RAPPORT SAE32_2025
BUT Informatique - 2ème année
SAE3.2-Application "Primitive Image Format"

Youness BOULALAM,Algassimou DIALLO,Ayoub ANHDIRE

07 janvier 2026

Développement d'une application de conversion d'une image au format PIF (1ère image) + Affichage dans une fenetre d'une image contenue dans un fichier PIF (2ème image).

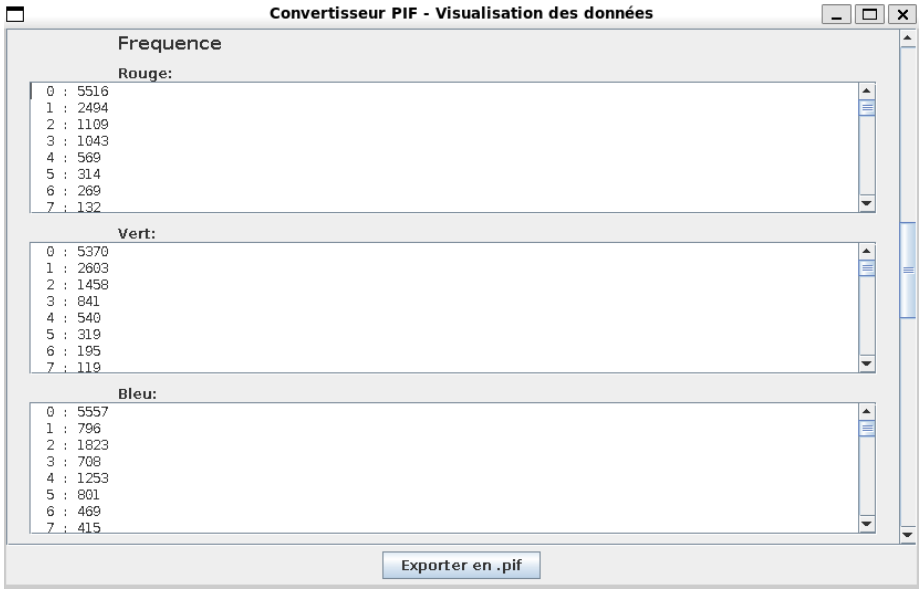

Figure 1: Interface principale du convertisseur , notamment avec les tables de fréquences

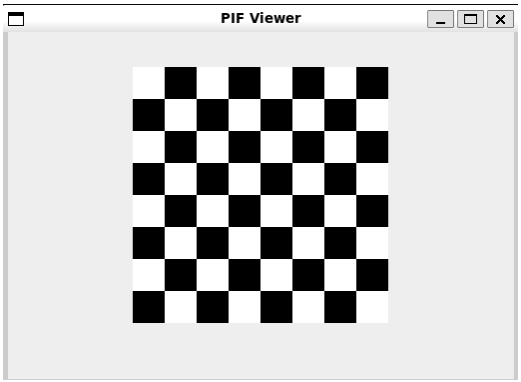

Figure 2: Interface principale du visualisateur ou l'utilisateur peut déplacer l'image avec la souris lorsqu'elle est trop grande

Comptes GIT utilisés pour ce projet

- **Youness BOULALAM** (Groupe 4) : `youness`
- **Algassimou DIALLO** (Groupe 4) : `Diallo-VM-fbleau`
- **Ayoub ANHDIRE** (Groupe 4) : `Ayoub ANHDIRE,Anhdire,Ayoub ANHDIRE`

Réalisé en JAVA : Architecture MVC
Dépôt Gitea : `https://grond.iut-fbleau.fr/dialloa/SAE32_2025`
Professeur : M.Luc Hernandez
Date de rendu : 11 janvier 2026

Sommaire

1 Introduction

Pour cette deuxieme SAE du semestre 3 , il nous a fallu réaliser **deux** programmes : un convertisseur d'une image au format png ou du moins supportable par la méthode *read* de la classe **ImageIO**. L'image sera donnée en argument ou sinon elle sera sélectionner par un **JFileChooser**(à part s'il y a un deuxième argument et qu'il represente un chemin et un nom acceptable). Pour ce programme , on aura une interface simple avec la tables des fréquences , la tables des codes initiaux (Huffman) et la table des codes canoniques. A la suite de cela , l'utilisateur a la possibilité ou non de pouvoir sauvegarder son image au format PIF.

Le deuxieme programme est un visualisateur ou celui-ci est censé afficher dans une fenetre une image contenue dans un fichier au format PIF. Celui-ci est donnée en argument ou la sélection du fichier sera réalise à l'aide d' un **JFileChooser**. La taille de la fenetre sera la taille de l'image à afficher avec une taille qui ne dépasse pas l'écran : si l'image est plus petite que le fenetre , elle sera centré et si elle ets plus grande que la fenetre : elle sera partiellement visible et pourra etre déplacé à la souris en maintenant le bouton gauche enfoncé.

Ce projet a été réalisé en trinôme, et nous avons développé l'application entièrementen Java, en respectant les bonnes pratiques vues en cours : architecture MVC, créationde makefile, et une interface graphique codée sans bibliothèque externe. Un soin particulier a été apporté à l'ergonomie : l'interface est pensée pour être simple, claire, et agréable à utiliser.

2 Répartition des taches

Nom	Tâches effectuées
Algassimou DIALLO	• Creation de la structure generale du projet (Dossier, UML) • Implémentation de BitOutputStream • Implémentation de BitInputStraam • Implémentation complète du contrôleur (ConverterController) • Génération et affichage des fréquences • J'ai aussi travailler sur le constructeur de HuffmanTree et Huffman node • Intégration de l'interface graphique (ConverterWindow) • Gestion du bouton Export et sauvegarde .pif dans un thread séparé • Creation et Test du MakeFile • Implémentation du PIFWriter (écriture des tables, pixels) • Implémentation du PIFReader (lecture, reconstruction, décodage) • Implémentation du contrôleur pour la vue
Youness BOULALAM	• Gestion des erreurs et messages utilisateur • Conversion RGBImage BufferedImage pour le Viewer • Conversion RGBImage BufferedImage pour le convertisseur • tache 1
Ayoub ANHDIRE	• Javadoc (Équipe) • Génération des codes Huffman • Génération des codes canoniques • Interface d'affichage des fréquences • Test de BitOutputStream • Lecture de l'en-tête • Diagramme de classe

3 Fontionnalités principales

3.1 Conversion au format PIF

4 Fonctionnalités principales

4.1 Conversion au format PIF

La conversion d'une image vers le format PIF repose sur plusieurs étapes : analyse des composantes RGB, calcul des fréquences, construction de l'arbre de Huffman, création des codes canoniques, puis écriture finale dans un fichier binaire structuré. Cette section présente les mécanismes mis en place et la contribution de chaque membre du groupe.

4.2 Contribution de Algassimou Pellel Diallo

pipeline global, le role du controleur, la navigation entre les etapes, les ecrans le workflow user i.e comment le user convetit un fichier ajout de diagramme

4.3 Contribution de Ayoub Anhdire

Un *arbre binaire* est une structure abstraite composé de noeuds dont la principale contrainte est qu'un noeud doit avoir au maximum deux enfants :

- un enfant gauche,
- un enfant droit.

4.4 Comment l'arbre d'Huffman est construit ? (Ayoub ANHDIRE)

Au préalable , pour construire l'abre binaire de Huffman , il nous faut calculer les fréquences pour les composantes R,G et B. Après avoir calculé ces fréquences , pour constuire l'arbre d'Huffman , il nous faut prendre les feuilles avec les plus basses fréquences , c'est-à-dire les plus proches de zéro : en l'occurence , il faut en prendre deux et à partir de ces deux feuilles , on crée un nouveau noeud qui a comme fréquence la somme des feuilles correspondantes. Lorsqu'il ne reste plus qu'une feuille et qu'on ne peut plus appliquer ce principe : alors cela signifie que c'est la racine de l'arbre. Elle est censé avoir la plus grande fréquence que les feuilles de départ. Autrement dit , plus la fréquence est grande , plus le code Huffman associé sera court.

Essayons d'expliquer ce principe avec un diagramme objet et un petit dessin de l'arbre. Prenons l'exemple le plus simple avec simplement deux feuilles, voici le diagramme objet basé sur notre code et un dessin de l'arbre de ce diagramme objet :

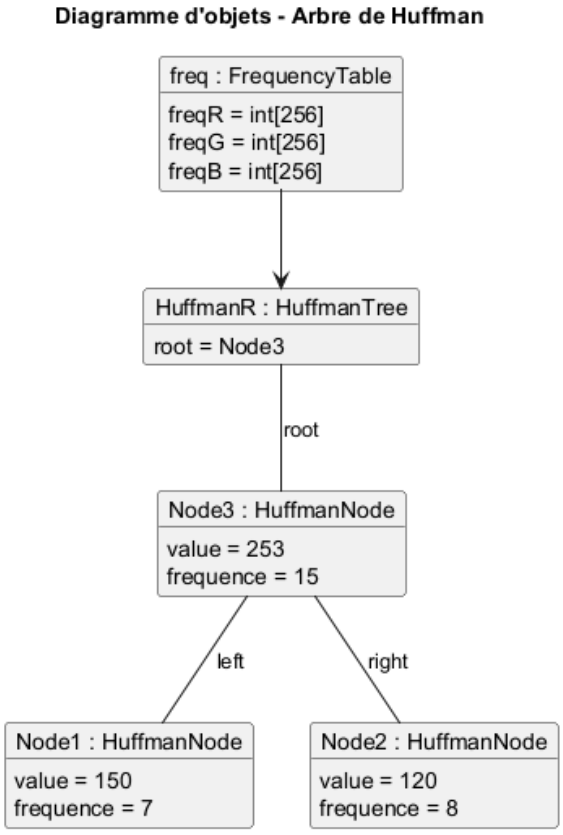

Figure 3: Diagramme Objet - Construction de l'arbre Huffman

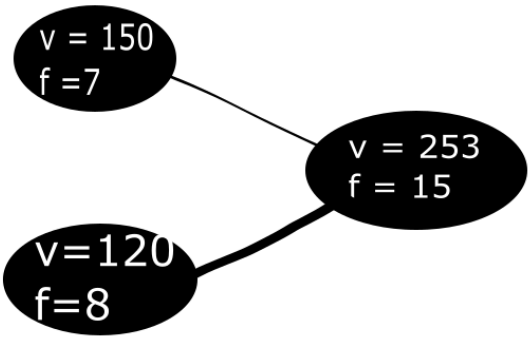

Figure 4: Dessin Arbre Huffman - Basé sur le diagramme objet de la figure 3

Puisqu'il y a trois composantes R,G et B , il est censé avoir 3 arbres Huffman mais pour simplifier la compréhension , nous n'en avons fait qu'un seul : celui de la composante Rouge. Expliquons le diagramme objet : nous avons un objet *freq* de la classe **FrequencyTable**. Cette classe permet d'initialiser les trois tableaux de fréquences(R,G et B) et ces tableaux sont passés en argument dans le constructeur de classe **HuffmanTree**.
Après ca on peut remarquer la présence de deux feuilles *Node1* et *Node2*. Elle ont comme fréquence respective 7 et 8. Comme répété plus haut , pour faciliter la compréhension ,nous avons choisis que deux feuilles. Voici la base de notre arbre. On remarque la présence d'un noeud *Node3* avec comme fréquence , il a été obtenu en faisant la somme des deux noeuds *Node1* et *Node2* et la fréquence obtenu est 15 , donc 7+8. C'est la seule feuille qui reste dans notre arbre : on en conlu que c'est la racine de notre arbre , comme en témoigne l'attribut root de la classe **HuffmanTree**.

Après avoir compris le principe de comment construire l'arbre Huffman , comment générer les codes Huffman ? Nous avons codé çela de manière récursive : si on saute vers un fils gauche on ajoute 0 et si on saute vers un fils droit on ajoute 1 : les codes sont enregistrés dans des dictionnaires : **Map<Integer,String>**. Pour chaque feuille , pour obtenir son code , on parcourt l'arbre de la racine jusqu'à la feuille.

La question qu'on se pose désormais : c'est est-ce-que cette solution est optimale ? La réponse est oui ! Pourquoi ? Comme on a placé les symboles fréquents près de la racine, la moyenne des longueurs de tous les codes est minimale. De plus, les codes sont différents, puisqu 'aucun code n'est le début d'un autre code, donc il n'y a pas d'erreur possible à la lecture. Même si certains symboles ont la même fréquence et que l'arbre peut être légèrement différent, la longueur moyenne reste toujours la plus courte possible. C'est pour cela que la génération des codes à partir de l'arbre de Huffman est optimale : aucun autre code ne peut donner une longueur moyenne plus courte pour les mêmes symboles.

4.5 Les codes canoniques et leur logique (Ayoub ANHDIRE)

Un *code canonique* est une version basé sur les codes Huffman : la longueur de chaque code Huffman est préservé mais les codes sont réorganisés de manière en commençant par les codes les plus courts. On commence par trier les codes initiaux par longueur du code puis par valeur. Les nouveaux codes s'obtiennent ainsi : le premier est rempli de zéro , le deuxieme commençant par 1 et rempli de zéro à droite mais bien faire attention à ce que la longueur ne soit pas dépassé, le troisième commençant par 11 et ainsi de suite jusqu'à avoir réalisé cela , jusqu'à ce que toutes les valeurs aient un code canonique.

Pour cela , la démarche que nous avons employé est celle la : on récupérait les entrées(des dictionnaires en l'occurence) des codes Huffman afin de les trier , puis on a trié la liste avec un **Comparator** que nous avons implémenté : on compare d'abord par la longueur des codes(longueur de la valeur dans le dictionnaire) ou sinon on trie par rapport à la valeur de la clé. Puis on fait une boucle qui parcoure toute la liste , on attribue un code canonique à chaque symbole qu'on ajoute dans un dictionnaire : **Map<Integer,String>**.

4.6 Pourquoi les codes canoniques au lieu des codes Huffman ? (Ayoub ANHDIRE)

Pour le décodage d'un fichier au format PIF , le fait de stocker l'arbre d'Huffman prendrait enormément de place et de temps mais on peut restituer ces codes Huffman grâce aux code canoniques. Il nous suffit juste de connaitre la longueur des codes et l'ordre des symboles pour pouvoir les reconstituer. On garde la même longueur que les codes initiaux, donc la compression reste optimale.

4.7 Le résumé de ces deux principes(Ayoub ANHDIRE)

L'algorithme de Huffman sert à coder les symboles avec des codes plus courts pour les symboles fréquents et plus longs pour les rares, ce qui permet de gagner de l'espace. Les codes sont optimaux, puisqu'aucun code n'est le début d'un autre, donc on peut les lire sans erreur. Les codes canoniques sont une version plus simple des codes Huffman : ils gardent la même longueur pour chaque symbole mais suivent d'autres contraintes qui les diffèrent des codes initiaux. Cela permet de stocker moins de données et de décoder plus vite, tout en gardant la même efficacité que Huffman.

4.8 Visualisateur au format PIF

5 MakeFile du Projet

6 Conclusion

Nous avons pensé que ce projet ... (à compléter)

6.1 Youness BOULALAM

6.2 Algassimou DIALLO

6.3 Ayoub ANHDIRE

Pour conclure, j'ai bien aimé ce projet en général, j'ai pris du plaisir à coder en JAVA d'autant plus que j'affectionne la programmation orientée objet. Ce projet m'a permis d'augmenter mes compétences techniques en JAVA, plus précisément dans la compréhension de structures abstraites notamment les dictionnaires ou encore les arbres. J'ai pu travailler d'autres notions comme la récursivité où j'avais du mal à comprendre la logique mais grâce à ce projet , j'ai pu m'améliorer et devélopper mes connaissances.

J'ai pu aussi développer mes qualités de communication avec mes camarades : chacun a joué un rôle ou il sait qu'il va perfectionner et la communication a été un enjeu majeur dans cette SAé car lorsque quelqu'un était bloqué, il faisait signe et ne restait pas tout seul sans avancer dans sa tâche. En conclusion, ce projet a été pour moi une expérience enrichissante d'où je tirerai certainement des profits.