Fiche Jalon 01

mmc
marc-michel dot corsini at u-bordeaux dot fr

Rev. 3 : 20 Janvier 2022

Vous devez récupérer le fichier `jalon_01.zip` qui contient l'arborescence dans laquelle vous allez travailler. Le dossier principal s'appelle `Projet_IA` dans lequel vous trouverez 2 répertoires `Code` et `Fiches`

1. Le répertoire `Fiches` contient les fiches `PDF` qui sont aussi directement accessibles sur `https://moodle-miashs.uf-mi.u-bordeaux.fr/`
2. Le répertoire `Code` contient 4 fichiers (pour le moment) `hexapawn.py`, `morpion.py`, `allumettes.py` et `divide_left.py` ainsi que 3 sous-répertoires `tests`, `classes` et `tools`. Vous ne **devez pas** altérer cette architecture et vous **n'avez pas à travailler** dans les sous-répertoires. **Toutes** vos opérations se font dans le répertoire `Code`
   - le répertoire `classes` contient 2 fichiers qui sont les classes abstraites permettant de créer des jeux et des joueurs.
   - le répertoire `tests` contiendra les tests à passer pour valider les jalons, pour le moment aucun test n'a été configuré et vous aurez dans `Code` un fichier qui servira à passer les tests.
   - le répertoire `tools` contient de petits utilitaires tels que `ezCLI`

   Les fichiers `allumettes` et `divide_left` sont des implémentations complètes de 2 des jeux présentés en cours. Les fichiers `hexapawn` et `morpion` sont des extraits de l'implémentation des 2 jeux de plateau étudiés cette année.

En fonction de votre niveau en informatique vous allez choisir le jeu sur lequel vous allez travailler pour ce premier jalon. Faible niveau en programmation, ce sera `hexapawn`, sinon ce sera `morpion`
Quelque soit le fichier dans lequel vous allez travailler, vous n'avez qu'une seule chose à faire : écrire la fonction `valid_state`

Cette fonction prend en entrée un `tuple` et renvoie un `bool`. Elle renvoie `True` si l'état est valide, `False` Sinon.

Dans chaque fichier, vous avez un petit code permettant de vérifier des cas simples. Vous pouvez dores et déjà lancer l'exécution des codes et voir les résultats attendus sur les cas simples.

1 Quelque soit le jeu choisi

Il faut vérifier les points suivants :

1. Que le tuple est de taille 2
2. Que la première valeur est une chaîne de caractères de la bonne longueur
3. Que la seconde valeur est un entier
4. Que la chaîne ne contient que des valeurs dans `self.PAWN`

2 Ce qu'il faut faire pour le jeu « HexaPawn »

Il faut vérifier les 4 situations suivantes :

1. Qu'il n'y a pas plus de pions d'une couleur que de colonnes
2. Qu'il n'y a pas plus d'un pion dans le camp adverse
3. Qu'il n'y a pas, à la fois un pion 'X' dans le camp adverse et un pion 'O' dans le camp adverse. En effet, avoir un pion dans le camp adverse signifie que la partie est terminée
4. Que le nombre de déplacements visibles est bien inférieur ou égal au nombre de coups joués

3 Ce qu'il faut faire pour le jeu « Morpion »

Il faut vérifier les 4 premières situations, la cinquième est **bonus**

1. Que le nombre de 'X' et le nombre de 'O' sont compatibles avec la valeur entière
2. Que le nombre de 'X' est égal au nombre de 'O' ou au nombre de 'O' + 1
3. Que si phase vaut 0, la valeur entière n'excède pas le nombre de pierres
4. Que si phase n'est pas 0, la valeur entière n'excède pas la limite du temps de jeu
5. **Uniquement pour les plus aguerris** Qu'il n'y ait pas plus d'un alignement gagnant disjoint ('O' et 'X' ne peuvent pas avoir aligner, un joueur ne peut pas avoir plusieurs alignements sans une unique pierre commune).[1] Commencez par traiter le cas simple de la grille 3×3.

[1] Les alignements sont délicats, surtout lorsque l'on a un terrain qui est un tore (cela n'arrive que pour les tailles de grilles 5 et 7). Pour une grille 3×3, le fait que ce soit un tore n'a aucune incidence car il n'y a que 10 alignements possibles et ils sont de taille 3.

4 Comment procéder

Les fichiers fournis vont évoluer au cours des jours et semaines à venir, les mise à jour vont donc **écraser** les anciennes versions. Il est donc **important** que les fichiers sur lesquels vous travaillez n'ait pas le même nom que ceux que je vous distribue.

Pour ce premier jalon, une fois que vous avez choisi le jeu, recopiez sous un nouveau nom le fichier du jeu. Je vous encourage à utiliser comme nom de fichier `projet.py` et à n'éditer que ce dernier.

Vous cherchez dans le fichier `projet.py` la méthode `valid_state` et vous remplacez la ligne `return False` par votre code.

4.1 Quelques petites astuces

Il est plus efficace d'écrire des tests simples qui échouent qu'un gros test qui rate. Par exemple supposons que vous cherchiez à accepter si une liste est de longueur paire, qu'elle ne contient que des chaînes de caractères, et qu'il n'y ait pas plus de voyelles que de consonnes. Au lieu d'écrire :

```
def valid_liste(self, L:list) -> bool:
    if longueur == 2 and contient que des chaines and ... :
        return True
    else:
        return False
```

Il vaut mieux écrire :

```
def valid_liste(self, L:list) -> bool:
    if longueur != 2: return False
    if ne contient pas que des chaines: return False
    if nb_voyelles != nb_consonnes: return False
    return True
```

4.2 Codage

- Pour accéder aux paramètres du constructeur, il faut utiliser la commande

  ```
  self.get_parameter('key')
  ```

  par exemple pour connaître le nombre de lignes et le nombre de colonnes, il faudra utiliser les valeurs `'nbl'` et `'nbc'`

- En python pour savoir combien il y a d'éléments x dans une chaîne `S` (ou dans une liste `L`) on utilisera la commande `S.count(x)` ou `L.count(x)`

- Pour savoir si la variable `x` est un `int` on a le choix entre les 2 commandes suivantes

  - `type(x) == int` ancienne syntaxe python2.7

  - `isinstance(x, int)` nouvelle syntaxe python3.+

- Je vous encourage à augmenter le code de tests qui se trouve après la ligne

  ```
  if __name__ == ...
  ```

  pour vérifier pas à pas votre code. On fait un cas, on vérifie que la détection de ce cas est correct et on avance à l'étape suivante.