Création d'interfaces graphiques simples avec EZ-Draw

Projet Algorithmique L2 Informatique/Mathématiques St Charles

Enseignants : Antoine Bonnefoy et Sokol Koço

Le but de ce TP est de présenter les bases de la boite à outils EZ-Draw, qui permet de créer facilement des interfaces graphiques en C. Vous pouvez trouver plus d'informations, notamment la documentation officielle et divers tutoriels, sur cette page :

http://pageperso.lif.univ-mrs.fr/~edouard.thiel/ez-draw/

1 Prise en main de EZ-Draw

1.1 Création d'une fenêtre

Dans cette partie nous allons regarder un cas très simple de l'utilisation d'EZ-Draw, celui de la création d'un fenêtre vide. Le code minimal permettant d'avoir une fenêtre est le suivant [1].

```
1  #include "ez-draw.h"
2
3  int main ()
4  {
5    if (ez_init() < 0) exit(1);
6
7    ez_window_create (400, 300, "Demo 01 : Hello World", NULL);
8
9    ez_main_loop ();
10   exit(0);
11 }
```

Programme 1 – fenetre.c

La première ligne permet d'inclure la bibliothèque EZ-Draw, dont les types et les prototypes des fonctions sont définis dans le fichier `ez-draw.h`. Le role de la fonction `ez_init()` est d'initialiser le module et le mode graphique, et donc, de mettre en place les différents mécanismes de gestion de l'interface graphique et des interactions. Il est donc nécessaire de vérifier que tout s'est bien passé avant de continuer, d'où la condition de la ligne 5.

La création d'une fenêtre passe par la fonction `ez_window_create()`. Cette fonction prend quatre arguments :

1. la largeur de la fenêtre (entier) ;
2. la hauteur de la fenêtre (entier) ;
3. le titre de la fenêtre (chaine de caractères) ;
4. fonction de gestion des événements (nom d'une fonction).

Finalement, la boucle principale `ez_main_loop()` gère le tout : affichage des fenêtres, détection des événements, etc.

Pour compiler le programme `fenetre.c`, il suffit de lancer la commande suivante sous un système Unix :

1. Ce code correspond au premier exemple dans le tutoriel officiel : http://pageperso.lif.univ-mrs.fr/~edouard.thiel/ez-draw/ez-tutorial.html#sec-premier-prog.

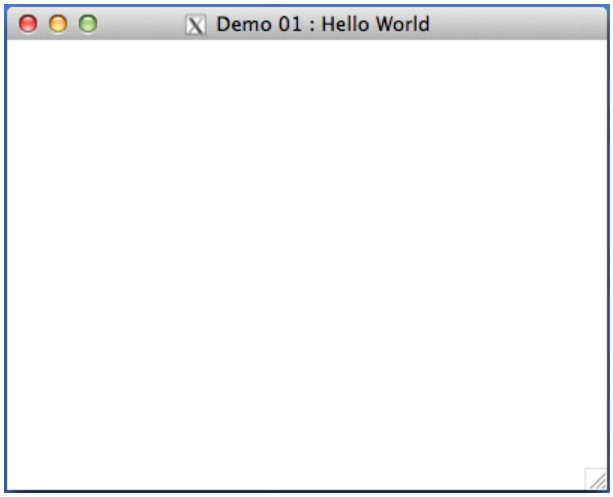

FIGURE 1 – fenetre.c : création d'une fenêtre basique avec EZ-Draw.

```
gcc -Wall fenetre.c ez-draw.c -o fenetre -lX11 -lXext -L/usr/X11R6/lib
```

et sous Windows :

```
gcc -Wall fenetre.c ez-draw.c -o fenetre.exe -lgdi32
```

N.B. Dans le cas des systèmes Mac OS X, il faut aussi indiquer le chemin vers les bibliothèques `X11`. Pour une installation standard, cela revient à ajouter `-I/opt/X11/include` dans les options de `gcc`.

1.2 Dessiner des points

Avoir des fenêtres, c'est bien, mais avoir des fenêtres avec du contenu, c'est encore mieux. Dans ce document, nous allons nous limiter à deux types très simples de contenus/dessins : les points et les lignes. Une liste plus exhaustive de ces types est disponible à l'adresse suivante :

http://pageperso.lif.univ-mrs.fr/~edouard.thiel/ez-draw/ez-tutorial.html#sec-ref-dessins

Le code donné ci-dessous permet de dessiner plusieurs points rouges alignés horizontalement, comme montré dans la figure 2.

```
#include "ez-draw.h"

void win1_on_expose(Ez_event *ev)
{
  int id, distance;
  distance = 40; /* distance entre les centres des points*/

  ez_set_color(ez_red);

  for (id=1; id<11; id++){
    ez_set_thick(id);
    ez_draw_point(ev->win, 10+distance*(id-1), 150);
  }
}

void win1_event(Ez_event *ev)
{
  switch (ev->type) {
    case Expose : win1_on_expose(ev); break;
  }
```

```c
}

int main ()
{
  if (ez_init() < 0) exit(1);

  ez_window_create (400, 300, "Demo 02 : Dessiner des points", win1_event);

  ez_main_loop ();
  exit(0);
}
```

Programme 2 – points.c

D'abord, remarquez que dans le `main()`, le quatrième argument de la fonction `ez_window_create()` est le nom d'une fonction, dans ce cas `win1_event()`. Le role de la fonction `win1_event()` est de gérer les différents événements liés à une fenêtre, tels que le fait de redessiner le contenu de la fenêtre, les clicks, les frappes au clavier, etc. Très souvent, le corps de cette fonction est réduit à un grand `switch` permettant de distinguer entre les différents événements et d'agir en conséquence. Dans le code ci-dessus, le seul événement géré est `Expose`, qui est déclenché à chaque fois que la fenêtre doit être redessinée (par exemple, lors de la première fois qu'elle apparaît, lors de animations *cf.* section 1.3, etc.). L'action associée au déclenchement de cet événement est l'execution de la fonction `win1_on_expose()`.

`win1_on_expose()` dessine itérativement 10 points rouges, dont la distance entre les centres est de 40 pixels. La fonction `ez_set_color()` permet de définir la couleur utilisée pour les dessins et elle prend en argument le nom d'une couleur. Différentes couleurs sont définies par défaut (ez_black, ez_white, ez_grey, ez_red, ez_green, ez_blue, ez_yellow, ez_cyan, ez_magenta), mais il y a aussi la possibilité d'en définir des nouvelles (*cf.* la documentation de EZ-Draw). Pour définir l'épaisseur d'un type de dessin (points, ligne, rectangle, etc.) il faut utiliser la fonction `ez_set_thick()`, dont le seul argument est le nombre de pixels. Dans la fonction `win1_on_expose()`, l'épaisseur change itérativement de 1 pixel à 10 pixels.

Finalement, pour dessiner un point, il faut utiliser la fonction `ez_draw_point()` (pour une ligne : `ez_draw_line()`). `ez_draw_point()` prend en argument les deux coordonnées du point, tandis que la fonction `ez_draw_line()` prend en argument les deux coordonnées du point de debut et celles du point de fin (donc, 4 arguments en total).

Attention : dans EZ-Draw, l'origine (le point $(0, 0)$) est située en haut à gauche et les coordonnées des points sont des entiers. Ainsi, soit w la largeur de la fenêtre et h sa hauteur, alors le coin en haut à droite a pour coordonnées $(w, 0)$, celles de celui en bas à gauche sont $(0, h)$ et le dernier coin (en bas à droite) a les coordonnées (w, h).

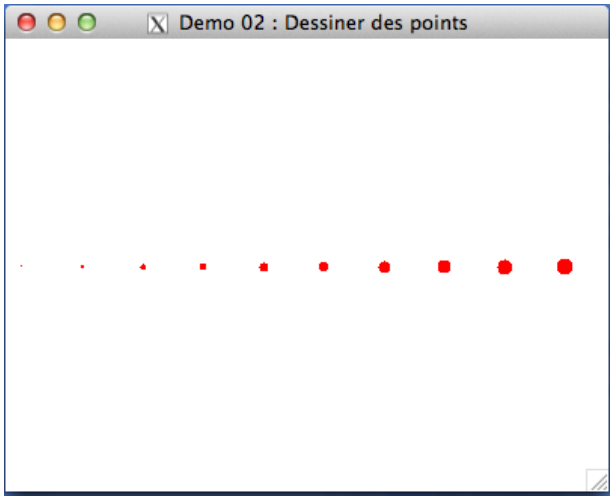

FIGURE 2 – points.c : dessin de plusieurs points alignés horizontalement.

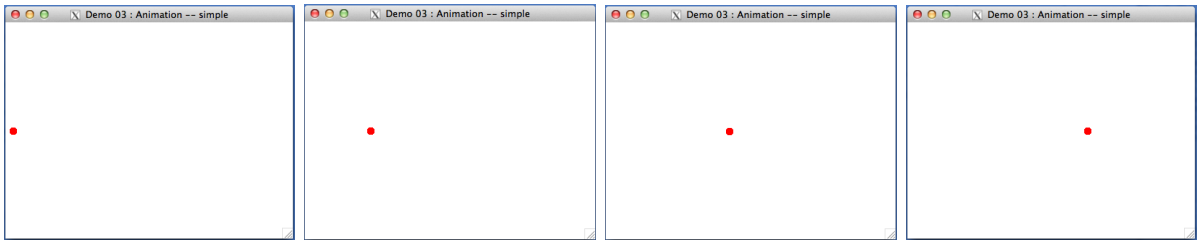

FIGURE 3 – Exemple d'une animation : de gauche à droite, les images aux instants 1, 2, 3 et 4.

1.3 Les animations

Une animation consiste en une séquence d'images qui se succedent les unes après les autres. Dans la section précédente, nous avons vu que le contenu d'une fenêtre en EZ-Draw peut être considéré comme une surface de dessin. Il en suit que pour réaliser une animation en EZ-Draw, il suffit de redessiner le contenu d'une fenêtre à chaque fois qu'il y a un changement de la position des objets (les points, lignes, rectangles, etc.). La figure 3 montre un exemple d'animation d'un point (rouge) qui se déplace de la gauche vers la droite. Le contenu de la fenêtre est redessiné à chaque fois que les coordonnées du point changent, donnant ainsi l'illusion du déplacement. Par exemple, le changement des coordonnées intervient après un certain laps de temps[2]. L'animation passe donc par les étapes suivantes :

1. initialiser un timer à une certaine valeur (positive) et decrémenter le timer jusqu'à 0 ;
2. mettre à jour les objets à afficher (par exemple, les coordonnées du point dans fig. 3) ;
3. dessiner la scène et revenir à l'étape 1.

Dans EZ-Draw, l'animation est essentiellement gérée par la réception et l'émission d'événements. Nous avons vu que l'événement `Expose` est utilisé pour dessiner le contenu de la fenêtre (l'étape 3). Pour les deux premières étapes, il faut utiliser un timer et détecter l'émission de l'événement `TimerNotify`, qui est émis lorsque le timer arrive à 0. Le code suivant permet de réaliser l'animation présentée dans la figure 3. Comparé au code donné dans la Section 1.2, la différence consiste dans l'affichage des différents points l'un après l'autre, à l'aide d'un timer.

```
1  #include "ez-draw.h"
2  #include <stdlib.h>
3  #include <time.h>
4
5  int cpt1 = 0, delay1 = 30;
6
7  void win1_on_expose(Ez_event *ev)
8  {
9    ez_set_color(ez_red);
10   int id, distance;
11
12   distance = 40; /* distance entre les centres des points*/
13
14   id = ((cpt1%10)+1);
15   ez_set_thick(id);
16   ez_draw_point(ev->win, 10+distance*(id-1), 150);
17 }
18
19 void win1_on_timer_notify (Ez_event *ev)   /* Le timer est arrive' a
       echance */
20 {
```

2. Dans le cas de l'animation du contenu des fenêtres, l'affichage n'est pas fait à chaque changement, mais bien après un laps de temps fixé et il tient en compte tous les changements arrivés pendant cette periode. Cela permet de réduire d'impact de l'affichage — qui est une opération très couteuse — sur les ressources du système.

```
21    /* On incremente le compteur cpt1 pour modifier la position de l'objet
22     que l'on veut animer */
23    cpt1 = (cpt1 + 1);
24    /* On envoie l'evenement Expose pour que la fenetre soit redessinee */
25    ez_send_expose (ev->win);
26    /* On re'arme le timer, pour entretenir une "boucle" de TimerNotify
27     qui est iteree tous les delay1 millisecondes */
28    ez_start_timer (ev->win, delay1);
29  }
30
31  void win1_event(Ez_event *ev)
32  {
33    switch (ev->type) {
34      case Expose : win1_on_expose(ev); break;
35      case TimerNotify : win1_on_timer_notify(ev); break;
36    }
37  }
38
39  int main ()
40  {
41    Ez_window win1;
42
43    if (ez_init() < 0) exit(1);
44
45    win1 = ez_window_create (400, 300, "Demo 03 : Animation -- simple",
       win1_event);
46
47      /* On associe un double-buffer d'affichage pour eviter tout
       clignotement */
48    ez_window_dbuf (win1, 1);
49
50    /* Provoque l'envoie d'un evenement TimerNotify dans delay1 millisecondes
         :
51     c'est le point de depart de la "boucle" de TimerNotify */
52    ez_start_timer (win1, delay1);
53
54    ez_main_loop ();
55    exit(0);
56  }
```

Programme 3 – animation.c

Les variables globales `cpt1` et `delay1` permettent de gérer l'animation : `delay1` définit le temps d'attente entre deux affichages (et donc le temps entre deux déclenchements successifs de l'événement `TimerNotify`), tandis que `cpt1` est un compteur incrémenté à chaque affichage, dont la valuer définit la taille et la position du point à afficher (*cf.* la fonction `win1_on_expose()`).

Dans la fonction `main()`, la fenêtre est gardée dans une variable, ce qui permet de lui associer un timer à l'aide de la fonction `ez_start_timer()`. C'est cette fonction qui émet l'événement `TimerNotify` une fois `delay1` millisecondes se sont écoulées. Comme précédemment, les événements sont gérés par la fonction `win1_event()`, et dans ce cas, elle contient aussi `TimerNotify`, qui est associé à la fonction `win1_on_timer_notify()`. Le role de la fonction `win1_on_timer_notify()` est de mettre à jour la valeur du compteur `cpt1` (ligne 23), déclencher l'événement `Expose` (ligne 25) et redémarrer le timer (ligne 28).

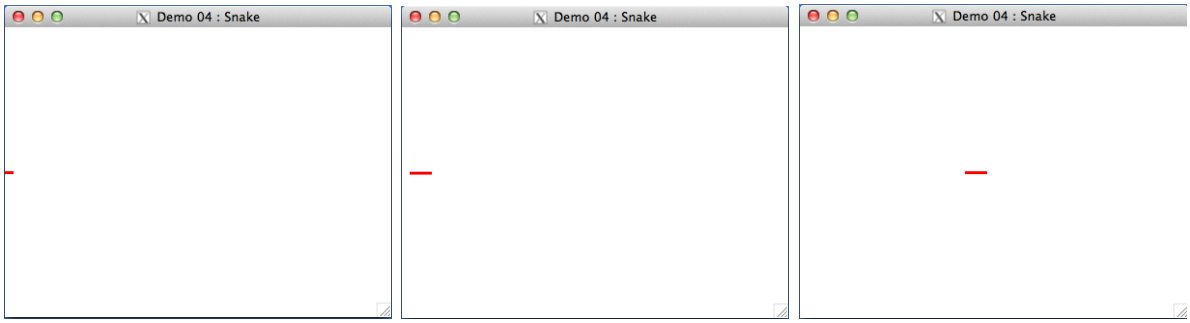

FIGURE 4 – Un serpent qui se déplace de gauche à droite.

2 Travail à réaliser ...

2.1 ... pour ce TP

En s'inspirant des exemples donnés dans la première partie, écrire un programme représentant un serpent qui se déplace de gauche à droite :

- on commence par un simple point, représentant le serpent en repos ;
- ensuite, le serpent commence à se déplacer (sa taille augmente) ;
- une fois sa taille maximale atteinte, il continue à se déplacer vers la droite ;
- on suppose que le serpent vit dans un tore : si il arrive toute à droite, il réapparaît à nouveau à gauche.

Améliorer le serpent précédent de sorte qu'il puisse tourner aléatoirement à gauche et à droite.

2.2 ... pour le projet

Dans la seconde partie du projet, le but est de créer une interface graphique permettant de représenter une partie de chasse entre les chats et la souris. Pour cela, vous pouvez vous inspirer de l'interface précédente sur le serpent pour afficher le déplacement des différents acteurs. Un exemple du résultat final est donné dans la figure 5, où les chats sont représentés par des points magenta et leurs trajectoires par des lignes vertes, tandis que la souris est le point cyan et la trajectoire rouge. Vous êtes libre de proposer et d'utiliser d'autres interfaces et représentations du jeu, sous condition qu'elles permettent d'identifier les différents acteurs et de visualiser le déroulement de la partie. Des ameliorations possibles pour l'interface :

- une fenêtre secondaire contenant des informations diverses ;
- afficher le nombre de fois qu'un chat arrive à attraper la souris ;
- afficher les directions les plus propices pour chaque chat ;
- la possibilité de mettre le jeu en pause et d'avancer pas à pas ;
- etc.

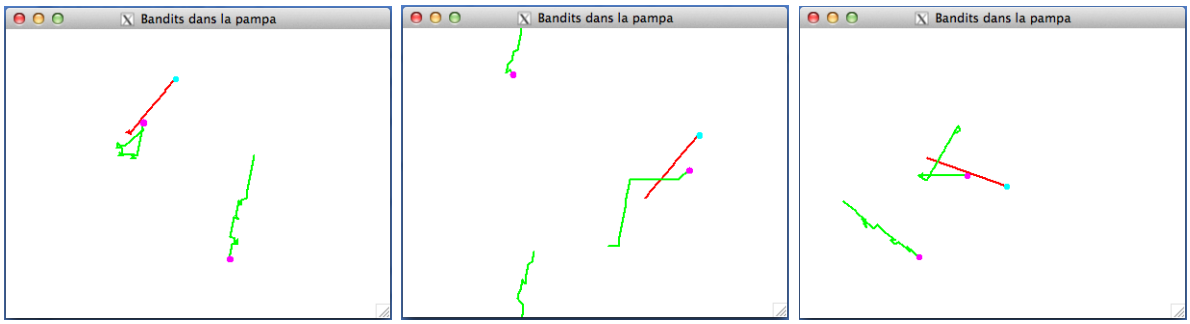

FIGURE 5 – Les bandits dans la pampa : une simulation du jeu de chat et souris.