TP4 C++ : Héritage et polymorphisme
Conception

Edern HAUMONT and Théo THIBAULT

B3133

February 4, 2016

1 Introduction

This is the document of conception of the application. It is ended by a class diagram which summarize its content.

2 Objects

Objects are instanciated when the corresponding form is created. Active objects are strored in a tree-like structure with their names as keys. Therefore two objects cannot have the same name. We will implement this tree as a map in c++.

2.1 Point

Point is not an object but it composes them. A Point is described by his two coordinates. We need to add a getter and a setter so we can read (to check a hit) and update (to move an object) coordinates of a point.

2.2 Segment

A Segment is made with two points. The hit method has to perform a test of linear regression coeficient. To speed it up it will have a bounding box test first.

2.3 Polygon

A [convex] Polygon is an Object. It will be defined by a dynamic array of points. The convexity test and the hit method will have to compute vectorial products.

2.4 Rectangle

A Rectangle is a polygon only defined by its top-left and bottom-right point. It will redefine some Polygon methods such as the hit method to gain some performance at execution.

2.5 Multi-Objects

Multi-objects, like Reunion or Intersection, are made with other objects. All objects which composed the multi-object are stored in a vector of Objects. Multi-object is a derived class of Object. It is permited that a multi-object is composed by other multi-objects. To do an action on a multi-object (like a move) means do an action on all objects which composed it. However, as it is constructed on copies of other objects, a modification of a Multi-Object does not affect previously

defined objects. Multi-Objects is an abstract class because a multi-object is always an intersection or a reunion.

2.6 Reunion

Reunion is a kind of multi-object. A point is into a Reunion-Object if it hits any object which composed it.

2.7 Intersection

Intersection is a kind of multi-object. A point is into a Intersection-Object only if it hits all objects which composed it.

3 Historic of actions

To implement the ability for the user to undo and redo actions, we chose a quite simple model :

- When the user calls a valid action, it is executed.
- Meanwhile, the command of this action is registered in a list, detailed later.
- The program creates an oppostit command which can be executed if the user decides to undo the action. This command is stored in another list, but with the same iterator.
- Then, when undo and redo are called, we just have to navigate in this list and (re)execute corresponding actions

There are some exceptions to this rule : when the user calls a list, exit, save, hit, undo or redo command, historics are not updated with new entries.

3.1 command historic storage

Commands are strings which are stored in lists (implemented by a double-linked list in the STL). Their "undo counterpart" is stored at the same index but in a different list. We chose to cap the size of both lists to 20, which is sufficient for a normal utilisation of the undo possibility. If an action is made but there are already 20 commands registered, we erase the oldest to make some space to the new.

3.2 Undo & Redo

When Undo and Redo are called, we execute the corresponding command of the historic. Then we update iterators so that we can Undo or Redo the next action of the list. Undo make us execute older and older actions, and Redo the opposite. If a new action is performed as we are not on the top of the historic, we erase all memory of actions from the top to where we are in the historic and we add again new actions at this point.

3.3 Implementation of "Actions"

We chosed not to use Command Design patterns as we wanted to deal ourselves with every aspects of the command call. Therefore, we just stored commands in strings.

4 Persistance

4.1 Load

We will take each element of our list and generate a command which is able to recreate them. We will store these commands in the file asked by the user. We will create temporary elements for short periods to be able to recreate complex elements such as Multi-Objects.

4.2 Save

The program just has to read a input file as it would read user inputs, and restore objects described in the file. It does not erase existing Objects of the draw when load command is called.

- Conception.png