Untitled

Correction exercice 1

Sujet 04 - Taquin

Questions 1 à 3

1. Dans `tab = [5, 3, 8, 0, 1, 2, 7, 6, 4]`, la case portant le numéro 2 est à l'indice `5`.
2. Après les instructions données, on obtient `tab = [5, 3, 8, 1, 0, 2, 7, 6, 4]`, soit la grille :

```
5 3 8
1   2
7 6 4
```

3. Expression booléenne :

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

Question 4

```
def est_gagnant(self):
    return self.tab == [0, 1, 2, 3, 4, 5, 6, 7, 8]
```

Question 5

```
def indice(self, numero):
    assert type(numero) == int, "numero doit être entier"
    assert 0 <= numero <= 8, 'numero de case non valide'
    i = 0
    while self.tab[i] != numero:
        i = i + 1
    return i
```

Question 6

```
def jouer(self, numero):
    if self.est_possible(numero):
        i = self.indice(numero)
        j = self.indice(0)
        self.tab[j] = numero
        self.tab[i] = 0
```

Question 7

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

Question 8

Au départ, la pile est vide.

Après les coups de mélange `1`, `4`, `5`, puis le coup joué `2`, la pile contient :

```
[1, 4, 5, 2]
```

La résolution automatique dépile donc les coups en sens inverse. Les coups joués sont :

```
2, 5, 4, 1
```

États successifs de la pile :

```
[1, 4, 5, 2]
[1, 4, 5]
[1, 4]
[1]
[]
```

Question 9

```
def resoudre(self):
    self.mode_resolution = True
    while not self.est_gagnant() and not self.pile.est_vide():
        numero = self.pile.depiler()
        print(numero)
        self.jouer(numero)
```

Question 10

Si on oublie de passer `mode_resolution` à `True`, chaque appel à `jouer` pendant la résolution empile à nouveau le coup joué. La pile ne se vide donc pas correctement, ce qui peut provoquer une boucle sans fin.

Question 11

```
if not self.mode_resolution:
    if not self.pile.est_vide():
        sommet = self.pile.depiler()
        if sommet != numero:
            self.pile.empiler(sommet)
            self.pile.empiler(numero)
    else:
        self.pile.empiler(numero)
```

Programme complet possible

```
import random

class Pile:
    def __init__(self):
        self.valeurs = []

    def est_vide(self):
        return self.valeurs == []

    def empiler(self, valeur):
        self.valeurs.append(valeur)

    def depiler(self):
        if not self.est_vide():
            return self.valeurs.pop()

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

    def est_gagnant(self):
        return self.tab == [0, 1, 2, 3, 4, 5, 6, 7, 8]

    def indice(self, numero):
        assert type(numero) == int, "numero doit être entier"
        assert 0 <= numero <= 8, 'numero de case non valide'
        i = 0
        while self.tab[i] != numero:
```

```
            i = i + 1
        return i

    def est_possible(self, numero):
        i = self.indice(numero)
        j = self.indice(0)
        ligne_i = i // 3
        colonne_i = i % 3
        ligne_j = j // 3
        colonne_j = j % 3
        return abs(ligne_i - ligne_j) + abs(colonne_i - colonne_j) == 1

    def coups_possibles(self):
        coups = []
        for numero in range(1, 9):
            if self.est_possible(numero):
                coups.append(numero)
        return coups

    def jouer(self, numero):
        if self.est_possible(numero):
            i = self.indice(numero)
            j = self.indice(0)
            self.tab[j] = numero
            self.tab[i] = 0
            if not self.mode_resolution:
                if not self.pile.est_vide():
                    sommet = self.pile.depiler()
                    if sommet != numero:
                        self.pile.empiler(sommet)
                        self.pile.empiler(numero)
                else:
                    self.pile.empiler(numero)

    def melanger(self, n):
        precedent = None
        i = 0
        while i < n:
            possibilites = self.coups_possibles()
            choix = random.choice(possibilites)
            if choix != precedent:
                self.jouer(choix)
            precedent = choix
            i = i + 1

    def resoudre(self):
        self.mode_resolution = True
        while not self.est_gagnant() and not self.pile.est_vide():
            numero = self.pile.depiler()
            print(numero)
            self.jouer(numero)
```