Programmation Fonctionnelle
Cours 09

Michele Pagani

Université Paris Diderot
UFR Informatique
Laboratoire Preuves, Programmes et Systèmes

pagani@pps.univ-paris-diderot.fr

27 novembre 2014

Graphisme

La bibliothèque `graphics.cma`

- une bibliothèque très rudimentaire de fonctionnalités graphiques
  - pour GUI plus avancés voir: `http://lablgtk.forge.ocamlcore.org`

- L'interprète OCaml n'a par défaut pas les fonctionnalités graphiques. Il y a plusieurs possibilités :
  - Charger bibliothèque dans interpréteur : `#load "graphics.cma";;`
  - Inclure bibliothèque au moment du lancement interpréteur : `ocaml graphics.cma`
  - Créer une nouvelle instance interpréteur (voir le manuel): `ocamlmktop -o mytop graphics.cma`

- Pour compiler un programme qui utilise le graphisme : `ocamlc` *other options* `graphics.cma` *other files*

La bibliothèque `graphics.cma`

- une bibliothèque très rudimentaire de fonctionnalités graphiques
  - pour GUI plus avancés voir: `http://lablgtk.forge.ocamlcore.org`
- L'interprète OCaml n'a par défaut pas les fonctionnalités graphiques. Il y a plusieurs possibilités :
  - Charger bibliothèque dans interpréteur : `#load "graphics.cma";;`
  - Inclure bibliothèque au moment du lancement interpréteur : `ocaml graphics.cma`
  - Créer une nouvelle instance interpréteur (voir le manuel): `ocamlmktop -o mytop graphics.cma`
- Pour compiler un programme qui utilise le graphisme : `ocamlc` *other options* `graphics.cma` *other files*

La bibliothèque `graphics.cma`

- une bibliothèque très rudimentaire de fonctionnalités graphiques
  - pour GUI plus avancés voir:
    `http://lablgtk.forge.ocamlcore.org`
- L'interprète OCaml n'a par défaut pas les fonctionnalités graphiques. Il y a plusieurs possibilités :
  - Charger bibliothèque dans interpréteur :
    `#load "graphics.cma";;`
  - Inclure bibliothèque au moment du lancement interpréteur :
    `ocaml graphics.cma`
  - Créer une nouvelle instance interpréteur (voir le manuel):
    `ocamlmktop -o mytop graphics.cma`
- Pour compiler un programme qui utilise le graphisme :
  `ocamlc` *other options* `graphics.cma` *other files*

La bibliothèque `graphics.cma`

- une bibliothèque très rudimentaire de fonctionnalités graphiques
  - pour GUI plus avancés voir: `http://lablgtk.forge.ocamlcore.org`
- L'interprète OCaml n'a par défaut pas les fonctionnalités graphiques. Il y a plusieurs possibilités :
  - Charger bibliothèque dans interpréteur : `#load "graphics.cma";;`
  - Inclure bibliothèque au moment du lancement interpréteur : `ocaml graphics.cma`
  - Créer une nouvelle instance interpréteur (voir le manuel): `ocamlmktop -o mytop graphics.cma`
- Pour compiler un programme qui utilise le graphisme : `ocamlc` *other options* `graphics.cma` *other files*

La bibliothèque `graphics.cma`

- une bibliothèque très rudimentaire de fonctionnalités graphiques
  - pour GUI plus avancés voir: `http://lablgtk.forge.ocamlcore.org`
- L'interprète OCaml n'a par défaut pas les fonctionnalités graphiques. Il y a plusieurs possibilités :
  - Charger bibliothèque dans interpréteur : `#load "graphics.cma";;`
  - Inclure bibliothèque au moment du lancement interpréteur : `ocaml graphics.cma`
  - Créer une nouvelle instance interpréteur (voir le manuel): `ocamlmktop -o mytop graphics.cma`
- Pour compiler un programme qui utilise le graphisme : `ocamlc` *other options* `graphics.cma` *other files*

La bibliothèque `graphics.cma`

- une bibliothèque très rudimentaire de fonctionnalités graphiques
  - pour GUI plus avancés voir: `http://lablgtk.forge.ocamlcore.org`
- L'interprète OCaml n'a par défaut pas les fonctionnalités graphiques. Il y a plusieurs possibilités :
  - Charger bibliothèque dans interpréteur : `#load "graphics.cma";;`
  - Inclure bibliothèque au moment du lancement interpréteur : `ocaml graphics.cma`
  - Créer une nouvelle instance interpréteur (voir le manuel): `ocamlmktop -o mytop graphics.cma`
- Pour compiler un programme qui utilise le graphisme : `ocamlc` *other options* `graphics.cma` *other files*

La fenêtre graphique

(`Open Graphics;;` plus besoin de la notation pointée pour cette bibliothèque)

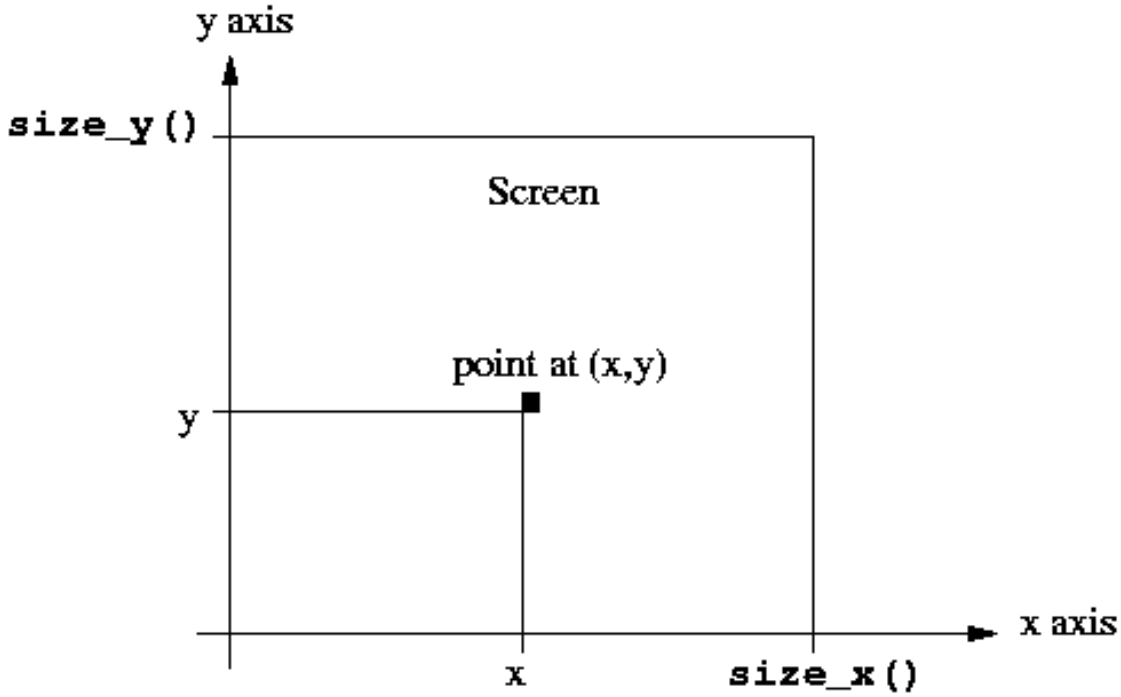
- `open_graph " lxh"`: crée fenêtre graphique, où *l* et *h* sont le nombre de pixels pour la largeur (l) et hauteur (h), e.g.:

  `open_graph " 600x400"`

  - attention espace début argument
    (obligatoire pour UNIX, voir manuel pour Windows)
  - on peut avoir une seule fenêtre graphique

- `close_graph:unit->unit` ferme fenêtre graphique
- `clear_graph:unit->unit` efface contenu fenêtre graphique
- `set_window_title:string->unit` donne un titre à la fenêtre

Coordonnés sur le canevas graphique



L'origine (0,0) est en bas à gauche

Dessiner

- Il y a un curseur, qui au début se trouve à l'origine (0,0).
- `current_point:unit->int*int` renvoie position du curseur
- `moveto x y` positionne curseur en (x, y)
- `plot x y` dessine point à position (x, y) et positionne curseur en (x, y)
- `lineto x y` dessine une ligne de position actuelle curseur à (x, y), et positionne le curseur en (x, y)
- `set_line_width n` sélectionne n pixels comme épaisseur lignes
- `draw_char c` affiche caractère c à position curseur
- `draw_string s` affiche chaîne s à position curseur

Exercice (grille)

- Définir une fonction `grille:int->int->unit`, qui, étant donnés deux entiers `col` et `lin`, dessine sur la fenêtre graphique une grille de `lin` lignes et `col` colonnes.

Solution (grille)

```
1  open Graphics;;
2
3  open_graph " 200x600";;
4  set_window_title "Grid";;
5
6  let grid nx ny =
7   let diffx = size_x()/(nx+1) in
8   let diffy = size_y()/(ny+1) in
9   let i = ref 0 in
10  while (!i <=size_x()) do
11   moveto !i 0 ;
12   lineto !i (size_y()) ;
13   i := !i+diffx
14  done ;
15  i := 0 ;
16  while (!i <=size_y()) do
17   moveto 0 !i ;
18   lineto (size_x()) !i  ;
19   i := !i+diffy
20  done
```

Polygones et courbes

- `draw_rect x y l h` dessine un rectangle avec vertex bas gauche en (x, y), largeur l, hauteur h
- `draw_poly_line:(int * int) array->unit` dessine une ligne qui joint les points donnés dans le tableau
- `draw_poly:(int * int) array->unit` dessine le polygone (ligne fermé) qui joint les points donnés dans le tableau
- `draw_circle x y r` dessine un cercle de centre (x, y) et rayon r
- `draw_ellipse x y rx ry` dessine ellipse de centre (x, y), rayon horizontal rx et vertical ry
- `draw_arc x y rx ry a1 a2` dessine arc elliptique de centre (x, y), rayon horizontal rx et vertical ry, entre angles $a1$ et $a2$ (en dégrées)

Exercice (arbre de Noël)

- Définir une fonction `noel:int -> unit` qui, étant donné un entier *n* dessine un arbre de Noël de *n* triangles de plus en plus petits. Utiliser l'interface suivante:

```
(*type contenant base, hauteur et sommet superieur*)
type triangle

(*move_scale_triangle t s dx dy                    *)
(*change base et haut en function de scale s *)
(*et deplace sommet de dx et dy                    *)
val move_scale_triangle:triangle->int->int->int->unit

val draw_triangle:triangle->unit

val noel:int->unit
```

Solution (arbre de Noël)

```
1  open Graphics;;
2  open_graph " 500x500";;
3
4  type triangle = {
5    mutable base : int ;
6    mutable hauteur : int ;
7    mutable sommet : int*int
8  }
9
10 let move_scale_triangle t s dx dy =
11  t.base <- (t.base/s) ;
12  t.hauteur <- (t.hauteur/s) ;
13  let (x,y) = t.sommet in
14  t.sommet <- (x+dx,y+dy)
15
16 let draw_triangle t =
17  let (x0,y0) = t.sommet in
18  let b1 = (x0-t.base/2, y0-t.hauteur) in
19  let b2 = (x0+t.base/2, y0-t.hauteur) in
20  draw_poly [|t.sommet; b1; b2|]
21
22 let noel n =
23  let s = 2 in (*scale factor*)
24  let t = {base = size_x() -4;
25       hauteur = size_y()/2;
26       sommet = size_x ()/2,size_y()/2+4}
27       in
28  for i = 1 to n do
29   draw_triangle t;
30   move_scale_triangle t s 0 (t.hauteur/s)
31   done
```

Couleurs

- `color` un type représentant les couleurs
- constantes prédéfinies de `color`:

  `black`, `white`, `red`, `green`, `blue`, `yellow`, `cyan`, `magenta`

- `rgb r v b` renvoie la couleur (type `color`) avec composantes rouge r, verte v et bleue b.
  Les valeurs légales pour arguments: de 0 à 255.
- `set_color c` sélectionne c comme la couleur courante
- Function `fill_XXX` : remplir le polygone `XXX` dans la couleur courante:

  `fill_rect`, `fill_poly`, `fill_arc`, `fill_ellipse`, `fill_circle`

Exercice (arbre de Noël coloré)

- Modifier la fonction `noel:int -> unit` de l'exercice précédent pour qu'il dessine un arbre coloré.

Solution (arbre de Noël coloré)

```
1  open Graphics;;
2  open_graph "␣500x500";;
3
4  type triangle = {
5    mutable base : int ;
6    mutable hauteur : int ;
7    mutable sommet : int*int
8  }
9
10 let move_scale_triangle t s dx dy =
11  t.base <- (t.base/s) ;
12  t.hauteur <- (t.hauteur/s) ;
13  let (x,y) = t.sommet in
14  t.sommet <- (x+dx,y+dy)
15
16 let fill_triangle t =
17  let (x0,y0) = t.sommet in
18  let b1 = (x0-t.base/2, y0-t.hauteur) in
19  let b2 = (x0+t.base/2, y0-t.hauteur) in
20  fill_poly [|t.sommet; b1; b2|] (*on colore deadans le poly*)
21
22 let noel n =
23  let s = 2 in (*scale factor*)
24  let t = {base = size_x() -4;
25       hauteur = size_y()/2;
26       sommet = size_x ()/2,size_y()/2+4}
27       in
28  set_color green;  (*change la couloeur en vert*)
29  for i = 1 to n do
30   fill_triangle t;
31   move_scale_triangle t s 0 (t.hauteur/s)
32   done
```

Interaction avec l'utilisateur

Le type event

```
type event =
  Button_down | Button_up | Key_pressed | Mouse_motion
```

- Un événement se produit quand l'utilisateur clique sur un bouton de la souris, déplace la souris ou presse une touche du clavier. Le type `event` contient les formes différentes des événements.

- `wait_next_event:event list -> status`
  prend comme argument une liste *l* d'événements et attend le prochain événement appartenant à la liste *l* (les autres événements seront ignorés). Quand le premier événement se produit une description détaillée est renvoyée, du type `status`.

Le type event

```
type event =
  Button_down | Button_up | Key_pressed | Mouse_motion
```

- Un événement se produit quand l'utilisateur clique sur un bouton de la souris, déplace la souris ou presse une touche du clavier. Le type `event` contient les formes différentes des événements.

- `wait_next_event:event list -> status`
  prend comme argument une liste *l* d'événements et attend le prochain événement appartenant à la liste *l* (les autres événements seront ignorés). Quand le premier événement se produit une description détaillée est renvoyée, du type `status`.

Le type status

```
type status =
{
mouse_x    : int;   (* coordonnee x de la souris          *)
mouse_y    : int;   (* coordonnee y de la souris          *)
button     : bool;  (* bouton de la souris est enfonce ?  *)
keypressed : bool;  (* touche clavier a ete pressee ?     *)
key        : char;  (* touche pressee clavier si le cas   *)
}
```

- Remarque : il n'y a aucune distinction entre les boutons différents de la souris.

Exemple (paint.ml)

```
1  open Graphics;;
2  open_graph " 500x500";;
3  exception Quit;;
4  let rec loop t =
5    let eve = wait_next_event [Mouse_motion;Key_pressed]
6    in
7    if eve.keypressed
8    then
9      match eve.key with
10       | 'b' -> set_color black; loop t
11       | 'r' -> set_color red; loop t
12       | 'g' -> set_color green; loop t
13       | 'q' -> raise Quit
14       | '0'..'9' as x -> loop (int_of_string (String.make 1 x))
15       | _ -> loop t
16   else begin
17     fill_circle (eve.mouse_x-t/2) (eve.mouse_y-t/2) t;
18     loop t
19   end
20 in
21 try loop 5
22 with Quit -> close_graph ();;
```

Exercice (arbre de Noël avec décorations)

- Écrire une fonction qui permets de placer des boules sur l'arbre de Noël de l'exercice précédent, en utilisant la souris.

Solution (arbre de Noël avec décorations)

```
1  (*changement pour faire les boules*)
2  let rec boule_souris r =
3   let event = wait_next_event [Button_down; Key_pressed] in
4   if event.keypressed = true then
5    if event.key = 'q' then () else boule_souris r
6   else
7    let (x,y) = (event.mouse_x, event.mouse_y) in
8    fill_circle x y r ; boule_souris r
9
10
11 let noel n =
12  let s = 2 in (*scale factor*)
13  let t = {base = size_x() -4;
14       hauteur = size_y()/2;
15       sommet = size_x ()/2,size_y()/2+4}
16       in
17  set_color green;  (*change la couloeur en vert*)
18  for i = 1 to n do
19   fill_triangle t;
20   move_scale_triangle t s 0 (t.hauteur/s)
21   done ;
22  set_color red ;
23  boule_souris 8
```

Exercice (morpion graphique)

- Écrire une interface graphique pour le jeu du morpion défini dans le TP5.
- Une interface possible (demandant quelque changement dans `programme.ml`) est:

```
(*draw O ou X with center x y*)
val draw_pion : int -> int -> pion -> unit

(*draw an empty grid with center x y and color*)
val draw_grid : int -> int -> color -> unit

(*draw, at the center of window, a grid with its pions*)
val afficheGrille : grille -> unit

exception Exit

(*read from mouse next coordinates*)
(*raise Exit if pressed 'q' *)
val read_mouse : unit -> char * int
```