Rapport de projet

SAÉ 3.2 FI - PIF

2025 - 2026

Aylane SEHL Séri-khane YOLOU Jenson VAL

UPEC
UNIVERSITÉ PARIS-EST CRÉTEIL
IUT
SENART
FONTAINEBLEAU

TABLE DE MATIÈRES

I. INTRODUCTION

1.Présentation du projet et contexte

Le projet PIF s'inscrit dans le cadre de la SAÉ 3.2 du BUT Informatique à l'IUT de Sénart-Fontainebleau.

L'objectif principal de cette SAÉ est de comprendre et de manipuler un format de fichier, le format .pif, à travers la réalisation de deux programmes distincts.

Le premier programme permet l'affichage d'une image au format .pif, le second a pour but de convertir une image classique vers ce même format.

Un fichier .pif (Program Information File) est un fichier utilisé historiquement par Microsoft Windows pour contenir des informations sur la manière d'exécuter un programme MS-DOS, comme les paramètres de lancement, la mémoire et le comportement de la fenêtre. Il sert de raccourci/configuration pour les applications DOS, mais n'est plus couramment utilisé aujourd'hui.

Bien que ce format ne soit plus utilisé dans les systèmes actuels, il reste quand même intéressant à étudier pour mieux comprendre comment les fichiers sont organisés et traités par un programme. Dans ce projet, le format ".pif" est utilisé dans un contexte différent de son utilisation basique, afin de représenter et convertir des images.

Ce travail permet d'aborder des notions essentielles telles que la lecture et l'écriture de fichiers, la manipulation de données binaires et la création de programmes clairs et structurés, en respectant les consignes du projet.

Le travail a été mené en équipe, à l'aide de la plateforme Gitea, afin d'assurer le suivi du code, la répartition des tâches et la gestion des versions du projet.

La suite de ce rapport présentera les fonctionnalités principales de l'application, sa structure technique, ainsi que les choix de conception réalisés tout au long du développement.

I. INTRODUCTION

2.Objectifs du projet

L'objectif principal de cette SAÉ est de comprendre et de manipuler un format de fichier, le format .pif, à travers la réalisation de deux programmes distincts.

L'objectif étant de concevoir deux programmes, le premier permettra l'affichage d'une image au format ".pif". Le second permettra de convertir une image basique au format ".pif".

Un fichier .pif (Program Information File) est un fichier utilisé historiquement par Microsoft Windows pour contenir des informations sur la manière d'exécuter un programme MS-DOS, comme les paramètres de lancement, la mémoire et le comportement de la fenêtre. Il sert de raccourci/configuration pour les applications DOS, mais n'est plus couramment utilisé aujourd'hui.

Ce projet nous permet de mieux comprendre la manière dont un format de fichier est conçu, stocké et interprété par un programme.

Ce projet vise à améliorer nos compétences en programmation, notamment en ce qui concerne la gestion des fichiers binaires, la lecture et l'écriture de données structurées, ainsi que la manipulation des pixels d'une image.

Un autre objectif important est d'apprendre à développer des programmes capables de gérer des erreurs telles que des fichiers invalides, des formats incorrects ou des problèmes de lecture.

I. INTRODUCTION

3.Technologies employées

Pour la réalisation de ce projet, nous avons utilisé le langage Java.

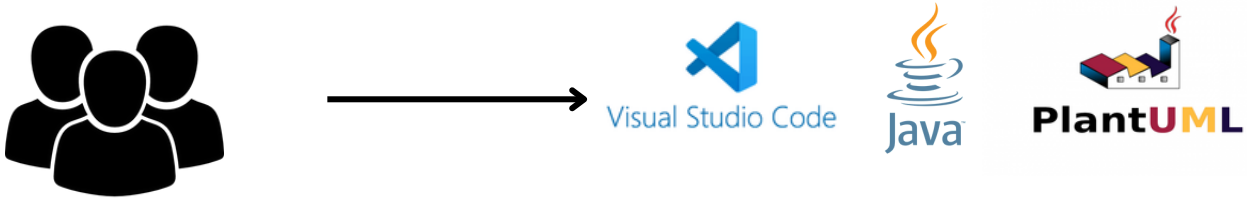
Le développement du projet a été effectué à l'aide de l'éditeur de code "Visual Studio Code", offrant un environnement de travail clair et efficace.

La gestion de version du projet a été assurée à l'aide de Grond, mise à disposition par l'IUT de Fontainebleau.

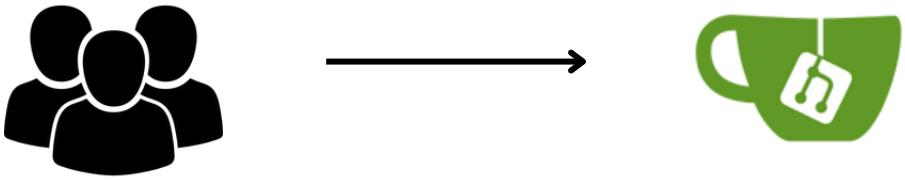
Cet outil nous facilier le travail en équipe, le suivi des modifications et la sauvegarde des différentes étapes du développement.

Cet outil nous a permis de répartir les tâches au sein du groupe et d'assurer une bonne organisation du travail.

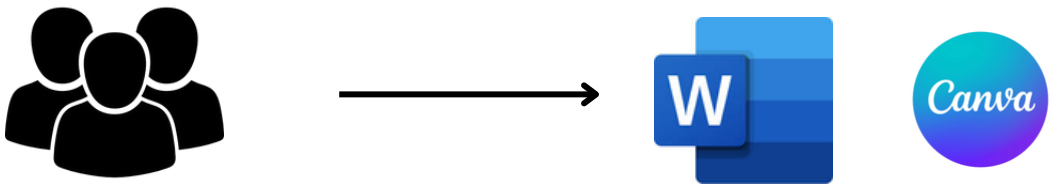
Enfin, pour la rédaction et la mise en page du rapport, nous avons utilisé Canva pour le design visuel et Word pour la mise en forme du contenu avant exportation en PDF. Ces outils nous ont permis de produire un document soigné, lisible et professionnel, en cohérence avec l'identité visuelle du projet Papillon.



Pour rédiger, compiler et exécuter le code source de l'application, tout en bénéficiant d'un environnement de développement clair et efficace.



Pour versionner les codes et collaborer en équipe efficacement via Git tout au long du développement



Pour rédiger, mettre en page et illustrer le rapport de projet, en assurant une présentation claire et professionnelle du travail réalisé.

II. PRÉSENTATION GÉNÉRALE ET FONCTIONNEMENT

1.1 Fonctionnalités principales

Implémentation des fonctionnalités

- **S**élection d'une image

Lorsque l'utilisateur lance le visualisateur, il peut indiquer l'image qu'il souhaite en indiquant son chemin puis le nom du fichier si rien est indiquer une page s'ouvre permettant à l'utilisateur de sélectionner un fichier dans son explorateur de fichier.
Pour le convertisseur le processus est le même sauf que l'utilisateur doit obligatoirement indiquer le chemin de son fichier ainsi que le chemin et le nom qu'il souhaite donner au nouveau fichier après conversion.

→ **S**élection d'une image via la ligne de commande

Convertisseur :

```
PS C:\Users\tayzi\Downloads\pif8> java -jar convertisseur.jar .\res\rotsnake.png
Image chargée : rotsnake.png
Dimensions : 800 x 800
Conversion terminée : res\rotsnake.pif
```

ou

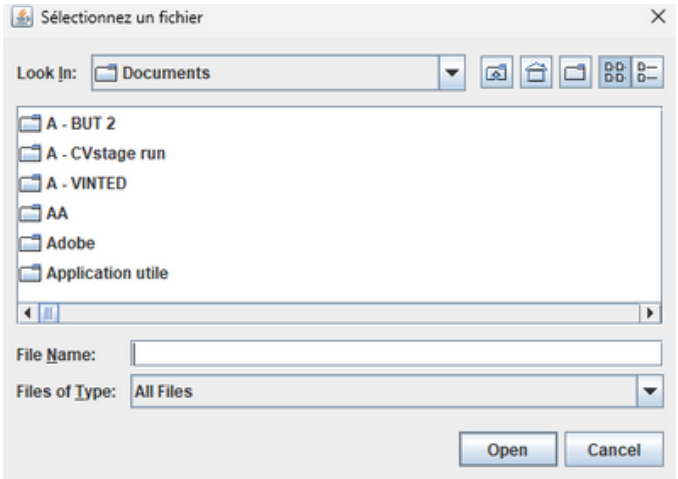
```
PS C:\Users\tayzi\Downloads\pif8> java -jar convertisseur.jar .\res\rotsnake.png .\res\rotsortie.pif
Image chargée : rotsnake.png
Dimensions : 800 x 800
Conversion terminée : .\res\rotsortie.pif
```

: Visualisateur

```
PS C:\Users\tayzi\Downloads\pif8> java -jar visualisateur.jar
Lecture du fichier : rotsnake.pif
Image decodée
Dimensions : 800 x 800
PS C:\Users\tayzi\Downloads\pif8> java -jar visualisateur.jar .\res\rotsnake.pif
Lecture du fichier : rotsnake.pif
Image decodée
Dimensions : 800 x 800
```

→ **S**élection d'une image via le sélecteur de fichier

```
PS C:\Users\tayzi\Downloads\pif8> java -jar convertisseur.jar
```

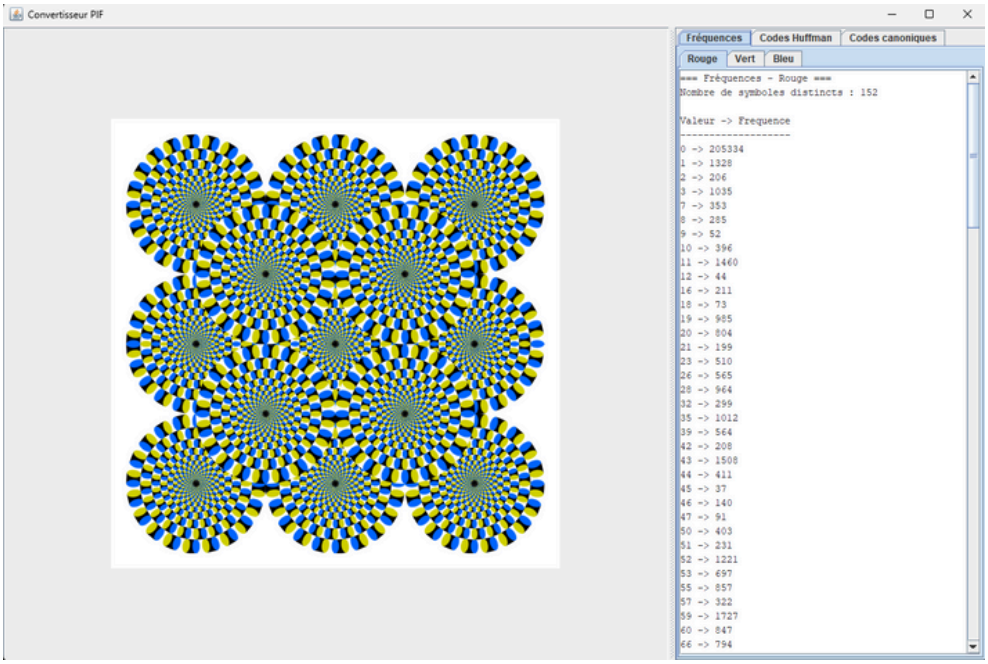


1.2 Fonctionnalités principales

Implémentation des fonctionnalités

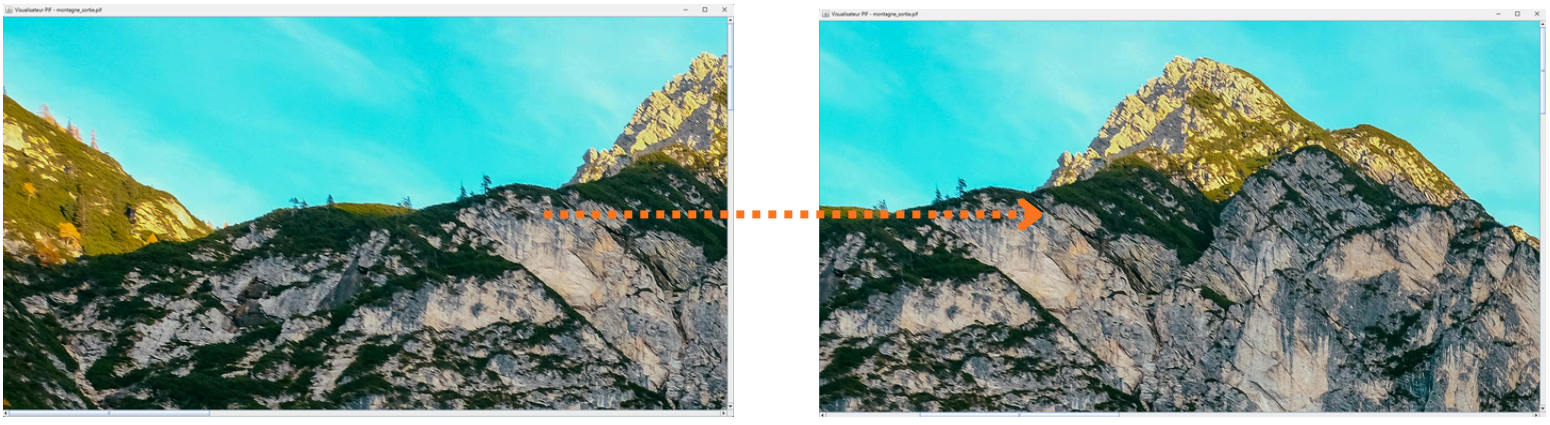
- **C**onversion de l'image

A l'aide d'algorithmes le second programme "Convertisseur" est capable de convertir une image vers un nouveau fichier au format ".pif".



- **D**éplacement de l'image

Lorsque l'utilisateur visualise ou convertit une image si elle trop grande pour la fenêtre qui l'affiche, l'utilisateur peut déplacer l'image grâce à son clique gauche



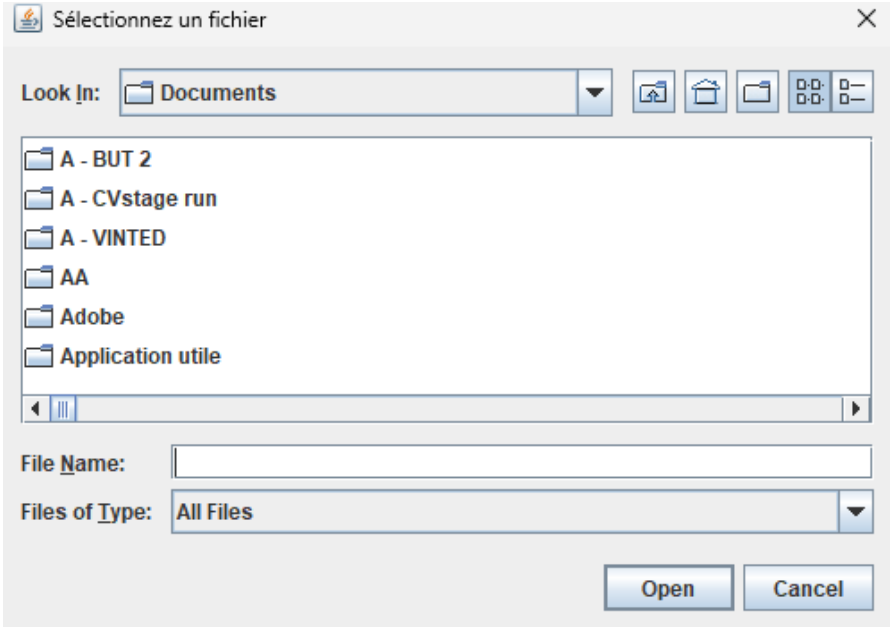
III.STRUCTURE DU PROGRAMME

1.1 Structure et logique du code

Le programme développé permet de convertir des images classiques (PNG, JPG, etc.) vers un format compressé propriétaire nommé PIF, basé sur l'algorithme de compression de Huffman appliqué aux composantes RGB. Il propose également un visualiseur capable de lire un fichier PIF, de le décoder et d'afficher l'image reconstruite.
Le projet est organisé autour de deux programmes principaux : un convertisseur et un visualiseur.

1. Convertisseur d'images vers le format PIF

Le convertisseur prend en entrée une image sélectionnée depuis la ligne de commande ou à l'aide d'une fenêtre graphique.



L'image est chargée en mémoire sous la forme d'un objet PIFImage, contenant ses dimensions et ses pixels.

Les principales étapes de la conversion sont les suivantes :

- extraction des composantes Rouge, Vert et Bleu de l'image ;
- construction d'une table de fréquences pour chaque composante ;
- génération d'un arbre de Huffman pour chaque composante ;
- création des codes Huffman, puis transformation en codes canoniques ;
- encodage des pixels en un flux binaire compressé ;
- écriture du fichier de sortie au format PIF, contenant les tables canoniques et les flux compressés.

III.STRUCTURE DU PROGRAMME

1.2 Structure et logique du code

1) Le projet contient deux exécutables :

- Convertisseur : lit une image (PNG/JPG…), sépare en composantes R/V/B, calcule Huffman, transforme en codes canoniques, affiche les tables, puis écrit un fichier .pif.
- Visualiseur : lit un fichier .pif, reconstruit les codes canoniques, décode les flux compressés R/V/B, reconstruit les pixels et affiche l'image.

2) Organisation en blocs (architecture) + MVC

Pour garder le projet lisible, on a structuré le code en plusieurs parties, en s'appuyant sur le modèle MVC (Modèle / Vue / Contrôleur) :

- Modèle (Model) : données + logique (image, Huffman, tables…)
- Vues (View) : fenêtres Swing (convertisseur / visualisateur)
- Contrôleurs (Controller) : gestion des interactions (ex : souris, affichage)
- Entrées / Sorties (I/O) : lecture/écriture du format .pif
- Outils (Utilities) : helpers (conversion RGB, format texte, image Swing, sortie automatique)

On a choisi MVC parce que ça sépare clairement les responsabilités :
la partie "calcul" ne dépend pas de Swing, et l'interface ne contient pas l'algorithme.

3) Rôle des classes (résumé)

Les deux programmes

- MainConvertisseur : charge une image → compresse (Huffman) → écrit un .pif → affiche la fenêtre avec l'image + tables.
- MainVisualiseur : lit un .pif → décode → reconstruit l'image → affiche l'image.

Données / traitement

- PIFImage : image en mémoire (largeur, hauteur, pixels).
- CodageRGB : découplé les pixels en rouge / vert / bleu.
- Huffman : calcule fréquences, construit l'arbre, fait les codes, encode en octets.
- Noeud / NoeudFeuille / NoeudInterne : l'arbre de Huffman.
- TableCodesCanoniques + EntreeCanonique : transforme les codes Huffman en codes canoniques (plus simple à stocker).
- NoeudDecodage + EntreeLongueur : côté lecture, sert à reconstruire et décoder.

Lecture / écriture du format PIF

- EcrivainPIF : écrit le fichier .pif (entête + tables + flux compressés).
- LecteurPIF : lit le .pif (tables + flux) et décode.

Interface (MVC)

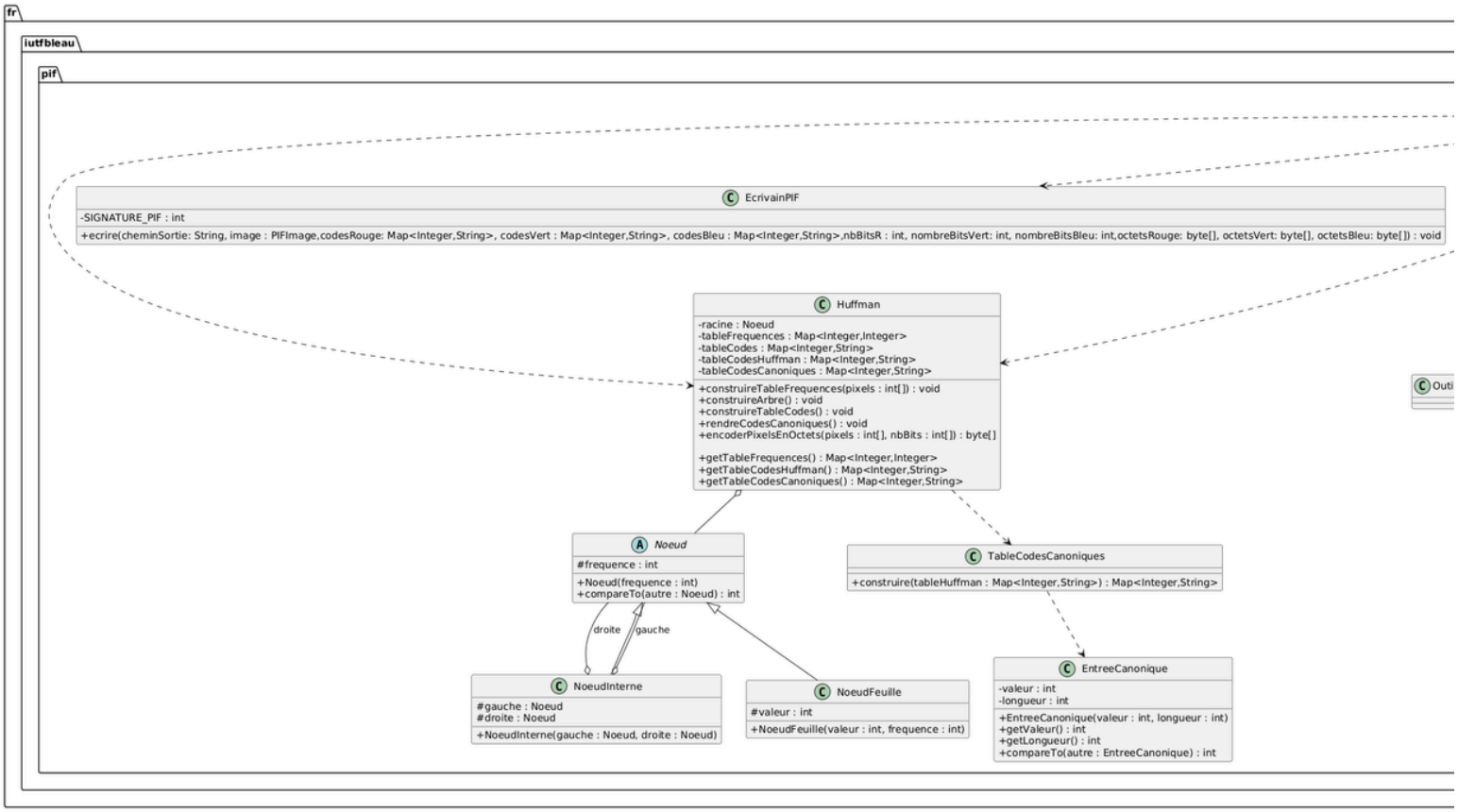
- FenetreConvertisseur (Vue) : affiche l'image + onglets (fréquences / codes Huffman / codes canoniques).
- FenetreVisualisateur (Vue) : affiche l'image décodée.
- ControleurVisualisateur (Contrôleur) : gère les actions utilisateur (si tu en as : clic, déplacement, etc.).

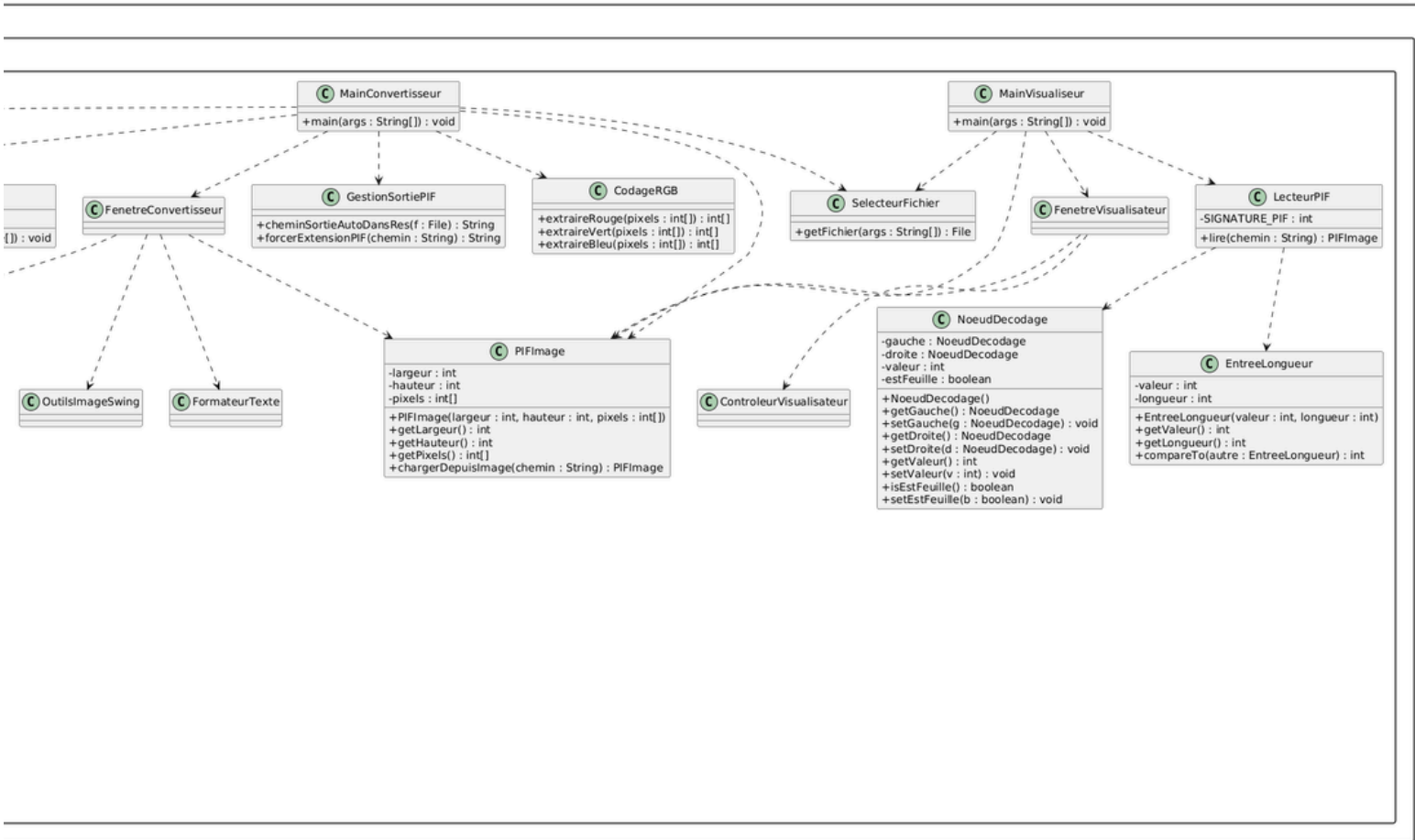
Outils

- SelecteurFichier : choisir un fichier (args ou fenêtre).
- GestionSortiePIF : crée automatiquement res/nom.pif (crée res/ si besoin).
- OutilsImageSwing : convertit PIFImage en image Swing + réduction si trop grande.
- FormateurTexte : met les tables en texte lisible pour l'affichage

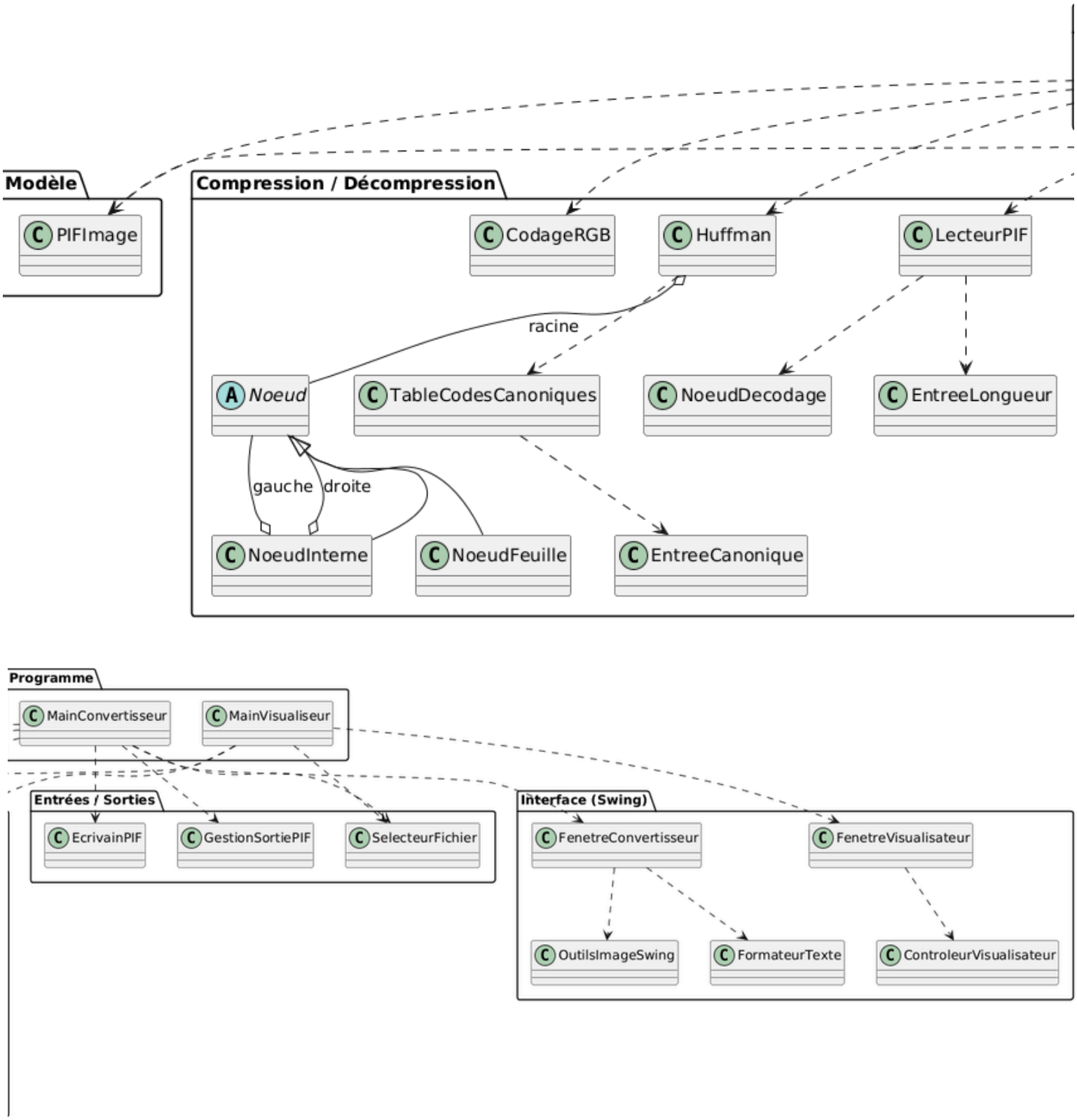
III.STRUCTURE DU PROGRAMME

2. Diagramme de classe simplifié





2. Diagramme de classe simplifié



IV. ARBRE ET TABLES

1.1 Exposition des classes de l'arbre binaire d'interface

1) Exposition des classes de l'arbre binaire (convertisseur)

Dans notre projet, l'algorithme de Huffman repose sur un arbre binaire.

Cet arbre existe sous deux formes selon le programme :

- dans le convertisseur : on construit l'arbre de Huffman (avec fréquences)
- dans le visualiseur : on reconstruit un arbre de décodage (avec codes canoniques)

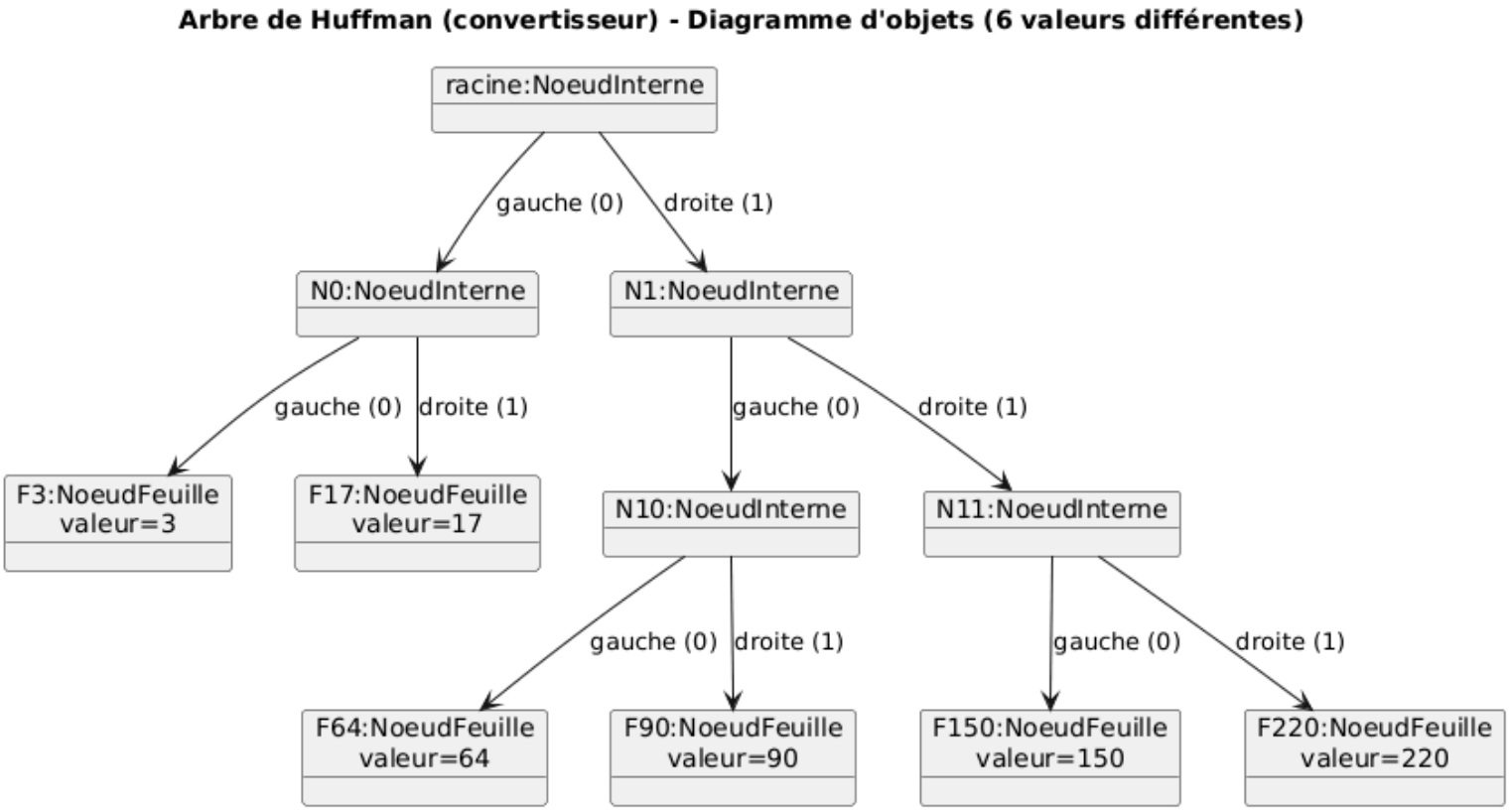
Exemple d'arbre binaire de décodage pour 6 valeurs
On suppose que le convertisseur a produit la table de codes suivante :

Valeur	Chemin dans l'arbre	Code binaire
3	racine → 0 → 0	**0**
17	racine → 0 → 1	**1**
64	racine → 1 → 0 → 0	**100**
90	racine → 1 → 0 → 1	**101**
150	racine → 1 → 1 → 0	**110**
220	racine → 1 → 1 → 1	**111**

1.2 Exposition des classes de l'arbre binaire d'interface

Arbre binaire correspondant (diagramme d'objets)

Voici l'arbre de l'encodage reconstruit dans le convertisseur à partir de cette table :



1.3 Exposition des classes de l'arbre binaire d'interface

2) Arbre de décodage (visualisateur)
Quand on lit le fichier .pif, on ne reconstruit pas l'arbre de Huffman original.
On reconstruit un arbre de décodage à partir de la table de codes canoniques.

Classe concernée

- NoeudDecodage
  - gauche (bit 0)
  - droite (bit 1)
  - estFeuille
  - valeur

Le visualiseur lit les bits du flux, descend dans l'arbre, et dès qu'on arrive sur une feuille → on récupère la valeur.

Si on a une table :

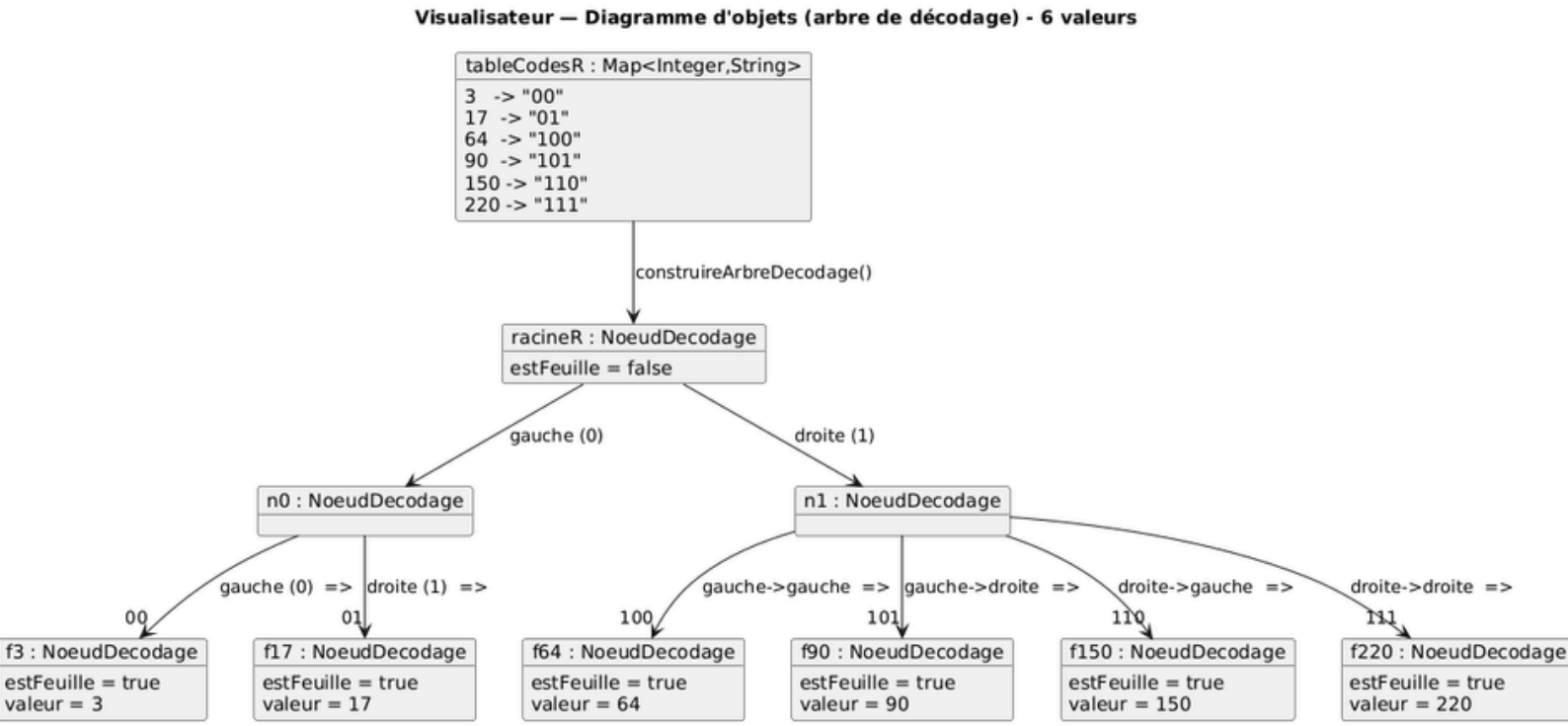
Valeur	Code canonique	Longueur
3	**0**	2
17	**1**	2
64	**100**	3
90	**101**	3
150	**110**	3
220	**111**	3

Les codes respectent la propriété de Huffman :
aucun code n'est préfixe d'un autre.

1.4 Exposition des classes de l'arbre binaire d'interface

Arbre binaire correspondant (diagramme d'objets)

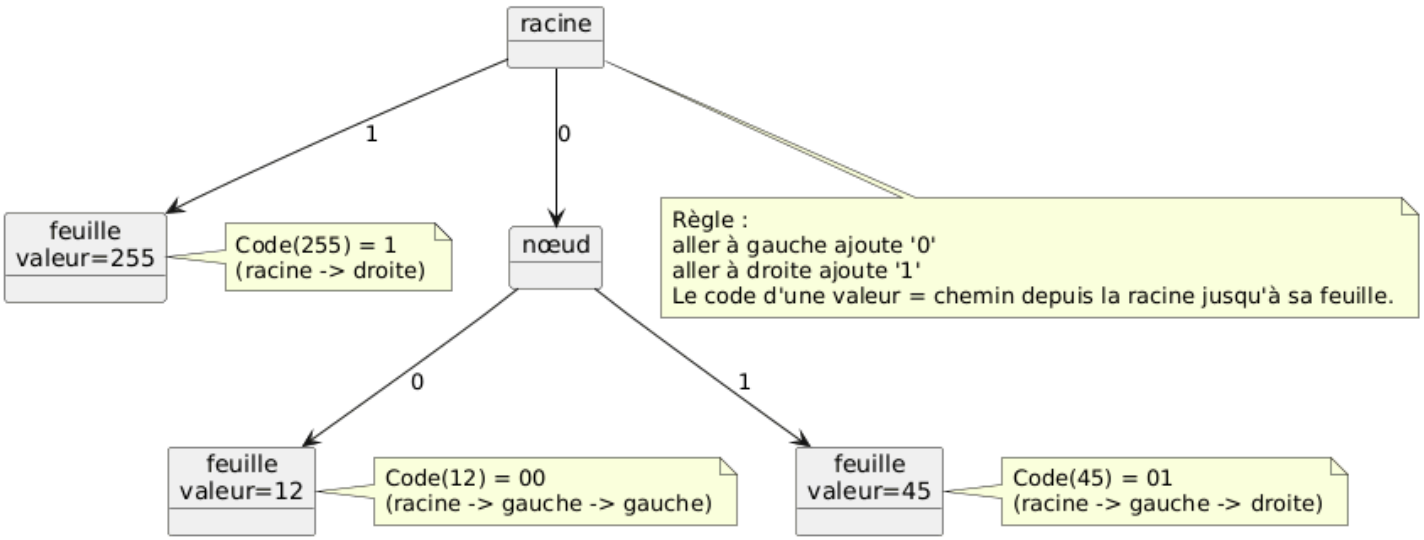
Voici l'arbre de décodage reconstruit dans le visualiseur à partir de cette table :



2.1 Explication table Convertisseur

La convertion repose sur une table de codes qui associe chaque valeur de composante RGB à un code binaire.
Cette table est représenter par un mappage, ou la clé correspond à une valeur de pixel entre 0 et 255 et la valeur qui lui est associé est le code binaire utilisé pour l'encodage.

Les codes sont premièrement générés à partir de l'arbre de Huffman. Chaque feuille de l'arbre correspond à une valeur de pixel et le chemain depuis la racine permet de construire son code binaire (0 pour un déplacement à gauche et 1 pour à droite).



Une fois cette première table réalisé, elle est transformé en table de codes canoniques. Dans cette forme, il n'y a que la lougueurs des codes qui est conservées les codes binaires sont reconstruits à partir de cela en suivant un ordre précis basé sur la longueur puis sur la valeur.
Cette façon de faire permet de stocker moins d'informations dans le fichier PIF et permet aussi la reconstruction lors du décodage.

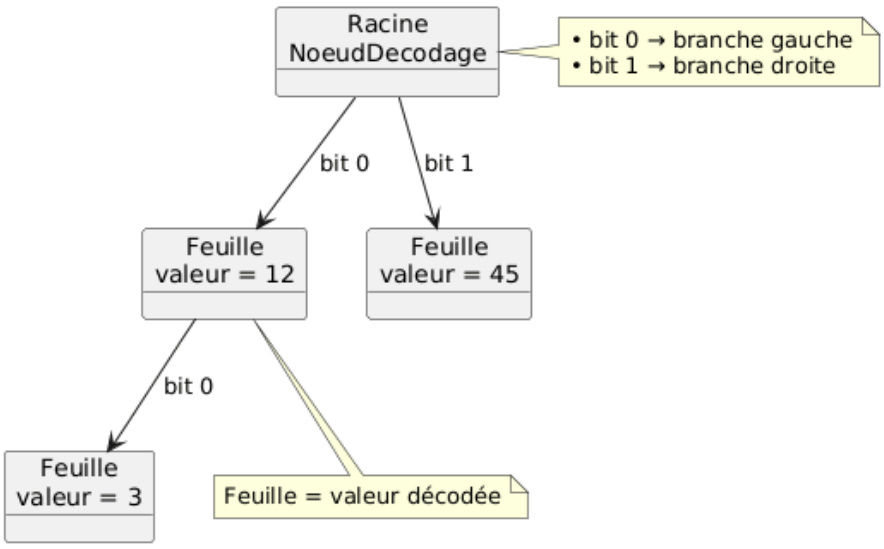
Pendant l'encodage, le programme parcourt le tableau de pixels de l'image. Pour chauqe pixel, sa valeur est utilisé comme clé pour accéder à son code binaire dans la table. Les bits de ce code sont ajouté à un flux binaire, ce flux est progressivement assembler en octets pour réaliser un tableau d'octets compressés. Le nombre total de bits réellement utilisés est également mémorisé dans le but d'ignorer les bits de remplissage pendant le décodage.
Ce mécanisme permet un encodage efficace car il évite de parcourir l'arbre Huffman pour chaque pixel.

2.2 Explication table Visualisateur

Le visualisateur réalise le mécanisme inverse, il retrouve les valeurs originales à partir d'un flux de bits.
Le fichier PIF ne stocke pas directement les codes binaires complets. Il contient seulement une table canonique sous la forme : valeur, longueur du code, pour chaque composante (Rouge,Vert,Bleu).
A partir de ces informations, le visualisateur peut reconstruire exactement les mêmes codes canoniques binaires que ceux utilisés pendant la compression (Convertisseur).

Une fois les codes reconstruits, ils sont utilisés pour bâtir un arbre de décodage Huffman.
Dans cet arbre :

- chaque feuille correspond à une valeur de pixel,
- chaque branche gauche représente le bit 0,
- chaque branche droite représente le bit 1.



Le mécanisme de décodage consiste alors à lire le flux binaire bit par bit.

En partant de la racine de l'arbre :

- un bit 0 fait descendre vers le fils gauche,
- un bit 1 fait descendre vers le fils droit.

Lorsque le parcours atteint une feuille, la valeur associée est récupérée et ajoutée au tableau des pixels décodés. Le parcours recommence ensuite depuis la racine pour décoder la valeur suivante.

V. DIFFICULTÉS RENCONTRÉES ET SOLUTIONS

1.Contraintes techniques

Problèmes majeurs rencontrés :

- Réalisation des algorithmes
- Gestion de l'image
- Vitesse d'exécution de la conversion

- **Réalisation des algorithmes**

Pendant de la conception des algorithmes nous avons rencontré plusieurs difficultés. Nous avons eu une certaine difficulté à comprendre la construction de l'arbre de Huffman. Il y a eu confusion entre les noeuds internes et feuilles, et entre fréquences et valeurs. Pour faire face à ce problème nous avons séparé de manière claire nos classes Noeud, NoeudInterne et NoeudFeuille. Cette organisation nous a permis de mieux distinguer les rôles de chaque type de nœud et de faciliter la mise en œuvre de l'algorithme.

- **Gestion de l'image**

Lors du développement de l'interface graphique, nous avons rencontré quelques problèmes. Premièrement, les grandes images ne s'afficher pas correctement, l'image dépassait la fenêtre sans contrôle. Pour résoudre ce problème nous avons fait appel à un JScrollPane, un redimensionnement dynamique.
De plus, nous avons eu beaucoup de mal avec le déplacement de l'image lorsqu'elle était trop grande, les événements souris étaient mal gérer. Par la suite, nous avons particulièrement fait attention a bien séparer contrôleur et vue.

- **Vitesse d'exécution de la conversion**

Dans certains cas, la conversion s'exécutait très lentement voir pas du tout, nous avions une mauvaise gestion des bits et des octets, l'utilisation de chaînes trop longues causaient des erreurs de mémoire. Pour résoudre cela, nous avons utilisé un encodage direct en bits regroupés dans des tableaux d'octets.

- **Contraintes fonctionnelles du visualisateur PIF**

Le visualisateur doit permettre d'afficher des images de grande taille en n'en montrant qu'une partie à l'écran. La navigation dans l'image se fait par déplacement à la souris, ce qui répond aux contraintes demandées.

Bien que des fonctionnalités comme le zoom aient pu être envisagées, elles n'étaient pas explicitement requises. Nous avons donc choisi de ne pas les implémenter afin de rester conformes au sujet et de conserver un visualisateur simple et fonctionnel.

VI. CONCLUSIONS

1.1 Conclusion personnelle de chaque membre

Jenson VAL

Dans un premier temps, cette SAÉ nous a permis d'approfondir considérablement nos connaissances sur le développement des algorithmes. Pour être honnête, c'est une facette du code qui me faisait assez peur, mais en prenant notre temps, en communiquant et en nous entraident avec mes camarades, j'ai pu mieux comprendre certains concepts nécessaires au bon déroulement de cette SAÉ.

Au début, nous n'avions pas bien compris le sujet de la SAÉ, certains points étaient à nos yeux très flous. Cependant, en prenant notre temps, en relisant plusieurs fois le sujet et en le découpant étape par étape, notre compréhension du sujet s'est grandement éclairci.
Sur l'aspect technique, j'avais aussi beaucoup de mal à comprendre l'arbre nécessaire à l'algorithme de Huffman ainsi que les codes canoniques, mais avec de la patience et de la communication, j'ai fini par beaucoup mieux comprendre le fonctionnement et la construction d'un arbre.

De plus, malgré les difficultés à découper notre code, nous avons pu confirmer notre compréhension de la séparation des classes. Le concept de « une classe, un rôle » nous a beaucoup aidés à avancer, malgré le fait que certains points étaient difficiles à comprendre au premier abord. L'expérience acquise lors de notre précédente SAÉ (3.1) nous a permis de comprendre à quel point ce concept était important.

Enfin, le travail en équipe sur cette SAÉ, a pour moi, été une réussite. Grâce à l'expérience accumulée lors de notre dernière SAÉ (3.1), nous avons réussi à mieux répartir nos tâches et notre temps de travail, malgré les difficultés rencontrées.

Cette SAÉ m'a permis d'appliquer un algorithme dans un contexte et un projet concret, ce qui m'a grandement aidé dans la compréhension de cette facette du code qui me parraissait pourtant flou au premier abord.

1.2 Conclusion personnelle de chaque membre

Séri-Khane YOLOU

Cette SAE a été compliquée dès le début, surtout pour comprendre l'algorithme de Huffman, la construction de l'arbre, l'encodage et le décodage des données. J'ai eu beaucoup de difficultés avec la gestion des bits et des octets, ainsi qu'avec le décodage, qui produisait parfois des images incorrectes (couleurs fausses, images jaunes). Ces erreurs m'ont fait comprendre à quel point la moindre imprécision dans un algorithme de compression peut avoir de grosses conséquences.

On a aussi rencontré des problèmes pour séparer les métiers des classes, notamment entre la compression, les entrées/sorties et l'interface graphique. Mettre en place une structure plus claire, proche du modèle MVC, m'a aidé à mieux fait comprendre le rôle de chaque partie du programme, mais sur le moment c'était compliqué et parfois décourageant, car chaque modification pouvait casser autre chose.

Un point important que nous avons observé est que, dans certains cas, le fichier PIF généré est plus volumineux que l'image PNG d'origine. Cela s'explique par le fait que le format PNG est déjà fortement compressé à l'aide d'algorithmes optimisés, tandis que notre compression repose uniquement sur l'algorithme de Huffman appliqué séparément aux composantes RGB. Cette observation montre que la compression Huffman n'est pas toujours efficace lorsqu'elle est appliquée à des formats déjà compressés.

Dans notre cas, je pense que l'objectif du projet était avant tout pédagogique : comprendre le fonctionnement interne d'un algorithme de compression, la construction des arbres de Huffman, la gestion des flux binaires et le mécanisme d'encodage et de décodage, plutôt que de rivaliser avec des formats industriels existants.

Malgré les difficultés, ce projet m'a permis de mieux comprendre le fonctionnement interne des formats de fichiers et des algorithmes de compression, et m'a donné une vision plus concrète de ce qui se passe lors d'une conversion de fichier.

1.3 Conclusion personnelle de chaque membre

Aylane SEHL

Ce projet a été une expérience enrichissante, mais aussi exigeante. Il m'a permis de mieux comprendre le fonctionnement interne d'un programme de compression, notamment la manière dont les données sont transformées, organisées et stockées dans un format de fichier personnalisé. Le fait de travailler à la fois sur la partie conversion et sur la partie visualisation m'a aidé à mieux voir le lien entre encodage et décodage.

Cette SAE m'a surtout fait prendre conscience de la complexité cachée derrière des formats de fichiers que l'on utilise tous les jours sans y penser. J'ai rencontré des difficultés sur des points très précis du programme, notamment la manipulation des bits, la gestion des tables de codes et le lien entre les différentes classes. Même si le résultat final n'est pas toujours optimal en termes de taille de fichier, le projet m'a permis de mieux comprendre comment fonctionne une compression et pourquoi certains choix techniques sont faits.

J'ai particulièrement apprécié l'aspect concret du projet, car il ne s'agissait pas seulement d'appliquer un algorithme théorique, mais de le faire fonctionner dans une application complète, avec des entrées réelles et une interface graphique. Cela m'a aussi appris à structurer un programme plus proprement et à séparer les responsabilités des classes.

Même si certaines parties ont demandé du temps et de la persévérance, ce projet m'a permis de gagner en autonomie et en rigueur. Il m'a aussi montré l'importance de tester régulièrement et de comprendre les outils utilisés, plutôt que de simplement les appliquer. Globalement, cette SAE a été formatrice et m'a aidé à progresser dans ma manière de concevoir et d'analyser un programme Java.

Projet réalisé par

- Aylane SEHL

- Séri-Khane YOLOU

- Jenson VAL

Enseignant : LUC HERNANDEZ