

Rapport de projet informatique

Affichage d'une image en caractères ASCII

Projet réalisé du 10 mars 2022 au 8 mai 2022

Membres du groupe

Crevot Gauthier 4100393
Chefirat Naïm 41001020

Table des matières

1	Environnement	4
1.1	Notre environnement de travail	4
1.1.1	A l'université	4
1.1.2	A la maison	4
1.2	Configuration nécessaire	4
2	Présentation du projet	6
2.1	Objectif	6
2.2	Présentation générale	6
2.2.1	Fonctions d'affichage ASCII	6
2.2.2	Autres fonctions	8
3	Outils, fonctions et algorithmes utilisés	10
3.1	Fonctions essentielles	10
3.1.1	La fonction fopen	10
3.1.2	La fonction fread	10
3.1.3	La fonction fseek	11
3.1.4	La fonction malloc	11
4	Travail réalisé	12
4.1	Répartition du travail	12
4.2	Comprendre le format Bitmap	12
4.2.1	Les en-têtes	12
4.2.2	Les différents formats Bitmap	13
4.2.3	Stockage des pixels en mémoire	13
4.3	Lecture et stockage des données de l'image	14
4.3.1	Lecture des en-têtes	14
4.3.2	Lecture des pixels	14
4.3.3	La matrice	15
4.4	Attribution des caractères pour chaque pixel	16
4.4.1	Niveau de gris d'un pixel	16
4.4.2	Choix du caractère	17
4.5	Création des menus	18
5	Difficultés rencontrées et solutions	20
5.1	Stocker les valeurs de chaque pixel	20
5.2	Réduire l'image en groupes de pixels	20
6	Bilan	22
6.1	Bilan personnel de Gauvain	22
6.2	Bilan Personnel de Naïm	22
6.3	Conclusion	22
6.3.1	Perspectives	22
7	Annexe	23
A	Webographie	24

B	Manuel utilisateur	25
B.1	Créer une image Bitmap	25
B.2	Mode d'emploi	25
B.2.1	Fonctions d'affichage ASCII	26
B.2.2	Autres fonctions	27
B.3	Erreurs courantes	28

1 Environnement

1.1 Notre environnement de travail

Pendant les mois de mars à mai, nous avons travaillé sur ce projet ensemble, mais pas toujours physiquement. Une partie de nos efforts ont eu lieu à l'université, pendant nos heures de TD ou en-dehors, tandis que l'autre s'est effectuée de manière individuelle, depuis chez nous. Bien que la quantité totale de travail fourni entre ces deux configurations soit équivalente (en excluant la rédaction de ce rapport), les tâches effectuées n'étaient pas les mêmes.

Nous allons détailler dans ce chapitre notre environnement de travail ainsi que l'organisation de ce dernier.

1.1.1 A l'université

La majorité du travail fourni depuis l'université a eu lieu durant les heures de TD. Nous avons profité de ces moments où nous pouvions échanger de vive voix et utiliser le même ordinateur pour discuter de nos idées les plus importantes durant la phase de conception, mais également pour mettre à jour nos connaissances sur le fonctionnement de certains concepts, fonctions ou algorithmes, qui n'étaient pas toujours équitables après une semaine de travail personnel.

C'est aussi à l'université que nos plus grandes avancées et découvertes ont eu lieu. Une grande partie des fonctions-clés de notre programme comme la lecture des informations de l'en-tête d'une image, la lecture des valeurs individuelles des pixels, ou encore la première ébauche de la fonction d'affichage ASCII ont été initialement rédigées en cours.

1.1.2 A la maison

En dehors de l'université, nous travaillons chacun depuis chez-soi, sur nos PC Windows, en utilisant Code Blocks ou Visual Studio. Tout était mis en commun grâce à un repository GitHub créé pour l'occasion. Nous communiquons à propos de nos avancées et questions avec les réseaux sociaux.

Notre travail personnel consistait essentiellement à de la réécriture et de implémentation des fonctions tests que nous ébauchions en TD. Nous avons également tiré parti de ces périodes pour entretenir notre code en le maintenant lisible grâce à l'intégration de structures composées et de fonctions séparées, mais aussi en ajoutant de nombreuses lignes de commentaires.

1.2 Configuration nécessaire

La configuration nécessaire pour utiliser notre programme est très peu contraignante. Il suffit, en plus de disposer du fichier exécutable principal, d'avoir des images au format Bitmap 24 bits/pixel afin de l'utiliser. Il fonctionne sous Windows et Linux à la seule condition que la machine sur laquelle il est utilisé stocke les variables entières sur 4 octets et les caractères sur 1 seul, ce qui est le cas de la majorité des processeurs modernes. Nous reviendrons sur l'origine de ces spécificités plus tard.

Notre projet est intégralement rédigé en langage C standard et n'utilise aucune fonction provenant de bibliothèques qui ne sont pas pré-installées. L'établissement d'un environnement de travail personnel n'a donc pas été une contrainte pour nous. Il nous

suffisait de disposer d'un accès internet pour publier notre progrès sur GitHub, d'un IDE au choix, et d'un compilateur standard.

```
Naim Chefirat et Gouvain Crevot

****MENU PRINCIPAL****

~Que voulez-vous faire?~

1: Affichage ascii
2: Autres fonctions
3: Quitter
```

FIGURE 1 – Menu principal

2 Présentation du projet

2.1 Objectif

Notre objectif lors de l'élaboration de ce projet était d'écrire un programme qui soit le plus versatile possible et qui laisse à l'utilisateur le choix entre de nombreuses fonctions afin de satisfaire les besoins les plus spécifiques. Réussir à rédiger et à implémenter un aussi grand nombre de fonctions et de paramètres tout en maintenant un code organisé, lisible et en évitant les redondances a donc été un enjeu majeur pour nous.

2.2 Présentation générale

Notre programme s'ouvre sur un menu principal qui met en avant notre objectif : il propose des choix à l'utilisateur. Chacune des deux options proposées entraînera l'utilisateur à répondre à d'avantage de questions, de plus en plus spécifiques, afin de cibler au mieux son besoin, tout en restant ergonomique et interactif.

2.2.1 Fonctions d'affichage ASCII

Lorsque l'utilisateur choisit cette première option, il lui est tout d'abord demandé la couleur de son arrière plan, afin de savoir si l'image doit être affichée en négatif ou non.

L'utilisateur est ensuite amené à choisir une image, qui doit être présente dans le dossier du programme avant d'être confronté à de nouvelles options. Le nombre de caractères disponibles a une grande importance dans le rendu final. Dans la majorité des cas, le meilleur choix est le premier, mais avec certaines images, il peut être utile d'avoir accès à ces options. En effet, les options 16 et 64 pixels ne sont viables qu'avec les images sombres, avec peu de contraste.

```

Quelle est la couleur de l'arriere-plan? (1: noir; 2: blanc)
1

~Affichage ASCII~

Choisir une image: test.bmp

Combien de caracteres? (1: 8; 2:16; 3: 64)
1

Reduire la hauteur?
1: Non
2: Afficher 1 pixel sur 2
3: Afficher 1 pixel sur 3
4: Afficher 1 pixel sur 4
2

Reduire la largeur?
1: Non
2: Afficher 1 pixel sur 2
3: Afficher 1 pixel sur 3
4: Afficher 1 pixel sur 4
1

```

FIGURE 2 – Menu ASCII

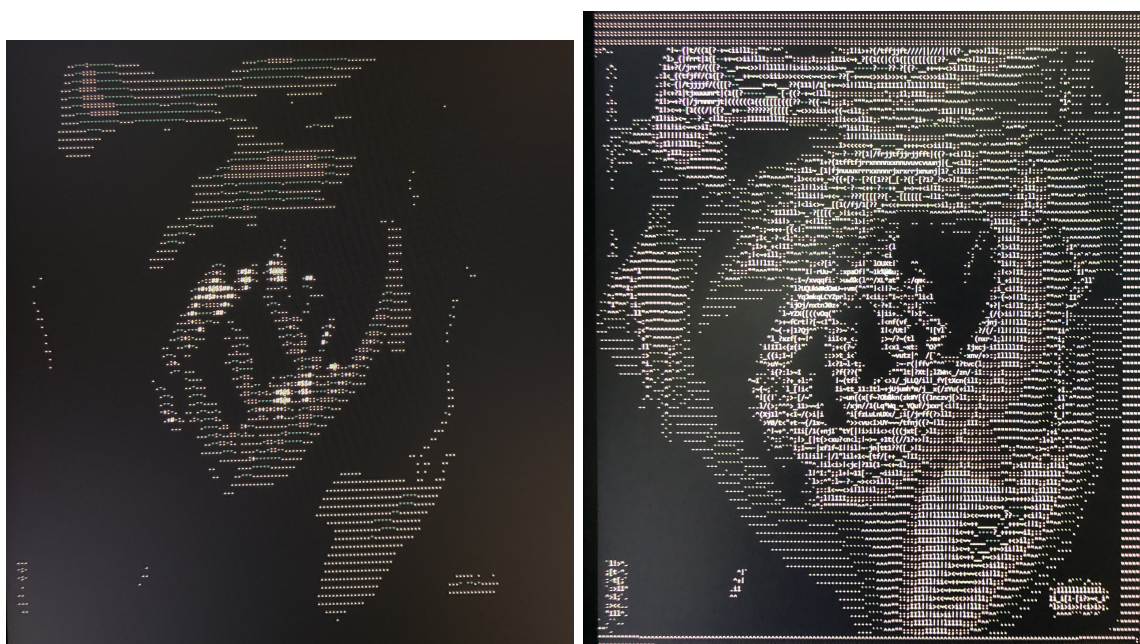


FIGURE 3 – Exemple d'image dont le rendu est meilleur avec 64 caractères

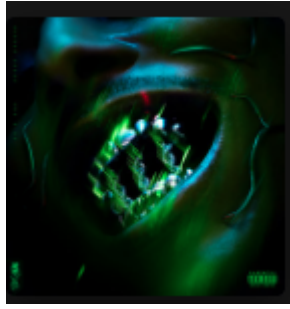


FIGURE 4 – Image originale

L'utilisateur est aussi amené à réduire la taille de son image afin qu'elle puisse être affichée entièrement dans la console sans distorsions et découpages. Ces options permettent également d'afficher l'image dans des proportions plus proches de l'originelle. En effet, sans aucune modification, l'image affichée paraîtra étirée verticalement à cause du fait que les caractères ASCII sont plus longs que larges. Ce problème est facilement résolu en choisissant d'afficher un pixel de hauteur sur 2, mais tous les pixels de largeur. Le programme peut alors garantir un rendu satisfaisant sur des images de 50x50 comme de 1200x1200 pixels, si l'utilisateur choisit les paramètres adéquats (voir le manuel utilisateur en annexe).

2.2.2 Autres fonctions

Ce menu secondaire regroupe des fonctions qui ont été très utiles lors de l'écriture du projet. Elles permettent à l'utilisateur de comprendre le fonctionnement du code, et de vérifier que l'image qu'ils essaient d'utiliser est valide.

```
~Autres Fonctions~  
  
Choisir une image: test.bmp  
  
~Que voulez-vous faire?~  
  
1: Afficher les infos de l'image  
2: Afficher la matrice  
3: Afficher les valeurs de chaque pixel, pixel par pixel  
█
```

FIGURE 5 – Menu des fonctions additionnelles

La première proposition affiche les informations contenues dans l'en-tête essentielles à l'analyse de l'image. Par exemple, le numéro du premier octet contenant les valeurs des pixels, le nombre de bits alloués à chaque pixel, ou encore la taille de l'image sans l'en-tête. Des informations très utiles en cas d'éventuel bug.


```
Infos:

Type d'image: BM
Taille de l'image (headers inclus): 769006
Offset (nombre d'octets avant le premier pixel): 54

Taille du header DIB: 40
Largeur: 369px
Hauteur: 694px
Color Planes: 1
Bits par pixel: 24
Methode de compression: 0
Taille de l'image sans headers(PixelArraySize): 768952

Taille de chaque row: 1108
Nombre total de pixels: 256086
Nombre d'octets de padding: 694
Padding par row: 1
Pixels par row: 369

Continuer? (1: oui; 2: non)
█
```

FIGURE 6 – Informations importantes sur une image

Les deux autres fonctions disponibles affichent de façon différente une liste ordonnée des valeurs de tous les pixels de l'image, l'une sous forme de grille, l'autre par lignes. Les pixels sont affichés de bas en haut et de gauche à droite, et leurs valeurs sont dans l'ordre Bleu, Vert, Rouge, car c'est ainsi qu'ils sont stockés en mémoire.

```
Matrice:

Affichage Matrice:

(0,0): 0,0,255 (0,1): 255,255,255
(1,0): 255,0,0 (1,1): 0,255,0

Continuer? (1: oui; 2: non)
2
Retour au menu principal
```

```
Pixels:

Pixel 1: 0, 0, 255
Pixel 2: 255, 255, 255
Pixel 3: 255, 0, 0
Pixel 4: 0, 255, 0

Continuer? (1: oui; 2: non)
█
```

FIGURE 7 – Affichage des valeurs des pixels sous format de liste et de matrice

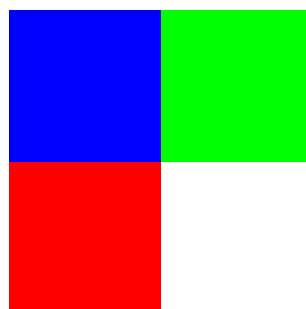


FIGURE 8 – Image utilisée (2x2px)

3 Outils, fonctions et algorithmes utilisés

Aucune bibliothèque externe n'a été utilisée. Notre programme a été réalisé avec l'aide de très peu d'outils, toutes les fonctions utilisées proviennent de `stdlib.h`, `stdio.h`, `stdbool.h` et `string.h`. Nous allons expliquer les plus importantes et leur utilité dans ce chapitre.

3.1 Fonctions essentielles

3.1.1 La fonction `fopen`

La fonction `fopen` nous était déjà familière avant la réalisation de ce projet, mais l'utilisation que nous en avons fait ici est légèrement différente.

A partir d'un pointeur de type `FILE` passé en argument, la fonction `fopen` permet d'ouvrir les fichiers de tout type, avec plusieurs modes de lecture ou écriture (ou les deux). Ici, le type de fichier ouvert est Bitmap (extension `.bmp`), et en mode lecture. Or, nous ne souhaitons pas ici lire du texte avec le mode `"r"` comme nous y étions habitués mais des données en binaire. Le mode de lecture que nous utiliserons est donc `"rb"`. Nous allons ainsi pouvoir lire tout le contenu de l'image octet par octet grâce aux fonctions suivantes.

3.1.2 La fonction `fread`

Pour la lecture de fichiers texte remplis de caractères, nous utilisons les fonctions `fscanf` ou `fgets/fgetc`. La fonction `fread` fonctionne de la même façon mais permet la lecture de données en binaire. elle prend en paramètres l'adresse d'une variable dans laquelle on va stocker l'élément lu, la taille en octets de l'élément à lire, le nombre d'éléments à lire et le pointeur de type `FILE` associé au fichier à lire.

```
char valeur1;
short int valeur2;
int valeur3;
FILE *fp = fopen("test.bmp", "rb");
if (fp == NULL){
    printf("erreur");
    return 1;
}
else{
    fread(&valeur1, 1, 1, fp);
    fread(&valeur2, 2, 1, fp);
    fread(&valeur3, 4, 1, fp);
}
```

FIGURE 9 – exemple d'utilisation de la fonction `fread` pour des valeurs de tailles différentes

3.1.3 La fonction fseek

Comme pour les autres fonctions de lecture de fichiers, à chaque lecture, la fonction `fread` déplace le curseur du pointeur derrière l'élément qui vient d'être lu. Cela permet de lire tout un fichier dans l'ordre, mais cela pose problème lorsqu'on veut accéder à une partie spécifique du fichier ou, par exemple, le lire à l'envers.

La fonction `fseek` est faite pour ça. Elle prend en paramètres le pointeur de type `FILE`, le nombre d'octets à avancer (doit être négatif pour reculer), et l'endroit du fichier à partir duquel effectuer ce décalage. Ce dernier est spécifié à partir de 3 constantes : `"SEEK_SET"` désigne le début du fichier, `"SEEK_CUR"` désigne la position actuelle du curseur, et `"SEEK_END"` désigne la fin du fichier.

Grâce à ces outils, on peut déplacer le curseur librement dans le fichier et assigner à des variables des valeurs binaires de toutes les tailles.

```
fseek(fp, bmp_header.offset, SEEK_SET);
printf("\nPixel Array: \n"); // affichage des octets 1 par 1
for (int i = 1; i <= dib_header.height; i++)
{ // ligne i
    for (int j = 1; j <= dib_header.width; j++)
    { // pixel j de la ième ligne
        fread(&pixel, 3, 1, fp);
        printf("%d | %d | %d | ", pixel.b, pixel.g, pixel.r);
    }
}
```

FIGURE 10 – Exemple d'utilisation de la fonction `fseek` pour se rendre directement au début du Pixel Array, indiqué par la valeur "offset" du header BMP

3.1.4 La fonction malloc

Une fois lues, pour stocker les valeurs de tous les pixels de façon ordonnée et facile d'accès, nous utilisons un tableau à deux dimensions : une matrice. Il est possible d'en déclarer avec la syntaxe `"int matrice[][]"`, mais comme notre programme est écrit pour fonctionner avec des images de toutes tailles, nous avons besoin d'allocation dynamique pour attribuer une quantité propre à chaque image de mémoire persistante.

Nous utilisons donc la fonction `malloc` pour déclarer un pointeur vers un tableau de tableaux de pixels.

```
int **matrice;
int i;
matrice = (int **)malloc(hauteur * sizeof(int *));
for (i = 0; i < hauteur; i++)
{
    matrice[i] = (int *)malloc(largeur * sizeof(int));
}
return matrice;
```

FIGURE 11 – Exemple de déclaration d'une matrice d'entiers avec la fonction `malloc`

4 Travail réalisé

Nous allons dans ce chapitre expliquer en détail tout notre travail au cours de ces deux mois, en détaillant de manière précise le fonctionnement des grandes parties de notre code.

4.1 Répartition du travail

Le travail fourni par chaque membre du groupe n'est pas quelque chose de facile à quantifier. Nous avons consacré notre temps personnel à des tâches assez différentes. Nous nous sommes répartis de la manière suivante :

Naïm a apporté les idées clés, les lignes directrices sur la tournure que devait prendre le projet et la façon dont devait fonctionner le code. C'est lui qui a déterminé les outils et fonctions à utiliser au départ. De ce fait, il était celui qui accomplissait les grandes avancées et débloquent les difficultés majeures. Il est à l'origine des algorithmes de lecture des données des en-têtes et des pixels, de la création d'une matrice aux dimensions de l'image, et de l'algorithme assignant des caractères à chaque pixel.

Gauvain a également apporté sa part d'idées, notamment celle de stocker les données lues dans des structures globales qui pouvaient être passées en paramètres de fonctions facilement. Il a également apporté des directives sur l'imbrication des différents menus et a passé du temps à paufiner l'interface utilisateur. La plus grande partie de son travail a été de maintenir le code propre, lisible, et de faire en sorte que les différentes fonctions se complémentent sans soucis.

Tandis que Naïm rédigeait de longs blocs de code incorporant déclarations, affectations, algorithme principal, et affichage des résultats pour déboguer, Gauvain se chargeait de les incorporer au programme principal en les séparant en petites fonctions et en remplaçant les variables temporaires par celles provenant des structures adéquates.

La rédaction de ce rapport a suivi le même processus. Naïm s'est occupé d'apprendre le fonctionnement de \LaTeX pendant que Gauvain a créé un plan et rédigé certaines parties. Naïm a ensuite écrit et ajusté un bon nombre de parties directement en \LaTeX et a ajouté les images.

On peut donc estimer la part de travail fournie par les deux membres à environ 40% pour Gauvain et environ 60% Pour Naïm.

4.2 Comprendre le format Bitmap

La première étape dans la conception de ce projet a été de comprendre comment les images fonctionnaient et pouvaient être lues ou écrites en C. A l'origine, l'objectif était de créer un algorithme qui fonctionnait pour les formats d'images les plus courants : PNG, JPEG, etc, mais nous avons compris très tôt que nous devrions nous concentrer sur un seul. Nous nous sommes alors rapidement orientés vers le format Bitmap car il s'agit du plus simple et "pur" : pas de calques de transparence, aucune compression, seulement quelques informations sur l'image puis un tableau listant les valeurs individuelles de chaque pixel.

4.2.1 Les en-têtes

La première partie de tous les fichiers Bitmap est le "BMP header", une en-tête de précisément 14 octets contenant des informations essentielles. Par exemple, les deux

premiers octets de l'image contiennent deux caractères : 'B' et 'M', pour indiquer le type de l'image. le header BMP contient aussi la taille totale de l'image en octets, ainsi que l'offset : l'octet contenant les valeurs du premier pixel de l'image. Le header suivant est le "DIB" et occupe 40 octets, il contient des informations comme la longueur et hauteur de l'image.

4.2.2 Les différents formats Bitmap

Il existe plusieurs types de format Bitmap, que l'on distingue par le nombre de bits associés a chaque pixel. Le format 1bit/pixel par exemple ne supporte que deux couleurs, noir et blanc. Le format 8bits/pixel permet d'en afficher 256, chaque pixel occupant 1 octet dans la mémoire. Ces formats avec un nombre limité de couleurs réservent certains octets à une "color palette", entre la fin du header DIB et le début des valeurs des pixels. Sa taille est différente en fonction du nombre de bits/pixels, et elle est de 0 pour les formats 24 et 32 bits. Le format 32 bits possède en revanche une autre spécificité : un niveau d'alpha pour chaque pixel. Nous avons donc choisi d'utiliser le format le plus pratique et standard, 24 bits/pixels. Chaque pixel occupe 3 octets en mémoire : 1 octet pour la valeur de bleu, 1 pour la valeur de vert, et 1 pour la valeur de rouge, dans cet ordre précis. Ce format permet d'afficher 16 777 216 couleurs différentes.

Notre programme ne fonctionne qu'avec ce format car nos appels à la fonction fread lisent spécifiquement 3 octets pour chaque pixels. Essayer de lire des pixels occupant un nombre de bits différent ne fonctionnerait donc pas.

4.2.3 Stockage des pixels en mémoire

Le "Pixel Array", le tableau où tous les pixels sont stockés à la suite dans le fichier, commence à l'octet "offset" contenu dans le header BMP et s'arrête à la fin de l'image. Les pixels y sont rangés par lignes, il y en a autant qu'il y a de pixels de hauteur dans l'image, et le nombre de pixels par ligne correspond au nombre de pixels de largeur de l'image. Une spécificité importante du format est que la taille en octets de chaque ligne (3*la largeur de l'image car 3 octets par pixels) doit être un multiple de 4. Lorsque ce n'est pas le cas, des octets de "padding" (valant souvent 0) sont ajoutés à la fin de chaque ligne.

Par exemple, une image d'un seul pixel de large occupera 3 octets par ligne (car 3 octets par pixel), il y aura donc un octet de padding à la fin de chaque ligne. Si l'image faisait 3 pixels de large, alors chaque ligne contiendrait 9 octets de données et 3 octets de padding car $12/3 = 4$.

On peut déterminer le nombre d'octet de padding par ligne assez facilement car le header DIB nous indique la taille de l'image sans ses headers ("Pixel Array Size"). En multipliant la largeur et la hauteur de l'image, on obtient le nombre total de pixels de l'image, qu'on peut multiplier par 3 pour connaître le nombre d'octets alloués à des pixels. Il suffit alors de soustraire au nombre total d'octets du Pixel Array le nombre d'octets alloués à des pixels. Nous obtenons ainsi le nombre d'octets de padding sur toute l'image, qu'on peut diviser par la hauteur de l'image (le nombre de lignes) pour connaître le nombre d'octets de padding à la fin de chaque ligne. Ce calcul est essentiel pour la lecture d'une image, car sans lui nous ne pourrions pas utiliser la fonction fseek pour sauter le bon nombre d'octets après chaque ligne.

4.3 Lecture et stockage des données de l'image

Afin de maintenir le code organisé et les valeurs importantes accessibles, nous avons créé 4 structures de données. Les deux premières contiennent les données des headers BMP et DIB. Certaines valeurs occupant plus d'octets que d'autres, on les assigne à des variables de type adéquat : unsigned char pour celles occupant 1 octet, unsigned short int pour 2, et unsigned int pour 4. La troisième structure utilise les informations des deux premières pour effectuer des calculs essentiels au traitement de l'image, comme celui expliqué au dessus.

```
struct info
{
    int rowsize; // taille en octets de chaque row
    int pixels_nb; // nombre total de pixels sur l'image
    int padding_total; // nombre total d'octets de padding
    int padding_row; // nombre d'octets de padding par row
    int pixels_row; // nombre total de pixels par row
};
```

FIGURE 12 – Déclaration de la structure "info"

La quatrième structure correspond au type de donnée "pixel" que nous avons créé. Une variable de type pixel est composée de 3 octets : un pour sa valeur de bleu, un pour sa valeur de vert, et un pour sa valeur de rouge.

4.3.1 Lecture des en-têtes

Afin de lire les données des en-têtes et de les stocker, il nous suffit de déclarer les deux structures "bmp_header" et "dib_header" et de les passer en argument de fonctions qui vont faire appel à fread. Il est même possible de d'attribuer les valeurs lues à toute une structure d'un coup, au lieu de le faire variable par variable.

```
fread(bmp_header.name, 2, 1, fp);
fread(&bmp_header.size, 4, 1, fp);
fread(&bmp_header.app1, 2, 1, fp);
fread(&bmp_header.app2, 2, 1, fp);
fread(&bmp_header.offset, 4, 1, fp);

fseek(fp, 14, SEEK_SET);
fread(&dib_header, sizeof(struct dib_header), 1, fp);
```

FIGURE 13 – Lecture du header BMP variable par variable, et du DIB d'un coup. Ces deux fonctions de lecture retournent ensuite le header passé en argument.

4.3.2 Lecture des pixels

La lecture des valeurs des pixels repose sur le même principe. On commence par créer une variable de type pixel (3 unsigned char correspondants chacun à une couleur),

puis, avec des boucles for imbriquées, on peut utiliser fread pour lire individuellement tous les octets de l'image et fseek pour sauter les éventuels octets de padding en fin de ligne.

```
struct pixel pixel;
fseek(fp, bmp_header.offset, SEEK_SET);
for (int i = 1; i <= dib_header.height; i++)
{
    for (int j = 1; j <= info.pixels_row; j++)
    {
        fread(&pixel, 3, 1, fp);
    }
    fseek(fp, info.padding_row, SEEK_CUR);
}
```

FIGURE 14 – Lecture des valeurs de chaque pixel successivement

Un problème se pose : les données ne persistent pas, la variable pixel change de valeur à chaque tour de boucle.

4.3.3 La matrice

Pour remédier à ce problème, notre solution a été d'allouer dynamiquement une matrice de pixels, avec la méthode expliquée plus haut. Pour cela nous avons créé une fonction qui alloue la quantité de mémoire nécessaire à chaque image, grâce aux valeurs de ses headers DIB et infos passées en paramètres. Il suffit alors d'utiliser le même algorithme que le précédent, en stockant chaque pixel dans la matrice au moment de sa lecture.

```

pixel **matrice;
int i;
matrice = (pixel **)malloc(dib_header.height * sizeof(pixel *));
for (i = 0; i < dib_header.height; i++)
{
    matrice[i] = (pixel *)malloc(dib_header.width * sizeof(pixel));
}

pixel px;
fseek(fp, bmp_header.offset, SEEK_SET);
for (i = dib_header.height - 1; i >= 0; i--)
{
    for (j = dib_header.width - 1; j >= 0; j--)
    {
        fread(&px, 3, 1, fp);
        matrice[i][j].b = px.b;
        matrice[i][j].g = px.g;
        matrice[i][j].r = px.r;
    }
    fseek(fp, info.padding_row, SEEK_CUR);
}

```

FIGURE 15 – Allocation dynamique d’une matrice aux dimensions de l’image et remplissage de cette dernière

Ici, on lit les images de ”haut en bas” car elles sont en réalité stockées de bas en haut.

4.4 Attribution des caractères pour chaque pixel

Maintenant que nous disposons d’une matrice contenant les valeurs de chaque pixels, il ne nous reste plus qu’à choisir pour chacun d’entre eux le pixel à afficher. Pour commencer, il nous faut un tableau de caractères triés du plus ”foncé” au plus ”clair”. Certains caractères occupant plus de pixels que d’autres, ils serviront à représenter les zones sombres de l’image, tandis que les plus vides serviront aux zones claires (inversement pour les arrières plans foncés). Il nous faut alors un moyen de déterminer la ”luminosité” ou ”clarté” d’un pixel.

4.4.1 Niveau de gris d’un pixel

Pour résoudre ce problème, nous allons transformer ce pixel coloré en un pixel en ”noir et blanc”, ou plutôt en niveau de gris. Il existe plusieurs méthodes : la plus simple serait de diviser la somme des valeurs des trois composantes bleu, vert et rouge par 3. Bien que cette méthode fonctionne, le résultat est loin d’être optimal. Le meilleur moyen de déterminer le niveau de gris d’un pixel RGB est d’utiliser la formule NTSC : $(0.299 * R) + (0.587 * G) + (0.114 * B)$.

4.4.2 Choix du caractère

Notre idée initiale pour choisir un caractère était d'utiliser un switch pour afficher un caractère plus ou moins opaque en fonction de la valeur du niveau de gris du pixel. Nous avons plus tard trouvé une méthode plus précise et moins redondante, en utilisant des boucles for imbriquées. Nous avons besoin pour cela d'un tableau "ascii" dont la taille est multiple de 256 (8 dans l'exemple suivant) de caractères ordonnés, et d'une fonction permettant d'appliquer la formule NTSC à un pixel.

Nous parcourons la matrice à l'aide de boucles imbriquées et transformons chaque pixel en niveau de gris avec la fonction gray_pixel. La valeur de $R = G = B$ du pixel est assignée à une variable unsigned char. Nous affichons ensuite la case de cette valeur RGB divisée par 32 du tableau "ascii", car cette valeur est comprise entre 0 et 255. Il y a donc 256 valeurs possibles, et $256/32 = 8$, le nombre de cases de notre tableau. Nous divisons donc la valeur RGB de notre pixel par 32 pour obtenir une valeur comprise entre 0 et 8.

```
unsigned char charr;
char ascii[] = {'@', '$', '#', '+', ':', '-', '.', ' '};
for (i = 0; i < dib_header.height; i++)
{
    for (j = dib_header.width - 1; j >= 0; j--)
    {
        charr = gray_pixel(matrice[i][j]);
        printf("%c", ascii[charr / 32]);
    }
    printf("\n");
}
```

FIGURE 16 – Affichage d'un caractère ASCII à partir d'un tableau de taille 8

Cette fonction peut être adaptée facilement à des tableaux de différentes tailles multiples de 256, comme 16 ou 64. Il faut pour ça remplacer 32 par le quotient de la division de 16 ou 64 par 256.

Pour afficher l'image en négatif, afin que notre programme fonctionne sur fonds des clairs et foncés, il n'est pas nécessaire de créer un autre tableau dont l'ordre est inversé. Dans l'affichage "ascii[charr / (256/nb_chars)]", avec "nb_chars" la taille du tableau, "charr / (256/nb_chars)" est une valeur positive comprise entre 0 et la fin du tableau. Pour l'inverser, on la soustrait à la taille du tableau - 1 car les valeurs du tableau commencent à 0 : "ascii[(nb_chars-1) - charr / (256/nb_chars)]". On obtient alors le même tableau, mais les premières valeurs se retrouvent à la fin et inversement. Les tableaux de caractères que nous utilisons ont été créés à partir de ressources en ligne et ajustés pour que leur taille soit adéquate.

```

if (negatif == 1)
{
    for (i = 0; i < dib_header.height; i = i + h)
    {
        for (j = dib_header.width - 1; j >= 0; j = j - w)
        {
            charr = gray_pixel(matrice[i][j]);
            printf("%c", ascii[(nb_chars - 1) - charr / (256 / nb_chars)]);
        }

        printf("\n");
    }
}

else
{
    for (i = 0; i < dib_header.height; i = i + h)
    {
        for (j = dib_header.width - 1; j >= 0; j = j - w)
        {
            charr = gray_pixel(matrice[i][j]);
            // printf("%c", ascii[7 - charr / 32]);
            printf("%c", ascii[charr / (256 / nb_chars)]);
        }

        printf("\n");
    }
}

```

FIGURE 17 – Affichage en ascii à partir d'un tableau de taille indéterminée, en négatif si désiré

4.5 Création des menus

La dernière étape, une fois tous les algorithmes créés et séparés en fonctions à été de créer des menus pour rendre le programme utilisable sans avoir à modifier des lignes de codes. Sur ce point, nous sommes restés fidèles à notre vision d'origine et avons élaboré des menus minimalistes, qui s'imbriquaient de façon logique et compréhensible afin de maintenir un code lisible, tout en proposant une expérience satisfaisante à l'utilisateur. Nous avons fait de notre mieux pour les incorporer aux bons endroits dans le code, mais des redondances étaient inévitables puisque nous avons choisi de vérifier que chaque réponse de l'utilisateur était censée.

Nos menus sont composés de switches, afin de rediriger l'utilisateur vers la fonction demandée, qui peut elle aussi incorporer un menu. Ces switches sont plus efficaces que les alternatives if et else en grand nombre, car leur syntaxe est plus compacte et facile à lire.

Afin de vérifier si la réponse de l'utilisateur est valide, une boucle do-while est utilisée. On peut ainsi poser une question à l'utilisateur puis enregistrer sa réponse avec la fonction scanf et reposer la question si la valeur scanée ne correspond pas à celle attendue.

```
do
{
    printf("\nContinuer? (1: oui; 2: non\n");
    scanf("%d", &reponse);
    if (reponse != 1 && reponse != 2)
    {
        printf("\n*****\tReponse incorrecte\t*****\n\n");
    }
} while (reponse != 1 && reponse != 2);
```

FIGURE 18 – Exemple d'utilisation d'une boucle do-while pour la création de menus

5 Difficultés rencontrées et solutions

Certains algorithmes détaillés dans le chapitre précédent nous ont pris plus de temps à développer et implémenter dans le code. Nous allons les expliquer ici.

5.1 Stocker les valeurs de chaque pixel

Une fois le format bitmap compris et les fonctions `fread` et `fseek` maîtrisées, il nous a fallu peu de temps pour être capables de lire les valeurs d'un pixel et les afficher à l'écran. C'est à ce moment qu'a été rédigée la fonction permettant de tous les afficher à la suite, qui est toujours disponible dans le code final. Nous pouvions même, en nous basant sur cette fonction, afficher un caractère par pixel, au lieu d'afficher ses valeurs. Malgré cela, l'idée d'associer le Pixel Array à une matrice afin de pouvoir y accéder sans avoir à effectuer une lecture complète de l'image est vite apparue. Notre objectif était d'éviter les redondances, de rendre le programme plus rapide et léger, et de rendre d'avantage de manipulations d'image nécessitant une réécriture possibles (créer une copie de l'image en noir et blanc par exemple).

L'allocation dynamique de cette matrice nous a posé problème car nous nous sommes trompés dans les dimensions et avons créé une matrice de hauteur "largeur" et de largeur "hauteur" sans nous en rendre compte. Nous avons perdu beaucoup de temps sur cette erreur car nous pensions que le problème venait de notre fonction qui remplissait la matrice, ou de celle qui l'affichait à l'écran. Ces fonctions avaient elles aussi des problèmes de dimensions, nous confondions parfois la hauteur et la largeur, et parfois non. Le débogage de cette partie du code a été laborieux car nous avions des difficultés à cibler la ou les erreurs. Malgré les nombreux schémas que nous avons dessinés et consultés en ligne, le concept des boucles `for` imbriqués pour manipuler un tableau de tableaux de pixels était difficile à se représenter.

Nous avons fini par réaliser que notre erreur principale provenait de l'allocation de notre matrice après de multiples tests sur des petites matrices d'entiers. Nous avons alors compris que depuis le début, nous tentions d'associer la première dimension de la matrice à la largeur de notre image, et non la hauteur. Le point que nous appelions (1,2) était en fait le point de coordonnées (2,1) pour le programme.

5.2 Réduire l'image en groupes de pixels

A la base, notre solution pour adapter le code aux grandes images était d'attribuer et d'afficher des caractères pour des groupes de pixels (des carrés de 2x2, 4x4 ou 8x8 pixels par exemple). Plusieurs contraintes se sont rapidement posées. Premièrement, fonctionner avec des carrés signifiait que nous devions lire les deux premiers pixels d'une ligne, aller à la ligne suivante et lire ses deux premiers pixels, puis revenir à la ligne du dessus pour poursuivre la lecture. Cet algorithme est bien plus complexe qu'une lecture ligne par ligne, mais ce n'est pas le plus gros problème.

La raison qui nous a empêché de poursuivre cette idée est que les dimensions des images ne sont pas toutes multiples de la taille de nos carrés. Pire, certaines combinaisons de hauteur et largeur sont premières entre elles. Si nous avions continué dans cette direction, nous aurions dû implémenter un calcul des diviseurs communs des dimensions de chaque image, et certaines n'auraient pas pu être réduites autant que d'autres.

La solution à ce problème de réduction d'image nous est venue alors que nous cherchions à en résoudre un autre : l'étirement vertical de l'image. Comme expliqué précédemment, les caractères étant plus longs que larges, les images que nous affichions paraissaient

étirées. Nous avons alors eu l'idée de ne pas afficher tous les pixels de hauteur, afin de rétablir les proportions. Nous avons ensuite décidé d'appliquer cette solution à la largeur de l'image, afin de pouvoir réduire efficacement la taille qu'occupait nos images dans la console. L'utilisateur a donc le choix de n'afficher qu'un pixel sur deux, trois ou quatre pour chacune des dimensions. Cela a malheureusement pour coût d'entraîner une perte de détails non négligeable dans le rendu final, mais c'est le meilleur compromis que nous ayons trouvé. Pour connaître la meilleure façon de tirer parti de ces réductions d'image, vous pouvez vous référer au manuel utilisateur en annexe.

6 Bilan

6.1 Bilan personnel de Gauvain

Le fait de travailler en groupe a permis à Gauvain de se familiariser avec de nouvelles méthodes et outils de travail apportés par Naïm. En prenant part au développement d'un projet complet, il a pu surmonter des difficultés sur certaines notions du cours, en les appliquant à des situations concrètes. Il a également beaucoup appris sur le fonctionnement et la structure d'un long programme interactif, notamment grâce à son travail sur la rédaction de fonctions séparées se faisant appel les unes les autres.

6.2 Bilan Personnel de Naïm

En plus d'avoir beaucoup progressé en programmation, Naïm a acquis d'autres nouvelles compétences très utiles grâce à ce projet. Par exemple l'utilisation de \LaTeX pour la rédaction de rapports, la gestion d'un travail de groupe en se familiarisant notamment avec GitHub, ainsi que la méthodologie pour aborder une telle tâche. L'enjeu de devoir maintenir le code clair et concis pour les autres et le fait de devoir expliquer chaque algorithme mis en place l'ont poussé hors de sa zone de confort. Il se sent ainsi prêt à prendre part à des efforts collaboratifs d'une plus grande envergure, grâce à de nouvelles méthodes et compétences, et à un niveau en informatique plus solide.

6.3 Conclusion

Le bilan de ce projet est très positif pour les deux membres du groupe, nous sommes satisfaits de notre travail. Nous avons énormément gagné en maîtrise sur le langage C et la programmation en général, en développant de nombreuses compétences qui nous serviront à l'avenir. Nos erreurs nous ont beaucoup apporté, nous avons réalisé que c'est en nous trompant que nous avons le plus appris en informatique, et nous nous sommes beaucoup trompés l'un comme l'autre pendant ce projet. Ces deux mois nous ont également fait gagner en efficacité dans notre répartition du travail et lors de la gestion d'imprévus et de blocages majeurs, et se sont donc avérés très formateurs.

6.3.1 Perspectives

Le programme final est assez proche de la vision que nous en avons au départ, avec quelques différences, comme celle de la méthode de réduction de la taille de l'image expliquée plus tôt. Si nous devons expandre notre travail à partir de ce programme, nous rajouterions des fonctions de manipulation d'image autres que l'affichage ASCII. Par exemple, nous pourrions réécrire une copie de l'image originale en noir et blanc, négatif, ou à l'envers. Une fonction bonus est disponible dans notre programme qui crée une image en noir et blanc à partir de l'image fournie. Nous ne l'avons pas mit en avant car elle n'entre pas de le cadre principal de ce projet et ne fonctionne pas parfaitement. Nous l'avons écrit et ajouté au code tardivement alors qu'elle n'était pas finie, nous n'avons pas réussi à réécrire les octets de padding, ce qui crée un décalage entre les lignes. Cette fonction représente la direction que nous aimerions poursuivre avec ce programme et c'est la raison pour laquelle nous l'avons inclus.

7 Annexe

A Webographie

Comprendre le format Bitmap : https://en.wikipedia.org/wiki/BMP_file_format#:~:text=The%20BMP%20file%20format%2C%20also,and%20S%2F%20operating%20systems.

La fonction fread : https://www.tutorialspoint.com/c_standard_library/c_function_fread.htm

<https://www.educative.io/edpresso/c-reading-data-from-a-file-using-f>

La fonction fseek : https://www.tutorialspoint.com/c_standard_library/c_function_fseek.htm

Allocation dynamique d'une matrice : <https://youtu.be/aR7tkVj3UU0>
Choix des caractères à utiliser : <http://paulbourke.net/dataformats/asciiart/>

B Manuel utilisateur

B.1 Créer une image Bitmap

Ce programme fonctionne exclusivement avec les images Bitmap avec une profondeur de couleur de 24. Plusieurs images de ce type sont déjà fournies dans le dossier du programme, mais si vous pouvez en ajouter facilement.

Il vous suffit d'ouvrir votre image avec un programme comme Paint ou Photoshop, et de l'enregistrer dans le bon format. Sur paint par exemple, il s'agit du format "Bitmap 24 bits".

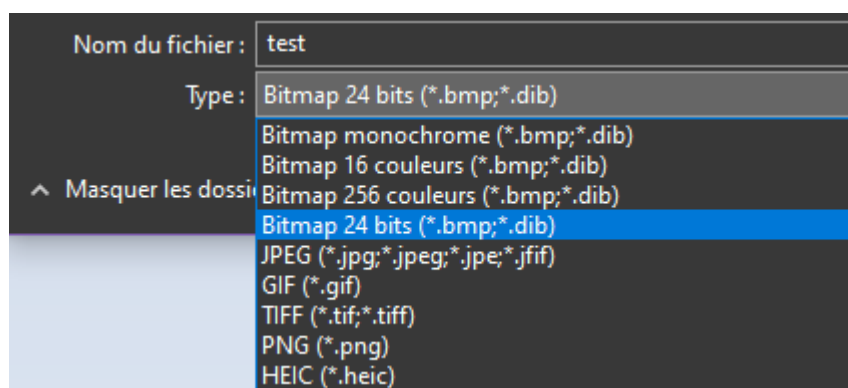


FIGURE 19 – Enregistrer une image au bon format avec Paint

D'autres logiciels comme Photoshop ne vous proposeront pas ces différents formats. Choisissez alors "BMP", puis sélectionnez la bonne profondeur de couleur sur l'écran suivant.

Certains programmes inversent l'ordre des lignes en donnant une hauteur négative à l'image, afin que la ligne 1 soit placée en haut de l'image et pas en bas comme c'est habituellement le cas. Pour éviter ce problème nous vous recommandons donc les logiciels Paint et Photoshop que nous avons testé.

B.2 Mode d'emploi

Une fois votre image Bitmap placée dans le dossier du programme, vous pouvez exécuter ce dernier. Au menu principal, plusieurs options vous sont proposées. Pour choisir, tapez le chiffre correspondant et appuyez sur "entrée". La troisième option permet de quitter le programme, nous allons expliquer le fonctionnement des deux premières dans la partie suivante.

```
Naim Chefirat et Gauvain Crevot

****MENU PRINCIPAL****

~Que voulez-vous faire?~

1: Affichage ascii
2: Autres fonctions
3: Quitter
```

FIGURE 20 – Menu principal du programme

B.2.1 Fonctions d’affichage ASCII

Pour afficher une image en caractères, tapez 1. Tapez ensuite le nom de votre image suivi de l’extension de fichier “.bmp”.

Il vous sera alors demandé de choisir le nombre de caractères à utiliser pour dessiner chaque pixel. Dans la plupart des cas, l’option recommandée est 8, mais les autres options sont accessibles si vous voulez expérimenter. Ces options peuvent s’avérer utiles si votre image est sombre et manque de contraste, comme celle de la figure 2.3.

Vous allez ensuite pouvoir réduire les dimensions de l’image d’abord en hauteur, puis en largeur. Vous êtes libres de choisir n’importe quelle valeur mais il n’est pas recommandé d’aller au dessus de 6, car la perte de détails sera trop grande. Le meilleur moyen de ne perdre aucun détail tout en pouvant voir l’image en entier et de rétrécir la police de la fenêtre du terminal avec les touches ”control”+”-” ou ”control”+molette vers le bas. La police plus petite vous permettra de voir toute l’image à l’écran sans avoir à la réduire excessivement. Il est cependant recommandé de réduire la hauteur d’avantage que la largeur pour un résultat non étiré. Par exemple, pour une image de dimensions 200*200, il est recommandé de réduire la hauteur à 1 pixel sur 2 et de laisser la largeur telle quelle. Si le terminal a été suffisamment dézoomé, l’image sera complètement visible. Pour les images de dimensions 400*400 et plus, la taille de la police doit être extrêmement réduite pour afficher l’image sans problèmes. Ce programme peut donc fonctionner avec les images de plus de 1500*1500 pixels et donner un résultat satisfaisant, mais il faudra réduire la taille de la police à un stade où le texte ne sera plus lisible. Pour une expérience optimale, utilisez des images dont la taille est inférieure à 800*800 pixels. Ainsi, vous obtiendrez un rendu satisfaisant avec une réduction de 3 en largeur et 2 en hauteur. Notez que ces paramètres sont propres à votre machine et sa résolution d’écran. Nous vous recommandons donc d’ex-

périmenter avec différentes tailles de police et paramètres de réduction pour trouver ce qui fonctionne le mieux.



FIGURE 21 – Utilisation du programme sur une très grande image(1600*1163 pixels) en dézoomant le terminal. Paramètres utilisés : Caractères : 8, Réduction de la hauteur : 3, Réduction de la largeur : 2

B.2.2 Autres fonctions

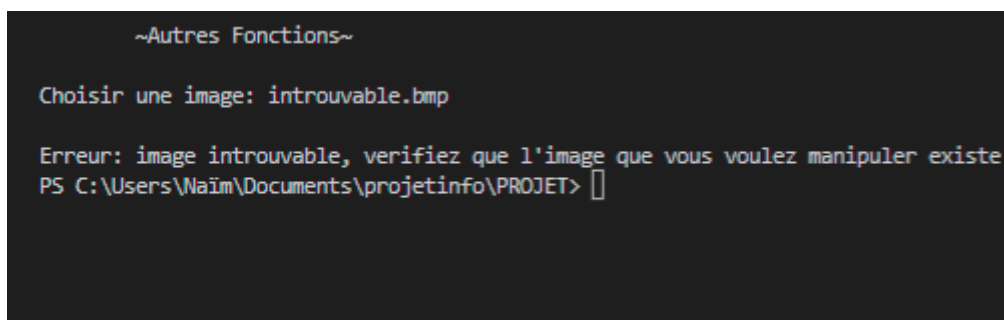
En choisissant l'option 2 du menu principal, vous pourrez accéder à des fonctions plus simples d'usage, qui nous ont aidé lors de la conception de ce programme. Ce sont d'anciennes fonctions qui ont servi à tester et déboguer notre code. La première affiche à l'écran des informations utiles sur l'image comme ses dimensions, le nombre de bits par pixel ou son nombre d'octets de padding. Elle peut s'avérer utile si vous avez besoin de connaître la taille d'une image avant de l'afficher en caractères ASCII.

Les deux suivantes vous seront moins utiles, elles affichent à l'écran les valeurs de tous les pixels de l'image. La première affiche la matrice associée à l'image, tandis que la seconde affiche les valeurs de chaque pixel directement après la lecture. Il n'est pas recommandé d'utiliser ces deux fonctions avec de grandes images car le résultat peut être assez long.

B.3 Erreurs courantes

Image introuvable

Lorsque cette erreur apparaît, comme le message l'indique, le programme n'a pas réussi à ouvrir votre image. C'est certainement dû au fait que l'image que vous voulez ouvrir n'est pas placée dans le dossier du programme, qu'elle n'existe pas, que vous avez mal saisi son nom, ou que le format d'image n'est pas supporté.

A screenshot of a terminal window with a dark background. The text is displayed in a monospaced font with syntax highlighting. The first line is a prompt '~Autres Fonctions~'. The second line shows a command 'Choisir une image: introuvable.bmp'. The third line is an error message: 'Erreur: image introuvable, verifiez que l'image que vous voulez manipuler existe'. The fourth line shows the command prompt 'PS C:\Users\Naïm\Documents\projetinfo\PROJET>' followed by a cursor.

```
~Autres Fonctions~  
Choisir une image: introuvable.bmp  
Erreur: image introuvable, verifiez que l'image que vous voulez manipuler existe  
PS C:\Users\Naïm\Documents\projetinfo\PROJET> █
```

FIGURE 22 – Message d'erreur lorsque le programme n'arrive pas à lire l'image

Boucle infinie

Si l'utilisateur saisit un caractère ou une série de caractères lorsqu'on lui demande de saisir un nombre, le programme va afficher sans cesse le message d'erreur habituel, sans demander à l'utilisateur de saisir une autre valeur. La seule solution est de fermer le terminal, ou d'appuyer sur "control"+"C" pour stopper de force le programme.

Image trop large

Quand l'image est trop large pour être affichée en une seule ligne dans le terminal, chaque ligne de l'image sera représentée sur 2 lignes de la console. Pour ne pas que cela arrive, vous pouvez réduire la taille de la police de votre console, et réduire la largeur de l'image.