

Programmation Fonctionnelle

Cours 10

Graphisme

Delia Kesner

Bibliothèque graphics

- ▶ Elle est assez "datée", mais d'usage simple et encore largement disponible
- ▶ Sa documentation :
<<https://ocaml.github.io/graphics/graphics/Graphics/index.html>>

Utilisation

- ▶ Par défaut, l'interprète OCaml n'a pas de primitives graphiques. Plusieurs solutions:
 - ▶ Charger la bibliothèque dans l'interpréteur :
`#load "graphics/graphics.cma";;`
 - ▶ Inclure la bibliothèque au moment du lancement de l'interpréteur :
`ocaml graphics/graphics.cma` au lieu de `ocaml`
 - ▶ Créer une nouvelle instance de l'interpréteur à l'aide de la commande `ocamlmktop` (voir le manuel)
- ▶ Pour compiler un programme qui utilise le graphisme, le plus simple est d'utiliser l'outil 'dune' et d'indiquer une dépendance envers la bibliothèque `graphics`.

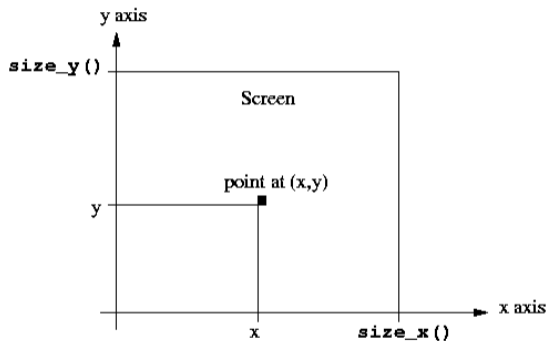
La fenêtre graphique

- ▶ `Open Graphics;;`
Toutes les fonctions (types, exceptions) de cette bibliothèque deviennent disponible (on n'a plus besoin de la notation pointée `Graphics.quelquechose`)
- ▶ On peut avoir une seule fenêtre graphique, créée par `open_graph " l x h "` où *l* et *h* sont le nombre de pixels pour la largeur (*l*) et hauteur (*h*). Par exemple:
`open_graph " 800x600 "`
Attention l'espace au début de l'argument est obligatoire en Unix mais à enlever sous Windows.
- ▶ `close_graph: unit -> unit` ferme la fenêtre graphique.
- ▶ `clear_graph: unit -> unit` efface le contenu de la fenêtre graphique.

Utilisation de graphics

```
(*la localisation de graphics dans votre machine*)  
#directory "/usr/lib/ocaml/graphics";;  
  
#load "graphics.cma";;  
  
open Graphics;;  
  
open_graph " 600x400";;  
  
close_graph ();;
```

Coordonnées sur le canvas graphique



L'origine (0,0) est en bas à gauche

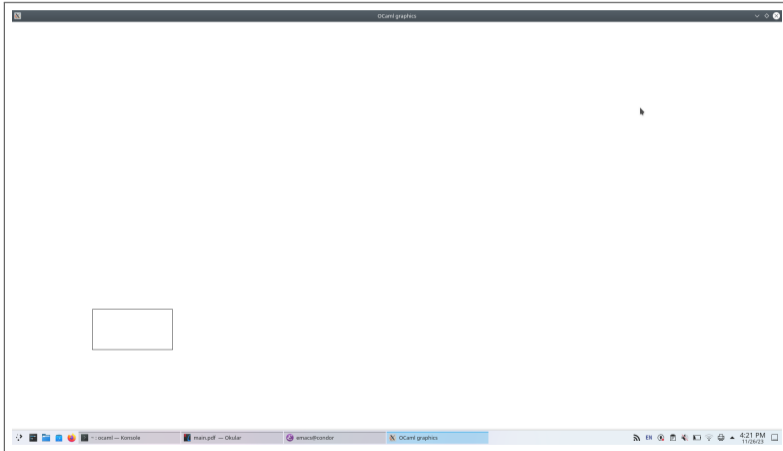
Dessiner

- ▶ Il y a un curseur, qui se trouve au début à l'origine $(0,0)$, en bas à gauche.
- ▶ `(plot x y)` dessine un point à la position (x,y) et positionne le curseur graphique en ce point ;
- ▶ `(moveto x y)` positionne le curseur graphique en (x,y) ;
- ▶ `(lineto x y)` dessine un trait du curseur graphique de la position actuelle du curseur à (x,y) , et positionne le curseur graphique en (x,y) ;
- ▶ Il existe également des versions **relatives** de ces fonctions : `rmoveto` et `rlineto`.
- ▶ `(set_line_width n)` sélectionne n pixels comme épaisseur des lignes.

Dessiner

```
#directory "/usr/lib/ocaml/graphics";;  
#load "graphics.cma";;  
open Graphics;;  
  
open_graph " 800x600";;  
  
moveto 200 200;;  
lineto 400 200; lineto 400 300;  
lineto 200 300; lineto 200 200;;  
  
set_line_width 5;;  
moveto 150 150;;  
lineto 450 150; lineto 450 350;  
lineto 150 350; lineto 150 150;;  
  
close_graph ();;
```


Résultat



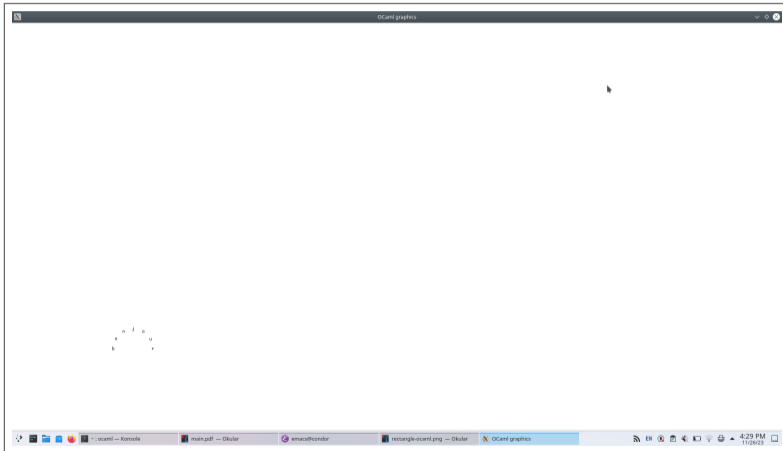
Textes

- ▶ `(draw_char c)` affiche le caractère `c` à la position actuelle du curseur graphique
- ▶ `(draw_string s)` affiche la chaîne `s` à la position actuelle du curseur graphique
- ▶ `(set_font_size n)` sélectionne `n` comme taille de police de caractère, mais ... ne marche pas !
- ▶ `(text_width s)` renvoie la paire `(largeur, hauteur)` de la chaîne `s` quand elle est affichée dans la fonte courante.

Dessiner

```
#directory "/usr/lib/ocaml/graphics";;
#load "graphics.cma";;
open Graphics;;
open_graph " 800x600";;
let pi = 3.1415927 ;;
let dessine (xo,yo) radius l =
  let rec des alpha i = function
    | [] -> ()
    | h::r ->
      moveto
      (xo - (int_of_float (cos(alpha *. i)*.radius)))
      (yo + (int_of_float (sin(alpha *. i)*.radius)));
      draw_char h;
      des alpha (i +. 1.) r
  in des (pi /. (float_of_int ((List.length l)-1))) 0. 1;;
dessine (300,200) 50. ['b'; 'o'; 'n'; 'j'; 'o'; 'u'; 'r'];;
```

Résultat



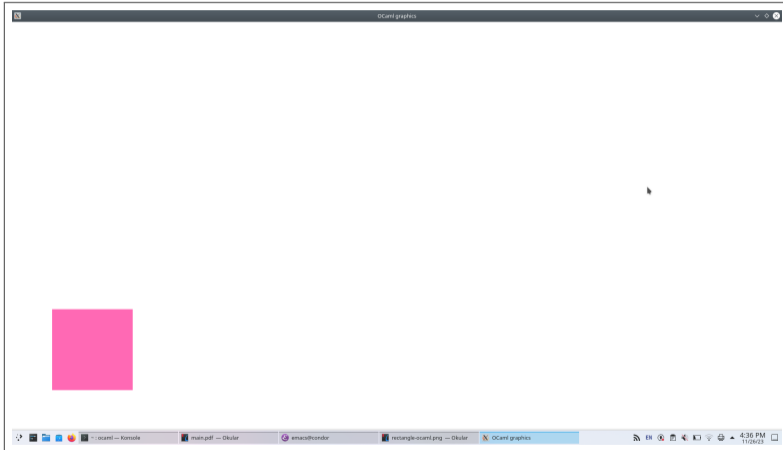
Couleurs

- ▶ Il y a un type `color` représentant les couleurs.
- ▶ Les constantes prédéfinies du type `color` sont `black`, `white`, `red`, `green`, `blue`, `yellow`, `cyan`, `magenta`.
- ▶ `(rgb r v b)` renvoie la couleur (du type `color`) avec les composantes rouges r , verte v et bleue b . Les valeurs légales pour les arguments sont de 0 à 255.
- ▶ `(set_color c)` sélectionne c comme la couleur courante ;
- ▶ `(fill_rect x y l h)` remplit le rectangle de largeur l , de hauteur h et de point inférieur gauche (x, y) par la couleur courante.

Couleurs

```
#directory "/usr/lib/ocaml/graphics";;  
#load "graphics.cma";;  
open Graphics;;  
  
open_graph " 600x400";;  
  
let shocking_pink = rgb 255 105 180;;  
  
set_color shocking_pink;;  
  
fill_rect 100 100 200 200;;  
  
close_graph ();;
```

Résultat



Événements

- ▶ 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 :

```
type event =  
  Button_down | Button_up | Key_pressed | Mouse_motion ;;
```
- ▶ La fonction `wait_next_event` prend comme argument une liste / d'événements et attend le prochain événement appartenant à la liste / (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;  (* coordonnée x de la souris      *)  
  mouse_y      : int;  (* coordonnée y de la souris      *)  
  button       : bool; (* un bouton de la souris est enfoncé ? *)  
  keypressed   : bool; (* une touche du clavier a été pressée ? *)  
  key          : char; (* touche pressée du clavier le cas échéant*)  
}
```

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

Événements I

```
open Graphics;;  
# # # # - : unit = ()  
open_graph "□500x500";;  
# exception Quit  
exception Quit;;  
# - : unit = ()  
let rec loop t =  
  let eve = wait_next_event [Mouse_motion;Key_pressed]  
  in  
  if eve.keypressed  
  then  
    match eve.key with  
    | 'b'  -> set_color black; loop t  
    | 'r'  -> set_color red; loop t  
    | 'g'  -> set_color green; loop t
```

Événements II

```
    | 'q' -> raise Quit
    | '0'..'9' as x -> loop (int_of_string (String.make 1 x))
    | _ -> loop t
else begin
    fill_circle (eve.mouse_x-t/2) (eve.mouse_y-t/2) t;
    loop t
end
in
try loop 5
with Quit -> close_graph ();;
```