Jalon 01

marc-michel dot corsini at u-bordeaux dot fr

12 janvier 2021

Pour ce premier jalon, nous allons mettre en place la classe de base pour jouer au jeu du « Puissance 4 » et à quelques variations.
Vous allez, dans un premier temps récupérer l'archive `projet.zip`. Une fois décompressée vous obtiendrez une arborescence dont la racine est `Projet_IA`, dans laquelle vous trouverez (à la racine) le fichier `Readme.md`. Ce fichier est à **lire**.

Vous partirez du fichier `connect4.py` que vous *dupliquerez* en changeant de nom. Ce nom **devra** être composé d'un préfixe n'utilisant que des lettres non accentuées, le sous-ligné (tiret du 8) et des chiffres ; le suffixe sera quant à lui `.py`
Chaque semaine de TD vous m'enverrez un instantanné de vos codes, et **uniquement** vos codes.
Date de l'évaluation Elle sera spécifiée sur le site.

Conventions Tout au long du projet, les classes seront définies par un nom commençant par une majuscule. Les attributs et méthodes auront des identifiants anglophones, sans majuscule en première lettre, s'ils sont constitués de plusieurs mots, on utilisera le séparateur « souligné (aka tiret du 8) ».
Il n'y a aucun traitement d'erreur à moins qu'ils n'aient été explicitement demandés dans les fiches, il n'y a pas de directives *assert* dans votre code.

1 Rappels du projet

On souhaite réaliser un joueur efficace pour le jeu du « Puissance 4 » et certaines de ses variations. Les variations étudiées sont

1. Un terrain de dimensions variables, pour un nombre quelconque de pierres à aligner
2. Un terrain de forme cylindrique, i.e. dans lequel la dernière colonne est considérée comme adjacente à la première colonne.

Les mécanismes de base du jeu sont en place, reste à établir quand on obtient une configuration gagnante. On se focalisera ici **uniquement** sur les configurations gagnantes passant par la dernière pierre posée.
Une fois que l'on sera capable de déterminer quand une partie est gagnée on s'interressera à la mise en place de joueurs plus ou moins aptes à gagner au jeu. Chaque technique sera l'objet d'un jalon.

2 Travail pour le Jalon 01

Il va falloir reprendre la méthode `win` qui détecte si la dernière action a permis de réaliser un alignement de `p` pierres de même couleur.

3 Classe de base

Le fichier `connect4.py` doit nous permettre de jouer. Il y a une seule classe dans ce fichier : `Board`. Nous allons passer en revue les différents constituants de la classe.

3.1 Constructeur

Le constructeur accepte 4 paramètres, dont les valeurs par défaut correspondent au jeu classique. Les paramètres sont, dans l'ordre

1. `nl` le nombre de lignes du plateau, il ne peut pas être inférieur à 3
2. `nc` le nombre de colonnes du plateau, il ne peut pas être inférieur à 3
3. `p` le nombre de pierres à aligner, horizontalement, verticalement ou en diagonale pour obtenir une configuration gagnante. Ce nombre ne peut pas être inférieur à 2 et ne peut pas être supérieur à la plus petite dimension du tablier.
4. `cylinder` un booléen (valant **True** ou **False**) indiquant si le tablier est considéré comme un cylindre ou non.

Ainsi

```
>>> b = Board()
```

crée un tablier avec 6 lignes, 7 colonnes non cylindrique pour lequel il faut aligner 4 pierres afin d'obtenir une configuration gagnante.

```
>>> b = Board(p=5, cylinder=True)
```

crée un tablier avec 6 lignes, 7 colonnes cylindrique pour lequel il faut aligner 5 pierres si on souhaite obtenir une configuration gagnante.

Quelle instruction ?
Quelle est l'instruction pour créer un tablier 3x4x3 non cylindrique ?
Quel est le tablier obtenu si on tape `b=Board(2,4,5,True)` ?

3.2 Attributs

Différents attributs sont associés à cette classe.

3.2.1 En lecture seule

Les attributs en lecture seule (appelés aussi « getter » sont signalés par la directive `property`.

1. `nbl` renvoie le nombre lignes
2. `nbc` renvoie le nombre colonnes
3. `stones` renvoie le nombre de pierres à aligner
4. `cylinder` renvoie un booléen indiquant la nature du tablier
5. `timer` renvoie le nombre de pierres présentes sur le terrain i.e. le nombre de coups joués
6. `turn` renvoie 'J' si c'est le premier joueur qui a le trait, 'R' sinon
7. `opponent` renvoie l'adversaire du joueur ayant le trait
8. `actions` renvoie un n-uplet des colonnes dans lesquelles on pourra jouer le prochain coup. Une colonne étant représentée par une lettre majuscule de l'alphabet. Le n-uplet est ordonné en fonction de l'ordre alphabétique.
9. `board` est un n-uplet d'entiers. La taille de ce n-uplet est `nbl x nbc`. Un 0 indique une case vide, 1 indique une case occupée par une pierre du joueur 'J', 2 une case occupée par le joueur 'R'.

3.2.2 En lecture écriture

Les attributs en lecture écriture (« getter et setter ») sont signalés par deux directives, une pour le getter, une pour le setter. Il n'y a qu'un attribut dans cette catégorie : `state`. C'est un n-uplet constitué des coups qui ont été joués. Les positions impaires sont les coups joués par 'J' ; les positions paires les coups joués par 'R'.
Le « getter » se contentent de renvoyer le n-uplet. Le « setter » s'assure que l'information fournie est conforme aux conditions de jeu. Si l'information fournie est impossible parce qu'elle ne respecte pas les axiomes ci-après, l'affectation est refusée. Dans ce cas on revient à la configuration antérieure, i.e. celle avant la tentative de modification.
Une configuration `cfg` est valide si elle est de la forme

$$(x_1, y_1), (x_2, y_2) \dots (x_k, y_k)$$

1. $\forall 1 \leq j \leq k, 0 \leq x_j < nbl$
2. $\forall 1 \leq j \leq k, 0 \leq y_j < nbc$
3. $\forall 1 \leq j \leq k$, si $x_j = 0$ alors $\forall i < j, \nexists y_i = y_j$
4. $\forall 1 \leq j \leq k$, si $x_j > 0$ alors $\exists! i, i < j, x_i = x_j - 1 \wedge y_i = y_j$

Le premier axiome impose que le numéro de ligne est borné, le second axiome impose que le numéro de colonne est borné, le troisième axiome impose que la ligne 0 est la première occurence de la colonne y_j, enfin le 4ème axiome impose qu'on a vu exactement une fois chaque ligne plus petite que x_j pour la colonne y_j.

3.3 Méthodes

Dans ce qui suit, une méthode dite « sans paramètre » signifie sans paramètre **autre** que le paramètre `self`. Une méthode avec paramètre est une méthode qui reçoit des paramètres **en sus** du paramètre `self`.

1. `__repr__` permet d'afficher le constructeur valide qui a été utilisé. Méthode sans argument qui renvoie une chaîne de caractères
2. `__str__` permet d'afficher le tablier sous sa forme bi-dimensionnelle grâce à la commande `print`. Méthode sans argument qui renvoie une chaîne de caractères.
3. `move` prend en entrée une action (le nom d'une colonne) et ne renvoie rien. Si l'action est autorisée, elles met à jour les variables `timer, turn, state, board`
4. `undo` sans argument, défait le dernier mouvement en mettant à jour les variables `timer, turn, state, board`.
5. `reset` sans argument, permet de recommencer une partie
6. `over` sans argument renvoie un booléen. Permet de savoir si une partie est terminée. Soit parce que tous les coups sont joués, soit parce que le dernier coup a créé une configuration gagnante.
7. `win` sans argument renvoie un booléen. Renvoie **True** si et seulement si la configuration courante est une configuration gagnante.
8. `show_msg` sans argument, renvoie une chaîne de caractères. Cette méthode est utilisée par `__str__` pour savoir la nature du terrain, le nombre de pierres jouées et si on a une victoire (pour qui), un match nul ou si la partie doit continuer.

4 Aide pour le jalon

Il va donc falloir travailler sur la méthode `win` qui, pour le moment renvoie toujours **False**. Le travail a effectué ne consiste pas à déterminer si pour une situation de jeu quelconque, il existe une configuration gagnante. On souhaite déterminer si le **dernier coup** a conduit à une situation gagnante. C'est-à-dire que en examinant l'attribut `board`, on sait que la détection va se faire à partir du **dernier** élément.
Le dernier élément d'un n-uplet de longueur quelconque est en position -1. La commande `board[-1]` permet donc

d'accéder à cette information.
On sait que l'attribut `board` est organisé de telle sorte que les index pairs correspondent au joueur 'J', les index impairs correspondent au joueur 'R'.
Il existe une commande permettant d'extraire d'un n-uplet (ou d'une liste) tous les éléments ayant un index pair `board[::2]` – cette commande veut dire « faire une copie de la variable en prenant depuis le début, jusqu'à la fin, un index sur 2.
Pour obtenir tous les index impair, il suffit de dire qu'on commence à l'index 1 et qu'on va jusqu'à la fin, en ne prenant qu'une information sur 2 `board[1::2]`
Une fois la sous-liste extraite, il ne reste plus qu'à vérifier s'il est possible de trouver un alignement de longueur `stones` passant par la dernière valeur.

```
>>> b = Board()
>>> b.state = state = [ (0,1), (0,2), (1,1), (2,1), (3,1) ]
>>> b.state
((0, 1), (0, 2), (1, 1), (2, 1), (3, 1))
>>> b.stones
4
>>> b.state[::2]
((0, 1), (1, 1), (3, 1))
>>> b.state[1::2]
((0, 2), (2, 1))
>>> b.state[-1]
(3, 1)
>>> b.turn
'R'
>>> b.opponent
'J'
>>> b.timer
5
>>>
```

Dans cette situation, on s'attend à ce que `b.win()` renvoie **False**, puisqu'il est impossible qu'un joueur ai pu aligner 4 pierres de sa couleur.