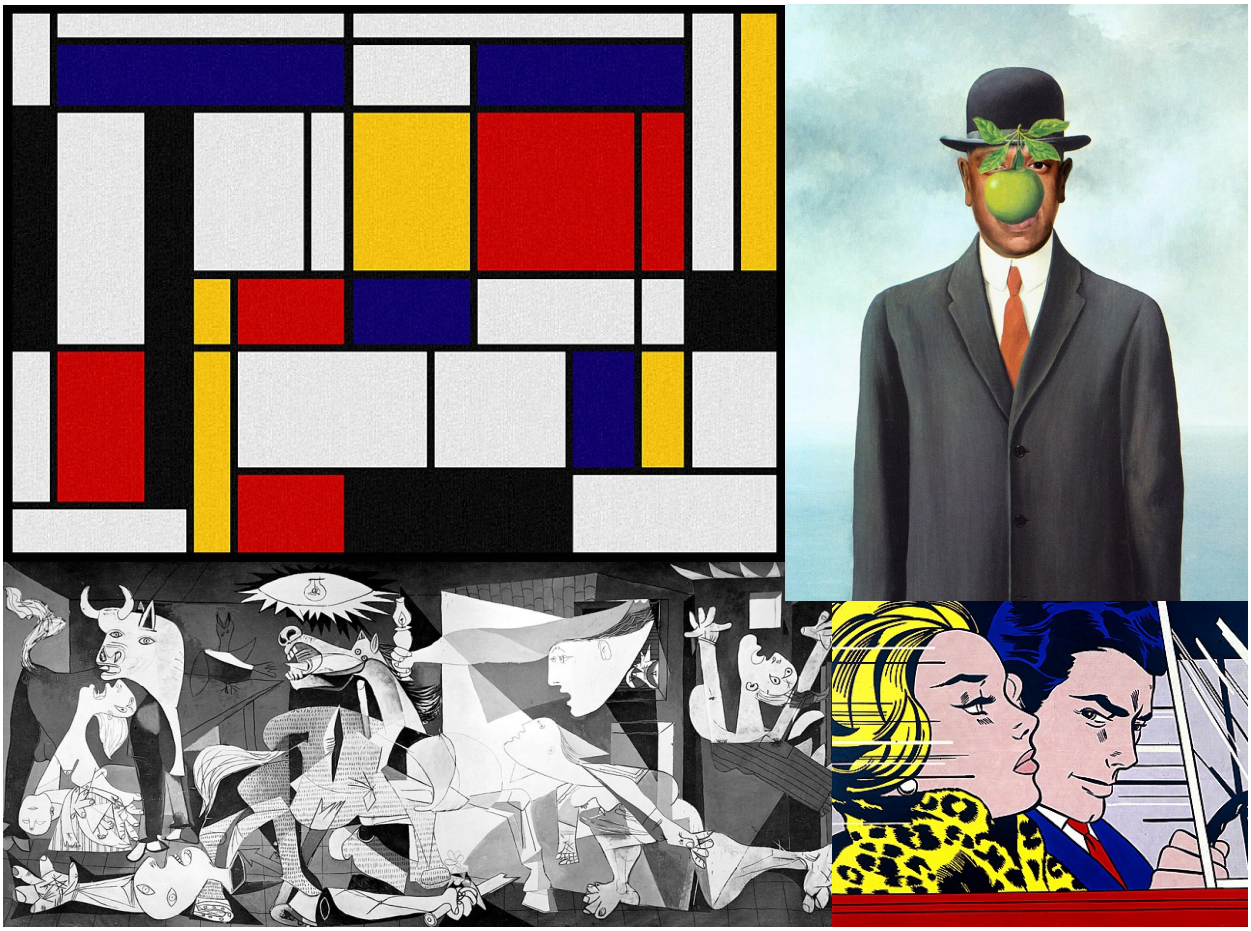
INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

APN



L'Art Pour les Nuls

☼ Promo n°59 ☼
Groupe 50 lanière M

MATHON Lara
MOUTAOUAKIL OUDGHIRI Youssef
RABAT Thomas
REGNAULT Marie

SOMMAIRE

I. Introduction

Qui n'a jamais rêvé de connaître l'histoire d'une peinture qu'on voit dans un musée ?

L'application que nous avons développé, l'**Art Pour les Nuls**, permet justement, à partir d'une photo de peinture, de l'analyser et de renvoyer les données la concernant.
En effet, notre programme repose sur l'analyse des couleurs de la photo ainsi que sur la comparaison avec les peintures enregistrées dans la base de données.
Après son traitement, l'application sera capable de vous préciser le titre de l'œuvre, l'artiste l'ayant peinte, mais aussi une description détaillée de la peinture.

Pourquoi utiliser l'application APN ?

Cette application est simple d'utilisation. Il suffit de prendre une photo pas forcément dans des conditions optimales, et puis de spécifier les coins de la peinture sur l'image.
En effet, comme décrit dans ce rapport, l'application a été conçue pour adapter l'analyse en fonction du format (paysage, portrait), de l'angle de vue et de la luminosité de la photo !

Un bonus par rapport au cahier des charges ?

L'**APN** ne se limite pas à identifier des peintures prises en photo.
Face à la taille astronomique de peintures à ajouter dans la base de données pour que l'application devienne réellement utilisable, nous avons décidé de rendre cette application Open Source. En effet, cette dernière permet à n'importe qui d'ajouter des œuvres à la base de données (base de données locale), et de compléter leurs informations.

Notre application est-elle infaillible ?

La réponse est évidente, comme c'est le cas pour tous les programmes qui fonctionnent sur la base de données statistiques et plusieurs facteurs aléatoires. Dans notre cas, les difficultés rencontrées concernaient le cadrage de l'image, l'angle de prise de vue, la luminosité, le contraste, le type de lumière sous lequel à été prise l'image... Nous avons ainsi mis en place certaines méthodes pour contourner ces problèmes, et ainsi avoir la comparaison la plus fiable possible.

Quels sont les principes clés de notre reconnaissance d'images ?

Premièrement, nous avons choisi de ne prendre qu'en compte la couleur du pixel, et non pas de la luminosité et saturation. Après avoir basé notre reconnaissance de couleurs sur les paramètres RGB, nous avons finalement opté pour la méthode HSV. Bien qu'au début nous pensions qu'elle entraînait une confusion entre les couleurs blanc et noir, nous avons contourné le problème en utilisant des restrictions supplémentaires sur la sélection des couleurs à analyser.

La comparaison des couleurs ne se fait pas points à points, mais d'un point comparé à carré de points. Ainsi, on évite les problèmes liés à un cadrage trop approximatif de l'image par l'utilisateur lorsqu'il sélectionne les coins de la peinture.

Quelque soit l'angle de vue de la photo, notre algorithme s'adapte et sélectionne avec précision les pixels pour analyser l'image et la comparer aux peintures de la base de données.

II. Description des classes

a- *La classe mère « Fenetre »*

La classe *Fenetre* est la classe gérant les interfaces de notre application. Elle permet d'afficher les différentes fenêtres guidant l'utilisateur tout au long de sa recherche, à savoir les classes *FenetreInitialisation*, FenetreSelection, *FenetreResultats* et *FenetreInconnue*.

Pour cela, elle est étendue JFrame et reçoit en paramètre un int numClasse, qui désigne la sous-classe à ouvrir parmi les quatre possibles.

Par ailleurs, elle implémente également un WindowListener et les méthodes qui l'accompagnent afin d'interroger et retarder la fermeture de la classe *FenetreSelection*.

1- FenetreInitialisation :

FenetreInitialisation est la première interface ouverte par l'application. Elle constitue la page d'accueil de l'utilisateur, qui peut choisir entre importer la photo d'une peinture qu'il souhaite identifier, ou ajouter une nouvelle image dans la base de données.

Pour ce faire, la classe est étendue JPanel et implémente un ActionListener. L'image à reconnaître ou à ajouter est sélectionnée dans la méthode *actionPerformed* au moyen d'un JFileChooser. Un JFileFilter est également utilisé par *FenetreInitialisation* dans le but de restreindre les extensions visibles et sélectionnables par le JFileChooser. A partir de là, soit *FenetreSelection* est ouverte (et le premier bouton devient inactif et invisible), soit une image est ajoutée à la base.

2- FenetreSelection

FenetreSelection correspond à la fenêtre de sélection des quatre coins de la peinture. Celle-ci est ainsi dotée d'un MouseListener qui permet de récupérer les coordonnées des coins de la peinture, puis de l'envoyer à la classe *PeintureAnalysee.* Pour des raisons de commodité, l'instanciation de notre base de données se fait à partir de cette classe. Cette dernière contient aussi les méthodes permettant de créer l'instance *PeintureAnalysee,* à partir de l'image sélectionnée par l'utilisateur dans *FenêtreInitialisation* et des coordonnées des coins de l'image récoltés. Enfin, on trouve la méthode *findPeinture,* qui permet avec la méthode de comparaison des œuvres (de la classe *PeintureData),* de retrouver les deux œuvres qui, statistiquement, ont la plus grande probabilité de correspondre à l'image ajoutée par l'utilisateur.

3- FenetreResultats

Cette classe permet d'afficher la peinture identifiée par l'application. Pour se faire, elle reçoit les deux peintures qui correspondent le plus à l'image à identifier. Dans un premier temps, elle affiche la première, accompagnée du nom de l'œuvre, de l'artiste et sa description. L'utilisateur à la possibilité d'appuyer sur deux boutons :

- "*Merci pour les informations* ": Cela signifie qu'il s'agit de la bonne œuvre : on ouvre donc une nouvelle fenêtre d'initialisation.
- "*Ce n'est pas le bon tableau...*": S'il s'agit de la première œuvre, la fenêtre est actualisée avec la seconde œuvre. Si la seconde œuvre n'est pas non plus la bonne, une fenêtre inconnue est ouverte.

4- FenetreInconnue

FenetreInconnue peut être ouverte à deux occasions: après *FenetreSelection,* ou après *FenetreResultat.* Dans le premier cas, l'application ne reconnaît pas l'image ; dans le second, elle a proposé deux œuvres erronées avant d'ouvrir *FenetreInconnue.* Par ailleurs, une fois affichée, *FenetreInconnue* offre deux options à l'utilisateur : fermer l'application, ou revenir à la page d'accueil (où le premier bouton est redevenu actif et visible). Pour cela, la classe est étendue JPanel et implémente un ActionListener.

En ce qui concerne les deux classes suivantes, *PeintureData* et *PeintureAnanlysee*, il faut savoir que malgré leurs appellations qui sont très proches, il n'y avait pas réellement d'intérêt à les faire hériter d'une classe mère. En effet, ces deux classes n'avaient pas assez d'éléments en commun, d'où notre décision de les séparer en deux classes distinctes sans aucun lien d'hérédité.

b- La classe *«PeintureData»*

La classe *PeintureData* correspond aux peintures de la base de données. Cette classe permet la création d'objets qui ont pour attribut un titre, un auteur, une description, et un tableau de coordonnées des couleurs caractéristiques de l'image. Ainsi, *PeintureData* dispose d'une méthode qui, à partir du nom de l'image correspondant à l'objet qu'on veut instancier, va créer une sorte "d'empreinte de couleurs". Cette dernière est en fait un tableau à deux dimensions (384*400), dont chaque case contient la coordonnée "hue"(teinte) d'un pixel, 384 correspondant au nombre de petits carrées analysés sur notre image, et 400 au nombre de pixels par carré. Enfin, on trouve la méthode de comparaison qui renvoie la probabilité (en pourcentage) que la peinture de la base de données soit la même que la peinture analysée. Cette comparaison se fait entre un point de l'image analysée et l'ensemble des points du carré de l'image de la base de données, afin d'éviter les soucis de cadrage approximatif. La comparaison se fait aussi sur la base du format des peintures : une peinture en paysage ne peut pas correspondre à une peinture en portrait.

c- La classe «PeintureAnalysee»

La classe *PeintureAnalysee* correspond à la peinture que l'on veut analyser. Celle-ci a deux fonctionnalités ou méthodes principales.

La première permet de répartir les points d'analyse à partir des quatre coins sélectionnés par l'utilisateur (dont on stocke la couleur), de manière à ce que leur couleur soit nécessairement comprise dans les carrés de pixels définis dans *PeintureData*. Ainsi, quelque soit l'angle de vue et la forme de la peinture (rectangle, trapèze, parallélogramme..), l'algorithme s'adapte parfaitement. Voici un exemple de photo prise d'un angle assez particulier, et les points calculés par l'algorithme pour en prélever la couleur.

Enfin à partir de ces points définis par la méthode précédente, la méthode *genererateColorData* permet de créer le tableau de taille 384 et qui contient la donnée “hue” des 384 pixels analysée.

d- La classe main *«testMain»*

Cette classe permet de lancer **APN** en créant la première fenêtre de l'application. Cette classe est simpliste, car les autres classes interagissent entre elles.

e- La classe *«BaseDeDonnees»*

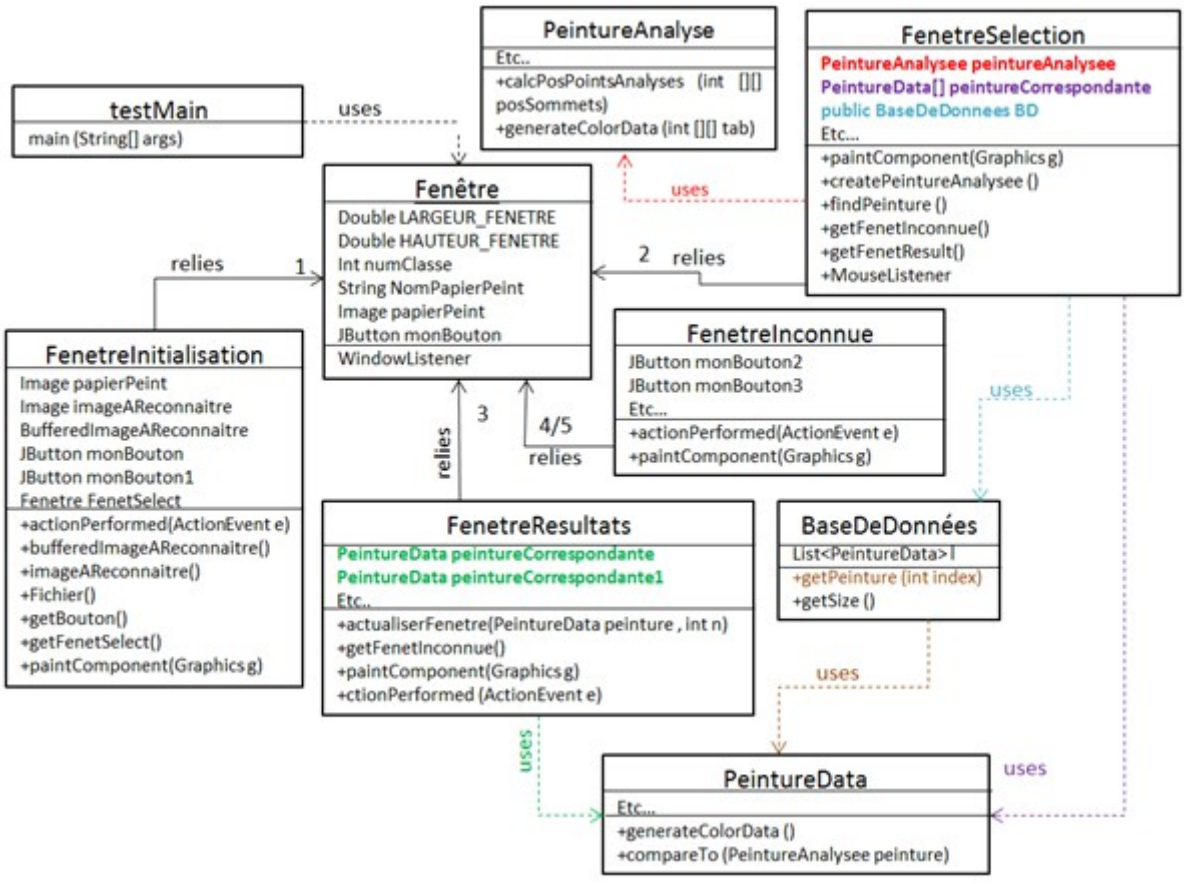
La création de la classe *BaseDeDonnees* a été décidée à la fin de notre projet afin de rendre plus simple la manipulation et la modification de notre base de données, à partir d’une classe qui lui serait propre. Ainsi, on trouve dans cette classe l’instanciation de tous nos objets PeintureData. Cette instanciation se fait en deux temps :

-L”instanciation à partir de la base de données non Open Source (cette instanciation fait partie de notre code et ne peut changer que si on change nous même notre code source)

-.L’instanciation automatique qui se fait en parcourant le fichier BDImagesOpenSource. Le programme parcourt les images qui s’y trouvent, et en fonction du titre des différentes images, crée les différentes instances nécessaires.

III. Interactions entre les classes

Voici le diagramme de classes simplifié du programme final :



IV. Fonctionnalités particulières utilisées

a- ArrayList

Déjà vues en TD, l'utilisation de listes chaînées permet de sauvegarder des données et de les organiser entre elles à l'aide de pointeurs, en s'affranchissant des problèmes d'ajout ou de suppression qu'on aurait rencontré en utilisant des tableaux.
Ainsi, cette structure correspondait parfaitement à notre volonté de créer une base de données regroupant les peintures authentiques.

Nous avons opté pour l'utilisation de l'*ArrayList*, car cette fonctionnalité de Java permet d'implémenter un tableau d'objet qui adapte continuellement sa taille.
Ainsi, dans la classe « *BaseDeDonnees* » nous avons initialisé une liste d'objets *PeintureData*. Chaque peinture de référence est créée en ajoutant un objet *PeintureData* à la liste.

b- Interfaces graphiques

Afin de rendre l'application **APN** interactive, il fallait que notre programme réagisse à des événements provenant de composants graphiques, comme par exemple des clics de souris sur des boutons. Il était donc nécessaire d'utiliser les packages *javax.swing* et *java.awt* de Java.
Par exemple, dans la classe *FenetreSelection* nous utilisons un *MouseListener* en redéfinissant ses méthodes (*mouseClicked, mousePressed..)* afin que le programme enregistre les coordonnées du point à chaque clic de souris et sélectionne les contours de la photo à analyser. De même on utilise également un *WindowListener* pour interroger et retarder la fermeture de *FenetreSelection.*

C-Création, récupération et traitement d'image

Lorsqu'un utilisateur ajoute en Open Source une œuvre, celle-ci est créée et enregistrée dans notre base de données OpenSource grâce à la méthode : *ImageIO.write(bufferedImage, "fomat-fichier", outputfile).*

La récupération de l'image à analyser se fait à partir du panel JfileChooser. La récupération des données correspondant à la couleur des pixels nécessite quant à elle l'utilisation des BufferedImage (c'est le seul format dont on peut récupérer la couleur d'un pixel), et des méthodes *getRGB(int x , int y),* et *Color.RGBtoHSB(int R , int G, int B , float[] hsv).*

V. Suggestions d'amélioration

Le projet, bien que fini, pourrait être approfondi. L'application **APN** pourrait, lors d'un ajout de peintures à la base de données, permettre à l'utilisateur de rentrer sa propre description de l'œuvre. En effet, à ce stade, il peut enregistrer uniquement le nom de l'œuvre et de l'artiste. On pourrait aussi ajouter la possibilité de modifier la description d'une peinture enregistrée, afin que l'utilisateur puisse ajouter les informations qu'il juge intéressantes.

L'aboutissement idéal du projet aurait été de développer l'**APN** sous forme d'application mobile afin de concrétiser son application (voire même de la présenter à des musées).

Nous aurions pu par exemple, améliorer la comparaison des œuvres en analysant à partir de quel intervalle de tolérance des couleurs on obtient le plus souvent la bonne peinture de référence. Ainsi, il faudrait faire des études statistiques sur un large panel d'images pour trouver les paramètres qui marchent le mieux.

VI. Carnet de route

Après le choix du sujet , nous avons organisé le projet en cinq classes provisoires : une classe contenant le main et la méthode de comparaison des peintures ; une classe contenant les informations sur la peinture ainsi que la base de données ; une classe permettant la sélection des coins de l'image, et finalement une classe affichant le résultat de la recherche.

Cependant, au vu de la longueur et de la complexité des programmes, nous avons préféré utiliser une classe de référence, comme la classe *«Fenetre »,* et d'en créer des nouvelles, comme par exemple une classe propre à la base de données au lieu d'initialiser la liste chaînée directement dans le main.

A la fin du projet, nous avons pu approfondir notre projet en recherchant les alternatives à nos problèmes sur Internet. Par exemple, nous avons rendu notre base de données *OpenSource* (cf. II.f)...

VII. Conclusion

En conclusion, ce projet d'informatique nous aura permis développer une application, en commençant par la prise de décision du sujet, jusqu'à l'élaboration finale.
Ce projet nous a permis de réutiliser des notions déjà abordées en cours. Étant responsables de l'avancée de notre programme, nous avons du nous investir entièrement en approfondissant des notions et des concepts informatiques que l'on ne connaissait pas.

De plus, ce projet nous aura appris à nous organiser en groupe. En effet, la communication entre les membres du groupe est primordiale, afin que nos actions soient coordonnées.