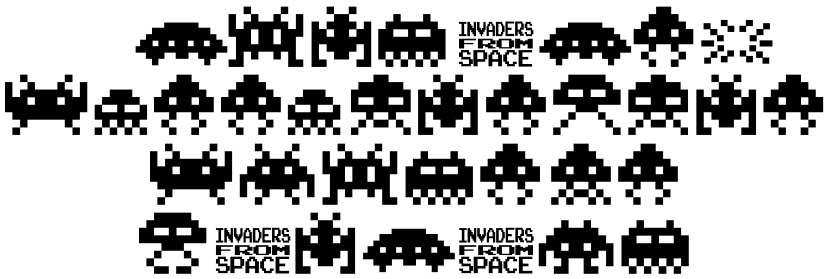


RAPPORT DE PROJET

Par Yuna LAURENCY et Carlos OKINDA

Introduction

a. Contextualisation du projet

i. Présentation du projet

Le projet consistait à la création d'un simulateur permettant de générer de manière grossière un donjon crawlers tel que ceux dans les jeux vidéo (Ex : Zelda ect…) ou encore les jeux de sociétés (Ex : Rumble in the House). Le langage utilisé était le « C ». Nous avons donc ainsi développé notre propre bibliothèque. Cette dernière nous as donc permis de d'ordonner nos idées et de prendre en compte chaque partie de la création des donjons (en passant par la création des salles, des éléments appartenant à ses dernières et par la conception des couloirs par exemple).

ii. En dehors du contexte académique

Lors de la présentation de ce projet nous nous pencherons sur les enseignements professionnels que nous avons tirés. Nous allons donc nous détacher de la nature académique du projet en donnant les raisons de nos choix ainsi que les limites de ces derniers.

b. Les contraintes, l'équipe et le mode de travail

iii. Les contraintes

- **Le langage :** le langage utilisé était imposé, il s'agissait du langage C. L'utilisation de ce langage nous a forcé à nous adapter à ses inconvénients. De l'usage de pointeurs (obligatoire dans la réalisation de notre projet) à la location de la mémoire ou encore sa libération.

- **Le temps :** le temps à notre disposition était également une contrainte importante. Entre le temps à notre disposition et l'ensemble des autres projets sur lesquels nous travaillions, il nous a fallu produire un programme à la fois fonctionnel en sachant que nous n'aurions pas forcément le temps d'implémenter l'ensemble de nos idées.

iv. L'équipe

L'équipe est composée de deux membres :

- **Yuna Laurency,**
- **Carlos Okinda**

v. Le mode de travail

Nous avons opté sur un travail composé de rendu journalier. A chaque modification et avancement nous créions des updates et jusqu'à ce jour nous sommes à la version « 0.7 » de notre programme soit la septième version en date. Pour éviter que nous travaillions sur les mêmes parties du projet nous avons divisés le travail en sous-travaux. Par exemple pour la création des salles, l'un de nous s'occupait des portes et de leurs positions et l'autres des objets présents dans les salles. Notre code à aussi été divisé en plusieurs sous-fichiers et bibliothèques permettant une meilleure lisibilité de ce dernier.

c. Présentation du plan du rapport

Dans ce rapport, nous allons présenter la réalisation du projet, à travers celle des salles (qui est très liés à celle du donjon en lui-même), des objets présents dans ces dernières et des couloirs. Nous avons aussi mis en évidence l'importance des projets que nous n'avons pas pu implémenter dans notre projet.

La réalisation du projet

a. Les salles

i. Le donjon

Lors de notre réflexion autour de ce projet nous avons optés pour plusieurs manières de créer un donjon. La première consistait à imprimer les caractères « # » sur une longueur et une largeur précise. Mais à la suite de ce choix nous avons malheureusement compris que ce n'étais pas la bonne manière de raisonner. On ne faisait qu'afficher des contours mais ces derniers n'avaient pas de place définie et donc on ne pouvait pas définir ni les contours ni les modifier.

Nous avons donc essayé une autre manière, celle de définir un type « Salle » permettant d'identifier une salle grâce à sa longueur et sa largeur et grâce à ces dernières de créer un tableau dynamique. Lorsque l'on va créer un donjon, on va donc allouer la mémoire d'une « Salle » précise et c'est dans cette grande salle que nous allons placés les petites salles qui vont composer le donjon. Pour un souci de distinction, nous avons opter pour la définition suivante pour un donjon : **Un donjon** est une **grande salle** dont **les entrées** sont situées uniquement au **nord** et **les sorties** uniquement au **sud**.

```c
/* TYPEDEF */

typedef struct s
{
    int H;           // Hauteur de la salle
    int L;           // Longeur de la salle
    char **uneSalle; // Tableau dynamique de la salle
} Salle;
```

ii. Les salles

Dans la continuité de la partie précédente, nous avons élaboré une manière de définir des portes à d'autres emplacements qu'au nord et au sud des salles. Lors de la réalisation de nos salles nous avons décidé de ne pas permettre à l'utilisateur de choisir le nombre de portes qu'il veut. En effet, nous avons mis une limite de portes afin qu'il ne puisse pas mettre les portes sur tous les murs.

```c
void insertion_portes(Salle *s, int n_salle) // Permet de placer un nombre de portes choisis
{
    char type;
    int nb;
    int n_max_portes = (((s->L * s->H) - 4) / 2);
    nb_portes(n_salle, n_max_portes);
    scanf("%d", &nb);
    while (nb > n_max_portes)
    {
        erreur();
        nb_portes(n_salle, n_max_portes);
        scanf("%d", &nb);
    }
    for (int i = 0; i < nb; i++)
    {
        type_porte(n_salle, i + 1);
        scanf(" %c", &type);
        while (type != 'F' && type != 'O')
        {
            erreur();
            type_porte(n_salle, i + 1);
            scanf(" %c", &type);
        }
        ajouter_porte_position(s, type, i + 1); // ajouter une porte à une position plus simple à comprendre
        afficher_salle(s);
    }
    system("cls");
    afficher_salle(s);
}
```

La fonction « ajouter_porte_position » permet de placer des portes plus facilement, avec des lettres **N pour le nord**, **S pour le sud**, **O pour l'ouest**, **E pour l'est**.

Pour nos salles nous avons également pensés à un moyen de placés des objets. Nous avons divisé les salles en 9 « cases » et c'est dans ses cases que l'utilisateur pourras inserer ses objets. Nous avons laissé le choix à l'utilisateur du type d'objets qu'il veut dans ses salles.

b. Le donjon

iii. L'insertion des salles dans le donjon

Pour nous, un donjon ne pouvait pas juste être défini par une grande salle vide. Il nous fallait donc un moyen d'inserer les salles dans le donjon initial. Nous avons opter pour deux fontions .une permettant de créer un nombre de salles prédéfinis et de les afficher et une seconde s'occupant excluisvement de les placés dans le donjon de bases. Cette méthode nous a donc permis d'assurer une liberté à l'utilisateur.

```c
void creation_salles(int *nbSalles)
{
    clear();
    Salle *donjon = creation_donjon();
    salle();
    nb_salles();
    scanf("%d", nbSalles);
    while (*nbSalles > 10 || *nbSalles < 1)
    {
        erreur();
        nb_salles();
        scanf("%d", nbSalles);
    }
    clear();
    Salle *sallesPrecedentes[*nbSalles];
    for (int i = 0; i < *nbSalles; i++)
    {
        Salle *s1 = creation_salle(i + 1);
        sallesPrecedentes[i] = s1;
    }
    clear();
    placement_salle();
    placer_salles(donjon, sallesPrecedentes, *nbSalles);
}
```

Pour avoir accès aux donjons nous avons aussi réfléchis à un moyen de pouvoir dupliquer les salles déjà fait. On peut avoir accès aux salles déjà créer afin de rendre plus rapide la conception de donjon. On utilise les lignes suivantes afin de créer une lystes dans laquelle se retrouve toute les salles que nous avons crée. Je tiens a préciser que lors de la création d'une nouvelle salle, si une salle existe déjà, l'ancienne salle est affichée à l'utilisateur.

« Salle*s1 = creation_salle(i+1) » et

« sallesPrecedentes[i] = s1 »

```
void placer_salles(Salle *donjon, Salle **sallesPrecedentes, int nbSalles)
{
    int nSalle = 1;
    int x, y;

    while (nSalle != 0)
    {

        for (int j = 0; j < nbSalles; j++)
        {
            printf("SALLE %d \n", j + 1);
            afficher_salle(sallesPrecedentes[j]);
        }

        choix_salle(nbSalles);
        scanf("%d", &nSalle);
        if (nSalle == 0)
        {
            break;
        }

        while (nSalle > nbSalles)
        {
            erreur();
            choix_salle(nbSalles);
            scanf("%d", &nSalle);
        }
```

Néanmoins, lors de l'insertion des salles sur le donjon, le programme écrase les données initialement présente à l'emplacement où l'utilisateur veut inserer. On peut donc se retrouver avec une salle qui est écraser par une autre salle.

```
        placer_salle_x(nSalle, donjon->L - (sallesPrecedentes[nSalle - 1]->L + 2));
        scanf("%d", &x);
        while (x < 2 || x > donjon->L - (sallesPrecedentes[nSalle - 1]->L + 2))
        {
            erreur();
            placer_salle_x(nSalle, donjon->L - (sallesPrecedentes[nSalle - 1]->L + 2));
            scanf("%d", &x);
        }

        placer_salle_y(nSalle, donjon->H - (sallesPrecedentes[nSalle - 1]->H + 2));
        scanf("%d", &y);
        while (y < 2 || y > donjon->H - (sallesPrecedentes[nSalle - 1]->H + 2))
        {
            erreur();
            placer_salle_y(nSalle, donjon->H - (sallesPrecedentes[nSalle - 1]->H + 2));
            scanf("%d", &y);
        }

        ajouter_salle(donjon, sallesPrecedentes[nSalle - 1], x, y);
        afficher_salle(donjon);
    }
}
```

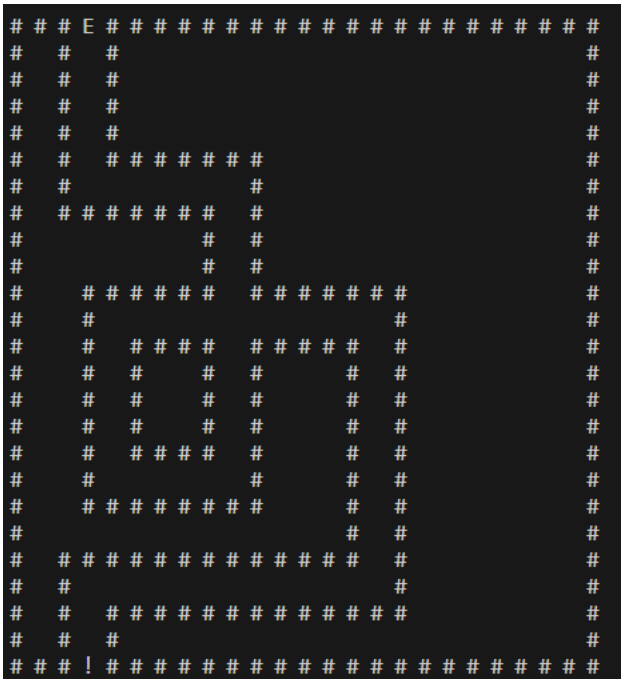
iv. Les couloirs

Voici, selon nous la partie qui nous a demandé le plus de réflexion. On voulais laissez le choix à l'utilisateur de créer les couloirs de son donjon sans pour autant le laisser sortir de ce dernier ou rentrer dans les salles.

Tout d'abord, nous avons défini un couloir comme une structure caractérisé par un nombre de déplacement (Ex : 4 ou 5), une séquence (Ex : SSOE), une longueur, une hauteur et un tableau dynamique de permettant de modifier le donjon.

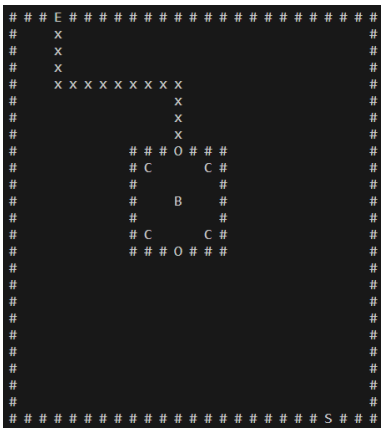
```
typedef struct
{
    int nb_deplacement;
    char *sequence;
    int H;
    int L;
    char **unCouloir;
} Couloir;
```

Pour tracer les couloirs, nous avons mis en place un moyen de demander à l'utilisateur à partir de quel endroit il veut commencer son couloir. Le départ du couloir est proposé par le programme en remplaçant une porte par un « **!** ». Si l'utilisateur veut l'utiliser comme début du couloir il valide, dans le cas contraire le programme va lui proposer une autre porte. Pour l'affichage des couloirs est réalisé grâce aux mouvement **Nord, Sud, Ouest et Est.** Tant que le couloir ne rencontre pas une autre porte, il ne s'arrête pas et il ne peut pas traverser un mur. Nous pouvons également créer **des intersections en deux couloirs** ou un couloirs qui **se recoupe lui-même (voir photo ci-dessous).** Ça nous a permis d'étendre les possibilités de création de l'utilisateur.

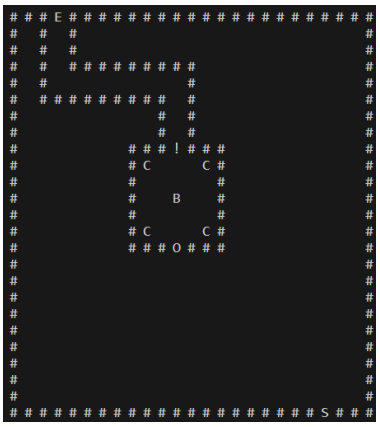


Voici un exemple **des différentes étapes** de la création des couloirs

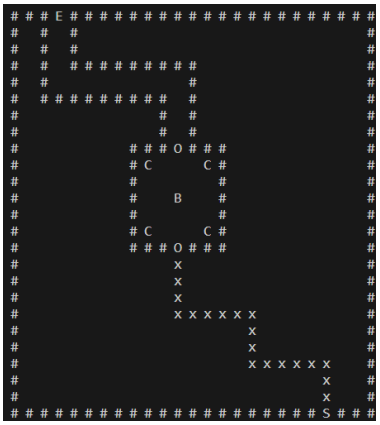
1) Sélection de la porte et définition de l'itinéraire



2) Validation du couloir



3) Sélection d'une nouvelle porte et définition d'un nouvel itinértaire



4) Validation du nouveau couloir et demande de selection d'une nouvelle porte ou fin du protocole de création des couloirs.

Cette méthode nous as permis de limiter la définition d'un couloir et de controler ce que l'utilisateur peut faire avec lui. Mais elle a néanmoins quelque inconvénients notamment le plus important étant que si l'utilisateur le veut, il peut ne pas mettre de couloir vers l'entrée ou vers la sortie.

c. La sauvegarde

Le projet ne consistant pas uniquement à la création d'un constructeur de donjon mais également à la sauvegarde des composant de ce derniers dans un fichier (ici un fichier .txt).

v. Sauvegarde des salles

La sauvegarde de nos salles est réalisée grâce aux informations de ces dernières et grâce à la fonction « fprintf ». Cette fonction appartient à une bibliothèque standard de C, « stdio.h ». Elle va formater les données présentes sur un fichier et réécrire de nouvelles informations sur ce dernier. Dans le programme si dessous on va récupérer chaque caractère présent à chaque emplacement pour les écrire dans le fichier et à chaque fin de ligne on n'oublie pas « \n » afin de retourner à la ligne.

```
void afficher_salle_fichier(FILE *fichier, Salle *s)
{
    for (int i = 0; i < s->H; i++)
    {
        for (int j = 0; j < s->L; j++)
        {
            fprintf(fichier, "%c ", s->uneSalle[i][j]);
        }
        fprintf(fichier, "\n");
    }
}
```

salle.txt

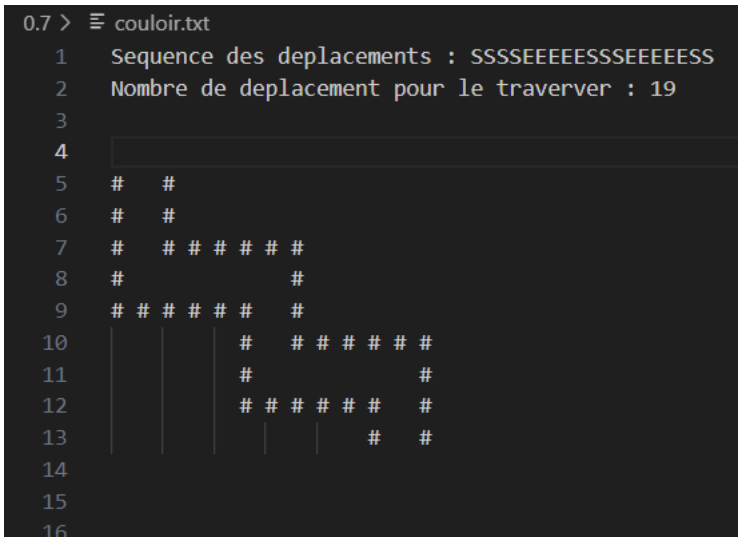
```
Une salle de dimension 7 sur 7
# # # O # # #
# C       C #
#           #
#     B     #
#           #
# C       C #
# # # O # # #

```

vi. Sauvegarde des couloirs

Pour la sauvegarde des couloirs, la logique n'est pas exactement la même. Une fois que ce dernier est tracé, on va enregistrer la séquence puis calculer la place nécessaire pour dessiner le couloir avec uniquement des « X ». Il va ensuite le finaliser avec des « # » et enfin il va enlever les « X » aux milieux des couloirs. Et c'est uniquement après ça que l'on pourra afficher le couloir dans un fichier .txt comme sur le document ci-dessous.

```c
void afficher_fichier_couloir(FILE *fichier, Couloir *c) // Afficher le couloir dans un fichier
{
    fprintf(fichier, "Sequence des deplacements : %s \n", c->sequence);
    fprintf(fichier, "Nombre de deplacement pour le traverver : %d \n", (int)strlen(c->sequence));
    fprintf(fichier, "\n");
    for (int i = 0; i < c->H; i++)
    {
        for (int j = 0; j < c->L; j++)
        {
            fprintf(fichier, "%c ", c->unCouloir[i][j]);
        }
        fprintf(fichier, "\n");
    }
}
```



vii. Sauvegarde du Donjon

Dans le cas des donjons. Nous avons fonctionné de la même manière qu'avec les salles. En affichant les entrées et sorties mais également les couloirs.

```c
void afficher_fichier_donjon(FILE *fichier, Salle *s) // Afficher le donjon dans le fichier.txt
{
    fprintf(fichier, "Un donjon de dimension %d sur %d \n", s->L, s->H);
    for (int i = 0; i < s->H; i++)
    {
        for (int j = 0; j < s->L; j++)
        {
            fprintf(fichier, "%c ", s->uneSalle[i][j]);
        }
        fprintf(fichier, "\n");
    }
}
```

donjon.txt

```
Un donjon de dimension 25 sur 25
# # # E # # # # # # # # # # # # # # # # # # # # # # # # #
#   #   #       # # # # # # #                           #
#   #   #       #           #                           #
#   #   # # # # # #   # # #   #                           #
#   #               #   #   #                           #
#   # # # # # # # # # #   #   #                           #
#                       #   #                           #
#                       #   #                           #
#                       #   #                           #
#                       #   #                           #
#                       #   #                           #
#                       #   #                           #
#                       #   #                           #
#                       #   #                           #
#                       #   #                           #
#                   # # # # #   #                       #
#                   #           #                       #
#                   #   # # # # # #                     #
#   # # # # # # # #   #                                 #
#   #               #                                   #
#   #   # # # # # # # #                                 #
#   #   #                                               #
#   #   #                                               #
# # # S # # # # # # # # # # # # # # # # # # # # # # # # #

```

d. La modification des éléments

Pour finir notre application, nous avons ajouté la fonctionnalité de modifier directement les éléments présents sur le donjon. Il s'agit de la fonction « **modifier_donjon ».** Elle contient de nombreuses conditions et demande notamment d'abord si l'utilisateur souhaite ou non effectuer des modifications. Si oui, il choisira l'emplacement en X et en Y par rapport au donjon. Il verra alors un « ! » apparaitre et la possibilité de confirmer ou non cette position. Cela permettra alors de simplifier la modification des éléments. En parlant des éléments, après la validation de cette position, l'auteur choisira ensuite parmi une liste d'éléments (Voir ci-dessous)

```
17  typedef enum
18  {
19      COFFRE = 'C',
20      MONSTRE = 'M',
21      DEPOUILLE = 'W',
22      AUTEL = 'A',
23      PIEGE = 'P',
24      CHAMPION = 'B',
25      MUR = '#',
26
27      ENTREE = 'E',
28      SORTIE = 'S',
29      PORTEF = 'F',
30      PORTEO = 'O',
31  } Element;
```

Cette liste contient donc tous les éléments du donjon. Nous laissons à l'utilisateur un choix complètement libre pour la position de chaque élément. S'il souhaite placer un boss dans un couloir, juste en dessous de l'entrée du donjon, il peut. Car après tout**, le client est roi.**

Conclusion

a. Projet

i. Attentes et réalisations

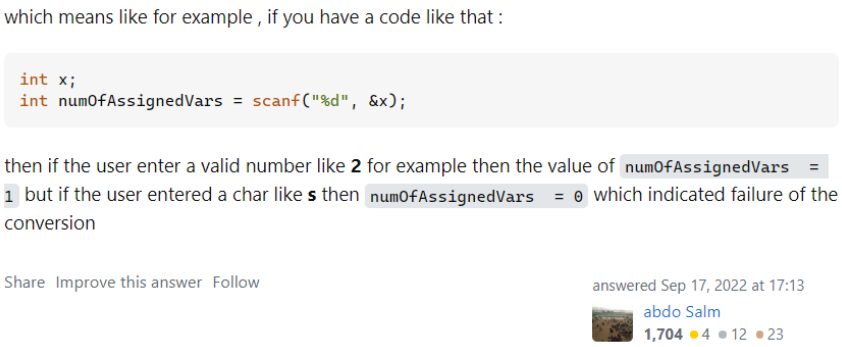
L'objectif principal de ce projet était de réaliser un programme opérationnel simulant la création d'un donjon. Dans lequel il fallait prendre en compte les dimensions des éléments (Donjon, salles et couloirs) et l'insertion d'éléments à l'intérieur du donjon.

D'un point de vue purement fonctionnel, le but a été atteint. Mais si l'on se penche un peu sur notre produit on a noté quelques éléments que nous n'avons pas pu implémenter. En premier lieu la possibilité d'adapter les dimensions des couloirs comme celle des salles. Puis nous avons eu quelques problèmes avec la vérification des types des inputs utilisateurs.

L'idée de créer un tableau avec la numérotation des lignes et des colonnes pour aider l'utilisateur à se repérer plus facilement dans le donjon nous a aussi traversé l'esprit. Cela aurait plus simplifier grandement le placement des objets et des portes dans notre donjon. Par manque de temps, cette fonctionnalité ne pourra pas être insérer dans notre application.

ii. Solutions

Dans le cas des inputs, nous avons recherché quelques solutions afin de forcer le type rentrer par l'utilisateur.



which means like for example , if you have a code like that :

```
int x;
int numOfAssignedVars = scanf("%d", &x);
```

then if the user enter a valid number like **2** for example then the value of `numOfAssignedVars = 1` but if the user entered a char like **s** then `numOfAssignedVars = 0` which indicated failure of the conversion

Share Improve this answer Follow

answered Sep 17, 2022 at 17:13

abdo Salm

1,704 ● 4 ● 12 ● 23

Dans notre cas, on va comparer les types et si celui de l'éléments rentrer par l'utilisateur n'est pas le bon on va rentrer dans une boucle qui redemanderas jusqu'à ce que le type soit bon.

Voici un exemple de code que l'on aurait pu utiliser afin de vérifier le type des inputs utilisateurs.

```c
int main()
{
    int x;
    int UserInput; // le nombre que l'utilisateur va rentrer

    printf("rentrer un numero");
    scanf("%d", &UserInput);
    while (scanf("%d", &UserInput)!= 1){ // tant que l'utilisateur ne rentre pas un nombre
        printf("erreur de type, veuillez rentrer un nombre");
        scanf("%d", &UserInput);
    }
    printf("vous avez rentrer %d", UserInput);
    return 0;
}
```

Dans le cas de la taille des couloirs nous avons pensés à demander aux utilisateurs avant de créer leurs couloirs quelles tailles ils aimeraient.

b. Bilan

Ce projet a été une source d'expérience. Il nous a permis de mieux se rendre compte de l'environnement qui nous attendras à la suite de nos études. D'un point de vue purement académique, à travers les heures de recherches liés à la réalisation de ce projet nous avons pu mieux approfondir nos connaissances et dans certains cas abordés une autre manière de voir la programmation en C. Que ce soit à travers l'exigence et l'organisation nécessaire pour ce projet nous avons pu nous améliorer sur plusieurs aspects.

Table des matières