Heritage, polymorphisme Compte rendu

Mehdi Kitane et Thomas Escure (B3424)

Introduction

Le TP Heritage et polymorphisme en c++ consiste en la création d'un éditeur de formes géométriques sous la forme d'une application console, sans partie graphique.

Généralités

1 Définitions

Notre application gère les cercles, rectangles, lignes, polylignes et objets agrégés. Elle permet aussi de les manipuler au travers de commandes tapées dans le terminal tel que move, delete, undo, redo, load, save, etc..

2 Eclaircissement des points ambigus

Plusieurs points n'ont pas été explicités par le cahier des charges fourni.
Nous avons donc décidé de les éclaircir dans ce paragraphe.
Il est convenu, par contrat, que notre application gère au maximum 1 million d'éléments.
Il est aussi convenu que « l'espace » de notre application, c'est-à-dire la position maximum d'un point d'un des éléments géométriques ne peut dépasser la taille d'un long. De plus, nous ne gérons pas les nombres à virgule, un point ne peut avoir que des coordonnées entières.
Finalement, la commande tapée dans le terminal par l'utilisateur doit respecter la syntaxe. Ainsi, un espace en trop en fin de ligne peut être toléré, 2 ou plus ne le sont pas, tout comme deux espaces entre deux paramètres.

3 Choix généraux

Si une commande entrée est erronée, que ce soit l'entrée d'un type de commande invalide, de l'entrée d'un nombre invalide de paramètre ou l'entrée de paramètres erronés, le programme renvoie une erreur en spécifiant en commentaire d'ou provient le problème.

Nos commandes sont divisées en deux principaux types de commandes, les commandes ayant un effet sur le modèle, qui sont stockées pour pouvoir les annuler, et les commandes n'ayant pas d'effet sur le modèle qui ne sont pas stockées.

Commandes ayant un effet sur le modèle :

- AjouterCercle
- AjouterRectangle
- AjouterLigne
- AjouterPolyligne
- AjouterObjetAgrege
- Suppression
- Deplacement
- Charger
- Clear

Commandes n'ayant pas d'effet sur le modèle :

- Enumeration
- Sauvegarder
- Fermer

Commandes ayant un effet sur le modèle mais pas stockées (gérées autrement : Voir Parenthese : Gestion des undos/redos)

- Undo
- Redo

Nous avons décidé de créer un namespace analyseur qui s'occupe de gérer la validité des commandes entrées par l'utilisateur, de créer les objets et les commandes entrées par les utilisateurs.

C'est un namespace composé de fonctions C qui permettent de gérer et d'analyser toutes les entrées de l'utilisateur et de créer les commandes à partir des paramètres ainsi que les éléments géométriques.

Nous avons choisi de pouvoir laisser le choix à l'utilisateur de faire autant de undo qu'il le veut, et ne pas se limiter à 20 undos/redos.

Lorsque l'on effectue LOAD, les éléments chargés sont ajoutés à notre modèle et ne suppriment pas les éléments déjà présents. Si on effectue, un load avec un fichier erroné ou corrompu, cela n'affecte en rien notre modèle. Effectuer un load avec un fichier contenant un élément ayant le même nom qu'un élément géométrique déjà présent dans notre modèle, annule le chargement du fichier.

La commande SAVE consiste à sauvegarder notre modèle dans un fichier au nom précisé. Il consiste en la sauvegarde des commandes permettant de recréer notre modèle lors du LOAD.

La gestion de la suppression des éléments combinée aux undos et redos nous a confronté à un dilemme. Garder un pointeur vers l'objet supprimé et donc en cas de undo, simplement remettre l'element dans notre structure ou alors recréer à chaque fois l'objet supprimé à chaque undo/redo. La première est plus consommatrice en mémoire tandis que la deuxième demande plus de travail a chaque undo/redo.
Nous avons finalement choisi la première solution. La destruction des éléments géométriques dans notre application au travers du undo, clear et delete est donc faite simplement au travers de la suppression de la référence de l'objet dans la liste des éléments principales. L'objet existe toujours en mémoire, celui ci n'est supprimé que quand on est sur que l'objet n'est plus utilisé, par exemple lors d'un undo puis de la création d'une nouvelle commande.

Les commandes de créations permettent de créer les objets correspondants et de les ajouter à notre modèle. Elles stockent une référence sur l'objet créé en cas de undo et de redo.

Finalement, nous avons décidé de ne pas laisser le choix à l'utilisateur de créer un objet agrégé vide. Cependant, si un objet contenu dans un agrégat est supprimé, il est tout à fait possible d'obtenir un agrégat vide.

Structure de données

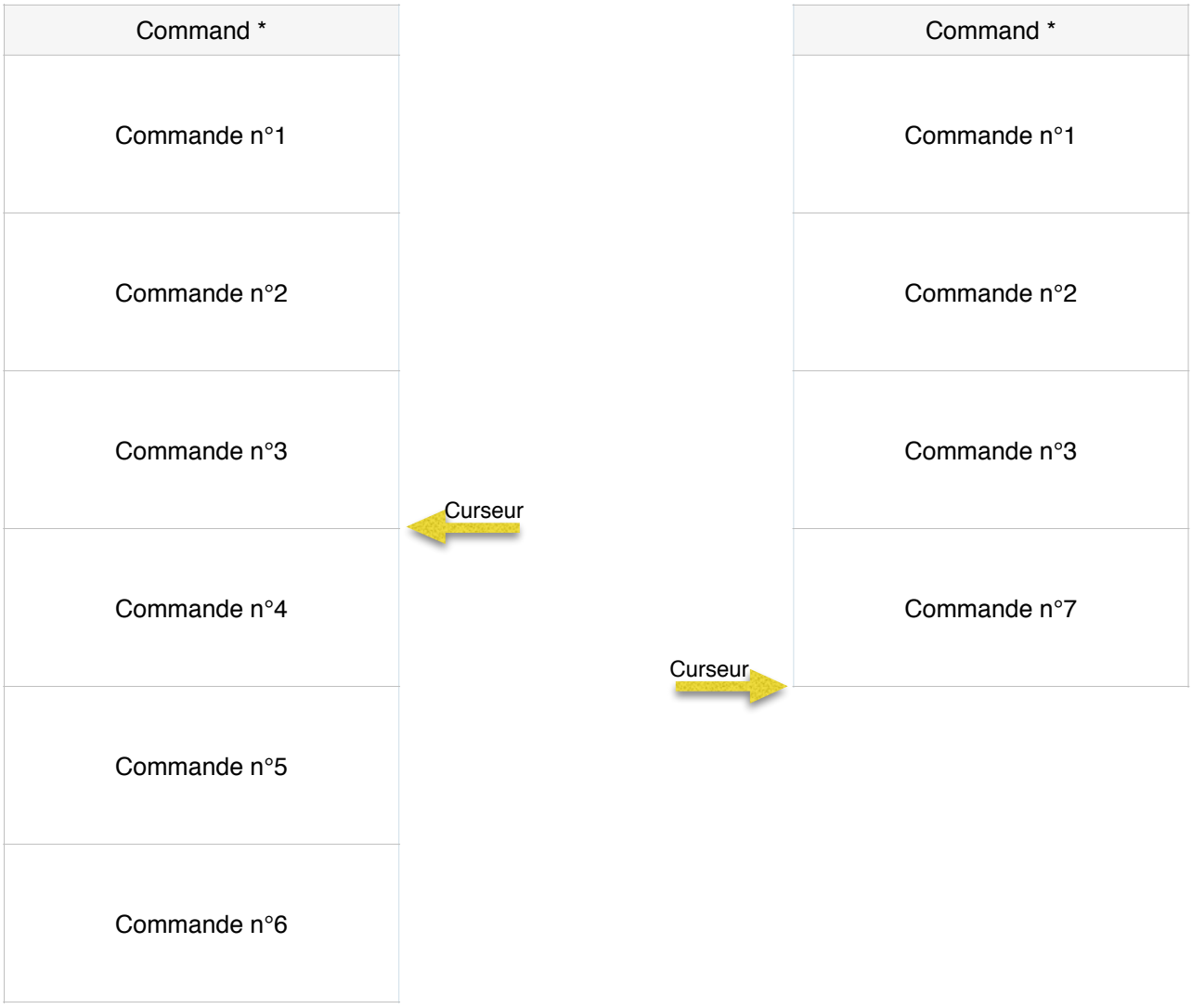
Nous avons deux structures principales qui gèrent notre modèle.
La première est un arbre ayant pour clef le nom de l'élément géométrique et pour valeur un pointeur vers l'élément géométrique.
Les informations nécessaires seront insérées dans la structure représentée ci dessous.
Nous avons décidé d'utiliser un arbre pour rendre la recherche des données plus rapide (complexité en log(n)).

Structure (Arbre)	
Clé nom	**Valeur** Pointeur vers Element Geometrique
cercle1	Cercle1 *
rectangle1	Rectangle1 *
...	...

La deuxième est une liste de pointeur vers nos commandes. Cette structure nous permet de stocker nos commandes effectuées.
Nous avons décidé d'utiliser un vector car nous n'avons pas besoin de rechercher dans nos éléments, les seules opérations à effectuer sont un retour vers la précédente commande dans le cas d'un undo et une annulation du undo dans le cas du redo. Pour ces raisons, nous pensons que l'utilisation d'un vector est la solution la plus adaptée.

Parenthèse : Gestion des Undos/Redos

Notre undo et redo est donc basé sur notre vecteur Historique. La classe qui gère l'archivage des commandes possède aussi un curseur (un itérateur sur le vecteur Historique) qui permet de savoir où l'on se trouve dans notre pile des commandes. Ainsi, celui ci se « déplace » sur notre vecteur Historique et execute la commande où il se trouve. Si on effectue plusieurs undos, le curseur recule d'autant de cases, effectue les undos des commandes mais ne les supprime pas pour pouvoir revenir lors d'un redo. Si on effectue plusieurs undos puis on crée une nouvelle commande, on détruit les commandes se trouvant après le curseur et les objets associés puis on crée notre nouvelle commande.



Exemple avec la création de 6 commandes, puis un 3 UNDO ensuite la création d'une 7 ème commande

Diagramme de classes