Apprendre a programmer avec le module Cartes

1 Présentation

Le module `Cartes` est un module développé à l'université de Lille 1 pour l'initiation à la programmation avec des étudiants de première année de licence. Il n'est donc pas livré avec les distributions standards de Python, et il est nécessaire de l'installer séparément. Nous supposons ici que cette installation est faite [1].

Le module `Cartes` offre une extension du langage Python afin d'écrire des programmes destinés à un robot virtuel manipulateur de cartes.

Le domaine de travail du robot est constitué de tas, numérotés 1,2,3,4, sur lesquels sont empilées des cartes à jouer. Ces cartes ont les couleurs habituelles (♣, ♢, ♡ et ♠) et les valeurs habituelles (par ordre croissant : as, deux, trois, ..., dix, valet, dame, roi). Les cartes proviennent de plusieurs jeux, il est donc possible de trouver plusieurs exemplaires d'une même carte.

Chacun des quatre tas peut contenir un nombre quelconque de cartes, y compris aucune.

2 Utilisation du module `Cartes`

Pour utiliser ce module, deux possibilités :

1. ```
   from Cartes import *
   ```
2. ```
   import Cartes
   ```

Avec la seconde méthode, les noms des différentes variables, fonctions et procédures déclarées dans ce module doivent être préfixés par le nom du module. Par exemple, l'instruction `init_tas` pour initialiser un tas (cf ci-dessous) doit être invoquée par le nom `Cartes.init_tas`. Pour se dispenser de préfixer systématiquement tous les noms par celui du module, il suffit de choisir la première méthode d'importation

Dans toute la suite, nous supposerons que c'est la première méthode d'importation qui a été choisie.

Ceci accompli, on peut faire référence à tous les éléments déclarés dans le module en utilisant leur nom non préfixé par celui du module.

```
>>> init_tas (1,"T")
```

3 Le langage du module `Cartes`

3.1 Description de tas

Les quatre tas sont numérotés de 1 à 4 de gauche à droite, et peuvent contenir des cartes à jouer d'un jeu de 52 cartes, en nombre quelconques choisies au hasard. Il est possible d'avoir présentes plusieurs occurrences d'une même carte. Les problèmes que nous traiterons nécessitent de pouvoir imposer certaines configurations de l'état de ces quatre tas.

Afin de décrire en début de programme, la configuration initiale des tas de cartes, le module `Cartes` offre une procédure `init_tas`. Cette procédure [2] est une fonction qui ne renvoie aucune valeur mais a pour

1. Il est possible d'installer une version de ce module avec l'utilitaire `pip3` : `pip3 install CartesInitProg`
2. nous appelons procédure les fonctions qui ne renvoient aucune valeur, mais possèdent un effet de bord.

effet de bord d'initialiser le tas dont le numéro est passé en (premier) paramètre avec des cartes selon une description fournie en (second) paramètre.

```
init_tas(num_tas, descr)
```

où `num_tas` est le numéro du tas à initialiser, et `descr` est une *chaîne de description* du tas.

La chaîne de description, lue de gauche à droite, indique les cartes d'un tas du bas vers le haut avec les conventions suivantes :

- la chaîne vide désigne un tas vide ;
- les lettres `T`, `K`, `C`, `P` représentent respectivement : ♣, ♢,♡, ♠ ;
- si A et B sont des chaînes de description, la chaîne AB est aussi une chaîne de description qui représente les cartes de A surmontées de celles de B.
- si A et B sont des chaînes de description, la chaîne $A + B$ est aussi une chaîne de description qui représente un choix entre les cartes de A ou celles de B ;
- si A est une chaîne de description, $[A]$ représente la répétition des cartes de A un nombre quelconque (défini de manière aléatoire) de fois (y compris zéro) ;
- les parenthèses sont autorisées pour éviter les ambiguïtés.

Exemples :

1. `init_tas(1, "")` déclare le tas numéro 1 vide.
2. l'instruction `init_tas(2, "TCK")` décrit le tas numéro 2 contenant de bas en haut un ♣, un ♡ et un ♢.
3. l'instruction `init_tas(3, "T+P")` décrit le tas numéro 3 contenant une carte qui est soit un ♣ soit un ♠.
4. `init_tas(4, "[T+P]")` décrit le tas numéro 4 contenant un nombre quelconque, non déterminé, de cartes de couleur ♣ ou ♠.
5. l'instruction `init_tas(1,"T+[P]CC")` décrit le tas numéro 1 contenant soit une seule carte de couleur ♣, soit un nombre quelconque de ♠ surmonté de deux ♡.
6. l'instruction `init_tas(1,"(T+[P])CC")` décrit le tas numéro 1 contenant soit trois cartes un ♣ surmonté de deux ♡, soit un nombre quelconque de ♠ surmonté de deux ♡. À noter qu'une autre façon de décrire la même initialisation est d'utiliser la chaîne `"TCC+[P]CC"`.

Une instance possible de l'état des quatre tas après exécutions des instructions numérotées 1, 2, 3 et 5 est donnée par la partie gauche de la figure 1 page 3.

Remarque : Tout programme de résolution d'un problème sur les cartes doit débuter par une initialisation des quatre tas à l'aide de l'instruction `init_tas`. Une fois cette initialisation effectuée, cette instruction n'est plus utilisée. Et comme, on le verra par la suite, la seule action que le robot peut effectuer est le déplacement d'une carte d'un tas vers un autre, il s'en suit qu'après l'étape d'initialisation, le nombre de cartes globalement présentes sur les quatre tas ne varie pas.

3.2 Action

La seule action permettant de modifier l'état des tas est le déplacement de la carte située au sommet d'un tas vers le sommet d'un autre tas. Cette action est déclenchée par un appel à la procédure `deplacer_sommet` paramétrée par deux numéros de tas :

```
deplacer_sommet(depart, arrivee)
```

où `depart` est le numéro du tas duquel on prend une carte, et `arrivee` est celui du tas sur lequel on la pose.

Exemple : L'instruction `deplacer_sommet(2, 4)` a pour effet de déplacer la carte au sommet du tas 2 pour la poser au sommet du tas 4. Appliquée à l'état des tas décrits par la partie gauche de la figure 1, cette instruction transforme les tas 2 et 4 comme le montre la partie droite.

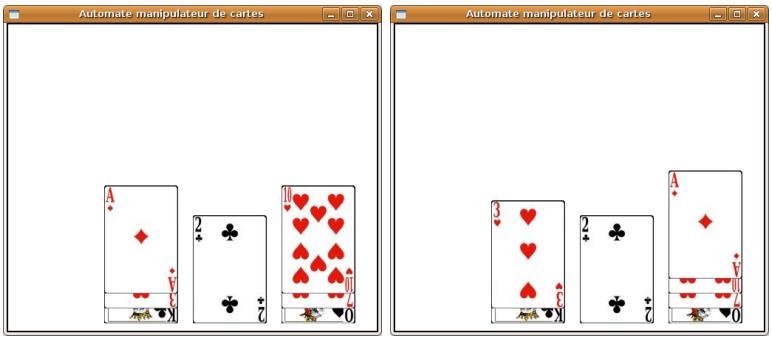

FIGURE 1 – Transformation des tas par déplacement d'une carte

Avertissement : Il n'est pas permis de déplacer une carte située sur un tas vide. Toute tentative d'action `deplacer_sommet(n,p)` déclenche l'exception `AssertionError: Le tas source est vide` si le tas n est vide.

3.3 Tests sur les tas

Certains traitements nécessitent des tests. Pour cela on dispose de fonctions.

3.3.1 Test de vacuite

Pour tester si un tas est vide, on fera appel à la fonction `tas_vide`.

```
tas_vide(num_tas)
```

qui donne la valeur booléenne `Vrai` (`True` en PYTHON) si le tas numéro `num_tas` est vide, et la valeur `Faux` (`False` en PYTHON) dans le cas contraire.

On peut faire le test contraire en faisant appel à la fonction

```
tas_non_vide(num_tas)
```

3.3.2 Test sur la couleur

Les quatre couleurs sont décrites dans le module `Cartes` par les quatre variables : `TREFLE`, `CARREAU`, `COEUR`, `PIQUE`.

Il est possible de connaître la couleur au sommet d'un tas en faisant appel à la fonction :

```
couleur_sommet(num_tas)
```

qui donne la couleur de la carte située au sommet du tas numéro `num_tas`.

Exemple : Une utilisation possible de cette fonction est dans les instructions conditionnelles.

```
if couleur_sommet(2) == PIQUE:
    ...
```

Avertissement : Il est normalement dénué de sens de tester la couleur de la carte située au sommet d'un tas vide. Toute tentative d'appel à `couleur_sommet(num_tas)` sur un tas vide déclenche l'exception : `AssertionError: Le tas est vide`.

Quatre autres fonctions permettent aussi de tester la couleur du sommet d'un tas

```
sommet_trefle(num_tas)
sommet_carreau(num_tas)
sommet_coeur(num_tas)
sommet_pique(num_tas)
```

qui donnent la valeur `Vrai` si le sommet du tas `num_tas` est un ♣, ♢, ♡,♠ et `Faux` dans le cas contraire.

Elles sont soumises à la même contrainte d'utilisation que la fonction `couleur_sommet`, et déclenchent la même exception en cas de non respect de cette contrainte.

3.4 Comparaison des valeurs

Une fonction permet de comparer les valeurs des cartes au sommet de deux tas :

```
superieur(num_tas1, num_tas2)
```

qui donne la valeur `Vrai` si la carte au sommet du tas numéro `num_tas1` a une valeur supérieure ou égale à celle du tas numéro `num_tas2`, et la valeur `Faux` dans le cas contraire.

Avertissement : On ne peut comparer les cartes situées au sommet de deux tas que s'ils ne sont pas vides. Tout appel à `superieur(num_tas1, num_tas2)` lorsque l'un des deux tas est vide déclenche l'exception `AssertionError: Le tas est vide`.

3.5 Contrôle de l'affichage

3.5.1 Modes d'affichage

À l'exécution, l'affichage peut se faire de trois façons :

1. mode graphique : Dans ce mode, on visualise l'évolution des tas de cartes dans une fenêtre graphique. C'est le mode par défaut.
2. mode texte : Dans ce mode, on visualise l'évolution des tas de cartes dans le terminal où s'exécute le programme. Voici un exemple d'affichage textuel produit par l'exécution d'une instruction :

```
>>> affichage_en_mode_texte()
>>> init_tas(1, "")
[]
[]
[]
[]
----
>>> init_tas(2, "C")
[]
[Carte<COEUR, 2>]
[]
[]
----
>>> init_tas(3, "")
[]
[Carte<COEUR, 2>]
```

```
[]
[]
----
>>> init_tas(4, "T")
[]
[Carte<COEUR, 2>]
[]
[Carte<TREFLE, 13>]
----
>>> deplacer_sommet(4, 1)
[Carte<TREFLE, 13>]
[Carte<COEUR, 2>]
[]
[]
----
```

3. **mode texte et graphique** : Dans ce mode, on visualise l'évolution des tas de cartes à la fois dans le terminal et dans une fenêtre graphique.

Le mode d'affichage par défaut est le mode graphique. L'instruction

```
affichage_en_mode_texte()
```

permet de passer en mode texte. L'instruction

```
affichage_en_mode_graphique()
```

permet de passer en mode graphique. Et l'instruction

```
affichage_en_mode_texte_et_graphique()
```

permet de combiner les deux modes texte et graphique.

Remarque : Le mode texte permet de conserver une trace de l'exécution d'un programme dans un fichier texte. Par exemple, si un fichier nommé `exo.py` contient un programme de résolution d'un problème sur les cartes, et s'il contient une instruction demandant un affichage textuel, alors dans un terminal la commande

```
$ python3 exo.py > exo.result
```

produit un fichier texte nommé `exo.result` contenant toutes les instructions exécutées suivies de l'état des tas de cartes qu'elles transforment.

3.5.2 Vitesse d'execution

Un délai est imposé entre l'exécution des instructions de manipulation des tas. Le temps d'attente peut être ajusté en fonction des besoins. Le module fournit une fonction (`fixer_delai(delai)`) qui permet de fixer cette temporisation.

Pour augmenter la vitesse d'exécution, il suffit de diminuer la valeur réelle.

```
fixer_delai(0.1)
```

L'instruction précédente permet de fixer la durée de l'attente à un dixième de seconde.

3.5.3 Pause

L'instruction `pause` permet de faire une pause durant l'exécution d'un programme. Elle s'utilise en communiquant un message en paramètre, message qui sera imprimé dans le terminal lors de la pause. La pause s'arrête dès l'appui sur la touche Entrée.

```
pause("un petit repos bien merité")
```

4 Exercices

Ils sont classés en facile (☺), moyen (😐)et difficile (☹)(les premiers sont très faciles).

4.1 Descriptions de tas

Exercice 1

Pour chacune des descriptions qui suivent, donnez l'instruction d'initialisation du tas qui convient :

- le tas 1 contient une carte de couleur ♣ ;
- le tas 1 contient une carte de couleur ♣ ou ♠ ;
- le tas 1 contient une carte de couleur quelconque ;
- le tas 1 contient deux cartes de couleur ♡ ;
- le tas 1 contient une carte de couleur ♡ surmontée d'un ♢ ;
- le tas 1 contient un nombre quelconque de ♠ ;
- le tas 1 contient un nombre quelconque de ♠ ou bien un nombre quelconque de ♡ ;
- le tas 1 contient un nombre quelconque de cartes de couleur ♠ ou ♡ ;
- le tas 1 contient un nombre quelconque de cartes de couleur quelconque ;
- le tas 1 contient au moins un carreau ;
- le tas 1 contient un ♣ surmonté soit d'un nombre quelconque de ♡, soit d'un nombre quelconque non nul de ♠ ;
- le tas 1 contient un nombre pair de ♡ ;
- le tas 1 contient un nombre impair de ♡ ;
- le tas 1 contient un nombre pair de ♣ ou un nombre multiple de 3 de ♠ ;
- les deux cartes extrêmes du tas 1 (la plus basse et la plus haute) sont des ♣, entre les deux il y a un nombre quelconque de successions de deux cartes de couleur ♢♡.

4.2 Séquence

Dans tous les exercices qui suivent, l'énoncé décrit une situation initiale des quatre tas de cartes (dans la syntaxe du module `Cartes`), et la situation finale à atteindre.

Exercice 2 (☺)

Situation initiale :

Tas 1 : `"TT"` Tas 2 : `""`
Tas 3 : `""` Tas 4 : `""`

Situation finale :

Tas 1 : `""` Tas 2 : `"TT"`
Tas 3 : `""` Tas 4 : `""`

Exercice 3 (☺)

Situation initiale :

Tas 1 : `"TK"` Tas 2 : `""`
Tas 3 : `""` Tas 4 : `""`

Situation finale :

Tas 1 : `"KT"` Tas 2 : `""`
Tas 3 : `""` Tas 4 : `""`

Exercice 4 (☺)

Situation initiale :
Tas 1 : "TKTK" Tas 2 : ""
Tas 3 : "" Tas 4 : ""
Situation finale :
Tas 1 : "KKTT" Tas 2 : ""
Tas 3 : "" Tas 4 : ""

Exercice 5 (☺)
Situation initiale :
Tas 1 : "TKCP" Tas 2 : ""
Tas 3 : "" Tas 4 : ""
Situation finale :
Tas 1 : "PCKT" Tas 2 : ""
Tas 3 : "" Tas 4 : ""

4.3 Conditionnelle

Exercice 6 (☺)
Situation initiale :
Tas 1 : "T+P" Tas 2 : ""
Tas 3 : "" Tas 4 : ""
Situation finale :
Tas 1 : "" Tas 2 : "[T]"
Tas 3 : "[P]" Tas 4 : ""

Exercice 7 (☺)
Situation initiale :
Tas 1 : "(T+K+C+P)(T+K+C+P)" Tas 2 : ""
Tas 3 : "" Tas 4 : ""
Situation finale :
Tas 1 : "" Tas 2 : ""
Tas 3 : "(T+K+C+P)(T+K+C+P)"↑ Tas 4 : ""
le symbole ↑ signifiant que les cartes sont dans l'ordre croissant (i.e. la carte du dessous a une valeur inférieure ou égale à celle du dessus).

Exercice 8 (☺)
Situation initiale :
Tas 1 : "T+K+C+P" Tas 2 : ""
Tas 3 : "" Tas 4 : ""
Situation finale :
Tas 1 : "[T]" Tas 2 : "[K]"
Tas 3 : "[C]" Tas 4 : "[P]"

Exercice 9 (☺)
Situation initiale :
Tas 1 : "(T+K+C+P)(T+K+C+P)" Tas 2 : ""
Tas 3 : "" Tas 4 : ""
Situation finale :
Tas 1 : "[K+C]" Tas 2 : "[T+P]"
Tas 3 : "" Tas 4 : ""

4.4 Itération

Exercice 10 (☺)

Situation initiale :

Tas 1 : `"[T]"` Tas 2 : `""`
Tas 3 : `""` Tas 4 : `""`

Situation finale :

Tas 1 : `""` Tas 2 : `"[T]"`
Tas 3 : `""` Tas 4 : `""`

Exercice 11 (☺)

Situation initiale :

Tas 1 : `"[K+C][T+P]"` Tas 2 : `""`
Tas 3 : `""` Tas 4 : `""`

Situation finale :

Tas 1 : `"[T+P][K+C]"` Tas 2 : `""`
Tas 3 : `""` Tas 4 : `""`

Exercice 12 (☺)

Situation initiale :

Tas 1 : `"[K]"` Tas 2 : `"[T]"`
Tas 3 : `""` Tas 4 : `""`

Situation finale :

Tas 1 : `"[K]"` Tas 2 : `""`
Tas 3 : `"[KT]"` Tas 4 : `""`

ou bien :

Tas 1 : `""` Tas 2 : `"[T]"`
Tas 3 : `"[KT]"` Tas 4 : `""`

Exercice 13 (☺)

Situation initiale :

Tas 1 : `"[T]"` Tas 2 : `""`
Tas 3 : `""` Tas 4 : `""`

Situation finale :

Tas 1 : `""` Tas 2 : `"[T]"`
Tas 3 : `"[T]"` Tas 4 : `""`

le nombre de cartes des deux tas 2 et 3 différant d'au plus 1 dans la situation finale.

Exercice 14 (☺)

Situation initiale :

Tas 1 : `"[T]"` Tas 2 : `"[K]"`
Tas 3 : `"[P]"` Tas 4 : `""`

Situation finale :

Tas 1 : `""` Tas 2 : `"[T]"`
Tas 3 : `"[K]"` Tas 4 : `"[P]"`

En faire deux versions, la seconde utilisant une procédure `vider_tas(depart,arrivee)` qui vide le tas `depart` sur le tas `arrivee`.

Exercice 15 (°°)

Situation initiale :

Tas 1 : "[T]" Tas 2 : "[K]"
Tas 3 : "[C]" Tas 4 : "[P]"

Situation finale :

Tas 1 : "[P]" Tas 2 : "[T]"
Tas 3 : "[K]" Tas 4 : "[C]"

Exercice 16 (°°)

Situation initiale :

Tas 1 : "[T][K][C][P]" Tas 2 : ""
Tas 3 : "" Tas 4 : ""

Situation finale :

Tas 1 : "[P][C][K][T]" Tas 2 : ""
Tas 3 : "" Tas 4 : ""

Exercice 17 (°°)

Situation initiale :

Tas 1 : "[T]" Tas 2 : "[K]"
Tas 3 : "[P]" Tas 4 : ""

Situation finale :

Tas 1 : "" Tas 2 : ""
Tas 3 : "" Tas 4 : "[TKP][XY][Z]"

où X et Y désignent les deux couleurs restantes lorsque l'une des couleurs manque, et Z désigne la couleur restante lorsque X ou Y manque.

Exercice 18 (°°)

Situation initiale :

Tas 1 : "[T+K+C+P]" Tas 2 : ""
Tas 3 : "" Tas 4 : ""

Situation finale :

Tas 1 : "[T]" Tas 2 : "[K]"
Tas 3 : "[C]" Tas 4 : "[P]"

Exercice 19 (°°)

Situation initiale :

Tas 1 : "T[T]" Tas 2 : ""
Tas 3 : "" Tas 4 : ""

Situation finale :

Tas 1 : "" Tas 2 : "T"$-$
Tas 3 : "[T]" Tas 4 : ""

le symbole $-$ indique que la carte est de valeur minimale.

Exercice 20 (°°)

Situation initiale :

Tas 1 : "[T]" Tas 2 : ""
Tas 3 : "" Tas 4 : ""

Situation finale :

Tas 1 : "" Tas 2 : "[T]$\uparrow$
Tas 3 : "" Tas 4 : ""

le symbole $\uparrow$ signifiant que les cartes sont rangées par ordre croissant de valeurs de bas en haut.

Exercice 21 (°°)

Situation initiale :
Tas 1 : "[T+K]" Tas 2 : ""
Tas 3 : "" Tas 4 : ""
Situation finale :
Tas 1 : "[X][Y]" Tas 2 : ""
Tas 3 : "" Tas 4 : ""
Le symbole X désigne la couleur (♣ ou ♢) la plus nombreuse, l'autre couleur étant désignée par Y.

Exercice 22 (☹)

Situation initiale :
Tas 1 : "K[T]" Tas 2 : ""
Tas 3 : "" Tas 4 : ""
Situation finale :
Tas 1 : "[T+K]" Tas 2 : "[T+K]"
Tas 3 : "[T+K]" Tas 4 : "[T+K]"
les trèfles étant équitablement répartis sur les quatre tas, l'unique carreau se trouvant n'importe où.

Remarque : ce problème est infaisable sans le carreau.

Exercice 23 (☹)

Situation initiale :
Tas 1 : "[T]" Tas 2 : "[K]"
Tas 3 : "[C]" Tas 4 : "[P]"
Situation finale :
Tas 1 : "[T]↑" Tas 2 : "[K]↑"
Tas 3 : "[C]↑" Tas 4 : "[P]↑"
le symbole ↑ signifiant que les cartes sont rangées par ordre croissant de valeurs de bas en haut.

4.5 Fonctions et procédures

Exercice 24

Soit la fonction définie en PYTHON par

```
def meme_couleur(couleur, tas):
    """
    renvoie
      - True si la carte au sommet du tas
            tas est de couleur couleur
      - Faux sinon
    """
    return couleur == couleur_sommet(tas)
```

Question 1 Quel problème pose cette fonction si lors d'un appel le tas `tas` passé en paramètre est vide ?

Question 2 Comment modifier la fonction `meme_couleur` pour qu'elle renvoie la valeur faux si le tas `tas` est vide.

Exercice 25 *Égalité de deux cartes*

Question 1 Écrivez une fonction qui teste l'égalité de la valeur de deux cartes situées au sommet de deux tas que l'on supposera non vide.

Question 2 Puis écrivez une fonction qui teste l'égalité de deux cartes (valeur et couleur) situées au sommet de deux tas supposés non vides.

Exercice 26 *Inverser l'ordre des cartes d'un tas*

Il s'agit d'écrire une procédure `inverser_tas1` pour inverser l'ordre des tas du tas 1. Par exemple, si on a `Tas 1:"TCCPK"` alors après exécution de l'instruction `inverser_tas1 ()`, on doit avoir `Tas 1:"KPCCT"`.

Question 1 Faîtes-le avec l'hypothèse que les autres tas sont vides.

Question 2 Réaliser la procédure sans supposer que les autres tas sont vides.
