Initiation à Python

LP ESSIG

2012 – 2013

1 Un jeu de cartes

Le but de ce TP est d'implémenter une classe permettant de distribuer un jeu de cartes. Puis d'utiliser l'héritage afin de pouvoir jouer à différents jeux.

2 Créer une classe

La classe `JeuDeCartes` va représenter la pioche. Cette classe va rester générique pour pouvoir représenter des comportements communs à tous les jeux de cartes. Elle ne va donc pas contenir de cartes directement.

Les cartes seront représentées par un couple valeur/couleur. Les valeurs allant de 1 à 13 (pour le roi) et les couleurs étant représentées par les lettres C, P, K, T (ou toute autre représentation qui vous paraît plus logique).

Écrivez la classe `JeuDeCartes` qui remplit les conditions suivantes :

- Elle contient une liste (initialement vide) de cartes.
- Elle n'initialise pas les cartes de la pioche.
- Elle possède une méthode permettant de mélanger les cartes de la pioche. Utilisez les méthodes connues de manipulation de liste et la fonction `choice` du module `random`.
- Elle possède une méthode statique qui prend un tuple en paramètre et affiche le nom de la carte.
- Elle possède une méthode permettant de tirer une carte. La carte tirée est la première de la liste et elle est ensuite retirée de la liste pour être renvoyé à l'appelant. S'il n'y à plus de carte à tirer, une exception doit être générée.
- Elle possède une méthode permettant de couper le jeu : la fin de la liste passe au début et le début à la fin. La fonction `randint` du module `random` vous permet d'obtenir un entier aléatoire dans l'intervale $[a, b]$.

2.1 Des jeux qui ont des cartes

On ne peut pas utiliser la classe `JeuDeCartes` pour jouer aux cartes puisqu'elle n'initialise pas les cartes de la pioche. Elle reste donc vide. Il faut donc créer de nouvelles classes dont les constructeurs vont servir à mettre des cartes dans la pioche.

Créez 3 classes, qui héritent chacune de `JeuDeCartes`, pour représenter 3 types de jeux différents : `JeuDe52` qui contient les cartes de 1 à 13 pour les 4 couleurs, `JeuDe32` qui contient les cartes de 7 à 13 plus l'as pour les 4 couleurs et `JeuDeTarot` qui contient les cartes de 1 à 14 (ajout du cavalier) pour 4 couleurs et de 0 (excuse) à 21 pour l'atout. Modifiez la méthode qui affiche le nom de la carte dans la classe `JeuDeTarot` pour tenir compte des cartes spéciales.

2.2 Testez

Créez différents jeux de cartes, mélangez-les, tirez les cartes une par une et affichez-les pour vérifier que tout fonctionne.

3 Jouer

Écrire une classe `Joueur` pour utiliser les jeux de carte. Chaque joueur est défini par un nom et une main (initialement vide) de cartes. Un joueur a aussi accès à une pioche.

Chaque joueur peut :

— Piocher une carte pour l'ajouter à sa main.
— Jouer une carte. La carte est retirée de la main et retournée à l'appelant. La carte à jouer est désignée par son indice dans la main du joueur.
— Annoncer son jeu : le joueur affiche à l'écran chaque carte de sa main et la position qu'elle occupe.
— Retourner le nombre de cartes qu'il a en main.

Écrivez ensuite une fonction prenant deux noms de joueurs en paramètres pour simuler un jeu :

— Le jeu se joue avec 52 cartes.
— Chaque joueur commence la partie en piochant 3 cartes.
— À chaque tour, le jeu du joueur en cours est affiché.
— Chaque joueur ne peut avoir que 7 cartes en main au maximum.
— À son tour un joueur peut soit piocher, soit joueur une carte (utilisez `input` pour gérer les choix).
— Si une carte a été posée par le joueur précédent, on ne peut poser une carte que si elle est de la même couleur et de plus haute valeur (à vous de décider si l'as vaut 1 ou 14).
— Si le joueur suivant ne pose pas de carte alors qu'on en a posé une, on marque un point et le jeu est réinitialisé (le prochain joueur à vouloir poser une carte pourra mettre celle qu'il veut).
— Lorsque la pioche est vide, le joueur qui a le plus de points a gagné.

4 Aller plus loin ?

Si vous souhaitez créer une fonction pour gérer des jeux plus complexes avec les différents jeux de cartes, voici quelques pistes :

— Utilisez une classe `Pli` pour stocker les différents plis remportés par chacune des équipes. Cette classe possède une liste par équipe qui sera remplie avec les différentes cartes remportées par chacun des joueurs d'une équipe.
— Utilisez une méthode `réunir` dans la classe `Pli` qui retourne la liste de tous les plis de toutes les équipes réunis. Cette liste pourra être réaffectée directement à l'objet responsable de la pioche.
— Dans la fonction qui décrit le déroulement du jeu, stockez les joueurs autour de la table dans une liste. Utilisez une variable pour connaître l'indice de celui qui doit jouer en premier et utilisez une fonction annexe qui, étant donnée une liste de cartes (la première jouée étant en position 0), renvoie l'indice de celle qui remporte le pli. Utilisez le modulo pour gérer les joueurs de façon circulaire.

5 Une correction possible

```
#!/usr/bin/python

import random

class NoMoreCards(Exception):
  pass

class JeuDeCartes:
  """Classe permettant de définir les actions génériques
  sur un jeu de cartes.
  """

  pioche = []

  def mélanger(self):
    """Mélange la pioche.

    Pourait utiliser la fonction random.shuffle à la place.
    """
    melange = []
    while len(self.pioche):
      carte = random.choice(self.pioche)
      self.pioche.remove(carte)
      melange.append(carte)
    self.pioche = melange

  def couper(self):
    """Place la fin de la pioche au début."""
    if len(self.pioche) > 1:
      coupe = random.randint(1,len(self.pioche-1))
      self.pioche = self.pioche[coupe:] + self.pioche[:coupe]

  def tirer(self):
    """Retourne la première carte de la pioche qui
    est, ensuite, retirée de la pioche.

    Génère une NoMoreCards exception s'il n'y a plus de
    cartes dans la pioche.
    """
    if self.pioche:
      return self.pioche.pop(0)
    raise NoMoreCards

  @staticmethod
  def afficheCarte(carte):
    """Affiche la carte réprésentée par le tuple contenu dans carte."""
    conversion = {
      11: "Valet",
      12: "Dame",
      13: "Roi",
```

```
      14: "As",
      "C": "Cœur",
      "P": "Pique",
      "K": "Carreau",
      "T": "Trèfle",
    }
    # unpacking
    valeur, couleur = carte
    if valeur == 1:
      valeur = 14
    print(conversion[valeur] if valeur > 10 else valeur, "de", conversion[couleur])

class JeuDe52(JeuDeCartes):
  """Classe spécialisée dans les jeux de 52 cartes."""

  def __init__(self):
    """Remplit la pioche"""
    for couleur in "CPKT":
      for valeur in range(1,14):
        self.pioche.append((valeur,couleur))

class JeuDe32(JeuDeCartes):
  """Classe spécialisée dans les jeux de 32 cartes."""

  def __init__(self):
    """Remplit la pioche"""
    for couleur in "CPKT":
      for valeur in range(7,14):
        self.pioche.append((valeur,couleur))
      self.pioche.append((1,couleur))

class JeuDeTarot(JeuDeCartes):
  """Classe spécialisée dans les jeux de tarot."""

  def __init__(self):
    """Remplit la pioche"""
    for couleur in "CPKT":
      for valeur in range(1,15):
        self.pioche.append((valeur,couleur))
    # Atout
    for valeur in range(22):
      self.pioche.append((valeur,"A"))

  @staticmethod
  def afficheCarte(carte):
    """Affiche la carte réprésentée par le tuple contenu dans carte."""
    conversion = {
      11: "Valet",
      12: "Cavalier",
      13: "Dame",
      14: "Roi",
```

```
      15: "As",
      "C": "Cœur",
      "P": "Pique",
      "K": "Carreau",
      "T": "Trèfle",
    }
    # unpacking
    valeur, couleur = carte
    if couleur != "A":
      if valeur == 1:
        valeur = 15
      print(conversion[valeur] if valeur > 10 else valeur, "de", conversion[couleur])
    else:
      if not valeur:
        print("Excuse")
      else:
        print(valeur, "d'atout")

class Joueur:
  """Un joueur de carte défini par son nom et sa main.

  La pioche de cartes lui est aussi fournit.
  """

  main = []
  def __init__(self, nom, pioche):
    self.nom = nom
    self.pioche = pioche

  def piocher(self):
    self.main.append(self.pioche.tirer())

  def jouer(self,indice):
    return self.main.pop(indice)

  def nombreDeCartes(self):
    return len(self.main)

  def afficherJeu(self):
    """Correspondance entre la main et les indices des
    cartes qui s'y trouvent.
    """
    for i in range(self.nombreDeCartes()):
      print("En position", i, ":")
      pioche.afficheCarte(self.main[i])

  def __str__(self):
    """Permet de simplifier la gestion des fins de partie."""
    return self.nom + " a gagné !"

class Triche(Exception):
```

```
  pass

def jeu(nom_joueur_1, nom_joueur_2):
  """En cas de triche, la partie est arrêtée."""
  pioche = JeuDe52()
  pioche.mélanger()
  joueur1 = Joueur(nom_joueur_1, pioche)
  joueur2 = Joueur(nom_joueur_2, pioche)
  for _ in range(3):
    joueur1.piocher()
    joueur2.piocher()
  score = [0, 0]
  last_card = None
  player1 = True
  try:
    while True:
      joueur = player1 and joueur1 or joueur2
      print("Carte jouée :", end=" ")
      if not last_card:
        print("Aucune")
      else:
        pioche.afficheCarte(last_card)
      print("Joueur", joueur.nom, "voici votre main.")
      joueur.afficherJeu()
      index = int(input("Jouer une de ces cartes ou piocher (tout autre nombre)."))
      if index in range(joueur.nombreDeCartes()):
        carte = joueur.jouer(index)
        # Contrôle de la triche
        if last_card:
          (v1,c1),(v2,c2) = carte,last_card   #Unpacking
          if c1 != c2 or v1 < v2:
            raise Triche(joueur.nom)   #La partie s'arrête
        last_card = carte
      else:
        joueur.piocher()
        if last_card:    #S'il y avait une carte et que le joueur pioche
          score[player1 and 0 or 1] += 1
          last_card = None
      player1 = not player1
  except NoMoreCard:
    if score1 == score2:
      print("Égalité")
    else:
      print(joueur1 if score1 > score2 else joueur2)

# Tests (à ne pas exécuter si on importe le fichier)
if __name__ == "__main__":
  jeu("toto", "lulu")
```