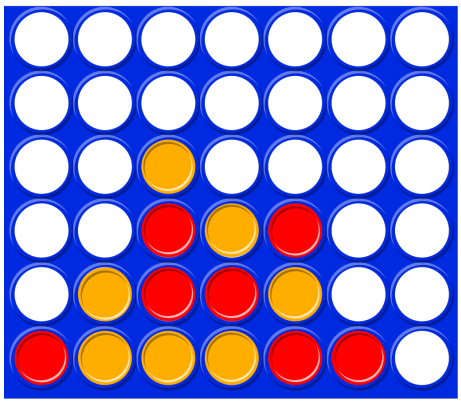
TP Puissance 4



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 la dite 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.
Une configuration de jeu sera représentée par une liste de liste `G` avec `G[x][y]` qui vaut 0, 1 ou 2 selon que la case de la colonne $x \in [\![0;6]\!]$ et de la ligne $y \in [\![0;5]\!]$ est libre, occupée par une pion du joueur 1 ou un pion du joueur 2.

Script 1

> Écrire une fonction `init()` qui renvoie une grille à la position initiale : toutes les cases sont vides.
> Tester votre fonction avec la fonction `affiche(G)` écrite dans le fichier.

Script 2

> Écrire une fonction `coups(G)` qui renvoie la liste des coups possibles dans la grille `G`. Chaque coup étant donné sous la forme d'un couple `(x, y)`.

La variable globale `tous` est définie dans le fichier qui donne la liste de tous les alignements possibles de 4 pions. Chaque alignement étant une liste de couples : `[(0, 0), (1, 0), (2, 0), (3, 0)]` étant l'alignement horizontal en bas à gauche de plateau.

Script 3

> Écrire une fonction `fini(G)` qui renvoie la valeur 1 si le joueur 1 a gagné (un alignement est composé de ses pions), 2 si le joueur 2 a gagné et 0 sinon.

Script 4

> Écrire une une fonction `joue_hasard` qui simule une partie aléatoire : à chaque étape on choisit un coup parmi les coups possibles à l'aide de la fonction `choice` importée du module `random`. La fonction renvoie la grille de fin de partie.
> Une partie comptant au plus 42 coups, on pourra utiliser une boucle `for` que l'on interrompra si la partie se finie avant que la grille soit remplie.
> Estimer la probabilité que le joueur 1 gagne la partie si les joueurs jouent au hasard.

Pour construire une heuristique, on teste tous les alignement possibles : si un alignement est occupé en partie par des pions du joueur 1 et des pions du joueur 2, alors il n'a plus d'influence sur une possible victoire. Sinon, on se donne un poids w_i pour chaque $i \in [\![0;4]\!]$ qui est ajouté à l'heuristique si l'alignement contient i pions du joueur 1 (et aucun du joueur 2) et qui est retiré à l'heuristique si l'alignement contient i pions du joueur 2 (et aucun du joueur 1).
En notant $n_k(A)$ le nombre de pions du joueur k pour l'alignement A, on a ainsi pour une grille G :

$$h(G) = \sum_{A \text{ tel que } n_2(A)=0} w_{n_1(A)} - \sum_{A \text{ tel que } n_1(A)=0} w_{n_2(A)}.$$

Script 5

Écrire une fonction `heuristique(G, w)` qui prend en argument une grille et une liste de poids et qui renvoie la valeur de l'heuristique décrite ci-dessus.

Script 6

Écrire une fonction `minmax(G, joueur, w, p)` qui prend en argument une grille `G`, le numéro du joueur `joueur`, la liste des poids `w` et la profondeur `p` et qui renvoie le score et un coup associé.

Script 7

Écrire une fonction `humain_vs_machine(w, p)` qui gère une partie entre un joueur humain (qui joue en premier) et la stratégie min-max pour l'heuristique définie par la liste de poids `w` et la profondeur `p`.
À chaque tour de l'utilisateur, on affichera la grille puis on demandera le numéro de colonne choisie par l'instruction :
`x = int(input("numéro de colonne : "))` et à chaque tour de la machine, on affichera la grille puis le numéro de colonne choisi.

Script 8

Écrire une fonction `partie_2machines(w1, w2, p1, p2)` qui prend en argument une liste de poids pour chaque joueur et une profondeur pour chaque joueur et qui retourne la grille obtenue en fin de partie entre deux joueurs suivant l'algorithme de min-max chacun avec son heuristique définie par sa liste de poids et sa profondeur.

Script 9

Comparer, pour une profondeur 2, l'efficacité des listes de poids proposées dans le fichier. Afficher dans un tableau les résultats.