Fiche TP01-B

marc-michel.corsini@u-bordeaux.fr

21 février 2015

Dans cette seconde partie, nous allons construire trois exemples de joueurs ainsi que les fonctions permettant de jouer au **Doo**. Toute cette partie se déroulera dans un nouveau fichier python. On utilise les mêmes notations que pour le TP01a et qui sont rappelées ci-après.

- **Pions** désigne l'ensemble { BLANCS, NOIRS, ROI, VIDE } ; de fait VIDE n'est pas un vrai pion c'est plutôt une absence de vrai pion. On notera « tpion » un élément de { BLANCS, NOIRS, ROI }
- un « tablier » est donc un élément de **Pions**^{12} et **Tabliers** désigne l'ensemble des tabliers possibles
- **Etats** est l'ensemble des états ou des configurations, un élément de l'ensemble sera noté « cfg »
- **String** l'ensemble des chaînes de caractères
- **Joueurs** désigne l'ensemble { J_ATT, J_DEF }
- **Coups** désigne l'ensemble des coups jouables (autorisés)
- **Positions** désigne l'ensemble des positions du tablier
- **Libres** désigne l'ensemble des positions où un joueur peut poser/déplacer un de ses pions

1 Nouveau fichier

Vous partirez du fichier `tp01b_skeleton.py` que vous recopierez ou renommerez (**Rappel** pas d'espace ni de caractères accentués, privilégiez les 26 lettres (majuscules ou minuscules) et les 10 chiffres).
Attention Les classes ont un nom qui commence par une majuscule, les fonctions ou méthodes ont un nom qui commence par une minuscule.

2 Joueurs

Trois classes de joueurs à définir :

1. `First` : choisit toujours le premier coup à sa disposition ;
2. `Random` : choisit un coup au hasard parmi les coups possibles ;
3. `Humain` : interface de saisie pour un coup entré au clavier

Chaque classe dérive de la classe `Player` et ne nécessite la définition que de son constructeur `__init__` et de la méthode `choixCoup`

- `__init__` **String** $\cup$ { None } $\rightarrow \emptyset$
  axiomes : crée un joueur dont le nom est la chaîne passée en paramètre, si pas de paramètre un nom par défaut est fourni
- `choixCoup` **Game** x **Joueurs** $\rightarrow$ **Coups** $\cup$ { None }
  axiomes :
  1. choixCoup(H,x) = None si et seulement si H.listeCoups(x) = []
  2. choixCoup(H,x) = y si et seulement si y $\in$ H.listeCoups(x)

Human
Pour cette classe, il n'y aucune différence, si ce n'est qu'il est souhaitable dans la méthode choixCoup de traduire la saisie humaine en une information compatible avec la syntaxe des coups. Il est nécessaire de boucler tant que l'utilisateur n'a pas choisi un coup autorisé. Vous pouvez utiliser les deux fonctions suivantes pour faciliter la communication - elles sont disponibles dans le fichier squelette fourni.

```
def humain2ordi(position):
    """ position est une chaine de 2 caracteres
        - le premier est un nombre entre 1 et 4
        - le second une lettre dans ABC
        renvoie le numéro de la position (entre 0 et 11)

        pos = humain2ordi( ordi2humain( pos ) )
    """
    lig = int(position[0])-1 # compte à partir de 0
    col = position[1].upper() # passage en majuscule
    return lig*3+"ABC".index(col)

def ordi2humain(position):
    """ position est un entier entre 0 et 11
        renvoie une chaine de 2 caracteres, le premier
        est un entier entre 1 et 4, le second une lettre dans ABC

        pos = ordi2humain( humain2ordi( pos ) )
    """
    i,j = 1+(position-(position%3))//3,position%3
    return "%d%s" % (i,"ABC"[j])
```

```
>>> humain2ordi( '2a' )
3
>>> ordi2humain( 3 )
'2A'
>>> ordi2humain( 0 )
'1A'
>>> humain2ordi( '1A' )
0
```

Exemples

```
if __name__ == "__main__":
    doo = Doo() # Remplacer Doo par le nom de la classe de votre jeu
    a = Random()
    b = First()
    print(a.nom,"en attaque",a(doo,J_ATT))
    print(a.nom,"en defense",a(doo,J_DEF))
    print( '='*10)
    print(b.nom,"en attaque",b(doo,J_ATT))
    print(b.nom,"en defense",b(doo,J_DEF))
```

```
mmc@vostro-mmc: Code python3 tp01b.py
ordi aleat en attaque (1, 2)
ordi aleat en defense None
==========
```

```
ordi first en attaque (1, 0)
ordi first en defense None
mmc@vostro-mmc: Code python3 tp01b.py
ordi aleat en attaque (2, 9)
ordi aleat en defense None
==========
ordi first en attaque (1, 0)
ordi first en defense None
mmc@vostro-mmc: Code python3 tp01b.py
ordi aleat en attaque (2, 0)
ordi aleat en defense None
==========
ordi first en attaque (1, 0)
ordi first en defense None
mmc@vostro-mmc: Code
```

3 manche

la fonction `manche` prend en entrée deux fonctions et retourne un entier entre 0 et 3. Les deux fonctions sont des instances de la classe `Player`, la première est associée au joueur `J_ATT`, la seconde au joueur `J_DEF` et renvoie le résultat de la confrontation sur le jeu de Doo. Si une fonction vaut `None` alors on crée un joueur `First`. L'algorithme `pseudo-python` est donc :

```
def manche(funA=None,funB=None):
    """ effectue le décompte des points de J_ATT pour une manche """
    if funA is None: funA = First()
    if funB is None: funB = First()
    x = Doo()
    afficher x
    jtrait = J_ATT
    initialiser l'historique
    while not x.finPartie( jtrait ) and not cycling( historique ):
        choisir le coup de jtrait
        mettre à jour l'historique
        calculer la prochaine configuration
        mettre a jour x
        afficher x
        jtrait = adversaire( jtrait )
    calculer les points de J_ATT
    renvoyer les points, le nombre de (demi-)tours et l'historique
```

Signature Player x Player → Points x Tours x **Coups** ^{n}
axiomes : Points est un nombre entre 0 et 3 ; Tours est un nombre supérieur à 7, le dernier élément est l'historique (liste `python`) de la suite de coups joués par les deux adversaires, sa longueur est le nombre de tours joués, et elle permet de reconstituer le déroulement de la manche. Ainsi on doit soit pouvoir détecter un cycle, soit vérifier que `finPartie` est vraie.

3.1 cycling

Petite fonction qui prend en paramètre une liste de **Coups** et qui renvoie vrai si un cycle a été détecté.
Une séquence $C_1 C_2 \dots C_{k-1} C_k$ est un cycle si :

1. $\forall i \in 1 \dots k, C_i \in$ **Coups** ;
2. $\forall i \in 1 \dots k, C_i$ n'est pas une prise
3. $C_1 = C_{k-1}$ et $C_2 = C_k$

Signature Coups$^n \rightarrow \mathbb{B}$
axiomes : Seul le dernier coup ajouté a pu déclencher la possibilité d'un cycle - sinon il aurait été détecté avant et la manche aurait été terminée. Les axiomes sont donc la vérification de la propriété de cycle avec C_{k-1} et C_k les deux derniers éléments de la liste.
Attention On ne cherche pas à vérifier que la liste de **Coups** reçue en entrée soit un cycle, mais bien qu'elle contienne (ou pas) un cycle passant par les deux derniers éléments de la liste.

```
def cycling( liste ):
    """ repere un eventuel cycle dans une liste de coups """
    if liste contient un cycle: return True
    else: return False
```

4 partie

La fonction `Partie` a 3 paramètres, la fonction du joueur qui sera le premier attaquant, la fonction du joueur qui sera le premier défenseur et un nombre minimum (valeur par défaut **1**) de manches à jouer, ce dernier paramètre ne sert à rien dans le jeu de Doo, il n'est là que pour garder une cohérence avec d'autres jeux.
Si le joueur est `None`, il faut le créer, ensuite il suffira d'appeler manche autant de fois que le besoin s'en fait sentir avec les bons paramètres (attention à l'ordre, le premier joueur est l'attaquant pour la manche). Reportez vous aux règles du jeu.
À la fin de la partie, on affichera le nom du gagnant, son score et le nombre de manches effectuées.
Partie renvoie un dictionnaire dont les index sont les numéros de manche, les valeurs étant la liste des coups au cours de chaque manche.
Signature Player x Player x $\mathbb{N} \rightarrow \mathbb{N}^2$ x Historique^p
axiomes Le nombre d'éléments p du dictionnaire (des manches jouées) est pair. Chaque liste du dictionnaire est une suite de coups menant à une fin de manche (testable par `finPartie`). La première valeur de retour est une paire des scores respectifs de chaque joueur, l'un des éléments de cette paire est un score supérieur ou égale à 5. Les deux valeurs de score sont différentes.

5 replay (facultatif)

Cette fonction prend en paramètre un historique et doit permettre de visualiser la suite des coups joués et leur effet au format compréhensible par l'humain (cf `ordi2humain`). C'est en fait un simple affichage pour permettre à un humain, d'analyser le déroulement d'une manche. Elle renvoie `True` si la simulation conduit à une fin de manche, renvoie `False` sinon.
Signature Coups $^n \rightarrow \mathbb{B}$
axiomes renvoie l'information concernant le déroulement d'une manche complète.

6 Validations

Si vous avez fait ce qui est demandé, le fichier de `validation01a` doit fonctionner sur votre second fichier, le fichier de `validation01b` est en cours de réalisation. Regardez sur le site pour vous assurez que vous disposez de la dernière version. Le fichier `validation01b` ne traite que la vérification des fonctions demandées. Elles (les fonctions) ne servent qu'à permettre de voir le déroulement d'une partie complète et ne servent pas directement l'introduction de l'aspect « intelligence artificielle » proprement dite qui sera développée dans les TPs suivants.