Projet individuel d'algorithmique-programmation
AP1 :
groupe 1.1 - complément bibliothèque graphique

octobre 2010

1 Affichage graphique statique

Il en existe trois bibliothèques graphiques en Ocaml : la petite bibliothèque `Graphics` intégrée à la distribution d' Ocaml d'une part et deux interfaces avec des grosses bibliothèques du domaine public GTK+ et TK qui demandent une installation particulière et un investissement beaucoup plus lourd. La petite bibliothèque suffit largement à nos besoins, nous allons donc l'utiliser ici.

2 Présentation de la bibliothèque graphique

Pour disposer de cette bibliothèque dans la boucle interactive lancée par la commande `ocaml`, nous allons ouvrir l'interface du module `Graphics`, mais aussi charger l'implémentation de ce module avec les deux commandes suivantes :

```
open Graphics;;
#load "graphics.cma";;
```

Nous allons présenter les fonctions qui nous seront nécessaires dans un premier temps. La figure 2 présente un tableau synthétique des principales fonctions. Le nom du module `Graphics` est omis dans la première colonne du tableau.

La plupart des fonctions de la bibliothèque graphique ont un résultat de type `unit`. Ce type n'a qu'une seule valeur notée `()` et qui signigie rien. Ceci veut dire que ces fonctions ne calculent rien, elles agissent par effet de bord, en modifiant la fenêtre graphique. On a quitté ici le monde du fonctionnel pur.

2.0.1 La fenêtre graphique

Dans cette bibliothèque, nous disposons d'une seule fenêtre graphique qui sous Unix peut être ouverte aux dimensions et à l'emplacement choisi par le programmeur par la fonction `Graphics.open_graph` de type `string -> unit`. Ces informations sont fournies sous forme d'une chaîne de caractères passée en

argument. Sous Windows cette information est ignorée. Nous nous plaçons ici dans le cadre le plus riche.

La chaîne de caractères [1] est de la forme *largeur*×*hauteur* pour ce qui est des dimensions de la fenêtre suivie de $\pm dx \pm dy$. Le signe `+` (resp. `-`) indique un décalage à partir de la gauche ou du bas (resp. de la droite ou du haut). Avant d'ouvrir la fenêtre graphique, on a intérêt à se demander si l'on a des souhaits particuliers dans ce domaine.

On ferme la fenêtre graphique et on efface tout son contenu par les fonctions `Graphics.close_graph` et `Graphics.clear_graph` de type `unit -> unit`.

On peut attribuer un titre à la fenêtre avec la fonction `Graphics.set_window_title` de type `string -> unit`.

Les dimensions de la fenêtre sont disponibles à tout instant (rien n'interdit à l'utilisateur de modifier la taille de la fenêtre) par les fonctions `Graphics.size_x` et `Graphics.size_y` de type `unit -> unit`. La plupart des fonctions tiennent

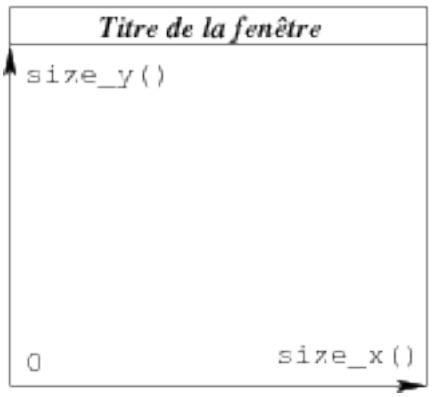

FIGURE 1 – Fenêtre graphique et repère

compte de la couleur courante, de la police courante (avec laquelle on écrit le texte par défaut), du point courant (la position courante de la plume du dessinateur), de l'épaisseur courante des traits, etc. ce qui permet de diminuer le nombre de leurs paramètres.

2.0.2 Les couleurs

Les couleurs sont représentées avec le codage rgb (*Red Green Blue*, connu aussi sous le nom RVB en français) : une couleur est représentée par un entier de 24 bits groupés en 3 paquets de 8 bits représentant un entier compris entre 0 (moins) et 255 (plus) qui indiquent les quantités de rouge, de vert ou de bleu présentes dans cette couleur.

1. On peut indiquer en outre le nom de la machine sur l'écran de laquelle la fenêtre doit s'afficher, pour plus de détails se reporter à la documentation d'Ocaml.

Il existe des couleurs prédéfinies : red, green, blue, yellow, magenta, cyan, black, white.

Nous nous intéressons ici seulement à la façon d'obtenir du gris puisque nous ne disposons pas de couleurs à proprement parler dans cet ouvrage, mais certaines pièces seront à fond gris. Les gris sont obtenus en utilisant la même quantité de rouge, vert et bleu, du plus foncé (avec 0 0 0) au plus clair (avec 255 255 255).

On saisit une couleur soit par un nombre hexadécimal au format `0xRRGGBB`, soit en utilisant la fonction `Graphics.rgb` qui prend les trois entiers dans l'ordre en arguments. On en fait la couleur courante avec la fonction `Graphics.set_color` de type `color -> unit` où `color` est une abréviation du type `int`.

2.0.3 Le texte

On écrit un caractère (resp. une chaîne de caractères) - dans la couleur courante, la police courante et au point courant - avec la fonction `Graphics.draw_char` (resp. `Graphics.draw_string`) de type `char -> unit` (resp. `string -> unit`). Pour changer la police de caractères, on utilise la fonction `Graphics.set_font` de type `string -> unit`. La chaîne de caractères passée en argument décrit la police. Nous reviendrons sur son format lorsque nous nous en servirons. La fonction `Graphics.set_font_size` de type `int -> unit` complète la fonction `Graphics.set_font` lorsque l'information sur la taille des caractères est indépendante de la police (sous Windows ou MacIntosh).

Pour effectuer des justifications, il est nécessaire avant d'écrire un texte, d'en connaître l'encombrement. Pour cela on utilise la fonction `Graphics.text_size` de type `unit -> int*int` qui indique la largeur et la hauteur de la boîte minimale qui peut contenir le texte pour la police et la taille de caractères courante. Cette fonction nécessite malheureusement que la fenêtre graphique soit déjà ouverte pour pouvoir être utilisée.

2.0.4 Le point courant, les mouvements et les traits

Pour connaître les coordonnées du point courant, on dispose des fonctions `Graphics.current_x`, `Graphics.current_y` et `Graphics.current_point` de type `unit -> int`, `unit -> int` et `unit -> int*int` respectivement. On s'en sert exceptionnellement car la plupart du temps on sait où l'on se trouve. [2]

On peut se déplacer soit en absolu par la fonction `Graphics.moveto`, soit en relatif par la fonction `Graphics.rmoveto`, toutes deux de type `int -> int -> unit`. L'expression `moveto x y` déplace le point courant au point de coordonnées `(x,y)` ; `rmoveto dx dy` déplace le point courant de `dx` pixels vers la droite et `dy` pixels vers le haut. Des valeurs négatives pour `dx` et `dy` provoquent bien évidemment des déplacements en sens inverse.

Le tracé d'un pont s'obtenient avec la fonction `Graphics.plot` de type `int -> int -> unit`. Les deux arguments sont les coordonnées du point à afficher.

2. On peut marquer un point ou un essaim de points dans la couleur courante ou retrouver la couleur du point courante, se reporter à la documentation d'Ocaml.

On obtient les tracés rectilignes équivalents avec les fonctions `Graphics.lineto` et `Graphics.rlineto`. On dispose de fonctions prédéfinies qui tracent des courbes telles que `Graphics.draw_rect`, `Graphics.draw_circle`. Nous détaillons ici les fonctions `Graphics.draw_rect` et `Graphics.draw_poly` :

- `draw_rect x y w h` dessine le rectangle de coin inférieur gauche de coordonnées `(x, y)`, de largeur `w` et de hauteur `h` ;
- `draw_poly p` dessine le polygône représenté par le tableau `p` des coordonnées de ses sommets consécutifs.

Enfin on peut modifier l'épaisseur du trait avec la fonction `Graphics.set_line_width` de type `int -> unit`. L'argument entier est le nombre de pixels de l'épaisseur du trait. Par défaut cette valeur est 1.

3 Utilisation de ces fonctions : la séquence et les effets de bords

Comme on l'a déjà dit précédemment, utiliser ces fonctions signifie programmer par effets de bord (i.e. que l'on modifie une structure annexe à savoir ici la fenêtre graphique). On aura besoin dans les fonctions du projet qui réalisent la représentation graphique des quadtrees d'utiliser la séquence (**et là uniquement**).

La construction `e1;e2` signifie que l'on évalue d'abord `e1` puis `e2` où `e1` est une expression de type `unit` et `e2` une expression quelconque. La séquence a le type de `e2`. En général `e1` effectue un effet de bord, comme par exemple écrire quelque chose dans un fichier ou dans une fenêtre graphique.

Nom	Type	Action
Fenêtre		
`open_graph`	`string -> unit`	`open_graph ""` ouvre la fenêtre graphique standard
`clear_graph`	`unit -> unit`	`clear_graph ()` efface la fenêtre
`close_graph`	`unit -> unit`	`close_graph ()` ferme la fenêtre
`size_x`	`unit -> int`	`size_x ()` et `size_y ()` sont les
`size_y`	`unit -> int`	largeur et hauteur actuelles de la fenêtre
`set_window_title`	`string -> unit`	`set_window_title s` donne le titre `s` à la fenêtre
Couleurs		
`set_color`	`color -> unit`	après l'exécution de `set_color c`, les traits et surfaces tracés ont la couleur `c`
`rgb`	`int -> int -> int -> color`	`rgb x x x` : couleur grise
`white`	`color`	la couleur blanche
`background`	`color`	la couleur de fond, pour effacer
`foreground`	`color`	la couleur d'écriture
Texte, affichage dans la fenêtre graphique		
`draw_char`	`char -> unit`	`draw_char c` affiche le caractère `c`
`draw_string`	`string -> unit`	`draw_string s` affiche la chaîne `s`
`set_font`	`string -> unit`	après l'exécution de `set_font f`, les textes sont affichés dans la police de caractères `f`
`text_size`	`unit -> int*int`	`text_size s` sont les largeur et hauteur de la chaîne `s` dans la police de caractères actuelle
Traits et surfaces		
`set_line_width`	`int -> unit`	après l'exécution de `set_line_width e`, les traits tracés ont l'épaisseur `e`
`draw_rect`	`int -> int -> int -> int -> unit`	`draw_rect x y w h` trace un rectangle de coin inférieur gauche `(x, y)`, de largeur `w` et de hauteur `h` dans la couleur courante avec l'épaisseur de trait courante
`fill_rect`	`int -> int -> int -> int -> unit`	`fill_rect x y w h` colorie un rectangle de coin inférieur gauche `(x, y)`, de largeur `w` et de hauteur `h` dans la couleur courante
`draw_poly`	`(int*int) array -> unit`	`draw_poly p` trace le polygône décrit par le tableau de sommets `p`

FIGURE 2 – Tableau récapitulatif des fonctions de la bibliothèque graphique utilisées dans ce chapitre