TP 6: Le fil d'Ariane

1 Résumé

Vous connaissez sans doute le mythe grec du Thésée et Ariane. Thésée, un héros Athénien, est entrée dans le labyrinthe où se trouvait le légendaire Minotaure, un monstre au corps d'un homme et à tête d'un taureau. Le brave Thésée a tué l'horrible monstre, mais il n'a réussi à trouver la sortie du labyrinthe que grâce à l'aide de la princesse Ariane, qui lui a fourni un fil qui lui a permis de retrouver son chemin.

Le but de cet exercice est d'écrire un programme en Haskell qui, étant donné la description d'un labyrinthe va aider Thésée à trouver le chemin vers le Minotaure[1] Votre programme devra être capable de :

1. Lire la description d'un labyrinthe (dans un format précisé ci-dessous) à partir d'un fichier.
2. Afficher le labyrinthe avec le plus court chemin vers le Minotaure clairement indiqué, si un tel chemin existe. Sinon, le programme doit afficher un message d'échec.

La manière la plus simple de comprendre ce qui est demandé est avec un exemple. Vous trouverez sur la page web les fichiers `lab1.txt-lab6.txt`, qui contiennent les descriptions de six labyrinthes. Une exécution du programme pourrait être la suivante :

```
> runhaskell mino.hs
File where labyrinth is stored:
lab2.txt -- NB Ici l'utilisateur donne le nom du fichier -- NB Ici l'utilisateur do
Labyrinth read:
* * * * * * * * * *
* o o o * M * o * *
* o * o * o o o o *
* o * o * o * * o *
* o * o * o * o o *
* o * o * o * * o *
* T * o o o * o o *
* * * * * * * * * *

Solution:
* * * * * * * * * *
* x x x * M * o * *
* x * x * x o o o *
* x * x * x * * o *
* x * x * x * o o *
* x * x * x * * o *
* T * x x x * o o *
* * * * * * * * * *

> runhaskell mino.hs
File where labyrinth is stored:
```

[1] Thésée a le fil d'Ariane pour l'aider à trouver la sortie après avoir tué le monstre.

```
lab4.txt
Labyrinth read:
* * * * * * * * * *
* o o o * o * o * *
* o * o * o o o o *
* o * o o o * * o *
* o * o * o * M o *
* o * o * o * * o *
* T * o o o * o o *
* * * * * * * * * *

Solution:
* * * * * * * * * *
* x x x * o * o * *
* x * x * x x x x *
* x * x x x * * x *
* x * o * o * M x *
* x * o * o * * o *
* T * o o o * o o *
* * * * * * * * * *
```

Comme vous pouvez le voir si vous examinez les fichiers donnés, le format qui spécifie la structure d'un labyrinthe est assez simple :

- Le labyrinthe est considéré comme un rectangle. Les deux premiers lignes du fichier précisent ses dimensions (nombre de lignes et colonnes).
- Par la suite, le fichier contient la description de chaque point du labyrinthe, ligne par ligne. On utilise les caractères `*`, `o`, `T`, `M` pour indiquer un mur (point où Thésée ne peut pas aller), un espace vide (point où Thésée peut aller), et les positions initiales de Thésée et le Minotaure respectivement. Ces caractères sont séparés par des espaces (un ou plusieurs).
- Dans la sortie du programme, on utilise le caractère `x` pour signifier le plus court chemin qui permet à Thésée d'atteindre le Minotaure[2]
- Vous pouvez supposez que les fichiers qui vous seront donnés ont le bon format, et qu'ils contiennent un seul Thésée et un seul Minotaure.
- Vous ne pouvez pas supposer qu'il existe forcément un chemin vers le Minotaure.
- Vous ne pouvez pas supposer que les caractères de la première et dernière ligne et de la première et dernière colonnes sont des `*` (donc il faut faire attention que Thésée ne sort pas du labyrinthe par erreur).

Pour simplifier votre tâche, on vous propose de découper cet exercice en deux parties quasi–indépendantes : la partie qui lit le fichier et s'occupe du bon affichage du labyrinthe ; et la partie qui analyse le labyrinthe pour trouver le bon chemin.

2 Entrée/Sortie et Représentation Abstraite

On vous demande d'utiliser les définitions suivantes :

```
data Tile = W | EU | EV | T | M
type Game = ([[Tile]], (Int,Int))
```

[2] On suppose que le Minotaure reste immobile.

Un objet de type `Tile` représente un point dans le labyrinthe. Ce point peut être un mur (`W`), un espace vide qui ne fait pas partie du chemin (`EU`), un espace vide qui fait partie du chemin (`EV`), Thésée (`T`) ou le Minotaure (`M`). Donc, dans le format décrite ci-dessus, les valeurs `W, EU, EV, T, M` correspondent aux caractères `*, o, x, T, M`.

Le type `Game` est une représentation abstraite de l'état du jeu : il contient une description du labyrinthe et les coordonnées actuelles de Thésée (en prenant en compte que `T` signifie la position initiale de Thésée).

On vous demande de programmer :

1. Une fonction `readGame :: Handle -> IO Game` qui prend comme argument le descriptif d'un fichier (qui a déjà été ouvert), lit et retourne la description du labyrinthe.

2. Une fonction `showGame :: Game -> String` qui retourne une chaîne de caractères qui représente l'état actuel du jeu (utilisez un format comme dans l'exemple ci-dessus).

3. Une fonction `main` qui fait l'interaction avec l'utilisateur (demande le nom du fichier, affiche le labyrinthe, etc.)

3 Trouver le Minotaure

Pour la deuxième partie, vous devez programmer la fonction qui résout le problème et trouve le plus court chemin vers le Minotaure. Vous pouvez traiter cette partie même si vous n'avez pas encore réussi avec `readGame`. Dans ce cas, il vous faudra entrer à la main la description de quelques labyrinthes dans votre programme. Par exemple, si vous écrivez

```
lab1 = [ [W,W,W,W,W], [W,EU,EU,EU,W], [W,EU,W,EU,W],
           [W,EU,W,EU,W], [W,T,W,M,W], [W,W,W,W,W] ]
```

cela correspond au labyrinthe du fichier `lab1.txt`[3].

Quelle que soit la manière que vous ayez obtenu une représentation de l'état initial du jeu (donc un objet de type `Game`), on vous demande de l'utiliser pour décider si un chemin existe (et de le trouver). Programmer les fonctions suivantes :

1. `step :: Game -> [Game]`, qui étant donné l'état actuel, produit une liste avec tous les états suivants possibles. Les règles ici sont : (i) Thésée peut faire un mouvement d'un point, vertical ou horizontal (mais pas diagonal) (ii) Thésée ne peut pas se trouver sur un point de type `W` (iii) tous les points déjà visités ont le type `EV` (iv) Thésée ne se rend jamais une deuxième fois dans un point de type `EV` (on cherche le plus court chemin, donc ce n'est pas la peine de répéter de mouvements) (v) si Thésée atteint le Minotaure le jeu est terminé.

2. Une fonction `solveink :: Int -> Game -> [Game]` qui prend comme paramètre un entier `k` et l'état actuel et retourne une liste qui représente tous les chemins qui atteignent le Minotaure avec $\leq$ `k` mouvements de Thésée.

3. Une fonction `solve :: Game -> Maybe Game` qui prend comme paramètre l'état actuel et retourne une solution qui représente le plus court chemin, ou `Nothing` si aucune solution n'existe. (Si plusieurs solutions optimales existent, vous pouvez retourner n'importe laquelle parmi elles).

Vous êtes conseillés (mais ce n'est pas obligatoire de suivre cette conseil) de profiter du fait que les listes sont de `Monads`. Par conséquent, c'est possible d'enchaîner les applications de `step` en utilisant l'opérateur `>>=`.

[3] Évidemment, c'est assez pénible d'entrer les données de cette façon. C'est pour ça que la partie entrée/sortie est proposée en première. On vous propose cette solution de dernier secours au cas où vous ne pouvez pas faire l'entrée de données correctement.

4 Sortie plus lisible (Partie Facultative)

NB: Cette partie est complètement facultative et ne sera pas notée. Faites la pour vous amuser !

Le format d'affichage qu'on a utilisé devient plus difficile à lire pour de grands labyrinthes. Pour le rendre plus lisible, si vous utilisez un terminal compatible avec les codes ANSI (c'est le cas pour tous les systèmes Linux modernes, et probablement pour la plupart d'autres systèmes que vous êtes susceptibles d'utiliser) vous pouvez demander au terminal de colorier le chemin. Pour le faire, vous pouvez utiliser la combinaison de caractères spéciale `\ESC[47m`, qui signifie que le texte qui suit sera colorié, et `\ESC[m` qui signifie que le texte qui suit redevient normal. Donc, afficher au terminal la chaîne `"\ESC[47mx"` affiche un `x` colorié (et rend le chemin plus facile à voir). Avec ce type d'affichage vous pouvez obtenir une sortie comme la suivante (pour `lab5.txt`).

Si cette partie vous intéresse et vous voulez allez plus loin (ou vous avez du mal à le faire marcher) on vous invite de jeter un coup d'œil ici : `http://hackage.haskell.org/package/ansi-terminal-0.5.0/docs/System-Console-ANSI.html`