Université Paris-Nord
Institut Galilée

Bibliothèque graphique TableauNoir

La librairie `libTableauNoir` permet d'utiliser facilement une fenêtre graphique, et fournit des fonctions permettant d'y dessiner, ainsi que des fonctions permettant de connaître la position de la souris, et l'état du clavier (la liste des touches enfoncées), à un instant donné.

Résumé des fonctions fournies

Le rôle des fonctions et leurs paramètres sont décrits plus en détail dans le fichier `libTableauNoir.h`. Le type `EtatSourisClavier` y est également défini et décrit. Il est donc primordial de regarder ce fichier avant d'utiliser les fonctions de la librairie.

A appeler impérativement avant toutes les autres

```
void creerTableau();
```

Cette fonction crée la fenêtre graphique et initialise la bibliothèque. Si une autre fonction de la bibliothèque est appelée avant `creerTableau`, le programme a toutes les chances de planter avec une *erreur de segmentation*. Il est conseillé d'appeler *creerTableau()* au tout début de la fonction *main*.

NB : Lorsque cette fonction est appelée, la fenêtre graphique (le tableau noir) s'affiche à l'écran. Lorsque le programme est terminé, la fenêtre disparaît aussitôt.

Fonctions pour agir sur la fenêtre graphique

```
void fixerTaille(int largeur, int hauteur);
void tableauPleinEcran();
void choisirCouleurFond( int rouge, int vert, int bleu );
void tableauRedimensionnable( int r );
void fermerTableau() ;
```

Notez que le choix de la couleur de fond ne prend pas effet immédiatement : il faut appeler la fonction `effacerTableau` (listée plus bas) pour en voir le résultat.

Fonctions qui attendent une action de l'utilisateur

Lorsqu'un programme se termine, la fenêtre graphique disparaît immédiatement. Pour éviter qu'elle ne disparaisse, il faut éviter que le programme ne se termine ! Pour cela, la bibliothèque fournit des fonctions pour *figer* le programme en état d'attente :

```
void attendreClicGauche(void);
void attendreClicDroit(void);
void attendreClicMilieu();
char attendreTouche(void);
void attendreNms(int ms);
void attendreFermeture() ;
```

Ces fonctions mettent le programme en pause, jusqu'à ce que l'utilisateur effectue une action donnée (clic souris ou appui sur une touche).

NB1 : `attendreTouche` renvoie le caractère correspondant à la touche enfoncée par l'utilisateur.

NB2 : `attendreNms` n'attend pas l'action de l'utilisateur, mais met le programme en pause pendant le temps passé en paramètre, donné en millisecondes.

NB3 : `attendreFermeture` attend que le gestionnaire de fenêtres demande la fermeture de la fenêtre, ce qui est souvent provoqué par un clic souris sur le bouton de fermeture de la fenêtre, ou par l'emploi de la combinaison de touches `Alt-F4` (selon les systèmes).

Fonctions de dessin

```
void effacerTableau();
void tracerPoint(int x, int y);
void tracerSegment(int x1, int y1, int x2, int y2);
void tracerRectangle(int x1, int y1, int x2, int y2);
void effacerPoint(int x, int y);
void effacerSegment(int x1, int y1, int x2, int y2);
void effacerRectangle(int x1, int y1, int x2, int y2);
```

Les paramètres à passer à ces fonctions sont faciles à comprendre ! Il faut cependant connaître le système de coordonnées : l'origine (0,0) se trouve au centre de la fenêtre ; (10,20) désigne le point se trouvant 10 pixels à droite de l'origine, et 20 pixels **au-dessus**.

Choix des styles et des couleurs

```
void choisirTypeStylo(int taille, int rouge, int vert, int bleu);
void choisirCouleurPinceau(int rouge, int vert, int bleu);
void choisirCouleurFond(int rouge, int vert, int bleu);
void fixerOpacite( int opacite );
```

Ces fonctions sont à appeler pour *changer* les caractéristiques des tracés ultérieurs, pour choisir les couleurs et la taille du *stylo*.

NB1 : Tracer un point avec une taille de stylo 10 provoque l'affichage d'un disque de 10 pixels de diamètre.

NB2 : Une couleur est codée selon ses 3 composantes : rouge, verte et bleue. Ces composantes doivent être de type entier, comprises entre 0 et 255.

NB3 : Le pinceau désigne la couleur de la surface d'un rectangle. Le stylo est utilisé pour les points, les segments, et le tour des rectangles. Le fond désigne la couleur de la fenêtre graphique.

NB4 : Le stylo initial est rouge et fait une taille de 1 pixel.

NB5 : `fixerOpacite` attend une valeur comprise entre 0 (pour un tracé transparent, donc pas de tracé du tout !) et 255 (tracé complètement opaque).

NB6 : Il est déconseillé d'appeler la fonction `choisirTypeStylo`. Elle n'est à appeler que quand on veut *changer* les caractéristiques des prochains tracés ; en effet, cette fonction prend beaucoup de temps à s'exécuter.

Sauvegarde de stylos

Comme créer un stylo avec *choisirTypeStylo* prend beaucoup de temps, il est préférable de demander à la librairie d'enregistrer les caractéristiques des stylos lorsque l'on en change souvent. Le changement de stylo se fait ensuite avec la fonction *selectionnerStylo*, qui, elle, est très rapide. Un stylo sauvé en mémoire doit être recyclé lorsqu'il n'est plus utile.

```
tStylo stockerStylo() ;
void selectionnerStylo( tStylo stylo );
void recyclerStylo( tStylo stylo );
```

La fonction `stockerStylo` enregistre les caractéristiques du stylo courant, choisies par le précédent appel à `choisirTypeStylo`. Elle retourne une valeur de type `tStylo`, qui est l'adresse en mémoire des caractéristiques sauvegardées.

La fonction `selectionnerStylo` permet de choisir un stylo en stock pour les tracés ultérieurs.

La fonction `recyclerStylo` permet de libérer la mémoire occupée par un stylo en stock dont on n'aura plus besoin.

Exemple : Imaginons que l'on veuille créer une animation de deux disques, l'un bleu, l'autre rouge, de 50 pixels de diamètre chacun. Voici une méthode lente :

```
double x1=0, y1=0, x2=0, y2=0;
while(1) {
  x1++; y1++; x2 ++; y2--;
  choisirTypeStylo(50, 0, 0, 255);  tracerPoint(x1,y1);
  choisirTypeStylo(50, 255, 0, 0);  tracerPoint(x2,y2);
  attendreNms(10);
  effacerPoint(x1,y1); effacerPoint(x2,y2);
}
```

Et voici une méthode beaucoup plus rapide à l'exécution :

```
double x1=0, y1=0, x2=0, y2=0;

tStylo s1, s2;
choisirTypeStylo(50, 0, 0, 255); s1 = stockerStylo();
choisirTypeStylo(50, 255, 0, 0); s2 = stockerStylo();

while(1) {
  x1++; y1++; x2 ++; y2--;
  selectionnerStylo( s1 );  tracerPoint(x1,y1);
  selectionnerStylo( s2 );  tracerPoint(x2,y2);
  attendreNms(10);
  effacerPoint(x1,y1); effacerPoint(x2,y2);
}
```

Fonctions pour consulter l'état de l'interface

```
EtatSourisClavier lireEtatSourisClavier(void);
void  lirePixel(int x, int y, int * rouge, int * vert, int * bleu);
int fermetureDemandee();
```

NB1 : `lireEtatSourisClavier` retourne une valeur de type `EtatSourisClavier`. C'est un type structuré, défini ainsi :

```
typedef struct {
    int sourisBoutonGauche;      /* vaut 1 si le bouton gauche est enfoncé, 0 sinon */
    int sourisBoutonDroit;       /* Idem pour le bouton droit */
    int sourisBoutonMilieu;      /* Idem pour le bouton du milieu (roulette) */
    int sourisX;                 /* Position du pointeur de la souris en X. */
    int sourisY;                 /* Position du pointeur de la souris en Y. */

    char touchesClavier[500];    /* Tableau des touches enfoncées (la case
                                    d'indice i contien 1 si la touche d'indice i
                                    est enfoncée, sinon elle contient 0)
                                    L'indice d'une touche correspond au code
                                    ASCII de la lettre minuscule inscrite
                                    dessus (pour les lettres, bien sur).
                                    Tableau de 500 cases.*/
} EtatSourisClavier;
```

NB2 : `lirePixel` permet de lire la couleur courante d'un pixel donné. La couleur est codée selon ses composantes rouge, verte et bleue. Il faut donc lui passer les coordonnées (x, y) d'un pixel, ainsi que les adresses de trois variables entières ; `lirePixel` modifie le contenu de ces trois variables selon la couleur actuelle du pixel (x, y).

NB3 : La fonction `fermetureDemandee` retourne 1 si l'utilisateur a demandé la fermeture de la fenêtre en cliquant sur l'icône de fermeture ou en employant la combinaison de touches *Alt-F4*. La fonction renvoie 0 tant que ce n'est pas le cas. Cette fonction est utile si l'on veut provoquer l'arrêt du programme quand l'utilisateur le demande. Par défaut, en effet, l'action de l'utilisateur ne provoque ni la fin du programme, ni même la disparition de la fenêtre graphique.

Fonctions pour l'affichage d'images prédéfinies

```
Image chargerImage( char * nom_fichier );
void fixerCouleurDeTransparence( Image image, int rouge, int vert, int bleu );
void afficherImage( Image image, int x, int y );
void libererImage( Image image ) ;
```

Ces fonctions utilisent le type `Image` défini dans la librairie pour stocker en mémoire les images chargées depuis des fichiers. Elles fonctionnent avec certains types de fichiers d'image mais pas tous. Elles ne permettent pas de redimensionner les images, qui seront donc affichées selon leurs dimensions originales.

Fonctions pour la bufferisation (fluidifier des animations)

```
void fixerModeBufferisation( int unOuZero );
void tamponner();
```

Ces deux fonctions servent surtout pour réaliser des animations. Une animation est le plus souvent obtenue en effaçant un objet à l'écran puis en l'affichant à nouveau à une position légèrement décalée ; mais chaque opération prenant un peu de temps, l'œil perçoit un objet clignotant. La bufferisation consiste à utiliser deux images : l'une est celle que l'on voit à l'écran dans la fenêtre graphique, l'autre est *virtuelle* et reste cachée en mémoire. En « mode bufferisation », toutes les fonctions de tracé de *libTableauNoir* modifient uniquement l'image virtuelle, sans toucher à l'image visible. Lorsque l'image virtuelle est « prête » à être affichée, le programme doit appeler la fonction `tamponner` pour la transférer vers l'écran.

Exemple avec l'animation des deux disques : le code suivant affichera la même animation, mais sans clignotement gênant.

```
double x1=0, y1=0, x2=0, y2=0 ;
tStylo s1, s2 ;
choisirTypeStylo(50, 0, 0, 255) ; s1 = stockerStylo() ;
choisirTypeStylo(50, 255, 0, 0) ; s2 = stockerStylo() ;

fixerModeBufferisation( 1 ) ;

while(1) {
  x1++ ; y1++ ; x2 ++ ; y2-- ;
  selectionnerStylo( s1 ); tracerPoint(x1,y1) ;
  selectionnerStylo( s2 ); tracerPoint(x2,y2) ;

  tamponner() ;

  attendreNms(10) ;
  effacerPoint(x1,y1) ; effacerPoint(x2,y2) ;
}
```

Gestion du temps

```
float delta_temps();
```

La première fois qu'on l'appelle, cette fonction retourne 0. A chaque appel suivant, elle retourne le temps écoulé depuis le dernier appel, en secondes.

Cette fonction est principalement utile pour contrôler précisément les vitesses de déplacement des objets dans les animations. Si l'on regarde le code précédent, on voit que l'instruction *attendreNms(10)* est utilisée entre deux affichages, de manière à ce que les deux disques aient des vitesses horizontale et verticale de 100 pixels par secondes (car on attend $0,01$ seconde avant d'incrémenter les coordonnées en x et en y). Mais cela n'a rien de précis : il se peut que la machine soit chargée par ailleurs, que votre programme passe du temps à calculer les images, que l'écran se fige, et qu'au final il y ait plus de $0,01$ seconde entre deux images ; l'impression donnée à l'utilisateur est une animation trop lente, ou irrégulière.

Pour corriger cet effet, on peut transformer le code précédent en utilisant la fonction *delta_temps* ainsi :

```
double vitesse = 100; /* vitesse de 100 pixels/seconde
                         horizontalement et verticalement */
double x1=0, y1=0, x2=0, y2=0;
tStylo s1, s2;
choisirTypeStylo(50, 0, 0, 255); s1 = stockerStylo();
choisirTypeStylo(50, 255, 0, 0); s2 = stockerStylo();

fixerModeBufferisation( 1 );

while(1) {
  float dt = delta_temps();
  x1 = x1 + vitesse * dt;
  y1 = y1 + vitesse * dt;
  x2 = x2 + vitesse * dt;
  y2 = y2 - vitesse * dt;

  selectionnerStylo( s1 ); tracerPoint(x1,y1);
  selectionnerStylo( s2 ); tracerPoint(x2,y2);

  tamponner();

  effacerPoint(x1,y1); effacerPoint(x2,y2);
}
```