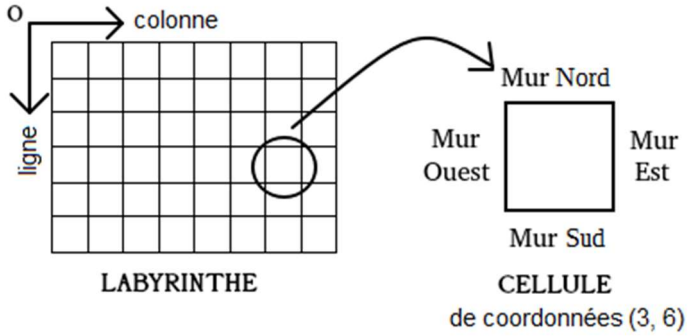
0.5 Diviser pour régner

NSI terminale - JB Duthoit

EXERCICE 5 (4 points)

Cet exercice aborde la programmation objet et la méthode diviser pour régner.



Un labyrinthe est composé de cellules possédant chacune quatre murs (voir ci-dessus). La cellule en haut à gauche du labyrinthe est de coordonnées (0, 0). On définit la classe `Cellule` ci-dessous. Le constructeur possède un attribut `murs` de type `dict` dont les clés sont `'N'`, `'E'`, `'S'` et `'O'` et dont les valeurs sont des booléens (`True` si le mur est présent et `False` sinon).

```
class Cellule:
    def __init__(self, murNord, murEst, murSud, murOuest):
        self.murs={'N':murNord,'E':murEst,
                   'S':murSud,'O':murOuest}
```

1. Recopier et compléter sur la copie l'instruction Python suivante permettant de créer une instance `cellule` de la classe `Cellule` possédant tous ses murs sauf le mur Est.

```
cellule = Cellule(...)
```

2. Le constructeur de la classe `Labyrinthe` ci-dessous possède un seul attribut grille. La méthode `construire_grille` permet de construire un tableau à deux dimensions `hauteur` et `longueur` contenant des cellules possédant chacune ses quatre murs. Recopier et compléter sur la copie les lignes 6 à 10 de la classe `Labyrinthe`.

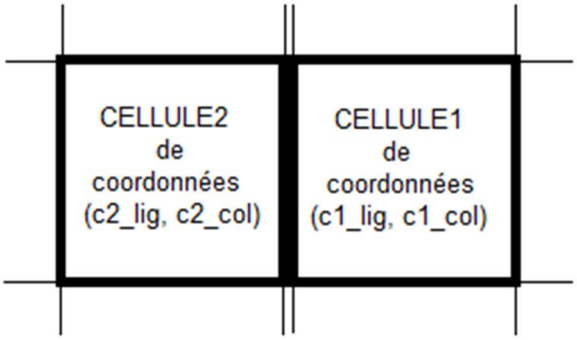
```
1  class Labyrinthe:
2      def __init__(self, hauteur, longueur):
3          self.grille=self.construire_grille(hauteur, longueur)
4      def construire_grille(self, hauteur, longueur):
5          grille = []
6          for i in range(...):
7              ligne = []
8              for j in range(...):
9                  cellule = ...
10                 ligne.append(...)
11             grille.append(ligne)
12         return grille
```

Pour générer un labyrinthe, on munit la classe `Labyrinthe` d'une méthode `creer_passage` permettant de supprimer des murs entre deux cellules ayant un côté commun afin de créer un passage. Cette méthode prend en paramètres les coordonnées `c1_lig`, `c1_col` d'une cellule notée `cellule1` et les coordonnées `c2_lig`, `c2_col` d'une cellule notée `cellule2` et crée un passage entre `cellule1` et `cellule2`.

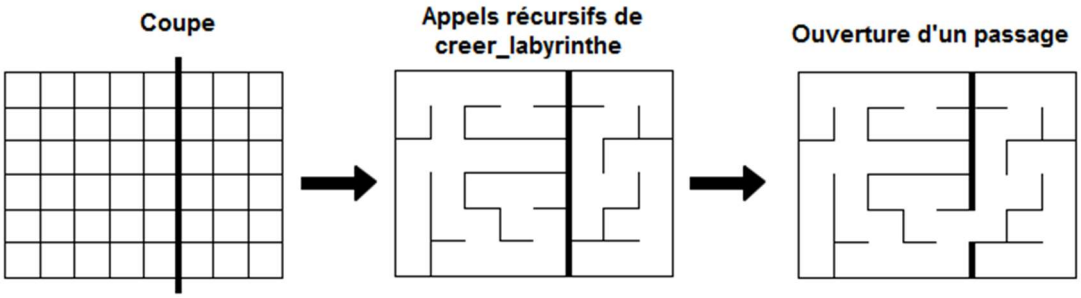
```
13      def creer_passage(self, c1_lig, c1_col, c2_lig, c2_col):
14          cellule1 = self.grille[c1_lig][c1_col]
15          cellule2 = self.grille[c2_lig][c2_col]
16          # cellule2 au Nord de cellule1
17          if c1_lig - c2_lig == 1 and c1_col == c2_col:
18              cellule1.murs['N'] = False
19              ....
20          # cellule2 à l'Ouest de cellule1
21          elif ....
22            ....
23            ....
```

3. La ligne 18 permet de supprimer le mur Nord de `cellule1`. Un mur de `cellule2` doit aussi être supprimé pour libérer un passage entre `cellule1` et `cellule2`. Écrire l'instruction Python que l'on doit ajouter à la ligne 19.

4. Recopier et compléter sur la copie le code Python des lignes 21 à 23 qui permettent le traitement du cas où `cellule2` est à l'Ouest de `cellule1` :



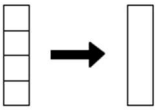
Pour créer un labyrinthe, on utilise la méthode diviser pour régner en appliquant récursivement l'algorithme `creer_labyrinthe` sur des sous-grilles obtenues en coupant la grille en deux puis en reliant les deux sous-labyrinthes en créant un passage entre eux.



La méthode `creer_labyrinthe` permet, à partir d'une grille, de créer un labyrinthe de hauteur `haut` et de longueur `long` dont la cellule en haut à gauche est de coordonnées (`ligne`, `colonne`).
Le cas de base correspond à la situation où la grille est de hauteur 1 ou de largeur 1. Il suffit alors de supprimer tous les murs intérieurs de la grille.



Exemple avec
haut = 1 et long = 4



Exemple avec
haut = 4 et long = 1

5. Recopier et compléter sur la copie les lignes 25 à 30 de la méthode `creer_labyrinthe` traitant le cas de base.

```
24      def creer_labyrinthe(self, ligne, colonne, haut, long):
25          if haut == 1 : # Cas de base
26                for k in range(...):
27                    self.creer_passage(ligne, k, ligne, k+1)
28          elif long == 1: # Cas de base
29                for k in range(...):
30                    self.creer_passage(...)
31          else: # Appels récursifs
32             # Code non étudié (Ne pas compléter)
```

6. Dans cette question, on considère une grille de hauteur `haut` = 4 et de longueur `long` = 8 dont chaque cellule possède tous ses murs.
On fixe les deux contraintes supplémentaires suivantes sur la méthode `creer_labyrinthe` :

- Si `haut` ≥ `long`, on coupe horizontalement la grille en deux sous-labyrinthes de même dimension.
- Si `haut` < `long`, on coupe verticalement la grille en deux sous-labyrinthes de même dimension.

L'ouverture du passage entre les deux sous-labyrinthes se fait le plus au Nord pour une coupe verticale et le plus à l'Ouest pour une coupe horizontale.

Dessiner le labyrinthe obtenu suite à l'exécution complète de l'algorithme `creer_labyrinthe` sur cette grille.