Fiche de TP no. 3

Exercice 1. Implémentez, en Haskell, le jeu du *nim*. Les règles de ce jeux sont comme suit :

- L'échiquier contient 5 lignes d'étoiles, chaque ligne contenant un nombre d'étoiles.
- Le jeu demarre de cette position :

```
1: * * * * *
2: * * * *
3: * * *
4: * *
5: *
```

- Deux joueurs enlèvent – alternativement – une ou plusieurs étoiles d'une seule ligne.
- Le gagnant est le joueur qui enlève la dernière(s) étoile(s) de l'échiquier.

Conseil : Représentez l'échiquier comme une liste de 5 entier qui représente les étoiles encore à enlever. Par exemple, la position initiale est représentée par `[5,4,3,2,1]`.

Exercice 2. On souhaite tester ce script :

```
type Pixel = Int
type Ligne = [Pixel]
type Image = [Ligne]
type Effet = (Pixel,[Pixel]) -> Pixel
type Point = (Int,Int)

taille_x,taille_y :: Image -> Int
taille_x = length . head
taille_y = length

sousliste :: Int -> Int -> [a] -> [a]
sousliste i j xs = drop (i-1) (take j xs)

voisinage :: Image -> Point -> (Pixel,[Pixel])
voisinage img (x,y) = (p,ps)
    where
      p = (img !! (y-1)) !! (x-1)
      ps = concat (map (sousliste (x-1) (x+1)) (sousliste (y-1) (y+1) img))

appliquerEffet :: Effet -> Image -> Image
appliquerEffet effet image =
    let
        (xmax,ymax) = (taille_x image,taille_y image)
        pts = [ [ (x,y) | x <- [1..xmax] ]  |  y <- [1..ymax] ]
    in
      map (map (\p -> effet (voisinage image p))) pts
```

A ce fin, copiez le script du document pdf ou du repertoire `code` sur le site web du cours

http://pageperso.lif.univ-mrs.fr/~luigi.santocanale/teaching/PF/code/

vers un éditeur de texte, sauvegardez le.

1. Ajoutez au script une fonction `showImage` qui transforme une image dans une chaîne de caractères prête à être imprimée. Par exemple, on pourrait avoir :

```
*Main> putStr (showImage [[0,1,0],[1,0,1],[0,1,0]])
```

```
 *
* *
 *
*Main>
```

2. Écrivez un effet `smooth :: Effet` qui noircit (resp. blanchit) un pixel si plus que les deux tiers des pixels voisins sont noirs (resp. blancs).
3. Testez cet effet sur les figures suivantes :

```
image1,image2 :: Image
image1 = [
 [1,1,1,0],
 [0,0,1,1],
 [1,0,1,0],
 [0,0,0,1]]
image2 = [
 [0,1,1,0,0,1,0,0,0,1],
 [1,0,0,1,0,1,1,0,1,1],
 [1,0,0,1,0,1,0,1,0,1],
 [0,1,1,0,0,1,0,0,0,1]]
```