TP de ReactiveML : Crible de Darwin

Le but de cet exercice est de programmer en ReactiveML le crible de Darwin qui a par exemple été présenté par Gérard Berry lors de ses cours au Collège de France (leçon inaugurale du 19/11/2009, cours 5 du 06/01/2010 et cours 8 du 27/01/2010).

Le principe de ce crible est de représenter un ensemble de nombres comme des processus ayant un mouvement arbitraire. Lorsque deux nombres se rencontrent, si un des deux est multiple de l'autre, alors il disparaît. Ainsi, au bout d'un certain temps, seuls les nombres premiers entre eux restent en jeu.

Le squelette du programme en ReactiveML est disponible à l'adresse :

`http://www.lri.fr/~mandel/mpri/tp-rml/darwin.tgz`

Remarque. Pour compiler un fichier ReactiveML qui s'appelle `file.rml`, il faut compiler le fichier ReactiveML en OCaml :

```
--> rmlc file.rml
```

Puis, il faut compiler le fichier OCaml généré et le lier aux bibliothèques `unix.cma` et `rmllib.cma` pour créer un exécutable :

```
--> ocamlc -o prog -I `rmlc -where` unix.cma rmllib.cma file.ml
```

1 Déplacement des nombres

Chaque nombre est représenté par un disque dont le comportement est d'aller tout droit et de rebondir lorsqu'il rencontre un mur. Pour représenter les nombres et les murs, vous pouvez par exemple utiliser les structures de données suivantes :

```
type coord = { x: float; y: float; }

type number_state =
    { id: int;
      pos: coord;
      speed: coord;
      radius: float;
      color: Graphics.color; }

type wall = { left: float; right: float;
              bot: float; top: float }
```

Question 1
Définir un processus

```
moving_number:
  number_state -> wall -> (number_state, 'a) event -> unit process
```

tel que `run (moving_number init_state s)` *provoque le mouvement d'un nombre dont l'état initial est* `init_state`*. À chaque instant, le processus doit émettre l'état courant du nombre sur le signal* `s`*.*

Question 2
Définir un processus `window: wall -> ('a, number_state) event -> unit process` *qui permette l'observation du crible de Darwin. Ce processus doit initialiser le mode graphique, puis à chaque instant récupérer sur le signal donné en second argument l'état des nombres participant au crible et afficher ces nombres.*

Vous pouvez utiliser la fonction `draw_number: number_state -> unit` *qui permet d'afficher un nombre sur la fenètre graphique.*

Question 3
Écrire une fonction `random_number_state: int -> wall -> number_state` *qui crée une valeur de type* `number_state` *telle que le champ* `id` *soit égal à l'entier passé en argument,* `pos` *et* `speed` *soient initialisés aléatoirement,* `radius` *soit égal à* `12.0` *et* `color` *soit égal à* `Graphics.cyan`*.*

Question 4
Écrire un processus `main: unit process` *qui exécute en parallèle cent instances du processus* `moving_number` *et une instance du processus* `window`*.*

Vous pouvez utiliser la construction `for/dopar` *de ReactiveML. Vous disposez de d'une fonction* `random_number_state: int -> wall -> number_state` *qui crée une valeur de type* `number_state` *telle que le champ* `id` *soit égal à l'entier passé en argument,* `pos` *et* `speed` *soient initialisés aléatoirement,* `radius` *soit égal à* `12.0` *et* `color` *soit égal à* `Graphics.cyan`*.*

2 Collisions

Nous voulons maintenant supprimer les nombres qui ne sont pas premiers lorsqu'ils rencontrent un de leurs diviseurs. Pour cela, on va associer un signal `kill` à chaque nombre :

```
type number_state =
    { id: int;
      pos: coord;
      speed: coord;
      radius: float;
      color: Graphics.color;
      kill: (number_state, number_state option) event; }
```

Question 5
Modifier votre programme pour prendre en compte l'ajout du champs `kill` *dans le type* `number_state`*.*

Question 6
Définir un processus

```
number: number_state -> wall -> (number_state, 'a) event -> unit process
```

qui se déplace selon le comportement de `moving_number` *jusqu'à ce que son signal* `kill` *soit émis avec la valeur de son état courant. Une fois le signal reçu, le nombre doit devenir rouge et sa taille doit réduire. Lorsque le rayon du nombre devient nul, le processus doit se terminer.*

Question 7
Gérer les collisions pour que les nombres qui ne sont pas premiers soient supprimés.

3 Création dynamique

Le but de cette partie est de créer des nouveaux nombres à chaque fois que l'utilisateur va cliquer sur le bouton de sa souris.

Question 8
Définir un processus récursif

```
add: ('a, coord) event ->
       wall -> int -> (number_state, 'b) event -> unit process
```

tel que `add new_number wall n s` *crée un nouveau nombre numéroté à partir de* `n` *à chaque fois qu'il reçoit une position sur le signal* `new_number`*.*

Question 9
Soit le code du processus `click_of_button_down: (coord, 'a) event -> unit process` *suivant qui émet la position de la souris à chaque instant où le bouton de la souris est pressé :*

```
let process click_of_button_down click =
  loop
    if Graphics.button_down() then begin
      let x, y = Graphics.mouse_pos() in
      emit click { x = float_of_int x; y = float_of_int y}
    end;
    pause
  end
```

Écrire un processus `read_click: (coord, 'a) event -> unit` *qui émet sur le signal passé en argument les coordonnées de la souris uniquement à l'instant qui suit le relâchement du bouton de la souris.*

Question 10
Réécrire le processus `main` *pour qu'il gère la création dynamique de nombres.*