nous avons ensuite testé d'autres algorithmes, sans revenir sur DQN.

3.2 DDPG, TD3, SAC

Nous avons ensuite exploré des algorithmes à actions continues pour éviter la complexité des nombreuses actions discrètes, mais cela n'a pas résolu le problème au départ. Les entraînements aboutissaient toujours à des politiques triviales qui empêchaient un progrès efficace. La politique initiale ne permettait pas une exploration suffisante, et le replay buffer se remplissait de transitions où l'agent restait coincé contre le mur, cessant d'apprendre. Ce cercle vicieux freinait considérablement l'entraînement. Voici les résultats d'une partie des expérimentations :

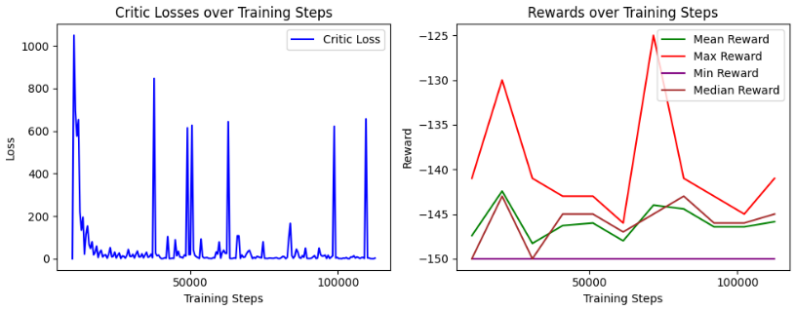

Figure 1: résultats de l'entrainement de DDPG

Figure 2: résultats de l'entrainement de TD3

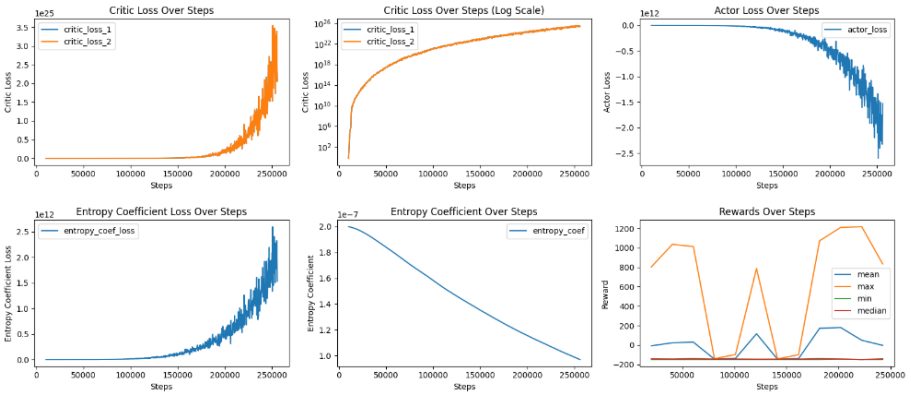

Figure 3: résultats de l'entrainement de SAC avec un acteur plus "petit" que le critique

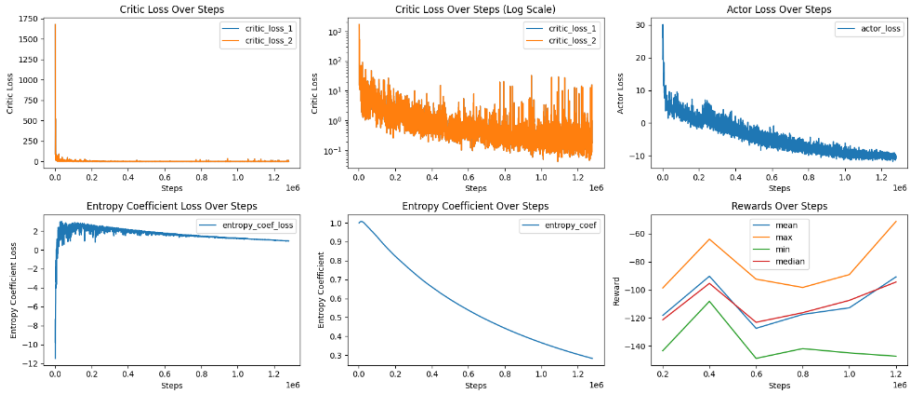

Figure 4: résultats de l'entrainement de SAC avec un acteur de la même taille que le critique

3.3 PPO, TQC

On a également tenté d'implémenter les derniers algorithmes vus en cours et TPs mais cela n'a pas abouti :

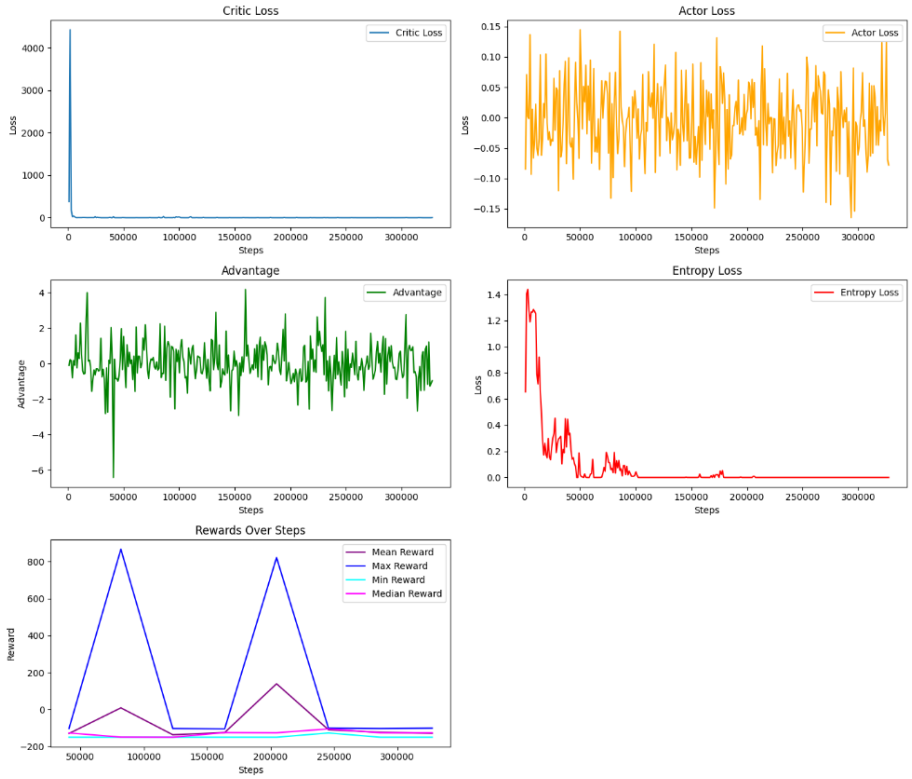

Figure 5: résultats de l'entrainement de PPO

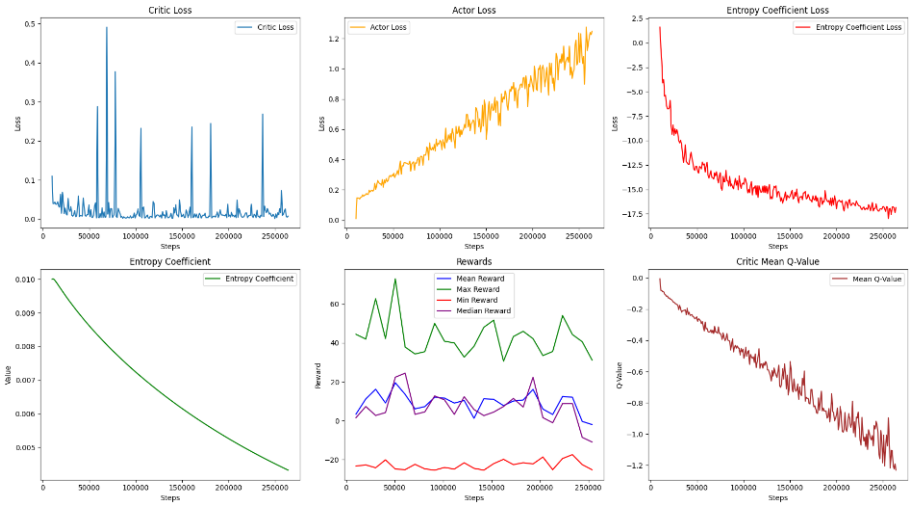

Figure 6: résultats de l'entrainement de TQC

Pour chaque test effectué sur les différents algorithmes, l'acteur peine à gagner en complexité et finit par converger vers des politiques plutôt triviales, cherchant simplement à maximiser les récompenses sans véritable amélioration. Il arrive même parfois que les Q-values divergent. Au fur et à mesure de l'entrainement, la politique converge vers des valeurs limites de l'espace d'actions comme (1, 1) pour l'espace d'actions $[-1, 1]^2$.
Après avoir testé la plupart des algorithmes, nous avions le choix entre prolonger la durée des entraînements, ce qui n'aurait probablement pas été plus concluant en raison de la présence de données non pertinentes dans le replay buffer, ou bien changer d'approche. Nous avons alors tenté des méthodes de régularisation.

4 Régularisations

4.1 Pénalisation dans la loss

Nous avons d'abord introduit une pénalité pour les actions extrêmes ou pour un manque d'accélération, dans le but d'orienter l'acteur vers des politiques favorisant un meilleur entraînement. Cependant, ces méthodes n'ont permis que de contourner le problème sans le résoudre réellement, et elles n'ont pas apporté d'améliorations significatives aux résultats.

4.2 LayerNorm

Nous avons ajouté de la normalisation dans les couches de l'acteur et du critique afin de permettre une meilleur généralisation et un apprentissage plus stable.

Par la suite, nous avons conservé la normalisation, mais abandonné la pénalisation car elle ne résolvait pas le problème et rendait l'apprentissage plus instable. Nous avons finalement opté pour un pré-training à l'aide de behavioral cloning.

5 Behavioral cloning

C'est finalement grâce au behavioral cloning que nous avons obtenu des résultats satisfaisants. L'entraînement, avec une politique initiale trop mauvaise, ne permettait pas une exploration suffisante de l'environnement. Nous avons donc choisi de 'tricher' sur l'exploration en partant d'une meilleure politique initiale, nécessitant moins d'exploration. D'autres techniques, telles que faire débuter l'acteur à des positions aléatoires sur le circuit pour forcer l'exploration, ou augmenter le nombre d'environnements parallèles, sont également envisageables pour améliorer l'exploration, bien qu'elles restent coûteuses en calcul.

Nous avons donc enregistré des transitions (observation, action, récompense, prochaine observation) lors de courses que nous avons jouées nous-mêmes à l'aide de Pygame. Ensuite, nous avons pré-entraîné l'acteur en utilisant ces données pour effectuer une classification, en maximisant la

vraisemblance des transitions observées. Il est également possible d'entraîner le critique, par exemple, en appliquant l'équation de Bellman.

En poussant cette méthode à son maximum, nous pourrions entraîner notre agent sur des exemples de courses 'tool-assisted speedrun', c'est-à-dire des courses exécutées à la perfection. L'agent pourrait ensuite être entraîné à se confronter à d'autres agents. Cela réduirait l'entraînement à l'optimisation de la course, car il ne serait plus nécessaire d'explorer pour trouver les meilleures trajectoires.

Sur PPO, cela n'a pas permis de compenser la trop grande diversité d'action. Il faut faire un wrapper adapté avec le moins d'action possible mais offrant le plus de possibilité tout de même pour éviter les répétitions.
Sur PPO, cela n'a pas permis de compenser la trop grande diversité d'actions. Il est nécessaires de créer un wrapper adapté avec le moins d'actions possibles mais qui offre tout de même suffisamment de possibilité afin d'éviter les répétitions.

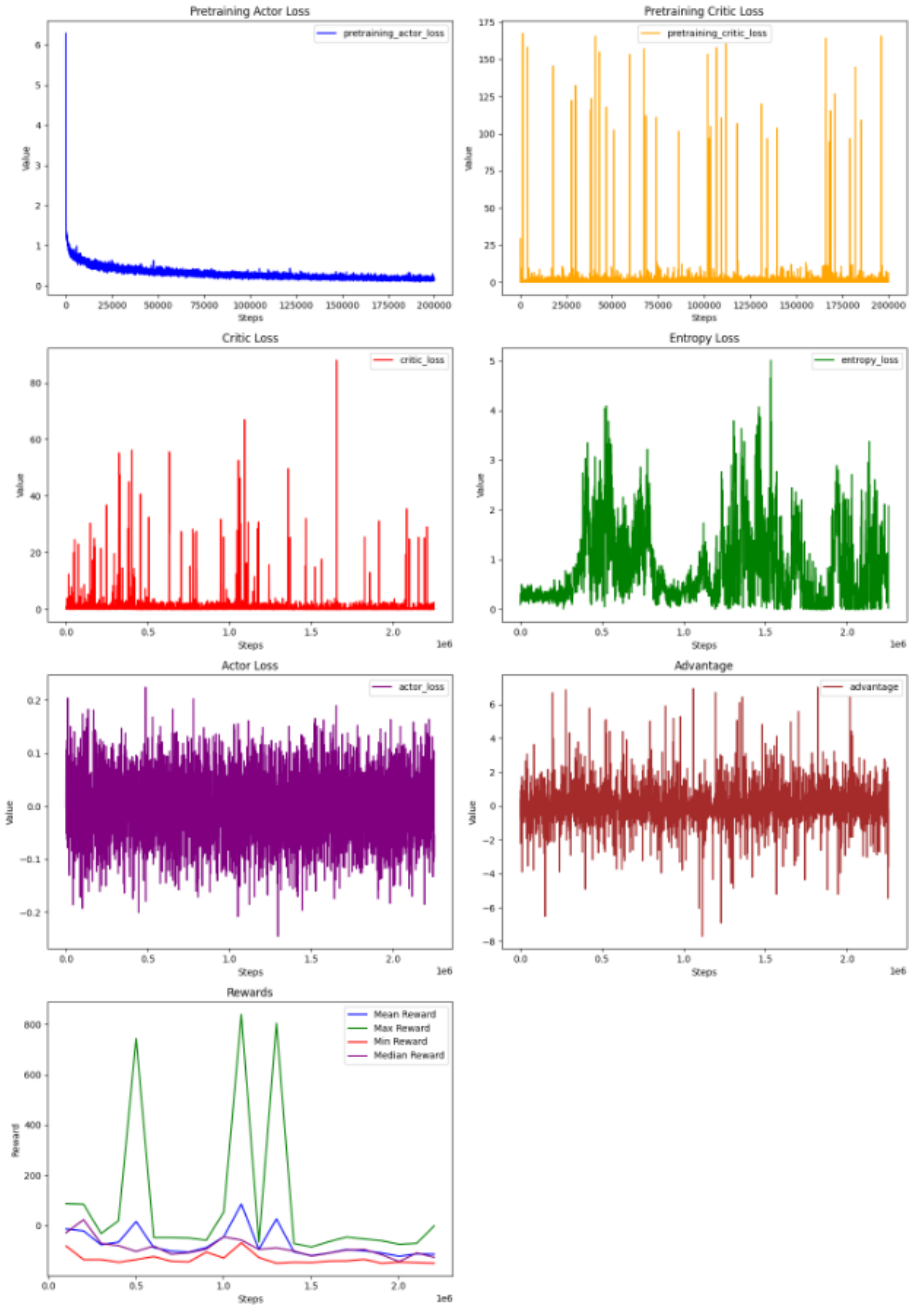

Figure 7: résultats de l'entrainement de PPO avec pré entrainement

En revanche pour TQC, cela nous a permis d'obtenir notre meilleur agent.

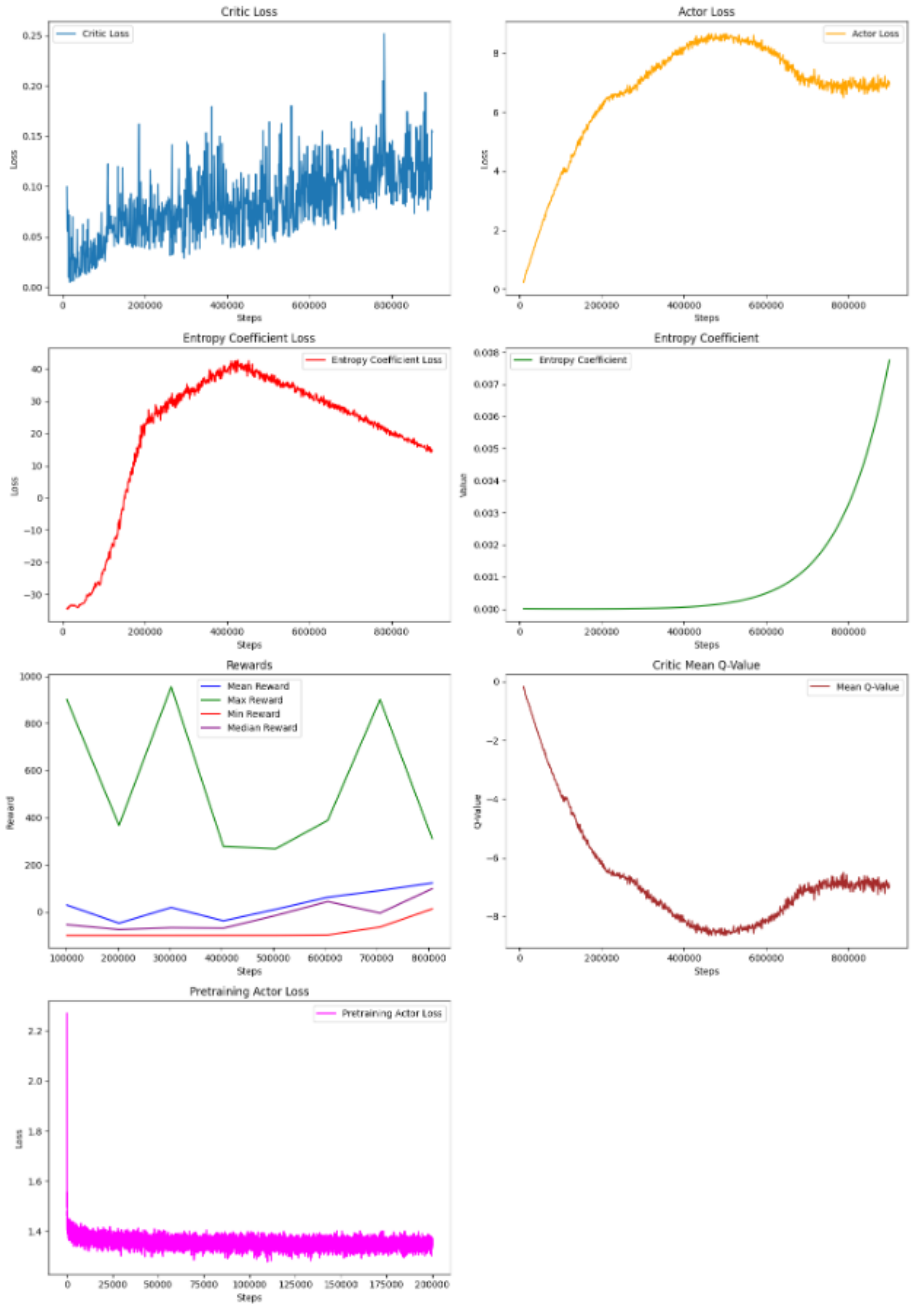

Figure 8: résultats de l'entrainement de TQC avec pré entrainement

Cette approche nous a permis d'obtenir un agent relativement rapide et robuste, bien que la reward très élevée à la fin de la course ait perturbé son comportement. En conséquence, l'agent emprunte des trajectoires absurdes en fin de course, allant parfois jusqu'à percuter le mur avant l'arrivée. Nous pensons que cela est dû à la propagation de la reward élevée, par le critique aux états voisins, ce qui a conduit l'agent à considérer des trajectoires comme bonnes alors qu'elles étaient en réalité sous-optimales En voici un exemple:

Figure 9: l'agent contourne l'arrivée

Figure 10: l'agent finit sa course derrière la ligne d'arrivée

6 Conclusion

Le modèle peut encore être amélioré et certaines pistes d'amélioration ont déjà été évoquées dans ce rapport. D'autres options, comme les wrappers de prévision d'actions ou de mémoire abordés en TP, pourraient également être envisagées. Un wrapper discret, avec peu d'actions mais de nombreuses possibilités, aurait pu être idéal pour d'autres entraînements. Cependant, malgré le petit problème en fin de course, notre agent parvient déjà à terminer la plupart des courses assez rapidement.