WINDOWTOOLS: introduction

Une extension graphique pour SIMULA

Auteur: J.Vaucher
Version: 3.1
Date: 6 octobre 1997

Introduction

Windowtools est une librairie graphique initiallement conçu par le professeur Kirkerud pour faciliter la création d'interface avec le standard X-windows. A Montreal, depuis 3 ans, nous avons réécrit une grande partie de ce logiciel afin de le rendre plus robuste et plus simple à employer.

Avec WindowTools, on peut dessiner des lignes, des ellipses et des rectangles avec diverses teintes de gris (ou de couleurs - a partir d'automne 97) et écrire du texte n'importe où sur une page. Il y a aussi des dialogues standards afficher des messages et pour demander des nombres et du texte. A peu près les mêmes possibilités qu'avec un BASIC ou Pascal pour ordinateur personnel. Avec un programme un peu plus compliqué, votre programme pourra attendre et réagir à divers événements; en particulier, vous pouvez specifier des procedures qui seront applées automatiquement suite à un **Click** de souris ou la **sélection** d'un Bouton.

En 1996-97, nos étudiants avaient accès à Windowtools avec une interface simplifiée appelée DESSIN; mais cet été (1997) nous avons intégré les bonnes idées de DESSIN directement dans WindowTools. Une conséquence de ceci est que Dessin n'est plus supporté mais il est relativement facile de convertir des vieux programmes [Voir section Conversion]. Nous avons aussi ajouté au package le traitement de la couleur et des images graphiques (.gif, .jpeg...etc...).

Le présent document présente un sous-ensemble des fonctions graphiques de WindowTools qui sont utiles pour IFT1010 et IFT1020. Pour plus de détails sur le package au complet, vous pouvez consulter:

- Le document de Jocelyn Houle.
- La documentation de C. de Lean sur la version originale d'Oslo.

Structure et compilation d'un programme

Programme simple

Un programme simple qui ne fait qu'afficher des graphiques aura la structure suivante:

```
BEGIN
 external class windowtools;

     inspect new FirstWindow("ift1010") do
     begin
         <...votre code ..>
     end


END
```

Exemples

- allo1.sim
- sinus.sim
- bounce.sim
- fig.sim

Application Interactive

Une application graphique interactive qui doit réagir à divers événements comme des CLICKS ou l'entrée de chaque caractère aura une strucutre légèrement différente. En particulier, l'application doit être un OBJET distinct (spécialisation de MainWindow) créé avec NEW, par exemple:

```
BEGIN
    EXTERNAL class  WINDOWTOOLS;

    MainWindow class exemple;
    begin
        ... votre code...
          ...avec procedure de captage d'evenements
    end;

    NEW exemple( "Titre..." );
END
```

Exemples

- keys.sim
- draw.sim

Compilation

Pour compiler l'un ou l'autre de ces programmes (disons qu'il porte le nom `tp.sim`) il faut invoquer 'sim' avec l'option *simwin* :

```
%  sim -simwin tp
```

Un premier programme

```
% ------------------------------------------
%    allo1.sim
```

```
%    =========
%    Programme qui affiche Hello dans une fenetre
% ----------------------------------------
begin
   external class windowtools;

   Inspect new FirstWindow("*** allo1 ***") do
   begin
       CenteredText(width/2, Height/2,"Hello World");
       Sleep(4);
   end;

end;
```

Un programme plus complexe est donné plus loin.

Généralités

Type de fenêtres

Chaque programme WindowTools doit avoir exactement UNE fenêtre principale (soit un FIRSTWINDOW ou un MAINWINDOW). En plus, on pourra creer des fenêtres auxiliaires (SIDEWINDOW) en nombre quelconque. A l'intérieur de ces fenetre de premier niveau, vous pourrez ajouter d'autres éléments graphiques ou Widgets. Au niveau de X, ces widgets sont qui sont des fenêtre dépendantes, sous-classes de SUBWINDOW.

Taille des fenêtres:

Une fenêtre **FirstWindow** reçoit automatiquement une fenêtre graphique de 600 x 400 pixels et les routines graphiques ci-dessous (comme ligne ou rectangle) dessinent dans cette fenêtre. Pour les **MainWindow** utilisˇes pour les applications avancˇes, il faut spˇcifier une taille avec SetSize (voir plus loin) et aussi les faire afficher explicitement (avec Show).

Il est aussi possible de créer plusieurs fenêtres avec **MakeSideWindow(titre)**.

Coordonnées:

Les coordonnées sont exprimées en PIXELS. L'origine (le point <0,0>) est située au coin supérieur gauche des fenêtres; X augmente en allant vers la droite et Y augmente en allant vers le bas. Ceci est conforme aux conventions de X et du Mac.

Procédures générales

integer procedure **Height**;
    Donne la hauteur de la fenêtre en pixels.

integer procedure **Width**;
    Donne la largeur de la fenêtre en pixels.

procedure **SetSize(X,Y)**; integer X,Y;
    Change la taille de la fenetre.

procedure **Hide**;
    Cache la fenetre.

procedure **Show**;
    Affiche la fenetre.

procedure **PlaceAt(X,Y)**; integer X,Y;

Positionne la fenetre au coordonnees <X,Y>.

procedure **Sleep** (n) ; real n ;

Suspend l'opération du programme pour n secondes, par exemple pour vous permettre d'admirer vos graphiques.

Routines graphiques

Résumé

- SetLineWidth
- DrawLine
- DrawPolygon

- DrawPoint
- DrawRectangle
- FillRectangle
- ClearRectangle

- DrawEllipse
- DrawCircle
- FillEllipse
- FillCircle

- Clear
- Invert
- Flash

---

procedure **SetLineWidth**(n); integer n;

Sert a changer la taille des lignes. Ce parametre gouverne aussi l'epaisseur des lignes utilisees pour les rectangles, cercles et autres figures geometriques. Initialement, la largeur des lignes est de 1 pixel.

NOTE: si la largeur de ligne est un nombre **pair**, X centre ses traits sur la ligne theorique rejoignant deux points donnes. Si la largeur est un nombre **impair** (en particulier 1), les traits seront decentres d'un pixel et le pixel extra sera mis soit a droite soit en bas de la ligne theorique.

procedure **DrawLine**(x1, y1, x2, y2); integer x1, y1, x2, y2;

Trace une ligne de <x1,y1> à <x2,y2>

procedure **DrawPolygon**(x, y, n);

Dessine un polygone. X et Y sont des tableaux [integer array (1:n)] qui donnent les coordonnées des N vertex du polygone.

procedure **DrawPoint**(x, y); integer x, y;

Trace une POINT à <x,y>. Plus exactement, X et Y represente le point superieur gauche du POINT. Ceci veut dire qu'un point affiche a <0,0> sera tout juste visible mais qu'un point a <width,y> ou a <x,height> se trouve tout juste en dehors de la zone visible de la fenetre et sera donc invisible.

procedure **DrawRectangle**(x, y, w, h);
integer x, y, w, h ;

Dessine un rectangle W pixels de large par H pixels de haut avec le coin superieur gauche situé a <x,y> .

procedure **FillRectangle**(x, y, w, h);
integer x, y, w, h ;

Utilise la couleur courante pour emplir une zone rectangulaire W pixels de large par H pixels de haut avec le coin superieur gauche situé a <x,y>.

Pour dessiner un rectangle de couleur avec un cadre, il faut faire deux appels, par exemple:

```
FillRectangle(50,10, 100, 50);
DrawRectangle(50,10, 100, 50);
```

procedure **ClearRectangle**(X, Y, W, H); integer x, y, w, h ;

Efface le rectangle specifie. Il est possible d'effacer a gauche ou en dessus de X,Y en utilisant des valuers negatives pour W et H.

procedure **InvertRectangle**(X, Y, W, H);

Inverse les couleurs (NOIR <-> Blanc) du rectangle specifié.

procedure **DrawEllipse**(X, Y, W, H);

Trace une ellipse inscrite dans un rectangle specifie par < x, y, W, H >.

procedure **DrawCircle**(x, y, rayon);

Trace un cercle centré a <x,y>.

procedure **FillCircle**(x, y, rayon);
procedure **FillEllipse**(X, Y, W, H);

Ces deux procedures fonctionnent comme celles pour les rectangles.

procedure **Clear**;

Efface tout l'écran et vide la mémoire.

procedure **Invert** ;

Inverse les couleurs de la fenetre.

procedure **Flash** ;

Inverse momentanement les couleurs (utile pour attirer l'attention).

Couleurs:

Les couleurs sont spécifiées sous forme de chaines de caratères (comme "red" ou "Orchid"). Celle-ci sont converties en format RGB par l'entremise d'un dictionnaire d'équivalences, "/usr/lib/X11/rgb.txt", fournies par le système X. Il est aussi possible d'utiliser des teintes de gris - allant de "gray0" (noir) a "gray100" (blanc) - ainsi que ^"black" et "white" Notez que le dictionnaire de couleur est basé sur l'anglais. Si vous entrez un nom de couleur en francais (p.e. vert) ou si vous faites une erreur d'orthographe (p.e. grean), une couleur aléatoire sera utilisée.

On peut aussi définir la couleur que l'on veut à l'aide de codes RGB. Le format de ces couleurs est simple: le premier caractère doit être le caractère '#', puis on code chaque couleur avec 1 ou 2 caractères hexadécimaux (de 0 à F). Ainsi, bleu serait `#00F` ou `#0000FF`.

Sur un terminal noir et blanc, les couleurs (et les gris) sont rendues par diverses combinaisons de points noir et blancs.

procedure **SetBackGround** (couleur) ; text couleur ;

> Choisit une *couleur* qui sera utilisée pour le fond de la fenetre. Initiallement, cette couleur est "white".

procedure **SetFill**(C); text C;

> Choisit une *couleur* qui sera utilisée pour le remplissage des formes subséquentes. Initiallement, le 'fillcolor' est "white".

procedure **SetForeGround** (couleur) ; text couleur ;

> Au départ, les dessins se font avec de l'encre noire ("black") mais on peut effacer un dessin en changeant la couleur à "white" et en refaisant le même dessin....n'oubliez pas de revenir à l'encre noire après

---

Affichage de texte:

procedure **DrawText** (x,y,message) ; integer x,y ; text message;

> Affiche le message donné à a la droite du point <x,y>. La ligne de base du texte est alignee verticalement sur Y et les lettres peuvent aller jusqu'a *font_ascent* en dessus de la ligne de base et *font_descent* en dessous. ON peut afficher un message multi-lignes en mettant des caracteres de Fin_de_ligne - CR ('!10!') ou LF ('!13!') - dans le message.

procedure **EraseText** (x,y,message) ; integer x,y ; text message;

> Efface le texte.

procedure **CenteredText** (x,y,message) ; integer x,y ; text message;

> Affiche un message d'une ligne centré sur le point <x,y>.

integer procedure **width_of_text** (T) ; text T;

> Donne la largeur (en pixels) du texte T

procedure **EZSetFont** (NomPolice, Taille); Text NomPolice; integer taille;

> Change la 'fonte' courante. Les polices connues sont: `helvetica`(défaut) `courier` et `times`. Les tailles (hauteurs) permises sont: 8, 10, 12, 14, 18, 20, 24 et 36.

Les boutons:

Dans une APPLICATION, les boutons sont déclarés comme référence à des *button* avec:

```
ref(button) mon_bouton ;
```

Puis ils sont créés avec `MakeButton` en leur donnant un libellé:

```
mon_bouton   :- MakeButton("Clear");
```

Typiquement, on les clicks sur ces boutons seront captés par la procédure d'événement *Button_press* (voir plus loin); mais on peut aussi attendre avec:

```
mon_bouton.wait;
```

Voici quelques operations permises sur les BOUTONs. Pour plus d'information, voir la documentation de *WindowTools*.

Bouton.Wait;
Le programme attendra que le bouton soit cliqué.
Bouton.Hide;
Cache le bouton.
Bouton.Show;
Ré-affiche le bouton.
Bouton.SetHeading(T); text T;
Change le texte affiché par le Bouton.

Par défaut, les boutons se mettent vers le haut de la fenêtre. On peut aussi les mettre ailleurs avec:

- `Bouton`.PlaceAt(X,Y);
- `Bouton`.PlaceDownLeft;
- `Bouton`.PlaceBelow();
- etc....

Lecture de paramètres:

Les procedure suivantes demande une donnee a l'usager avec un PROMPT. Ces procedures font aussi la verification du type de ce qui est entre. Il y a une procedure pour chaque type de donnee: TEXT, INTEGER, CHAR, REAL et BOOL.

- text procedure **ask_for_text**(prompt); TEXT prompt;
- integer procedure **ask_for_int**(prompt);
- character procedure **ask_for_char**(prompt);
- real procedure **ask_for_real**(prompt);
- boolean procedure **ask_for_Bool**(prompt);

En cas d'erreur, on peut afficher une fenetre avec un message (warning) avec:

- procedure **message**(Texte); TEXT Texte;
- procedure **TimedMessage**(Texte, Delais); TEXT Texte; INTEGER Delais;

  Dans ce cas, le message este affiche pour environs Delais secondes.

Dialogues de selection de Fichiers:

Les fonctions suivantes affiche des dialogues pour permettre a un usager de selectionner des noms de fichiers (ou de repertoires) de façon interactive.

- text procedure **GetFile**(Prompt,Path); TEXT Prompt,Path;
- text procedure **GetNewFile**(Prompt,Path,DefName); TEXT Prompt,Path,DefName;
- text procedure **GetDirectory**(Prompt,Path); TEXT Prompt,Path;

Traitement interactif avancé

Une application graphique interactive est normallement composée de procédures de réaction à des événements. Par exemple: un CLICK de souris ou l'entrée d'un caractère ou le déplacement d'une fenêtre. Pour traiter ces événements, une application doit être structurée selon le deuxième schéma en dˇfinissant une sous-classe de la classe APPLICATION dans laquelle sont définies les procédures pour traiter les différents types d'événements. Ces procédures doivent porter des noms pré-établis. Elles sont:

`procedure` **`Handle_ButtonClick`** `(B); ref(button) B;`
: Cette procedure sera appelée à chaque fois qu'un bouton sera clické dans la fenetre de l'application. Le paramètre B sera un pointeur vers le bouton en question.

`procedure` **`Handle_Click`**`(X,Y); integer X,Y;`
: Cette procedure est appelée chaque fois que l'usager cliquera ailleurs que dans un bouton. X et Y sont les coordonnées de ce CLICK.

`procedure` **`WaitForClick`**`(X,Y); integer X,Y;`
: Cette procedure fera attendre le programme pour un Click ailleurs que dans un bouton. X et Y sont les coordonnées de ce CLICK.

`Boolean procedure` **`Clicked`**`(X,Y); integer X,Y;`
: Cette procedure retourne TRUE s'il y a eu un Click depuis le dernier traité. X et Y sont les coordonnées de ce CLICK.

`procedure` **`handle_key_down`**`(c);CHARACTER c;`
: Cette procedure est appelée chaque fois que l'usager appuie sur une touche qui représente un caractère imprimable.

`procedure` **`handle_left_arrow;`**
`procedure` **`handle_right_arrow;`**
`procedure` **`handle_up_arrow;`**
`procedure` **`handle_down_arrow;`**
: Ces procedures sont appelée quand l'usager appuie sur une des touches de FLECHE..... Tres utile pour les jeux interactifs....

`procedure` **`refresh`**`;`
: Cette procedure est appelée chaque fois que le contenu de la fenêtre a été effacé par une autre fenêtre placée par dessus.

Programme d'animation

Souvent, un programme interactif crée les fenêtres initiales puis se met en attente; le plus souvent en faisant quelque chose comme **"bouton".wait**. Quand vous vous mettez en attente, ceci permet au système graphique sousjacent (X-windows) de traiter les événements et pour chaque événement, X appelle la procédure appropriée (si elle a été définie par l'application).

Pour les débutants, ce genre de programme réactif semble étrange car le corps du programme ne fait rien: tout le travail est fait dans les procédures de traitement d'événements qui sont appelées "magiquement" en réaction à des actions graphiques.

Dans un programme qui fait de l'animation, il faut procéder autrement. Typiquement, le programme principal se trouve dans une boucle ou il déplace des icônes (en les effaçant et les redessinant ailleurs). Si votre programme ne fait jamais SLEEP et jamais WAIT, les événements ne sont pas traités. Dans ce cas, if faut explicitement permettre la vérification des événements en appelant une des dexu routines suivantes:

```
Handle_pending_events;
```

Cette routine traite les evenements (s'il y en a) et ne fait rien sinon. La routine suivante attend et traite un evenement:

```
Handle_event;
```

Rafraichissement

Par défaut, les dessins que vous affichez ne sont pas mémorisés. Si une autre fenêtre recouvre temporairement une de vos fenêtres graphiques, vous remarquerez que ce qui avait été dessiné n'apparait plus. Pour traiter ceci, il est bon de prévoir dans votre Application une procédure nommée REFRESH qui redessine le contenu de la fenêtre. Cette procédure sera appelée chaque fois qu'une partie de votre fenêtre a été effacée.

Exemple

Le programme ci-dessous (*draw.sim*) montre l'emploi de boutons et de routines de captage d'événements. Remarquez les deux façon d'attendre que l'usager Clique sur le bouton STOP.

```
% >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
%   draw.sim : ce programme exploite la souris et les boutons pour
%              des dessins simples
% >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

    BEGIN

        EXTERNAL class  WINDOWTOOLS;

        MainWindow class exemple;
        begin

            integer X0, Y0;
            ref(button) Stop_b, Clear_b, Inverse_b;
            Boolean     Finir;

            ! procedures pour capter Click et choix de boutons ;

            procedure Handle_Click(X,Y); integer X,Y;
            begin
                DrawLine( X0,Y0,X,Y);
                X0 := X; Y0 := Y;
            end;

            procedure Handle_ButtonClick(B); ref(button) B;
               if      B == clear_b   then Clear
               else if B == inverse_b then Flash
               else if B == Stop_b    then finir := true;


            SetSize(600,400); Show;
            clear_b   :- MakeButton("Clear");
            inverse_b :- MakeButton("Invert");
            Stop_b    :- MakeButton("Stop");
            X0 := width // 2;
            Y0 := height// 2;
            SetLineWidth(2);
            CenteredText(X0,50,"Clicker a divers endroits pour dessiner");

%  pour attendre la FIN,
%  -------------------------------------
            while not finir do handle_event;
%     ou  ---------------------------------
%           stop_b.WAIT;
%         ---------------------------------

        end;

        new exemple("DRAW1.SIM");

    END
```

Fenêtres multiples

Pour créer ces fenêtres, il faut utiliser la commande `MakeSideWindow` en spécifiant le nom (T) de la fenêtre.

```
ref(SideWindow) procedure MakeSideWindow(T); text T;
```

Les fenêtres devront être désignées par des ref(dessin). Toutes les procédures graphiques décrites plus haut s'appliquent à ces fenêtes, mais il faut utiliser la notation pointée. Par exemple, pour créer un nouveau dessin dans lequel on affichera un cercle, on fera:

```
ref(SideWindow) D2;

D2 :- MakeSideWindow("Autre dessin");
D2.cercle( 100,100, 50, s_vide);
```

Exemples

- allo1.sim: Affichage de "Hello World"
- sinus.sim: Affichage d'un graphique
- draw.sim: mini-programme de dessin qui traite directement les evenements
- ballon.sim Rebondissements sans fin
- fig.sim: Fenetres multiples
- keys.sim: Un programme qui traitent divers types d'événements: Touches du clavier et Clicks de souris.
- fontwindow.sim: Programme plus complexe qui montre diverse variantes de Fontes
- fontlist.sim: Programme qui montre toutes les polices disponibles sur le système.

Ces programmes se trouvent aussi dans ~dift1010/exemples/Dessin.

Passage de DESSIN a WINDOWTOOLS

Pour montrer comment passer de DESSIN a WINDOWTOOLS, voici des exemples qui illustrent les différences. En gros, il faut remplacer **DESSIN** par **WINDOWTOOLS** et **Application** par **MainWindow** ou **FirstWindow**:

```
EXTERNAL class  DESSIN;

Application class exemple;
begin
    ... votre code...
      ...avec procedure de captage d'evenements
end;
```

devient donc:

```
EXTERNAL class  WINDOWTOOLS;

MainWindow class exemple;
begin
    ... votre code...
      ...avec procedure de captage d'evenements
end;
```

ALLO1.SIM: affiche Hello dans une fenetre

Version DESSIN

```
begin
   external class dessin;

   Inspect new Application("*** allo1 ***") do
   begin

        Texte(width/2, Height/2,"Hello World");

        Pause(4);
   end;

end;
```

Version WindowTools

```
begin
   EXTERNAL CLASS WindowTools;

   Inspect new FirstWindow("*** allo1 ***") do
   begin

        CenteredText(width/2, Height/2,"Hello World");
        Sleep(4);
   end;

end;
```

DRAW.SIM

La version avec WindowTools a été présentée plus haut.

Version DESSIN

```
% >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
%   draw.sim : ce programme exploite la souris et les boutons pour
%              des dessins simples
% >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

    BEGIN

        EXTERNAL class  DESSIN;

        Application class exemple;
        begin

            integer X0, Y0;
            ref(button) Stop_b, Clear_b, Inverse_b;
            Boolean     Finir;

            ! procedures pour capter Click et choix de boutons ;

            procedure Mouse_click(X,Y); integer X,Y;
            begin
                ligne( X0,Y0,X,Y);
                X0 := X; Y0 := Y;
            end;

            procedure Button_press(B); ref(button) B;
               if      B == clear_b   then Effacer
               else if B == inverse_b then Flash
               else if B == Stop_b    then finir := true;


            clear_b   :- MakeButton("Clear");
            inverse_b :- MakeButton("Invert");
            Stop_b    :- MakeButton("Stop");
```

```
            X0 := width // 2;
            Y0 := height// 2;
            plume(2);
            texte(X0,50,"Clicker a divers endroits pour dessiner");

 %  pour attendre la FIN,
 %  ---------------------------------------
            while not finir do handle_event;
 %     ou  ---------------------------------
 %           stop_b.WAIT;
 %         ---------------------------------

         end;

         new exemple("DRAW1.SIM");

     END
```

V.