Université Paris Diderot — Programmation Fonctionnelle
L2 Math-Info/L3 Informatique/M1 Linguistique — Année 2011-2012

TP n°6

Graphismes

Pour commencer, il vous faudra charger et ouvrir le module `Graphics` d'OCaml :

```
#load "graphics.cma";; (* '#' necessaire *)
open Graphics;;
```

Pour ouvrir une fenêtre graphique de 800 lignes et 500 colonnes :

```
open_graph " 500x800";; (* espaces necessaires *)
```

Après ouverture, le point courant est (0,0) – en bas a gauche. Les fonctions suivantes sont suffisantes pour traiter les exercices qui suivent. Vous pouvez consulter leur documentation et celle des autres fonctions du module `Graphics` sur `caml.inria.fr`.

```
moveto : int -> int -> unit (* deplacer le point courant *)
lineto : int -> int -> unit (* deplacer, en tracant une ligne, le point courant *)
clear_graph : unit -> unit  (* clear_graph() efface la fenêtre *)
set_color : Graphics.color -> unit           (* choix d'une couleur *)
rgb : int -> int -> int -> Graphics.color    (* construction de couleur *)
close_graph : unit -> unit                   (* fermeture de la fenêtre *)
```

Exercice 1. Cet exercice peut être rédigé en écrivant un seule fonction en `unit -> unit`. Entre deux étapes, vous pouvez utiliser la fonction `Graphics.read_key : unit -> char` qui attend que l'utilisateur appuie sur une touche – il suffit de négliger sa valeur de retour.

1. Ouvrir une fenêtre graphique de 800 lignes et 500 colonnes.
2. Dessiner les deux diagonales de la fenêtre.
3. Effacer. Dessiner un carré de côté 200 au centre de la fenêtre (avec `draw_line` ou `draw_rect`).
4. Effacer. Colorier en jaune toute la fenêtre (avec deux boucles imbriquées ou `fill_rect`).
5. Essayer d'autre couleurs par mélange de niveaux de rouge, vert et bleu, ou même, pourquoi pas, des dégradés de couleurs (on écrit `mod 256` pour calculer un modulo).
6. Fermer la fenêtre graphique.

Exercice 2. Ecrire une fonction `pavage : int -> int -> unit` telle que l'évaluation de `pavage n m` ait pour effet de quadriller la fenêtre en dessinant une ligne tous les `n` pixels, et une colonne tous les `m` pixels.

Exercice 3 (Ecran magique)**.** A l'aide de la fonction `Graphics.read_key : unit -> char`, ecrire une fonction `etch : unit -> unit` qui permet de déplacer un curseur vers le haut, la gauche, le bas ou la droite à l'aide des touches W, A, S ou D, et trace une ligne sur son passage. Un appui sur Q terminera la fonction. Modifiez ensuite votre programme pour que l'appui sur la barre d'espace active/désactive le tracé de la ligne (mais continue à déplacer le curseur).

Exercice 4. Le but de cet exercice est d'animer un cercle dans la fenêtre graphique. Le cercle doit bouger en diagonale, et rebondir quand il touche un bord de la fenêtre. La direction initiale peut être vers la droite et le haut. Le point de départ peut être n'importe où, sauf sur le bord de la fenêtre (c.f. plus bas). Pour déplacer le cercle, il suffit de :

- Effacer le contenu de la fenêtre,
- Déplacer le point courant : l'abscisse (resp. l' ordonné) doit être incrémentée si la direction horizontale courante est vers la droite (resp. la direction verticale courante vers le haut), decrémentée sinon.

  Quand le point courant touche le bord droit ou le bord gauche (resp. le bord bas ou le bord haut) de la fenêtre, la direction horizontale (resp. verticale) doit changer.
- Dessiner le cercle de centre le point courant (et de rayon, disons, 10).
- Recommencer

Astuce. On pourra représenter la direction comme une paire de booléens (verticale, horizontale).

Astuce. Pour faire en sorte que l'exécution de la boucle termine au premier clic de souris, on pourra utiliser la fonction `button_down : unit -> bool`. Pour ralentir de 5 millièmes de seconde l'affichage, insérer dans la boucle :

```
let t = Unix.gettimeofday () +. 0.005 in
while (Unix.gettimeofday () < t) do () done
```

(ceci nécessite `#load "unix.cma";;` au debut du fichier).

A partir de ces indications, définir une fonction `animer : unit -> unit` qui anime un cercle dans la fenêtre graphique, et dont l'exécution s'arrête au premier clic de souris.

Exercice 5 (Arbres binaires et Graphisme (?))**.** On considère dans cet exercice les arbres binaires non vides dont les nœuds et les feuilles sont étiquetés par des chaînes de caractères. Un arbre est donc :

- soit une feuille, encapsulant une chaîne de caractères,
- soit un noeud, encapsulant une chaîne de caractères, avec deux fils.

Pour afficher un arbre à une position (`x;y`), on commence par afficher la chaîne `s` de sa racine. La chaîne doit être centrée en `x`, et son bord supérieur doit être placé à la hauteur `y`. Lorsque la racine est un nœud, on dessine ensuite ses deux fils (voir la figure **??**).

Les positions verticales d'affichage des deux fils sont les mêmes, à une distance `b` (constante, fixée) du bord inférieur de `s`. Les deux fils sont séparés par un espace horizontal de taille `a` (constante, fixée). Les positions horizontales des deux fils sont choisies de manière à ce que la racine de l'arbre se trouve exactement au milieu du diagramme (pour cela, il faut en particulier connaître les largeurs d'affichage des deux fils).

Finalement, on dessine deux traits allant du milieu du bord inférieur de `s` jusqu'aux positions d'affichage des deux fils.

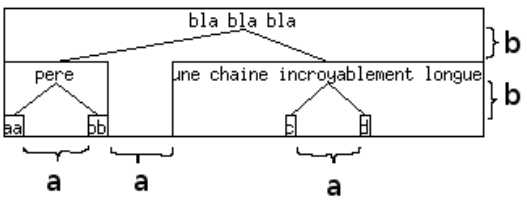

FIGURE 1 – Affichage d'un arbre. Les boites rectangulaires ne sont pas à afficher par la fonction demandée ; elles sont indiquées ici seulement afin d'illustrer l'algorithme d'affichage.

Questions :

1. Donner la définition OCaml d'un type `arbre` pour la représentation des arbres.
2. Ecrire une fonction `taille : arbre -> (int * int)` prenant en argument un arbre et renvoyant une paire consistant en la largeur et la hauteur de son affichage suivant les contraintes ci-dessus.

   On supposera les constantes `a` et `b` déjà définies (de type `int`). La fonction prédéfinie `text_size : string -> (int * int)` permet d'obtenir les dimensions (largeur, hauteur) de l'affichage d'une chaîne de caractères.
3. Ecrire une fonction `afficher : arbre -> unit` prenant en argument un arbre et affichant cet arbre à l'écran à l'aide des fonctions du module `Graphics`. La taille de la fenêtre graphique sera supposée stockée dans deux constantes entières `xsize` et `ysize`. L'arbre devra être centré dans cette fenêtre. Servez-vous de la fonction `taille` de la question précédente. Déclenchez une exception si l'arbre dépasse les dimensions de la fenêtre graphique.