Jalon 01 : Rev. 1

marc-michel dot corsini at u-bordeaux dot fr

18 janvier 2021

Révision On m'a fait remarquer à juste titre que je m'étais trompé d'attribut à scruter dans la section « Aide pour le jalon » section 4. J'ai aussi rajouté une explication sur les tests du fichier `test_win.py` du répertoire `tests`.

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 `state`, 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 `state[-1]` permet donc

d'accéder à cette information.
On sait que l'attribut `state` 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 `state[::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 `state[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.

4.1 test_win

Ce fichier est réservé aux tests pour la méthode `win`. Regardons, dans un premier temps ce qui se passe lorsque l'on lance `main_tests` et qu'on se focalise sur le résultat de `test_win`. J'utiliserai ici le fichier fourni (qui n'est pas votre fichier de travail) `connect4` – **attention** le fichier fourni est écrasé/remplacé à chaque nouvelle version du fichier `projet.zip`. Les résultats, sont ici ceux obtenus avec la version **v1** du fichier zip.

```
mmc@hobbes-lr:Code$ python3 main_tests.py connect4
tentative de lecture de connect4
select wich subtests you want
Pour répondre par oui, utiliser l'un des symboles 'oO0Yy'
Passer tous les tests ? 1 # non
Tests du jalon 01 ? 0 # oui
>>> Jalon 01: Test de board ? 1 # non
>>> Jalon 01: Test de win ? 0 # oui
win added
Vous avez 1 série(s) à passer
test_win_set_over (tests.test_win.TestWinIsOver_333)
board (1, 2, 0, 2, 1, 0, 1, 2, 0) ... FAIL
test_win_2 (tests.test_win.TestWinIsOver_332)
board (0, 1, 0, 0, 2, 0, 0, 0, 0) ... test_win_3 (tests.test_win.TestWinIsOver_332)
board (0, 1, 0, 0, 2, 0, 0, 1, 0) ...
======================================================================
FAIL: test_win_set_over (tests.test_win.TestWinIsOver_333)
board (1, 2, 0, 2, 1, 0, 1, 2, 0)
```

```
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/mmc/Cours/IntelligenceArtificielle/P4_2021/Projet_IA/Code/tests/test_win.py", line 30, in test_wi
    self.assertTrue(self.o.win(), "Expected a win")
AssertionError: False is not true : Expected a win

======================================================================
FAIL: test_win_2 (tests.test_win.TestWinIsOver_332) (choice='A')
board (0, 1, 0, 0, 2, 0, 0, 0, 0)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/mmc/Cours/IntelligenceArtificielle/P4_2021/Projet_IA/Code/tests/test_win.py", line 51, in test_wi
    self.assertTrue(self.o.win(), "Expected a win")
AssertionError: False is not true : Expected a win

======================================================================
FAIL: test_win_2 (tests.test_win.TestWinIsOver_332) (choice='C')
board (0, 1, 0, 0, 2, 0, 0, 0, 0)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/mmc/Cours/IntelligenceArtificielle/P4_2021/Projet_IA/Code/tests/test_win.py", line 51, in test_wi
    self.assertTrue(self.o.win(), "Expected a win")
AssertionError: False is not true : Expected a win

======================================================================
FAIL: test_win_3 (tests.test_win.TestWinIsOver_332) (choice='A')
board (0, 1, 0, 0, 2, 0, 0, 1, 0)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/mmc/Cours/IntelligenceArtificielle/P4_2021/Projet_IA/Code/tests/test_win.py", line 73, in test_wi
    self.assertTrue(self.o.win(), "Expected a win")
AssertionError: False is not true : Expected a win

======================================================================
FAIL: test_win_3 (tests.test_win.TestWinIsOver_332) (choice='C')
board (0, 1, 0, 0, 2, 0, 0, 1, 0)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/mmc/Cours/IntelligenceArtificielle/P4_2021/Projet_IA/Code/tests/test_win.py", line 73, in test_wi
    self.assertTrue(self.o.win(), "Expected a win")
AssertionError: False is not true : Expected a win

----------------------------------------------------------------------
Ran 3 tests in 0.001s

FAILED (failures=5)
mmc@hobbes-lr:Code$
```

Tout à la fin vous voyez apparaître le message **Ran 3 tests in ...** suivi d'un message précisant

1. si les tests sont réussis. Dans ce cas il y aura **OK**
2. sinon, la nature et le nombre d'erreurs identifiées

Dans cet exemple il y a eu 5 erreurs. On va maintenant repartir au tout début des messages afin d'identifier les problèmes en vu de les corriger. Les erreurs sont signalées par le mot **FAIL** et sont séparées par une ligne de '='

```
======================================================================
FAIL: test_win_set_over (tests.test_win.TestWinIsOver_333)
board (1, 2, 0, 2, 1, 0, 1, 2, 0)
----------------------------------------------------------------------
Traceback (most recent call last):
```

```
  File "/home/mmc/Cours/IntelligenceArtificielle/P4_2021/Projet_IA/Code/tests/test_win.py", line 30, in test_wi
    self.assertTrue(self.o.win(), "Expected a win")
AssertionError: False is not true : Expected a win
```

Le test est `test_win_set_over` et c'est une méthode de la classe `TestWinIsOver_333`. Juste après est affichée la « docstring » de la méthode, en l'occurence il s'agit ici d'un tablier de 9 cases :

$$(1, 2, 0, 2, 1, 0, 1, 2, 0)$$

Le message d'erreur est le suivant « à la ligne 30 dans le test, la commande `assertTrue(self.o.win(), ...)` a échoué, on attendait la réponse `True`, alors qu'on a obtenu la réponse `False`, provoquant le message d'erreur 'Expected a win' ».
On se reporte alors dans le fichier `test_win` pour comprendre la nature du test. Voici le code de la classe

```
    def setUp(self):
        self.K = getattr(tp, 'Board')
        self.o = self.K(3, 3, 3)

    def test_win_set_over(self):
        """
           board (1, 2, 0, 2, 1, 0, 1, 2, 0)
        """
        for i in range(3): self.o.move('A') # JRJ
        for i in range(3): self.o.move('B') # RJR
        self.assertEqual(self.o.board,
                         (1, 2, 0, 2, 1, 0, 1, 2, 0), "board is wrong")
        self.o.move('C')
        self.assertTrue(self.o.win(), "Expected a win")
        self.assertTrue(self.o.over(), "Expected game over")
        self.assertEqual(self.o.actions, (), "No action left")
```

On découvre 2 méthodes, la première `setUp` sert à initialiser le test, grosso modo cela signifie que l'on a fait l'équivalent de

```
self.o = Board(3, 3, 3)
```

On a donc créé un tablier de 3 lignes et 3 colonnes, et l'objectif pour gagner est d'aligner 3 pierres de sa couleur.
La seconde méthode `test_win_set_over` est le test à proprement parler son objectif est de vérifier que si on détecte un gagnant la partie s'arrêtera.
La première instruction, répète 3 fois `move('A')` cela veut dire que 'J' joue dans la première colonne, que 'R' joue dans la première colonne et que 'J' rejoue dans la première colonne, créant ainsi un empilement de 3 pierres dans la colonne A.
La seconde instruction fait la même chose pour la colonne 'B' les 4èmes, 5ème et 6ème coups sont joués dans cette seconde colonne.
Un test est effectué pour vérifier que le tableau est correctement rempli, puis on joue le coup 'C' étant donné que c'est le 7ème coup, il est joué par 'J'.
Récapitulons 'J' a joué en 'A' au coup 1 et 3, en 'B' au coup 5 et en 'C' au coup 7. Il a donc obtenu les positions (0,0) ; (2,0) ; (1,1) et (0,2) créant ainsi un alignement de 3 pierres en diagonale descendante. C'est un coup « gagnant », on devrait donc obtenir si on interroge `win()` la réponse **True**.
Les deux instructions suivant l'erreur n'ont pas été faites (les tests s'arrêtent à la **première** erreur constatée.
L'instruction suivante vérifie que `over()` renvoie bien **True** lorsque `win()` renvoie **True**.
La dernière commande du test contrôle, quant à elle le fait qu'il n'y a plus d'action disponible (de coup jouable après une victoire).

Je vous laisse explorer/comprendre ce qui se passe dans les autres tests.

4.2 Conseil pratique

Je vous déconseille de chercher à résoudre tous les problèmes en même temps. Il est bien plus malin de traiter **une** erreur à la fois. Une bonne façon de « comprendre » pourquoi il y a un problème est d'ouvrir un shell sur votre code et de reproduire le test que vous voulez réussir. Voilà la séquence dans votre shell, qui serait à essayer :

```
>>> b = Board(3,3,3) # la commande du setUp
>>> for i range(3): b.move('A')
...
>>> print(b) # je vérifie de visu que c'est comme je veux
>>> for i range(3): b.move('B')
...
>>> print(b) # je vérifie de visu que c'est comme je veux
>>> b.move('C')
>>> print(b) # ah oui j'ai bien un alignement pour 'J'
>>> b.win() == True # c'est le test
>>> False # <--- argh c'est pas bon, je dois faire qque chose
```

Puis lorsque ça marche, vous avez, soit la possibilité de relancer le test et de voir si, par chance c'était juste ça ; ou alors vous regardez « à la main » la suite.

```
>>> b = Board(3,3,3) # la commande du setUp
>>> for i range(3): b.move('A')
...
>>> for i range(3): b.move('B')
...
>>> b.move('C')
>>> b.win() == True # c'est le test
>>> True # <--- Yes !!!
>>> b.over() # c'est la prochaine commande à tester si True on fait la suivante
```

Normalement, ça devrait être ok pour tout ce qui touche aux tests relatifs aux méthodes autre que `win()`.

4.3 Pour conclure

Les tests pour la détection du gagnant vont être étoffés et seront publiés dans les versions suivantes du fichier `projet.zip`, veillez donc à ce que vous ayez toujours la dernière version disponible sur le site.