DM 2 - Terminale NSI

Exercice

Cet exercice porte sur la programmation orientée objet et les structures linéaires.

Le taquin est un casse-tête qui se joue avec une grille de 9 cases : une case vide et 8 cases numérotées de 1 à 8.

Le jeu démarre en position gagnante représentée comme sur la figure 1.

	1	2
3	4	5
6	7	8

figure 1. – Position gagnante d'une grille

Les cases situées directement à gauche ou à droite ou en-dessous ou au-dessus de la case vide peuvent permuter avec elle. Afin de jouer au jeu du taquin, on mélange la grille et l'objectif est de retrouver la position gagnante. On donne ci-dessous un exemple de grille mélangée.

5	3	8
	1	2
7	6	4

figure 2. – Exemple d'une grille mélangée

Par exemple, à partir de la situation de la figure 2, seules les cases numérotées 5, 1 ou 7 peuvent permuter leur position avec celle de la case vide.

On représente la grille d'un taquin à l'aide d'un tableau (type `list` en Python) dont les éléments sont les numéros des cases de la grille de gauche à droite et de haut en bas, la case vide portant le numéro 0. Le tableau ci-dessous contient les numéros de cases de la grille de la figure 2 :

```
tab = [5, 3, 8, 0, 1, 2, 7, 6, 4]
```

1) Donner l'indice de l'élément de `tab` correspondant à la case portant le numéro 2 de la grille.

2) Dessiner la grille du taquin représentée par `tab` après l'exécution des instructions ci-dessous :

```python
tab[3] = tab[4]
tab[4] = 0
```

3) Écrire une expression booléenne qui est évaluée à `True` si le taquin représenté par le tableau `tab` est en position gagnante et `False` sinon.

On souhaite utiliser une classe `Taquin` pour modéliser le jeu du taquin. On initialise cette classe de la façon suivante :

```python
class Taquin:
    def __init__(self):
        self.tab = [0, 1, 2, 3, 4, 5, 6, 7, 8]
```

4) Écrire la méthode `est_gagnant` de la classe `Taquin` qui renvoie `True` si l'instance de `Taquin` est en position gagnante et `False` sinon.

Voici le code partiel de la méthode `indice` qui renvoie l'indice de l'élément de l'attribut `tab` qui a pour valeur `numero` :

```python
    def indice(self, numero):
        assert type(numero) == int, "numero doit être entier"
        ... , 'numero de case non valide'
        i = 0
        while ...:
            i = i + 1
        return i
```

5) Recopier et compléter les lignes 3 et 5 de la méthode `indice`.

On admet que la classe `Taquin` dispose d'une méthode `est_possible` qui prend en paramètre un entier `numero`, compris entre 1 et 8 inclus, et qui renvoie `True` s'il est possible de déplacer la case portant ce numéro et `False` sinon.

6) Recopier et compléter les lignes 2 à 6 de la méthode `jouer` ci-après qui, s'il est possible de déplacer le numéro passé en paramètre, intervertit la case portant ce numéro et la case vide.

```python
    def jouer(self, numero):
        if ...:
            i = self.indice(numero)
            j = self.indice(0)
            ... = numero
            ... = ...
```

On admet que la classe `Taquin` posséde une méthode `coups_possibles` qui renvoie un tableau dont les éléments sont les numéros des cases déplaçables. Si `t` est une instance de la classe `Taquin` qui correspond à celui de la figure 2, `t.coups_possibles()` renverra `[5, 1, 7]`.

On importe le module `random` et on rappelle que, si `tab` est du type `list`, l'instruction `random.choice(tab)` renvoie au hasard un élément de `tab`.

Pour mélanger n fois le taquin, on choisit au hasard successivement n fois un numéro parmi les coups possibles. On décide de ne pas jouer deux fois de suite le même numéro.

7) Recopier et compléter les lignes 4, 5, 6 et 9 de la méthode `melanger` donnée ci-dessous.

```python
    def melanger(self, n):
        precedent = None
        i = 0
        while ...:
            possibilites = ...
            choix = ...
            if choix != precedent:
                self.jouer(choix)
                precedent = ...
                i = i + 1
```

On souhaite ajouter à l'implémentation du jeu du taquin un *mode résolution automatique* lors duquel le jeu jouera une série de numéros afin de revenir en position gagnante. On procède ainsi :

- au départ, la grille est en position gagnante, on initialise une pile vide et l'instance de la classe `Taquin` n'est pas en mode résolution automatique ;

- si l'instance n'est pas en mode résolution automatique, on empile les coups joués pour mélanger la grille ainsi que les coups joués par un joueur ;
- si l'instance est en mode résolution automatique, tant que le taquin n'est pas gagnant, on dépile la pile et on joue le coup associé.

Par exemple, à partir d'une grille en position gagnante, on considère que les numéros 1, 4 et 5 ont été joués dans cet ordre pour mélanger la grille. Ensuite le joueur a joué le numéro 2 puis a passé le jeu en mode résolution automatique.

8) Donner, pour la grille issue de l'exemple ci-dessus, les numéros des cases joués dans l'ordre lors de la résolution automatique en précisant à chaque étape l'état de la pile.

On dispose d'une classe `Pile` dont les méthodes sont `est_vide`, `empiler` et `depiler`. La méthode `est_vide` renvoie `True` si la pile est vide et `False` sinon. La méthode `empiler` prend en paramètre une valeur et l'empile dans la pile. La méthode `depiler` dépile une valeur de la pile si elle n'est pas vide et la renvoie.

On modifie l'initialisation de la classe `Taquin` de la façon suivante :

```python
class Taquin:
    def __init__(self):
        self.tab = [0, 1, 2, 3, 4, 5, 6, 7, 8]
        self.pile = Pile()
        self.mode_resolution = False
```

On ajoute à la fin du code de la méthode `jouer` les deux lignes ci-dessous afin d'empiler les coups à chaque appel de cette méthode si l'instance du jeu n'est pas en mode résolution automatique :

```python
            if not self.mode_resolution:
                self.pile.empiler(numero)
```

Lorsque le joueur souhaite avoir la résolution automatique, il appelle la méthode `resoudre` qui :

- passe en mode résolution automatique en donnant la valeur `True` à l'attribut `mode_resolution` ;
- à chaque étape de la résolution automatique, affiche le coup joué.

9) Écrire le code de la méthode `resoudre`.

10) Expliquer pourquoi l'oubli du passage en mode résolution automatique dans la méthode `resoudre` entraînerait sa non-termination.

Jouer deux fois de suite le même numéro est inutile. Pour éviter de stocker dans la pile deux tels coups, on souhaite modifier la méthode `jouer`.

Si l'instance du jeu `Taquin` n'est pas en mode résolution automatique alors :

- si la pile du taquin n'est pas vide alors :
  - on récupère le numéro au sommet de la pile ;
  - si ce numéro et celui qu'on vient de jouer sont différents alors on les empile tous les deux dans le bon ordre, sinon on ne fait rien.
- sinon, on empile le numéro qu'on vient de jouer.

11) Proposer une modification de la méthode `jouer` à partir de la ligne 8 afin de prendre en compte cette optimisation.