Saé 2.01 – Développement d'une application

Lecteur de diaporamas – Dossier d'Analyse et conception

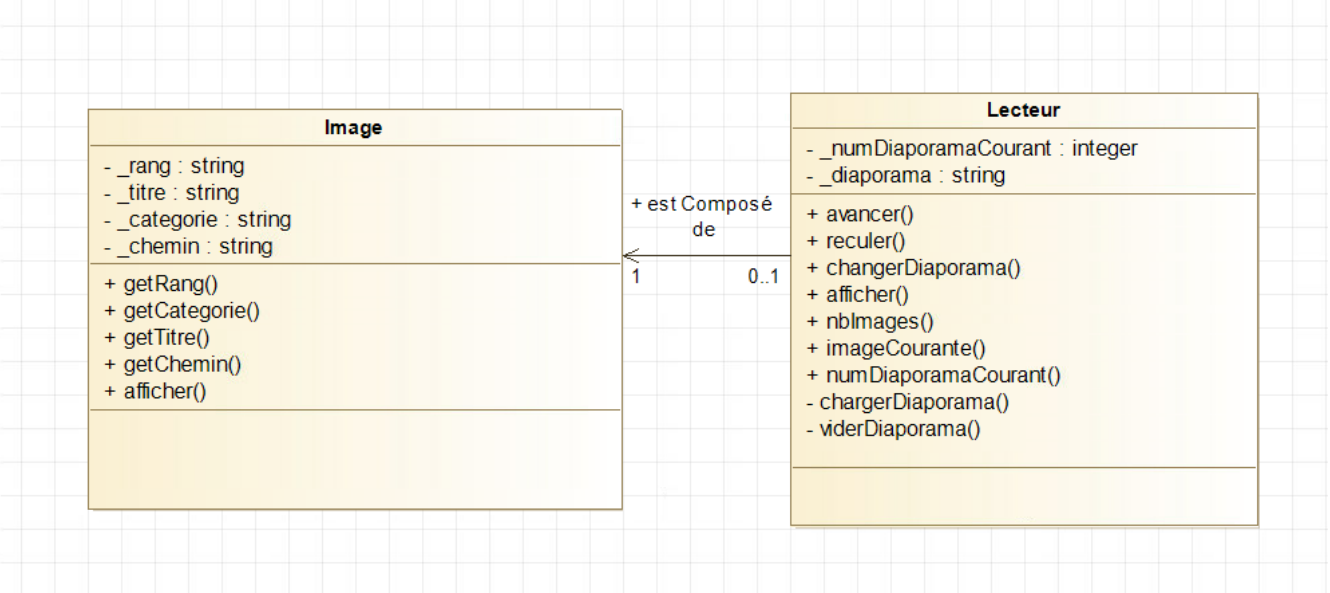
1. Compléments de spécifications externes.

Pas de points flous.

2. Scénarios

Acteur primaire		
Système	**Lecteur de diaporama**	
Acteur secondaire		
Niveau	**A DEFINIR EN FONCTION DES DIAGRAMMES**	
Préconditions		
Opérations	**Client**	**Système**
	L'utilisateur démarre le lecteur de diaporama.	
		Le système affiche le lecteur de diaporama.
1	L'utilisateur clique sur paramètre	
2		Le système affiche les choix disponibles.
3	L'utilisateur clique sur charger diaporama	
		Le système charge le diaporama.
	L'utilisateur clique sur "lancer le diaporama".	
4		Le système affiche le premier slide du diaporama.
5	L'utilisateur défile le diaporama manuellement.	
6		Le système affiche les différents slides.
	L'utilisateur clique sur le bouton "Arrêter le diaporama".	
		Le système arrête le diaporama.
	L'utilisateur clique sur Fichier.	
		Le système affiche les options possibles.
	L'utilisateur clique sur quitter.	
7		Le système se ferme.

3. Diagramme de classe (UML)



(a) Le diagramme de classes UML se focalise sur les classes **métier**, cad celles décrivant les éléments structurants de l'application, indépendamment des éléments d'interface.

(b) Dictionnaire des éléments pour chaque classe

Classe LecteurVue			
Nom attribut	**Signification**	**Type**	**Exemple**
`mode`	mode actuel	`string`	`Automatique`
`rang`			
`timer`			
`EtatLecteur`			

Tableau 2 : Dictionnaire des éléments - Classe xxx

(c) Dictionnaire des méthodes : vous pouvez fournir directement le fichier entête de chaque classe.

Exemple (classe lecteur de la version Console) :

Classe Image

```
class Image
{
public:
    Image(unsigned int pRang=0,
              string pCategorie="", string pTitre="", string pChemin = "");
    unsigned int getRang();
    string getCategorie();
    string getTitre();
    string getChemin();
    void afficher();            // affiche tous les champs de l'image

private:
    unsigned int _rang;          /* rang de l'image au sein du diaporama
                                    auquel l'image est associée */
    string _titre;               // intitulé de l'image
    string _categorie;           // catégorie de l'image (personne, animal, objet)
    string _chemin;              // chemin complet vers le dossier où se trouve l'image
};
```

Classe MainWindow

```
class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
    unsigned int nbImages();    // affiche la taille de _diaporama
    Image* imageCourante();     // retourne le pointeur vers l'image courante
    unsigned int numDiaporamaCourant();

private:
    Ui::MainWindow *ui;
    unsigned _numDiaporamaCourant;   // numéro du diaporama courant, par défaut 0
    Diaporama _diaporama;            // pointeurs vers les images du diaporama
    unsigned int _posImageCourante;  /* position, dans le diaporama,
                                        de l'image courante.
                                        Indéfini quand diaporama vide.
                                     Démarre à 0 quand diaporama non vide */
    Dialog maDlg;

private slots:

    void chargerDiaporama();    // charge dans _diaporama les images du _numDiaporamaCourant
    void viderDiaporama();      // vide _diaporama de tous ses objets image et les delete
    void indiquerModeBtnManuel();
    void indiquerModeBtnAuto();
    void lancerDiaporama();
    void arreterDiaporama();
    void enleverDiaporama();
    void vitesse();
    void aPropos();
    void avancer();             // incrémente _posImageCourante, modulo nbImages()
    void reculer();             // décrémente _posImageCourante, modulo nbImages()
    void changerDiaporama(unsigned int pNumDiaporama);    // permet de choisir un diaporama, 0 si aucun diaporama souhaité
    void afficher();            // affiche les informations sur lecteur-diaporama et image courante
```

Classe dialog

```
#ifndef DIALOG_H
#define DIALOG_H

#include <QDialog>

namespace Ui {
class Dialog;
}

class Dialog : public QDialog
{
    Q_OBJECT

public:
    explicit Dialog(QWidget *parent = nullptr);
    ~Dialog();

private:
    Ui::Dialog *ui;
};

#endif // DIALOG_H
```

(d) Remarques concernant le schéma de classes

1. On ne s'intéresse qu'aux attributs et méthodes métier. Notamment, on ne met pas, pour l'instant, ce qui relève de l'affichage car ce sont d'autres objets du programme (widgets) qui se chargeront de l'affichage. Par contre, on n'oublie pas les méthodes getXXX(), qui permettront aux objets métier de communiquer leur valeur aux objets graphiques pour que ceux-ci s'affichent.
2. On n'a mis ni le constructeur ni le destructeur, pour alléger le schéma.
3. D'autres attributs et méthodes pourront venir ultérieurement compléter cette première vision ANALYTIQUE de l'application. Il s'agira des attributs et méthodes dits DE CONCEPTION nécessaires au développement de l'application.

Version v0 – Version console seule

4. Implémentation et tests

4.1 Implémentation

Liste et rôle des fichiers de cette version :

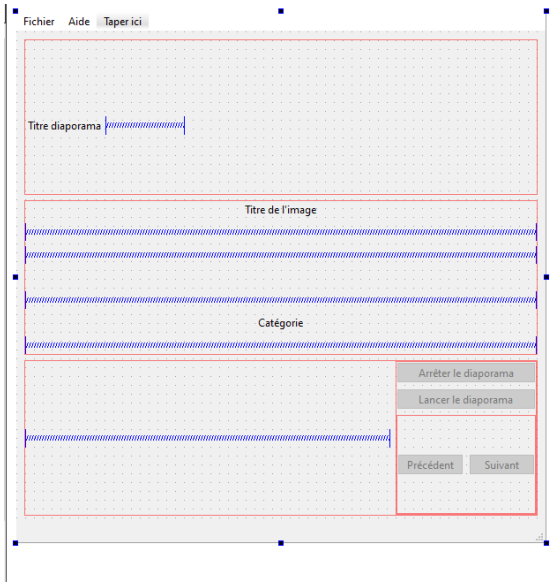
lecteurvue.h	Spécification de la classe lecteurvue
lecteurvue.cpp	Corps de la classe lecteurvue
image.h	Spécification de la classe image
image.cpp	Corps de la classe image
main.cpp	Teste les méthodes de la classe Lecteur
Dialog.h	Spécification de la classe Dialog
Dialog.cpp	Corps de la classe Dialog

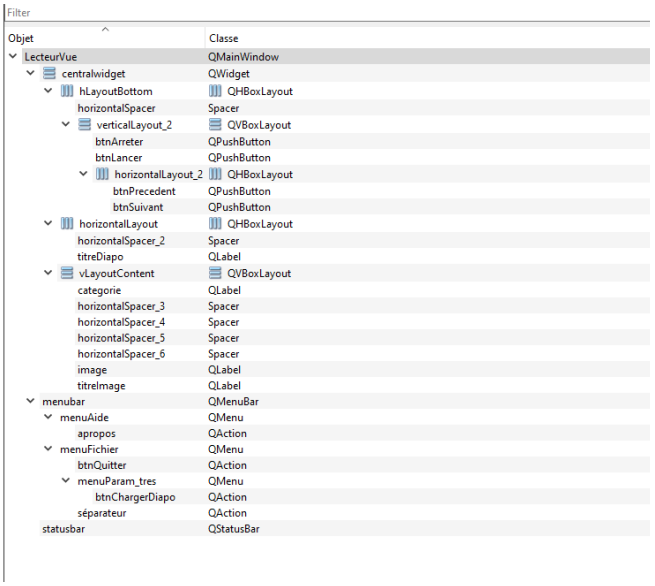
4.2 Test

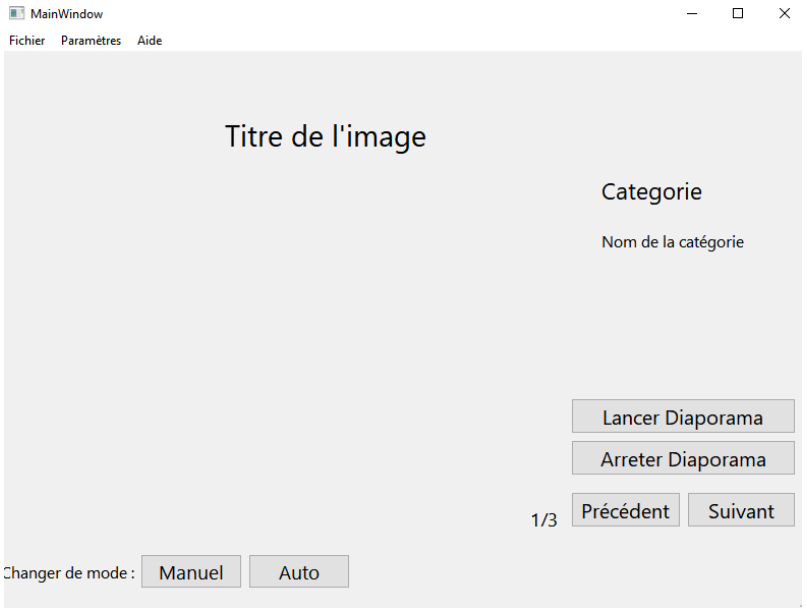
Test avec le programme fourni main.cpp

Version v1 – projet Graphique seul

5. Éléments d'interface







6. Implémentation et tests

6.1 Implémentation

Liste et rôle des fichiers de cette version :

lecteurVue.h	Spécification de la classe graphique Qt contenant l'interface du lecteur de diaporamas
lecteurVue.cpp	Corps de la classe LecteurVue
lecteurvue.ui	Fichier du dessin de l'interface réalisé par QtDesigner
main.cpp	Teste les méthodes de la classe Lecteur

Remarques sur l'implémentation :

Commenter brièvement les choix importants d'implémentation réalisés, ***comme par exemple, les signals/slots***

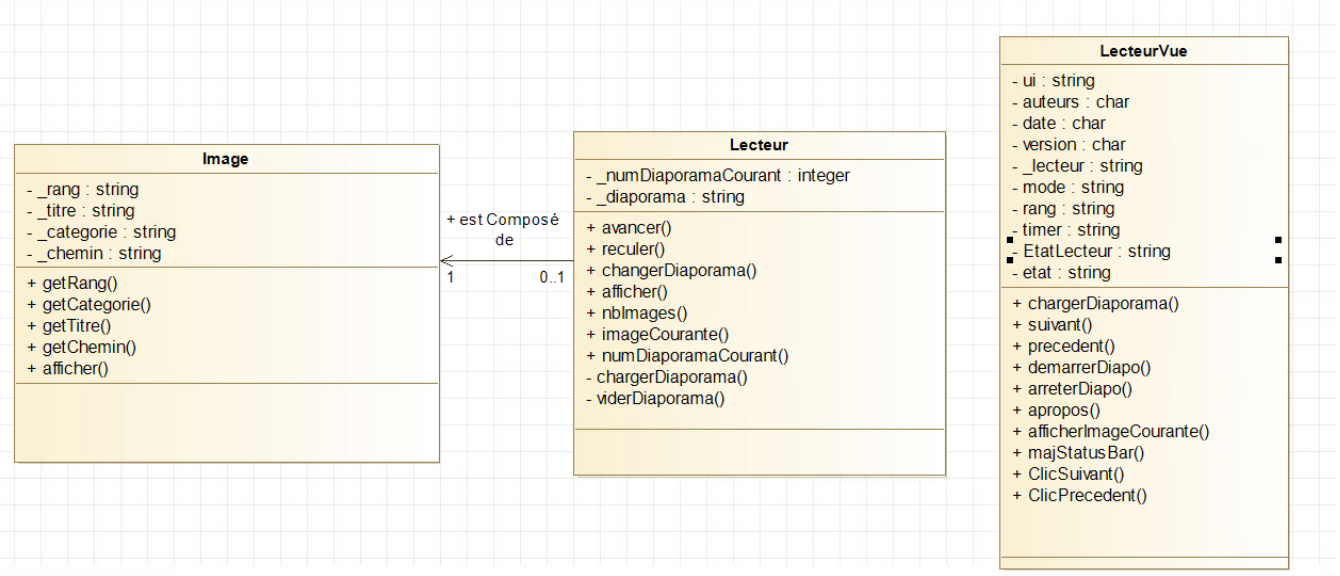
6.2 Test

- *Nous avons testé le programme sur les différents images pour vérifier si cela marcher (c'était le cas)*
- ***Nous avons aussi essayé d'agrandir et rétrécir la fenêtre pour voir si les layouts tenaient bien et dans un premier temps nous avons dû complètement reprendre la disposition des layouts mais à la fin cela marche .***

Version v2 –

7. Diagramme de classes (UML)

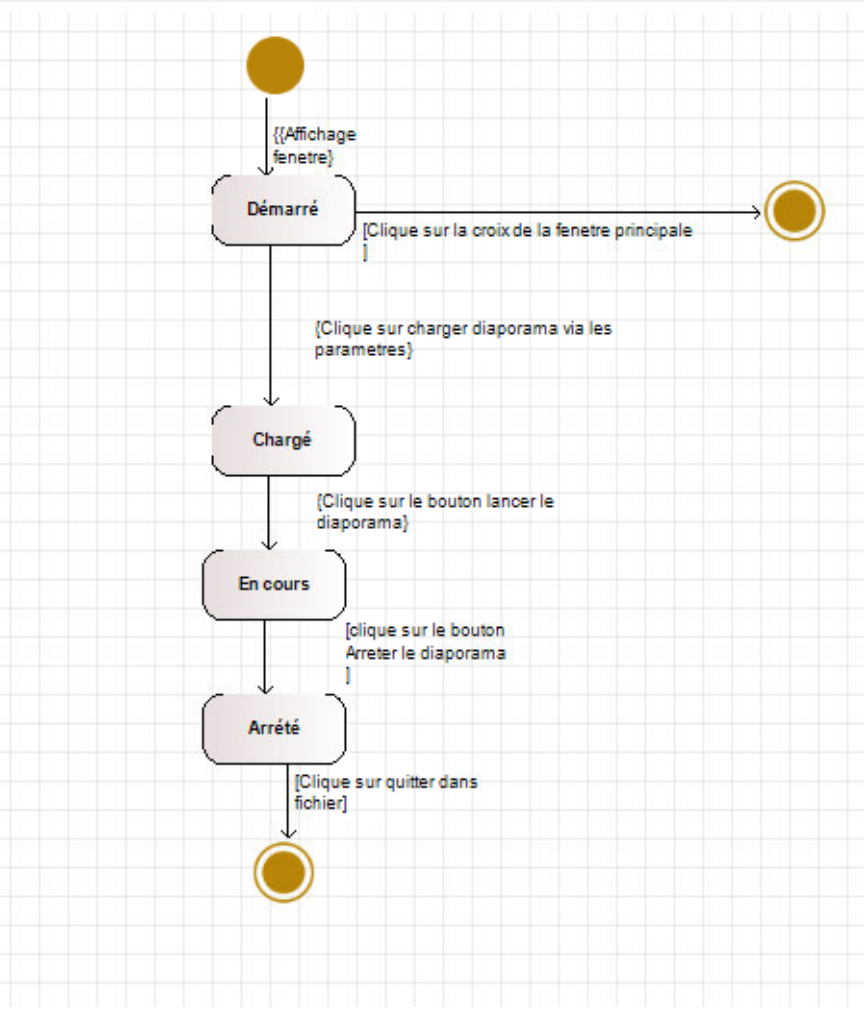
Certains éléments n'ont pas le bon type car nous n'avons pas réussi a mettre d'autres types que string int ou char sur Modelio .



8. Comportement de l'application

7.1 Diagramme états-transitions-actions du lecteur de diaporamas (v2)

Figure 9 : Diagramme états-transitions du lecteur de diaporamas – v2



7.2 Dictionnaire des états, événements et Actions (v2)

Dictionnaire des états du diaporama

EtatLecteur	*Signification*
`manuel`	Les images sont défilées via des boutons .
`automatique`	Les images défilent au bout d'un certain temps toutes seules .Les boutons de défilement sont désactivés
`nonCharger`	Aucun diaporama n'a été chargé

Tableau 2 : États du lecteur de diaporamas – v2

Dictionnaire des événements faisant changer le diaporama d'état

nomEvénement	*Signification*
`Clic sur le bouton chargerDiaporama`	Le diaporama n'est plus en état non chargé
`Clic sur le bouton automatique`	Le diapo passe du mode manuel à automatique

`Clic sur le bouton manuel`	Le diapo passe du mode automatique à manuel

Tableau 3 : Evénements faisant changer le diaporama d'état – v2

Description des actions réalisées lors de la traversée des transitions

nomAction	*Signification*
`Clic sur le bouton chargerDiaporama`	Charge le diapo
`Clic sur bouton suivant ou précédent`	Avance ou recul dans le diaporama
`Clic sur Quitter`	Fermer le lecteur de diaporama

Tableau 4 : Actions à réaliser lors des changements d'état – lecteur de diaporamas v2

7.3 Table T_EtatsEvenementsActions (v2)

Correspondance matricielle du diagramme états-transitions de l'application :
- en *ligne : les **états*** du lecteur de diaporamas (éventuel état de départ d'une transition)
- en *colonne : les **événements*** faisant changer le lecteur d'état (déclencheur d'une transition)
- dans chaque cellule : l'état d'arrivée de la transition + action/traitement à faire + éventuellement garde accompagnant la transition

L'intérêt de cette vue matricielle est qu'elle permet une préparation naturelle et aisée de l'étape suivante de programmation.

9. Implémentation et tests

8.1 Implémentation (v2)

Liste et rôle des fichiers de cette version :

lecteurVue.h	Spécification de la classe graphique Qt contenant l'interface du lecteur de diaporamas
lecteurVue.cpp	Corps de la classe LecteurVue.
lecteurvue.ui	Fichier du dessin de l'interface réalisé par QtDesigner
lecteur.h	Spécification de la classe Lecteur.
lecteur.cpp	Corps de la classe Lecteur
image.h	Spécification de la classe Image
image.cpp	Corps de la classe Image
main.cpp	Permet d'afficher la fenêtre principal

Remarques sur l'implémentation :

10. Bilan

Dépôt Git où trouver le projet complet (les versions réalisées):https://github.com/JohanRouyer/S2.01
Temps global de travail (pour le groupe):environs 30 h
Apprentissages majeurs:Utilisation approfondie de github/git, et amélioration sur QT
Difficultés majeures:Les layout et la V2
Points positifs / négatifs de l'activité:Nous trouvons le dossier d'analyse à remplir un peu compliqué .