Devoir Programmation Fonctionnelle

Un robot se déplace dans un environnement planaire représenté par une grille de NxN cases. Les cases sont représentées par leurs coordonnées. On souhaite développer un algorithme *A** qui permet au robot d'aller d'une case initiale (haut à gauche de la grille) à une case finale (en bas à droite de la grille). Le robot peut se déplacer à partir d'une case vers le haut, vers le bas, à gauche ou à droite. Parfois ces déplacements ne sont pas possibles lorsque le robot se trouve au bord de son environnement ou lorsque la case contient un obstacle. Un exemple d'une grille est donnée ci-dessous mentionnant la case de départ, la case d'arrivée et des cases obstacles mentionnées par des -XXXXX-

départ				
	XXXX		XXXX	
		XXXX		
		XXXXXX		**destination**

On souhaite traiter ce problème de deux manières différentes. Une approche orientée recherche heuristique et une approche orientée planification.

Partie 1: Recherche heuristique

Pour écrire l'algorithme A*, on a besoin de définir l'état initial, les successeurs d'un état et une fonction heuristique *h*. Pour cela, le problème est représenté par un espace d'états, avec un état initial et final puis la liste des cases où figurent des obstacles.

1. On vous demande de définir les concepts suivants pour l'exemple du robot.

```
data Etat =
etatInitial =
etatfinal =
obstacles =
estTerminal :: Etat -> Bool
estobstacle :: Etat -> Bool
```

2. On souhaite définir la fonction qui calcule les états successeurs d'un état donné (attention aux obstacles et aux bords de la grille). Ecrire la fonction successeur.

```
successeurs :: Etat -> [Etat]
```

3. Pour l'algorithme A*, on définit la fonction **f = g + h** telle que *la fonction* **g** est le nombre d'états nécessaires pour arriver dans l'état courant et *la fonction* **h** estime le coût pour atteindre l'état final à partir de l'état courant. Pour cela, nous avons retenu deux distances **h1** distance de Manhattan et **h2** distance euclidienne (partie entière de $\sqrt{.x^2 + y^2}$).

```
h  Etat -> Integer
```

4. On souhaite écrire une nouvelle fonction heuristique **h3** qui tient compte des états obstacles. Pour cela, on suppose que la fonction successeur donne tous les états successeurs possibles sans tenir compte des obstacles (les bords sont pris en compte par la fonction successeur). Proposer une fonction heuristique **h3** qui tient compte des états obstacles et qui permet de les éviter.

5. Comment étendre les fonctions développées dans les questions précédentes pour gérer plusieurs états finaux (tous les états finaux doivent être visités).

Partie 2 génération de plans

On souhaite traiter ce problème d'un point de vue génération de plans. Formellement, la problématique générale est celle d'un espace de recherche. On considère un ensemble d'états possibles, un état initial et des états finaux et des actions qui permettent des transitions d'un état à un autre. L'objectif est de déterminer la séquence d'actions qui permet d'aller de l'état initial à l'état final. Les actions que nous considérons dans notre cas sont G, D, H, B pour aller respectivement à gauche, à droite, en haut et en bas. Il faut aussi, éviter de passer par une case déjà visitée.

1. La formalisation est comme suit

```
data Etat =
data Action =
etatInitial =
etatfinal =
transition : Etat -> Action -> Etat
deja-Visite : Etat -> [Etat]->Bool
estTerminal :  Etat -> Bool
estobstacle  Etat -> Bool
```

2. La génération de plan part de l'état initial avec une séquence vide. La fonction qui génère le plan est comme suit

   genPlan : Etat -> [Etat] -> [Action]

Initialement, genPlan (etatinitial, [],[]). transition (e, a) = e' signifie en appliquant à l'état e l'action a, on transite à l'état e'. Pour un état, on peut avoir plusieurs actions qui soient applicables.

3. Comment utiliser les mêmes fonctions de la partie 1 pour sélectionner à chaque fois la meilleure action.