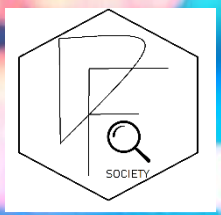
RAPPORT

Projet Fil Rouge phase 1

DuFonSociety

CAMINO Amaia
LANGLADE Alexandre
LOUAHADJ Aniss
SANTENE Jeremy
SUSGIN Jerome

1A SRI- Promotion 2019/2020
Groupe 6



Remerciements

Tout d'abord, nous souhaitons remercier et témoigner toute notre reconnaissance à l'ensemble de l'équipe pédagogique, pour son dévouement et son soutient dans la concrétisation de ce projet.
Nous tenons à remercier tout particulièrement nos clients Mme Ferrané et M Pinquier pour leurs conseils et leur disponibilité, M Vanderstraeten pour nous avoir épaulés tout au long du projet, ainsi que Mme Estadieu pour les ressources qu'elle nous a fournies afin d'améliorer la qualité de notre rapport.

Sommaire

Introduction

Notre projet fil rouge porte sur la réalisation d'un moteur de recherche permettant à un utilisateur d'effectuer une recherche parmi un corpus de textes, d'images, et de sons. Il pourrait par exemple rechercher des textes par mots-clefs, c'est à dire par nombre d'occurrences d'un mot dans un texte parmi tous ceux disponibles. Pour les images, une recherche par comparaison d'image ou par couleur dominante serait possible.

Ce projet a pour but de nous former sur plusieurs points en nous permettant de mettre en application les cours de programmation qui nous ont été dispensé lors de notre formation, et ce, à l'échelle d'un projet, nous offrant ainsi l'opportunité de nous améliorer. Il s'agit également d'un exercice de travail en groupe sur un projet informatique dans le but de nous préparer à nos futurs projets professionnels.

Dans ce rapport, nous allons commencer par vous présenter l'architecture générale de notre logiciel. Ensuite, nous détaillerons les structures qui ont été utilisées pour chacun de nos descripteurs. En suivant l'ordre de l'architecture générale, nous rentrerons plus en détail dans les fonctions conçues et utilisées pour chaque partie de notre projet. Enfin, chacun d'entre nous décrira quelles ont été les difficultés rencontrées, aussi bien celles liées à la programmation que celles liées au travail de groupe. Enfin, nous vous ferons part de nos retours personnels sur ce projet, et les enseignements que nous avons pu en tirer.

I. Conception

A. Décomposition en modules

Le logiciel est résultant de l'assemblage des modules présentés ci-dessous :

- Descripteurs Texte : ce module gère les descripteurs de type texte de la définition jusqu'à la manipulation.
- Descripteurs Image : ce module gère les descripteurs de type image de la définition jusqu'à la manipulation.
- Descripteurs Audio : ce module gère les descripteurs de type son de la définition jusqu'à la manipulation.
- Pile : ce module gère la définition et la manipulation de piles de type texte, image ou audio.
- Configuration : ce module gère l'accès, la sauvegarde et la modification des préférences de l'utilisateur qui serviront de paramètres pour l'indexation ou pour la recherche.
- Entrées : ce module gère les entrées de l'utilisateur pour les contrôler en termes de conformité et de pertinence.
- Indexation Texte : ce module gère le traitement des fichiers de type texte lors de leur indexation.
- Indexation Image : ce module gère le traitement des fichiers de type image lors de leur indexation.
- Indexation Audio : ce module gère le traitement des fichiers de type son lors de leur indexation.
- Recherche Texte : ce module gère la recherche parmi les éléments de type texte.
- Recherche Image : ce module gère la recherche parmi les éléments de type image.
- Recherche Audio : ce module gère la recherche parmi les éléments de type son.
- Menu : ce module défini l'intégralité des menus et sous menus du logiciel ainsi que la façon de naviguer des uns aux autres.
- Main : ce module gère l'initialisation, l'exécution et la fermeture du logiciel

B. Structures de données

1. Architecture générale

Le logiciel est découpé en deux fonctionnalités : une permet d'indexer un corpus, l'autre d'effectuer une recherche dans ce corpus. Les liens entre ces deux entités sont les piles. En effet, le fondement de l'indexation est de créer un descripteur associé à un document puis de l'empiler dans la pile de son type. Ainsi, à tout moment de l'exécution, le logiciel peut parcourir ces différentes piles pour examiner les descripteurs. L'utilisateur, en parcourant les menus où sont rassemblés les outils de recherche proposés et en sélectionnant une fonctionnalité, va lancer une recherche, qui, par l'étude de la pile appropriée pourra trouver quels descripteurs répondent pertinemment à la recherche.

Dans la fonction main, on retrouvera les appels des différents menus ainsi que la gestion des messages d'erreur.

2. Structures utilisées en traitement de texte

Structures utilisées lors de la phase d'indexation :

Pour le traitement des données textuelles, nous avons choisi de créer une structure de type descripteur_Texte_s ainsi qu'une structure de type Terme.

La première est créée afin de répondre au cahier des charges puisque nous créons une pile de descripteurs. Celle-ci contiendra l'identifiant propre à chaque descripteur crée, le nombre de mots du texte dont il est censé être la représentation synthétique, un pointeur vers la structure descripteur_Texte_s suivante, le nombre de termes retenus ainsi qu'une liste chainée de termes.

La seconde structure a été choisie afin de permettre de créer dynamiquement une liste chainée des termes retenus lors de l'indexation. Chaque structure terme créée est composé d'une chaine de caractères représentant le mot sélectionné, que le nombre d'occurrence de celui-ci dans le texte analysé ainsi qu'un pointeur vers la structure Terme suivante.

Structures utilisées lors de la phase de recherche :

Pour la recherche par mot clé, nous avons créé une structure **Element_s** définissant le type Element, ainsi qu'un pointeur vers une structure Element_s définissant le type **Liste**.

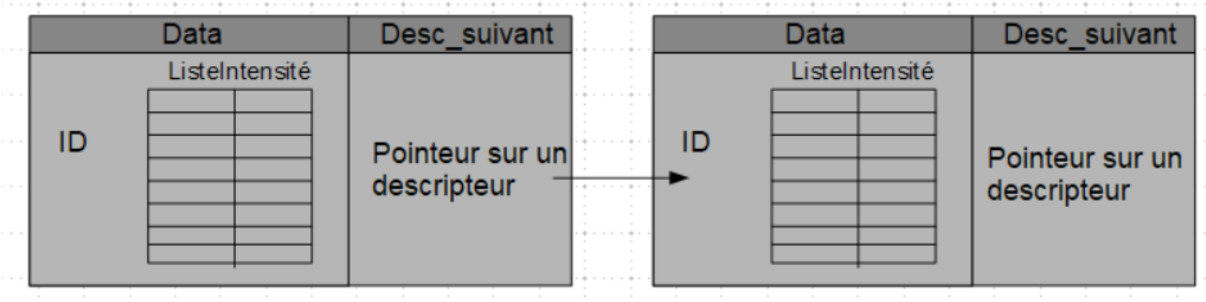
La structure créée servira à retenir les descripteurs où le mot recherché est présent ainsi que le nombre d'occurrence de celui-ci. Ce nombre nous permettra de classer les descripteurs par ordre décroissant du nombre d'occurrence du mot.

3. Structures utilisées en traitement de l'image

Chaque image de notre corpus était accompagnée d'un fichier texte contenant les informations suivantes :

- Nombre de ligne de la matrice
- Nombre de colonnes de la matrice
- le nombre de matrice (3 pour les images en couleurs)
- La/les matrices d'entiers représentant l'intensité de chaque pixel de l'image

Grâce à ces fichiers textes, nous avons créé une structure appelée descripteur. La forme de ce descripteur était la suivante :



Il y avait donc un entier servant d'identifiant (ID) à chacun de nos de nos descripteurs, et un tableau d'une autre structure appelée « couleurs ». Quelques explications sur ce tableau : Au début du projet, j'ai essayé de me représenter quelle forme pourrait avoir mon descripteur. Et par manque d'anticipation et à cause de quelques lacunes en codage au début, j'ai donc créé un tableau d'une structure contenant deux entiers, un pour l'intensité, et un pour le nombre d'occurrence de cette intensité dans l'image. C'est donc ma structure « couleurs ». J'ai donc fixé la taille de ce tableau à 512, car je pensais me limiter à une quantification sur 3 bits et donc avoir un panel de 512 intensités.

Le but de créer cette structure unique était qu'elle soit utilisable pour les images couleurs comme les images en noirs et blanc.

Enfin, Ces descripteurs ont vocation à être intégrés à une pile. Il à donc fallu y ajouter un pointeur vers une autre structure descripteur, la suivante dans la pile. En plus de cette structure descripteurs, j'ai finalement créé une autre structure comme étant un pointeur vers la première structure descripteur.

C. Méthode de manipulation

1. Manipulation des données en traitement de texte

Manipulation des données lors de l'indexation :

L'indexation d'un fichier texte (.xml) fait appel à la fonction **index_un_texte(char *chemin, Pile pile)** prenant en paramètre une chaine de caractères représentant le chemin du fichier à traiter ainsi qu'une pile servant à empiler l'ensemble des descripteurs textes.

En premier lieu, nous faisons appel à la fonction **deleteBalises(char *chemin)** prenant en paramètres le chemin du fichier.

Cette fonction fera appel à plusieurs commandes système linux successives afin de supprimer la totalité des balises, l'ensemble des caractères spéciaux que nous avons choisis d'enlever (. - , : ? ; % ' « » ( )) ainsi que la totalité des chiffres. Nous avons également choisi de retirer les lettres 's' présentes en fin de mot afin de pouvoir gérer les mots singulier et pluriels de la même manière. Cependant, cette méthode ne s'applique pas aux mots dont la forme plurielle est particulière (exemple : animal/animaux).

De plus, nous faisons également appel à une commande linux afin de transformer les lettres majuscules présentes dans le fichier en lettres minuscule. Cette transformation nous sera utile lors de la manipulation des mots dans la fonction **tri()**.

Enfin, la fonction **deleteBalises(char *chemin)** fait appel à la fonction **tri()**.

La fonction **tri()** est utilisée afin de lire les mots du textes un à un et de déterminer si ceux-ci sont des mots considérés « outils », c'est-à-dire des mots sans aucune utilité à la description du texte, et cela grâce à la fonction **isMotOutil(char *mot)** comparant le mot lu et passé en paramètre avec les mots outils présents dans une **stoplist** de 689 mots.

Si le mot n'est pas un mot outil celui-ci est écrit dans un fichier nommé « retranscriptionTri.txt ». Sinon, le mot lu est simplement ignoré.

Par la suite, après avoir effectué cette étape de « tri » du fichier d'origine, nous créons une variable de type **Descripteur_texte** dont nous remplirons l'identifiant en fonction du nombre de descripteur empilés, ainsi que le nombre de mot qui constituent le fichier texte.

Nous lisons ensuite le fichier « retranscriptionTri.txt » mot à mot. Pour chaque mot lu, s'il n'a pas déjà été indexé, nous créons une variable de type Terme. D'une part, le champ de cette variable correspondant au mot est rempli par le mot lu. D'autre part, le champ représentant le nombre d'occurrences est rempli à l'aide de la fonction **nbOccurence(char *mot)** permettant de connaître le nombre de fois que le mot passé en paramètre est présent dans le fichier « retranscriptionTri.txt » à l'aide d'une commande linux.

Ce terme est ensuite ajouté à la liste chainée des termes présents dans le descripteur.

Enfin, en fonction du seuil et de la valeur limite choisis par l'administrateur, le nombre de termes sera modifié afin de garder uniquement un nombre de termes inférieur ou égal à la valeur limite ainsi que les termes dont le nombre d'occurrences répond au seuil.

Le descripteur ainsi crée est ensuite empilé dans la pile passée en paramètre, et le fichier « links_txt » faisant le lien entre chaque descripteur et le texte dont il est la représentation synthétique est mis à jour.

Manipulation des données lors de la recherche :

La recherche d'un mot clé fait appel à la fonction **recherche_mot_cle(Pile pile, char *mot)**. Cette fonction prend en paramètres une pile de descripteurs textes ainsi que le mot requête et renverra un tableau contenant au maximum dix chemins où le mot recherché est présent. Ces chemins seront classés par ordre décroissant du nombre d'occurrence du mot dans chacun d'eux.

Pour chaque descripteur présent dans la pile envoyée en paramètre, nous faisons tout d'abord appel à la fonction **isMotExistant(Descripteur_Texte descripteur_texte, char *mot)**. Celle-ci

permet de savoir si le mot passé en paramètres est présent dans la liste des termes du descripteur texte lui aussi passé en paramètres.

Si le descripteur possède le mot, nous créons une variable de type **Element** que nous ajouterons à la liste des éléments sélectionnés par ordre décroissant du nombre d'occurrence du mot**.** Le champ « id » correspondra à l'identifiant du descripteur. Quant au champ « nbOccu », celui-ci contiendra le nombre d'occurrence du mot dans le fichier représenté par le descripteur.

Enfin, nous remplirons un tableau avec les chemins des textes où le mot est présent et ceci grâce aux éléments sélectionnés. En effet, en faisant appel à la fonction **getChemin(int id)** qui, grâce à l'identifiant passé en paramètre, retrouvera le chemin vers le fichier texte lui étant associé.

Le nombre de chemins sélectionnés ne pourra pas excéder dix.

Le tableau sera envoyé en sortie de la fonction afin de pouvoir afficher par la suite les chemins vers les fichiers textes où le mot requête est présent.

2. Manipulation des données en traitement de l'image

Autour de la structure descripteur image, nous avons créé deux fonctions :

1. L'écriture de descripteur : Cette fonction remplissait un document texte avec la structure du descripteur entré en paramètre. Elle est utile lors de la fermeture du logiciel pour enregistrer nos descripteurs.
2. La lecture de descripteur : Cette fonction lit un fichier texte au format descripteur et remplit une structure. Elle est utile au démarrage pour éviter d'avoir à relancer l'indexation si les descripteurs avaient déjà été créé.

Autour de la quantification :

Dans nos images, certaines étaient en couleurs, et avaient donc trois matrices au lieu d'une dans leur fichier txt. Il a donc fallu quantifier ces matrices pour n'en obtenir plus qu'une.

Pour commencer, j'ai dû créer une nouvelle structure qui est la structure RGBquantifie. Elle se compose d'un tableau d'entiers comprenant le nombre de lignes et de colonnes de notre matrice, et d'une matrice dynamique 2 dimensions d'entiers. C'est cette matrice que nous remplissons dans notre fonction de quantification et c'est cette structure qui nous permet de créer le descripteur.

Fonctions annexes créées pour la quantification :

Fonction	Entrée(s)	Sortie
IntToBinaire	Int	Tableau d'entier de 1 et de 0
BinaireToInt	Tableau de int	Int
QuantifierBinaire	Trois tableaux d'entiers (nos trois valeurs binaires à quantifier)	Un tableau d'entier (notre valeur quantifiée)
quantifer_3_matrices	Trois tableaux dynamiques d'entiers	Un tableau dynamique d'entier

Cette dernière fonction annexe est la plus importante, car c'est elle qui transforme nos trois matrices d'entier en une matrice de ces entiers quantifiés. Pour cela, elle fait appel aux trois autres fonctions annexes.

Rappel : Là où le nombre de bit à quantifiés était important d'être utilisé dans nos fonctions, une fonctions getConfig_image_NbBits() allait chercher cette valeur dans le fichier de configuration.

La fonction quantification, déroulement :

. Ouverture du fichier txt associé à l'image
. Création de trois matrices dynamiques d'entiers
. Remplissage de ces matrices avec les valeurs lûes dans le fichier txt
. Application de la fonction quantifier_3_matrices sur ces trois matrices dynamiques
. Remplissage de la structure RGBquantifie avec cette nouvelle matrice dynamique

En sortie, elle renvoie une Structure RGBquantifie qui nous sert pour l'indexation.

Autour de l'indexation :

Pour réaliser l'indexation d'un ensemble d'image dans la main de notre logiciel, il faut une fonction qui indexe une image. Cette dernière doit donc recevoir en entrée une pile, un chemin vers une image, et un entier indiquant s'il s'agit d'une image couleur ou noir et blanc. Son but est de créer le descripteur et de l'empiler dans la pile entrée en paramètre.

Fonctions annexes créées pour l'indexation :

Fonction	Entrées/sorties	But
InitDescripteur()	Sortie : un descripteur initialisé	Initialiser le descripteur
Remplissage_descripteur_imageNB	Entrées : chemin vers une image, pile (pour l'ID), et le descripteur à remplir	Remplir le tableau du descripteur et son ID en allant lire l'ID du descripteur en haut de la pile
Remplissage_descripteur_imageRGB	Entrées : RGBquantifie, descripteur, pile (pour l'ID)	Remplir le tableau d'un descripteur à l'aide de la matrice dynamique du RGBquantifie.
listing_descripteur	Entrées : descripteur, chemin vers l'image à lister, chemin vers le fichier de listing	Permet de lister sur un fichier texte quels descripteurs sont créés, et à quelles images ils correspondent

La fonction indexation, déroulement :

Pour les images Noir et blanc (identifiées grâce à l'entier en paramètre) :

. Initialisation du descripteur
. Remplissage du descripteur
. Empilage dans la pile des descripteurs Noirs et Blancs
. Ajout du descripteur au listing des descripteurs noirs et blancs

Pour les images couleurs (identifiées grâce à l'entier en paramètre) :

. Initialisation du descripteur
. Création d'une structure RGBquantifie par quantification de l'image
. Remplissage du descripteur (à l'aide de la structure RGBquantifie)
. Empilage dans la pile des descripteurs couleurs
. Ajout du descripteur au listing des descripteurs couleurs.

3. Manipulation des données en traitement audio

Nous avons divisé le travail en deux parties : l'indexation des fichiers audio dans un premier temps, puis la recherche et la comparaison des fichiers audio.
Ces deux parties ont été traitées de la manière suivante :

1. Conception d'un algorithme d'étapes menant de l'entrée du système informatique (le fichier audio d'entrée) à la sortie (le fichier descripteur).
2. Programmation et tests.

L'indexation audio

La fonction d'indexation audio avait pour objectif de transformer une suite de données contenues dans un fichier en un fichier descripteur contenant les données sous la forme d'un histogramme de taille k*m (ces coefficients sont calculés ou lus à partir du fichier « .config »).
L'algorithme d'indexation audio suit le modèle suivant :

1. Lecture du fichier audio.
2. Constitution de la matrice d'échantillonnage.
3. Écriture du fichier descripteur.

Étape 1.1 : Lecture du fichier audio

Lorsque nous nous sommes découpés le travail, la gestion des entrées-sorties ne m'étaient pas destinées. C'est la raison pour laquelle je n'ai pas traité cette partie. La fonction indexation est donc appelée avec des paramètres.

Lorsque nous entrons dans la fonction indexationAudio, nous avons donc les paramètres suivants connus :

1. unsigned int n, contenue dans le « .config ».
2. unsigned int m, contenue dans le « .config ».
3. int id, représentant le numéro d'identification du fichier audio.
4. char *nomFichierEntree, *nomFichierSortie, représentant respectivement le nom du fichier audio et du fichier descripteur.

La première étape de l'indexation audio consiste donc à lire le fichier audio afin d'en extraire les variables locales suivantes :

1. float *valFichier, stockant de manière dynamique et successive les valeurs contenues dans le fichier audio.
2. unsigned int nbrValeurs, comptant le nombre de valeurs contenues dans le fichier audio.

L'obtention de cette seconde valeur permet d'en déduire la variable suivante, nécessaire à la constitution de la matrice de stockage de données, et donc du fichier descripteur :

1. unsigned int k, calculée à partir de « n » et de « nbrValeurs » par le biais de la fonction « getK ».

La fonction est codée de manière à prendre absolument toutes les valeurs contenues dans le fichier,

quitte à pouvoir stocker plus de valeurs que nécessaires plutôt que pas assez.

Enfin, de manière parallèle sont déterminées les « M = m++ » valeurs d'échantillonnage qui correspondent aux intervalles de l'histogramme de la matrice de stockage.
float *intervallesM est un tableau de float dynamique contenant les valeurs séparant deux niveaux d'histogramme.

Étape 1.2 : échantillonnage de la matrice de stockage

Au cours de cette étape, nous construisons la matrice de stockage contenant les valeurs de l'histogramme. Cette matrice de stockage sera ensuite inscrite dans le fichier descripteur.

Tout d'abord, nous effectuons un pré-traitement : un changement de base sur le tableau de valeurs obtenues par la lecture du fichier audio afin d'obtenir un ensemble de valeurs contenues dans [-1;1].

1. float *valFichier ne contient maintenant plus que des valeurs comprises entre [-1;1].

Nous créons ensuite la matrice de stockage qui stockera l'histogramme. Une fenêtre de taille « n » sera lue dans *valFichier, et stockée temporairement dans *valTemp. Enfin nous construisons l'histogramme k-ième à partir de la k-ième lecture de fenêtre, que nous inscrivons dans la matrice de taille k*M.

1. matrice matEchantillon est un tableau dynamique à deux dimensions de taille « k » et « M ». Initialisée d'abord à 0, elle est ensuite complétée par les résultats obtenus par la transformation en histogramme de la lecture des valeurs du fichier audio.
2. float *valTemp contient les données lues par la fenêtre de taille « n ». Cette variable est une variable temporaire.

Étape 1.3 : écritude du descripteur

La troisième étape de l'indexation consiste à écrire le fichier descripteur. Le chemin est normalement écrit dans la variable nomFichierSortie. Le descripteur étant une structure avant d'être un fichier, l'étape à construire et initialiser une structure descripteur temporaire et de l'écrire dans un fichier.

Enfin, nous libérons les allocations mémoire dynamique et renvoyons le fichier dans les entrées-sorties.

Ce qui n'a pas été implémanté

Comme énoncé précédemment, l'indexation audio n'a pas pu être implémentée dans le programme principal, malgré son bon fonctionnement. Dans le code source, nous avons utilisé des valeurs directement et arbitrairement créées en entête de la fonction. Les fichiers entrées-sorties ne sont pas non-plus implémentés ; les chemins et noms des fichiers audio et des fichiers descripteurs doivent être directement modifiés dans le code source.

Une fonction de retour d'erreur n'a pas pu être implémentée dans le système. La plupart des fonctions d'indexation renvoie un entier qui code une erreur spécifique (allocation mémoire échouée, ouverture de fichier échouée, …), qui est stockée dans la variable entière « resultat ». L'idée de construire un tableau dynamique stockant les erreurs en fonction de l'étape d'éxécution du programme a été abandonnée par manque de temps.

La recherche audio

La recherche audio a pour objectif de prendre en entrée un chemin de fichier descripteur et un tableau de fichiers descripteurs représentant l'accès à l'intégralité des fichiers descripteurs stockés dans la base de données.

La recherche audio suit le modèle suivant :

1. Lecture du fichier de base de comparaison
2. Lecture des fichiers du corpus, comparaison et renvoi des résultats

Étape 2.1 : Lecture du fichier de base de comparaison

Le fichier de base de comparaison est le fichier descripteur correspondant au fichier recherché dans le corpus. Ses valeurs sont stockées afin de permettre de comparer plus rapidement avec les valeurs des descripteurs des fichiers du corpus.

Les données du fichier descripteur de comparaison sont obtenues après lecture du fichier descripteur correspondant et stockée dans une variable structure descripteur desBase. Au cours de la boucle de lecture des fichiers descripteurs du corpus, une seconde variable structure descripteur desTemp est créée afin de pouvoir comparer les fichiers.

La comparaison de fichier prend en compte le nombre de valeurs des descripteurs (déterminée par la valeur de « k » des descripteurs correspondants). Si les valeurs sont égales, la comparaison est bien plus rapide car elle effectue des instructions simplifiées. Dans le cas contraire, la comparaison est toujours effectuée du petit fichier dans le grand fichier, afin de déterminer si ce dernier contient le fichier de petite taille et à quel endroit.

Au cours de la comparaison, une fenêtre de la taille du petit fichier parcourt le fichier de plus grande taille de la manière suivante :

1. On parcourt l'intégralité du fichier de petite taille à partir de la position (0,0) du grand fichier. Une fois effectuée, on réitère l'action en partant de la case suivante ((0,1), puis (0,2), … (1,0), (1,1), … (max,max)) jusqu'à la position max correspondant à la ligne maximale du grand fichier moins la taille du petit fichier, de manière à lire l'intégralité des possibilités du grand fichier.

Ainsi, si les deux fichiers possèdent la même taille, il n'y a qu'un seul parcours.

2. A chaque passage est calculée une valeur de divergence obtenue à partir de la formule de « DKL = Distance de Kullback-Leibler ». Les valeurs extrêmes (la plus faible et la plus forte) de l'intégralité des parcours sont conservées dans une structure appelée curseur. Est conservée aussi la position de départ de la valeur de divergence la plus faible.

Nous précisons que la divergence obtenue est aussi divisée par le nombre total de valeurs lues par passage afin d'assurer une divergence moyenne et non une divergence totale.

3. Les fichiers descripteurs possédant des taux de divergence faible sont stockées dans une liste (tableau statique de structure listeFichier) qui stocke leur chemin, leur numéro d'identification et leur structure curseur contenant les informations de lecture et de similarité. La liste contient jusqu'à 10 fichiers descripteurs classés selon leur valeur de similarité (obtenue en inversant leur valeur de divergence). Si la liste contient des fichiers null (signifiant qu'il y a donc moins de fichiers que de place dans le tableau), ces derniers ne sont pas affichés par le programme.

Lorsqu'un résultat est stocké, on calcule sa précision (division du faible DKL par le fort DKL stockées dans la structure curseur du fichier correspondant) et l'instant de démarrage (calculé à partir de la position initiale stockée, en pourcentage sur la valeur totale du fichier comparé).

Ce qui n'a pas été implémenté

Comme pour l'indexation, les entrées-sorties et l'implémentation dans le programme ne fonctionnent pas. Par conséquent, l'intégralité du programme fonctionne en interne, et nécessite une modification du chemin des fichiers descripteurs directement dans le code source. Le résultat n'est pas renvoyé mais seulement affiché sur la sortie standard. Par conséquent, le fichier similaire ne peut pas être joué directement depuis le résultat.

Une erreur « stack smashing detected » est indiquée en fin de programme, mais nous n'avons pas réussi à la corriger tout en faisant fonctionner le programme correctement.

Lancement du programme

Le programme se lance en deux étapes : le lancement du programme d'indexation, puis le lancement du programme de recherche. Puisque le programme n'a pas pu être implémenté dans le programme principal, il doit être lancé manuellement.

Lancement du programme d'indexation

1. Assurez-vous d'avoir placé les fichiers dans le même dossier que le code source. Assurez-vous aussi d'avoir bien indiqué le nom du fichier audio correspondant (ici, fichierEntreeCourt et fichierSortieCourt, fichierEntreeLong et fichierSortieLong corresponde nt respectivement aux chemins des fichiers audio court « Jingle_fi.txt » et « Corpus_fi.txt » et des descripteurs ici nommés arbitrairement « descripteur_court.txt » et « descripteur_long.txt ».
2. Modifiez les valeurs de « m », de « n » et de « id » pour simuler les paramètres d'entrée de la fonction. Ces valeurs correspondent respectivement au nombre d'intervalles d'échantillonnage, à la taille de la fenêtre d'échantillonnage et du numéro d'identifiant du fichier, avec des valeurs similaires à celles inscrites dans le cahier des charges.

Dans notre programme, nous créons les descripteurs des deux fichiers en même temps, ainsi les deux fichiers possèdent le même numéro d'identification. Ce programme n'était à l'origine pas prévu pour gérer plusieurs indexations en même temps.

3. Entrez sur l'entrée standard la commande suivante : *make test_indexation.out*

Enfin, une fois la compilation terminée, entrez la commande suivante : ./test_indexation.out

Les fichiers descripteurs seront mis à jour (ou créés) dans le dossier.

Lancement du programme de recherche

1. Assurez-vous d'avoir placé les fichiers descripteurs dans le même dossier que le code source. Assurez-vous également d'avoir bien indiqué le nom du fichier audio de base dans le nomFichierBase, et le nom du fichier à comparer dans nomFichierComparaison1 ou 2.

Comme pour l'indexation, nous effectuons ici une recherche sur deux descripteurs d'un coup afin de montrer les possibilités du programme actuel, bien qu'à terme il était prévu de traiter les données comme indiqué dans le cahier des charges.

2. Si vous décidez de modifier le nombre de fichier à comparer, pensez à modifier le switch (ligne 62) et d'ajouter une case n pour tester le fichier. Nous avions à l'origine pensé à mettre une boucle de lecture de fichier ici, mais vu que les entrées-sorties ne fonctionnent pas, nous n'avons pas modifié cette partie de code.
3. Entrez sur l'entrée standard la commande suivante : *make test_recherhce.out*

Enfin, une fois la compilation terminée, entrez la commande suivante : ./test_recherche.out

Le résultat s'affiche sur la sortie standard sous la forme d'un tableau de fichier descripteur.

D. Difficultés rencontrées

1. Difficultés lors du traitement de texte

Lors de la phase d'indexation, lorsque nous lisons les mots présents dans le fichier « retranscriptionTri.txt » et que nous les ajoutons à la liste des termes, nous nous sommes aperçus qu'un terme fantôme était ajouté. Celui-ci possédait un champ « mot » vide et un champ « nombre d'occurrences » avec une valeur aléatoire.

N'ayant pu déterminer la cause de ce problème, nous nous sommes contentés de retirer simplement ce terme de la liste des termes présents dans le descripteur.

2. Difficultés lors du traitement de l'image

Pour ma part lors de ce projet les principales difficultés rencontrées ont été rapport à mon niveau en programmation. Pour être plus précis, c'est au début que mon niveau me posait problème, et comme j'ai défini mes structures et mon plan au début du projet, des erreurs de conception m'ont suivi jusqu'à la fin du projet. Par exemple, le choix de faire un tableau statique d'entiers ayant donc une taille fixe ne m'a pas permis par la suite de réaliser une quantification sur plus de 3 bits.

De la même manière, lors de ma quantification, mon manque de rigueur sur l'utilisation de tableaux dynamiques nous a posé problème lors de l'intégration. En effet, une perte beaucoup trop importante de bits lors de l'appel de mon indexation était en partie responsable du fait que notre indexation globale durait 12 minutes au début de l'intégration. J'ai donc passé ma dernière semaine à essayer de corriger mes erreurs/oublis qui posaient problème lors de l'intégration et qui n'avaient pas posés problème lors des tests unitaires. Cela ne m'a pas laissé le temps de rendre une recherche fonctionnelle. Pour corriger ça, nous avons utilisé le debugger gdb ainsi que Valgrind. Cela nous a beaucoup aidé à repérer nos défauts de conception, et nous a appris à utiliser un nouvel outil. Après ces corrections, l'indexation est passée de 12 min à 2min50.

En plus de ces lacunes en programmation, j'ai aussi rencontré de légers problèmes de communication avec mon équipe lors de la phase de conception et surtout sur le début du projet. Selon moi, ils étaient relativement faibles et presque inévitables sur un travail de groupe tel que celui-là.

3. Difficultés lors du traitement audio

Le codage de l'indexation et de la recherche de la partie audio m'a apporté une certaine difficulté, notamment sur deux points particuliers : les matrices dynamiques et la comparaison de matrices.

Les matrices dynamiques

Les pointeurs sont une notion essentielle à connaître pour pouvoir programmer efficacement en C, et la partie audio n'a pas dérogé à la règle. La déclaration, l'initialisation et la manipulation de matrices dynamiques a été un point d'arrêt important du développement.

Je ne maîtrisais pas vraiment les pointeurs, et l'utilisation d'un tableau à double dimension dynamique a été très difficile à comprendre et à utiliser. Cependant, cette difficulté a été passée suite à des heures de recherche et de programmation.

La comparaison de matrices

Une fois l'indexation des fichiers audio effectuée, il ne me restait que peu de temps pour programmer la recherche audio. Pour cela, il ne me restait plus qu'à coder ma fonction de comparaison d'histogrammes … Mais cela fut plus difficile que ce que j'avais prévu.

En effet, outre une manipulation difficile et bancale des matrices dynamiques, mes fonctions de comparaison n'étaient pas justes. J'ai codé et testé plusieurs fonctions de comparaison complexes, et aucune ne semblait faire l'affaire. La difficulté ne fut pas vraiment une question de programmation ici (quoique les matrices dynamiques …) mais plus une question de vérité mathématique. Au final, j'ai réussi à adapter une fonction utilisée dans l'industrie pour construire une fonction avec une réalité mathématique qui renvoie un résultat réel et crédible.

La gestion du temps

En l'état, notre planning n'était pas mauvais, mais nous n'avions pas prévu de prendre autant de retard suite à notre échec à coder certaines parties du programme, puis de l'implémenter. En outre, bien que nous ayons profité des vacances d'hiver pour développer le programme, nous n'avons pas réussi à être dans les délais estimés. Au final, nous avons à peine réussi à terminer les indexations et les recherches de fichier séparément que peu de temps avant la présentation. L'implémentation a été une catastrophe.

4. Difficultés lors de l'intégration

L'intégration fut la tache que nous avons trouvé la plus difficile. En effet, réunir et comprendre le code provenant de 5 personnes avec des réflexes de programmation différents ainsi que des découpages de code différents ne fut pas facile. Aussi, le fait que les interfaces entre les modules n'aient pas toujours été comprises ou respectées par tous, a rendu la tâche d'autant plus compliquée.

D'autre part, les multiples choix à faire pour trouver un bon équilibre entre optimisation et facilité/rapidité de développement n'étaient pas toujours simples à faire.

II. Organisation

A. Répartition du travail

Une fois les fonctionnalités du logiciel bien comprises lors de la phase de spécification, le groupe à cherchait une découpe cohérente du projet en sous-projets et en modules.

Ensuite, les taches ont été réparties de la manière suivante :

- Descripteurs Texte : LOUAHADJ Aniss
- Descripteurs Image : SANTENE Jeremy
- Descripteurs Audio : SUSGIN Jerome
- Pile : LANGLADE Alexandre
- Configuration : LANGLADE Alexandre
- Entrées : LANGLADE Alexandre
- Indexation Texte : LOUAHADJ Aniss
- Indexation Image : SANTENE Jeremy
- Indexation Audio : SUSGIN Jerome
- Recherche Texte : LOUAHADJ Aniss
- Recherche Image : SANTENE Jeremy – couleur dominante / CAMINO Amaia – comparaison
- Recherche Audio : SUSGIN Jerome
- Menu : CAMINO Amaia
- Main : CAMINO Amaia – main, gestion des erreurs / LANGLADE Alexandre – initialisation, fermeture

- Intégration : LANGLADE Alexandre

B. Coordination et communication interne

La coordination du groupe s'est construite autour d'un plan d'avancement global avec à chaque étape, des objectifs propres à chacun.

La communication interne s'est faite via plusieurs outils en fonction du besoin. Pour le partage de code fonctionnel nous avons utilisé l'outil Github. Pour partager du code en cours de développement dans le but de s'entraider et discuter autour nous avons utilisé Discord. Discord servait également à partager des documents divers ou à faire des vidéoconférences.

Un groupe Messenger été utilisé pour les échanges plus rapides d'informations.

C. Communication externe

Lors des différentes entrevues, le client a pu rencontrer tous les membres du groupe. Lorsqu'il était nécessaire de contacter le client à distance, par mail, un responsable des relations clients avait été nommé dans le groupe pour ne pas disperser les échanges.

III. Bilan

A. Bilan global

Le logiciel que nous avons créé respecte le cahier des charges fourni par notre client, malgré tout il manque des fonctionnalités pour le satisfaire entièrement.

Nous allons donc dresser un bilan des attentes du cahier des charges, ainsi que des fonctionnalités que nous avons pu livrer au client.

Notre logiciel devait permettre à un utilisateur de rechercher des fichiers de différents types. Pour cela, le logiciel devait se composer de deux modes, un mode utilisateur pour permettre à l'utilisateur d'effectuer les recherches, et un mode administrateur qui devait lui permettre de paramétrer ces recherches. En ouvrant notre logiciel, le choix du mode utilisateur et du mode administrateur est proposé.

Prenons d'abord le cas où l'utilisateur souhaite effectuer une recherche, il a donc choisi le mode utilisateur au démarrage du logiciel.

Le logiciel devait permettre à l'utilisateur de rechercher différents types de fichiers, c'est pourquoi notre logiciel propose de choisir entre une recherche de textes, de sons, ou d'images.

Chaque type de fichiers devait pouvoir être recherché par critère ou bien par comparaison. Ainsi on pouvait rechercher un texte par le biais d'un mot clef, une image par une couleur dominante, et un son long par un son plus court ou bien comparer un fichier, l'utilisateur devait alors choisir un fichier et le logiciel devait lui fournir la liste des fichiers les plus ressemblants. Notre logiciel ne permet malheureusement pas d'effectuer ces recherches, pour les sons et les textes ces fonctionnalités sont implémentées mais nous n'avons pas pu les intégrer au logiciel à temps.

Voyons maintenant le cas où l'utilisateur souhaite paramétrer le logiciel, il a donc choisi le mode administrateur au démarrage du logiciel.
Un mot de passe devait sécuriser l'accès au mode administrateur, notre logiciel demande donc à l'utilisateur d'entrer le mot de passe, s'il entre un mauvais mot de passe, le logiciel le prévient que le mot de passe est incorrect, et qu'au bout de trois essais il sera redirigé vers le mode utilisateur, qui lui ne nécessite aucun mot de passe.

Une fois le mot de passe correct entré, ce mode devait permettre à l'utilisateur de modifier les paramètres d'indexation, les paramètres de recherches, de lancer des indexations et de visualiser les descripteurs. Notre logiciel permet bien de modifier les paramètres d'indexation, mais pas ceux de la recherche, elle-même n'étant pas intégrée au logiciel.

Comme demandé dans le cahier des charges, les saisies de l'utilisateur sont traitées de manière à gérer les cas d'erreurs. Par exemple, si le logiciel attend que l'utilisateur entre un chiffre et que ce dernier entre un caractère, le logiciel signale l'erreur de saisie à l'utilisateur et lui propose à nouveau d'entrer un chiffre.

B. Retour d'expérience personnelle

CAMINO Amaia :

Etant redoublante, j'ai déjà une première expérience du projet fil rouge. Cependant ces deux expériences ont été pour moi très différentes. Tout d'abord alors que l'année dernière j'étais chargée de l'indexation et des recherches d'images, cette année j'étais chargée d'implémenter les menus et le mode administrateur. Au niveau programmation, cette année la difficulté était bien moindre, cependant j'ai aussi étudié la partie image cette année, car nous savions que nous ne pourrions pas tout coder à temps, mais j'ai eu des difficultés à comprendre le code de mon collègue.

Je regrette que nous n'ayons pas pris plus tôt conscience des difficultés, nous aurions pu modifier notre planning afin d'apporter de l'aide aux membres chargés des indexations et des recherches car malheureusement nous n'avons pas pu intégrer les recherches à notre logiciel.

J'ai aussi ressenti une différence au niveau du travail en équipe. Alors que l'année dernière nous travaillions souvent tous ensemble, nous prenions toutes les décisions ensemble, et nous communiquions énormément. Cette année nous nous sommes organisés différemment, nous avons choisi un « chef de groupe » qui nous fixait les objectifs et gérait le projet dans sa globalité. Je pense que l'idée de choisir un chef de groupe était bonne, même si nous n'avons pas su gérer le manque de niveau de la majorité des membres du groupe.

Ce projet reste pour moi une bonne expérience, l'entente au sein du groupe était bonne, et l'application de nos connaissances pour un projet concret est toujours pour moi très intéressante.

LANGLADE Alexandre :

Ce projet n'est pas mon premier projet informatique de groupe, cependant, cette première étape du projet fil rouge et les obstacles auxquelles je me suis confronté me permettent de tirer des leçons de cette expérience. Ces obstacles, nouveaux pour moi, concernent surtout un rôle que je tenais pour la première fois : chef de projet. J'ai pu découvrir ce que ce rôle impliqué et que les exigences étaient différentes de celles que l'on a lorsque l'on est membre de l'équipe.

Tout d'abord, la communication avec chaque membre doit être sans faille pour bien cerner l'avancement de chacun ou les problèmes rencontrés individuellement pour visualiser ensuite l'avancement de la mission dans sa globalité. Des manques de communication ont été responsables de nombreuses pertes de temps.

Aussi, j'ai ressenti une pression différente de celles que j'ai pu ressentir dans mes projets passés, je me sentais plus responsable du déroulé des évènements. Lorsque certaines choses ne se passaient pas correctement, je m'infligeais une plus forte culpabilité (de manière justifiée ou à tort) dû à ma responsabilité en tant que chef de projet.

Enfin, j'ai essayé d'être le plus à la hauteur possible en expérimentant des pratiques que je jugeais bonnes venant d'un chef de projet pour guider au mieux tout le groupe vers la réussite. J'ai essayé d'appliquer le plus de rigueur et d'organisation que je pouvais et les retours de mes camarades m'ont permis de comprendre les aspects à garder ou non de tout ce que j'ai pu tester.

LOUAHADJ Aniss :

Ce projet représente pour moi la première réelle expérience en programmation. En effet, il s'est révélé être un défi pour moi ainsi que pour mes compétences acquises au cours de mes deux années de pratique du langage C.

En faisant face à de nombreux obstacles au fur et à mesure de la programmation des différents aspects du traitement des texte, j'ai pu apprendre à utiliser un peu plus le système d'exploitation Linux mais aussi à mieux conceptualiser et à mieux intégrer les connaissances acquises lors de ma formation.

La réalisation de ce projet s'est avéré être un défi. Aussi bien sur le plan du travail du fait de mon manque d'aptitude et de connaissances actuelles en programmation C, mais aussi en raison du manque de communication et parfois du manque de motivation dont nous avons fait preuve au sein de notre équipe.

Heureusement, notre chef de projet LANGLADE Alexandre s'est révélé être un véritable atout, nous aidant à surmonter la plupart des obstacles auxquels nous avons eu à faire face.

Malgré notre incapacité à fournir un logiciel répondant en totalité au cahier des charges ; cette première phase du projet fil rouge est pour une expérience bénéfique du fait des connaissances acquises et de l'expérience acquise en travail de groupe qui était dans sa globalité agréable.

SANTENE Jeremy :

Ce projet était pour moi mon premier projet informatique en groupe. Il m'a donc permis de mieux comprendre les difficultés et le déroulement d'un projet informatique où un certain nombre de personnes implémentent un même logiciel. Aussi, j'ai pu constater que la rigueur et une bonne communication de groupe sont des notions importantes pour un projet comme celui-là.

En plus de me former à travailler sur ce type de projet, j'ai trouvé que c'était un très bon exercice de mise en application des cours de programmation, et ça m'a permis de progresser.

Enfin, comme je l'ai dit lorsque j'ai abordé les difficultés rencontrées, j'ai pu comprendre les difficultés que pouvait représenter le travail de groupe si nous n'avions pas une bonne communication.

SUSGIN Jerome :

Le projet fil rouge permet de voir les différents aspects d'un projet informatique, qu'il s'agisse du plan relationnel, organisationnel ou technique.

Le plan relationnel

Le projet fil rouge n'est pas mon premier projet universitaire, mais il m'a permis de confirmer mes dires sur le sujet : je préfère de loin le travail solitaire au travail de groupe, même si le travail d'équipe permet évidemment d'accomplir un travail plus important, de collaborer, d'avoir plus d'idées et d'obtenir des résultats plus satisfaisants.

Je dois admettre que j'ai peut-être travaillé un peu trop sur le côté de mon équipe, même si mon implication a été réelle et a permis de faire avancer le projet.

Le plan organisationnel

Contrairement aux autres projets universitaires auxquels j'ai participé, le projet fil rouge était de loin le mieux organisé par l'équipe pédagogique. J'ai pu très clairement voir la différence entre un projet bien géré et un projet mal géré. Cependant, je pense que l'organisation de notre équipe mérite d'être revue, car au final l'objectif final n'a pas été atteint : notre programme n'est pas complet.

De manière plus personnelle, j'ai commencé avec du retard le projet fil rouge, que j'ai réussi à rattraper au cours des vacances de fin d'année. Le développement du programme a cependant été stoppé car nous n'avons pas réussi l'implémentation.

Notre méthode de gestion de projet était prévue pour être agile, mais au final nous n'avons pas réussi à faire mieux qu'une méthode classique. La perte de temps qui en découle nous amène à l'échec.

Le plan technique

N'ayant pas de parcours d'informaticien, la technique était ce qui me faisait le plus peur. Les pointeurs m'ont causé pas mal de soucis d'une part, et la manipulation de données dynamiques d'autre part. Mais ces difficultés m'ont certainement permis de renforcer mes compétences sur les pointeurs, et la programmation C en général.

Outre l'aspect codage, la création d'histogrammes et le traitement mathématique de la comparaison m'a aussi apporté des techniques qui me seront peut-être utiles plus tard dans le cadre de la formation SRI. Je suis aujourd'hui plus serein avec l'utilisation des pointeurs, et je pense aussi pouvoir déclarer et manipuler des tableaux de n dimensions dynamiques sans trop de soucis. Cependant, j'ai encore des lacunes techniques vu que je n'ai pas réussi à corriger certaines erreurs de ma partie du programme.

IV. Annexes

A. Exemples de résultats

Indexation

Dans le dossier data/Descripteurs/ vous pourrez observer le résultat d'une indexation du corpus.

Recherche de texte

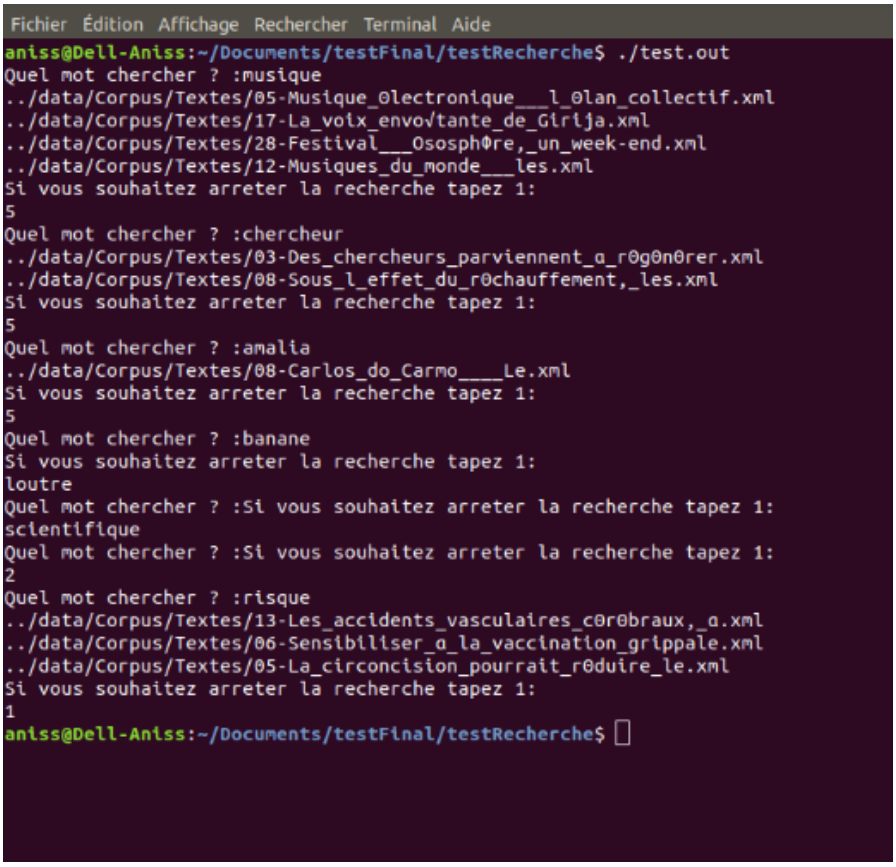
Dans l'image suivante, vous pourrez voir différents résultats de recherche par mot-clef.

Cette fonction n'étant pas implémenté elle a donc dû être testée séparément.

Lorsque nous rentrons le mot « musique » nous obtenons quatre chemins vers les fichiers audio où le mot musique est considéré comme un mot clé.

Lorsque nous rentrons le mot clé « chercheur » nous obtenons deux chemins correspondants aux fichiers où le mot est considéré comme un mot-clef.

Lorsque nous rentrons un mot qui n'est pas considéré comme un mot-clef dans aucun des documents du corpus rien ne s'affiche.



B. Manuel d'utilisation

A l'ouverture du logiciel, l'acteur doit choisir entre accéder au mode utilisateur en saisissant 1, et le mode administrateur en saisissant 2. Si l'acteur saisit 3, il quitte le logiciel. S'il saisit autre chose, il est averti que sa saisie est incorrecte et il peut à nouveau saisir son choix.

Si l'acteur a rentré 1, alors il entre dans le mode utilisateur. Il doit maintenant choisir entre rechercher des textes, des images ou des sons. Il peut aussi choisir de :

-S'il choisit les textes, il choisit ensuite entre rechercher un texte par comparaison ou par mot-clef. Dans les deux cas, l'utilisateur est alors averti que ces fonctions ne sont pas implémentées, et il peut continuer à naviguer dans le mode utilisateur.

-S'il choisit les images, il choisit ensuite entre rechercher une image par comparaison ou par couleur dominante. Dans les deux cas, l'utilisateur est alors averti que ces fonctions ne sont pas implémentées, et il peut continuer à naviguer dans le mode utilisateur.

-S'il choisit les sons, il choisit ensuite entre rechercher un son par comparaison ou par motif. Dans les deux cas, l'utilisateur est alors averti que ces fonctions ne sont pas implémentées, et il peut continuer à naviguer dans le mode utilisateur.

Si l'acteur a rentré 2 à l'ouverture du logiciel, il accède au mode administrateur. Ce mode est protégé par un mot de passe, après trois saisies incorrectes du mot de passe, l'acteur est redirigé directement vers le mode utilisateur.

Une fois que l'acteur a accédé au mode administrateur, le logiciel lui propose d'entrer des numéros correspondant à plusieurs choix : 1 lancer l'indexation complète, 2 lancer l'indexation d'un fichier seul, 3 modifier la configuration, 4 passer en mode utilisateur, ou 5 quitter le logiciel.

S'il choisit de modifier la configuration en entrant 3, les paramètres d'indexation actuels sont affichés et le logiciel propose à l'acteur de les modifier en entrant o pour oui et n pour non. Ensuite, s'il a saisi o, il doit entrer le numéro correspondant à l'indexation qu'il souhaite modifier : 1 ou 2 pour le paramétrage de l'indexation des textes, 3 pour celle des images et 4 ou 5 pour celle des sons.

Chaque paramètre est brièvement décrit afin que l'acteur puisse savoir comment modifier ces paramètres.

S'il choisit de lancer les indexations en entrant 1 ou 2, l'acteur est averti que ces fonctions ne sont pas implémentées et il peut continuer à naviguer dans le mode administrateur.