

RAPPORT DU PROJET DE PROGRAMMATION IMPÉRATIVE

IMPLÉMENTATION D'UN INTERPRÉTEUR POUR UN LANGAGE GRAPHIQUE

Janvier 2024

Alexis COMSIS

Encadré par :
Guillaume BUREL



Table des matières

1	Introduction	2
2	Fonctionnement du programme	3
2.1	Les variables, structures et données	3
2.1.1	Les pixels	3
2.1.2	Les couleurs	3
2.1.3	Les directions et les bords	3
2.1.4	Les piles dynamiques	3
2.2	Recherche du prochain pixel	4
2.2.1	Calcul du bloc du pixel actuel	4
2.2.2	Recherche de la frontière	4
2.3	Différence des couleurs et des luminosités	4
2.3.1	Différence des couleurs	4
2.3.2	Différence des luminosités	5
2.4	Actions	5
3	Les limites	5
3.1	Cas limite : euclide.ppm et power.ppm	5
3.2	Étape bloquante : pietquest.ppm	6
4	Conclusion	7

1 Introduction

Le but du projet est de réaliser un interpréteur pour un langage graphique. Le programme prend un fichier ppm en entrée et exécute une suite d'instructions qui dépend de l'image d'entrée.

Un manuel README.md est disponible dans le dossier du projet. Il explique comment exécuter le programme et notamment comment le lancer en mode DEBUG (bonus).

L'interpréteur commence sur le premier pixel (en haut à gauche) de l'image, avec comme direction Est et comme bord Bâbord. l'interpréteur va alors se balader sur les pixels de l'image et va éventuellement changer de direction (Est, Sud, Ouest, Nord) ou de bord (Bâbord, tribord) en fonction de la couleur et du nombre de pixels qu'il va rencontrer. À chaque itération, l'interpréteur effectuera éventuellement des actions sur une pile d'entiers qui est initialisée au lancement de ce dernier. (cf. sujet pour la liste des actions)

Chaque itération se déroule en deux étapes :

- Trouver le prochain pixel sur lequel se déplacer
- Effectuer ou non une action sur la pile d'entiers

Le programme itère jusqu'à ce que le curseur soit confronté à 8 pixels bloquant d'affilée

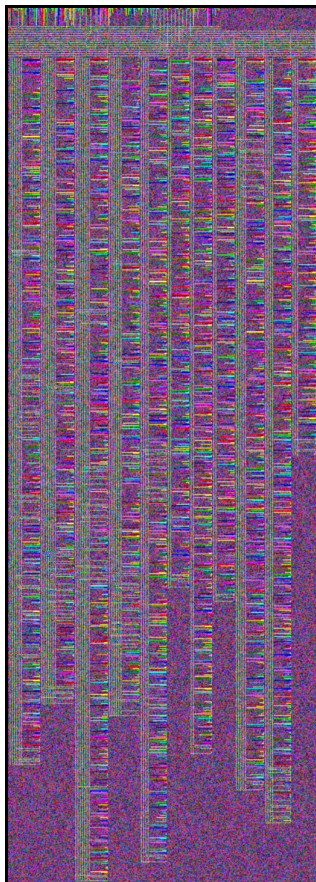


FIGURE 1 – pietquest.ppm

```
bigboug@Mars ~/projet (master*) $ make
gcc -Wall -Wextra -g -fsanitize=address -c -o main.o main.c
gcc -Wall -Wextra -g -fsanitize=address -c -o stack.o stack.c
gcc -Wall -Wextra -g -fsanitize=address -c -o extract.o extract.c
gcc -Wall -Wextra -g -fsanitize=address -c -o block.o block.c
gcc -Wall -Wextra -g -fsanitize=address -c -o color.o color.c
gcc -Wall -Wextra -g -fsanitize=address -o prog main.o stack.o extract.o
block.o color.o
bigboug@Mars ~/projet (master*) $ ./prog fichier_ppm/pietquest.ppm
=====
= Piet's Quest =
=====

You find yourself in a rather dark studio.
There is an easel here.
There is a ladder leading down.

Please select:

1 - paint
2 - go down ladder
2

You find yourself in a dusty, dim hallway.
There is a door to the kitchen here.
There is a door to the bedroom here.
There is a rickety loft ladder here.

Where do you want to go today?
1 - kitchen
2 - bedroom
3 - loft

```

FIGURE 2 – Exécution du programme pour pietquest.ppm

2 Fonctionnement du programme

2.1 Les variables, structures et données

2.1.1 Les pixels

L'interpréteur utilise des tableaux d'unsigned char de taille 3 afin de représenter les pixels. Car les unsigned char permettent de stocker des valeurs de 0 à 255 (inclu).

```
typedef unsigned char pixel[3]; //RED(0) GREEN(1) BLUE(2)
```

L'image est stockée dans un tableau 2D de pixels afin de faciliter la manipulation des pixels et les déplacements sur l'image.

Dans la suite du document "le pixel actuel" qualifiera un pixel quelconque sur lequel se trouve le curseur lors d'un pas du programme.

2.1.2 Les couleurs

Il est nécessaire de déterminer les couleurs de certains pixels (codants) pour calculer leur écart dans le cycle chromatique décrit dans le sujet. Il est également nécessaire de déterminer leur luminosité. Pour faciliter la détermination et les comparaisons des couleurs et des luminosités, elles sont définies grâce à des types enregistrements.

```
enum Brightness { //BRIGHT(0) -> MEDIUM(1) -> DARK(2) -> BRIGHT(0)
    BRIGHT, MEDIUM, DARK
};
enum Color { //RED(0) -> YELLOW(1) -> GREEN(2) -> CYAN(3) -> BLUE(4) -> MAGENTA(5) -> RED(0)
    RED, YELLOW, GREEN, CYAN, BLUE, MAGENTA
};
```

2.1.3 Les directions et les bords

De même que pour les couleurs, des types enregistrements sont utilisés pour faciliter l'implémentation des déplacements de l'interpréteur sur l'image.

```
//0 for babord 1 for tribord
enum Bord {
    BABORD, TRIBORD
};
//0 east 1 south 2 west 3 north
enum Dir {
    EAST, SOUTH, WEST, NORTH
};
```

2.1.4 Les piles dynamiques

L'ensembles des "actions" (comme par exemple "affichage de caractère" ou "changement de direction") que l'interpréteur va effectuer se fera à l'aide de la pile d'entiers.

```

struct stack {
    int* tab;
    int current_size;
};

```

2.2 Recherche du prochain pixel

À chaque itération, l'interpréteur cherche le prochain pixel sur lequel il placera son curseur.

2.2.1 Calcul du bloc du pixel actuel

Afin de trouver le pixel suivant, nous avons besoin de considérer le bloc de pixels de même couleurs et adjacents au pixel actuel. Pour cela, le programme utilise une fonction récursive (exploration) qui permet de lister l'ensemble des coordonnées des pixels appartenant à ce bloc. L'ensemble des coordonnées selon x est listé dans `block_coords_x` et selon y dans `block_coords_y` (dans l'ordre afin de pouvoir retrouver les pixels à partir de ces tableaux lors de la recherche de frontière.)

En partant du pixel actuel, l'algorithme suivi par exploration est le suivant.

Pour chaque pixel adjacent au pixel actuel qui n'est pas déjà testé :

- On marque le pixel comme testé
- On stock ses coordonnées dans `block_coords_x` et `block_coords_y`
- On appelle exploration sur le pixel

2.2.2 Recherche de la frontière

Dans la liste des pixels appartenant au bloc, on effectue maintenant des tests pour déterminer lequel est le plus au Nord, au Sud, à l'Ouest, à l'Est, à babôrd et à tribord en fonction de la direction et du bord du pixel actuel. L'interpréteur appelle alors la fonction "frontier" (correspondant à la direction du pixel.)

2.3 Différence des couleurs et des luminosités

Il est nécessaire de déterminer les différences de couleur et de luminosité pour savoir quelle action le programme va effectuer sur la pile d'entiers.

#FF8080 rouge clair	#FFFF80 jaune clair	#80FF80 vert clair	#80FFFF cyan clair	#8080FF bleu clair	#FF80FF magenta clair	BRIGHT
#FF0000 rouge	#FFFF00 jaune	#00FF00 vert	#00FFFF cyan	#0000FF bleu	#FF00FF magenta	MEDIUM
#800000 rouge foncé	#808000 jaune foncé	#008000 vert foncé	#008080 cyan foncé	#000080 bleu foncé	#800080 magenta foncé	DARK
RED	YELLOW	GREEN	CYAN	BLUE	MAGENTA	

FIGURE 3 – Code hexadécimal des couleurs codantes avec leurs couleurs et brillances correspondantes.

2.3.1 Différence des couleurs

On note des caractéristiques communes aux pixels d'une unique couleur codante. par exemple seuls les pixel RED et MAGENTA ont leurs composante rouge strictement supérieur à leurs composante verte.

La fonction `pixel_color` utilise ces caractéristiques afin de déterminer la couleur en minimisant le nombre de tests. `pixel_diff_color` donne ensuite la différence dans le cycle des couleurs.

2.3.2 Différence des luminosités

De même, il existe des caractéristiques communes pour les luminosités. Les pixels `BRIGHT` sont les seuls à n'avoir aucune composante nulle. La fonction `pixel_brightness` détermine les différentes luminosités. `pixel_diff_brightness` donne ensuite la différence dans le cycle des luminosités.

2.4 Actions

Une fois que l'interpréteur a calculé la différence de luminosité et de couleur il peut alors effectuer l'action correspondante en appelant "action". Mais avant de faire cela, il vérifie que le pixel duquel on vient est bien un pixel codant car il se peut que le pixel duquel on vient soit passant. (Cela se produit notamment lorsque le curseur rencontre un pixel bloquant après avoir rencontré un pixel passant.)

3 Les limites

3.1 Cas limite : euclide.ppm et power.ppm

Il semble qu'il ne soit pas possible d'exécuter correctement `euclide.ppm` et `power.ppm` avec le même programme. En effet, il n'est pas précisé dans le sujet si la fonction "plus grand" des actions retourne 1 lorsque le second élément dépilé est strictement plus grand que le premier ou si elle retourne 1 lorsque qu'il est simplement plus grand.

Si on choisit une inégalité stricte, alors `power.ppm` s'exécute correctement mais `euclide.ppm` ne fonctionne pas pour des entiers premiers entre eux (mais fonctionne pour deux entiers non premier entre eux).



```
bigboug@Mars:~/projet
bigboug@Mars ~/projet (master*) $ ./prog fichier_ppm/power.ppm
3
4
81
bigboug@Mars ~/projet (master*) $ ./prog fichier_ppm/euclide.ppm
13
12
12
bigboug@Mars ~/projet (master*) $ ./prog fichier_ppm/euclide.ppm
15
20
5
```

FIGURE 4 – Test si inégalité dans "plus grand" est stricte

Inversement, si on choisit de mettre une inégalité large, alors `power.ppm` itère une fois de trop lors de l'élévation à la puissance mais `euclide` fonctionne dans le cas de deux entiers premiers entre eux.

L'inégalité stricte a été choisie pour l'exécution des autres programmes.

```
bigboug@Mars ~/projet (master*) $ ./prog fichier_ppm/power.ppm
3
4
243
bigboug@Mars ~/projet (master*) $ ./prog fichier_ppm/euclide.ppm
13
12
1
bigboug@Mars ~/projet (master*) $ ./prog fichier_ppm/euclide.ppm
15
20
5
```

FIGURE 5 – Test si inégalité dans "plus grand" est large

3.2 Étape bloquante : pietquest.ppm

L'exécution du jeu pietquest.ppm se fait avec succès jusqu'au point où le joueur rentre dans la chambre (bedroom). La cause du problème n'a pas pu être identifiée. Après avoir discuté de l'exécution de pietquest avec des camarades, il semble que d'autres camarades soient bloqués au même niveau dans le jeu. Soit nous avons une erreur commune, soit il y a un problème dans le fichier ppm qui empêche la terminaison du jeu.

```
./prog fichier_ppm/pietquest.ppm

Your options:
1 - talk to her
2 - go back to the hallway
2

You find yourself in a dusty, dim hallway.
There is a door to the kitchen here.
There is a door to the bedroom here.
There is a rickety loft ladder here.

Where do you want to go today?
1 - kitchen
2 - bedroom
3 - loft
2

You find yourself in a cozy bedroom. The harsh sunlight exposes the u
de bed.
You feel a little guilty.
Your optio:
1 - return to the hallway

P1

You find yourself in a well-stocked kitchen.
It smells invitingly of apple pancake.
Your wife is here.
She gives you a look.

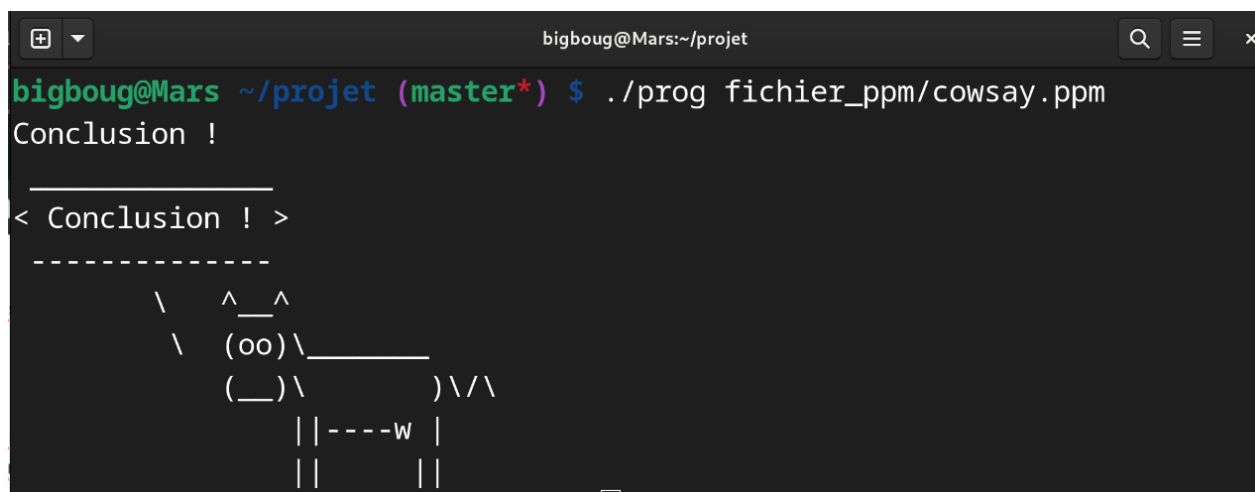
Your options:
1 - talk to her
2 - go back to the hallway
```

FIGURE 6 – Bug de pietquest.ppm

Excepté ces cas particuliers les programmes s'exécutent convenablement et fournissent les résultats attendus.

4 Conclusion

Le programme fonctionne raisonnablement avec tout les exemples fournis. Il pourrait néanmoins être optimisé pour renvoyer un résultat plus rapide dans le jeu pietquest.ppm. Un mode DEBUG est également disponible même s'il aurait été préférable d'en faire un graphique et j'ai un peu manqué de temps pour cela.

A screenshot of a terminal window with a dark background. The title bar at the top reads 'bigboug@Mars:~/projet'. The terminal shows a command prompt 'bigboug@Mars ~/projet (master*) \$' followed by the command './prog fichier_ppm/cowsay.ppm'. The output of the command is 'Conclusion !' followed by a ASCII art cow. The cow's body is a rectangle with the text '< Conclusion ! >' inside. The head is a semi-circle with two small circles for eyes, a line for a mouth, and a line for a nose. The legs are four vertical lines. The tail is a small triangle. The cow is facing right.

```
bigboug@Mars ~/projet (master*) $ ./prog fichier_ppm/cowsay.ppm
Conclusion !
_____
< Conclusion ! >
-----
      \   ^__^
       \  (oo)\_______
          (__)\       )\/\
              ||----w |
              ||     ||
```

FIGURE 7 – Exécution avec cowsay.ppm