

**Rapport de Laboratoire**

|  |  |
| --- | --- |
| **Étudiant(s)** | Ariane Beaudry Betournay  Vanessa Baquero |
| **Code(s) permanent(s)** | BEAA13529208  BAQV29529200 |
| **Numéro d’équipe** | Groupe C |
| **Cours** | GTI 411 |
| **Session** | Été 2019 |
| **Groupe** | 01 |
| **Chargé(e) de laboratoire** | Ted Julien Tchinde-Fotsin |
| **Date** | 19 juin 2019 |
| **Étudiant(s)** | Ariane Beaudry Betournay  Vanessa Baquero |
| **Code(s) permanent(s)** | BEAA13529208  BAQV29529200 |

Introduction

Le premier laboratoire nous a permis de nous familiariser avec les conversions des couleurs et les différentes classes du code qui nous était fourni. Pour le deuxième laboratoire, nous allons continuer avec le même code, mais il sera adapté aux nouveaux requis.

Pour commencer, il faudra implémenter les méthodes pour le remplissage intérieur des figures et pour le remplissage des contours. Pour le remplissage, il faudra valider que les pixels voisins sont de la même couleur et par la suite il faudra la modifier par celle préalablement sélectionnée. Pour ce qui est de la bordure, c’est le même principe, mais on arrête lorsqu’on atteint la bordure. Ces méthodes seront utilisées sur des images 2D qui seront importées dans le programme et les couleurs seront sélectionnées grâce aux sélecteurs de couleurs RGB et HSV. Pour terminer, il faudra également faire le calcul de tolérances entre deux couleurs.

Dans ce rapport, vous allez retrouver notre analyse, notre démarche de conception, les détails de notre implémentation et la discussion. Vous allez également retrouver un manuel de l’utilisateur et la conclusion.

# 

# Analyse

Nous devons implémenter des méthodes pour faire la modification des couleurs de certaines régions de l’image. Selon l’option choisie, le changement de couleur sera traité différemment. Il faudra également implémenter le principe de seuils pour limiter ou augmenter la sensibilité des couleurs semblables.

Les deux principaux concepts de modification des couleurs pour ce laboratoire sont les remplissages et le calcul des tolérances entre les couleurs. Avant de commencer à réfléchir à notre solution, nous avons étudié les principes qui n’avaient pas encore été couverts dans le cours théorique et par la suite nous avons établi un plan d’action.

Concepts importants:

* **Région**: La région de l’image est représentée par l’ensemble de pixels de la même couleur qui sont les uns à côté des autres.
* **Bordure**: Ensemble de pixels de la même couleur qui entourent un autre ensemble de pixels de la même couleur.
* **4 voisins**: Un type de voisinage qui prend en compte le voisin à droite, à gauche, en bas et en haut du pixel de référence.

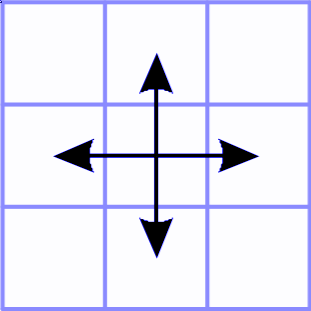


Image1: Principe des 4 voisins

* **Seuil**: paramètre de référence utilisé dans le laboratoire comme référence de ressemblance entre deux couleurs. Le système de couleurs HSV sera utilisé.

Pour ce qui est de l'approche que nous allons favoriser, nous allons utiliser les coordonnées du point sélectionné à l’écran comme point de départ et par la suite nous allons réaliser une des deux méthodes que l’utilisation choisira à l’aide d’un bouton, soit la méthode de remplissage intérieur ou de bordure. Une option pour indiquer le seuil de ressemblance entre les pixels sera également utilisée.

# Conception

Pour ce laboratoire, notre conception se limite à l’ajout des méthodes nécessaires dans une seule classe. ImageLineFiller est la seule classe qui sera modifiée afin de réaliser les transformations nécessaires aux images.

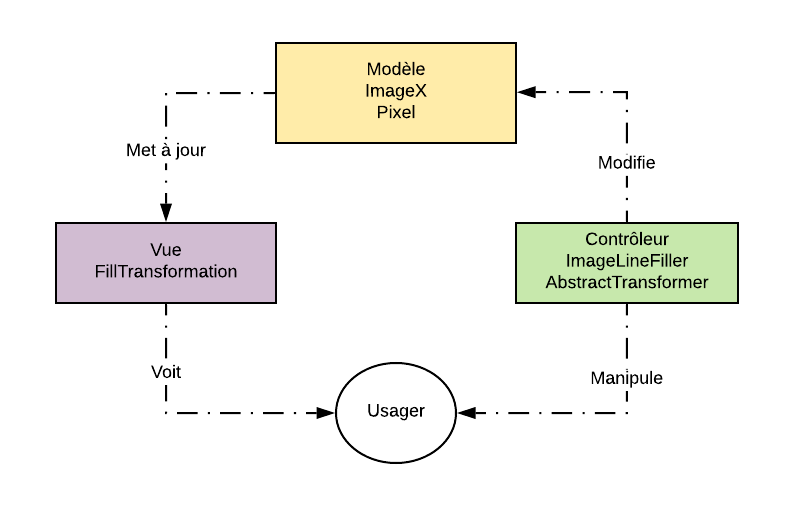


Diagramme 1: Modèle, vue, contrôleur avec les classes associées

Tel que nous avons vu dans le premier laboratoire, l'application suit le patron MVC, modèle vue contrôleur et grâce à ce patron nous allons minimiser les modifications qui seront faites au code par modifiant seulement le contrôleur qui nous intéresse. Pour ce qui est de la classe vue, FillTransformation, nous allons l’utiliser telle quelle.

Dans le diagramme UML suivant, on représente les modifications que nous avons apportées au code dans une seule classe et les relations avec les autres clases. De plus, on fait un rappel de couleur avec le premier diagramme pour différencier les différents packages, modèles, vues ou contrôleurs.

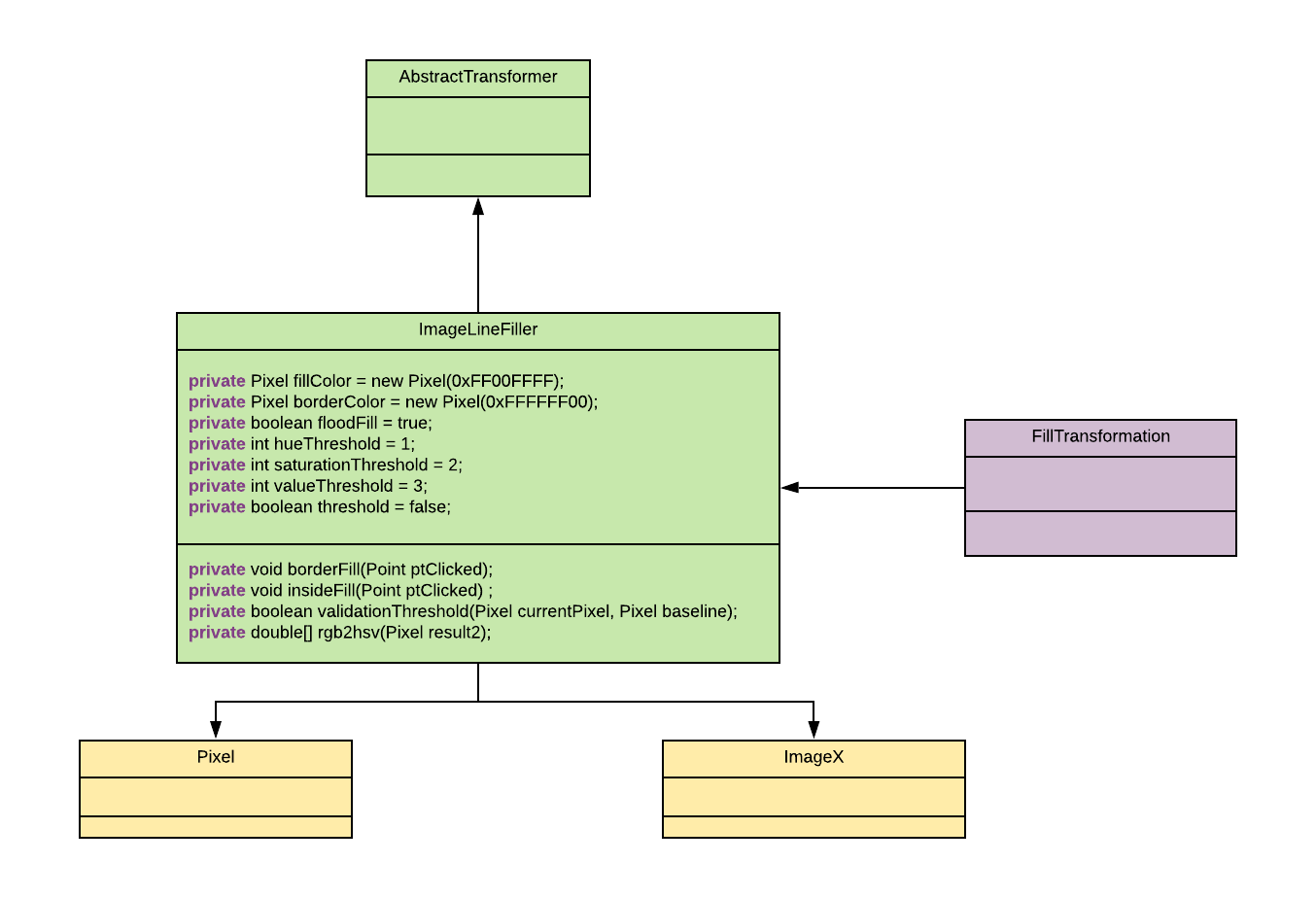


Diagramme 2: Diagramme UML

# Implémentation et algorithmes

Nous avons utilisé comme référence le code qui avait été fourni comme code de départ pour le remplissage d’une ligne horizontale de couleur et nous l’avons modifié afin de nous permettre d'implémenter les différentes méthodes de remplissage. Pour commencer, nous avons modifié la méthode lineFiller pour la transformer en insideFill et pour y arriver, nous avons ajouté le remplissage du voisinage grâce aux 4 voisins. Si le point de part est plus ou égal à zéro, et s’il est plus petit que la taille de l’image alors valider si le seuil est activé et par la suite procéder au changement de couleur des pixels grâce à la méthode setPixel. Une fois que le pixel d’origine est modifié avec la nouvelle valeur, il faudra avancer dans l’image afin de modifier tous les pixels. Pour avancer dans l’image on va aller chercher le pixel, de droite, de gauche, du haut et finalement celui du bas et ainsi de suite. Une fois que tous les pixels de la même couleur sont modifiés, il faudra arrêter afin de ne pas modifier les autres régions.

|  |
| --- |
| while (!stack.empty()) {  Point current = (Point)stack.pop();  if (0 <=x && x < currentImage.getImageWidth() & !currentImage.getPixel( x,y).equals(fillColor)) {  //Validation de la région thresholdHSV = validationThreshold(currentImage.getPixel(x,y), baseLinePixel);  if(thresholdHSV == true) {  currentImage.setPixel(current.x, current.y, fillColor);  // Méthode des 4 voisins  Point nextLeft = new Point(x-1,y);  Point nextRight = new Point(x+1,y);  Point nextDown = new Point(x,y-1);  Point nextUp = new Point(x,y+1);   stack.push(nextLeft);  stack.push(nextRight);  stack.push(nextDown);  stack.push(nextUp);  }   } } |

Maintenant que notre méthode de remplissage intérieur a été complétée, nous avons commencé l'implémentation de la méthode de remplissage de bordure. Les deux méthodes se ressemblent beaucoup, mais nous avons certaines différences. Une nouvelle méthode a été créée et a été ajoutée à la méthode qui traite les clicks de la souris.

|  |
| --- |
| //On verifie que le point d’origine est plus petit que la taille de l’image  if (0 <=x && x <currentImage.getImageWidth() && 0 <= y && y < currentImage.getImageHeight()) {  currentImage.beginPixelUpdate();  //Choix fait par l’utilisateur pour le type de remplissage  if (floodFill == true) {  insideFill(ptTransformed);  } else if (floodFill == false) {  borderFill(ptTransformed);  }  currentImage.endPixelUpdate(); return true; } |

Si on utilise la méthode de remplissage par bordure et que le point de part est plus ou égal à zéro, et s’il est plus petit que la taille de l’image alors valider si le seuil est faux et par la suite procéder au changement de couleur des pixels grâce à la méthode setPixel. Une fois que le pixel d’origine est modifié avec la nouvelle valeur, il faudra avancer dans l’image afin de modifier tous les pixels. Pour avancer dans l’image on va aller chercher le pixel, de droite, de gauche, du haut et finalement celui du bas et ainsi de suite. Une fois que tous les pixels de la même couleur sont modifiés et que nous arrivons à la bordure, il faudra arrêter afin de ne pas modifier les autres régions.

|  |
| --- |
| while (!stack.empty()) {  Point current = (Point) stack.pop();  if (0 <= x && x < currentImage.getImageWidth() && 0 <= y && y < currentImage.getImageHeight()&& !currentImage.getPixel(x, y).equals(borderColor)) {  thresholdBorder = validationThreshold(currentImage.getPixel(x,y), baseLinePixel);  if (thresholdBorder == true) {  currentImage.setPixel(current.x, current.y, fillColor);  //Méthode des 4 voisins  Point nextLeft = new Point(x-1,y);  Point nextRight = new Point(x+1,y);  Point nextDown = new Point(x,y-1);  Point nextUp = new Point(x,y+1);    stack.push(nextLeft);  stack.push(nextRight);  stack.push(nextDown);  stack.push(nextUp);  }  } } |

Pour les validations de seuil, nous avons une méthode pour faire ce calcul. On veut utiliser la méthode à plusieurs endroits dans la même classe et c’est pour cette raison que nous avons décidé de créer une méthode séparée. Dans cette méthode, nous avons deux valeurs de référence et on compare les pixels pour valider qu’ils se situent à l'intérieur de l’intervalle. Une fois, cette validation est faite, la méthode retourne un booléen.

|  |
| --- |
| private boolean validationThreshold(Pixel currentPixel, Pixel baseline) {  boolean thresholdCheck = false;  double hsvcurrentPixel[] = rgb2hsv(currentPixel);  double hsvbaseline[] = rgb2hsv(baseline);  int h1,s1,v1,h2,s2,v2;    h1 = (int)(hsvbaseline[0]- hueThreshold);  s1 = (int)(hsvbaseline[1]- saturationThreshold);  v1 = (int)(hsvbaseline[2]- valueThreshold);  h2 = (int)(hsvbaseline[0]+ hueThreshold);  s2 = (int)(hsvbaseline[1]+ saturationThreshold);  v2 = (int)(hsvbaseline[2]+ valueThreshold);   if(hsvcurrentPixel[0]>= h1 && hsvcurrentPixel[0]<= h2 && hsvcurrentPixel[1]>= s1 && hsvcurrentPixel[1]<= s2 && hsvcurrentPixel[2]>= v1 && hsvcurrentPixel[2]<= v2) {  thresholdCheck = true; }  return thresholdCheck; } |

Pour ce qui est des algorithmes de conversion de couleurs entre RGB et HSV nous avons utilisé le même algorithme que dans le premier laboratoire. Nous prenons les valeurs RBG et on calcule la teinte, la saturation et la valeur.

|  |
| --- |
| private double[] rgb2hsv(Pixel result2) {  //Variables nécessaires pour le calcul  int red = result2.getRed();  int green = result2.getGreen();  int blue = result2.getBlue();  double r,g,b;  double hsvReturn[] = new double[3];  double delta, cmax, cmin;  r = (double)red/255;  g = (double)green/255;  b = (double)blue/255;  cmax = Math.max(Math.max(r,g), b);  cmin = Math.min(Math.min(r,g), b);  delta = cmax - cmin;   // on calcule la teinte  if (delta == 0.0) {hsvReturn[0] = 0.0;}  if (cmax==r) {  hsvReturn[0] = 60\*(((g-b)/delta)%6);  }  if (cmax==g) {  hsvReturn[0] = 60\*(((b-r)/delta)+2);  }  if (cmax==b) {  hsvReturn[0] = 60\*(((r-g)/delta)+4);  }  //Calcule de la saturation  if (cmax==0.0) {  hsvReturn[1] = 0.0;  }else {  hsvReturn[1] = (delta/cmax)\*255;  }  //calcul de la valeur  hsvReturn[2] = cmax\*255;  return hsvReturn; } |

# 

# 

# Discussion

Pour l'implémentation nous avons profité du fait que le pattern MVC a avait été utilisé afin de modifier le moins de code possible. Nous avons réussi à tout garder simple et seulement modifier une seule classe qui joue le rôle du contrôleur. Un autre avantage d’utiliser une seule classe pour toutes les méthodes qui seront nécessaires est d’avoir une bonne cohésion dans son code.

On avait commencé les modification pour le flood fill et on avait réussi à l'implémenter des le premier essai, par contre pour le boundary fill nous avons dû revoir notre implémentation, car on avait mal compris les requis du laboratoire. On avait réalisé l'implémentation d’une méthode qui détectait l'extérieur de l’image d’origine et par la suite modifiant les couleurs du fond de l’image sans modifier les autres régions intérieures. Par contre, suite aux échanges entre notre équipe et le chargé le TP, nous avons modifié notre méthode pour répondre correctement aux requis du laboratoire. Suite aux modifications nous avons parti de la méthode floodfill, mais on l’a modifié de façon à reconnaître la couleur de la bordure. Nous avons également créé une nouvelle méthode pour le seuil et une pour la conversion de la couleur entre le RBG et le HSV. Ces deux méthodes seraient utilisées à l’intérieur de la classe.

Il aurait été possible d’utiliser la méthode de conversion que nous avons utilisée dans le premier laboratoire au lieu de créer une nouvelle méthode et de cette façon il aurait été possible de garder une consistance entre nos méthodes de calcul. Par contre, on voulait essayer d’éviter d’avoir trop de liens entre les méthodes et on a opté pour une nouvelle méthode.

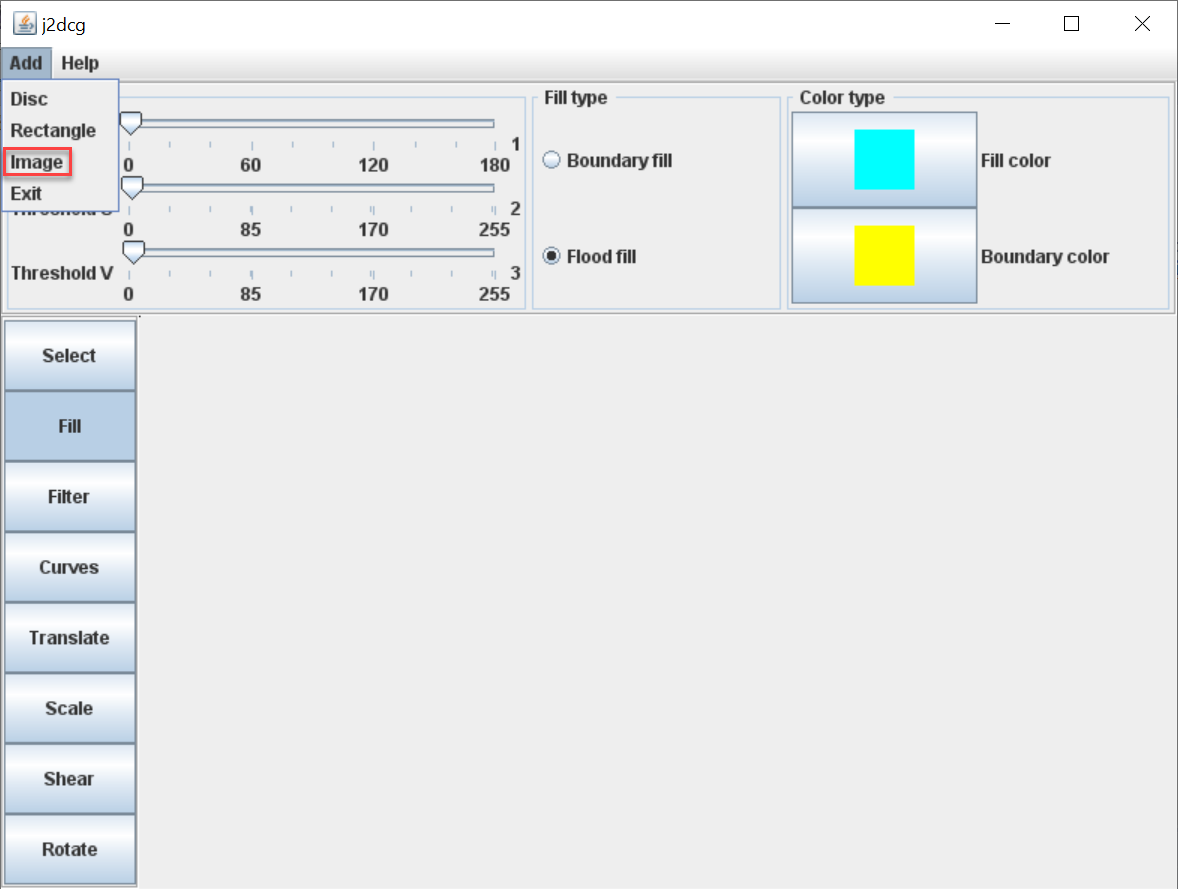
Pour ce qui est de la méthode de recherche des voisins on va choisit la méthode des quatre voisins, par contre on avait l’alternative des 8 voisins ce qui aurait ralenti le code et n’aurait pas apporté un avantage très grand pour ce que nous recherchons de notre code.

On croit avoir pris les meilleures décisions pour ce qui est de l’implémentation et nous sommes satisfaites que les fonctionnements tels que prévus.

# 

# Manuel de l’utilisateur

L’application Java requiert un environnement de développement Eclipse avec la version Java 8 ou plus récente. Pour commencer, le projet doit être importé et doit être exécuté afin d’avoir l’interface graphique que l'usager pourra utiliser. Une nouvelle fenêtre ouvrira et l’utilisateur devra aller à l’onglet gauche qui se nomme Fill. Dans cet onglet, l’usager devra importer une image à partir du menu supérieur. En choisissant l’option image, une nouvelle fenêtre ouvrira et l’image pourra être importée.



Voici les trois options qui s’offrent aux usagers:

1. Sélectionner le bouton Flood fill:

Utiliser le sélecteur de couleur, soit le bouton fill color et choisissez la nouvelle couleur désirée. Par la suite, simplement utilisez la souris pour cliquer sur la région intérieure que vous désirez modifier.

1. Sélectionner le bouton Flood Fill en modifiant le seuil:

Utiliser le sélecteur de couleur, soit le bouton fill color et choisissez la nouvelle couleur désirée. Une fois que la couleur est choisie, utilisez les trois curseurs de défilement pour faire varier les seuils. Les trois seuils qui peuvent être modifiés sont la teinte, la saturation et la valeur. Par la suite, simplement utilisez la souris pour cliquer sur la région intérieure que vous désirez modifier.

1. Sélectionner le bouton Boundary Fill:

Utiliser le sélecteur de couleur, soit le bouton fill color et choisissez la nouvelle couleur désirée. Faites de même avec le bouton boundary afin de sélectionner la couleur qui délimite la région qui doit être modifiée. Pour modifier l’image, simplement utilisez la souris pour cliquer sur la région intérieure que vous désirez modifier.

# Conclusion

En conclusion, notre équipe a réussi à atteindre les objectifs mentionnés dans les premières sections de ce rapport. En effet, nous avons réussi à implémenter les méthodes nécessaires au remplissage des régions des images importées dans l’application. En plus du remplissage, nous avons réussi à implémenter la sélection de seuils. Étant donné que l’application avait implémenté le patron modèle vue contrôleur, nous avons réussi à modifier seulement une seule classe afin d'implémenter nos modifications.

Dans la section analyse, nous identifiés les principes nécessaires pour réaliser le laboratoire et nous avons approfondi notre compréhension dans la section conception dans laquelle nous avons concentré les informations sur le code. Pour ce qui est de l'implémentation et l’analyse, nous avons décrit nos méthodes et les algorithmes utilisés. Pour ce qui est des autres options, nous aurions peut utiliser des algorithmes différents pour la modification des couleurs en utilisant par exemple la méthode des 8 voisins.

Ce laboratoire nous a permis de comprendre des principes importants, mais il serait utile de fournir les diapositives descriptives du laboratoire avec l'énoncé afin de faciliter la compréhension des futurs étudiants qui le réaliseront.

# 

# Références

* Notes des cours du cours GTI411 et notes d’explication du laboratoire