Aide Jalon 01

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

Rev. 1 : 21 Janvier 2022

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
```

1.1 Les pièges à éviter

Les points à ne pas oublier

- Vous ne devez pas travailler pour le cas de base (damier 3 x 3) mais pour n'importe quel damier
- Certaines informations ne sont pas pertinentes pour la fonction de validation :
  - **hexapawn** : vous n'avez pas besoin de savoir si le damier est plan ou pas car cela n'impacte que les actions autorisées. Vous n'avez pas besoin de la règle indiquant si on doit ou pas manger en priorité, cela n'impacte que les actions.
  - **morpion** : vous avez besoin de savoir si phase vaut 0 ou pas, car cela impacte sur la durée maximale de la partie – la durée est le nombre de coups et non une notion d'écoulement du temps

- Vous ne devez pas travailler sur 'X' et 'O' mais sur le fait que la variable `self.PAWN` contient toujours en case 0 le symbole pour le vide, que la case 1 est le symbole du premier joueur et la case 2 le symbole du second joueur.

1.2 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'`
  Pour savoir quels sont les paramètres disponibles la commande est `key_parameter` elle est utilisée dans les testcode pour que vous voyiez quels sont les paramètres disponibles.

- 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.+

- Quand on veut vérifier que tout caractère d'une chaîne `S` possède la propriété **p** on écrit

```
for x in S:
    if not p(x): return False
```

- En python, que ce soit une liste ou une chaîne de caractères (en fait tout objet dit `Enumerable`) on peut accéder à un élément de la structure avec l'utilisation de la syntaxe `[p]` où p désigne la position. Si $p \geq 0$, on lit de la gauche vers la droite en commençant à 0. Si $p < 0$ on lit de la droite vers la gauche en commençant à -1

```
S = "abcdefghij"
S[3] == "d"
S[-3] == "g"
S[len(S)-1] == S[-1]
S[0] == S[-len(S)]
```

- Une chaîne de caractères en python peut être découpée avec la notation `[x:y]` « slicing ». On sait que l'information fournie est une chaîne de longueur nbl × nbc on peut donc la découper en nbl blocs de longueur nbc. Par exemple 5 lignes de 3 colonnes, va correspondre à une chaîne S de longueur 15. Pour accéder au différentes lignes on aura

```
S[:3] # première ligne
S[3:6] # seconde ligne
S[6:9] # troisième ligne
S[9:12] # quatrième ligne
S[12:] # cinquième ligne
```

- Le « slicing » est une opération très puissante, avec sa notation `[a:b:i]` qui dit qu'on souhaite regarder la zone entre l'index a et $b-1$ et en ne regardant que toutes les i positions. Ainsi, en reprenant l'exemple d'une chaîne S codant l'information 5 lignes de 3 colonnes, si l'on souhaite récupérer les informations qui sont en 2de colonne on utilisera `S[1::nbc]` qui se lit « regarde la chaîne S à partir du caractère en position 1 (2nde position), jusqu'à la fin et en ne regardant que les caractères $1, 4, 7, \ldots$ »

- 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.