TP wxWidgets n°3

Objectif général

Le but des 5 TP est 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 des TP, vous pouvez 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 3ème TP, nous allons introduire la structure de données triangle, et voir comment interagir entre l'application et des fichiers.

Manipulations

1. Structure de données triangle

Téléchargez le fichier triangle.h, qui contient la définition d'une structure *point*, contenant deux coordonnées *x* et *y*, ainsi que la classe *Triangle*, qui contient 2 sommets *p1*, *p2*, une couleur *colour*, et une épaisseur de trait *thickness*.

2. Chargement d'un fichier triangle

Téléchargez le fichier trian.tri qui contient un exemple de fichier texte contenant des triangles. Il est écrit dans le format que nous avons choisi pour sauvegarder des fichiers.

Ce format est constitué d'une première ligne contenant un entier *n* représentant le nombre de triangles à dessiner.
Puis, après une ligne vide, viennent *n* blocs de données contenant chacun 3 lignes :

- la première contient 6 entiers correspondant aux coordonnées 2D des 3 sommets :
  `x1 y1 x2 y2 x3 y3`
- la seconde contient 3 entiers compris entre 0 et 255 qui correspondent à la couleur de l'intérieur du triangle au format RGB
- la troisième contient un entier représentant l'épaisseur de trait du bord du triangle.

Ajoutez à la classe CMainFrame une variable publique entière `num_tri`, qui représentera le nombre de triangles à afficher, et qui devra être initialisée à 0 au lancement de l'application ou lorsqu'on recommence un dessin.
Ajoutez également une variable publique `tab_tri` de type tableau de triangles qui va (pour simplifier) contenir un maximum de 5 triangles (ou pointeurs sur des triangles).

Lorsqu'un utilisateur choisit de cliquer sur « Ouvrir » ou sélectionne l'icône d'ouverture de fichier, l'application devra ouvrir un fichier triangle, dont le format sera celui décrit ci-dessus, en demandant à l'utilisateur de choisir le fichier à ouvrir.
Pour cela, on dispose de la classe *wxFileDialog* de wxWidgets, qui permet d'ouvrir une boîte de dialogue de choix de fichier, en spécifiant quelques options du type :

- répertoire par défaut
- nom par défaut
- extension par défaut,
- etc.

Parmi ces options, se trouve également le style de boîte de dialogue à afficher en fonction du but de ce choix de fichier : *ouverture* ou *sauvegarde*. Dans le cas présent, nous choisirons le style wxOPEN (ce qui indique une boîte d'ouverture de fichier). Regardez dans la doc pour voir le fonctionnement de cette classe.

Lorsqu'on affiche une telle boîte de dialogue, si l'utilisateur clique sur OK, alors on doit ouvrir le fichier choisi.

Pour cela, on va utiliser la commande suivante :
`std::ifstream fo(filedialog.GetPath().fn_str(), std::ios::in);`
qui ouvre un fichier en lecture pour récupérer son contenu (c'est la version C++ de fopen et fclose).
Dans l'exemple ci-dessus, la boîte de dialogue de choix de fichier s'appelait `filedialog`, on a récupéré le nom complet du fichier choisi grâce à la fonction `GetPath`, puis transformé ce nom (qui était au format wxString) en chaîne de caractères grâce à la fonction fn_str. Le paramètre `std::ios::in` sert à indiquer que le fichier sera ouvert en lecture. Si on voulait l'ouvrir en mode écriture, on mettrait `std::ios::out`.

Si l'ouverture de ce fichier échoue, il faut afficher un message d'erreur :
```
// if open file failed, show an error message box
if (!fo)
{
      wxString errormsg, caption;
      errormsg.Printf(wxT("Unable to open file "));
      errormsg.Append(filedialog.GetPath());
      caption.Printf(wxT("Erreur"));
      wxMessageDialog msg(this, errormsg, caption, wxOK | wxCENTRE | wxICON_ERROR);
      msg.ShowModal();
      return ;
}
```

Notez dans cet exemple que le test d'échec d'ouverture se fait simplement en testant si `fo` (notre descripteur de fichier) est « vrai » (fichier ouvert) ou « faux » (fichier non ouvert).

On peut alors commencer à récupérer les données selon le modèle suivant :
```
fo >> num_tri;
fo >> x >> y;
```

Les étranges `>>` sont appelés *opérateurs d'extraction*. En gros, ils indiquent simplement que l'on veut lire une donnée du fichier `fo`, et qu'on veut la ranger dans `num_tri`. Ce qui est particulièrement sympathique ici c'est qu'il n'est pas nécessaire de spécifier le type de la valeur à lire (entier, chaîne, flottant, etc.) : le compilateur le détermine automatiquement en fonction du type de la variable dans laquelle on veut stocker la valeur (polymorphisme). Bien entendu, il faut que ce qui est lu du fichier corresponde (par exemple, lire « abcde » dans un entier ne fera pas quelque chose de très intéressant…).
Notons également au passage que les espaces et retours à la ligne sont interprétés comme des séparateurs pour les différentes valeurs à lire.
Une fois qu'on a récupéré toutes les données du fichier et qu'on les a placées dans `tab_tri`, il faut penser à activer le bouton de Gestion des triangles dans le menu Options. Ce bouton ne doit être actif que si `tab_tri` contient des triangles à afficher.

3. Sauvegarde d'un fichier triangle

Nous allons maintenant nous intéresser à l'opération inverse qui est celle de la sauvegarde de fichiers triangles.
Cette opération ressemble beaucoup à la précédente, excepté que le style de la boîte de dialogue sera wxSAVE, agrémenté d'un wxOVERWRITE_PROMPT, que l'ouverture du fichier pour écriture dans ce fichier (et non plus simple lecture) se fait par :

```
std::ofstream fs((char*)filedialog.GetPath().c_str(), std::ios::out);
```

et que l'écriture dans un fichier se fait par des instructions de type :

```
fs << num_tri;
```

sans oublier d'ajouter des caractères de fin de ligne :

```
fs << std::endl;
```

Vous noterez la similitude entre les >> de tout à l'heure et les <<. En effet, ces opérateurs, appelés opérateurs d'insertion, font exactement l'inverse des opérateurs d'extraction : ils écrivent dans le fichier, une valeur formatée correctement en fonction du type de la variable écrite. Pour insérer un retour à la ligne, vous pouvez utiliser :

```
fs << std::endl;
```

`std::endl` étant une constante indiquant d'insérer un retour à la ligne dans le fichier.

4. Liens avec les événements

Il faut maintenant s'arranger pour que les événements qui peuvent survenir sur les widgets aient un effet sur le tableau de triangles et réciproquement :

- la boîte de dialogue de gestion des triangles doit refléter le contenu de ce tableau, et lorsqu'on choisit de visualiser les propriétés d'un des triangles, celui-ci doit être mis à jour.
- on doit pouvoir supprimer un triangle de la liste grâce au bouton supprimer.
- la suppression de tous les triangles doit entraîner la désactivation du bouton de Gestion des triangles.
- L'appui sur le bouton Nouveau doit effacer tout.

A vous de jouer pour toutes ces fonctionalités…

Et la semaine prochaine, on affichera ces triangles avec OpenGL.