UNIVERSITÉ DE NANTES

Université de Nantes

Interface Homme-Machine

Totoscope

Logiciel de rotoscopie avec Qt

Étudiants :
Marie Lenogue
Nicolas Brondin

Encadrant :
M. Languénou

1er Mars 2015

Table des matières

1 Introduction

L'IHM, ou Interface Homme-Machine, est une part importante de toute conception de logiciel possédant une interface graphique. Chaque application ayant un public cible, il est primordial de s'adapter à celui-ci.

En effet, un utilisateur lambda n'aura pas les mêmes habitudes d'utilisation d'un logiciel qu'un utilisateur avancé.

Il est toutefois important de produire une application qui conviendra au plus grand nombre. En fournissant une interface simpliste donnant accès aux fonctionnalités de base d'un simple clic, le logiciel satisfera un utilisateur occasionnel ; quant aux utilisateurs plus avancés, ils s'accorderont à des menus plus enrichis et des interfaces modulables.

C'est dans cette optique qu'il nous a été demandé d'implémenter un logiciel de rotoscopie respectant au mieux les principes de conception d'interface homme-machine.

2 Choix d'implémentation

Totoscope est donc un logiciel de rotoscopie utilisant la librairie Qt. Nous avons donc essayer d'utiliser les fonctionnalités de cette librairie au mieux.

Notre logiciel est composé de plusieurs fenêtres, nous allons décrire les fonctionnalités et les choix opérés pour chaque fenêtre.

2.1 Fenêtre d'ouverture

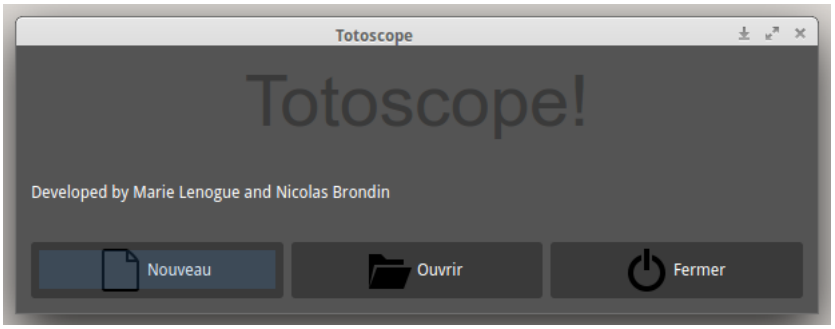


FIGURE 1 – Fenêtre d'ouverture du logiciel

La fenêtre d'ouverture offre plusieurs possibilités à l'utilisateur :

La création d'un nouveau projet

Le choix de cette option, vous ouvre une nouvelle fenêtre permettant d'entrer le nom du projet désiré, la vidéo que l'on souhaite traiter ainsi que les options la concernant.

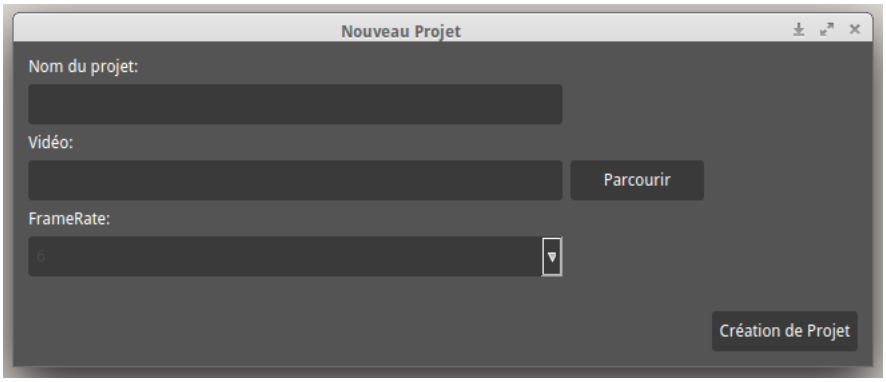


FIGURE 2 – Fenêtre de création d'un projet

Nous avions dans un premier temps envisagé d'y ajouter un système de prévisualisation de la vidéo choisie. Cette possibilité n'a pas été retenu pour le rendu final, en effet nous pouvons supposer que l'utilisateur connaît déjà la vidéo sur laquelle il veut travailler. Les informations fournies alors ne seraient pas d'une grande utilité.

L'ouverture d'un projet précédemment créé

L'ouverture d'un projet déjà existant vous ouvrira une fenêtre de navigation. Il suffit alors à l'utilisateur de sélectionner le fichier .tots qui concerne le projet qu'il souhaite modifier.

La fermeture de l'application

Cette action met tout simplement fin à l'exécution de l'application.

2.2 Fenêtre principale

Comme décrit dans notre premier rendu, nous avons décidé de rendre les fonctionnalités principales les plus accessibles possible.

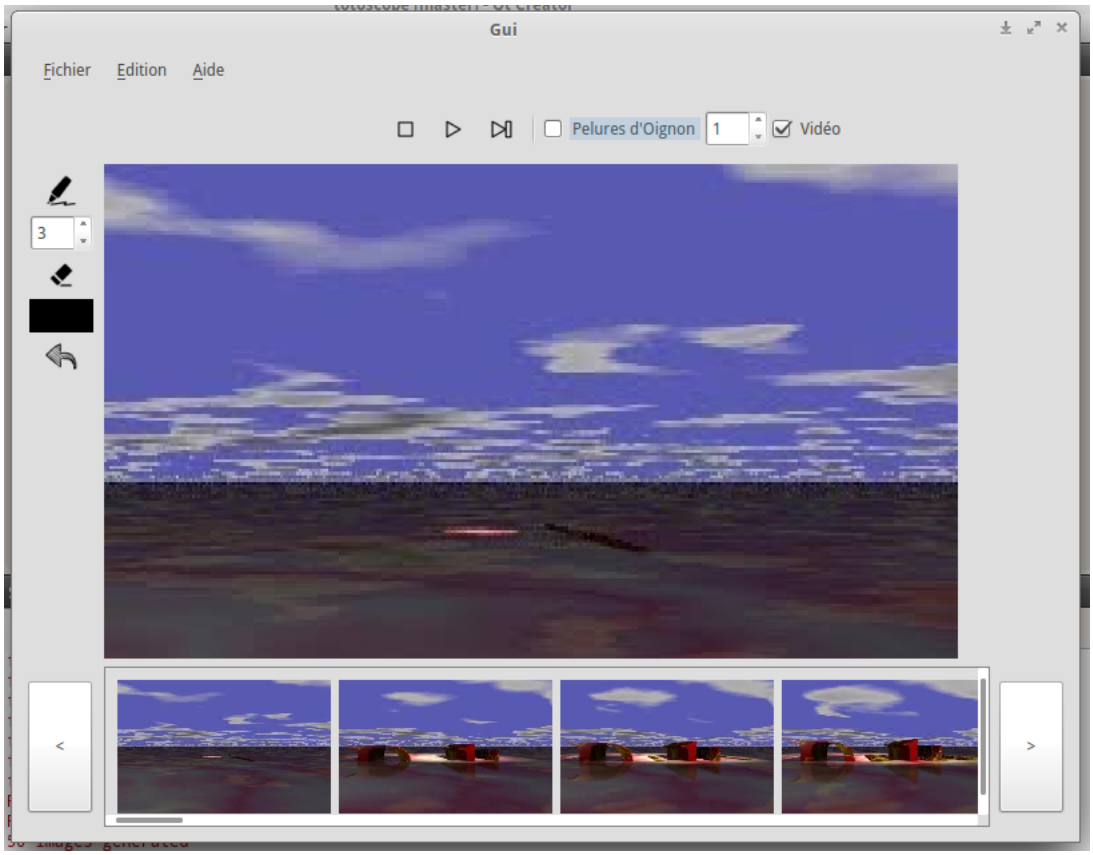


FIGURE 3 – Fenêtre principale de Totoscope

Notre interface bien que simple et épurée, permet d'effectuer les actions essentielles à l'utilisation du logiciel.

La fenêtre peut être divisée en cinq parties distinctes.

2.2.1 La barre de menu principale

Elle répertorie toutes les actions possibles de l'application. Elle est fractionnée en trois sous-menus :

Fichier
: Ce menu contient des actions telles que créer un nouveau projet, ouvrir un projet déjà existant, enregistrer ou exporter le travail courant ou encore fermer le projet et le logiciel.

Édition
: Le menu Édition présente les actions fonctionnelles de l'application comme le fait d'annuler la dernière action, ou de la rétablir, l'activation ou non des pelures d'oignon et de la vidéo en arrière plan, et les actions concernant la lecture des dessins déjà effectués.

Aide
: Ce menu quant à lui, ne contient que l'action A Propos qui donne accès aux informations concernant l'application.

2.2.2 La barre de contrôle de la vidéo

Elle comprend les actions concernant la vidéo telles que les boutons lecture/pause, stop ou encore image suivante ; elle contient aussi les boutons d'activation des pelures d'oignon (ainsi que leur nombre) et de la vidéo en arrière plan.

2.2.3 La barre latérale de dessin

La barre latérale regroupe les différentes actions concernant le dessin comme le crayon ou la gomme, un champ pour leur affecter une taille, la palette de couleur ou encore un bouton pour annuler la dernière action.

2.2.4 La zone de dessin

Elle occupe la part la plus importante de la fenêtre principale puisqu'elle correspond à la fonction première de notre logiciel.

2.2.5 Les miniatures

Les miniatures de la vidéo découpée sont accessibles en partie basse de la fenêtre. Elles permettent une meilleure navigation entre les différentes partie de la vidéo sans encombrer la fenêtre principale.

2.3 Fenêtre d'exportation

La fenêtre d'exportation se divise en deux onglets distincts :

L'exportation au format image

Il comprend les options pour l'exportation des dessins au format image. Chaque dessin sera sauvegardé dans un fichier à part dans le dossier mentionné, il sera alors possible de les manipuler grâce à d'autres logiciels.

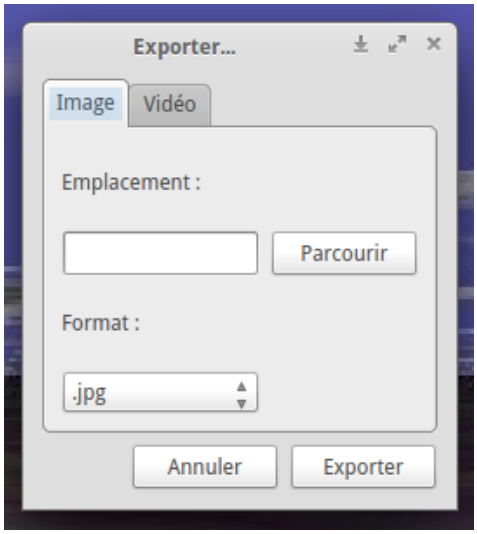


FIGURE 4 – Fenêtre d'exportation au format image

L'exportation au format vidéo

Cet onglet regroupe les options de l'exportation au format vidéo comme l'emplacement du dossier de destination de la vidéo, le codec de la vidéo ou encore son nom.

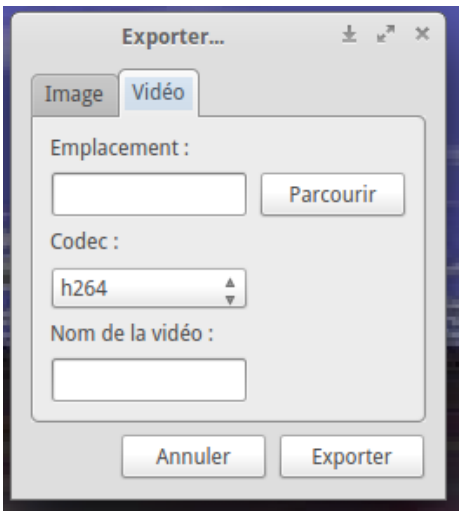


FIGURE 5 – Fenêtre d'exportation au format vidéo

3 Interactions entre les classes

3.1 Patron de conception

Pour le patron de conception, il était évident d'utiliser un modèle en MVC.

Le plus compliqué dans un patron comme celui-ci est d'en faire une implémentation propre avec une réelle séparation des différents composants, tout en gardant une architecture facilement compréhensible.

3.1.1 L'interface

Pour l'interface, nous avons choisi de découper chaque fenêtre (à l'exception des pop-ups) dans une nouvelle classe. Ce qui permet de ne pas surcharger la classe de la fenêtre principale, tout en gardant un nombre de classes raisonnable pour notre application.

Nous avons aussi choisi de découper deux grosses parties de la fenêtre principale qui sont la zone de dessin ainsi que les miniatures qui sont donc des classes à part, héritantes de classes graphiques Qt.

3.1.2 Le modèle

Toutes les classes du modèle sont des classes qui héritent de la super-classe QObject car cela permet d'utiliser la technologie des signaux/slots et ainsi rendre cette partie asynchrone.

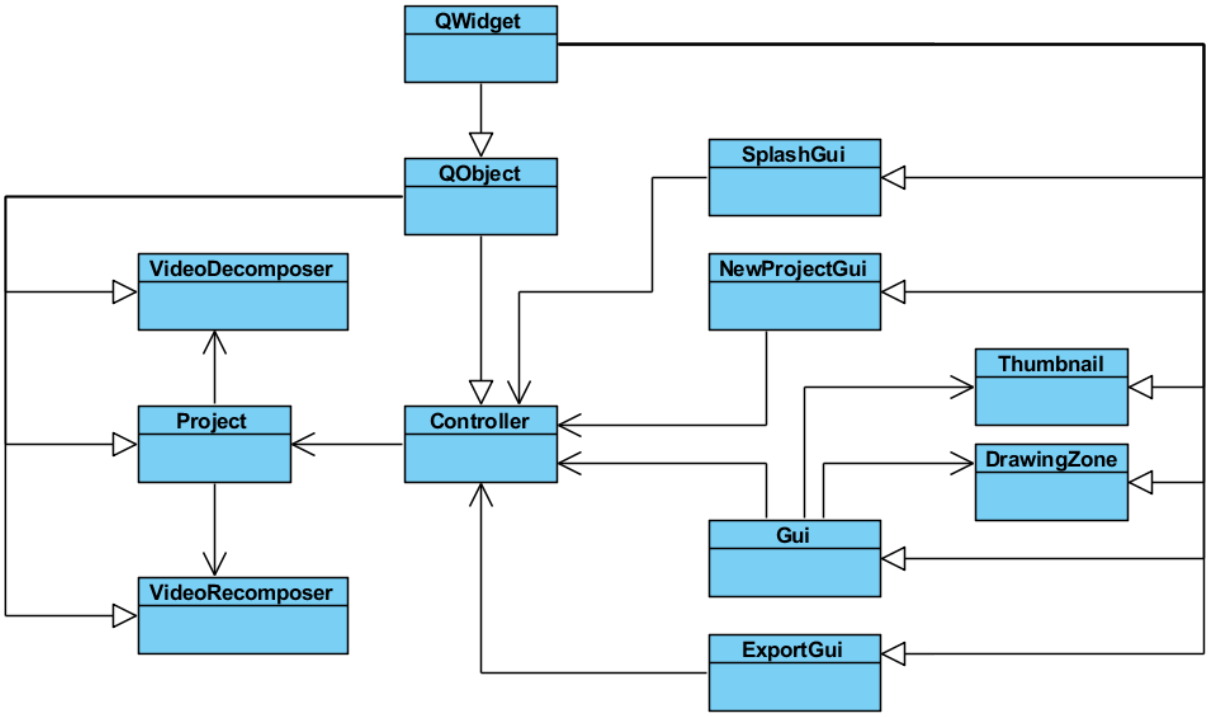
Pour les interactions avec le système, nous avons utilisé la classe QProcess afin que ces interactions soient faciles à mettre en place et à observer.

3.1.3 Le contrôleur

Le contrôleur est la pièce maîtresse de ce modèle, c'est réellement lui qui fait le lien entre le modèle et l'interface (qui ne discutent JAMAIS directement ensemble).

Il hérite de QObject pour avoir accès aux signaux/slots et n'effectue que très peu d'opérations, il est principalement présent pour coordonner les actions de l'utilisateur avec les réactions de l'application.

3.2 Diagramme de classe



4 Conclusion

Bien que notre application soit fonctionnel, il est certain qu'un délai supplémentaire lui aurait été bénéfique. Nous avons été contraint d'alléger certaines fonctionnalités qui auraient pu être plus travaillées.

Toutefois, notre logiciel est conforme au paper prototype que nous avions conçu et répond à toutes les fonctionnalités demandées.

Ce fut très instructif de concevoir une application depuis l'interface jusqu'au cœur, quand l'inverse est plus dans nos habitudes.