**Compte Rendu TP4 C++ : Héritage, polymorphisme**

**Document de spécification et de conception**

**I\ Spécifications**

Le but de ce TP est de manipuler des figures géométriques différentes. L’utilisateur utilisera l’application à travers des lignes de commandes. Il pourra ainsi créer, modifier et supprimer des figures. Il lui sera aussi possible de sauvegarder son travail ou de le charger.

**Définitions**

**Cercle :** Figure géométrique caractérisé par son origine et son rayon

**Rectangle :** Figure géométrique caractérisé par deux points étant des extrémités opposées du quadrilatère.

**Ligne :** Figure géométrique caractérisé par deux points.

**Polyligne :** Figure géométrique caractérisé par n points. Une polyligne peut avoir ses arêtes qui se croisent.

**Figure :** Ensemble regroupant les quatre formes énoncées ci-dessus.

**Sélection :** Objet capable d’englober plusieurs figures situées dans une délimitation.

**Graphics :** Ensemble regroupant les figures et les sélections

**Container :** Espace dans lequel toutes nos figures et sélections sont créées et gérées, il reçoit les demandes de l'utilisateur..

**Command :** Élément représentant une action dans le modèle. Permet de faire et défaire cette action.

**Choix généraux**

***Spécifications du cahier des charges :***

Pour pouvoir gérer tous les cas de configuration possibles, il a été nécessaire d'éclaircir le cahier des charges. Ainsi, un cercle peut avoir un rayon nul mais pas négatif. Une polyligne est composée d'un point au minimum tandis qu'une ligne et un rectangle sont toujours composés de seulement deux points.

La sélection ne contient que les figures qui sont entièrement incluses dans celle-ci (et non pas partiellement).

Seules les commandes propres aux figures sont affichées si l’utilisateur fait appel à la méthode LIST. Les sélections ne sont pas affichées.

Un objet étant recréé (par un UNDO) retourne dans les sélections qui le contiennent si celles-ci sont toujours présentes.

**II\ Conception**

Nous avons choisi une structure composée de 17 classes, une structure Point et un main.

**La classe main:** Gère la saisie des commandes saisies par l’utilisateur et fait appel aux méthodes de la classe container en fonction de celles-ci.

**La classe Container:** Classe qui contient toutes les méthodes exigées dans le cahier des charges. Ces méthodes sont appelées depuis le main en fonction des entrées saisies par l'utilisateur.

Cette classe gère également le stockage des objets Command et implémente le UNDO/REDO grâce à deux piles.

C'est la classe qui contient tous nos éléments graphiques et par ce fait les créé, supprime, et modifie.

**La classe Graphics:** Classe abstraite définissant tous les éléments que l'on peut implémenter dans le container (Cercle, Rectangle, Polyligne, Ligne ou Sélection). Un élément graphique est caractérisé par un nom, une ligne de commande qui permet d'afficher ses caractéristiques, ainsi que d'un état déterminant si cet élément est actif ou nom dans le dessin (on positionne sa valeur avec les constantes GRAPHICS\_STATE\_IN\_DRAW et GRAPHICS\_STATE\_NOT\_IN\_DRAW).

**La classe Figure:** Classe abstraite héritant de Graphics. Elle définit toutes les formes géométriques (les sélections ne sont pas des figures). Elle contient les attributs/méthodes communs à toutes les figures : un point d'origine, deux points (*leftUpCorner* et *downRightCorner*) formant la zone qui entoure exactement la forme, ainsi qu'une liste de pointeur sur Selection permettant à l'objet de savoir quelles Selections le contiennent.

**La classe Selection:** Classe héritant de Graphics. Elle permet d'implémenter les sélections comme prévu dans le cahier des charges. Une sélection peut contenir plusieurs figures mais n'est pas persistante. Elle comporte donc un vecteur de pointeur de Figures. Stocker les pointeurs est obligatoire car Figure est une classe abstraite. La création d'une sélection renseigne les Figures qu'elle contient de leur appartenance à celle-ci.

**La classe Circle:** Classe héritant de Figure. Elle permet d'implémenter des cercles. Un cercle est caractérisé par son origine et son rayon.

**La classe Rectangle:** Classe héritant de Figure. Elle permet d'implémenter des rectangles. Un rectangle est caractérisé par deux points.

**La classe Line:** Classe héritant de Figure. Elle permet d'implémenter des lignes. Une ligne est caractérisée par deux points.

**La classe Polyline:** Classe héritant de Figure. Elle permet d'implémenter des polylignes. Une polyligne est caractérisée par une série de points.

**La classe Command:** Classe abstraite de base qui représente une action de l'utilisateur sur le modèle. Elle possède deux méthodes *Execute* et *Unexecute* permettant respectivement de refaire et défaire une action spécifique. Cette implémentation est inspirée du *Design Pattern Command*. Elle possède également un attribut *whichList* qui renseigne sur la pile où se trouve actuellement la commande.

**La classe ComposedCommand:** Classe abstraite héritant de Command. Cette classe permet grâce a un vecteur de pointeur de Command l'execution de plusieurs Command en un seul appel.

**La classe SimpleCommand:** Classe abstraite héritant de Command. Contrairement à la ComposedCommand, cette classe n'execute qu'une seule action.

**La classe DeleteCommand:** Classe héritant de ComposedCommand. Utilisée pour supprimer un ensemble d'élements. Elle sera par exemple créée lors d'un DELETE de plusieurs élements ou lors d'un DELETE d'une Selection.

**La classe LoadCommand:**Classe héritant de ComposedCommand. Utilisée pour créer un ensemble d'élements. Elle sera utilisée pour le LOAD.

**La classe DeleteElementCommand:**Classe héritant de SimpleCommand. Utilisée pour supprimer un seul élement. Elle sera créée pour chaque element d'un DELETE.

**La classe CreateElementCommand:**Classe héritant de SimpleCommand. Utilisée pour créer un seul élément. Elle sera créée pour chaque élément d'un LOAD, ainsi que pour un nouvel élément venant de C, L PL, R.

**La classe MoveCommand:**Classe héritant de SimpleCommand. Utilisée pour déplacer un seul élément (Selection comprise). Elle sera créée lors d'un MOVE.

**La classe Tools:** Classe outil qui contient des méthodes statiques pour manipuler les chaines de caractères (fonction split) et tester l'existence d'un fichier. Cette classe améliore la visibilité et la clarté du code.

**Structures de données et algorithmes**

*Comment choisir la structure de données?*

Le stockage des éléments graphiques dans le container se fait par une map<string,Graphics\*>. La clé sera le nom du Graphics et la valeur un pointeur sur ce Graphics. Ainsi les éléments sont triés par leur nom et l'affichage de LIST est directement dans le bon ordre.

Le Container possède également un unordered\_set<string> contenant les également les noms des Graphics présents dans la map.

Le stockage de l'ensemble de Command pour UNDO/REDO se fait par deux list<Command\*> utilisées comme des piles.

*Comment savoir si une Figure est incluse dans une sélection ?*

Afin de savoir si toute figure est contenue ou non dans une sélection, il a été décidé de mettre en place un rectangle « fictif » propre à chaque figure. Ce rectangle est définit comme étant le plus petit rectangle dans lequel la figure est entièrement incluse. C'est en comparant les coordonnées de ce rectangle pour chaque Figure avec les coordonnées de la sélection que nous pouvons déterminer rapidement les figures qui sont incluses dans la sélection.

*Pourquoi séparer Selection et Figure ?*

Une Figure est undo-able (sur sa création, suppression, déplacement) alors qu'une sélection ne l'est pas totalement (seulement sur la suppression). De plus, une sélection ne doit pas apparaitre dans la liste des objets présents dans le container si l'utilisateur appelle la commande LIST. Pour finir, les figures sont sauvegardées dans un fichier à l'appel de la commande SAVE tandis que les sélections ne le sont pas. Tous ces éléments nous ont conduit à séparer ces deux types d'objets dans deux classes différentes. Enfin une Selection 'contient' (seulement leur pointeurs) des Figures.

*Comment sauvegarder/afficher uniquement les figures(et non pas les sélections) ?*

Pour la sauvegarde et l'affichage, il a été décidé de placer l'attribut commande dans la classe Graphics. La commande d'une figure est la chaine de caractères saisie par l'utilisateur (mise a jour à chaque déplacement de la Figure) alors que la commande d'une sélection est toujours vide (chaine vide). Par conséquent, dans le fichier, nous sauvegardons les commandes de création de tous les Graphics. Ainsi, les commandes des sélections sont sauvegardées mais n'introduisent aucun caractère dans le fichier de sauvegarde. Seules les commandes de création de figures sont écrites dans le fichier.

*Fonctionnement UNDO/REDO ?*

Afin de pouvoir effectuer des UNDO/REDO, nous nous sommes inspiré du *Design Pattern Command*  et avons implémenté deux liste de pointeurs de commandes : Une liste UNDO et une liste REDO. Ainsi, à chaque modification du container (par des ajouts, déplacements, suppression...), ces listes sont mises à jour de la manière suivante :

-Pour chaque nouvelle commande du type création / suppression de Figure, suppression de Selection, ainsi que MOVE, LOAD et CLEAR, la commande exécutée vient s'ajouter au début de la liste de UNDO et la liste de REDO devient vide.

-Pour chaque nouvelle commande du type création de sélection, ainsi que LIST, SAVE et EXIT, les listes UNDO et REDO ne sont pas modifiées.

-L'appel à UNDO dépile la liste des undo et *execute* cette Command et l'empile dans la liste des redo.

-L'appel à REDO dépile la liste des redo et *unexecute* cette Command et l'empile dans la liste des undo.

*Comment est gérée la mémoire ?*

L'allocation d'un objet ne se fait qu'une fois lors de sa création. Il n'y a pas de duplication de l'objet dans la mémoire car toutes les classes ayant besoin d'un même objet ne contiennent que son pointeur référant la même instance.

La libération de la mémoire d'un Graphics ne se fait pas lors du DELETE de cet élément. Pour économiser les appels à new et delete et ainsi optimiser l’exécution des UNDO/REDO potentiels. Lorsqu'un élément doit être supprimé, on enlève son pointeur de la map qui stocke les objets actifs, et donne ce pointeur à la Command qui l'a supprimé. La destruction des objets ne se fait donc que dans 3 situations :

-Fin de l'exécution et suppression du Container. On le vide de ses Graphics.

-Si une Commande de type DeleteElementCommand est supprimée et qu'elle se situait dans la pile des undo, la suppression de l’élément est donc 'validée' et on libère la mémoire du Graphics associé.

-Si une Commande de type CreateElementCommand est supprimée et qu'elle se situait dans la pile des redo, l'annulation de création de l’élément est donc 'validée' et on libère la mémoire du Graphics associé.

*Comment est mise à jour la liste de Figure de la classe Selection ?*

Un problème peut arriver si une figure contenue dans une sélection est supprimée du container(mais toujours en mémoire) puis que la sélection est amenée à être supprimée puis recréée (UNDO). Il faut donc que la sélection sache si ses figures sont actives ou non. Pour cela, nous utilisons l'attribut *state* des Graphics.

Si une figure est libérée de la mémoire il faut en avertir les sélections qui la contiennent. Pour cela, il faut mettre à jour les listes de Figure de chaque Sélection (avec la méthode EraseFigure ). Un élément Graphics connaissant les sélections qui le contiennent.

Comment est mise à jour la liste de Selections de la classe Figure ?

Dans le cas où l'on supprime une sélection contenant des Figures puis que l'on recrée ces figures avec un UNDO, la sélection ne revient pas. Il faut alors informer les Figures qu'elles ne sont plus contenues dans cette sélection. Pour cela, on fait appel à la méthode EraseSelect.

**Spécification des tests fonctionnels**

Index des tests :

0\_x=Test de création

1\_x=Test de déplacement

2\_x=Test de suppression

3\_x=Test UNDO/REDO

4\_x=Test LOAD/SAVE

5\_x=Cas limites/ cas d’erreur

**Test 0\_1 :** Création d’un cercle

On vérifie que le cercle à bien été créé avec les bons paramètres

**Test 0\_2 :** Création rectangle

On vérifie que le rectangle à bien été créé avec les bons paramètres

**Test 0\_3 :** Création ligne

On vérifie que la ligne à bien été créée avec les bons paramètres

**Test 0\_4 :** Création polyligne

On vérifie que le polyligne à bien été créé avec les bons paramètres

**Test 0\_5 :** Création sélection

On vérifie que la sélection a bien été créée et qu’elle n’apparait pas si on LIST

**Test 1\_1 :** Déplacement figure

On vérifie que les coordonnées de la figure passée en paramètre ont été incrémentées de dx et dy

**Test 1\_ 2:** Déplacement sélection

On vérifie que les coordonnées de toutes les figures présentes dans la sélection passée en paramètre ont été incrémentées de dx et dy

**Test 1\_3 :** Déplacement d’une figure qui se trouve dans une sélection

On vérifie que les coordonnées de la figure passée en paramètre ont été incrémentées de dx et dy mais que les autres figures de la sélection n’ont pas bougé

**Test 1\_4 :** Déplacement d’une sélection qui englobe une autre sélection

On vérifie que les coordonnées de toutes les figures présentes dans les deux sélections ont été incrémentées de dx et dy

**Test 1\_5 :** Déplacement d’une sélection qui ne contient pas entièrement une figure

On vérifie que les coordonnées de la figure ne figurant pas entièrement dans la sélection n’ont pas été incrémentées

**Test 1\_6 :** Déplacement d’une figure contenue dans une sélection qui vient à dépasser les limites de la sélection

On vérifie que les coordonnées de la figure sont correctes, que seule cette figure s’est déplacée et qu’elle est toujours présente dans la sélection

**Test 2\_1 :** Suppression figure

On vérifie que la figure passée en paramètre ne fait plus partie de la liste des figures présentes dans le container

**Test 2\_2 :** Suppression plusieurs figures

On vérifie que les figures passées en paramètre ne font plus partie de la liste des figures présentes dans le container

**Test 2\_3 :** Suppression sélection

On vérifie que les figures présentes dans la sélection passée en paramètre ne font plus partie de la liste des figures présentes dans le container

**Test 2\_4:** Suppression figure faisant partie d’une sélection

On vérifie que la figure passée en paramètre ne fait plus partie de la liste des figures présentes dans le container mais que la sélection existe toujours

**Test 2\_5 :** Suppression d’une sélection suivie de la suppression d’une figure qui faisait partie de cette sélection dans la même commande ?

On vérifie que toutes les figures présentes dans la sélection ont été retirées de la liste des figures présentes dans le container comme si la sélection seule avait été supprimée

**Test 3\_1 :** UNDO création figure

On vérifie que la figure créée précédemment ne fait plus partie de la liste des figures présentes dans le container

**Test 3\_2 :** UNDO déplacement figure

On vérifie que la figure a récupéré les mêmes coordonnées qu’elle possédait avant son déplacement

**Test 3\_3 :** UNDO suppression figure

On vérifie que la figure fait a nouveau partie de la liste des figures présentes dans le container

**Test 3\_4 :** UNDO CLEAR

On vérifie que toutes les figures qui ont été supprimées par le CLEAR font à nouveau partie de la liste des figures présentes dans le container

**Test 3\_5 :** UNDO LOAD

On vérifie que toutes les figures introduites dans le container par le LOAD ont disparues

**Test 3\_6 :** UNDO création sélection

On vérifie que le UNDO s’applique à la commande précédant la création de la sélection car une sélection n’est pas undo-able

**Test 3\_7 :** UNDO suppression figure qui faisait partie d’une sélection

On vérifie que la figure est bien revenue mais qu’elle ne fait plus partie de la sélection

**Test 3\_8 :** UNDO suivi d’une commande puis d’un REDO

On vérifie que l’on ne peut pas REDO

**Test 3\_9:** 21 UNDO

On vérifie que l’on ne peut pas faire le 21ème UNDO

**Test 4\_1 :** SAVE

On vérifie qu’un nouveau fichier a été créé et qu’il contient les commandes associées aux Graphics présents dans le container

**Test 4\_2 :** LOAD

On vérifie que les commandes contenues dans le fichier sont exécutées dans le container

**Test 5\_1 :** Pas le bon nombre de paramètres

On obtient un message d’erreur et un commentaire explicatif

**Test 5\_2 :** Rayon négatif

On obtient un message d’erreur et un commentaire explicatif

**Test 5\_3 :** SAVE avec nom de fichier déjà existant

On obtient un message d’erreur et un commentaire explicatif

**Test 5\_4 :** LOAD avec un nom de fichier inexistant

On obtient un message d’erreur et un commentaire explicatif

**Test 5\_5 :** Nom déjà pris

On obtient un message d’erreur et un commentaire explicatif

**Test 5\_6 :** DELETE dont un des noms n’existe pas

On vérifie qu’aucun des noms valides de la commande n’a été supprimé. On obtient un message d’erreur et un commentaire explicatif

**Test 5\_7:** Création sélection vide

On vérifie que la création d’une sélection vide ne pose aucun problème

**Test 5\_8**: Sélection qui contient un objet + MOVE Selection + UNDO + UNDO + REDO +REDO

On vérifie que l'objet est recrée et refait partie de la sélection

**Test 5\_9** : Création Figure + Sélection qui contient l'objet + DELETE Figure + DELETE Selection + UNDO + UNDO

On vérifie que la Figure a été UNDO et a donc été recréée mais pas la séléction (n'est pas undo-able)

