DE ROBERT DE BOUSQUET Emilien 1AB2
HAEN Rémi 1AB2

SAÉ S1.01 - Implémentation d'un besoin client

La demande client initiale consiste à implémenter un programme permettant de retranscrire un fichier .PGM en Ascii Art dans la console.

Première Version

Dans un premier temps, nous devons procéder à l'ouverture de notre fichier .PGM, dans cette première version, le nom du fichier est renseigné dans le code source mais ultérieurement il sera possible à l'utilisateur de spécifier le fichier qu'il souhaite transcrire en Ascii Art..

```
//Ouverture du fichier "img1.pgm" en mode binaire :
std::ifstream fichier("img1.pgm", std::ios_base::binary);
```

Les fichiers .PGM sont assez simple pour une retranscription en Ascii Art étant donné qu'il ne contient que 4 lignes de mise en contexte dans lesquelles on nous venons récupérer des information dans la première version de notre programme, nous affichons ses 4 lignes afin de contrôler que les informations prélevées soient les bonnes:

- Ligne 1: nous vérifions que la version est bien P5 pour éviter les erreur que l'utilisation d'une mauvaise version pourrait entraîner

```
// Première ligne
std::string inputLine;
std::getline(fichier, inputLine);
if (inputLine.compare("P5") != 0) std::cerr << "Version error";
else std::cout << "Version : " << inputLine << "\n";
```

- Ligne 2: il s'agit d'un commentaire, nous l'affichons ici mais dans une version ultérieur le commentaire est uniquement récupérer pour pouvoir passer à la ligne suivante.

```
// Deuxième ligne : commentaire
std::getline(fichier, inputLine);
std::cout << "Comment : " << inputLine << "\n";
```

- Ligne 3: il s'agit d'une ligne contenant la largeur et la hauteur de notre image, ces informations nous seront utiles ultérieurement, nous les mettons dans deux variables distinctes et les transformons en valeurs.

```
//Troisième ligne : longueur et largeuer de l'image
//On transforme la chaine de caractère en tableau, chaques éléments du tableau sont les mots séparé par un espace
std::getline(fichier, inputLine);
std::stringstream ss(inputLine);
std::string element;
std::vector<std::string> elements;
while (std::getline(ss, element, ' ')) {
    elements.push_back(element);
}
//On transformes les valeurs du tableaux en int pour s'en servir comme dimention du tableau
int numline = std::stoi(elements[0]);
int numcols = std::stoi(elements[1]);
std::cout << numline << " " << numcols << "\n";
```

- Ligne 4: il s'agit de la valeur 255, nous n'en avons pas l'utilité dans ce programme, nous affichons seulement pour vérifier sa présence autrement nous passons aux pixels de l'image.

Nous passons donc à la lecture de l'image ligne par ligne. Pour cela nous créons un tableau lui-même constitué de tableaux de caractères, Ce tableau sera appelé Table.
De plus, nous créons un tableau de caractère appelé donnes qui correspond à une ligne de l'image que nous allons par la suite ajouter à Table pour former l'image.

```
//Début du programme V1
std::vector<std::vector<char>>Table;
std::vector<char> donnes(numline);
//Récupération lignes par lignes puis ajout dans le tableau Table qui les stockes toutes
for (size_t i = 0; i < numcols; i++)
{
    fichier.read(donnes.data(), numline);
    Table.push_back(donnes);
}
```

Finalement, pour l'affichage nous créons une double boucle qui parcours chaques pixels de l'image et les affiches après qu'ils ai étants converti en caractère Ascii par la fonction mapPixelToChar

```
for (size_t i = 0; i < Table.size(); i++)
{
    for (size_t j = 0; j < Table[i].size(); j++)
    {
        std::cout << mapPixelToChar(Table[i][j]);
    }
    std::cout << "\n";
}
```

La fonction mapPixelToChar situé dans le fichier mapPixelToChar.cpp prend en compte un pixel de l'image et en fonction de sa valeur (comprise en 0 et 255 étant donné que les caractères sont compris entre codés sur un Octet) renvoie un caractère correspondant a une nuance de gris plus ou moins importante.
Cette fonction est amenée à changer dans le futur si on veut que l'utilisateur puisse choisir sa propre palette de caractères.

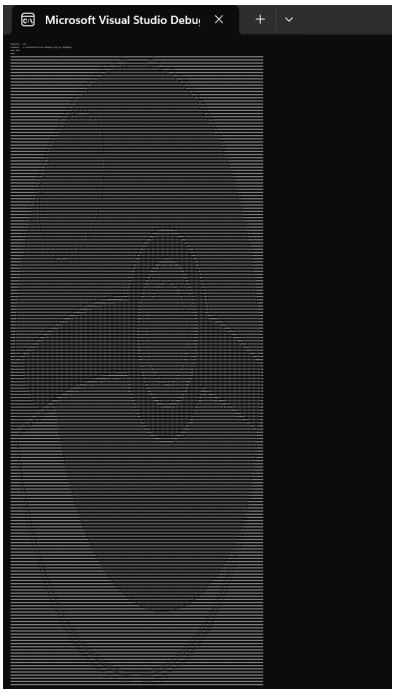
```
char mapPixelToChar(int pixel) {
    //Dans le cas où des valeurs serai inférieur
    //On à 8 nuances toutes les 32 valeurs
    if (pixel < 0) {
        pixel = pixel + 128;
    }
    if (pixel >= 0 && pixel < 32)
        return 'W';
    else if (pixel >= 32 && pixel < 64)
        return 'w';
    else if (pixel >= 64 && pixel < 96)
        return 'l';
    else if (pixel >= 96 && pixel < 128)
        return 'i';
    else if (pixel >= 128 && pixel < 160)
        return ';';
    else if (pixel >= 160 && pixel < 192)
        return ',';
    else if (pixel >= 192 && pixel < 224)
        return '.';
    else if (pixel >= 224 && pixel < 255)
        return ' ';
}
```

Nos bibliothèques et notre fonction sont déclaré dans le fichier d'entête mapPixelToChar.h

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <array>
#ifdef _WIN32
#include <Windows.h>
#endif // _WIN32

char mapPixelToChar(int pixel);
```

Résultat de sortie de la Version 1 et son image de référence :



Seconde Version

Notre première version ayant satisfait le client, nous avons débuté sur la base de la version 1 une seconde version dans laquelle il serait possible d'enregistrer l'Ascii Art dans un fichier texte.

Les changement se font principalement dans la partie affichage du programme, ici il n'est plus question de simplement afficher dans la console notre Ascii Art, a la place nous demandons à l'utilisateur de saisir le nom du fichier de sortie qu'il souhaite a l'aide de la fonction AskFileName puis nous ajoutons pixel par pixel le caractère correspondant dans le fichier texte.

```
std::string filename = AskFileName();
std::ofstream output(filename);
for (size_t i = 0; i < Table.size(); i++)
{
    for (size_t j = 0; j < Table[i].size(); j++)
    {
        char pixel = mapPixelToChar(Table[i][j]);
        output << pixel;
    }
    output << std::endl;
}
```

La fonction AskFileName est déclarée et définie dans un nouveau couple de fichier Console.cpp/Console.h. Dans de futur versions, nous ajoutons d'autres fonctions à ce couple de fichiers.

```
#include "PixelToChar.h"

std::string AskFileName();
```

```
#include "Console.h"

//On demande à l'utilisateur de renseigner le nom qui souhaite pour le fichier
std::string AskFileName() {
    std::string filename;
    std::cout << "Veuillez renseigner le nom du fichier que vous souhaitez creer : ";
    std::cin >> filename;
    std::cout << std::endl;
    std::cout << "Votre fichier est en cours de creation\n";
    //on prend le soint d'ajouter l'extention pour le confort d'utilisation
    filename = filename + ".txt";
    return filename;
}
```

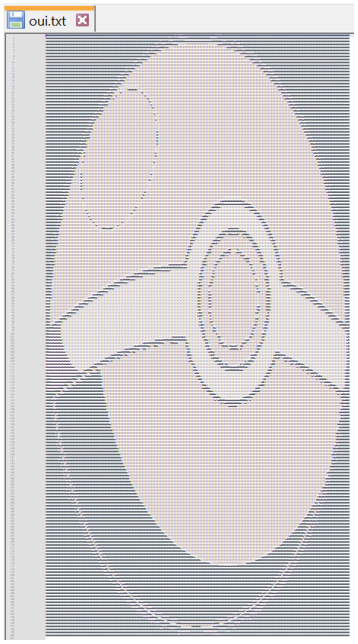
La fonction elle, demande a l'utilisateur de renseigner le nom souhaité pour son fichier et renvoie ce nom avec l'extension .txt

Le résultat:

```
Veuillez renseigner le nom du fichier que vous souhaitez creer : oui

Votre fichier est en cours de creation
```

je demande à mon programme d'appeler mon fichier oui.txt et le fichier est correctement créé,de plus il contient bien l'ascii art de l'image d'origine



Troisième Version

Le client, satisfait par notre vitesse d'exécution, souhaite pour une V3 que nous apportions la possibilité de changer de palette de caractères par le choix d'un fichier de palette afin que l'utilisateur puisse choisir ses caractères.

Pour cela nous allons remplacer la fonction mapPixelToChar par la fonction PixelToStr qui prend en entré le pixel et la palette (qui aura été définie plus haut dans le programme à l'aide de la fonction SetNewPalette qui prend en compte une autre fonction appelé AskPaletteName.

```
pal palette = SetNewPalette(AskPaletteName());
```

```
for (size_t i = 0; i < Table.size(); i++)
{
    for (size_t j = 0; j < Table[i].size(); j++)
    {
        std::string pixel = PixelToStr(Table[i][j], palette);
        output << pixel;
    }
    output << std::endl;
}
```

PixelToStr va calculer les valeurs intervalles dans lesquelles on choisira un caractère
Par exemple: si notre palette contient 8 caractères, alors nous aurons un caractère toutes les 32 valeurs car 256 / 8 = 32 cependant si la palette contient 5 caractères, il y aura un changement toutes les 51 valeurs.

```
std::string PixelToStr(int pixel, pal palette) {
    int step = 256 / palette.size();
    for (int i = 0; i < palette.size(); i++) {
        if (pixel < 0) {
            pixel = pixel + 128;
        }
        if (pixel >= i * step && pixel < (i + 1) * step) {
            return palette[i];
        }
    }
}
```

La fonction SetNewPalette prend en paramètre le nom de la palette souhaiter et renvoie un tableau de strings, une palette étant un fichier texte dont chaque lignes est une nuance de gris du plus sombre au plus clair, nous ajoutons le contenu de chaque lignes du fichier texte au tableau des nuance de gris, chaque ligne sera une nuance de gris différente à traiter avec PixelTostr

```
using pal = std::vector<std::string>;
```

```
pal SetNewPalette(std::string paletteName) {
    std::ifstream palettefile(paletteName);
    std::string line;
    pal newPalette{};
    while (std::getline(palettefile, line)) {
        if (line.empty()) {
            newPalette.push_back(" ");
        }
        else {
            newPalette.push_back(line);
        }
    }
    return newPalette;
}
```

Enfin la fonction AskPaletteName demande simplement à l'utilisateur de rentrer dans la console le nom de la palette qu'il souhaite utiliser

```
std::string AskPaletteName() {
    std::string palettename;
    std::cout << "Veuillez renseigner le nom de votre palette : ";
    std::cin >> palettename;
    std::cout << std::endl;
    return palettename + ".txt";
}
```

Quatrième Version

Après avoir fini cette version, notre client nous à demander de faire en sorte que la configuration du programme ne se fasse plus par la saisie des noms de fichiers dans la console en cours d'exécution du programme mais par paramétrage à l'aide d'arguments du programme.Nous nous sommes donc nous nous sommes attelés à la tâche pour produire une 4eme version de notre projet. Nous avons donc commencé par ajoeut en paramètre dans la fonction main, les arguments..

```
int main(int argc, char* argv[])
```

Nous avons ensuite créé un message d'aide affichant les options, ainsi que leur utilisation.

```
for (int i = 1; i <= argc; i++) {
    std::string args = std::string(argv[i]);
    std::cout << args;
    if (args == "--help") {
        std::string HelpMsg =
            "\nUsage :\n"
            "pgm2txt [options]\n\n"
            "Options :\n"
            "--input fichier     Spécifie le fichier image à convertir\n"
            "                    Si ce paramètre n'est pas spécifié, le fichier est demandé via la console.\n"
            "--output fichier    Spécifie le nom du fichier texte qui contiendra l'Ascii Art.\n"
            "                    Si ce paramètre n'est pas spécifié, l'Ascii Art est affiché dans la console.\n"
            "--palettefichier    Spécifie un fichier texte contenant la palette de couleur Ascii.\n"
            "                    Chaque ligne du fichier contient un caractère en UTF-8, du plus sombre au plus clair.\n"
            "                    Si ce paramètre n'est pas spécifié, la palette par défaut est \n "
            "                     \"W\",\"w\",\"l\",\"i\",\":\",\",\",\".\",\" \".\n"
            "--help              Affiche cette aide.";
        std::cout << HelpMsg;
        return 0;
    }

    else if (args =="--input") {
        image = std::string(argv[i + 1]);
        std::cout << "Image : "<<image << std::endl;
    }
    else if (args == "--output") {
        filename = std::string(argv[i + 1]);
        std::cout << "Filename : " <<filename << std::endl;
    }
    else if (args == "--palette") {
        palettefile = std::string(argv[i + 1]);
        std::cout << "PaletteFile : " << palettefile << std::endl;
    }
}
```

Cependant, nous nous sommes heurtés à un problème. Tandis que le message d'aide s'affichait correctement lors de la saisie de la commande, les autres options elles n'affichent rien lorsque nous saisissons les commandes. Les entrées étaient reconnues cependant par le programme, mais elles n'étaient pas traitées.

Finalement, la 4ème version n'étant pas abouti, nous avons remis la troisième version du projet, celle-ci étant parfaitement fonctionnelle. Nous espérons pouvoir collaborer avec ce client à nouveau.