Projet de programmation générique.

Julien Allali : allali@univ-mlv.fr

Animateur Générique.

1 Présentation.

Le but du projet est d'écrire un animateur. Un animateur est un objet gérant un collection d'objets ansi qu'un intervalle de temps entre ces objets. La réalisation finale du projet sera de 6 programmes:

- 3 afficheurs de texte.
- 3 afficheurs d'image.

Ces différents programmes seront détaillés par la suite. Ce projet sera à effectuer par groupe d'au plus 3 personnes et donnera lieu à des soutenances.

2 Le type abstrait Animator.

Le type abstrait (ou concept) d'animateur est décrit par la figure 1. Ce type type abstrait ce rapproche du type Container de la STL. Une "image" sera un couple objet/delai, le delai étant le temps à attendre avant de passer à l' "image" suivante.

3 Implémentation du type abstrait.

Il est demandé de founir au moins 3 types d'animateurs:

- ConstFrameRateAnimator : Un animateur faisant l'affichage à cadence regulière (par exemple 25 images/ secondes): attention à la fonction *scale*.
- StandardAnimator : Un animateur affichant les images suivant le temps défini pour chaque image.
- AdvancedAnimator : Un animateur héritant de *StandardAnimator* mais prenant en compte dans la fonction *play* le temps mis par le *Player* à afficher l'objet en cours.

D'autre versions d'animateurs peuvent être proposées (voire remplacer celle-ci sous réserve de mon accord).

Class T, Class Player
Animator
type interne FrameType type interne DelayType type interne pair<FrameType,DelayType> Frame
void insertBefore(int frameNumber, const Frame &) void insertAfter(int frameNumber, const Frame &) void pushBack(const Frame &) void setDelay(int frameNumber, const DelayType &) Frame & operator[](int) remove(int frameNumber) replace(int frameNumber, const FrameType &) removeAll(const FrameType &) replaceAll(const FrameType &, const FrameType &) scale(int frameIdBegin, int frameIdEnd, const DelayType &) scale(const DelayType &begin, const DelayType &end, const DelayType &) playAnimation()

Figure 1: Le type abstrait Animator.

4 Les autres classes et fonctions.

Outre les animateurs, il vous faudra écrire les classes et fonctions suivantes:

4.1 Pour l'afficheur de texte: T = string.

Les objets de l'animation seront des *string* correspondants à des phrases devant être affichées à l'écran.

- *TextLoader :* contenant une méthode *string load()* demandant à l'utilisateur de saisir une chaîne de caractère et la renvoyant.
- *TextPlayer :* contenant les méthodes:
  - void init() Pour initialiser le player.
  - void show() Pour montrer l'afficheur.
  - void start() Indiquant le début d'une session d'affichage.
  - void setCurrentObject(string) Affichant le texte.
  - void stop() Fin de l'affichage.
  - void hide() Pour cacher l'afficheur.
- *TextEditor :* contenant l'opérateur *void operator()(string* `&`*)* permettant d'éditer un objet texte.
- *TextPreviewer :* contenant l'opérateur *void operator()(const Animator* `&`*)* (l'animateur est un argument template) affichant une prévisulation de la séquence. Celle-ci pourra être par exemple:

  ```
  1.Voici la prem...                        10 sec.
  2.La deuxième p...                         2 sec.
  3.La dernière p...                        20 sec.
  ```

  aide: voir l'exemple à
  `http://www.cplusplus.com/ref/iostream/ios_base/_internal_left_right.html`.

4.2 Pour l'afficheur d'image: T = string.

Pour l'animateur d'image, les objets de l'animation seront des *string* correspondantes au fichier d'une image.

- *ImageLoader*
- *ImagePlayer* : fourni.
- *ImageEditor* : Ici plusieurs solutions sont possibles : changement du fichier de l'image, utilisation de Image Magick++ (http://www.imagemagick.org/www/Magick++/) ....
- *ImagePreviewer :* Là encore plusieurs solutions :)

4.3 Le programme principal.

Le programme principal sera la fonction template :

```
template<class T, class Loader,
         class Player, class Editor,
         template<class > class Previewer,
         template<class,class> class Animator>
int submain(Player *,int argc, char **argv)
{.....}
```

La figure 2 est une vue d'ensemble du problème.

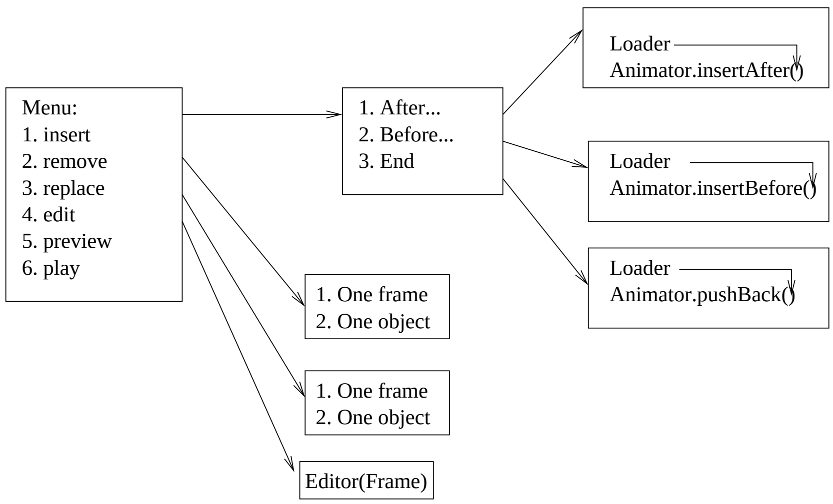

Figure 2: Vue d'ensemble.

5 Les Options.

Les options possibles sont multiples:

- Ajout de fonction au type abstrait *Animator*.
- Ajout au type abstrait *Animator* d'un *OutputIterator* (voir STL).
- Autre type d'animation.
- ...

6 Travail à fournir.

Il vous est demandé de fournir:

- Les 6 programmes décrits plus haut.
- Ces programmes devront compiler à l'aide d'un makefile (ou automake, autoconf ...).
- Un rapport en 3 parties:
  - Présentation générale du sujet, diagramme des classes.
  - Une réflexion sur une solution héritage/classes abstraites au problème.
  - Présentation du code.

La notation tiendra surtout compte de:

- Utilisation des namespaces (obligatoire).
- Commentaire dans le code.
- Warning à la compilation : compilation avec "-Wall" obligatoire.
- Lisibilité du rapport.
- Allocation/désallocation.
- Originalité.

Le Player d'image est disponible à l'adresse:
`http://igm.univ-mlv.fr/~allali/`