Division des enseignements en informatique

TP MiniMax

Dans ce TP, nous nous proposons d'implémenter une intelligence artificielle pour un jeu de Morpion utilisant l'algorithm MinMax. Dans un premier temps, nous écrirons les fonctions permettant de réaliser la boucle de jeu. Ensuite nous créerons une intelligence artificielle afin de l'affronter dans notre jeu de Morpion.

1 Morpion

But du jeu : deux joueurs s'affrontent sur une grille carré (généralement de taille 3*3 cases). A tour de rôle, chaque joueur place une marque sur la grille de jeu. Le premier joueur à aligner horizontalement, verticalement ou en digonale un nombre de marques égal à la longueur de la grille remporte la partie.

Nous allons créer une version du Morpion dont la taille de la grille pourra varier.

La grille de jeu sera représentée par une liste de listes, tel que `grille[x][y]` corresponde à la case de la x^{ieme} ligne, y^{ieme} colonne et `grille[0][0]` corressponde à la case en haut à gauche de la grille. Les valeurs des cases pourront être :

- `' '` pour une case libre
- `'x'` pour une case occupée par le joueur 1
- `'o'` pour une case occupée par le joueur 2

1.1 Interaction

1. Ecrivez une fonction `affichage(grille)` qui permet d'obtenir un affichage de la grille du type :

```
*-------*
| x o   |
| o o x |
|   x   |
*-------*
```

2. Rédigez une fonction de saisie utilisateur permettant de récupérer le coup joué par un joueur. La saisie se fera en tappant la ligne (`x`) suivie d'une virgule puis la colonne (`y`) : `x,y`. On pourra utiliser la fonction `split` des chaînes de caractères pour récupérer les deux coordonnées. On renverra finalement le coup choisi par l'utilisateur sous forme de `tuple` : `(x,y)`. Les erreurs de saisie du joueur seront gérées en utilisant le mécanisme des **interruptions**. On prendra notament en compte les interruptions de type `ValueError` lorsque la syntaxe de la saisie n'est pas bonne et `IndexError` lorsque le joueur souhaite placer sa marque hors des limites de la grille.

1.2 Mécanique

3. Implémentez une fonction `coups_possibles(grille)` qui renvoie un `tuple` comportant tous les coups potentiels qui peuvent être joués sur une grille donnée.

1.3 Boucle de jeu

Nous allons maintenant réaliser la fonction principale de notre programme permettant de gérer la boucle de jeu.

Dans le fichier *mecanique.py*, vous trouverez la fonction d'évaluation de la grille de jeu. Cette fonction prend comme argument une grille, et renvoie sous forme de `tuple` les marques présentes dans la grille formant un alignement de `n` cases verticales, horizontales ou diagonales. On peut ainsi déterminer la condition de fin du jeu. La fonction renvoie un tuple vide s'il n'y a aucun alignement gagnant de marques.

L'architecture de la boucle de jeu sera la suivante :

— 1) Création de la grille
— 2) *Début de la boucle* Affichage de la grille de jeu
— 3) Récupération du coup du joueur actif
— 4) Ajout du coup du joueur actif à la grille
— 5) Evaluation de la grille
— 6) Si l'un des joueurs a gagné, fin de la boucle, sinon étape suivante
— 7) Changement du joueur actif et retour à l'étape 2)

4. **En réutilisant les fonctions précédentes**, écrivez une fonction `jouer(taille_grille)` qui lance une partie de Morpion à deux joueurs sur une grille de taille `taille_grille*taille_grille`. On vérifiera que le coup proposé par le joueur fait bien parti de la liste des coups valables à chaque tour.

2 IA

Maintenant que notre jeu de Moripion est fonctionnel, nous allons remplacer l'un des joueurs humains par un adversaire qui sera controlé par l'ordinateur.

2.1 Coup aléatoire

5. Ecrivez une fonction `coup_aleatoire(grille)` retournant un coup valable aléatoire choisi parmi les coups possibles dans la grille.
6. Modifiez votre fonction `jouer` de telle sorte que l'un des joueurs ne soit plus controlé par un humain, mais choisisse un coup de façon aléatoire à chaque tour.

3 Algorithme MiniMax

La sélection aléatoire nous permet de générer un semblant d'intelligence chez l'adversaire controlé par l'odinateur, cependant, il existe des métodes bien plus performantes !

Nous nous proposons dans ce TP d'implémenter l'agorithme *MiniMax* pour déterminer quel coup sera le plus intéressant à jouer pour l'adversaire artificiel.

"L'algorithme minimax est un algorithme qui s'applique à la théorie des jeux pour les jeux à deux joueurs à somme nulle (et à information complète) consistant à minimiser la perte maximum (c'est-à-dire dans le pire des cas)." (Wikipedia)

Principe de l'agorithme : l'agorithme MiniMax considère tous les coups possibles pour une grille donnée, puis choisit celui qui est le plus bénéfique au joueur. Pour évaluer quel coup est optimal, l'agorithme considère tout les coups possiblement réalisables par l'adversaire pour chaque coup réalisé par le joueur, et choisit celui qui est le moins avantageux pour le joueur (on part du principe que l'adversaire va jouer le coup le plus optimal pour lui). Et ainsi de suite jusqu'à une profondeur donnée ou jusqu'à ce qu'un des deux joueurs ait gagné.

En d'autres termes, l'idée de l'algorithme est de développer complètement l'arbre de jeu (c'est à dire tous les coups possibles à chaque tour de jeu), d'associer à chaque feuille de l'arbre une valeur (qui évalue l'état de la grille au niveau de la feuille au regard du joueur), puis de faire remonter ces valeurs avec l'hypothèse que chaque joueur choisit le meilleur coup pour lui.

Fonction d'evaluation : dans un premier temps, pour évaluer la grille à un instant donné nous choisirons trois valeurs : 1 si la grille est gagnante pour le joueur, -1 si l'adversaire gagne et 0 sinon.

3.1 Exemple

Suppossons que l'on soit à un instant t de la grille, associé à la configuration suivante, et que ce soit au joueur `x` de jouer :

```
*-------*
|   x   |
| x o o |
| o x   |
*-------*
```

L'arbre des coups possibles pour cette grille correspond à la figue 1 :

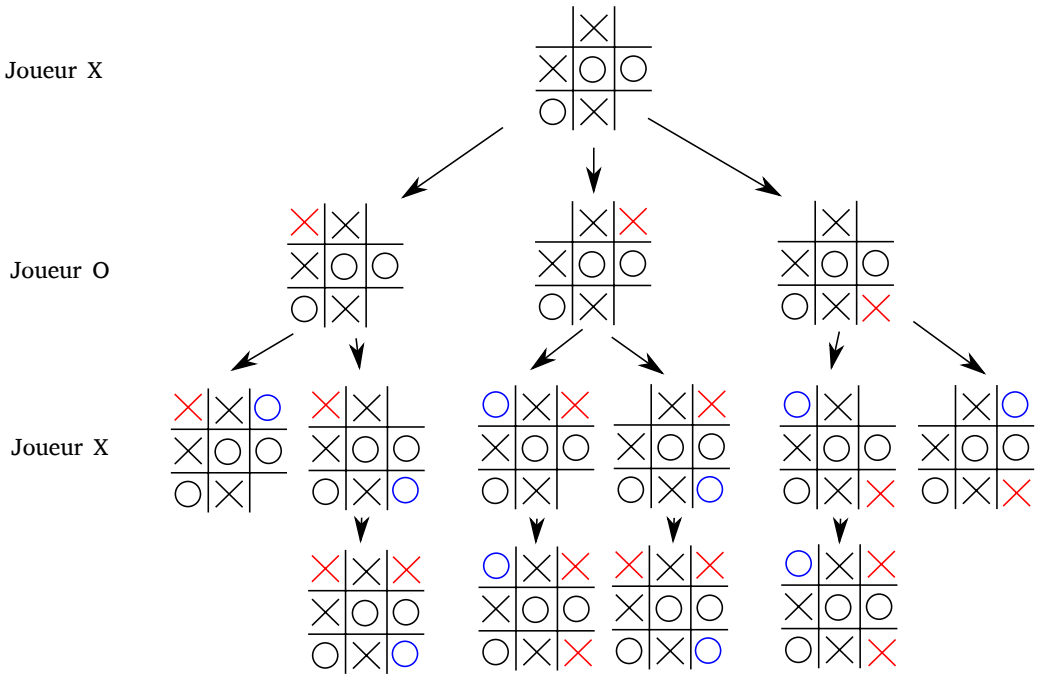

FIGURE 1 – Arbre de jeu

On peut ensuite évaluer la grille à chaque feuille (cf. figure 2) de l'arbre.

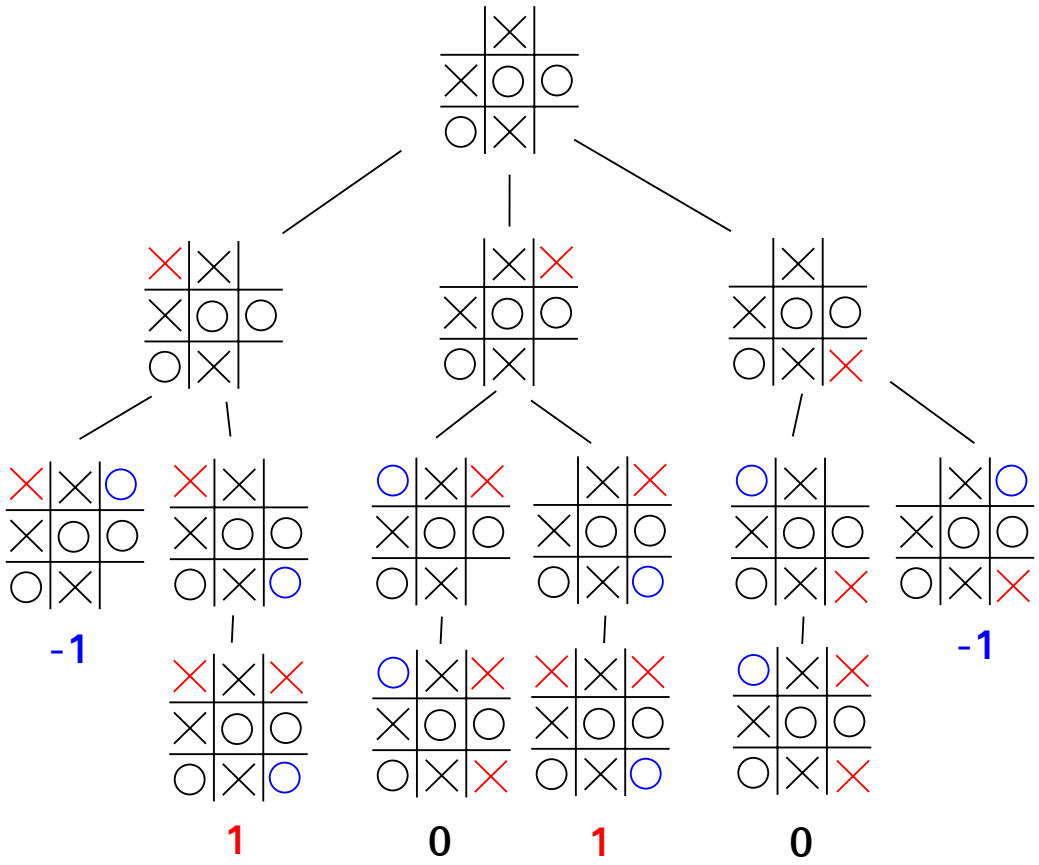

FIGURE 2 – Score des grilles

On fait ensuite "remonter" les valeurs vers le haut de l'arbre, en choisissant la plus faible si c'est à l'adversaire de jouer (le plus mauvais coup pour le joueur corresond au meilleur coup pour l'adversaire) et la plus grande si c'est au joueur de jouer (cf. figure 3) :

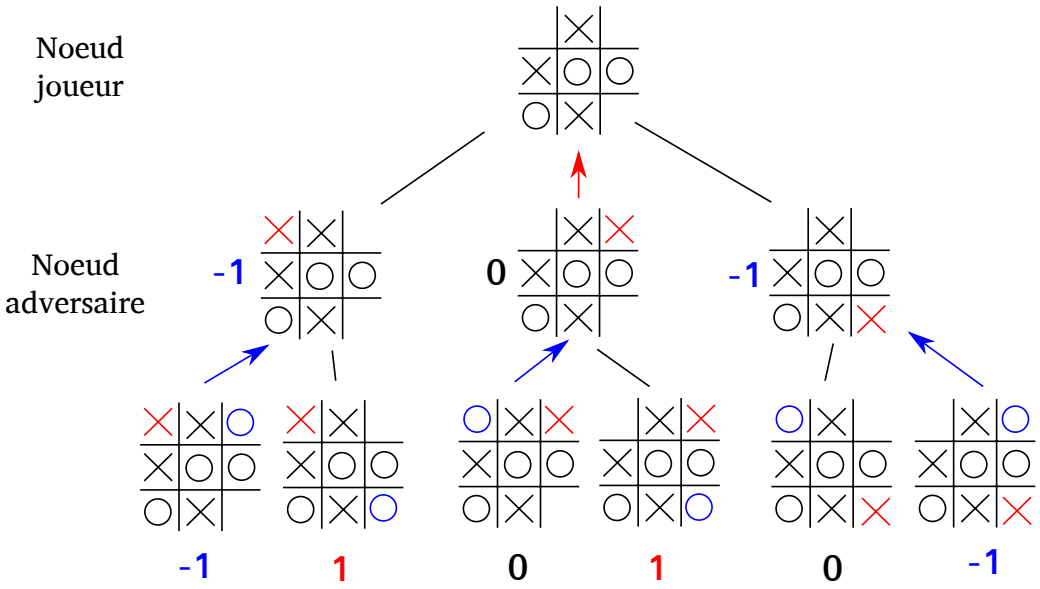

FIGURE 3 – Score des grilles

Le coup optimal est donc :

```
*--------*
|    x x |
| x o o  |
| o x    |
*--------*
```

En effet, c'est le seul qui nous permette à coup sur de ne pas perdre si l'adversaire joue convenablement.

Dans cet exemple, nous avons simulé 3 tours de jeu : nous dirons donc que l'agorithme MiniMax a été utilisé avec une **profondeur** de 3.

3.2 Implémentation

7. Ajoutez une fonction `evaluation(grille, marque)` qui permet d'évaluer une position de la grille par rapport au joueur possédant la marque passée en argument.
8. Créez une fonction `jouer_coups(grille, coups)` qui renvoie une liste composée de plusieurs grilles qui correspondent à l'évolution de la grille initiale si l'on avait joué les `coups` passés en argument. **Attention**, la modification de la grille à l'intérieur de la fonction la modifiera également à l'extérieur, parce que c'est une liste ! Il faudra donc la *copier* pour chaque coup joué. On utilisera donc la fonction `deep_copy` du package `copy` :

```
import copy

grille = creer_une_nouvelle_grille()
copie_de_grille = copy.deep_copy(grille)
```

9. Ecrivez la fonction `minimax_p1(grille,marque)` qui va exécuter l'algorithme MiniMax mais avec une profondeur de **1**, c'est-à-dire que l'on va simplement évaluer la grille de jeu après le tour du joueur et choisir la meilleur option. Vous pourrez (*devez !*) réutiliser les fonctions des parties 1 et 3 !
10. Testez votre nouvelle intelligence artificielle dans votre fonction de jeu !
11. Améliorez la fonction précédente pour que l'algorithme calcule le coup optimal avec une profondeur donnée. Le pseudo-code de la fonction d'évaluation complète est le suivant :

```
fonction minimax(grille, profondeur, tour_adversaire)
    Si profondeur = 0 ou grille est une grille final
        retourne evaluation de la grille

    Si tour_adversaire est FAUX
        Pour chaque coup possible de la grille
               nouvelle_grille := grille modifie par coup
               v := minimax(nouvelle_grille, profondeur - 1, VRAI)
        meilleur_score := max(v)
        retourne meilleur_score

    Sinon    (* tour adverse *)
        Pour chaque coup possible de la grille
               nouvelle_grille := grille modifie par coup
               v := minimax(nouvelle_grille, profondeur - 1, FAUX)
        meilleur_score := min(v)
        retourne meilleur_score
```

Il s'agit d'une fonction **récursive**, c'est-à-dire qui est suceptible de s'appeler elle-même ! Au moment de l'implémentation, faites bien attention à traiter les cas d'arrêts (profondeur = 0 ou fin de jeu) pour ne pas tomber dans des appels récursifs infinis. *Python limite la quantité d'appels récursifs à 999 et renverra une erreur sinon.*

12. Testez votre fonction pour différentes profondeurs et différentes tailles de grille. Que constatez vous ?

4 Aller plus loin

13. Proposez une amélioration de la fonction d'évaluation de l'agorithme MiniMax. On pourra notament rajouter une part d'aléatoire dans le cas ou plusieurs coups possèdent le même score, ou trouver un moyen d'évaluer une grille qui ne comporte aucun gagnant autrement qu'en y associant le score 0.
14. Modifiez votre code pour pouvoir jouer au puissance 4 à la place du Morpion. Certaines fonctions n'auront pas du tout besoin d'être modifié ou très légérement : les deux jeux sont en réalité très proches !
15. Confrontez votre IA avec celles de vos camarades (modifiez la fonction `jouer` pour que deux ordinateurs s'affrontent entre eux).