



CentraleSupélec

PROJET : REINFORCEMENT LEARNING

CENTRALESUPÉLEC

Ibrahim Rakib

---

# Codage d'un agent de Puissance 4

---

2023

*En groupe avec :*

Saad CHTOUKI - Guilhem PRINCE

Table des matières

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Implémentation</b>	<b>1</b>
2.1	Méthode utilisée . . . . .	1
2.2	Résultats . . . . .	3
2.2.1	Entraînement . . . . .	3
2.2.2	Scores de l’agent . . . . .	3
2.2.3	Choix des paramètres . . . . .	4
<b>3</b>	<b>Conclusion</b>	<b>4</b>

# 1 Introduction

Dans ce projet de Reinforcement Learning, nous essayons deux méthodes afin de coder un agent capable de jouer au Puissance 4. Nous avons conjointement, avec Saad, essayé un apprentissage par Q-Learning, qui vient compléter le contrôleur Monte Carlo élaboré par Guilhem.

Le Q-Learning est une technique d'apprentissage par renforcement qui permet à un agent d'apprendre à prendre des décisions en interagissant avec un environnement, et ce grâce à une table de gains Q, informant sur la récompenses pour chaque action possible au niveau de l'état de l'agent.

Mon approche se base sur une méthode naïve de Q-Learning, qui consiste à construire une table de valeurs Q ne permettant pas une vision complète du jeu pour l'agent. Nous verrons que cette méthode peut présenter des résultats surprenants.

Dans ce papier, je vais décrire les étapes de l'implémentation, en exploitant l'environnement PettingZoo. Je présenterai également une évaluation des performances de l'agent en le confrontant aux adversaires fournis dans `project-strater.ipynb`.

## 2 Implémentation

### 2.1 Méthode utilisée

Le but du Q-Learning est de trouver une matrice multidimensionnelle Q, qui assigne chaque couple (état de l'agent, action possible) à une récompense, traduisant l'intérêt d'une telle action. Pour un état  $s_t$  et une action  $a_t$ , l'évolution de  $Q(s_t, a_t)$  est la suivante :

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{ancienne valeur}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left( \underbrace{r_t}_{\text{récompense}} + \underbrace{\gamma}_{\text{facteur de réduction}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimation de la future valeur optimale}} \right)}^{\text{valeur apprise}}$$

La difficulté de l'implémentation réside ainsi dans le choix de l'état d'un joueur à un moment donné au jeu du puissance 4. Si on considère toutes les situations possibles, le jeu comptant 42 cases, 7 actions étant possibles et chaque case admettant trois possibilités (vide, jeton du joueur 1 ou jeton du joueur 2), il faudrait que la matrice Q soit de taille  $3^{42} \cdot 7$ , ce qui semble peu raisonnable en termes d'espace mémoire.

Nous décidons conjointement avec Saad d'opter pour une méthode dite naïve, sur laquelle nous nous baserons pour élaborer une méthode plus complexe (Cf. Rendu de Saad Chtouki), qui consiste à considérer l'état d'un joueur comme la dernière position  $(x, y)$  qu'il a joué,  $x$  et  $y$  étant respectivement dans  $[0, 5]$  et  $[0, 6]$ . Cette idée d'implémentation est bien entendu limitée, puisque l'agent n'a pas une vision complète du déroulement de la partie, il n'a même pas une vision sur les coup de son adversaires mais agit uniquement en fonction de son dernier coup. On peut parler d'agent à l'aveugle. L'objectif est de vérifier si ce type d'agent peut battre les agents fournis.

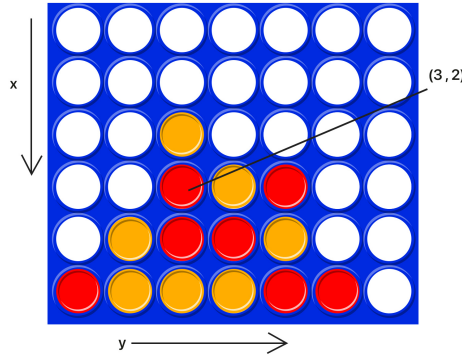


FIGURE 1 – Etat = Position x,y du dernier coup joué

Une entité de la classe `Player` présente alors les attributs suivants :

1. Un identifiant `_id`, un nom `name` conçu à partir de `_id` et un booléen `left` indiquant si notre agent est un `left_player`.
2. Une matrice `Q` de taille  $(6, 7, 7)$ .
3. Un état `st` (couple  $(x,y)$ ).
4. Un "facteur d'exploration" `epsilon` correspondant à l'approche *Epsilon-Greedy* : Il s'agit de la probabilité, pour l'agent, de choisir un coup au hasard à chaque tour. On parle ainsi de facteur d'exploration puisque l'agent, au lieu de choisir l'action maximisant la récompense, choisit un coup au hasard afin d'explorer certaines situations. Il est indispensable que `epsilon` soit supérieur à 0 en phase d'entraînement, afin que l'agent essaye de nouvelles voies, et ne joue pas tout le temps de la même manière. Notons que si `epsilon` = 1, notre agent devient un agent Random.

Nous avons co-écrit les fonctions relatives au fonctionnement de l'agent avec Saad (puisque celles-ci servent en majorité aussi pour sa partie). Voici une présentation de l'enchaînement de celles-ci :

```
def play_game(env, agent0, agent1, alpha, gamma, display=False): #e epsilon coeff d'exploration
    done = False
    env.reset()
    obs, _, _, _ = env.last()
    l=[agent0, agent1]
    random.shuffle(l)
    while not done:
        #print("")
        #print("")
        for i, agent in enumerate(l):
            last_state=agent.st
            e=agent.epsilon
            action = agent.get_action(obs, epsilon=e) #Epsilon = taux d'exploration #Il s'agit de obs du point de vue de l'autre joueur mais on ve
            env.step(action) #Préciser à l'environnement pettingZoo que l'action prise
            obs, reward, terminated, _, _ = env.last()
            agent.update_state(action, obs, terminated) #Mise à jour de l'état de l'agent

            if display:
                clear_output(wait=True)
                plt.imshow(env.render())
                plt.show()
            done = terminated #Déroulement fini
            if np.sum(obs["action_mask"]) == 0: #Vérification d'une éventuelle égalité
                if display:
                    print('Draw')
                done=True
                break

            if last_state != (None, None): #Mise à jour des récompenses dans Q
                agent.update_Q(last_state, obs, action, reward, alpha, gamma, show=False)
```

Récupération de l'action maximisant la récompense dans Q

Mise à jour de l'état (nouvelles position jouée)

S'il n'y a pas match nul, on met Q à jour

FIGURE 2 – Fonction `play_game()`

## 2.2 Résultats

Avant d'étudier les résultats, voici quelques remarques préliminaires que l'on peut noter afin d'expliquer certains choix d'implémentation :

1. Lorsque l'on fait jouer deux randoms l'un contre l'autre un grand nombre de fois, on comprend que le joueur qui commence a environ 55% de chances de l'emporter (et non 50%). **Commencer présente donc un avantage**, c'est pourquoi dans la fonction `play_game()`, le joueur qui commence est tiré au sort.
2. Il a fallu faire un choix sur le partenaire d'entraînement de notre agent (un random, un `left_player` ou un agent similaire au notre qui apprend aussi). Nous avons remarqué avec Saad que les résultats sont légèrement meilleurs lorsque l'agent affronte un joueur random qui n'apprend pas lors de la phase d'entraînement, les entraînements sont donc ainsi réalisés contre un joueur random (`epsilon=1`). La meilleure performance peut être expliquée par la multitude de situations rencontrées en jouant contre un random, et donc la richesse de la matrice  $Q$ .

### 2.2.1 Entraînement

L'entraînement de l'agent consiste en **20.000** parties contre un joueur random avec un facteur d'exploration `epsilon=0.3` (afin que l'agent puisse assez souvent mettre sa matrice  $Q$  à jour sans pour autant jouer tout le temps de la même manière puisque qu'il explore 30% du temps). Les paramètres sont les suivants :

1. `alpha=0.01`
2. `gamma=0.5`

Le choix d'alpha et gamma sera justifié plus bas. Voici par exemple un aperçu des valeurs de la matrice multi-dimensionnelle  $Q$  pour l'état (5,0) (c'est à dire si on a joué en bas à gauche) :

[5,43e-3, 1.02e-2, 9.06e-3, 1.99e-2, 9.68e-2, **1.67e-1**, 1.33e-2]

FIGURE 3 – Récompenses pour chaque action lors de l'état (5,0), 20.000 parties d'entraînement

Ainsi, lorsque l'agent est dans l'état (5,0), la récompense de l'action 0 (ie jouer tout à gauche) est de  $5,43e-3$ . La meilleure action à jouer est l'action 5 (à savoir deuxième colonne en partant de la droite).

### 2.2.2 Scores de l'agent

L'évaluation de l'agent consiste aussi en **20.000 parties**. On relève le nombre de victoires de l'agent lors de ces dernières. Voici les résultats :

Adversaire affronté	Nb de victoires	Nb de matchs nuls	Pourcentage de victoires
Random_player	17346	6	86,7%
left_player	10057	7	50,2%

L'agent est donc performant contre un joueur aléatoire, ce qui est étonnant puisque l'agent codé à un point de vue très limité sur la partie en cours (connaissance uniquement du dernier coup joué). Il est néanmoins moins performant contre le **left\_player**, même s'il s'entraîne contre ce dernier. Cela peut s'expliquer par le fait que la stratégie du **left\_player** est facile à contrer si on a accès à ses coups, ce qui n'est pas le cas de notre agent, qui prend uniquement en compte que son propre coup le plus récent.

### 2.2.3 Choix des paramètres

Une recherche de grille fut implémentée afin de déterminer les paramètres **alpha** et **gamma** optimaux. On affiche dans le tableau les pourcentages de victoire contre un adversaire aléatoire sur **10 000 parties** après entraînement :

	alpha = 0.001	alpha = 0.01	alpha = 0.1
gamma = 0.5	61.5%	76.8%	76.5%
gamma = 0.9	59.3%	70.7%	76.8%

#### PERFORMANCE DES DIFFÉRENTS COUPLES DE PARAMETRES

Nous avons ainsi choisi **alpha**=0.01 et **gamma**=0.5. Notons que lors de l'évaluation sur 20.000 parties, l'agent a été encore plus performant (86.7% de victoires contre le joueur aléatoire comme évoqué plus haut).

## 3 Conclusion

La méthode naïve implémentée, bien qu'à priori sans intérêt puisque trop limitée, se révèle performante contre un joueur aléatoire (victoire dans 86.7% des cas lors de l'évaluation) mais demeure limitée contre le **left\_player**, probablement car le jeu de ce dernier est redoutable lorsque son adversaire n'a pas accès à ce qu'il joue. Ces résultats sont à l'opposé de ceux de l'agent contrôlé par méthode de Monte Carlo implémenté par Guilhem, performant contre le **left\_player** mais pas contre le joueur aléatoire. Dans le rendu de Saad, figure une piste d'amélioration de la méthode présentée ici, ou l'état ne prendra plus en compte uniquement les coups de l'agent mais aussi ceux de son adversaire.