TDLOG – séance n° 3
Dominos : exercices

Xavier Clerc – xavier.clerc@enpc.fr
Mathieu Bernard - mathieu.a.bernard@inria.fr
Clémentine Fourrier - clementine.fourrier@inria.fr
Basile Starynkevitch - basile-freelance@starynkevitch.net

5 octobre 2020

À rendre au plus tard le 10 octobre 2020 sur educnet

Les exercices suivants sont à faire en binôme, et en anglais (noms des classes, fonctions, méthodes, variables et commentaires). Merci de placer en en-tête de votre fichier les noms des deux étudiants ainsi que le temps consacré aux exercices. S'agissant de la suite des exercices de la séance précédente, vous pouvez au choix repartir de votre propre programme ou du corrigé disponible sur educnet.

1 Classe `Domino`

La première étape consiste à créer une classe dont les instances permettront de représenter les dominos manipulés par le jeu. Cette classe, `Domino`, doit donc avoir deux attributs, pour les deux parties du domino et deux propriétés pour y accéder. Il est également possible de définir une troisième propriété dont la valeur est le nombre de points sur le domino. Toutes ces propriétés ne sont définies qu'en lecture.

Pour rendre la classe plus utile, il est demandé de définir deux méthodes *spéciales*, liées à la transformation en chaînes de caractères :

- `__repr__(self)` renvoie une chaîne de caractères correspondant à la représentation *Python* (cette méthode est par exemple utilisée par l'interpréteur pour afficher la valeur d'une variable) ;
- `__str__(self)` renvoie une chaîne de caractères correspondant à l'affichage à destination de l'utilisateur (cette méthode est par exemple utilisée par la fonction `print` pour afficher ses paramètres).

Dans notre cas, pour le domino 4-3, la première doit renvoyer la chaîne de caractères :

```
Domino(4, 3)
```

et la seconde :

```
+-----|-----+
|*   *|*    |
|     |  *  |
|*   *|    *|
+-----|-----+
```

Enfin, on demande de définir les méthodes *spéciales* suivantes :

- `__eq__(self, other)` qui doit renvoyer `True` si `self` et `other` sont égaux (utilisé par *Python* pour évaluer `==`) ;
- `__ne__(self, other)` qui doit renvoyer `True` si `self` et `other` sont différents (utilisé par *Python* pour évaluer `!=`).

Attention, on demande que l'égalité considère les deux parties du domino comme non ordonnées, pour traduire le fait que par exemple 2-3 et 3-2 sont en fait égaux.

2 Classe `Solitaire`

La seconde étape consiste à créer les classes nécessaires pour implémenter le reste du jeu. On demande explicitement la création d'une classe `Solitaire` pour représenter l'état du jeu et pouvoir y jouer. Le reste de la modélisation, c'est-à-dire d'éventuelles autres classes et les relations entre elles, est laissé libre.

La classe `Solitaire` doit disposer d'au moins deux attributs représentant respectivement le contenu de la main et de la pioche. Elle doit aussi proposer au minimum les méthodes suivantes :

- `is_game_won(self)` qui doit renvoyer `True` ssi la partie est terminée sur une victoire ;
- `is_game_lost(self)` qui doit renvoyer `True` ssi la partie est terminée sur une défaite ;
- `turn(self)` qui doit permettre de faire un tour de jeu (*i. e.* affichage de l'état de la partie, saisie de l'utilisateur et modification éventuelle de l'état) ;
- `play(self)` qui doit permettre d'enchaîner les tours de jeu jusqu'à la fin de la partie.

Au final, l'objectif est de disposer d'un code fonctionnellement équivalent à celui de la session précédente, mais se fondant sur une modélisation objet. En conséquence, la plupart (voire toutes) des fonctions de la séance précédente vont disparaître (*e. g.* celles d'affichage), leurs codes étant adaptés pour écrire les méthodes des classes développées.