TP6 : Interface graphique de Caml Light

Christophe Rose

Vendredi 11 avril 2008

christophe.rose@ens.fr

http ://www.eleves.ens.fr/home/rose/caml

Dans ce TP, nous allons utiliser l'interface graphique de Caml Light.

1 Initiation à l'interface graphique

Pour utiliser les fonctions graphiques de Caml, il faut charger le module correspondant avec la commande `#open "graphics"` puis créer une fenêtre avec `open_graph "640x480"`. On la ferme avec la commande `close_graph ()`.

- `clear_graph ()` permet d'effacer la fenêtre graphique.
- Pour changer la couleur utilisée, on utilise la fonction `set_color` suivi de la couleur (`black`, `white`, `blue`, `yellow`, etc.).
- `rgb r g b` crée une couleur pour des arguments entiers compris entre 0 et 255, correspondant à la quantité de rouge, vert et bleu (synthèse additive).
- `moveto x y` permet de placer le point courant en (x, y). Le point $(0, 0)$ se situe au sud-ouest.
- `plot x y` dessine un point en (x, y).
- `lineto x y` dessine une droite partant du point courant jusqu'en (x, y).

Question 1. Avec les fonctions `int_to_float` et `float_to_int`, dessinez la fonction cosinus. L'unité mesurera 200 points et l'origine sera en $(320, 240)$. On pourra également dessiner les axes.

Question 2. Dessinez la courbe polaire d'équation $\rho = \sin\theta$ pour $\theta \in [0; \pi]$ et la courbe paramétrée $(t^3 + t + 1, t^4 + t + 1)$ pour $t \in [-1; 1]$.

Une matrice de type `'a` est un tableau de tableaux de type `'a`.

On ne peut pas créer une matrice simplement par les commandes suivantes :

```
# let A =
  let v = make_vect nb_colonnes 0 in
  make_vect nb_lignes v ;;
```

Sinon, la commande `A.(0).(0)<-1` serait l'équivalent de `v.(0)<-1` et modifierait toute la première ligne et pas seulement le premier élément de la matrice.

Il faut donc recréer avec une boucle tous les vecteurs ligne.

```
# for i = 0 to (nb_lignes - 1) do
  let v = make_vect nb_colonnes 0 in
  A.(i)<-v;
  done;
  ;;
```

Question 3. Écrivez une fonction `matrice_vide` telle que `matrice_vide colonnes lignes` crée une matrice booléenne de taille donnée remplie de `false`. Écrivez de même une fonction `matrice_aleatoire` qui crée une matrice aléatoire.

2 Fourmi de Langton

On part d'une matrice 640×480 remplie de booléens, en utilisant la convention `true=black` et `false=white`. Un agent (appelé *fourmi de Langton*) se déplace dans la matrice avec une direction donnée (nord/ouest/sud/est).

Lorsqu'il tombe sur une case noire, il la repeint en blanc et tourne à gauche. Lorsqu'il tombe sur une case blanche, il la repeint en noir et tourne à droite. Après avoir tourné, il se déplace d'une case en avant.

On utilisera le type direction défini ci-dessous.

```
type direction = N | S | O | E ;;
```

Question 4. Programmez deux fonctions `gauche` et `droite` de type `direction -> direction` qui calculent respectivement la gauche et la droite d'une direction.

Question 5. Écrivez une fonction `affiche_matrice` de type `color * color -> bool vect vect -> unit` qui affiche une matrice de booléens de taille 640×480 à partir du coin sud-ouest en prenant la première couleur pour `true` et la deuxième pour `false`.

Question 6. Programmez une fonction `fourmi` telle que `fourmi tab` dessine le trajet d'un tel agent, en partant du point $(320, 240)$ suivant la direction Nord, sur la matrice `tab`. La fonction s'arrête lorsque la fourmi a atteint l'un des bords. Pour mieux voir le trajet de la fourmi, on pourra dessiner la matrice de départ en noir et blanc et le trajet en bleu et jaune.

Question 7. Programmez une fonction `fourmi_tore` où l'agent peut se déplacer comme dans un tore. On pourra utiliser l'opérateur `mod`.

3 Le jeu de la vie

Le jeu de la vie est un automate cellulaire inventé par Conway en 1970.

On part d'une matrice 64×48 de booléens, représentant une grille remplie par des cellules. À chaque tour, une cellule survit si elle est entourée par 2 ou 3 autres cellules et meurt sinon. Si une case libre est entourée par exactement 3 cellules, alors une nouvelle cellule y nait.

Question 8. Écrivez une fonction `voisins` telle que `voisins m x y` contient le nombre de cellules vivantes parmi les 8 cases voisines de `m.(x).(y)`. On supposera qu'on n'est pas sur un bord.

La commande `fill_rect x y w h` permet de tracer un rectangle plein compris entre les points (x, y) et $(x + w, y + h)$.

Question 9. Écrivez une fonction `affiche_grille` qui affiche une grille 64×48 avec des carrés de côté `10`.

Question 10. Écrivez une fonction `jdlv` de type `bool vect vect -> unit`, qui affiche une matrice puis qui la modifie avec les règles du jeu de la vie. La boucle ne modifiera pas les cases du bord.

Question 11. Écrivez une fonction `voisins_tore` qui calcule le nombre de voisins en considérant que la grille est un tore. En déduire une fonction `jdlv_tore`.

4 Tris sur des listes

Question 12. Écrivez une fonction `liste_test` de type `int -> int list` qui crée une liste de taille donnée remplie d'entiers aléatoires, et une fonction `est_triee` qui détermine si une liste est triée dans l'ordre croissant.

Question 13. Programmez une fonction `insere` de type `'a -> 'a list -> 'a list` qui insère un élément à la bonne position dans une liste suposée triée et en déduire une fonction `tri_insert`.

Le principe du tri fusion est le suivant : on prend une liste à trier, qu'on découpe en deux listes. On trie récursivement ces deux listes puis on les fusionne. Lorsqu'on tombe sur des listes de longueur 0 ou 1, il n'y a plus besoin de les trier.

Question 14. Programmez une fonction récursive `separe` qui prend une liste et la découpe en un couple de listes.

Question 15. Programmez une fonction récursive `fusionne` qui fusionne deux listes supposées triées en une seule. En déduire une fonction `tri_fusion`.

Le pricipe du tri rapide est similaire au tri fusion. On part d'une liste `t::q`, on sépare `q` en `q1` qui contient les éléments de `q` plus petits ou égaux à `t` et `q2` qui contient les autres. On trie récursivement les deux listes, et la liste triée est la concaténation `(tri_rapide q1)@[t]@(tri_rapide q2)`.

Question 16. Programmez une fonction `compare` de type `int -> int list -> int list * int list` qui sépare une liste `q` suivant un élément `t`.

Question 17. Que dire de la vitesse d'exécution de chacun des algorithmes ? On fera des essais avec des listes de taille variant entre 0 et 4 000 000.

Question 18. Si vous avez le temps, programmez les tris sélection et à bulles avec des listes.

5 Calcul formel

Question 19. Écrivez une fonction `decompose` qui prend en argument un entier $n = p_1^{r_1} \cdots p_j^{r_j}$ et qui rend la liste des couples (p_i, r_i).

Question 20. Écrivez une fonction `dim` de type `'a vect vect -> float * float`, qui prend un matrice en argument et qui renvoie son nombre de lignes et son nombre de colonnes. Si l'argument n'est pas une matrice (pensez au triangle de Pascal), la fonction renvoie `(-1, -1)`.

Question 21. Écrivez des fonctions pour les opérations suivantes sur les matrices : addition, multiplication, transposée, trace, déterminant, inverse, rang. On utilisera le type `float vect vect`.