MiniProjet 4 : Sortir d'un labyrinthe

Voici ci-contre un labyrinthe (les « X » sont des murs), avec une case d'entrée notée « E » et une sortie notée « S » dont on cherche à sortir :

Ce labyrinthe est stocké dans un fichier ***dedale.txt***

```
XXXXX
X X S
X   X
XEX X
X X X
XXXXX
```

Pour trouver la sortie, il faut réaliser un programme récursif dont voici le principe :

- Il parcourt toutes les cases vides à partir de la position courante en les marquant avec un « **.** », afin d'éviter de tourner en rond.
- Il essaie successivement les directions droite, gauche, bas puis haut, de façon récursive.

Ici, en suivant ce principe, on aurait :

Labyrinthe :

```
XXXXX
X X S
X   X
XEX X
X X X
XXXXX
```

Solution :

```
XXXXX
X X.S
X...X
X.X X
X X X
XXXXX
```

Objectif :

MVP (ou Minimum Valuable Product)

Créer le programme qui permet le parcours d'un labyrinthe de ce type.

Il faudra fournir un document écrit, en plus du fichier python :

- Un schéma représentant le « trajet » du programme dans le labyrinthe type fourni (essai à droite, à gauche, etc.)
- Pour les fonctions récursives, on explicitera les conditions d'arrêt, la manière dont le problème est réduit, la place de l'appel récursif (voir cours CH6)
- On précisera l'évolution de la pile de variables en fonction des différents appels récursifs, empiler / dépiler (voir documents plus loin)

En plus

Possibilité d'introduire une interface graphique (tkinter ou pygame)

EVALUATION :

ANA	Décrire et Spécifier les caractéristiques d'un processus, les données d'un problème, ou celles manipulées par un algorithme ou une fonction
	Montrer des capacités d'abstraction et de généralisation dans l'analyse d'un problème
COM	Communiquer à l'écrit en utilisant un langage rigoureux et des outils pertinents

Fonctionnalités demandées :

Résolution du labyrinthe	/5	*Description de(s) fonction(s) récursive(s)*	/2
Schéma du trajet	/2	*Description du fonctionnement de la pile d'instruction*	/3

Fonctionnalités plus : (+1pt par fct)

Utilisation des classes		*Interface graphique*	
...			

Code :

Lisibilité du code	/2
Variables explicites	/2
Pas de répétitions, utilisation de boucles et de fonctions	/2
Commentaires pertinents	/2
Documentation de fonctions	/2

DOCUMENTS :

La récursivité permet de résoudre certains problèmes d'une manière très rapide, alors que si on devait les résoudre de manière itérative, il nous faudrait beaucoup plus de temps et de structures de données intermédiaires.

La récursivité utilise toujours la pile du programme en cours. On appelle **pile** une zone mémoire réservée à chaque programme. Son rôle est de stocker les variables locales et les paramètres d'une procédure.
Supposons que nous sommes dans une procédure ***proc1*** dans laquelle nous avons des variables locales.
Faisons ensuite appel à une procédure ***proc2*** ; comme le microprocesseur va commencer à exécuter ***proc2*** mais qu'ensuite il reviendra continuer l'exécution de ***proc1*** , il faut bien stocker quelque part les variables de la procédure en cours ***proc1*** ; c'est le rôle de la pile. Tout ceci est géré de façon transparente pour l'utilisateur.

Dans une procédure récursive, toutes les variables locales sont stockées dans la pile, et empilées autant de fois qu'il y a d'appels récursifs.
Donc la pile se remplit progressivement, et si on ne fait pas attention on peut arriver à un débordement de pile (*stack overflow* en anglais). Ensuite, les variables sont dépilées. Dans notre exemple du labyrinthe, les variables seront les coordonnées des cases du labyrinthe. Chaque fois qu'une case sera marquée, elle sera mise dans la pile. Quand la marque sera effacée, la case sera dépilée. C'est l'idée-clé pour bien comprendre la récursivité.

Pour voir en détails ce que fait le programme, mettons des coordonnées aux cases et affichons
toutes les cases parcourues, dans l'ordre :

```
  01234
0XXXXX
1X  X S      S se trouve aux coordonnées (1 ; 4)
2X    X
3XEX  X      E se trouve aux coordonnées (3 ; 1)
4X  X X
5XXXXX
```

```
3 1          case de départ ; on marque la case avec un « . »
3 2          on essaie à droite : c'est un mur
3 0          on essaie à gauche : c'est un mur
```