TP wxWidgets n°1

Objectif général

Le but des 5 prochains TP va être d'apprendre à créer et manipuler des interfaces graphiques, en créant un petit programme destiné à l'affichage et à la saisie de triangles. Pour cela, nous allons utiliser la librairie multi-plateforme wxWidgets (anciennement wxWindows), à titre d'exemple. Vous devrez programmer en C++. Tout au long de ces TP, vous pourrez utiliser la documentation présente sur le site web de wxWidgets :
http://docs.wxwidgets.org/stable/wx_classref.html#classref

Objectif de ce TP

Dans ce premier TP, nous allons tout d'abord voir ce que contient une application interactive de base avec cette librairie. Puis, nous allons introduire quelques éléments supplémentaires, comme une barre de menus, une barre d'outils, ainsi que quelques boîtes de dialogue.

Manipulations

1. Chargement d'un squelette d'application.
Afin de nous faciliter la tâche, nous allons partir d'une application de base. Copiez dans un répertoire perso les fichiers situés à l'emplacement suivant…
Répertoire : TPWX_fichiers/

- Le fichier mainframe.h contient la déclaration de la classe CMainFrame, qui comme son nom l'indique va servir de cadre principal à l'application, c'est-à-dire une fenêtre réduite pour l'instant à sa plus simple expression : un rectangle avec un titre et 3 boutons de manipulation classiques.
- Le fichier mainframe.cpp contient l'implantation du constructeur de cette fenêtre, qui pour l'instant ne fait rien du tout.
- Le fichier main.cpp contient la définition de classe ainsi que l'implantation de l'application elle-même, MyApp, qui pour l'instant se contente d'ouvrir la fenêtre principale et de l'afficher.

On remarque que les 2 classes CMainFrame et MyApp, dérivent toutes les 2 de classes de base de wxWidgets, respectivement wxFrame et wxApp, ce qui leur fournit automatiquement les fonctionnalités de base.

Compilez le programme, et exécutez-le. Vous devez obtenir la vue suivante :

2. Création du menu

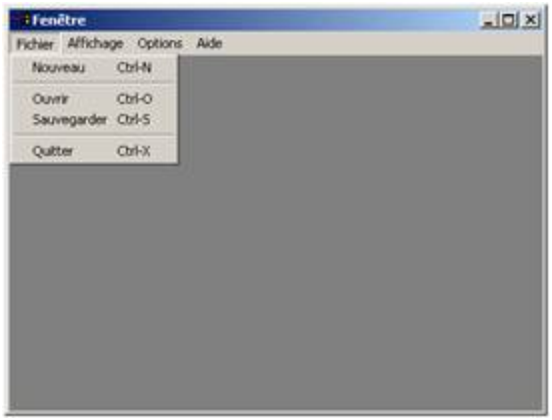
On va maintenant créer la barre de menu. Nous allons le faire dans la fonction OnInit de la classe MyApp.

- Commençons par déclarer un pointeur sur la barre de menu (de type prédéfini wxMenuBar) par l'instruction :
  ```
  wxMenuBar *menu_bar = new wxMenuBar;
  ```
- Ensuite déclarons les éléments de menu (type prédéfini wxMenu) par des instructions de type :
  ```
  wxMenu *file_menu = new wxMenu;
  ```
  Vous devez en définir en tout 4, pour "Fichier", "Affichage", "Options", et "Aide".
- Une fois que les éléments de menu sont déclarés, il faut les ajouter à la barre de menu. Pour cela, on utilise des instructions du type :
  ```
  menu_bar->Append(file_menu, wxT("Fichier"));
  ```
  Vous devez ajouter les 4 menus précédemment déclarés à la barre de menu.
- Nous allons maintenant ajouter des rubriques aux différents menus. Ces rubriques ne seront pas toutes du même type. Certaines seront des rubriques classiques, comme par exemple la rubrique "Nouveau" du menu "Fichier" :
  ```
  file_menu->Append(MENU_NEW, wxT("Nouveau\tCtrl-N"));
  ```
  On remarque que cet ajout de rubriques se fait grâce à la fonction `Append` de la classe wxMenu, qui prend 2 paramètres. Le premier est l'identificateur de la rubrique, qui doit être (un entier) unique. Le mieux est de le déclarer en tant que constante dans un enum (par exemple dans le fichier mainframe.h) (voir ce site sur les enum). Le 2ème paramètre est une chaîne de caractère qui sera le texte (titre) écrit sur la rubrique. Cette chaîne de caractère (ici "Nouveau") peut être suivie par une chaîne de type "\tRaccourci", ce qui permet de définir facilement un raccourci pour cet élément (ici "\tCtrl-N", qui permet d'accéder à l'élément en appuyant simultanément sur Ctrl et N).
  Les rubriques à ajouter sont : "Nouveau", "Ouvrir", "Sauvegarder", et "Quitter" pour le menu Fichier, "Epaisseur trait", "Couleur", et "Gestion des triangles" pour le menu Options, et "Version" pour le menu Aide. N'oubliez pas de déclarer dans l'enum un identificateur différent pour chaque élément.
- Ajoutons maintenant un élément de type "rubrique avec marque" dans le menu Affichage. Il s'agit d'un bouton intitulé "Barre d'outils" qui nous permettra plus tard d'afficher ou de masquer la barre d'outils. Selon que la barre d'outils sera affichée ou masquée, une petite marque s'inscrira ou non à côté du texte de la rubrique. Pour cela, on doit utiliser la fonction `AppendCheckItem`, qui prend les mêmes paramètres que `Append`. N'oubliez pas de déclarer un identificateur unique pour cet élément également (par exemple `MENU_TOOLBAR`).

On précisera alors ensuite que cet élément doit être coché au démarrage par l'instruction :
```
aff_menu->Check(MENU_TOOLBAR,TRUE);
```
si aff_menu est la variable concernant le menu Affichage. De la même façon, on aurait pu décocher l'élément au démarrage en remplaçant TRUE par FALSE.

- On souhaite maintenant rendre la rubrique "Gestion des triangles" grisée par défaut en début d'application, puis la rendre active par la suite dès qu'au moins un triangle sera chargé ou dessiné. Pour cela, on va ajouter l'instruction :
```
menu_bar->Enable(MENU_MANAGE,false);
```
si MENU_MANAGE est l'identificateur concernant la rubrique "Gestion des triangles".
- Ajoutons encore quelques séparateurs pour faire joli, entre "Nouveau" et "Ouvrir", entre "Sauvegarder" et "Quitter", en plaçant entre leurs instructions d'ajout une commande de type :
```
file_menu->AppendSeparator();
```
- Il ne reste plus qu'à dire à la fenêtre quelle est la barre de menu qu'elle doit afficher grâce à l'instruction :
```
m_MainFrame->SetMenuBar(menu_bar);
```

Compilez et essayez les boutons du menu. L'élément "Barre d'outils" doit alternativement se cocher ou se décocher lorsque vous cliquez dessus. L'élément "Gestion des triangles" doit être grisé et inutilisable. Les autres doivent être cliquables, mais pour l'instant ne rien produire.
Votre application doit ressembler plus ou moins à :



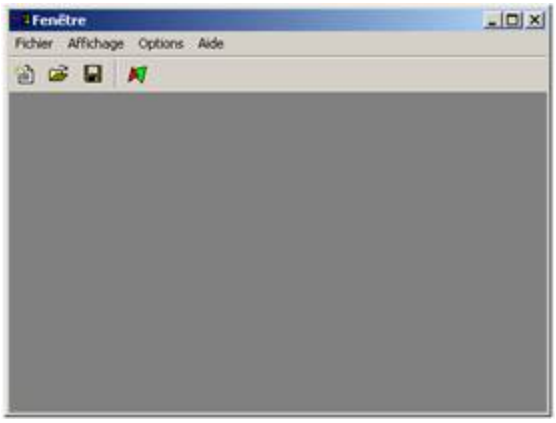
3. Création de la barre d'outils

C'est maintenant au tour de la barre d'outils. Créons une variable privée contenant cette barre d'outils dans la classe CMainFrame. Elle utilise le type prédéfini wxToolBar :
```
wxToolBar *m_toolbar;
```
Créons également une petite fonction membre publique "CreateMyToolbar" dans la classe CMainFrame.
Dans l'implémentation de cette fonction, nous allons ajouter les instructions pour la création et l'initialisation de cette barre :

- Créons tout d'abord la barre vide, en précisant ses propriétés, grâce à l'instruction suivante :
```
m_toolbar=CreateToolBar(wxNO_BORDER | wxTB_HORIZONTAL,TOOLBAR_TOOLS);
```
Cette barre d'outils n'aura donc pas de bord, et aura une disposition horizontale. En réalité, ce sont les 2 styles par défaut, qui n'ont donc pas vraiment besoin d'être précisés. On peut cependant en ajouter d'autres, comme par exemple wxTB_FLAT, qui donne un aspect plat aux boutons. Les styles sont toujours séparés par des barres verticales "|". Le dernier paramètre est l'identificateur (unique) de la barre d'outils, qui doit être déclaré dans l'enum.

- Chargeons ensuite les images bitmap qui seront utilisées par les boutons de la barre d'outils. Ajoutez les instructions :
```
wxBitmap toolBarBitmaps[4];
toolBarBitmaps[0] = wxBitmap(wxT("new.bmp"),wxBITMAP_TYPE_BMP);
toolBarBitmaps[1] = wxBitmap(wxT("open.bmp"),wxBITMAP_TYPE_BMP);
toolBarBitmaps[2] = wxBitmap(wxT("save.bmp"),wxBITMAP_TYPE_BMP);
toolBarBitmaps[3] = wxBitmap(wxT("draw.bmp"),wxBITMAP_TYPE_BMP);
```
  afin de mettre les images "new.bmp", "open.bmp", save.bmp", et "draw.bmp" dans un tableau de wxBitmaps.
- Ajoutez l'instruction :
```
m_toolbar->SetToolBitmapSize(wxSize(toolBarBitmaps[0].GetWidth(),
                                    toolBarBitmaps[0].GetHeight()));
```
  afin d'ajuster la taille de la barre d'outils à la taille des images.
- Ajoutons ensuite les 3 premiers boutons d'outils, correspondant aux 3 premières images bitmaps chargées, grâce à des instructions de type :
```
m_toolbar->AddTool(MENU_NEW, wxT("Nouveau"), toolBarBitmaps[0]);
```
  Les 3 premiers outils ("Nouveau", "Ouvrir", "Sauvegarder") correspondant aux mêmes fonctionnalités que les éléments de menus portant le même nom, ils seront rattachés aux mêmes identifiants. Pas la peine donc d'en créer de nouveaux. Ceci permet de rediriger les événements des 2 sortes de widgets (les boutons de la barre d'outils et les éléments du menu) vers les mêmes fonctions de traitement, ce qui nous évite de redéfinir plusieurs fois les mêmes comportements.
  Le deuxième paramètre sert à donner un texte au bouton dans le cas où on aurait choisi le style "image + texte" pour l'apparence des boutons, en ajoutant "`| wxTB_TEXT`" lors de la création de la barre d'outils.
  Le troisième paramètre précise l'image à utiliser pour le bouton. Vous devez pour cela utiliser les éléments du tableau de wxBitmaps.
  Un quatrième paramètre peut être ajouté pour définir un texte d'aide qui apparaît lorsque la souris passe sur le bouton, par exemple "`,"Nouveau dessin""`
- Ajoutons ensuite le 4ème bouton d'outil grâce à la fonction `AddCheckTool`, qui prend les mêmes paramètres que `AddTool`, mais qui crée un bouton qui peut prendre 2 positions stables : enfoncé ou relâché (dans le même esprit que "coché ou non coché"). Créer un nouvel identifiant pour cet outil.
- Ajoutons enfin un petit séparateur entre le 3ème et le 4ème outil pour faire joli (décidément, qu'est-ce qu'elle va être belle cette appli…), en mettant en bonne place l'instruction :
```
m_toolbar->AddSeparator();
```
- Nous pouvons maintenant déclarer que la barre d'outils est prête, et dire à la fenêtre quelle est la barre d'outils à afficher, grâce aux instructions :
```
m_toolbar->Realize();
SetToolBar(m_toolbar);
```
- N'oublions pas d'appeler CreateMyToolbar dans MyApp::OnInit, juste avant de rendre la fenêtre visible.

Compilez et essayez les boutons. Seul celui de droite doit rester enfoncé lorsqu'on appuie dessus. Les autres se relâchent immédiatement. Essayez également différents styles pour la barre et ses outils.
Votre application doit ressembler plus ou moins à :



4. Création de boîtes de dialogue

Nous allons maintenant créer 2 nouveaux fichiers "dialogs.h" et "dialogs.cpp" afin d'y mettre les déclarations de classes de chaque type de boîte de dialogue, ainsi que l'implémentation de leurs fonctions.

N'oublions pas de mettre les ordres #ifndef pour éviter les inclusions récursives du fichier .h.

Nous allons créer 5 boîtes de dialogue différentes. Commençons par la plus simple : l'affichage de la version.

- Créez dans dialogs.h la classe VersionDialog contenant uniquement un constructeur :
```
class VersionDialog: public wxDialog
{
public :
     VersionDialog(wxWindow *parent, wxWindowID id,
                   const wxString &title);

private :

DECLARE_EVENT_TABLE()
};
```
  Cette classe dérive de wxDialog, la classe de base des boîtes de dialogue de wxWidgets, ce qui lui confère toutes les fonctionnalités de base qui vont bien.
- Puis, dans dialogs.cpp, implémentez le constructeur :
```
VersionDialog::VersionDialog( wxWindow *parent, wxWindowID id,
                              const wxString &title) :
                             wxDialog( parent, id, title)
{
}
```
  Comme vous pouvez le constater, ce constructeur, pour l'instant vide, va commencer par appeler directement le constructeur de base de la classe wxDialog, grâce à l'instruction juste derrière le double point ":". Le constructeur de base nous débarrasse de toutes les initialisations de base, qui seront donc faites automatiquement.
  Afin de compiler, vous devez ajouter ces deux lignes dans dialogs.cpp (avant le constructeur) :
```
BEGIN_EVENT_TABLE(VersionDialog, wxDialog)
END_EVENT_TABLE ()
```
  Nous verrons au prochain TP à quoi servent ces deux lignes.
- Ensuite, nous allons ajouter et organiser les widgets. Au final, la boîte de dialogue devra avoir plus ou moins l'allure suivante :

  Autrement dit, nous devons placer 2 widgets : un texte et un bouton OK l'un au-dessus de l'autre. Nous allons donc créer un conteneur vertical (qu'on nommera par exemple item0) pour y insérer les 2 widgets :
```
wxBoxSizer *item0 = new wxBoxSizer( wxVERTICAL );
```
- Le texte est du type wxStaticText. On crée un tel widget de la façon suivante :
```
wxStaticText *item1 = new wxStaticText( this, ID_TEXT, wxT("Version 1.0"),
            wxDefaultPosition, wxDefaultSize, wxALIGN_CENTRE );
```
  Le premier argument est un pointeur sur le widget auquel ce nouveau widget va être rattaché, autrement dit son parent. Ici, le parent de ce texte sera la boîte de dialogue de version, donc "this".
  Le second argument est un identificateur unique pour ce widget. Il sera défini par exemple dans un enum dans dialogs.h : `ID_TEXT = 10000`. On peut également utiliser comme identificateur un identificateur prédéfini dans wxWidgets, comme par exemple wxID_OK pour le bouton OK.

Attention à ne pas définir pour un nouvel identifiant un entier déjà utilisé par les identificateurs prédéfinis de wxWidgets.

Le 3ème argument est le texte à afficher.
Les 4ème et 5ème arguments sont des positions et tailles par défaut.
Le dernier argument est comme d'habitude le style. Par exemple ici pour centrer le texte.

- Le bouton OK est de type wxButton. On utilisera pour identifiant wxID_OK :
```
wxButton *item2 = new wxButton( this, wxID_OK, wxT("OK"),
                                wxDefaultPosition);
```
Dans toutes nos boîtes de dialogue, les boutons OK auront toujours l'identifiant wxID_OK.
- Nous devons maintenant ajouter ces widgets dans le conteneur item0 grâce à des instructions du type :
```
item0->Add( item1, 0, wxALIGN_CENTRE|wxALL, 5 );
```
Ici, les arguments sont : le widget à ajouter, 0 qui veut dire qu'on ne peut pas changer la taille, des styles, et la dimension de l'espace minimal à respecter autour du widget ajouté (par exemple, il y aura un espace vide de 5 tout autour du texte).
- On ajoute ensuite ces quelques lignes pour dire quel est le conteneur à afficher dans la boîte de dialogue, et pour préciser que le placement doit se faire de façon correcte et optimale :
```
this->SetAutoLayout( TRUE );
this->SetSizer( item0 );
item0->Fit( this );
item0->SetSizeHints( this );
```

Voilà, vous avez créé une belle boîte de dialogue. Mais pour l'instant, elle ne s'affiche pas. Si vous voulez contempler votre œuvre, vous pouvez momentanément ajouter les instructions suivantes à la fin de MyApp::OnInit, juste avant le "return TRUE;", afin que la boîte de dialogue s'ouvre directement au démarrage :
```
VersionDialog vdlg(p_MainFrame, -1, wxT("Version"));
vdlg.ShowModal();
```
On crée en fait une variable de ce type de boîte de dialogue, en envoyant comme paramètres la fenêtre parente à laquelle elle doit être rattachée, un identificateur bidon, et un titre. Puis, on l'affiche de façon modale en faisant ShowModal.
Lorsque vous aurez fini vos tests de boîte de dialogue, n'oubliez pas de supprimer ces deux lignes.

Maintenant, **à vous de jouer** !
Créez les boîtes de dialogue suivantes, en vous servant de la liste de types de widgets donnée juste à côté. Utilisez la documentation cd wxWidgets pour trouver les constructeurs adéquats.

wxStaticText
wxSlider (NB : les graduations ne marchent pas sous linux...)
wxButton

wxStaticText

wxRadioBox
wxButton

utilisez wxString strs8[] = { wxT("Rouge"), wxT("Vert"), wxT("Bleu")};
comme 7ème argument lors de la création de la radio box.

wxStaticText
wxListBox
wxButton

ici, 3 wxBoxSizer sont nécessaires

wxStaticText
wxTextCtrl
wxStaticText
wxSpinCtrl
wxRadioBox
wxButton

ici, 3 wxBoxSizer sont nécessaires