TP Morpion

Sylvain Gault

9 septembre 2024

1 Introduction

Ce TP est à réaliser par trois en Python sous l'environnement de développement de votre choix. Il ne sera pas à rendre.

Dans ce TP vous écrirez un programme qui applique les principes de l'apprentissage par renforcement au jeu de Morpion afin de trouver une stratégie optimale. On supposera que c'est toujours le joueur `X` qui commence.

2 Le morpion

Exercice 1 Une classe TicTacToe

Le but de cet exercice est d'implémenter une classe `TicTacToe` pour faciliter la suite du TP. Elle gérera uniquement la grille. C'est votre `main` qui va faire les interactions avec l'utilisateur.

1. Écrivez une classe `TicTacToe` qui initialise une grille 3x3 vide dans son constructeur. Stockez également à quel joueur c'est le tour de jouer.
2. Écrivez une méthode `play` qui prend en paramètre un numéro de case entre 0 et 8. Cette méthode pourra lever une exception si la case est occupée. Sinon elle remplit la case et passe au joueur suivant.
3. Écrivez une méthode `__str__` afin d'afficher la grille à chaque tour de boucle. Elle ne servira que pour vos tests.
4. Écrivez une méthode `has_winner` qui renvoie un booléen indiquant si le dernier joueur a avoir joué a gagné.
5. Écrivez une méthode `is_draw` qui indique si c'est un match nul.
6. Écrivez une méthode `reset` qui réinitialise la grille et le premier joueur.
7. Écrivez une méthode `undo` qui défait le dernier coup joué. Par simplicité, vous pourrez passer la coordonnée du dernier coup joué à la méthode `undo`. Vous n'en avez normalement pas besoin en jeu normal. Mais ça vous sera utile par la suite.
8. Rajoutez enfin une méthode (ou propriété) `allowed_moves` qui liste les numéro de cases encore disponibles.

3 Apprentissage par renforcement

Exercice 2 Jouer !

Dans cet exercice, on implémente les simulations de jeu entre notre IA par renforcement et un joueur qui ne voit pas plus loin qu'au prochain coup.

1. Implémentez une fonction `opponent_random` qui prend en argument une instance de `TicTacToe` et joue un coup au hasard parmi ceux autorisés.
2. Implémentez une fonction `opponent_next` qui choisi parmi les coups autorisés. Il choisit en priorité un coup qui le fait gagner. Si elle ne peut pas gagner, elle joue préférentiellement un coup qui empêche l'adversaire de gagner. Sinon, elle joue au hasard.
3. Faites jouer ces deux IA ensemble pour vérifier qu'elles fonctionnent.
4. Créez une *« value function »* sous la forme d'un dictionnaire python. Celui-ci sera indexé par les états du jeu et contiendra la valeur associée à cet état. Étant donné que les états inconnus doivent avoir la valeur `0.5`, vous pourrez utiliser la classe `defaultdict` contenue dans le module `collections`.
5. Pour pouvoir utiliser une instance de `TicTacToe` comme index de dictionnaire, il faut que cette classe soit `hashable`. Implémentez une méthode `__hash__` dans votre classe `TicTacToe`. Celle-ci devra renvoyer un entier unique pour chaque état du jeu. Pour cela, construisez un entier ou une chaîne de caractère qui représente la grille et appelez la fonction `hash` dessus. Note : vous pouvez réutiliser `__str__`, mais ses performances laisseront probablement à désirer.
6. Écrivez une fonction `greedy_move` qui, étant donné la *« value function »* et un état du jeu renvoie le coup à jouer qui a la plus grande valeur. En cas d'égalité, vous choisirez aléatoirement parmi tous les coups qui ont le score max.
7. Écrivez une fonction `play_game` qui prend en paramètre la *« value function »* et la fonction de l'adversaire (soit `opponent_random` soit `opponent_next`) et joue une partie complète de manière entièrement *« greedy »*.
8. Vérifiez que votre code ne plante pas et joue de manière totalement aléatoire. Vous devriez trouver que le premier joueur gagne 59% du temps, le second 29% et 12% de match nuls.

Exercice 3 Temporal-Difference

On va ici enfin pouvoir implémenter la mise à jour de la *« value function »*.

1. Modifiez `play_game` pour mettre à jour la *« value function »* à 0 ou 1 si la partie est terminée.
2. Modifiez encore cette fonction pour mettre à jour la *« value function »* après chaque coup `greedy`. En utilisant la formule de la *« temporal-difference »*. Vous pourrez choisir une taille de pas assez petite, par exemple 0.1.
3. Question : Que se passe-t-il si on sélectionne uniquement les coups de manière `greedy` ? Est-ce que cette IA jouerait nécessairement moins bien ? Nécessairement mieux ? Est-ce qu'une autre valeur initiale que 0.5 pourrait remédier à ce problème ?
4. Implémentez les coups *« exploratoires »* avec une probabilité de 5%. Dans ce cas, il ne faut pas mettre à jour la *« value function »* pour ce coup.
5. Question : que se passerait-il si on mettait à jour la *« value function »* aussi pour les coups

exploratoires ? Quelles probabilité notre *« value function »* serait-elle en train de calculer ? Quelles probabilités calcule-t-elle sans apprendre pendant les coups exploratoires ? Lequel est meilleur ?

6. Note : vous pourrez utiliser le module `pickle` afin de sauvegarder votre *« value function »* après entraînement afin de pouvoir la réutiliser de manière interactive si vous le souhaitez.
7. Question : que se passerait-il si les deux joueurs utilisent la même stratégie de RL (qu'on est en train d'implémenter) au lieu de jouer contre un joueur aléatoire ? Est-ce que la *« value function »* apprise serait différente ?
8. Testez ce cas en ayant deux joueurs, chacun avec sa propre *« value function »*. Note : vous devrez probablement créer une fonction `play_game2` au lieu de tenter de modifier `play_game`.
9. Question : Les grilles de morpion sont très symétriques et beaucoup d'états sont en fait identiques. Que pourrait-on faire (théoriquement) pour améliorer le processus d'apprentissage ? Quelle effet cela aurait ? Et si l'adversaire n'avait pas une stratégie symétrique et privilégiait par exemple le côté droit ? Est-ce que les états identiques à une symétrie près devraient vraiment avoir une même valeur ?
10. Il y a d'autres améliorations faisables. Notamment pour accélérer le temps d'entraînement. Proposez-en quelques unes et implémentez-les.