TP OCI n°3

1 Vers la gestion des outils de dessin

A Création des outils

Vous avez précédemment créé la classe `DrawingZone` dont l'instance dans votre interface sert à la fois à réaliser les dessins (*cf.* TP2) et à récupérer les interactions pour créer de nouveaux dessins ou autres actions.

Le principe que je propose pour arriver au résultat demandé pour le TP est de ne pas « câbler en dur » les signaux de la `DrawingZone` avec un comportement (et un gros *switch* pour leur sélection[1]), mais d'utiliser les possibilités de connexion dynamique des signaux. En gros, quand l'utilisateur choisit un outil, l'outil sait quels sont les interactions qu'il doit capter sur la `DrawingZone` (et à quelles actions elles correspondent), alors c'est lui qui va faire son câblage lui-même.

Le fonctionnement proposé pour réaliser cela est le suivant :

- Lors du choix d'un outil (objet de la classe `Tool`, *cf.* ci-dessous), les signaux existants sur la `DrawingZone` sont supprimés et les nouveaux signaux sont connectés.
- Tout pendant qu'un outil est sélectionné, les interactions font appel aux fonctionnalités de cet outil.
- Un outil qui aura besoin de couleur récupérera celle définie par l'utilisateur dans le `ColorSelector` (directement ou avec des intermédiaires). Il n'y a pas de raison d'avoir de couleur en membre comme membre d'un outil.
- La `DrawingZone` offre la possibilité d'ajout et de suppression de forme par un outil.

Cette méthode de conception offre une grande souplesse pour le développement de plugins puisqu'on ne limite aucunement les interactions possibles avec la `DrawingZone`. On laisse donc la porte ouverte à toute forme d'outils.

Pour cela, il faut créer une classe abstraite `Tool` ayant l'allure suivante :

```
class Tool {
protected:
 string _name; //Nom de l'outil utilisé dans l'interface
public:
 Tool():_name("no name"){};
 virtual ~Tool(){};

 /**
  * Implémentation du comportement lors de la sélection de l'outil
  */
 virtual void selected(DrawingZone *dz) =0;

 void setName(string n) {_name=n;};
 const string name() const {return _name;};
};
```

Les classes qui héritent de la classe `Tool` devront :

- implémenter des fonctions (`protected` ou `public`) décrivant le comportement (ou l'effet) de l'outil lors des interactions. Pour les outils d'ajout de forme, dans la mesure où on souhaite avoir un retour visuel sur la forme en cours de construction, il est nécessaire d'implémenter au moins les fonctions correspondants aux évènements suivants :
  - clic initial du bouton,
  - mouvement de la souris (bouton enfoncé),

1 De plus, j'ai l'impression que cette solution ne serait pas facile à gérer avec l'utilisation de plugins.

- relâchement du bouton.

- implémenter la fonction `selected` qui établis les connexions entre les signaux du `DrawingZone` (qui aura été passé en paramètre) et les *handlers* définis dans votre classe.
- il faudra également penser à implémenter la suppression des connexions déjà existantes pour le `DrawingZone` lors d'un changement d'outil,. Il sera donc nécessaire de conserver une instance des connexions établis pour faire ensuite appel à un `disconnect()` (*cf.* cours). *Attention*, il n'est pas question de laisser la charge aux outils de se déconnecter aux mêmes : dans le cadre du développement de plugins, on ne peut pas prendre le risque qu'un développeur oublie cet étape et endommage le fonctionnement global de l'application.

B Intégration dans l'application principale

Le propos est simplement de faire fonctionner des outils avec la `DrawingZone`. La gestion intégrée des outils dans l'interface n'est pas essentielle ici (*i.e.* toutes les histoires de menus et de radio-boutons). Ici, on veut jusque que lorsqu'on clique sur un bouton, on passe sur un outil et lorsqu'on clique sur un autre bouton, on ait un autre outil.

Il nous faut donc au préalable deux outils, par exemple :

- l'ajout d'une forme rectangle (*cf.* TP précédent pour la forme de rectangle),
- l'ajout d'une forme de rectangle creux (pas beaucoup plus compliqué)

On pourra utiliser les deux boutons du TP précédent pour les deux outils.

Dans la classe de votre fenêtre principale, créez une fonction `void select(Tool *t)` qui 1) déconnecte les signaux existants, et 2) fait appel à la fonction `selected()` de `t` pour le connecter à la `DrawingZone`.

Cette fonction `select` sera connectée aux deux signaux `signal_clicked()` des deux boutons de sorte à obtenir le comportement décrit plus haut.

2 Système de plugins

Un exemple général sur le fonctionnement des modules de Gtkmm vous offre une classe quasiment clé-en-main pour la construction d'un système de gestion de plugins.

Dans cette partie, il faut donc :

- adapter cet exemple pour automatiser le chargement de plugins d'outils au démarrage de votre application,
- créer un ou deux plugins pour votre application.

Dans notre cas, les plugins seront des objets qui héritent de la classe `Tool`. Pour un outil d'ajout de forme, un plugin contiendra une classe pour l'outil proprement dit et une classe pour la forme (héritée de `Shape`) qui sera produite par l'outil. Normalement, si vous vous y prenez bien, il est suffisant de ne charger que des objets de la classe héritée de `Tool` pour ajouter des nouvelles formes.

3 Développement des outils génériques

Jusque là, nous ne nous sommes intéressé qu'à des outils d'ajout de forme. Il reste deux outils spécifiques qui sont demandés pour ce TP :

- la suppression d'une forme,
- le déplacement d'une forme.

Dans les deux cas, pour déterminer les forme sur lesquels l'utilisateur clique, vous aurez besoin de la fonction `first_over` de la `DrawingZone`, et donc d'implémenter des fonctions `over` dans les classes héritées d'une `Shape`.