

**IFT-3100: Infographie**

**Projet de session (TP#1)**

présenté à

**Philippe Voyer**

|  |  |
| --- | --- |
| *matricule* | *nom* |
| 910 101 463 | Alexandre Gagnon |
| 111 147 051 | Samuel Parent |

Université Laval

11 mars 2018

**Sommaire**

L’objectif du projet de session est de développer une application qui permet de construire, éditer et rendre des scènes visuelles. Le projet de session a été divisé en deux parties. La première partie du projet porte sur les 5 catégories suivantes : l’image, les dessins vectoriels, les transformations, la géométrie et les textures. Chaque catégorie a 5 sous-catégories possibles et nous devons choisir d’en implémenter au moins 3 pour chaque catégorie. Pour notre projet, nous avons décidé d’implémenter les sous-catégories de manière rudimentaire pour commencer. Au fur et à mesure que le projet avance, nous pouvons ajuster les aspects de l’application pour être plus concentrés sur une idée précise. Avec cette méthode, nous nous laissons la porte ouverte à beaucoup plus d’implémentation possible des catégories sans avoir à nous inquiéter de la manière qu’ils vont intégrer notre application. Une fois que plusieurs aspects vont être complétés, nous allons pouvoir décider où nous voulons aller avec les aspects suivant pour créer une application avec une idée et un but claire.

Pour la première partie du projet, les aspects que nous avons décidé d’implémenter sont les suivants : Bla.

Avec ces aspects implémenter, notre application…

**Interactivité**

[todo]

**Technologie**

[todo]

**Compilation**

[todo]

**Architecture**

Pour nous, la définition de l’architecture logicielle est de concevoir et structurer un programme d’une telle manière que nous pouvons améliorer :

* La qualité
* La conception
* L’évolution
* L’entretien

Nous avons essayé de créer une architecture qui rend ces aspects faciles à gérer. Pour l’évolution de l’application, nous avons voulu créer une architecture qui nous laisse ajouter des éléments à notre projet sans difficulté. C’est pour cette raison que nous avons divisé les différentes composantes de notre projet en différente classe. Ceci facilite grandement l’ajout d’un nouvel aspect sans avoir à chercher à travers tout le code pour trouver un endroit à le placer. Il faut simplement créer une nouvelle classe pour notre aspect, et lui donner les bons héritages, ou l’ajouter dans une classe qui remplit déjà un rôle similaire.

Les sous-classes sont aussi très utiles pour l’entretien de l’application. Si un problème est soulevé, nous avons juste à trouver la classe où se trouve le problème et commencer à le corriger de là. Même si cette erreur cause des problèmes dans d’autres classes, il est assez facile de tracer un chemin à partir de l’origine du problème en analysant les héritages de classe qui sont impliqués et déduire les problèmes qui sont possibles à partir de ça.

Pour ce qui est de la qualité, nous voulions implémenter une architecture qui n’était pas trop complexe, pour ne pas se perdre dans le code. Si une architecture a un mauvais design et une mauvaise organisation, il est très difficile de se retrouver dans le code. La manière dont le nôtre est implémenté rend la tâche de retrouver des blocs de code assez facile, étant donné que tout est séparé en sous-aspects. Avec les héritages des classes, il est assez facile de trouver d’où vient une certaine fonction et ainsi, le code est facile à comprendre.

**Fonctionnalités**

La première section du projet de session visait à nous faire travailler sur les aspects d’image de notre projet. Nous avons choisi implémenter les aspects suivant.

* Il est possible d'importer des fichiers images et de les afficher dans une scène sous une forme ou une autre (1.1) : Le projet offre deux manières d’importer une image : importer à partir d’un bouton d’importation ou « drag and drop » d’une image dans la scène. Si le bouton d’importation est sélectionné, un fichier de dialogue ouvre et donne l’option à l’utilisateur de choisir son fichier image. Quand une image valide est choisie, les paramètres de hauteur et de largeur sont mis à jour dans l’objet « ofImage » et l’image peut être placé et affichée. Le « drag and drop » est comme n’importe quelle autre version de cette mécanique. Quand l’image est « dragger » et « dropper » dans la fenêtre de l’application, elle est ajoutée où la souris de l’utilisateur la laisser tomber.
* Il est possible d’exporter des rendus d’une scène dans des fichiers images (1.2): Il a un bouton pour commencer l’exportation des images. Quand il est activé, un fichier de dialogue ouvre qui donne l’option à l’utilisateur de choisir où il veut sauvegarder son fichier image. L’utilisateur doit donner un nom au fichier pour qu’il puisse le sauvegarder et il va être sauvegardé avec l’extension « .PNG ». Un objet « ofImage » est créé pour garder l’image prise par la capture d’écran (.grabScreen) et la sauvegarder à l’endroit choisi par l’utilisateur.
* Il est possible de sélectionner une couleur parmi un ensemble de couleurs et de l’assigner à un élément visuel (1.4): Puisqu’il est possible de placer des éléments visuels dans la scène de l’application, nous avons donné l’option à l’utilisateur de choisir la couleur de ces éléments. En utilisant « ofxGUI », nous avons créé un panneau qui contrôle le niveau des couleurs RGB. Si, avant de placer un élément visuel dans la scène, le niveau d’une des couleurs est changé, l’élément va prendre cette couleur. Tant que la couleur n’est pas modifiée, les objets placés vont continuer d’avoir la couleur qui avait été sélectionnée.

La deuxième partie du projet était fixé sur les dessins vectoriels. Pour cette partie, nous avons implémenté les aspects suivants.

* Il existe au moins 5 représentations visuelles différentes du curseur dessinées à partir de primitives vectorielles (2.1) : nous avons créé 5 différents cas où le curseur peut avoir une représentation visuelle différente. Quand le bouton « select tool » est actif, nous pouvons changer le curseur. Il y a deux moyens de faire : peser les touches 1-5 sur le clavier ou cliquer sur les options qui sont visibles sur l’écran grâce à un panneau créé avec « ofxGUI ». En gros, une fois qu’un curseur autre que le curseur original est choisi, nous le cachons avec « ofHideCursor ». En prenant les coordonnées du curseur, nous dessinons de nouveaux curseurs à partir de ligne ou de primitives vectorielles et on les donne le même fonctionnement que le curseur d’origine.
* Il est possible de modifier de manière interactive la valeur des outils de dessin vectoriel tel que l’épaisseur des lignes de contour, la couleur des lignes de contour, la couleur des zones de remplissage et la couleur d’arrière-plan de la scène (2.2) : il est possible de changer les couleurs des lignes qui sont placées dans la scène, comme nous avons mentionné dans 1.4. Il est aussi possible de changer l’épaisseur des lignes avec un panneau en utilisant le même principe de « slider » que les couleurs. De plus, si nous créons une forme vectorielle avec ces lignes, il est possible de changer la couleur de remplissage aussi. Comme avec les lignes, nous avons un panneau « ofxGUI » avec les valeurs RGB et quand les « slider » sont bougé, la couleur de remplissage est modifiée. Pour changer la couleur d’arrière-plan, il faut être dans la fenêtre du bouton « select tool ». Le principe pour changer la couleur est le même qu’avec les deux autres aspects.
* Un ou des éléments d’interface graphique offrent de la rétroaction informative visuelle à l’utilisateur et des contrôles interactifs pour influencer les états de l’application (2.5) : cet aspect est lié de près avec 2.2, car les contrôles sont évidemment interactifs (il faut glisser les contrôles pour changer la couleur). Il est aussi possible de voir le niveau de chaque élément de couleur RGB(A), l’épaisseur de la ligne, le curseur qui est actif, la couleur de remplissage et la couleur d’arrière-plan.

**Ressources**

[todo]

**Présentation**

[todo]

[todo]