Rapport du projet sur DeepRTS - Reinforcement Learning and advanced Deep Learning

Bonnard Jules, Treü Marc

février 2020

Sciences Sorbonne Université

Code disponible sur https://github.com/marc-treu/Projet_RLD

Table des matières

Introduction

Dans ce projet, à partir de l'environnement DeepRTS nous devons dans un premier temps définir des actions et des rewards associé que nous voulons , puis appliquer les algorithmes de Reinforcement Learning vue pendant le cours de RLD afin de créer des agents capables de résoudre les taches choisies.

DeepRTS est une simulation de jeu de stratégie en temps réel (type Warcraft/Starcraft). Bien que cela soit un jeu multijoueur, nous allons dans ce projet nous concentrer sur une variante à un seul joueur, et sur une seule carte FIFTEEN

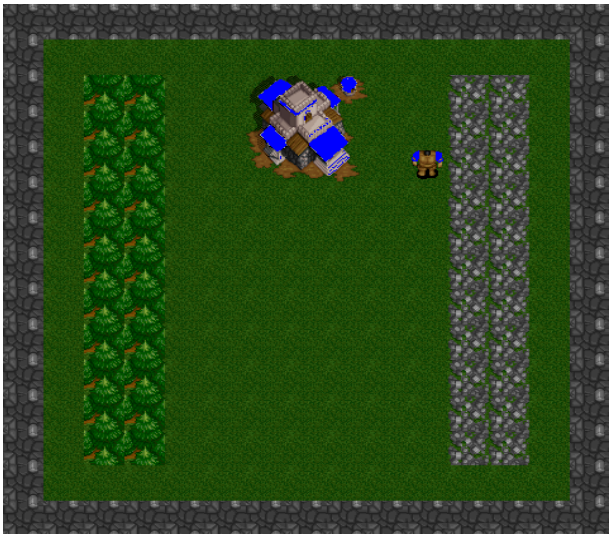

FIGURE 1 – carte FIFTEEN

Les algorithmes considéré

Pour jouer dans cet environnement, on va utiliser les algorithmes suivants : PPO, Dyna-Q et SARSA. On ne pouvait utiliser que des algorithmes pour des environnements discrets, donc pas de DDPG.

Comme première tâche, nous allons reprendre ce qui est proposé par le module, c'est-à-dire ramasser de l'or.

Collecte d'or

Cette tâche d'amasser suffisamment d'une ressource est sans doute la plus simple. La fonction reward, bien que déjà donnée, est évidente. Pour chaque temps t, si l'agent a récupéré une unité d'or, alors l'agent gagne un point, si au contraire, il n'a pas amassé d'or, alors il perd 0.01 point.

Dans la suite, on va tester et comparer plusieurs algorithmes de Reinforcement Learning à deux baselines, une minorante qui sera constituée d'action random, et une majorante qui sera l'optimum. On va aussi tester plusieurs valeurs de ressource à récupérer (10, 50 et 100).

Collecte 10 or

On va commencer par une mission très simple, c'est-à-dire récolter 10 d'or. c'est, dans le cas optimal, réalisable en 122 actions avec un reward de 8,88. Dans le cas random on trouve sur une moyenne de 1000 simulations, 4860 actions et un reward de -38.5.

Dans le graphique suivant, pour les algorithmes considérés, on a l'évolution de leurs rewards en fonction du nombre de simulations.

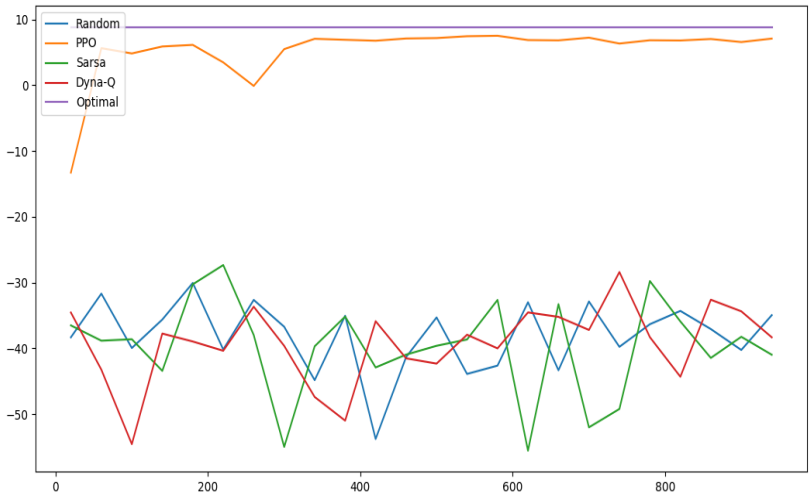

FIGURE 2 – Rewards de tous les algorithmes

Dans le tableau suivant, on va prendre les 100 premières et les 100 dernières simulations pour comparer les algorithmes entre eux, et voir leurs évolutions.

Agent	Reward (premières)	Actions	Reward (dernières)	Actions
Random	-36.722 ±33.17	4682 ±3317	-42.32 ±40.53	5242 ±4053
Dyna-Q	-44.18 ±42.36	5428 ±4236	-40.85 ±34.31	5095 ±3431
Sarsa	-37.27 ±39.38	4737 ±3938	-40.35 ±50.8	5045 ±5081
PPO	-2.376 ±42.643	1230 ±4145	6.88 ±1.06	322.12 ±105.48
Optimal	8.88 ±0	122 ±0	8.88 ±0	122 ±0

À part PPO, les autres algorithmes ne font pas mieux que le hasard sur notre la mission. Il y a clairement un problème d'apprentissage, ces algorithmes avaient des performances correctes sur GridWorld ou sur CartPole. La simulation que l'ont considéré ici, n'a pas un facteur exponentiel de possibilité supplémentaire. Il s'agit d'un problème de mise en oeuvre.

Toutefois, PPO obtient un score très honorable, pour un nombre d'actions 3 fois supérieur à l'optimal. Il arrive aussi très rapidement à converger, puisque comme on a pu l'apprécier sur le premier graphique, après seulement quelques simulations, il est parvenue à cette bonne politique.

Si on compare, PPO à nos deux baselines, on a :

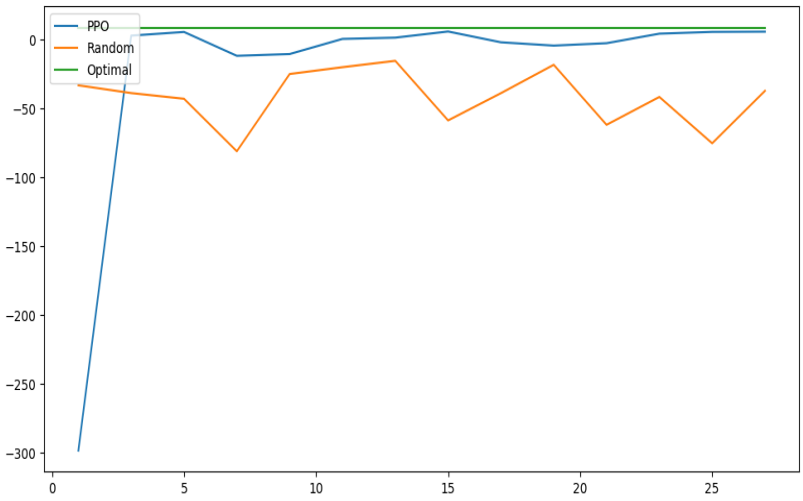

FIGURE 3 – Reward de PPO sur 30 simulations

On voit que PPO converge au bout de 3 simulations, c'est un résultat attendu étant donnée la simplicité de la carte et du problème.

Pour la suite on ne va plus que tester PPO, les autres algorithmes ne donnant pas de bons résultats.

collecte 50 or

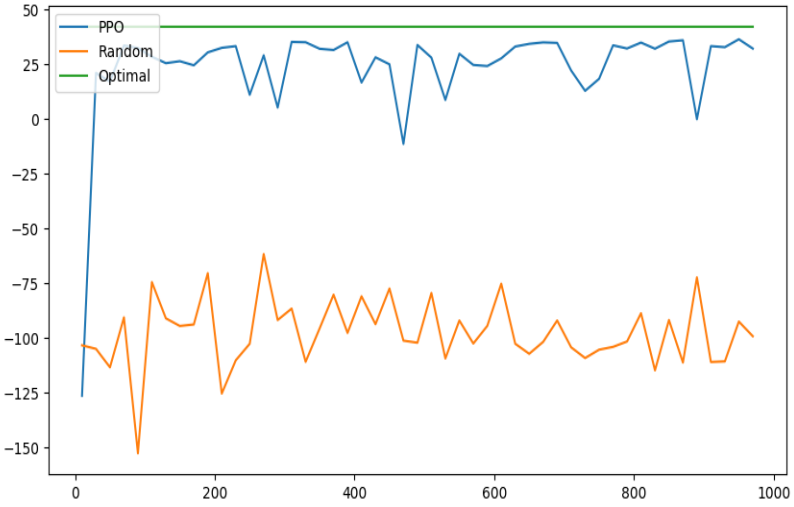

FIGURE 4 – Évolution du Reward de PPO pour 100 d'or

collecte 100 or

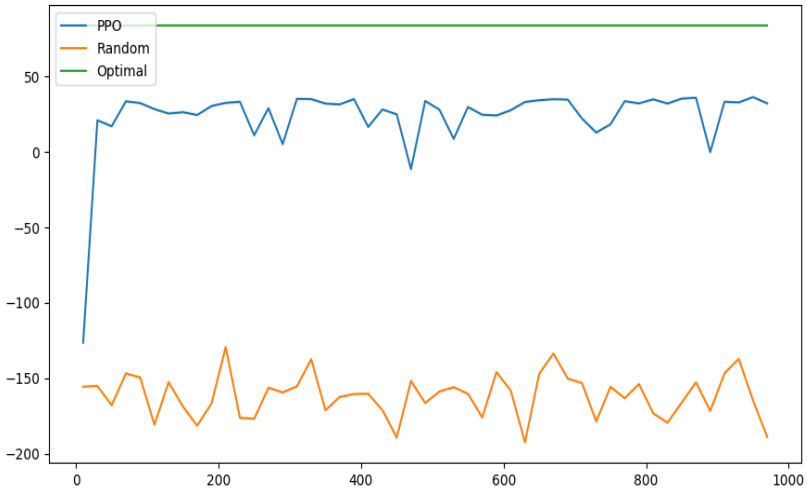

FIGURE 5 – Évolution du Reward de PPO pour 100 d'or

Que cela soit pour 50 ou 100 d'or, PPO apprend une politique en seulement 3 simulations. Le problème est n'évolue pas, comme les performances de PPO.

Collecte de bois

Après être arrivé aux mêmes conclusions que pour l'or, ce qui est naturel vu la symétrie du problème, nous avons décidé de voir si PPO pouvait voir son score moyen se rapprocher de l'optimal. C'est-à-dire que son nombre moyen d'action diminue. On va reprend un problème où il faut collecté 100 unités de bois, mais avec un nombre limité d'actions.

Pour cette mission le score optimal est de 94.49 points de reward pour 880 actions, on va donc limiter à 2000 actions pour voir les performances de PPO.

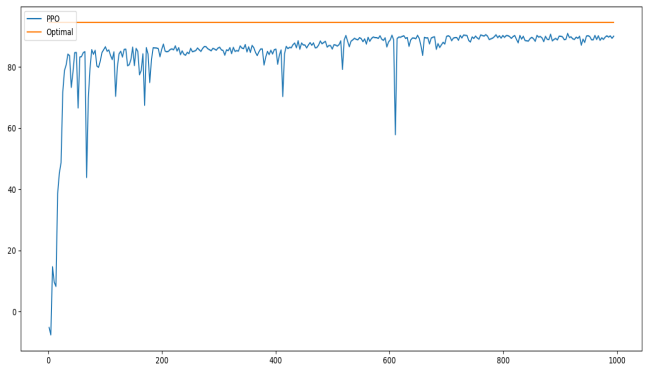

FIGURE 6 – Évolution du Reward de PPO pour 100 d'or

Comme on peut le voir sur la figure ci-dessous, les résultats de PPO tendent encore très rapidement vers la solution optimale. On a aussi considérablement amélioré le score max de notre solution. Puisqu'il est passé de 43.6 à 91.4. PPO a trouvé en quelques centaines de simulations une solution stable est quasi optimale.

On peut conclure en disant que PPO est un algorithme très adapté pour ce genre de problème, simple et où l'on a une connaissance de vers où ce trouve la solution optimale. (Dans ce dernier exemple, il s'agissait de diminuer le nombre d'actions maximales pour voir émerger des solutions de bien meilleurs qualités)

Conclusion

Malheureusement nous n'avons réussi à utiliser que l'algorithme de PPO sur notre simulation, nous somme aussi resté sur des actions de collecte élémentaire. Après plusieurs tentatives pour définir un scénario où l'objectif été de créer plusieurs unités. Aucun algorithmes n'arrivaient à produire quelques choses d'intéressants.

Toutefois, les résultats que nous avons obtenus avec PPO sont intéressants, tant sur la rapidité pour trouver de bonne solution que sur la qualité si l'on contraint un peu son espace de recherche.