TP n°7 - Algorithme Minmax pour le puissance 4

Notions abordées

- modélisation du jeu Puissance 4
- algorithme MinMax
- fonction heuristique

Exercice 1 : Un algorithme capable de jouer au puissance 4

Dans cet exercice de programmation, on développe un algorithme de min-max capable de jouer une partie de puissance 4. Plus précisément, l'algorithme doit simuler un joueur en décidant les coups à jouer pour répondre aux coups du joueur adverse, qui eux sont ceux renseignés manuellement à travers d'une interface graphique. Autrement dit on permet la réalisation de partie humain contre machine.

Règles du jeu. ♣ *Le but du jeu est d'aligner une suite de 4 pions de même couleur sur une grille comptant 6 rangées et 7 colonnes. Chaque joueur dispose de 21 pions d'une couleur (par convention, en général jaune ou rouge). Tour à tour, les deux joueurs placent un pion dans la colonne de leur choix, le pion coulisse alors jusqu'à la position la plus basse possible dans ladite colonne à la suite de quoi c'est à l'adversaire de jouer. Le vainqueur est le joueur qui réalise le premier un alignement (horizontal, vertical ou diagonal) consécutif d'au moins quatre pions de sa couleur. Si, alors que toutes les cases de la grille de jeu sont remplies, aucun des deux joueurs n'a réalisé un tel alignement, la partie est déclarée nulle.*

Représentation du jeu en machine. Afin de représenter le jeu du puissance 4 en machine, on se munit des types suivants.

- Le type `couleur`, admettant les deux valeurs `Rouge` et `Jaune`, permet de représenter la couleur des pions, mais aussi les joueurs car chacun a une couleur

  ```
  type couleur = Rouge | Jaune
  ```

- Le type `game` permet de reprséneter un état du plateau de jeu par une matrice d'éléments de type `couleur option`. En effet une case vide (sans jeton) est représentée par la valeur `None`, une case avec un jeton rouge (resp. jaune) par la valeur `Some(Rouge)` (resp. `Some(Jaune)`).

  ```
  type game = couleur option array array
  ```

  On suppose fixées dans toute la suite deux grandeurs `nb_colonnes` et `nb_lignes` valant respectivement 7 et 6. Ainsi dans la suite on supposera que les éléments de type `game` sont de dimensions `nb_ligne`$\times$`nb_colonne`.

  ```
  let nb_colonnes = 7
  let nb_lignes   = 6
  ```

♣. Merci Wikipedia : `https://fr.wikipedia.org/wiki/Puissance_4`

- Le type `etat` permet de réprésenter un état du jeu par l'état du plateau de jeu et le joueur dont c'est le tour de jouer.

```
type etat = { game : game ; joueur : couleur }
```

`compagnon.ml`. Dans le fichier `compagnon.ml` on fournit :
- une fonction `draw_game : game -> unit` permettant l'affichage du plateau de jeu ;
- quelques fonctions "boites à outils" vous permettant une manipulation simplifiée du type `game` ;
- un squelette de fonction `main` ouvrant l'interface graphique.

1. Définition d'une fonction d'heuristique

On commence par se munir d'une fonction heuristique d'évaluation de la qualité d'un état du jeu pour un joueur j. L'heuristique de score d'un plateau pour un joueur j est définie de la manière suivante.
- Si le joueur j est gagnant, le score est $+\infty$;
- Si l'autre joueur j est gagnant, le score est $-\infty$;
- Sinon, on somme les "valeurs" des cases occupées sur le plateau, en comptant positivement celles de j et négativement les autres. La valeur de chaque case représente le nombre d'alignements de 4 cases qui la couvre. Ainsi les cases près du centre ont une plus grande valeur. L'idée est que la situation est d'autant plus favorable pour un joueur qu'il a de jetons en position de former des alignement, mais c'est très approximatif car on compte les possibilités comme si le pions était seul sur le plateau, alors que certains des alignements potentiels comptabilisés sont peut-être déjà impossibles. On fournit, dans `compagnon.ml`, une matrice `val_cases` donnant les valeurs des cases.

Q. 1 Définir une fonction `heuristique : etat -> couleur -> int` prenant en argument un état du jeu, un joueur et retournant le score heuristique de l'état du jeu pour ce joueur.

2. Graphe d'états

On se munit maintenant d'une représentation implicite du graphe du jeu du puissance 4.

Q. 2 Définir une fonction `place_jeton : etat -> int -> etat option` prenant en arguments un état du jeu et un indice j de colonne et retournant l'état de jeu après que le joueur (dont c'était le tour) a placé son jeton dans la colonne j. Dans l'éventualité où un tel coup n'est pas possible, si la colonne est déjà pleine, on retourne `None`.

Q. 3 En déduire une fonction `next : etat -> etat list` prenant en argument un état du jeu et retournant la liste de tous les états accessibles en un coup.

Cette fonction `next` nous fournit donc une représentation en machine du graphe d'états du jeu. Le graphe n'est cependant jamais explicitement construit dans la mémoire.

3. Minmax

Q. 4 Définir une fonction `minmax : int -> etat -> couleur -> int` prenant en arguments : une profondeur p de recherche maximale, un état du jeu e, un joueur j et retournant le score de l'état du jeu e pour le joueur j à horizon p coups.

Q. 5 En déduire une fonction `joue : etat -> int -> etat` prenant en argument un état du jeu, une profondeur de recherche maximale et retournant l'état successeur de l'état courant ayant le plus grand score obtenu par l'algorithme de minmax.

4. Partie contre la machine

La fonction `main` fournie dans `compagnon.ml` lance une partie de puissance 4 contre l'algorithme codé dans la fonction `joue`.

Q. 6 Tester cet algorithme en jouant contre lui au puissance 4.

5. Élagage

Q. 7 Remplacer l'algorithme de minmax de la question 4 par un algorithme de minmax avec élagage $\alpha - \beta$.

6. Ouverture

Q. 8 Modifier la fonction `main` pour que deux algorithmes de jeu puissent s'affronter.

Q. 9 Proposer d'autres fonctions d'heuristiques dans le but d'en trouver une qui bat l'algorithme proposé dans les sections précédentes. ♣

♣. Cette question est une queston ouverte, aucune réponse particulière n'est attendue