Exercice 2 (6 points)

Cet exercice porte sur la structure de pile, la programmation objet et l'algorithmique.

Défi Tubes est un jeu à un joueur. Le joueur dispose de 4 tubes. Chaque tube peut contenir de 0 à 3 phases. Chaque phase possède une couleur. Il y a 3 couleurs possibles. On peut s'imaginer ces phases comme des palets de couleur dans le tube. Pour modéliser les couleurs, on utilisera les entiers 1, 2 et 3. Lorsqu'un tube contient 0 phase, on dit que le tube est vide. Lorsqu'il en a 3, on dit qu'il est plein. Lorsqu'un tube n'est pas vide, sa **dernière couleur** est la couleur de sa phase supérieure.

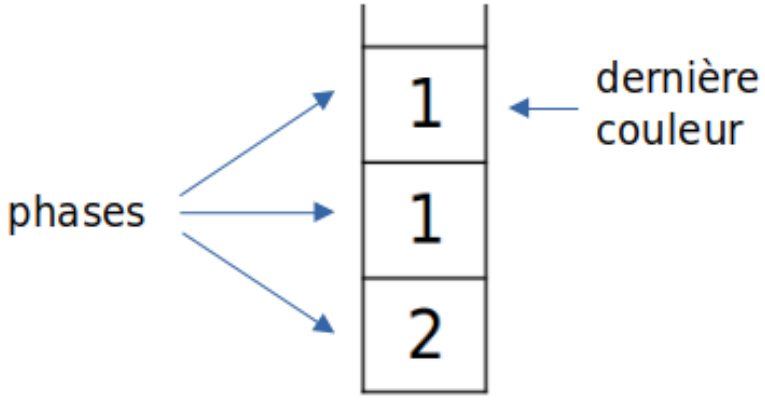

Figure 1. Exemple de tube.

Le jeu **Défi Tube** consiste à verser successivement la dernière couleur des tubes dans les autres tubes avec les contraintes suivantes :

- on ne peut rien verser dans un tube plein ;
- pour verser un **tube 1** dans un **tube 2**, il faut que la dernière couleur du **tube 1** soit la même que celle du **tube 2** ou que le **tube 2** soit vide. Dans ces deux cas, on retire la dernière couleur du **tube 1** pour qu'elle devienne la dernière couleur du **tube 2**. On réitère cela tant que la dernière couleur du **tube 1** est la même et que le **tube 2** n'est pas plein.

Le jeu se termine lorsque 3 des 4 tubes sont pleins et que leurs 3 phases sont de même couleur.

Les figures 2, 3, 4 et 5 ci-après représentent un exemple de partie du jeu **Défi Tube**.

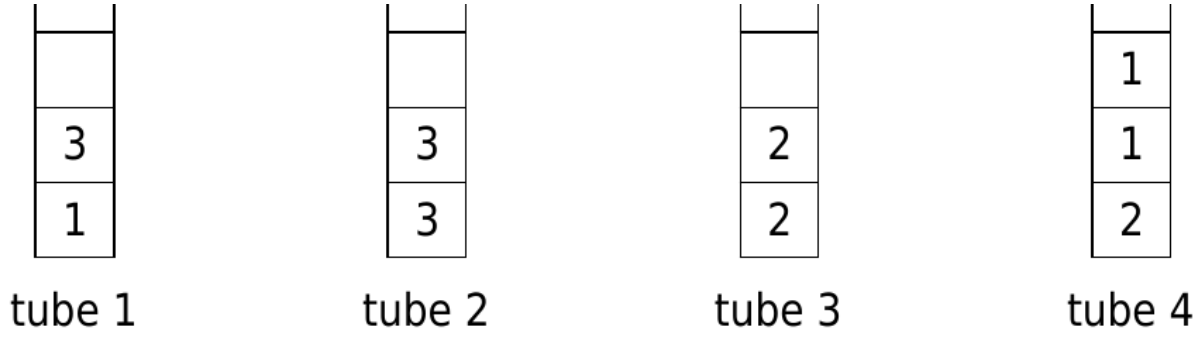

Figure 2. État initial du jeu.

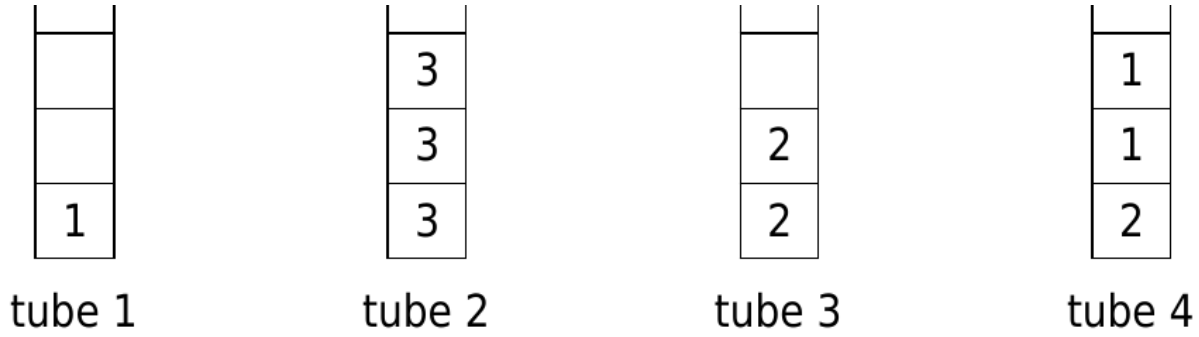

Figure 3. On a versé le tube 1 dans le tube 2.

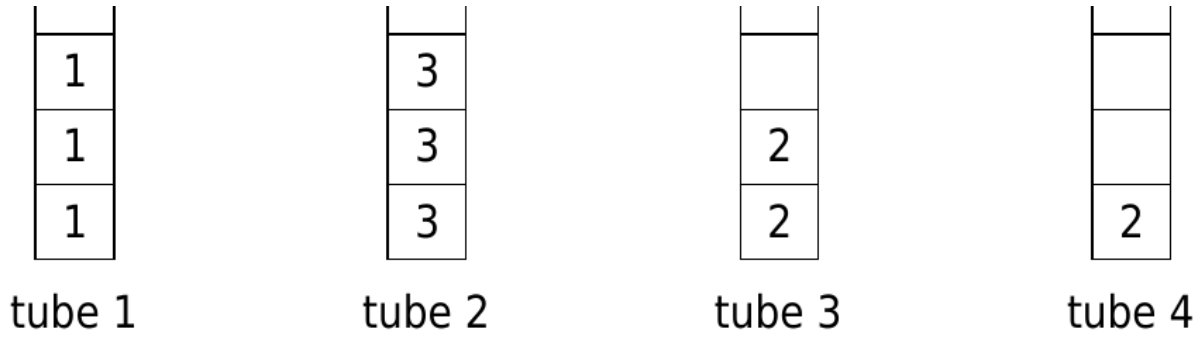

Figure 4. On a versé le tube 4 dans le tube 1.

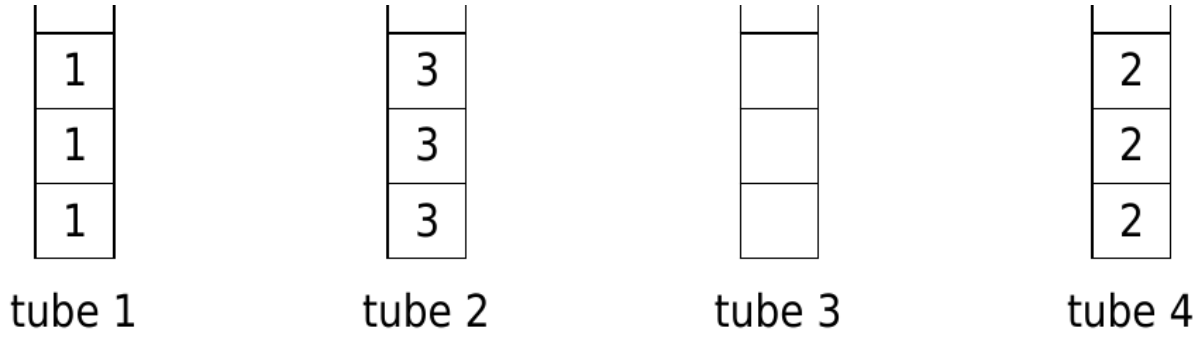

Figure 5. On a versé le tube 3 dans le tube 4.

À la figure 5, la partie est terminée.

1. Donner un exemple d'une autre séquence de versements qui aurait permis de terminer le jeu en partant de la situation de la figure 4.

Ainsi le déroulement du jeu n'est pas unique.

Partie A : Les tubes

Pour modéliser le jeu **Défi Tube**, chaque tube sera représenté par une pile finie de taille maximale 3. Les tubes sont modélisés par des objets de la classe `tube` dont le code est donné ci-dessous.

```
class tube:
    def __init__(self):
        self.taille = 0
        self.contenu = [0, 0, 0]

    def est_vide(self):
        return self.taille == 0

    def empiler(self, couleur):
        if self.taille < 3:
            self.contenu[self.taille] = couleur
            self.taille = self.taille + 1

    def depiler(self):
        if self.taille > 0:
            self.taille = self.taille - 1
            couleur = self.contenu[...]
            self.contenu[self.taille] = 0
            return ...
        else:
            return ...
```

Chaque instance de la classe `tube` a deux attributs :

- l'attribut `taille` représente le nombre d'éléments non nuls dans le tube;
- l'attribut `contenu` représente la liste (de taille 3) des éléments du tube. Lorsqu'une phase n'est pas vide, elle contiendra une couleur 1, 2, ou 3. Lorsqu'une phase est vide, sa valeur est 0.

Par exemple, le tube suivant :

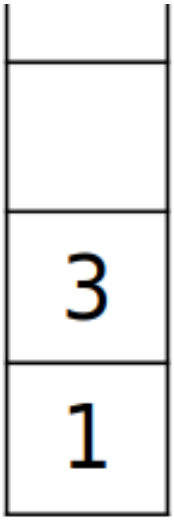

Figure 6. tube1

sera modélisé avec la classe `tube` par le code :

```
1 t = tube()
2 t.taille = 2
3 t.contenu = [1, 3, 0]
```

2. Expliquer ce qu'est la structure de pile en précisant ce que sont les méthodes `empiler` et `depiler`.

3. Expliquer les lignes 11 et 12 du code de la classe `tube`.

4. Recopier et compléter le code de la méthode `depiler` précédente. Lorsque le tube est vide, la méthode `depiler` doit renvoyer -1.

5. Écrire une méthode `est_plein` de la classe `tube`. Cette méthode renvoie `True` si le tube est plein et `False` si le tube n'est pas plein.

6. Écrire une méthode `est_homogene` de la classe `tube` qui renvoie `True` si le tube est plein et si son contenu est composé de trois fois la même couleur, et qui renvoie `False` sinon.

7. Écrire une méthode `derniere_couleur` de la classe `tube` qui renvoie le numéro de la dernière couleur du tube. Si le tube est vide, la méthode renverra la valeur `-1`.

Le code incomplet d'une méthode `verser` de la classe `tube` est donné ci-dessous :

```
1 def verser(self, other):
2     while ...
3         couleur = self.depiler()
4         other.empiler(couleur)
```

8. Recopier et compléter le code de cette méthode `verser` afin de verser l'instance `self` de la classe `tube` dans l'instance `other`. On veillera à vérifier toutes les conditions nécessaires au bon déroulement de cette opération.

Partie B : Le jeu

Pour modéliser le jeu, on appellera **état** du jeu une liste de 4 tubes. Le code suivant permet de représenter l'**état** de la figure 2.

```
 1 tube1 = tube()
 2 tube1.contenu = [1, 3, 0]
 3 tube1.taille = 2
 4 tube2 = tube()
 5 tube2.contenu = [3, 3, 0]
 6 tube2.taille = 2
 7 tube3 = tube()
 8 tube3.contenu = [2, 2, 0]
 9 tube3.taille = 2
10 tube4 = tube()
11 tube4.contenu = [1, 1, 2]
12 tube4.taille = 3
13 etat = [tube1, tube2, tube3, tube4]
```

9. En utilisant la méthode `verser` et la variable `etat` représentant la figure 2, écrire un code permettant de faire passer la variable `etat` de la représentation en figure 2 à celle de la figure 3.

10. Écrire une fonction `gagne` qui prend comme argument un état et qui renvoie `True` si la partie est terminée et `False` sinon.