Examen Réparti 1 - PAF 2021: *Lemmings*

PAF 2021 - Master STL S2

17 Mars 2021

Préliminaires

Aucun barême n'est donné, les copies seront évaluées sur les points suivants:

- qualité de la programmation fonctionnelle,
- maitrises des spécificités d'Haskell vues en cours et en TD,
- pertinence de la modélisation : respect des principes vus au TD3, écriture systématique de propriétés (invariants, post-conditions), preuves et tests.

Il est conseillé de traiter les parties 1, 2 et 3 dans l'ordre, éventuellement en sautant des questions marquées d'un `(+)` (qui sont moins nécessaires pour traiter les questions suivantes), puis d'imaginer une modélisation répondant à un point de la partie 4. Les trois questions marquées d'un `(I)` attendent des réponses particulièrement conséquentes et détaillées, et constituent le coeur du partiel.

Le langage du partiel est le *Haskell* utilisé en cours et en TD. Une syntaxe approximative n'est pas pénalisée.

Le sujet comporte une annexe décrivant graphiquement le comportement de certains Lemmings, afin d'illustrer les explications données dans la partie 3.

Description du jeu

L'objectif du partiel, et du projet associé, et la modélisation et l'implémentation sûre d'une version (discrète, c'est-à-dire "case par case") du jeu *Lemmings* développé par *DMA Design* et édité par *Psygnosis* en 1991.

Lemmings est un jeu de réflexion en temps réel se déroulant dans des niveaux en deux dimensions vus de côté. Ces niveaux sont composés de cases pleines (terre ou métal) ou vides.

A intervalle régulier, des personnages (appelés **lemmings**) apparaissent d'un point du niveau (l'entrée). Le joueur n'a pas de contrôle direct sur les lemmings, qui marchent dans la même direction jusqu'à rencontrer un obstacle (ils se mettent alors à marcher dans l'autre sens) ou une falaise/un trou (ils tombent et, si la chute n'est pas mortelle, continuent à marcher dans la même direction une fois arrivés sur le sol).

Les niveaux contiennent tous une case de sortie. Si un lemming marche sur la sortie, il disparait (et est considéré comme "sauvé"). Le but du jeu est de sauver le plus de lemmings possible.

Les actions du joueurs prennent une unique forme: le joueur choisit (en temps réel) un lemming et lui assigne une **classe**, qui modifie son comportement: par exemple en le faisant creuser à travers des obstacles ou en lui faisant construire un pont au dessus d'un trou. Le jeu se termine quand un nombre fixé à l'avance de lemmings a été créé et quand il n'y a plus de lemmings dans le niveau, le score du joueur correspond au pourcentage de lemmings sauvés par rapport aux lemmings apparus.

Partie 1 : Coordonnées

Le jeu se déroulant dans des niveaux "vus de côté", une première étape dans la modélisation et l'implémentation consiste en l'écriture de fonctions permettant la représentation et les déplacement dans une telle grille.

Les cases d'un niveau seront identifiées par des coordonnées x et y (abscisses et ordonnées) telle que la case **tout en bas à gauche** d'un niveau aura comme coordonnées $(0, 0)$.

On définit le type suivant pour représenter ces coordonnées

```
data Coord = C Int Int
    deriving (Show, Eq)
```

On définit aussi un type pour les *déplacement élémentaires* correspondant aux 8 directions standard (gauche, droite, bas, haut et les diagonales gauche-haut, ...) et au neutre (ne pas bouger).

```
data Deplacement = N | G | D | H | B | GH | GB | DH | DB
    deriving (Eq, Show)
```

1.1 Ecrire la fonction `bougeCoord :: Deplacement -> Coord -> Coord` qui prend en entrée les coordonnées d'une case et renvoie les coordonnées obtenues en se déplaçant depuis cette case selon un des 9 déplacements élémentaires.

1.2 `(+)` Pour traiter correctement des `Map` dont les clefs sont des coordonnées (par exemple à la question **2.3**), on a besoin de parcourir un ensemble de coordonnées selon un ordre particulier: on veut commencer par la case **la plus en haut, et la plus à gauche** (dans cet ordre), puis se diriger vers la droite, et quand ca n'est plus possible, descendre à une ligne suivante.

Instancier `Ord` avec `Coord` pour que les ensembles de coordonnées soient parcourus dans cet ordre.

1.3 Ecrire une propriété `prop_bougeCoordGaucheDroite` qui vérifie que bouger à gauche puis à droite revient à ne pas bouger et une propriété `prop_bougeCoordGaucheHaut` qui vérifie que bouger à gauche puis en haut revient à bouger selon la diagonale gauche-haut.

1.4 `(+)` Prouver `prop_bougeCoordGaucheDroite c == True` pour tout `c`.

1.5 `(+)` Pour travailler facilement avec différents type qui contiennent des coordonnées, on définit une classe de types `Placable`. Pour que `a` instantie `Placable`, il faut trois fonctions : `coordP` qui donne les coordonnées d'un `a`, `bougeP` qui déplace un `a` a l'aide d'un déplacement élémentaire et `deplaceP` qui déplace un `a` directement dans une case donnée (par ses coordonnées).

On imagine une unique loi pour `Placable`: déplacer élémentairement avec `bougeP` un élément d'un type (instantiant `Placable` et `Eq`) vers la gauche revient à déplacer cet élément avec `deplaceP` sur la case à sa gauche.

Donner la définition de la classe `Placable` et écrire une propriété qui correspond à la loi.

Partie 2 : Niveau

Un niveau représente le *décor* (le terrain) d'une partie. Il se présente sous la forme d'une grille rectangulaire de cases.

2.1 Donner un type `Case` pour représenter le contenu d'une case du niveau. Une case peut être vide, en terre, en métal, une entrée ou une sortie.

On donne le type suivant pour un niveau:

```
data Niveau = Niveau { hNiveau :: Int,
                       lNiveau :: Int,
                       casesNiveau :: Map Coord Case}
                       deriving Eq
```

Un niveau est donc constitué d'une hauteur et d'une largeur (les dimensions de la grilles) et d'un `Map` allant des coordonnées dans le type `Case`, représentant le contenu des cases de la grille.

2.2 `(I)` Un niveau est cohérent quand les propriétés suivantes sont vraies:

- Un niveau possède exactement une entrée et une sortie.
- Un niveau est entièrement fermé par des cases de métal (c'est-à-dire que les "bords" du niveau sont composés exclusivement de cases `Metal`)
- L'entrée se trouve au dessus d'une case vide et la sortie au dessus d'une case de `Metal`.
- Toutes les coordonnées comprises entre 0 (inclus) et la hauteur/largeur du niveau sont associées à une `Case` et, réciproquement, tous les coordonnées associées à une `Case` sont comprises entre 0 (inclus) et la hauteur/largeur.

Ecrire un invariant pour le type `Niveau`.

2.3 `(+)` Instancier `Show` et `Read` pour les niveaux de telle sorte qu'on puisse lire et écrire des niveaux ressemblant à ça (`X` et `0` représentent terre et métal, `E S` représentent entrée et sortie) :

```
XXXXXXXXXX
X E      X
X        X
X0000    X
X        X
X        X
X   00000X
X        X
X        X
X 0000000X
X        X
X       SX
XXXXXXXXXX
```

Pour la suite on suppose qu'on dispose de deux fonctions `passable :: Coord -> Niveau -> Bool` et `dure :: Coord -> Niveau -> Bool` qui décident, respectivement, si une case d'un niveau est *passable*, c'est-à-dire si elle est vide ou une entrée, et si elle est *dure*, c'est-à-dire si elle en métal ou en terre. On ne demande pas d'écrire ces fonctions.

Partie 3 : Lemmings

Dans un premier temps, les entités mobiles (*mobs*) du jeu sont uniquement des Lemmings, d'un des trois types décrits ci dessous. On suppose qu'un lemming occupe la case à laquelle il se trouve **et** la case juste au dessus de lui (comme s'il était "haut de deux cases"). La case au dessus de celle d'un Lemming doit donc toujours être passable.

- Le *marcheur* regarde dans une direction et représente l'état "par défaut" du lemming, qui marche tout droit. A chaque tour de jeu:
  - il regarde d'abord si la case sous lui est *dure* (cf. plus haut), si ça n'est pas le cas, il devient un tombeur,
  - si c'est le cas, il regarde les cases situées à gauche et en haut à gauche (s'il regarde vers la gauche, sinon c'est "inversé") et si les deux cases sont passables, il se déplace vers la gauche,
  - sinon, si la case à sa gauche est dure, que la case en haut à gauche et passable et que la case en haut-haut-gauche est passable aussi, il se déplace en haut à gauche (il monte un escalier).
  - sinon, il reste sur place mais se met à regarder dans l'autre direction.
- Le *tombeur* regarde dans une direction (gauche ou droite) et représente un lemming en train de chuter:
  - il regarde si la case sous lui est *dure* (cf. plus haut), si ce n'est pas le cas, il se déplace vers le bas.
  - si la case sous lui est dure et qu'il a chuté d'une hauteur plus haute que la hauteur mortelle (à définir, par exemple 8 cases), il devient un lemming mort.
  - si la hauteur n'était pas mortelle, il devient un marcheur qui regarde dans la même direction.
- Le *lemming mort* représente un lemming décédé, à qui il ne peut plus rien arriver.

3.1 Donner un type pour les Lemmings.

3.2 `(+)` Instantier `Placable` avec ce type.

3.3 `(I)` Implémenter une fonction `tourLemming` qui prend un lemming au sein d'un niveau et renvoie le lemming apres un tour de jeu, en suivant les comportements décrits ci-dessus.

3.4 `(I)` Inventer une (ou plusieurs) propriété(s) représentant des post-conditions de `tourLemming`. Décrire ensuite un exemple de séquence de tests *Hspec* pour cette (ces) propriété(s).

Partie 4 : Contenu

Pour finir (le partiel), ajouter du contenu au jeu, au choix :

4.1 Représenter l'état du jeu, composé d'un niveau et d'un ensemble de lemmings munié d'identifiants uniques, évoluant dans ce niveau. Donner des fonctions permettant d'ajouter, enlever, déplacer ou transformer un lemming de l'état à partir de son identifiant. Donner un invariant pour ce type et des propriétés pour ces fonctions.

Puis écrire un *moteur* de jeu, qui d'un état courant permet de passer à un état suivant ou à une sitation de fin du jeu. Le moteur fait apparaître régulièrement des Lemmings depuis l'entrée, supprime les lemmings morts et les lemmings ayant atteint la sortie.

4.2 Ajouter un lemming *creuseur* qui creuse la case directement sous lui jusqu'à arriver dans du vide ou du métal (qu'il ne peut pas creuser). A chaque fois qu'il creuse sous lui, il vide aussi les deux cases à gauche et à droite de la case qu'il creuse, si c'est possible (si elles ne sont pas en métal).

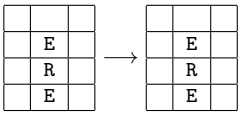
4.4 Ajouter un lemming *stoppeur* qui bloque les autres lemmings en agissant comme un mur pour eux.

4.4 Ajouter un lemming *poseur* qui pose un escalier de terre montant dans la direction dans laquelle il regarde.

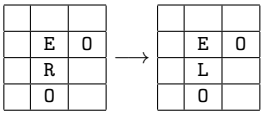
Annexe 1: Comportement du marcheur

Les schémas suivants décrivent le comportement du marcheur droitier (le comportement pour le gaucher est symétrique). Un `R` (resp. un `L`) indique une case vide contenant le lemming considéré, qui est droitier (resp. gaucher), un `E` indique que la case est vide, un `O` indique que la case est un obstacle, une case blanche ne donne pas d'information (la case peut être de n'importe quelle nature)

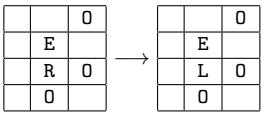
Cas 1 - La case en bas est vide, le lemmings devient un tombeur (il commencera donc sa chute au prochain tour).



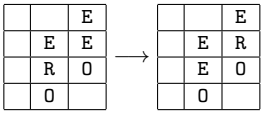
Cas 2.a - La case en haut à droite est un obstacle, le lemmings rebrousse chemin.



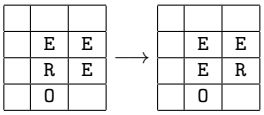
Cas 2.b - Les cases à droite et la case à droite, deux cases en haut sont des obstacles, le lemmings rebrousse chemin.



Cas 3 - La case à droite est un obstacle, mais les deux cases au dessus sont libres, le lemmings monte.



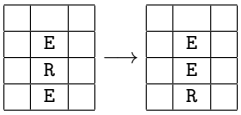
Cas 4 - Les deux cases à droite sont libres, le lemmings avance.



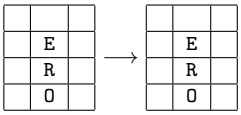
Annexe 2: Comportement du tombeur

Les schémas suivants décrivent le comportement du tombeur droitier (le comportement pour le gaucher est symétrique).

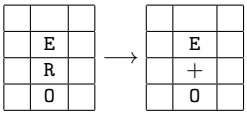
Cas 1 - La case en bas est vide, le lemmings continue à tomber.



Cas 2 - La case en bas est un obstacle, le lemmings tombe depuis moins de (hauteur mortelle) cases, il devient un marcheur.



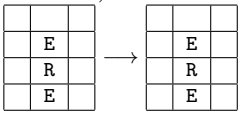
Cas 3 - La case en bas est un obstacle, le lemmings tombe depuis plus de (hauteur mortelle) cases, il meurt.



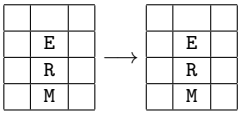
Annexe 3: Comportement du creuseur

Les schémas suivants décrivent le comportement du creuseur droitier (le comportement pour le gaucher est symétrique). Toutes les possibilités du cas 3 ne sont pas données. (`M`: case metal, `D`: case terre)

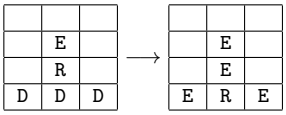
Cas 1 - La case en bas est vide, le lemmings s'arrête de creuser et devient un tombeur (il commencera donc sa chute au prochain tour).



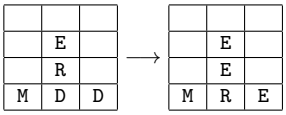
Cas 2 - La case en bas est métal, le lemmings s'arrête de creuser et devient un marcheur.



Cas 3.a - La case en bas est terre, le lemmings creuse (ici les deux autres cases en bas sont terre aussi) et se déplace.



Cas 3.b - La case en bas est terre, le lemmings creuse (ici une des deux autres cases en bas est métal) et se déplace.



Cas 3.c - La case en bas est terre, le lemmings creuse (ici les deux autres cases en bas sont vides) et se déplace.