UFR sciences & techniques

UNIVERSITÉ DE BOURGOGNE

BrodUI

LACHAUD Samuel - PAZOLA Loïs - AL-HALABI Zeina Hélène - TISSERAND Orlane - Corentin DEBRABANT - Vincent GAY
M1 Informatique

SOMMAIRE

I) Introduction

- Qu'est-ce que BrodUI
- Présentation du cahier des charges
- Présentation de l'application

II) Fonctionnement de l'application

- Pages de tutoriel et de paramètres
- Page de convertion
- Page d'export

III) Traitement de l'image

- Réduction du nombre de couleurs
- Utilisation des modèles de couleurs
- Analyse de Performance
- Analyse de résultats

IV) Fonctionnalités supplémentaires

- Tests unitaires et Code Coverage
- Documentation Doxygen
- Organisation du travail
- Conclusion

I) Introduction

Qu'est-ce que BrodUI

BrodUI est un projet de traitement d'image codé en **C#** portant sur la création d'un patron de broderie depuis une image. Pour cela il est nécessaire de réduire son nombre de couleurs afin que la brodeuse ou le brodeur n'ai pas besoin d'un nombre trop élevé de fils différents. BrodUI est un projet étudiant créé par **6 personnes**, chacune ayant son propre rôle. Nous retrouvons donc :

- Optimisations et accessibilité : Loïs PAZOLA
- Développement logiciel et UI : Samuel LACHAUD
- Implémentation de l'algorithme de réduction du nombre de couleurs : Corentin DEBRABANT - Vincent GAY
- Translation des modèles de couleurs : Orlane TISSERAND - Zeina Hélène AL-HALABI
- Calcul de la longueur de fil : Zeina Hélène AL-HALABI

Présentation du cahier des charges

Le sujet consiste à créer un programme produisant un **patron permettant la broderie par point de croix** à partir d'une **image de référence**. La broderie par point de croix consiste à créer des croix de fil à espace régulier créant au finale une image. L'image ainsi formée est comparable à une image numérique avec chaque croix représentant un pixel. Un patron de broderie indique quel fil utiliser à chaque "pixel". Dans ce programme, nous devons d'abord redimensionner l'image de référence afin d'obtenir un nombre de croix humainement réalisable puis segmenter l'image de façon à n'avoir qu'un certain nombre de couleurs de fils différents et finalement de trouver les références DMC de fil dont la couleur est la plus proche de chacune des couleurs de l'image segmentée.

Il serait également intéressant pour le brodeur de connaître la **longueur de fil de chaque couleur ainsi que les couleurs nécessaires**. Au terme du programme le brodeur pourra obtenir une interface présentant le patron de broderie de l'image, ainsi que les informations précédentes concernant les fils. De plus un export **PDF** lui permettra d'exporter **les informations ainsi obtenues hors de l'application**.

La première barrière pour un brodeur qui souhaite obtenir un patron par rapport à une **image donnée** est la taille de celle-ci. Il est donc ainsi primordial de commencer par un **recadrage** de celle-ci afin d'obtenir un patron humainement faisable. Ensuite vient le problème des couleurs. En effet de manière courante, nous utilisons des couleurs au format RGB (ou RVB), pourtant dans le domaine de la broderie, le format utilisé est le format DMC. L'enjeu de ce point est donc de convertir le RGB en DMC en prenant les couleurs les plus proches. De la même manière que pour la taille, **le nombre de couleurs est une grande contrainte pour le brodeur**. En effet plus il y aura de couleurs différentes, plus la broderie sera longue et fastidieuse, et plus il faudra de couleurs différentes, plus la broderie sera chère à réaliser (par exemple retirer les couleurs peu représentées dans l'image).

Pour régler le problème du nombre de couleurs, on regroupe les différentes couleurs de l'image et on en élit une représentante pour chacun de ces groupes. Pour faire cela, on peut utiliser plusieurs algorithmes. Un des premiers envisagés est celui de l'accroissement de zones appliquées à des images en couleurs, celui-ci ne sera finalement pas choisi au profit d'autres solutions. Le premier de ces algorithmes est celui de **K-Means** qui permet d'obtenir un nombre demandé de représentants des couleurs. Le deuxième est **Mean-Shift**, une version de K-Means où le nombre de représentants n'est pas fixé. Un troisième algorithme est envisagé, il s'agit de celui nommé **Fuzzy C-Means** qui est similaire à K-Means. **Nous avons finalement choisi d'utiliser l'algorithme K-Means**.

Une fois les algorithmes appliqués, nous obtenons donc le nombre de couleurs, leurs codes DMC et le patron. Nous pouvons donc afficher au brodeur les informations souhaitées. C'est ici que la partie graphique de l'application entre en jeu. L'interface permet d'afficher un tableau qui est le patron de broderie, à côté, on retrouve l'image résultat que l'on doit avoir après avoir brodé le patron. En dessous, nous retrouvons la légende de chacun des symboles du patron qui forment les différentes couleurs de fils.

L'application BrodUI (mélange de **Brod**erie et de **UI** signifiant « User Interface ») doit répondre à un certain nombre de critères afin d'être utilisable par un utilisateur, de lui permettre de s'en servir facilement, et potentiellement d'être utilisé dans un usage professionnel. Voici donc les différents critères que doit respecter BrodUI :

- Une **page de tutoriel** doit être présente afin d'aider l'utilisateur à **mieux comprendre le logiciel** et son fonctionnement. Le but étant qu'un utilisateur lambda puisse utiliser l'application sans avoir de connaissances dans le domaine.
- L'interface utilisateur doit être **accueillante**, **explicite** et **accessible**.
- Une **image** doit pouvoir être chargée dans l'application, être **redimensionnée**
- L'utilisateur doit pouvoir **visualiser le patron de broderie** et pouvoir **l'exporter en PDF** afin de l'imprimer.
- L'utilisateur doit pouvoir choisir la **langue** de l'application (Français ou Anglais) et le **thème** de l'application.

Elle doit également répondre à des critères d'analyse de données, en effet, l'application permet également d'analyser les résultats d'un algorithme de réduction de couleur dans une image avec différents paramètres. Voici donc les différents critères d'analyse :

- L'utilisateur doit pouvoir activer le « **Terminal** » dans les paramètres afin d'obtenir les différentes informations d'actions sur l'interface et les calculs effectués
- L'algorithme de réduction de nombre de couleur doit implémenter l'algorithme **K-Means**.
- Avant l'exécution de l'algorithme K-Means, l'utilisateur doit pouvoir choisir le **nombre de couleurs** souhaité, le **nombre d'itérations** de l'algorithme ainsi que le modèle de couleurs (**RGB** ou **HSL**)

Le cahier des charges prends également des directives pour les développeurs, ainsi nous devons prendre en compte les points suivants :

- En implémentant l'algorithme **K-Means**, les personnes chargées de ce point doivent avoir une bonne connaissance de celui-ci, être capable de **l'expliquer** et de le **manipuler** (le but n'est pas uniquement de l'implémenter, mais d'en comprendre le fonctionnement et les enjeux).
- Les couleurs des fils de broderie sont (dans la majorité des cas) au format DMC. Il est donc nécessaire de pouvoir transformer du **RGB** ou du **HSL** en **DMC**.
- Le calcul de la longueur de fil doit prendre en compte le fait que nous utilisons **du point du croix** et le passage des fils **derrière le patron**.

Présentation de l'application

BrodUI est réalisé sur **Windows 11 et pour Windows 10 ou 11 avec langage C#.** l'interface graphique utilisera le Framework **WPF** (tous deux disponibles sur Visual Studio de Microsoft). Pour la partie graphique, l'interface utilisera **WPFUI** pour adapter l'application WPF à la charte graphique de Windows 11. En effet les applications comme le Microsoft Store, la calculatrice Windows, ou bien Paint utilisent actuellement WPFUI, ainsi BrodUI s'intègre donc bien dans l'environnement Windows et utilise les thèmes et couleurs d'accentuations que vous avez définit pour toutes les applications dans les paramètres Windows Ainsi, en l'absence de contrainte quant à la plateforme d'exécution, nous avons choisi que l'application sera uniquement disponible sur Windows. BrodUI se compose en **4 pages différentes**. Nous allons vous les lister et montrer ici :

Page de tutoriel :

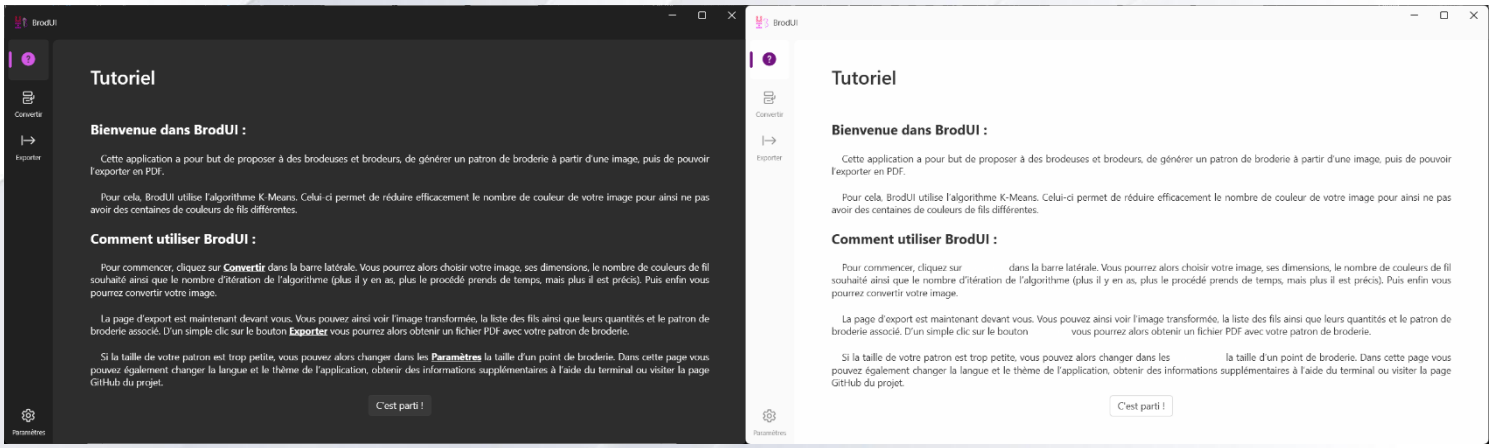


Figure 1 : Page de tutoriel de BrodUI (thèmes sombre et clair)

Cette page est la page d'accueil et permets d'expliquer l'utilisation de l'application (cf. Cahier des charges)

Page de conversion :

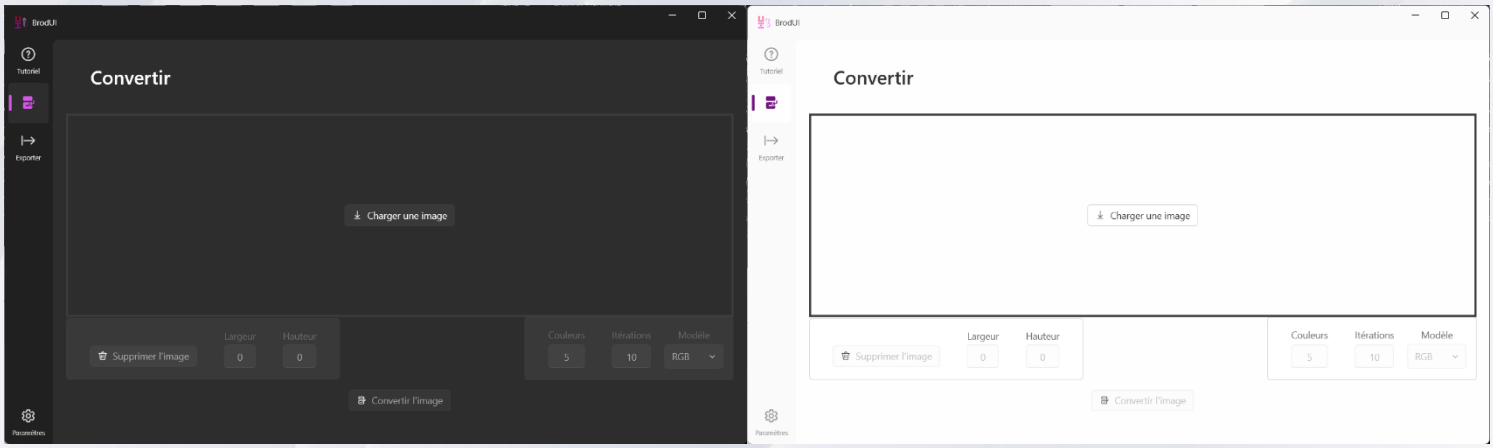


Figure 2 : Page de conversion de BrodUI (thèmes sombre et clair)

Cette page est la page de conversion d'image, l'utilisateur peut charger une image, définir les paramètres puis lancer la conversion

Page d'export :

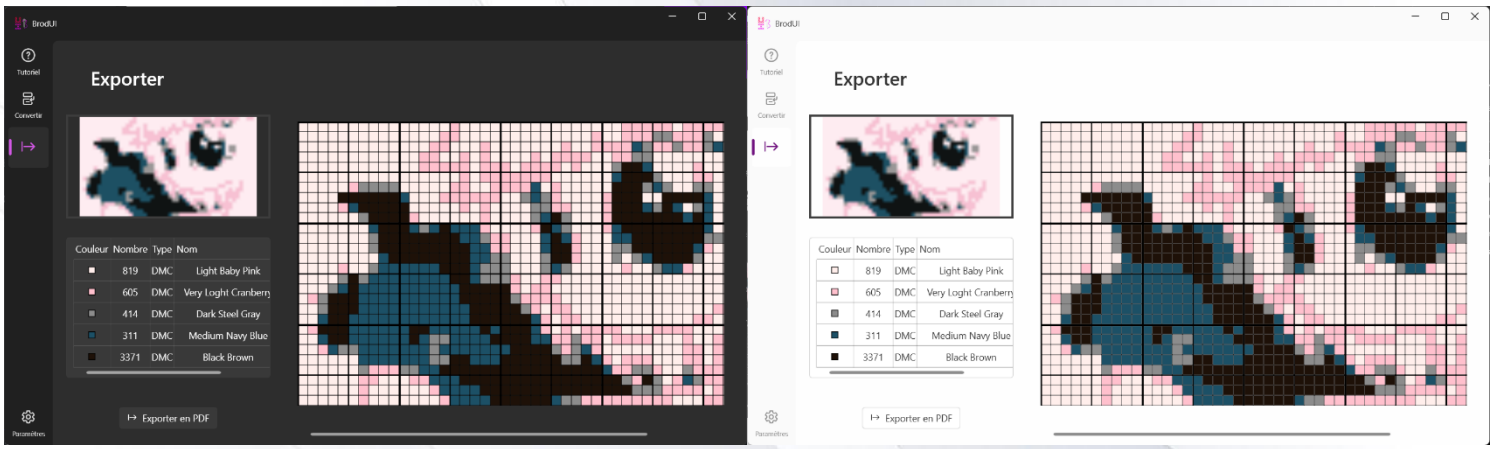


Figure 3 : Page d'export de BrodUI (thèmes sombre et clair)

Cette page permet d'afficher le patron de broderie de l'image que l'on a chargé. Elle permet également d'exporter le patron en PDF.

Page des paramètres :

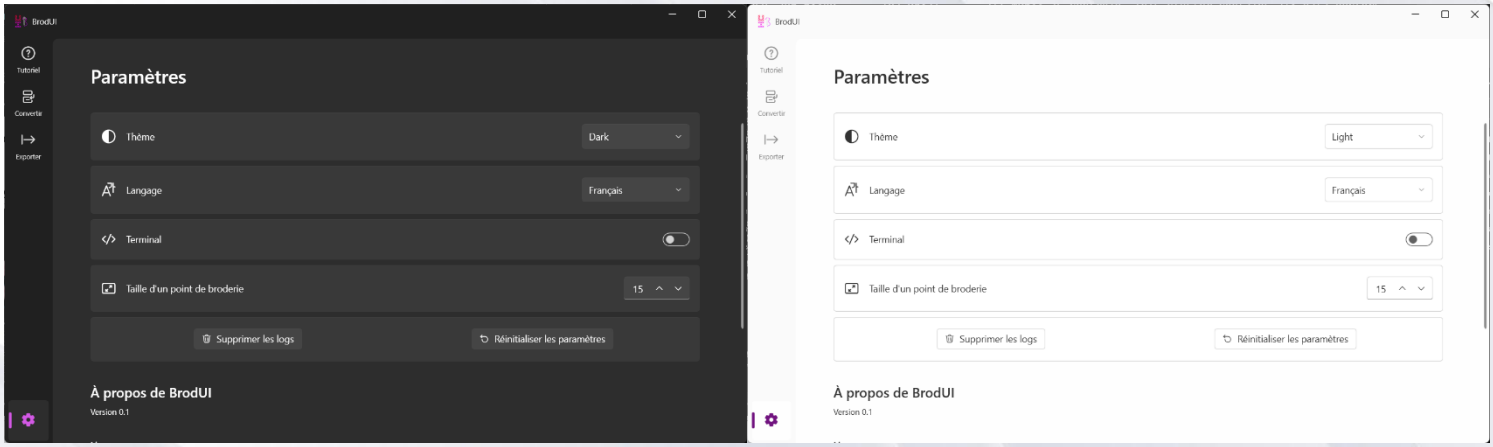


Figure 4 : Page de paramètres de BrodUI (thèmes sombre et clair)

Cette page permet de paramétrer l'application en choisissant des options pour modifier l'expérience utilisateur.

II) Fonctionnement de l'application

Dans cette partie, nous allons voir en profondeur les différentes pages de notre application, leurs fonctionnalités et des interactions utilisateurs. Nous expliquerons uniquement les pages car le **bandeau de menu** ainsi que la « **TitleBar** » de l'application sont générés automatiquement (du moins choisis par le développeur) à la création de l'application **WPFUI**.

Pages de tutoriel et de paramètres

Nous allons commencer par la **page de tutoriel**. En effet celle-ci est assez basique et ne comporte pas beaucoup d'interactions importantes. Le contenu de celle-ci se concentre sur l'explication du sujet et de l'utilisation de l'application. Ainsi pour permettre à l'utilisateur de voir de quoi parle chaque partie de ce tutoriel, des liens ont été mis dans celui-ci afin de naviguer facilement. Ces liens ont une utilisation semblable au bandeau de menu situé à gauche de la page. On retrouve également un bouton en bas de la page permettant d'aller directement de la même manière la page de conversion :

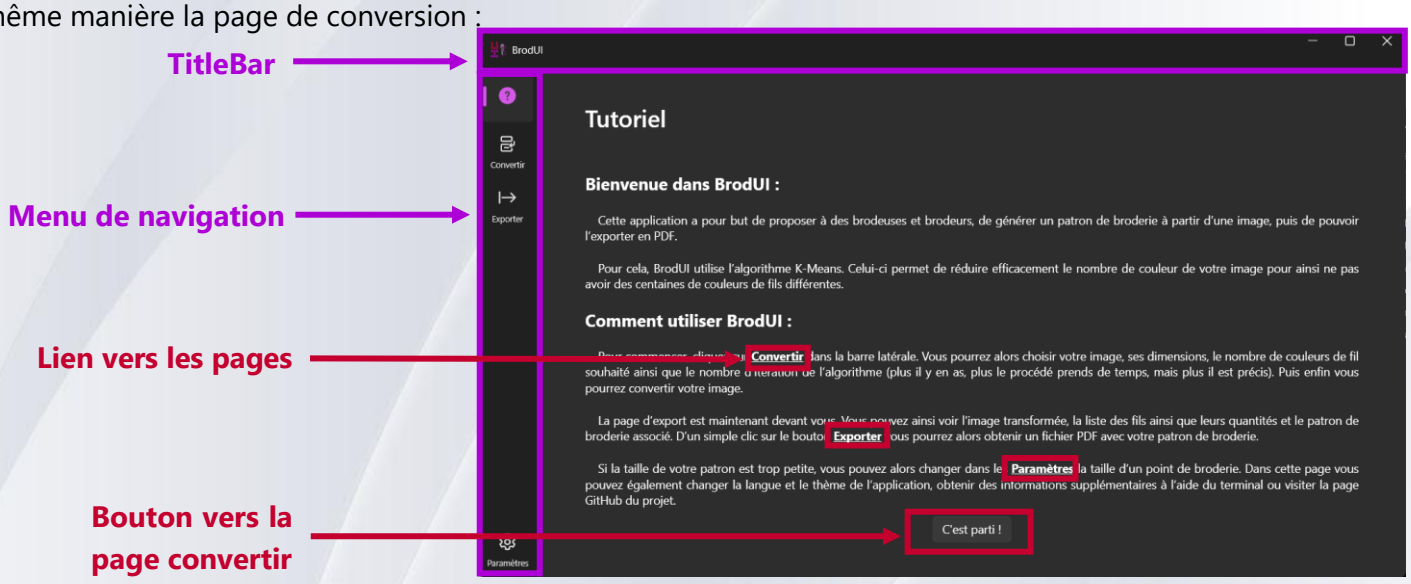


Figure 5 : Description de la page de tutoriel

Ensuite nous avons la **page des paramètres**, celle-ci est accessible depuis le menu de navigation ou depuis la page de tutoriel. Dans cette page, vous retrouverez des paramètres basiques comme le choix du thème, ou bien le langage. Mais vous retrouverez également le choix de la taille du point de broderie, en effet un grand patron peut nécessiter un point de broderie plus petit, etc. Ici vous pouvez donc choisir cette taille (même après avoir convertit votre image). On retrouve également un **Terminal** activable à la demande. Celui-ci est plutôt utile pour la partie analyse de données de K-Means, mais également pour les utilisateurs expérimentés souhaitant connaitre les différentes itérations de l'algorithme, du temps que prends celui-ci ou de toutes autres interactions avec l'interface. Enfin, qui dit terminal, dit logs. En effet tout ce qui est marqué dans le terminal est horodaté et stocké dans un fichier de log, celui-ci étant rechargé dans le terminal au lancement du logiciel. Vous avez donc deux boutons servant à réinitialiser les **logs** et les **paramètres**.

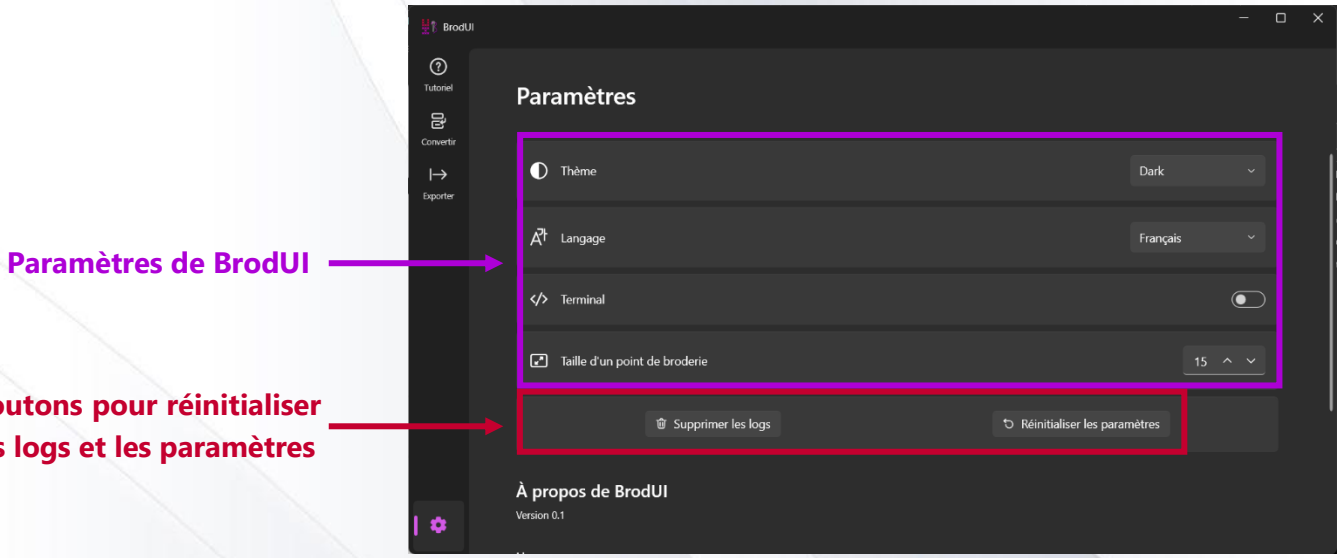


Figure 6 : Description de la page des paramètres

Nous retrouvons également en bas de ces paramètres le numéro de version de BrodUI, les liens vers le GitHub du projet, vers la release GitHub et vers le logo original avant modification. On retrouve également les liens vers les bibliothèques externes :

Figure 7 : Desciption de la partie "à propos"

Pages de conversion

Nous sommes maintenant (après le tutoriel) sur la **page de conversion**. Celle-ci nous permet de charger une image de notre choix (au format, JPG, GIF, BMP ou PNG). Puis l'utilisateur choisis la taille de l'image (la hauteur et la largeur ne peux pas être inférieure à 20 pixels). Le redimensionnement de l'image garde le ratio de l'image. Si vous voulez changer d'image, vous pouvez appuyer sur le bouton prévu à cet effet. Une fois la partie image remplie, il vous reste à remplir la partie des paramètres K-Means. Pour cela, il faut tout d'abord donner le nombre de couleurs souhaité sur l'image résultante (après traitement). Il est important de choisir un nombre de couleur inférieur au nombre actuel de couleurs sur l'image sinon K-Means ne pourra fournir un résultat convainquant. Enfin, il reste à choisir le nombre d'itérations de K-Means (plus le nombre est élevé, plus le résultat est précis, mais plus c'est long. La valeur par défaut étant 10, c'est la valeur que nous préconisons). Enfin le choix du modèle de couleur est lui aussi très important car le résultat différera légèrement.

Par un appui sur le bouton « **Convertir** » une fois avoir saisi toutes les informations nécessaires, vous pourrez apprécier une barre de chargement des différentes itérations de K-Means demandées (10% par 10% si vous avez choisis 10 itérations). Le processus s'effectue en arrière-plan, il est donc possible de naviguer vers une autre page pendant le traitement. De plus, sur Windows 11, il est possible d'ajouter des **animations** en dessous du **logo de l'application** dans la **barre des tâches**, il y en as donc une qui indique le processus.

Image chargée via un clic sur le bouton ou via un « Glisser -déposer »

Contrôles de l'images et de K-Means (sur chaque contrôle vous pouvez passer la souris pour avoir une description en « Hover »

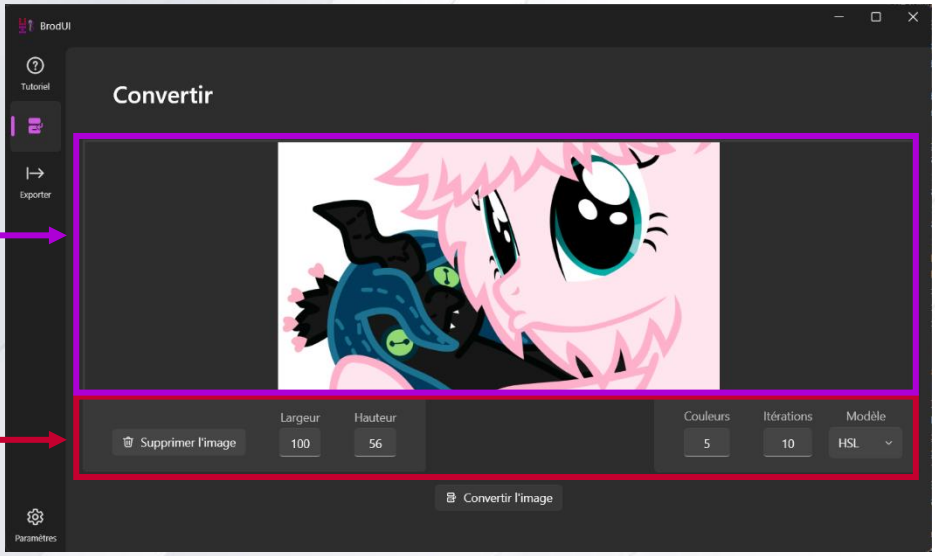


Figure 8 : Description de la page de conversion

Barre de chargement de la conversion (Redimensionnement et K-Means)

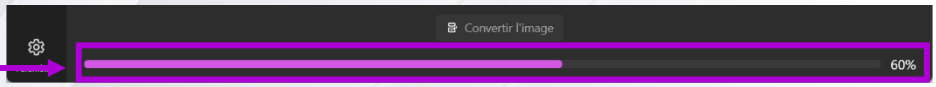


Figure 9 : Description de la barre de chargement

Indication de processus en cours

Figure 10 : Description de l'indicateur de processus en cours

Pages d'export

Une fois l'image convertie et la **barre de chargement terminée** (les derniers pourcentages peuvent être très longs à cause de la génération de **la page d'export** en fonction de la taille de l'image demandée). Vous pouvez donc apercevoir, comme sur l'introduction, l'image convertie, le patron ainsi que la liste des fils. Il y a également un bouton pour exporter le tout dans un PDF.

Image convertie

Liste des fils : le numéro DMC le nom et le nombre nécessaire

Patron de broderie (la taille du point de broderie est définie dans les paramètres)

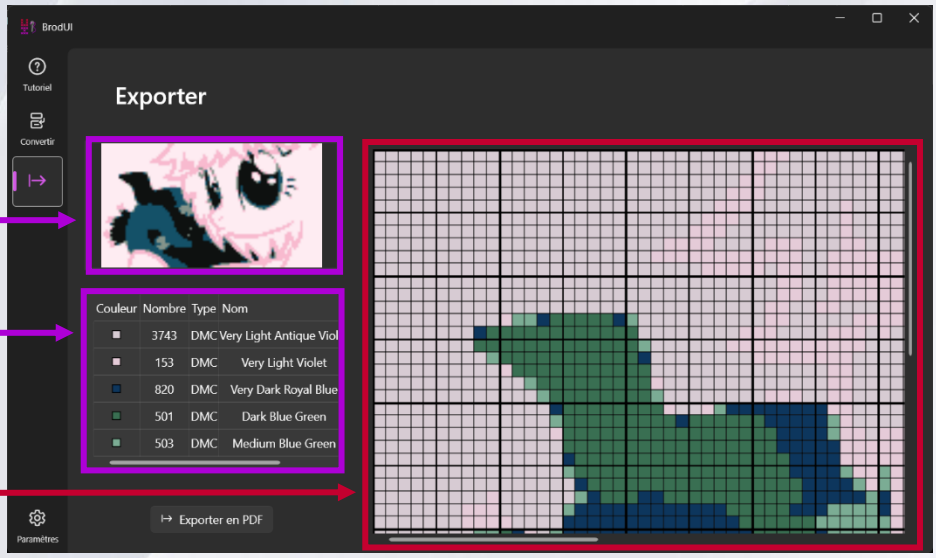


Figure 11 : Description de la page d'export

Une fois le PDF exporté, le nom du patron prends le nom du **PDF**. Il est composé en **3 pages** : La première c'est **l'image convertie** ainsi que le **nom du patron**, la deuxième c'est **la liste des fils** et enfin la troisième c'est le **patron de broderie**. Si le patron sort de la page (patron trop grand) il faut **changer la taille du point de broderie**.

III) Traitement de l'image

Réduction du nombre de couleurs

Pour la réduction des couleurs, nous avons utilisé l'algorithme **K-Means**. KMeans est un algorithme de **nuées dynamiques**. En effet, lors des cours de traitement d'images ce semestre avec notre professeur Mr Migniot, nous avons fait de la classification par nuées dynamiques en utilisant (sans le savoir) K-Means.

K-Means, est un algorithme de nuées dynamiques, permettant de **transformer un ensemble de valeurs d'une certaine taille en un ensemble de valeurs de taille inférieur** (ou égale, mais dans ce cas-là cela devient inutile) ou plus précisément, il s'agit d'un ensemble de valeurs représentant des groupes de valeurs où chaque valeur de départ est contenue dans un et un seul groupe. Dans notre cas, nous ne nous intéressons pas à quel groupe chaque valeur de départ appartient mais à la valeur représentant ce groupe.

Expliquons maintenant le principe, nous regroupons les **valeurs** dans un certain nombre de classes (ou clusters) avec une valeur représentant **chaque classe**, ce sont les noyaux (ou **centroïdes**). On peut ainsi créer une **fonction de transformation** qui pour chaque valeur de départ associe le noyau représentant la classe où elle se trouve. Le nombre de classes et donc de valeurs représentant ses classes est inférieur ou égal au nombre de valeurs de départ, par conséquent on arrive à une diminution du nombre de valeurs. Ici, une valeur peut être un vecteur de réels de dimension quelconque tant que tous les vecteurs ont la même dimension.

Nous allons maintenant parler du **fonctionnement de l'algorithme**. Pour l'initialisation de K-Means, nous devons spécifier :

- **Les valeurs** (des vecteurs) d'entrée (dans notre cas, ce sont les couleurs présentes sur l'image d'origine)
- **Le nombre de classe voulu** (soit le nombre de couleurs sur l'image résultante).

L'algorithme définit ensuite un **noyau de départ** pour **chaque classe**. Ces valeurs sont choisies au hasard dans le domaine de définition des valeurs (par exemple entre 0 et 255 pour un canal d'une couleur. Chaque valeur dans un vecteur peut avoir un **domaine différent** comme la taille et l'âge d'une personne) ou au hasard parmi les vecteurs d'entrée (ce qui fait que les valeurs du noyau sont forcément dans leur domaine de définition).

Le déroulement de l'algorithme se sépare en plusieurs itérations avec chaque itération contenant deux opérations et une vérification. Les deux opérations sont, dans l'ordre :

1. Chacune des valeurs d'entrées est associée à la classe du noyau le plus proche d'elle peu importe sa classe précédente.
2. Les noyaux sont recalculés comme étant la moyenne (Means) des valeurs présentes dans la classe qu'elle représente. Plus exactement, un noyau est composé des moyennes de chaque valeur de chaque vecteur de sa classe. Par exemple, les deux vecteurs (0,2,5) et (2, 4, 3) sont dans une même classe, le nouveau noyau de cette classe est (1,3,4) car (0+2)/2=1 etc…

La vérification consiste à vérifier si les noyaux ont changé pendant l'itération. Si les noyaux sont différents, l'algorithme continue mais si les noyaux sont identiques alors l'algorithme s'arrête puisque même après les itérations suivantes, les noyaux et classes ne changeront pas.

Voici donc, un schéma représentant les deux premières itérations de l'algorithme avec six vecteurs à deux dimensions en entrées et 2 noyaux. Les noyaux sont choisis au hasard dans l'espace :

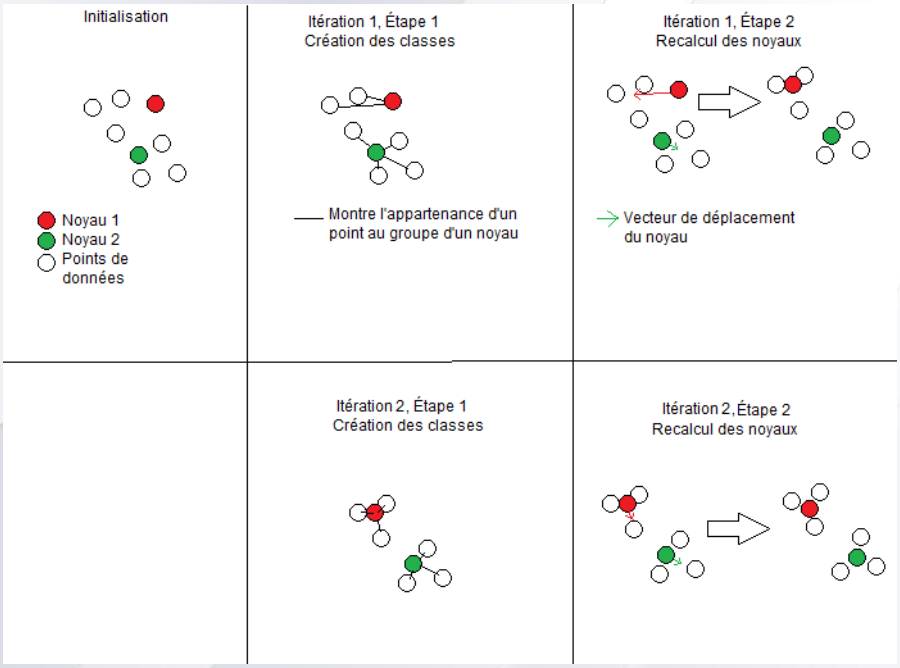


Figure 12 : Schéma des deux premières itérations de K-Means

Dans le **déroulement réel** de l'algorithme avec ces valeurs, **il n'y aurait qu'une troisième itération** puisque les classes auraient été les mêmes que dans cette deuxième itération, les noyaux étant les moyennes des points dans la classe, si les points sont identiques, le noyau l'est aussi.

Pour **l'implémentation de l'algorithme**, nous avons, comme demandé par le professeur, **une version déjà implémentée par quelqu'un d'autre**. Nous avons donc dû chercher une implémentation. Nous nous sommes d'abord tournés vers les **librairies**. Nous avions trouvé une librairie (**ML.NET**) écrite directement par **Microsoft**, le propriétaire du langage C# dans lequel nous avons écrit le projet. On ne pouvait donc pas mieux demander. Cependant, l'implémentation de K-Means présente dans cette librairie était **différente** de ce qui était attendu. En effet, il fallait **entraîner le modèle avec des données** avant de pouvoir utiliser K-Means avec nos vraies données. À la suite d'un entretien avec notre professeur en charge, il a été décidé que **cette librairie ne conviendrait pas.** Nous avons donc continué à chercher, sans résultat.

Ne trouvant pas de librairie, le choix a été fait de se tournés vers les implémentations fournies sur GitHub. Nous en avons trouvé plusieurs, d'autres avec entrainement du modèle et certaines où la dimension des vecteurs étaient fixées. Parmi celles-ci, nous en avons trouvé une qui **ne contenait que l'implémentation de K-Means** et d'autres classes nécessaires. C'est à « **CeesJanNolen** » que nous devons cette implémentation.

Il y a quelques particularités dans cette version. En effet, à **l'initialisation**, nous fixons le **nombre maximum d'itérations de l'algorithme après laquelle il s'arrête même si les noyaux finaux n'ont pas été trouvés**, cela permet de **limiter l'attente de l'utilisateur dans les cas complexes**. Un tel cas serait dans notre projet une image de grande taille avec peu de couleurs finales. En effet, dans ce cas-là, chaque classe contiendra beaucoup de valeur dont de nombreuses éloignées du noyau de leur classe, ce qui causera beaucoup de changement de classe pour ces valeurs.

De plus les **noyaux de départ sont pris aléatoirement dans les couleurs d'entrée**, ce qui fait que si celles-ci forment des groupements isolés (comme dans un drapeau où de nombreux pixels sont de même couleur mais éloignés des autres couleurs) alors deux noyaux peuvent se retrouver dans un de ces groupements et ne pas en sortir. Si le nombre de groupement est égal au nombre de noyaux alors un noyau se trouvera entre deux groupements alors que la solution optimale serait d'avoir un noyau par groupement.

Pour limiter ceci, dans cette implémentation, **le même vecteur** (en tant qu'objet) **ne peut pas être utilisé pour deux noyaux** cependant deux vecteurs **différents** (en tant qu'objet) mais de **valeurs égales** peuvent être utilisés comme deux **noyaux de départ.**

C'est pour cela que pour obtenir une bonne solution il est obligé d'utiliser l'algorithme plusieurs fois avec les mêmes données en entrée et de prendre la meilleur des solutions proposées (cette utilisation de l'implémentation était déjà présente sur le **GitHub d'origine**). Cela a évidemment pour effet d'augmenter le temps d'exécution. Ce problème est réduit dans les images où les couleurs sont répartis sur l'entièreté du spectre. En théorie, le problème devrait même être supprimé si la répartition et le nombre d'itérations sont suffisant. La mesure de la qualité d'une solution se fait par le calcul de la somme des distances au carré des valeurs de chaque classe avec le noyau de cette classe. Plus la somme est proche de zéro meilleur est la solution.

Les **limitations de cet algorithme** sont dû au fait que le résultat dépend des noyaux de départ dans certains cas (si on limite le nombre d'itérations ou si les couleurs sont regroupés) et le fait que le nombre de noyaux soit fixés ne donnant pas la séparation optimale. Dans le cas où il y a quatre regroupements de valeurs mais seulement 3 noyaux, l'algorithme ne donnera pas la solution optimale évidente. Ce nombre final de couleur est un désavantage pour la classification mais un avantage pour le programme de broderie puisque l'utilisateur peut choisir le nombre de couleur dont il aura besoin. Il obtiendra donc un résultat correspondant à son besoin au prix d'un clustering pouvant être de moindre qualité. **K-Means peut aussi rencontrer un problème dans le cas de groupes longs (comme des ellipses) ou de groupes rectangulaires.**

Comme on peut le voir sur cette image (qui n'est qu'une illustration sans calcul), on peut identifier 2 groupes distincts avec un noyau chacun cependant au moins un des points qui devrait probablement être dans le groupe rouge et en fait plus proche du noyau vert. On peut donc voire qu'effectivement K-Means peut obtenir des résultats avec des erreurs (ou du moins qui ne semble pas correct à l'œil humain).

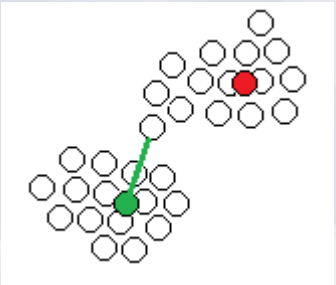


Figure 13 : Schéma groupes K-Means

D'autres algorithmes de clustering auraient pu être utilisés, tel que Mean-Shift ou Fuzzy C-Means. Ce sont tous les deux des algorithmes proches de Kmeans.

Mean-Shift peut être vu comme une généralisation de Kmeans (ou plus exactement, une variation de Mean-Shift est une généralisation de K-Means). En effet, Mean-shift fonctionne plus ou moins de la même manière que K-Means mais n'a pas besoin d'un nombre de classes. Ce qui peut faire que Mean-shift ait de meilleurs résultats mais cela cause aussi une augmentation du temps de calculs. De plus, Mean-Shift peut commettre des erreurs dans le cas de valeurs très isolés ou former des classes de petites tailles proches de grands groupes.

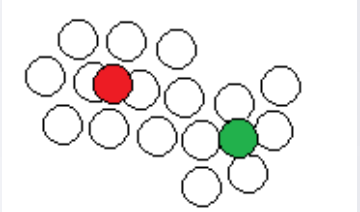


Figure 14 : Schéma groupe séparés

Dans cette image (qui n'est qu'une illustration sans calcul et qu'un extrait de l'espace), on peut voir qu'un groupe est séparé en deux sous-groupes alors qu'on pourrait imaginer qu'un seul noyau pour l'ensemble suffirait.

Dans cette image (qui n'est qu'une illustration sans calcul et qu'un extrait de l'espace), on peut voir qu'un groupe est séparé en deux sous-groupes alors qu'on pourrait imaginer qu'un seul noyau pour l'ensemble suffirait.

Ne pas avoir de nombre fixé de classes permet dans la plupart des cas d'avoir une meilleur séparation des valeurs, cela permet ici d'avoir une meilleure image finale mais force l'utilisateur concernant le nombre de couleurs. **Mean-Shift** a le même problème que K-Means concernant les groupements longs ou rectangulaires.

Fuzzy C-Means est l'algorithme de « **Fuzzy Clustering** » le plus utilisé. Dans les algorithmes de « Fuzzy Clustering » (aussi appelé « Soft Clustering »), les valeurs ont un pourcentage d'appartenance au groupement de chaque noyau au lieu d'appartenir ou non au groupe. Dans **Fuzzy C-Means**, les noyaux sont recalculés en utilisant la moyenne pondérée des points de leur classe, le poids étant le pourcentage d'appartenance à cette classe passer dans une fonction de rigidité (**stiffness**). En manipulant cette fonction, on peut obtenir un **vecteur binaire** montrant le groupe avec le plus d'appartenance et ainsi obtenir le même résultat que K-Means (tout en gardant un temps d'exécution plus important).

Fuzzy C-Means, comme Mean-Shift, a tendance à donner de meilleurs résultats que K-Means au prix d'une augmentation de la complexité (aussi bien spatiale que temporelle) et donc du temps de calculs. Cependant, contrairement à K-Means et Mean-Shift, Fuzzy C-Means gère mieux le cas des groupes longs.

Utilisation des modèles de couleurs

L'un des fournisseurs majeurs de fils pour la broderie est la marque DMC. Chaque fil possède un code indiquant une couleur. Pour passer d'une couleur RGB au code couleur DMC, on a utilisé un fichier donnant pour chaque couleur DMC leur valeur RGB. Le site https://floss.maxxmint.com/dmc_to_rgb.php possède l'ensemble des couleurs DMC avec leur équivalent RGB.

La classe **RgbToDmc** sert de liaison entre un couleur RGB et sa valeur DMC. Toutes les couleurs sont stockées dans un tableau de int. La fonction **Initialization** lit le fichier des couleurs et remplit le tableau. Elle lit chaque ligne du fichier et prend chaque nombre séparé d'un espace et l'ajoute dans la bonne cordonne du tableau commençant par la couleur DMC (3-ème cordonné), les cordonnées RGB de la couleur (respectivement 0, 1 et 2 -ème cordonnées). Les fonctions **getRed**, **getBlue** et **getGreen** prennent la couleur DMC en paramètre et revoie la cordonnée correspondant de la valeur RGB. La fonction **GetValDmc** prend comme paramètre la couleur RGB et recherche la couleur DMC la plus proche et la renvoie.

On a décidé d'utiliser un autre format de couleur pour comparer si la même couleur DMC était donné. Le format HSL utilise la teinte, la saturation et la luminosité pour définir une couleur. Un fichier a été créé de la même manière que pour le format RGB. Le fichier donne sur chaque ligne la couleur HSL et le code DMC, le format RGB a juste été convertit au format HSL comme il n'existe pas de liste faite avec ce format.

La classe **HslToDMC** sert ainsi de la même façon que la classe **RgbToDmc** de liaison avec le format HSL. Les couleurs sont aussi stockés dans un tableau. La fonction **Initialization** est la même que celle de la classe **RgbToDmc**. Les fonctions **getHue**, **getSaturation** et **getLightness** ont pour paramètre le code DMC et renvoie la cordonné correspondante du format HSL. La fonction **GetValDmc** prend en paramètre la couleur HSL et compare chaque couleur DMC avec la couleur placée en paramètre pour renvoyer le code DMC de la couleur la plus proche.

Le calcul de longueur de fil se base sur l'utilisation d'une toile **Aïda 5.5**, ce qui veut dire que chaque carreau de la toile mesure 0.5cm sur 0.5 cm. C'est le tissu standard et qui est recommandé pour réaliser du point de croix. On commence par prendre en compte la quantité de fil qu'il faudra pour faire un noeud avant de commencer à broder avec notre fil. Cette quantité vaudra 2cm et sera la même pour le noeud de fin. On définit aussi la longueur qu'il faut pour broder une croix. Pour broder une croix, on va d'abord piquer l'aiguille à un coin du carreau pour la repiquer dans le coin qui est à la diagonale du premier coin, ce qui nous fait une demi-croix. Après on repique l'aiguille dans le coin juste au-dessus du deuxième coin, pour enfin piquer l'aiguille dans le coin à la diagonale du troisième coin. (Image). Il faudra remonter l'aiguille à la verticale pour faire la croix suivante. Ceci nous donne comme longueur de fil pour une croix : (**2 * Math.Sqrt(0.5)) + 2*0.5**

La longueur de fil pour réaliser un **noeud et une croix** nous permet calculer la longueur de fil dont on a besoin pour broder toute l'image. On calcul une longueur pour chaque couleur de fil et correspondante aux couleurs de l'image après conversion.

Pour cela, on utilise une fonction **WireSize** qui prend en paramètre la couleur de notre fil en question et l'image dont on veut faire un patron de broderie. Cette fonction va initialiser un compteur length à 1, qui nous permettra de savoir si plusieurs pixels à la suite sur une même ligne sont de même couleur, et lancer une double boucle for qui va regarder chacun des pixels de l'image et comparer sa couleur avec la couleur passée en paramètre. Si les couleurs coïncident, on vérifie que la couleur du pixel voisin de droite possède aussi sa couleur qui coïncident. Si c'est le cas on **incrémente notre compteur length**. Sinon on calcule la longueur de fil qu'il faut pour broder la suite de pixels, on l'ajoute à la longueur totale de fil qu'on aura besoin pour notre couleur et on remet noter compteur à 1. On fera de même si on arrive à la fin d'une ligne dans l'image. On continue ainsi jusqu'à atteindre le dernier pixel de l'image.

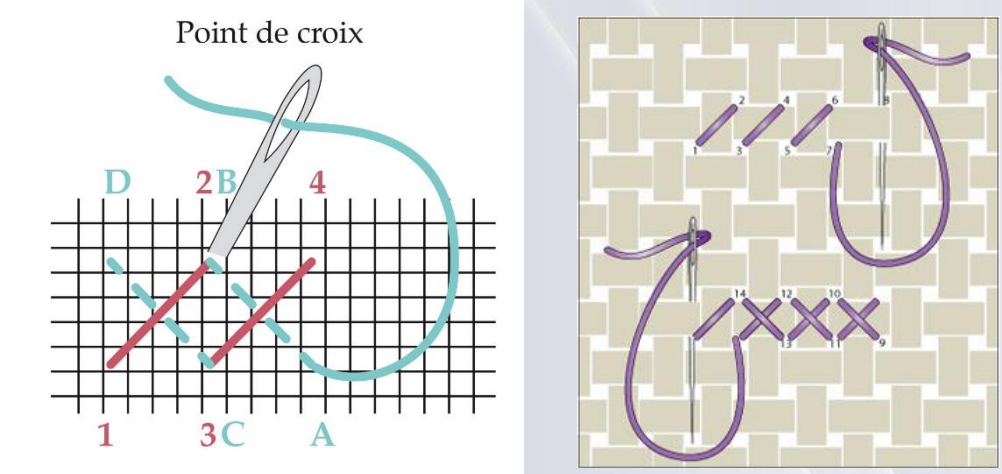


Figure 15 : Illustration d'un point de croix en broderie

Analyse des performances

Lorsque nous avons commencé à pouvoir afficher une grille de broderie (sans K-Means ni conversion vers des couleurs DMC), nous avons remarqué que l'affichage de la grille semblait prendre trop de temps : une dizaine de secondes pour une image **100x100**

Après analyse du code qui créé la grille de broderie, nous avons remarqué qu'une partie du code prenait beaucoup trop de temps pour ce qu'elle faisait : compter le nombre de pixels pour chaque couleur

```
2405 (54,11 %)  137  foreach (Wire? wire in from wire in WireArray let color1 = ((SolidColorBrush)wireTable[i, j]).Color let color2 =
                       ((SolidColorBrush)wire.Color).Color where color1 == color2 select wire)
                138  {
                139      found = true;
                140      wire.Quantity++;
                141  }
```

Figure 16 : Ancien code

Après optimisation, nous obtenons ce code qui dure un temps beaucoup plus raisonnable, visible par le temps d'exécution à gauche qui passe de **2405ms à moins de 10ms**

```
                129  // Use a dictionary to count the quantity of each color
                130  Dictionary<Color, int> colorQuantity = new();
                131  for (int i = 0; i < width; i++)
                132  {
                133      for (int j = 0; j < height; j++)
                134      {
                135          count++;
1 (0,05 %)      136          ShowProgression(value: count, max: imageSize);
                137
                138          // We can't use wireTable[i, j] directly,
                139          // because even those with the same color are considered not the same object
                140          Color color = ((SolidColorBrush)wireTable[i, j]).Color;
                141
1 (0,05 %)      142          if (colorQuantity.ContainsKey(color))
                143          {
                144              colorQuantity[color]++;
                145          }
                146          else
                147          {
4 (0,21 %)      148              colorQuantity.Add(color, 1);
                149          }
                150      }
                151  }
                152
                153  // Add the wires to the WireArray
1 (0,05 %)      154  foreach (KeyValuePair<Color, int> color in colorQuantity)
                155  {
                156      SolidColorBrush scbColor = new(color.Key);
                157      // TODO use the RGB values of color.Key to add color number, type and name here when the work is done
                158
                159      WireArray.Add(new Wire(scbColor, 404, "DMC", "White", color.Value));
                160  }
```

Figure 17 : Nouveau code

Une fois K-Means intégré à l'UI, nous avons également remarqué que celui-ci prenait beaucoup de temps, mais nous ne savons pas vraiment si cela était normal. Malgré tout, en analysant le code avec le profileur nous avons quand même trouvé cette fonction qui est exécuté très souvent (plusieurs fois par sous-itérations) et avons donc essayé de l'optimiser.

```
int clusterId = Centroids!
    .OrderBy(v => GenericVector.Distance(vector, v.Value))
    .Select(v => v.Key)
    .FirstOrDefault();
return clusterId;
```

BrodUI.KMeans.KMeans.GetNearestCluster(BrodUI.KMeans.GenericVector) 5548

Figure 18 : Ancien code

Après optimisation, nous avons effectivement grandement amélioré le temps d'exécution.

```
int clusterId = -1;
double smallestDistance = double.MaxValue;

// Get the clusterId that has the smallestDistance with vector
foreach (KeyValuePair<int, GenericVector> v in Centroids!)
{
    double tempDistance = GenericVector.Distance(vector, v.Value);
    if (tempDistance < smallestDistance)
    {
        clusterId = v.Key;
        smallestDistance = tempDistance;
    }
}
```

BrodUI.KMeans.KMeans.GetNearestCluster(BrodUI.KMeans.GenericVector) 266

Figure 19 : Nouveau code

L'optimisation consiste simplement à recoder la fonction pour s'assurer qu'elle exécute uniquement le code dont elle a réellement besoin. Ici, la fonction OrderBy triait tous les centroids par leur distance alors qu'on avait seulement besoin du premier.

Une dernière optimisation réalisée fût sur le nombre de sous-itérations, qui parfois allez jusqu'au maximum (100) même si la précédente sous-itération était identique.

Le problème venait du fait que lorsque l'on divise un double par 0, on obtient NaN ; et lorsque l'on comparait la sous-itération actuelle avec la précédente, on exécute "NaN != NaN" qui retourne true : aux yeux du programme, la sous-itération était différente de la précédente et il fallait donc continuer les sous-itérations.

Analyse des résultats

Pour analyser les résultats nous allons utiliser des exemples et les étudier. A travers ces 4 exemples nous allons pouvoir voir à quel point notre programme est efficace, ses points forts, ses points faibles.

Pour le premier exemple, nous allons utiliser cette **image format png de résolution 1065x827** :

Figure X : Image PNG Myrtille

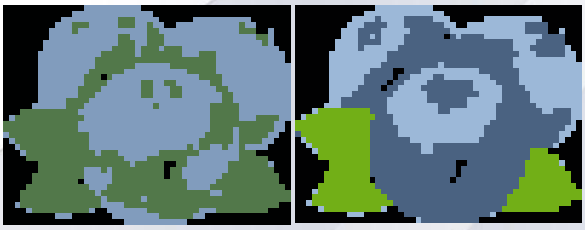


Figure X : Myrtille 3 et 4 couleurs (50x38 pixels)

Étant donné que l'image est un fichier au format png, le fond est transparent mais notre programme ne pouvant pas traiter la transparence, le fond est considéré comme noir, ce sera le cas pour toutes les images de cette partie.

Nous allons maintenant montrer 2 résultats ayant des paramètres similaires sauf au niveau du nombre de couleur. Voici donc deux images de taille **50x38 avec 3 et 4 couleurs respectivement**

On voit très bien que pour 3 couleurs, on ne peut pas reconnaître l'image de départ avec le bleu foncé et le vert qui se retrouve de la même couleur, ce qui change la géométrie des objets présent, alors que **pour 4 couleurs**, l'image ressemble à celle de départ et nous pouvons reconnaître de façon plus aisé les objets.

Nous allons ensuite voir le résultat du programme sur **une image de résolution 679x1175** et ses transformations en **images de taille 28x50 et de 3, 4 et 8 couleurs respectivement**.

Figure X : Image PNG Tigre et Tigres 3, 4 et 8 couleurs (28x50 pixels)

Encore une fois, pour **3 couleurs** l'image n'est pas reconnaissable cependant elle ne l'est pas non plus pour 4 couleurs. Pour **8 couleurs**, le tigre devient reconnaissable quand l'image est petite et n'est pas zoomé.

Dans cet exemple, on voit une anomalie, en effet, entre les pattes du tigre, sur l'image d'origine, nous pouvons voir le fond de l'écran, mais cette zone disparaît sur les images résultats (ou plus exactement devient un point noir). Ce problème est dû à la **diminution de la résolution**. Nous avons donc pu identifier deux raisons pouvant causer un changement **dans la géométrie des objets de l'image**.

Une autre observation est que le nombre de couleurs minimum pour avoir une image reconnaissable et satisfaisante n'est pas le même pour toutes les images. En effet, la première image était reconnaissable dès **4 couleurs** alors que la deuxième a nécessité **8 couleurs**. Une première hypothèse serait que la cause vient du nombre de couleurs sur l'image d'origine. Cela ne semble pas être le cas ici car dans la première image nous pouvons identifier trois groupes de couleurs (vert, bleu et le fond noir) et trois aussi dans la deuxième (orange, blanc et noir pour les rayures et le fond) pour des nombres de couleurs nécessaires différents. Une autre hypothèse serait que la cause se trouve dans une différence de la diminution de la résolution. Cela pourrait être possible. En effet, pour la première image, la résolution est divisée par **21.3** là où pour la deuxième la résolution est **divisée par 23.5**. Cependant après un test supplémentaire, en divisant la résolution de la **deuxième image par 21.3 avec 4 couleurs** on obtient :

Figure X : Tigre 21.3

Le résultat n'est pas meilleur que la version précédente (excepté concernant l'espace entre les pattes). Cette deuxième hypothèse est donc rejetée pour ce cas. La dernière hypothèse est que la cause soit la complexité de l'image, la présence de détails nécessaires à une bonne reconnaissance de l'image. En effet, dans la deuxième image, **il y a les rayures et le « visage » du tigre** sont nécessaires pour bien reconnaitre un tigre or ces deux objets sont soit eux-mêmes des détails (les rayures) ou composé de détails (les yeux). Ces détails étant de petite taille, ils ont tendances à disparaître lors de la diminution de la résolution. Ainsi, nous avons besoin d'augmenter le nombre de couleurs pour que ces détails soit mis en avant. Pour être sûr de ces conclusions, nous allons analyser une nouvelle image. **Celle-ci a une résolution de 992x992**. Les résultats proposés ont une résolution de 50x50 et ont 4, 7 et 8 couleurs respectivement.

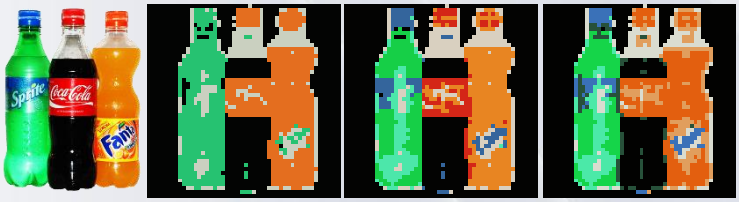


Figure X : Bouteilles PNG et Bouteilles 4, 7 et 8 couleurs (50x50 pixels)

Cette image contient des formes simples et reconnaissables (les bouteilles), des détails complexes (les étiquettes et marques) et plus de couleurs que les images précédentes. Elle va donc nous permettre de tester une nouvelle fois **nos hypothèses 1 et 3**. Dès 4 couleurs, on arrive à reconnaître la forme générale des objets, c'est-à-dire identifier la présence de bouteilles. En réalité, pour 3 couleurs, on pouvait déjà voir la forme bouteille mais il n'y avait aucun détail sur les bouteilles. **A 7 couleurs**, on retrouve les couleurs principales des bouteilles et une partie des détails (plus précisément le logo Fanta™). Cependant, à 8 couleurs, on perd la couleur rouge en échange de plus de détails sur les ombres et les lumières.

Comme on peut le voir, les formes simples sont reconnues avec peu de couleurs mais on a besoin de plus de couleurs pour identifier plus précisément les informations (ici les marques des sodas) soit par leurs couleurs représentantes soit en rendant plus visible les détails.

En conclusion, le nombre de couleurs nécessaires change selon plusieurs paramètres, tel que la complexité de l'image et le nombre de couleur d'origine. Cela a pour effet que l'utilisateur doit utiliser l'outil avec plusieurs paramètres jusqu'à trouver une solution qui lui convient et qui équilibre nombre de couleurs/fils et la bonne reconnaissance de l'image, l'un au détriment de l'autre. Pour éviter ce problème du nombre de couleurs changeant pour chaque image, l'utilisation de l'algorithme Mean-Shift à la place de K-Means aurait pu être utile mais cela aurait forcé les utilisateurs sur le nombre de couleurs de fil différentes. Il nous aurait fallu utiliser une variation de cet algorithme où l'on pourrait sélectionner le nombre maximum de groupe mais nous ne connaissons pas si un tel algorithme est possible et il aurait fallu l'implémenter nous-même contrairement à ce qu'il nous a été demandé.

IV) Fonctionnalités supplémentaires

Dans cette partie, nous allons voir des points supplémentaires par rapport à l'application ou à son développement. Ces points sont assez importants dans le **processus de création de BrodUI** et sont également important pour **les poursuites d'études de certains développeurs du projet**.

Tests unitaires et code coverage

La première amélioration réalisée sur notre application est **les tests unitaires**. En effet, la réalisation de ceux-ci nous est rapidement apparue comme une évidence Les tests unitaires de la plupart des méthodes des **classes Helpers**, **KMeans** et **Models** ont donc été réalisés. Cependant, nous voulions aller plus loin en vérifiant la couverture de tests de l'ensemble de ces méthodes. En effet, il se peut que certaines conditions des méthodes ne soient pas testées dans les tests. Cette partie est d'autant plus importante pour certains étudiant, voulant intégrer **le Master 2 ISL (Ingénierie et Systèmes Logiciels) à Besançon**.

Ayant **Github Student** (grâce à l'université), nous pouvons bénéficier gratuitement de l'intégralité des services **Jetbrains** gratuitement. Cela inclut **ReSharper**, un lot d'extensions pour Visual Studio visant à aider le développement, mais surtout ce qui nous intéresse ici, les tests. Cet outil nous permet de vérifier rapidement nos tests déjà effectués, mais surtout de bénéficier d'un outil de couverture de tests (**Tests Coverage**).

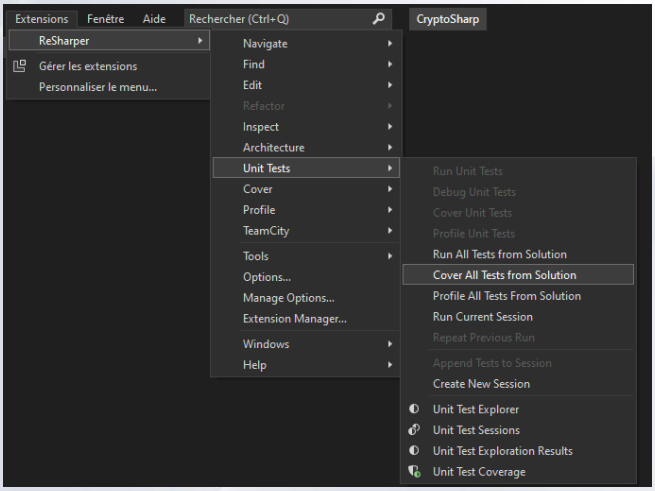


Figure X : Activation du coverage sous Visual Studio avec Resharper

Nous obtenons donc deux documents d'informations. Le premier présente l'exécution des tests et si les tests passent. Nous pouvons voir ici que nous avons à ce jour, **69** méthodes de tests différents dans le projet. Nous pouvons voir ici que tous les tests passent :

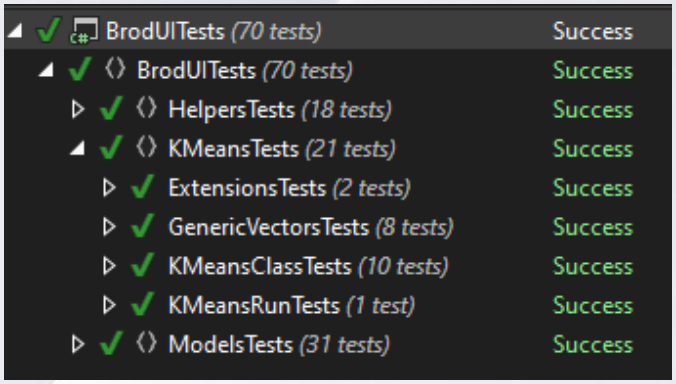
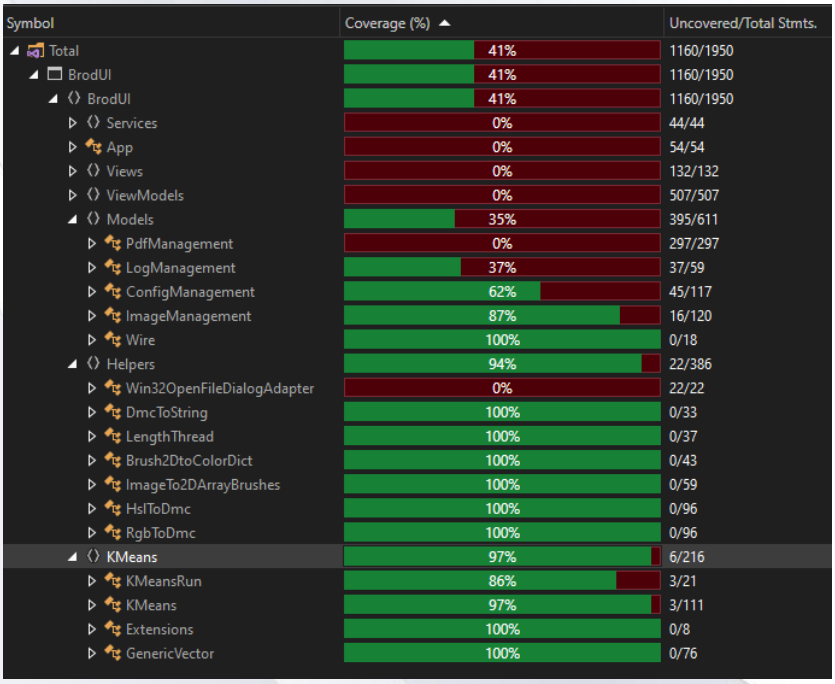


Figure X : Nombre de tests et tests qui passent avec succès

Cependant, c'est le deuxième document qui est vraiment important pour nous : **l'Unit Test Coverage**. Il présente une arborescence comme suit : Les projet → Les classes → Les méthodes. Ainsi pour chaque projets, classes et méthodes, on récupère l'informations de si tout est testé dedans. Prenons l'exemple d'une méthode avec un if puis else, il se pourrait qu'on ait testé que le if, ou bien que le else. Le code coverage nous dira donc le pourcentage de couverture de tests de chaque méthode.



Symbol	Coverage (%)	Uncovered/Total Stmts.
Total	41%	1160/1950
BrodUI	41%	1160/1950
BrodUI	41%	1160/1950
Services	0%	44/44
App	0%	54/54
Views	0%	132/132
ViewModels	0%	507/507
Models	35%	395/611
PdfManagement	0%	297/297
LogManagement	37%	37/59
ConfigManagement	62%	45/117
ImageManagement	87%	16/120
Wire	100%	0/18
Helpers	94%	22/386
Win32OpenFileDialogAdapter	0%	22/22
DmcToString	100%	0/33
LengthThread	100%	0/37
Brush2DtoColorDict	100%	0/43
ImageTo2DArrayBrushes	100%	0/59
HslToDmc	100%	0/96
RgbToDmc	100%	0/96
KMeans	97%	6/216
KMeansRun	86%	3/21
KMeans	97%	3/111
Extensions	100%	0/8
GenericVector	100%	0/76

Figure X : Code Coverage de BrodUI

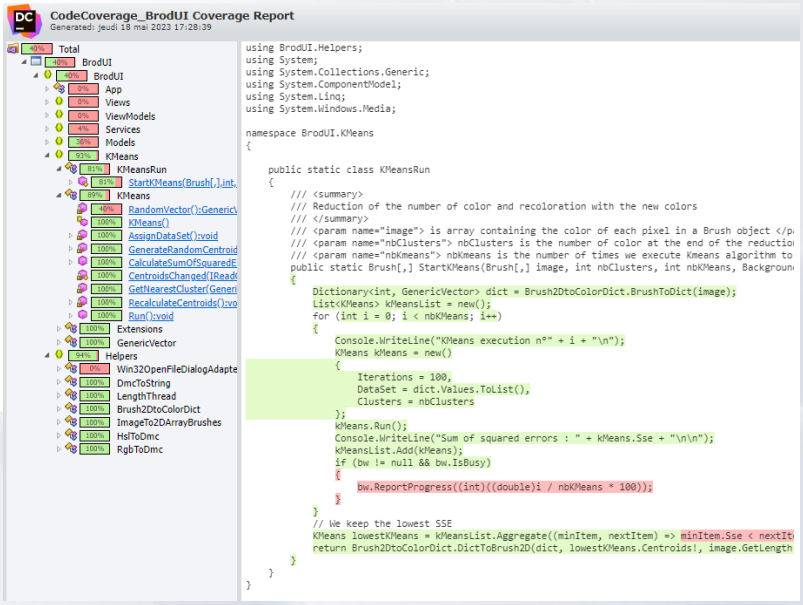


Figure X : Export du code coverage en HTML

Un document de couverture de code a été généré, vous le retrouverez dans **/Coverage/CodeCoverage_BrodUI.html**. En voici un court aperçu, il nous montre ainsi quelles lignes sont testées efficacement en dehors de Visual Studio ce qui permets un partage facilité. Il est également plus ergonomique :

Documentation Doxygen

L'autre amélioration du projet, c'est la documentation. En effet celle-ci est super importante. Elle permet tout d'abord de nous souvenir de ce que l'on a fait, d'à quoi sert une méthode ou une fonction, mais elle permet également à un développeur tiers de comprendre ce que l'on a fait. Notre projet étant Open Source et disponible à tous les développeurs et surtout à tout le monde, il est important que celui-ci soit complètement compréhensible. Tout est donc rédigé en Anglais (commits, documentations, variables, méthodes etc.), et une documentation **Doxygen** a été réalisée. Voici un exemple de commentaire permettant la documentation de notre projet :

```
/// <summary>
/// Constructor of the class
/// </summary>
/// <param name="color">Color of the wire</param>
/// <param name="number">Number of the wire</param>
/// <param name="type">Type of the wire</param>
/// <param name="name">Name of the wire</param>
/// <param name="quantity">Quantity of the wire</param>
7 références
public Wire(Brush color, int number, string type, string name, int quantity)
```

Figure X : Exemple de bloc de documentation en C#

Une fois la documentation de chaque méthode de chaque classe de chaque projet réalisé, nous pouvons maintenant à l'aide de Doxygen et de notre fichier DoxyFile (fichier de configuration de la documentation Doxygen) générer un site HTML pour notre documentation. De plus après la génération de la documentation, le fichier « **DoxygenWarnings.txt** » nous donne les emplacements qui ne sont pas documentés. Ici le ce fichier est vide, tout est bien documenté.

Il nous reste plus qu'à lancer la documentation, celle-ci est disponible à la racine, puis **/Doc/html/index.html**. En voici un court extrait :

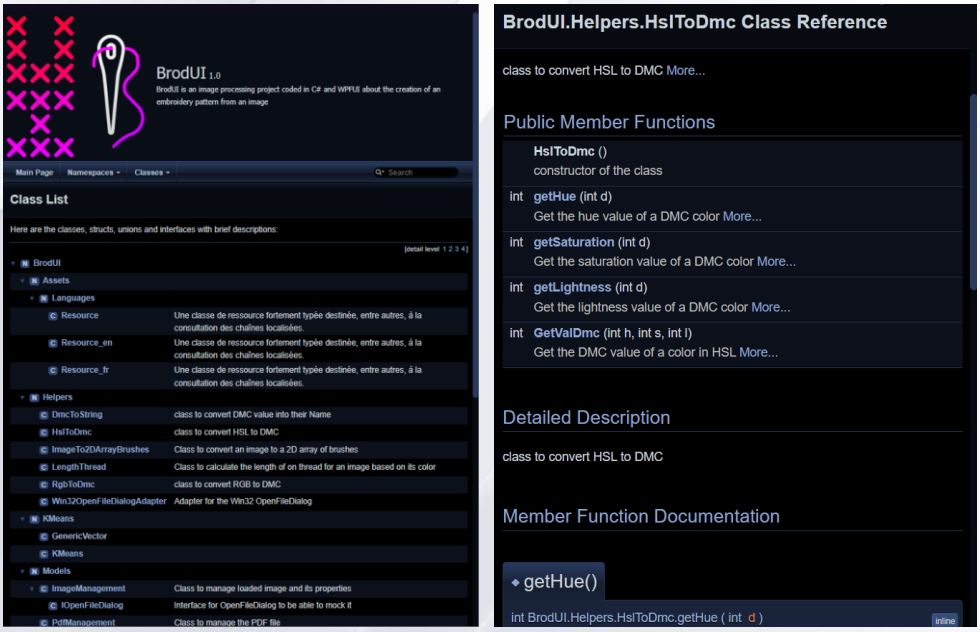


Figure X : Extrait de la documentation Doxygen (HTML) de BrodUI

Organisation du travail

Le travail a été découpé dès le départ en fonction des affinités de chacun avec le projet et les ambitions de chacun en termes de poursuite d'étude. Vous retrouverez cette répartition en introduction. Ainsi les membres du projet voulant poursuivre dans l'image se sont occupés de la partie traitement d'image et les autres se sont occupés du logiciel et de l'implémentation des algorithmes de traitement de l'image.

Des rôles ont également été définit au sein du groupe, ainsi nous avons :

- Samuel LACHAUD – Chef de projet
- Zeina Hélène AL HALABI – Responsable communication
- Loïs PAZOLA – Responsable qualité

Des réunions entre le tuteur de projet (Mr Migniot) **et notre groupe via Microsoft Teams** afin de s'assurer que le projet parte dans la bonne direction et qu'il avance bien.

Pour le travail au sein de l'équipe, nous avons utilisé **Discord** pour la communication et la gestion du projet via des salons dédiés, et différents jalons. Puis nous avons utilisé git pour la mise en version du code et la gestion de celui-ci.

Conclusion

Pour démarrer cette conclusion, vous trouverez à coté de ce document, le projet Visual Studio, le document de code coverage, la documentation Doxygen, ainsi que le code compilé dans un exécutable tout prêt.

Tout d'abord, nous allons parler de notre ressenti sur ce projet. En effet, il est important pour nous de passer par la pratique pour comprendre un principe, des algorithmes, et plus encore, il est important pour nous de savoir à quoi sert réellement ce qu'on apprend. Le fait de passer par un projet, qui plus est, en langage libre nous a permis de réaliser l'algorithme et les principes vus en cours, de pouvoir, dans une démarche scientifique, expérimenter celui-ci.

Ce projet a été très enrichissant sur la partie traitement d'image. En effet le fait d'appliquer dans un cas concret, de faire des tests, d'expérimenter K-Means nous a permis de mieux comprendre comment il fonctionne (même pour les personnes n'ayant pas travaillé dessus, ne l'ayant uniquement implémenté dans le logiciel).

Ce projet a été également très enrichissant sur la partie développement logiciel. Nous avons pu prendre grand soin de faire un logiciel optimisé, bien codé, bien documenté et surtout bien testé. Nous avons pu expérimenter des bibliothèques graphiques comme WPFUI qui nous font sortir de notre zone de confort, et de ce fait nous avons essayé de fournir un logiciel le plus professionnel possible (un professionnel du domaine de la broderie est d'ores et déjà intéressé par ce logiciel).

Pour fini, le projet est open source et disponible à l'adresse suivante : https://github.com/samlach2222/BrodUI. Merci d'avoir lu ce rapport et nous espérons qu'il aura répondu à vos attentes et que tout y a bien été expliqué. Si des détails ne sont pas clairs ou si vous voulez des informations supplémentaires, nous sommes bien sûr disponibles via nos adresses électroniques étudiantes.