



VECTOR TEXT-BASED EDITOR

Projet programmation en C



ASSOUAD Adrien

RAFALIMANANA—CHAN PENG Julien

Table des matières

Introduction	1
Présentation fonctionnelle	2
Présentation des résultats	6
Bilan du projet	10

I / Introduction (Description et objectifs)

Dans le cadre de ce projet nous allons nous intéresser à la réalisation des images vectorielles et plus particulièrement au fonctionnement des logiciels de dessin vectoriel. Le dessin vectoriel est très utilisé pour la création d'illustrations, de graphiques ou de cartes, car il offre à l'utilisateur la possibilité d'observer avec précision et sans perte de détails les zones qui l'intéressent en fonction de l'échelle souhaitée.

L'objectif de ce projet est de reproduire ce système de dessin vectoriel en programmation C en binôme. Ce projet va également nous permettre d'utiliser nos compétences en langage C et de les approfondir en utilisant différentes fonctionnalités dans le dessin vectoriel.

A travers ce rapport, nous allons vous faire :

- Une présentation fonctionnelle du code (les principales fonctionnalités, la description des principaux algorithmes réalisés, les difficultés techniques rencontrées et les solutions apportées) ;
- Une présentation des résultats (tests et résultats obtenus) ;
- Un bilan du projet (les enseignements retenus de la réalisation du projet, apprentissages sur le plan technique, apprentissages sur l'organisation du travail et la communication au sein de l'équipe, apprentissage sur la gestion du temps).

Selon le choix de l'action de l'utilisateur, une fonction différente sera appelée :

- **A : Ajouter une forme** : le programme affiche un autre menu proposant les différents types de formes proposées. L'utilisateur peut choisir le type de forme à ajouter. En fonction de la forme choisie, le programme demande à l'utilisateur de saisir les informations nécessaires pour créer la forme correspondante (coordonnées), puis ajoute la forme à une liste de formes stockées en mémoire.
- **B : Afficher la liste des formes** : le programme parcourt la liste des formes qui sont stockées en mémoire et affiche les informations de chaque forme à l'écran.
- **C : Supprimer une forme** : le programme demande à l'utilisateur de saisir l'identifiant (id) de la forme à supprimer et la supprime de la mémoire.
- **D : Tracer le dessin** : le programme va afficher les différentes formes choisies par l'utilisateur en fonction de leurs paramètres
- **E : Aide** : cette fonctionnalité n'a pas été mise en place dans le code fourni.

2) Formes

L'utilisateur a le choix entre plusieurs formes à tracer :

- Un point ;
- Une ligne ;
- Un carré ;
- Un rectangle ;
- Un cercle ;
- Un polygone.

Ces formes sont réalisées à l'aide de fonctions et de structure. Chaque forme a une structure propre à elle-même avec différents paramètres. Par exemple, le point de structure Point a deux composantes : son abscisse x et son ordonnée y.

3) Les fonctions

Le stockage en mémoire repose sur des pointeurs. Les fonctions de création (ex : « *create_point ») permettent l'allocation d'espace en mémoire pour chaque structure et elles renvoient un pointeur qui pointe vers cet espace. Les autres fonctions telles que « void delete_point », « void print_point », « void erase_area » permettent de réaliser des opérations de suppression et d'affichage des formes géométriques.

Les fonctions principales de ce projet sont celles qui :

- Créent l'espace de dessin ;
- Suppriment une zone dans l'espace de dessin ;
- Allouent / suppriment de l'espace en mémoire pour les formes choisies ;
- Allouent / suppriment de l'espace en mémoire pour les pixels des formes ;
- Affichent / suppriment les formes dans l'espace de dessin.

4) Les structures

Chaque forme a une structure (ex : structure Point). Les structures permettent à chaque forme d'avoir ses propres paramètres (structure Point, structure Line, structure Square, etc.). Cela est donc très pratique pour les formes avec différentes caractéristiques. De ce fait, on a créé des structures comprenant d'autres structures comme celle de Line puisqu'une ligne est composée de deux points.

Les structures principales sont :

- Les structures de chaque forme ;
- La structure Shape permet de convertir chaque forme au même format car on ne peut avoir qu'un seul type dans une liste (ex : une liste d'entiers) ;
- La structure Pixel permet de convertir les formes en pixels sur l'espace de dessin (area) ;
- La structure Area permet l'espace de dessin. Elle permet également de supprimer les formes incluses dans l'espace de dessin.

Difficultés rencontrées :

Les principales difficultés qu'on a rencontré lors de la réalisation de ce projet sont :


Les problèmes se sont surtout situés dans la deuxième partie. Les fonctions pixels étaient compliquées pour trois raisons :

- Bien comprendre d'où venaient les paramètres, leurs types et leur utilité dans la fonction
- Les problèmes d'allocation de mémoires surtout pour les formes comme circle, line et polygon où l'espace mémoire varie encore plus suivant si le nombre est pair ou non.
- Résoudre les bugs de ces fonctions.



Les solutions apportées :

- Bien comprendre les doubles et triples pointeurs et comment les utiliser.
- Pour les problèmes d'allocation de mémoire on a fait des boucles pour certaines fonctions qui permettent de compter le nombre de pixels à utiliser et donc calculer l'espace à allouer.
- Pour résoudre les bugs de ses fonctions il a fallu clarifier le code pour remonter facilement dans les fonctions qui étaient appelées par d'autres fonctions.

Afin de simplifier le code, nous avons divisé le code en plusieurs fichiers en différents modules ce qui permet d'alléger le programme.

 main.c

Le fichier « main.c » appelle le fichier « area.c » pour créer la zone de dessin. Il appelle également le fichier « commandes.c » pour que l'utilisateur puisse créer ses formes

 commandes.c
 commandes.h

Le fichier « commandes.c » appelle le fichier « shape.c » pour créer la shape de la forme et appelle aussi « area.c » pour ajouter les shapes des formes à la zone de dessin.

 pixels.c
 pixels.h



Le fichier « pixels.c » permet d'allouer de la mémoire et de créer les pixels pour les formes

 area.c
 area.h



Le fichier « area.c » va appeler le fichier « pixels.c » pour la fonction « draw_area ». Ce fichier permet de créer la zone de dessin, d'afficher / supprimer les formes.

 id.c
 id.h

Le fichier « id.c » sert à donner un identifiant pour les formes (cela sera utile quand il y aura plusieurs calques).

 shapes.c
 shapes.h

Le fichier « shape.c » va convertir toutes les formes au format Shape et leur alloue une mémoire. Pour cela, il appelle le fichier « formes.c »

 formes.c
 formes.h

Le fichier « formes.c » permet d'allouer de la mémoire pour les formes, ainsi que de les afficher / supprimer.

III / Présentation des résultats

Prenons par exemple la création d'un point de coordonnées (5, 8).

On crée d'abord l'espace de dessin qui est une matrice carrée grâce à « Area* create_area » qui va allouer de la mémoire pour la zone de dessin.

```
Area* create_area(unsigned int width, unsigned int height)
{
    Area* area = (Area*) malloc(sizeof(Area));
    area->width = width;
    area->height = height;
    area->mat = (B00L**) malloc(width * sizeof(B00L*)); // allouer de la mémoire pour la matrice des pixels
    for (int i = 0; i < height; i++)
    {
        area->mat[i] = (B00L**) malloc(height * sizeof(B00L*));
    }
    return area;
}
```

Ensuite, on va allouer de la mémoire pour la création du point avec « Point *create_point » et de la mémoire pour le pixel qui va être le point (« Pixel* create_pixel »). On lui crée également une shape avec « Shape *create_point_shape ».

```
Point *create_point(int px, int py)
{
    Point *p = malloc( Size: sizeof(Point));
    p->pos_x = px;
    p->pos_y = py;
    return p;
}
```

```
Pixel* create_pixel(int px, int py)
{
    Pixel* new_pixel = (Pixel*) malloc( Size: sizeof(Pixel));
    new_pixel->px = px;
    new_pixel->py = py;
    return new_pixel;
}
```

```
Shape *create_point_shape(int px, int py)
{
    Shape *shp = create_empty_shape( shape_type: POINT);
    Point *p = create_point(px, py);
    shp->ptrShape = p;
    shp->id = get_next_id();
    return shp;
}
```

Le point devient un pixel avec « void pixel_point ».

```
void pixel_point(Point* point, Pixel*** pixel, int* nb_pixels)
{
    *nb_pixels = 1;
    *pixel = (Pixel**) malloc ( Size: *nb_pixels*sizeof (Pixel*));
    (*pixel)[0] = create_pixel( px: point->pos_x, py: point->pos_y);
}
```

On ajoute la shape du point à l'espace de dessin (area) avec « void add_shape_to_area » :

```
void add_shape_to_area(Area* area, Shape* shape)
{
    area->shapes[area->nb_shape++] = shape;
}
```

Une fois cela, on dessine l'espace de dessin avec le point avec « void draw_area », « void print_area » et « void afficher_formes » :

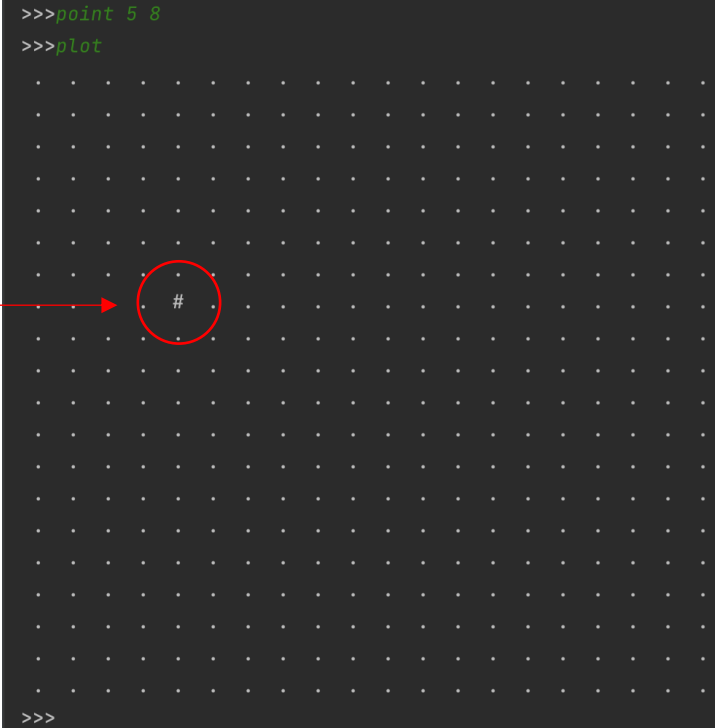
```
void draw_area(Area* area) {
    int nb;
    for (int i = 0; i < area->nb_shape; i++) {
        Pixel **list_pixel = create_shape_to_pixel(area->shapes[i], &nb);
        for (int j = 0; j < nb; j++) {
            Pixel *pixel = list_pixel[j];
            area->mat[pixel->py - 1][pixel->px - 1] = 1;
        }
    }
}
```

```
void afficher_formes(Area* area)
{
    for(int i = 0; i < area->nb_shape; i++)
    {
        printf( format: "%d\t", i+1);
        print_shape(area->shapes[i]);
        printf( format: "\n");
    }
}
```

```
void print_area(Area* area)
{
    for (int x = 0; x < area->width; x++)
    {
        for (int y = 0; y < area->height; y++)
        {
            if (area->mat[x][y]==0)
            {
                printf( format: " . "); // affiche un pixel vide
            }
            if (area->mat[x][y]==1)
            {
                printf( format: " # "); // affiche un pixel rempli
            }
        }
        printf( format: "\n"); // passe à la ligne suivante
    }
}
```

Résultat :

```
>>>point 5 8
>>>plot
```



```
>>>
```


Si l'on souhaite supprimer le point, on efface la zone de dessin et on libère la mémoire avec « void delete_area ». On peut également supprimer toutes les formes de la zone avec la fonction « erase_area »

```
void delete_area(Area* area)
{
    erase_area(area);
    for (int i = 0; i < area->width; i++)
    {
        free( Memory: area->mat[i]);
    }
    free( Memory: area->mat);
    free( Memory: area);
}
```

```
void erase_area(Area* area)
{
    for (int i = 0; i < area->nb_shape; i++)
    {
        free( Memory: area->shapes[i]);
    }
    area->nb_shape = 0;
}
```

On va ensuite supprimer le point, le pixel et la shape qui lui sont associés en libérant la mémoire (« void delete_pixel », « void delete_shape », « void delete_point »).


```
void delete_pixel(Pixel* pixel)
{
    free( Memory: pixel);
}
```

```
void delete_shape(Shape* shape) {
    switch (shape->shape_type) {
        case POINT:
            delete_point( point: shape->ptrShape);
            break;
    }
}
```

```
void delete_point(Point * point)
{
    free( Memory: point);
    point = NULL;
}
```

Résultat :

```
>>>list
1  POINT 5 8
>>>delete 1
>>>plot
```



```
>>>
```

Voici d'autres exemples de résultats obtenus :

```
>>> polygon 10 10 20 20 4 20
>>> plot
```

```
>>>|
```

POLYGON (10 10 20 20 4 20)


```
>>>rectangle 4 4 e 3  
>>>square 12 12 4  
>>>plot
```

```
>>>
```

RECTANGLE (4 4 6 3)

CARRE (12 12 4)

```
>>> line 2 4 7 9
>>> plot
```



```
>>>
```

LINE (2 4 7 9)

```
>>>circle 4 4 3
>>>plot
```

CERCLE (4 4 3)

IV / Bilan du projet

Ce projet a été très instructif malgré les nombreuses difficultés rencontrées au cours de la réalisation de ce dernier. Cela nous a permis d'utiliser nos compétences de programmation en C mais aussi de découvrir comment les utiliser intelligemment pour créer les fonctionnalités présentes dans ce code (ex : triple pointeur).

En ce qui concerne la répartition des tâches et de l'organisation du travail, nous avons trouvé le temps assez juste pour la quantité de travail à effectuer, notamment en raison des examens de fin d'année qui arrivent en même temps que la date de remise du projet. De plus, il y avait certaines erreurs dans les consignes du projet, ce qui nous a fait perdre un temps précieux. Cependant, nous avons découvert l'utilisation de Git et Git-Hub pour faciliter les partages de code lorsqu'on travaillait chacun de notre côté nous permettant d'avoir toujours le code à jour. Aussi, cela nous permettait de récupérer une version stable du jeu si nous avions des bugs dans une de nos nouvelles versions.

En globalité, nous sommes satisfaits du travail que nous avons fourni et nous pensons que ce genre de projet est un bon moyen d'apprentissage puisque cela demande de la communication et de l'entraide au sein de l'équipe.