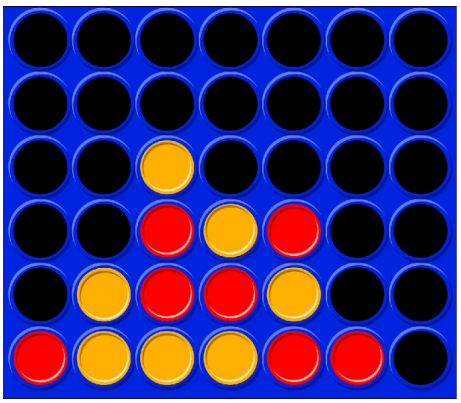
TP noté

PTSI Lycée Eiffel

18 décembre 2020

Le but de ce TP noté est de programmer en Python une version basique du jeu Puissance 4™. Aucune manipulation d'interface graphique ne sera nécessaire, on se contentera d'afficher la grille en mode texte après chaque coup. Rappelons le principe du jeu : chaque joueur dépose tour à tour un de ses pions dans l'une des colonnes d'une grille verticale de six lignes fois sept colonnes, jusqu'à ce que l'un des joueurs ait réussi à aligner quatre pions (horizontalement, verticalement ou en diagonale). Chaque pion joué tombe nécessairement à la position la plus basse disponible dans la colonne choisie. Un exemple de grille de jeu après sept coups joués par chacun des joueurs (les pions jaunes correspondent aux coups du premier joueur, les pions rouges à ceux du deuxième joueur) :



Sauf mention du contraire, aucune méthode ou fonction permettant de manipuler les listes n'est autorisée, à l'exception de la fonction **len** et de la méthode **append**. En particulier, il est hors de question d'utiliser des commandes du type **max** ou **count** qui pourraient rendre totalement immédiate la résolution de certaines questions.

Les différentes parties du sujet étant en partie indépendantes, il n'est absolument pas nécessaire d'avoir fait toute la partie I pour tenter de traiter les premières questions des parties II et III (qui sont elles-mêmes assez indépendantes, sauf la question 12 qui reprend tout ce qui a été fait auparavant). La dernière partie n'est à aborder que si vous avez bien réussi tout le reste.

I. Pour s'échauffer : quelques programmes manipulant les listes.

Certaines des fonctions programmées dans cette première partie pourront bien sûr être utilisées dans des programmes ultérieurs, mais le but de cette partie introductive est surtout de tester votre capacité à écrire des fonctions « simples » sur les listes, ne soyez donc pas surpris si certaines fonctions n'ont pas grand rapport avec la suite de l'énoncé.

Question 1 : Écrire une fonction Python **appartient** qui prend comme arguments une liste de nombres L et un nombre x, et qui renvoie True si le nombre x apparait au moins une fois comme élément de L, False sinon.

Exemple : La commande appartient([3,2,1,6],5) doit renvoyer False.

Question 2 : Écrire une fonction Python **compte** qui prend comme arguments une liste de nombres L et un nombre x, et qui compte le nombre d'éléments dans la liste L qui sont égaux à x (il existe une méthode **count** qui fait exactement celà mais qu'il est bien entendu interdit d'exploiter pour cette question).

Exemple : La commande compte([3,3,2,1,3,4,5,3],3) doit renvoyer la valeur 4.

Question 3 : Écrire une fonction Python **remplace** qui prend comme arguments une liste de nombres L et deux nombres x et y, et qui remplace chaque élément de la liste initialement égal à x par un y.

Exemple : La commande remplace([1,2,3,1,4,3,2],2,5) doit renvoyer [1,5,3,1,4,3,5].

Question 4 : Écrire une fonction Python **successifs** qui prend comme arguments une liste de nombres L et un nombre x, et qui renvoie le nombre de fois où x apparait successivement au début de la liste L. La fonction doit donc renvoyer la valeur 0 si le premier élément de la liste n'est pas égal à x, et on fera attention à gérer correctement le cas où tous les éléments de la liste sont égaux à x.

Exemple : La commande successifs([2,2,2,1,3,2,2,5],2) doit renvoyer la valeur 3.

Question 5 : Écrire une fonction Python **maxsuccessifs** qui prend comme arguments une liste de nombres L et un nombre x, et qui renvoie le nombre maximal d'apparitions successives de la valeur x dans la liste L.

Exemple : La commande maxsuccessifs([2,2,1,2,2,2,1,2],2) doit renvoyer la valeur 3.

Question 6 : Écrire une fonction Python **memelongueur** qui prend comme arguments une liste de listes L, et qui renvoie True si tous les éléments de L sont des listes contenant le même nombre d'éléments, False sinon.

Exemple : La commande memelongueur([[1,2],[3,1],[2,3]]) doit renvoyer True.

II. Gestion de la création et de l'affichage de la grille.

Dans cette partie, nous allons écrire les fonctions Python permettant un affichage simple mais lisible de la grille de jeu. L'objectif est de créer un affichage en mode texte, où le 0 sera utilisé pour indiquer les cases vides dans la grille, et les chiffres 1 et 2 les cases contenant les jetons des deux joueurs. Ainsi, la grille correspondant à l'image donnée au début de l'énoncé pourra être représentée ainsi :

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	0	2	1	2	0	0
0	1	2	2	1	0	0
2	1	1	1	2	2	0

Question 7 : Écrire une fonction Python **initialisegrille** qui prend comme arguments deux entiers n et p, et qui renvoie une liste de n listes contenant chacune p entiers initialisés à la valeur 0.

Exemple : La commande initialisegrille(4,3) doit renvoyer la liste $[[0,0,0],[0,0,0],[0,0,0],[0,0,0]]$.

Question 8 : Écrire une fonction Python **affichegrille(g)** qui prend comme argument une liste de listes g et affiche la grille de jeu correspondante sous le format donné ci-dessus (le but principal est que les différentes lignes de la grille soient affichées les unes en-dessous des autres, et on essaiera de séparer les différents éléments de chaque ligne par des barres verticales. Les lignes horizontales encadrant la grille sont accessoires.

III. Programmation des fonctions indispensables au fonctionnement du jeu.

Il est maintenant temps de programmer les principales fonctionnalités du jeu : gestion des coups des deux joueurs, vérification de la position pour voir si l'un des deux joueurs a gagné, et écriture de la fonction principale permettant de jouer. On programmera des fonctions laissant aux joueurs la possibilité de jouer sur une grille de taille différente de celle du jeu classique.

Question 9 : Écrire une fonction Python **couppossible(g,i)** qui prend comme arguments une grille de jeu g (liste de listes de nombres entiers) et un entier i représentant le numéro de colonne où le joueur souhaite placer un pion, et qui renvoie True si le coup est possible, et False sinon (donc si le numéro de colonne est incohérent avec la taille de la grille, ou si la colonne i est déjà remplie). On supposera que le joueur numérote les colonnes à partir de 1 et non à partir de 0 comme le fait Python.

Exemple : La commande couppossible([[0,1,0],[1,2,0],[2,1,2]],3) doit renvoyer True.

Question 10 : Écrire des fonctions Python **coupj1** et **coupj2** qui prennent comme arguments une grille de jeu g et un entier i et qui modifient (quand le coup est possible) la grille en plaçant un jeton du joueur 1 ou du joueur 2 dans la colonne i. Le return de la fonction doit renvoyer la grille de jeu, modifiée ou non.

Exemple : La commande coupj2([[0,0,0],[0,2,1][1,1,2]],1) doit renvoyer la liste $[[0,0,0],[2,2,1],[1,1,2]]$.

Question 11 : Écrire une fonction Python **verifiegain** qui prend comme argument une grille de jeu g et qui vérifie si l'un des deux joueurs a quatre pions alignés (cette fonction étant pénible à programmer, on pourra se contenter d'une version où on ne vérifie que les alignements horizontaux ou verticaux).

Question 12 : Écrire une fonction Python **puissance4** qui prend comme arguments deux entiers n et p, et qui permet à deux joueurs d'effectuer une partie de Puissance 4™ sur une grille à n lignes et p colonnes. On essaiera bien sûr de réutiliser les questions précédentes, et de proposer une interface de jeu compréhensible (un message du type « Joueur 2, à vous de choisir une colonne » s'affichera pour indiquer à chaque joueur que c'est à son tour de jouer). Toute version, même inaboutie, sera récompensée.

IV. Complément : programmation d'une IA basique.

Cette dernière partie a pour but de programmer une version du jeu contre l'ordinateur pour ceux qui ont trouvé trop triviales les questions précédentes.

Question 13 : Écrire une fonction Python **puissordibete** qui prend comme arguments deux entiers n et p et qui permette de joueur une partie contre un ordinateur idiot qui choisit pour chacun de ses coups une colonne de la grille de façon totalement aléatoire. On rappelle que la fonction **randint(a,b)** issue du module **random** permet d'obtenir un entier aléatoire compris entre les valeurs a et b, incluses toutes les deux.

Question 14 : Écrire une fonction Python **puissordi** qui prend comme arguments deux entiers n et p et qui permette de jouer une partie contre un ordinateur « intelligent » qui utilise la tactique suivante : à chacun de ses coups, il choisit de jouer dans la colonne lui permettant de créer le plus long alignement de ses pions possibles (ainsi, s'il a au moment de jouer au maximum deux pions alignés dans la grille, il choisira une colonne lui permettant d'aligner un troisième de ses pions si possible ; si plusieurs colonnes conviennent, il choisit la première disponible, ou il en choisit une au hasard, au choix).

Quelques remarques pour finir pour ceux qui seraient assez motivés pour creuser encore plus ce sujet :

- la pseudo-intelligence artificielle de la question 14 joue en fait encore plus mal que l'ordinateur jouant au hasard puisqu'elle ne tient aucun compte des coups joués par l'autre joueur. Si on joue en premier, il suffit donc de créer son alignement tranquillement dans son coin sans se préocupper de ses coups, et on gagnera systématiquement (et si on joue en deuxième c'est à peine plus dur).
- pour l'améliorer, une possibilité est de programmer l'ordinateur pour qu'il anticipe un coup à l'avance : quand il doit jouer un coup, pour chacune des colonnes possibles, il imagine le coup que produira l'autre joueur (par exemple en supposant que l'autre joueur va systématiquement tenter de créer un alignement provisoire le plus long possible), puis il choisit la colonne qui, sous cette hypothèse, lui permettra **au tour d'après** de jouer le coup qui lui donnera la situation jugée optimale. On peut bien sûr ensuite augmenter la profondeur de calcul (prévoir deux ou trois coups à l'avance), mais le temps de calcul explose assez rapidement. Mais un tour d'avance suffira déjà à éviter que l'ordinateur ne laisse au joueur humain la possibilité de compléter un alignement de quatre pions s'il a à disposition un coup pouvant le bloquer.
- le critère « le plus de pions alignés possible » est en fait très mauvais pour juger de la qualité d'un coup. Par exemple, ajouter un troisième pion à un alignement de deux pions quand on sait qu'on ne pourra jamais en ajouter un quatrième ensuite n'a aucun intérêt (par exemple, si on a des pions à hauteur 4 et 5 dans une colonne, ajouter un troisième pion à hauteur 6 ne permettra jamais de conclure un alignement vertical dans cette colonne). Le problème évident, c'est que trouver un critère vraiment pertinent de jugement de la qualité d'un coup (et facilement calculable par une machine) est très difficile. Si vous avez du temps à perdre...