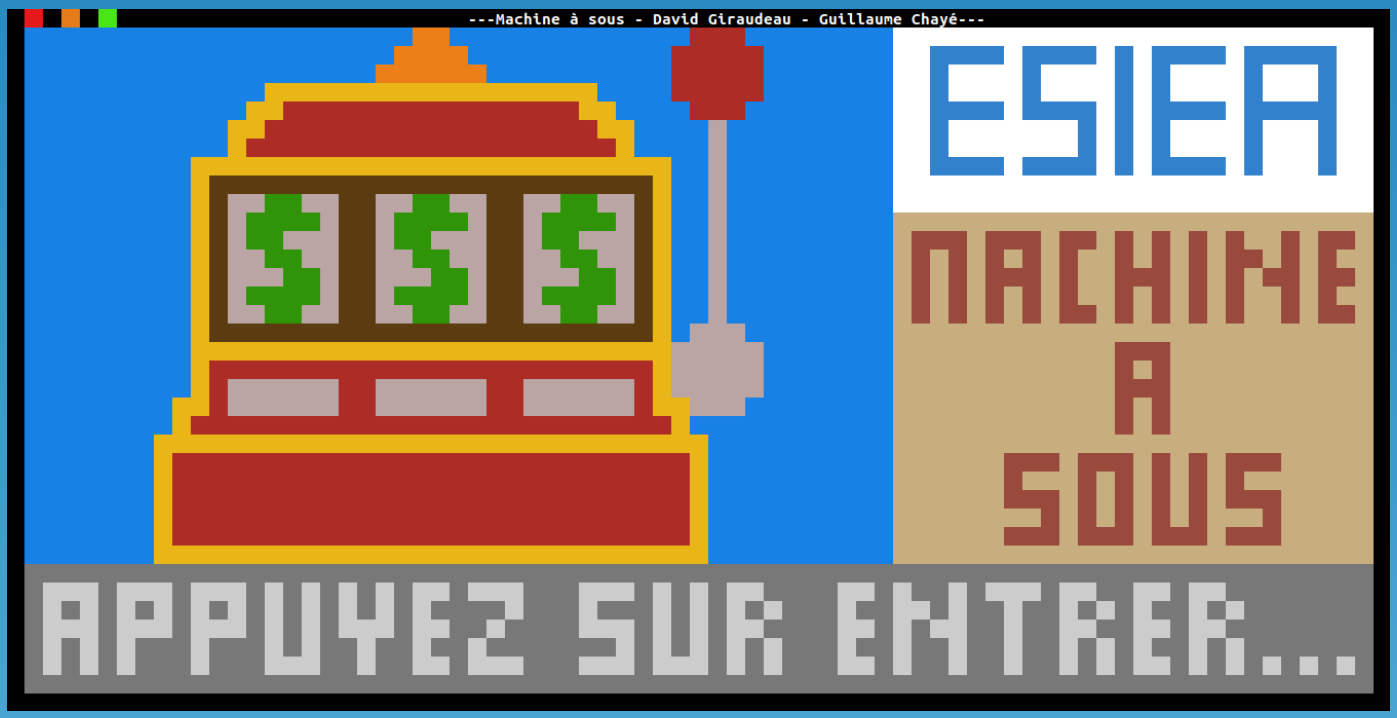
CHAYÉ Guillaume
GIRAUDEAU David
Classe 12

ESIEA
1ère année

Projet d'informatique : MACHINE A SOUS



Mai 2020

Table des matières

Introduction

Dans ce rapport, nous allons vous présenter les enjeux et les solutions mises en place du projet de réalisation d'une machine à sous. Avant même de commencer à le coder en C, nous nous étions fixés quelques objectifs, notamment au niveau des animations. Nous voulions simuler une roue qui tourne, puis qui s'arrête aléatoirement sur une lettre. Quelquechose que nous n'envisagions pas au départ, en revanche, est l'affichage d'une image bitmap sur le terminal. Cet ajout a permis de réaliser un programme plus ergonomique pour l'utilisateur, avec par exemple la création du menu de sélection du thème de la machine. Le projet est programmé dans un langage qui n'est pas orienté objet, ce qui rend la tâche encore plus complexe. En effet, le type de donnée le plus évoluée en C est la structure. Dans ce rapport nous allons donc vous expliquer les étapes et les difficultés rencontrées dans la réalisation de ce projet.

I- Affichage et simulation d'une roue qui tourne

L'enjeu de cette partie a été de créer un affichage rapide et efficace pour simuler la rotation d'une roue. Nous avons pour cela utilisé un affichage ligne par ligne et non pas caractère par caractère. Tout l'affichage du projet est fait autour de pixels. Il fallait donc que l'affichage des caractères se fasse de manière pixélisé. Afin de maximiser les performances du programme, nous avons fait le choix de minimiser la complexité de l'algorithme, qui se traduit cependant par une allocation de mémoire plus importante.

Comme vous pouvez le voir, un caractère est composé de pixel de deux couleurs : blanc et orange. Pour pouvoir afficher deux couleurs sur une même ligne, nous avons utilisé les « escape sequences » qui permettent de changer la couleur d'arrière-plan et de premier plan du curseur.

Ainsi pour afficher un pixel de la couleur de l'arrière-plan, il faut afficher un espace. Mais pour afficher un pixel de la couleur du premier plan ce n'est pas si facile.

Techniquement il faudrait utiliser le caractère 219 « █ » aussi appelé « block character ». De plus, il est impossible d'afficher des caractères de la table ascii dont le numéro dépasse 127 sur le terminal. Il a donc fallu utiliser des caractères dit « larges ». Le type wchar_t est

défini dans l'en-tête « `wchar.h` ». On peut afficher l'intégralité des wchar_t en changeant les paramètres d'affichage des caractères de la fenêtre. C'est pour cela qu'on retrouve la ligne en début de programme :

```
setlocale(LC_CTYPE,"");
```

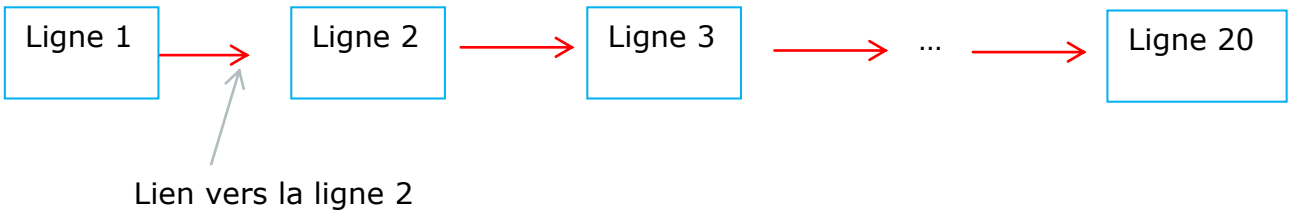
Ainsi chaque ligne est une chaîne de wchar_t. Pour l'afficher, il suffit de la formater dans un `printf` avec `%ls`.

```
printf("%ls",L"Ceci est une chaine de wchar_t");
```

Le 'L' devant les guillemets est caractéristique des wchar_t.

Le sujet du projet imposait une dimension de 40x20 cases pour la représentation d'un caractère. Il faut donc allouer 20 chaines de 41 wchar_t (40 plus le `L'\0'` de fin de chaine).

Le stockage de la représentation du caractère n'est pas fait dans un tableau de 20 lignes mais dans une chaine de 20 lignes. Pour mieux expliquer voici un schéma :



Pour réaliser cette structure des données en C, on utilise les listes chaînées. Voici celle utilisée dans le projet :

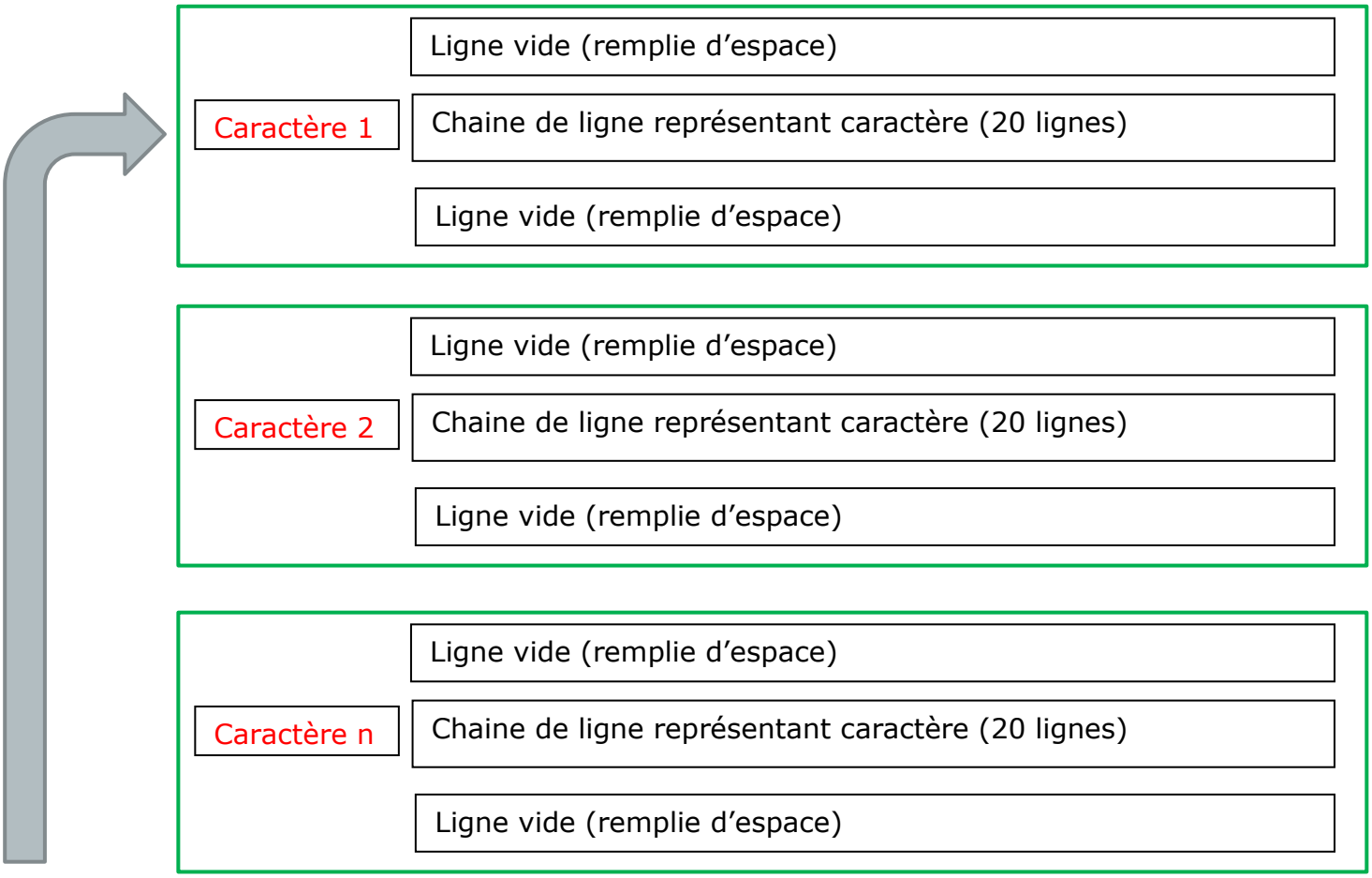
```
typedef struct line{
    wchar_t char_list[CHARACTER_WIDTH+1]; //pour le \0
    char character;
    struct line *next;
}line;
```

Cette structure a une particularité : on ne peut passer que d'une ligne à la ligne suivante. On ne peut pas revenir à la ligne précédente depuis une ligne.

Chaque représentation de caractère forme des morceaux de chaines. Le programme va assembler chaque morceau de chaine représentant un caractère les uns à la suite des autres puis va refermer la chaine au niveau des extrémités afin de créer une chaine de ligne fermée. Autrement dit, on va lier la 20ème ligne de chaque caractère à la première du caractère suivant. Lorsqu'on arrive à la fin

de la liste des caractères, on lie la dernière ligne du dernier caractère à la première ligne du premier caractère.

Le schéma ci-dessous illustre cette idée :



Pour afficher un caractère, il faut utiliser une fonction que nous appellerons « DisplayLine » dont le pseudo code est le suivant :

```
Fonction DisplayLine(ligne_de_départ)
|       ligne=ligne_de_depart
|       placer le curseur en haut de l'écran
|       pour i allant de 0 à 20 exclue faire
|       |       afficher ligne
|       |       afficher un retour à la ligne
|       |       affecter à ligne la ligne suivante
|       fin pour
Fin fonction
```

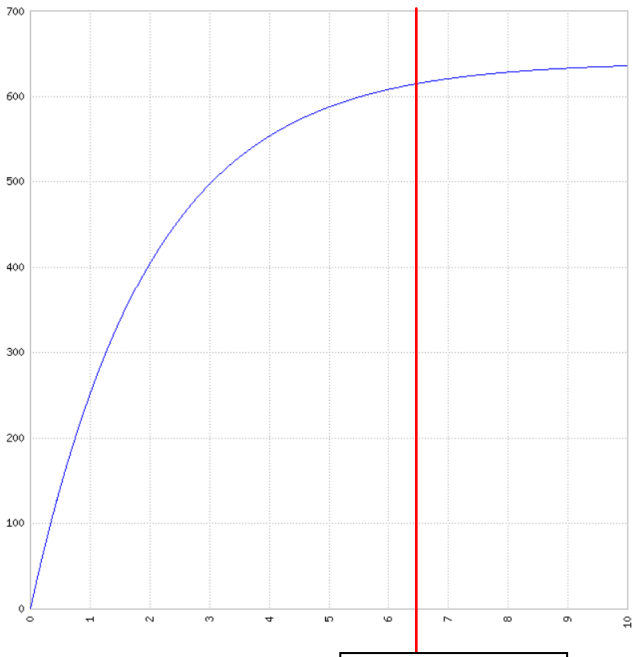
Pour faire défiler les caractères, on utilise le principe suivant :

```
Ligne_courante désigne la première ligne de la représentation d'un caractère
Affecter 0 à i
Affecter 0 à t
Tant Que vrai faire
|       DisplayLine(Ligne_courante)
|       Affecter à Ligne_courante la valeur de la ligne suivante
|       Affecter i+1 à i
|       Attendre 20 millisecondes
|       Affecter t+0.02 à t
Fin Tant Que
```

La variable i désigne la position de la ligne affichée à l'instant t (en secondes).

Dans le programme, les lignes défilent à une vitesse décroissante jusqu'à s'arrêter. Pour contrôler la vitesse du déroulement des caractères le programme va imposer qu'à l'instant t, on affiche la ligne de position n. La ligne de départ lors du lancement de la roue porte la position 0.

Voici la courbe de la position en fonction du temps (en seconde) :



Point d'arrêt

Cela correspond à la fonction f telle que f(t)=$\frac{Vo}{k}*(1-e^{-k*t})$

Où Vo est la vitesse initiale et k un coefficient tel qu'ici :

$$0.4 \leq k \leq 0.5$$

k représente le coefficient de frottement.

Avant le lancement de la roue le programme va tirer au hasard le point d'arrêt de la roue puis calcule la vitesse initiale adéquate.

Pour que les trois roues s'arrêtent les unes à la suite des autres le programme va faire varier le coefficient de frottement ainsi que la position finale d'arrêt.

La position finale est un multiple de la hauteur d'un caractère afin de ne pas se retrouver entre deux caractères à l'arrêt.

Le mode « sans animation » tire la position finale aléatoirement puis s'y rend immédiatement sans afficher le défilement intermédiaire.

Si vous remontez sur la déclaration de la « structure line », vous verrez qu'elle contient une variable « character » de type « char ». Elle permet de trouver directement le caractère associée à sa représentation.

Pour récupérer la combinaison de trois caractères, il suffit de lire le caractère associé à la ligne d'arrêt.

Les fonctions permettant de charger les caractères sont présentes dans les fichiers « `character.c` » et « `character.h` ». Celle liées à l'affichage des caractères sont dans les fichiers « `display.c` » et « `display.h` »

Ce qui nous mène à la deuxième partie sur le stockage des combinaisons.

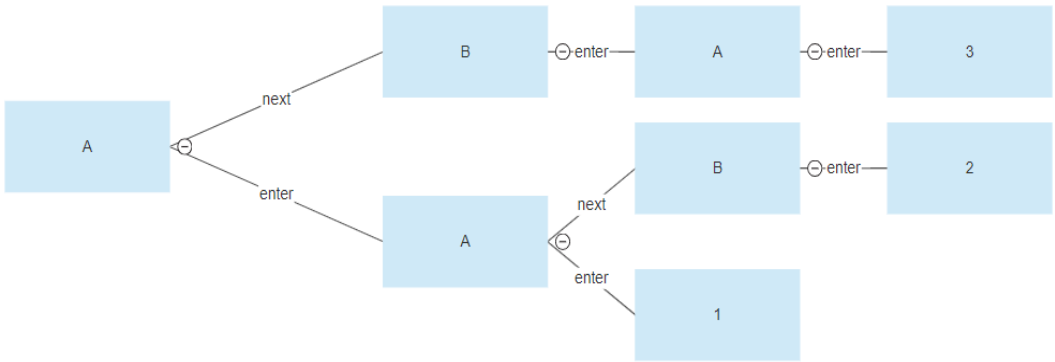
II- Stockage des combinaisons et gains associés

Le stockage de combinaisons consiste à attribuer une valeur de gain à une clé de trois caractères donnés. C'est ce qu'on peut appeler un dictionnaire en informatique. Cet outil est présent en python mais pas en C. Plusieurs solutions existent pour en créer. On peut par exemple passer par des tables de hachage (fonction qui transforme une chaine de caractère en un nombre unique).

Nous avons utilisé une autre technique de stockage qui est basée sur un arbre à double entrée. Voici la déclaration de la structure :

```
typedef struct tree{
    struct tree *next,*enter;
    int value;
}tree;
```

Pour comprendre ce mode de stockage, nous allons prendre un exemple. On veut attribuer à « AA » la valeur 1, « AB » la valeur 2 et « BA » la valeur 3.



L'arbre est composé de blocs reliés entre eux. Chaque bloc stocke une valeur. Depuis un bloc, on peut accéder à deux autres blocs (enter, next).

On cherche à trouver la valeur associée à « AB ». On part du bloc A tout à gauche. On cherche la première lettre de « AB » autrement dit « A ». Justement ce bloc porte la lettre A ; on passe au bloc suivant par la voie « enter ». On recherche maintenant la seconde lettre de « AB » (B). Ce bloc porte la lettre A. Ce n'est pas la bonne lettre donc on passe au bloc « next ». Il porte la lettre B donc on passe au bloc « enter ». Comme B était la dernière lettre de « AB », on retourne la valeur du bloc « enter » c'est-à-dire 2.

De manière générale, on passe au bloc « next » tant que la valeur du bloc ne correspond pas à la lettre cherchée. Dès qu'on a trouvé la lettre on passe dans le bloc « enter » et on recherche la lettre suivante. La valeur recherchée correspond à celle présente dans le bloc « enter » de la dernière lettre de la clé.

Les fonctions concernant les combinaisons sont présentes dans les fichiers « `tree.h` » et « `tree.c` ». On distingue notamment deux fonctions: `tree_add_value` (permet d'ajouter une clé associée à une valeur dans un arbre), `tree_get_value` (permet de récupérer une valeur associée à une clé). Pour être plus précis la fonction `tree_get_value` renvoie un pointer vers le bloc contenant la valeur. Elle renvoie le pointer `NULL` en cas d'erreur. La fonction `generate_gain_grid` permet de générer un tableau des gains selon la combinaison et la mise dans un fichier.

III- Stockage d'un thème

Toutes les informations concernant la roue de casino sont stockées dans une structure définie dans « `main.h` » dont la déclaration est la suivante :

```
typedef struct random_wheel{
    tree* combinations;
    line* start;
    uint16_t money;
    bool animation;
}random_wheel;
```

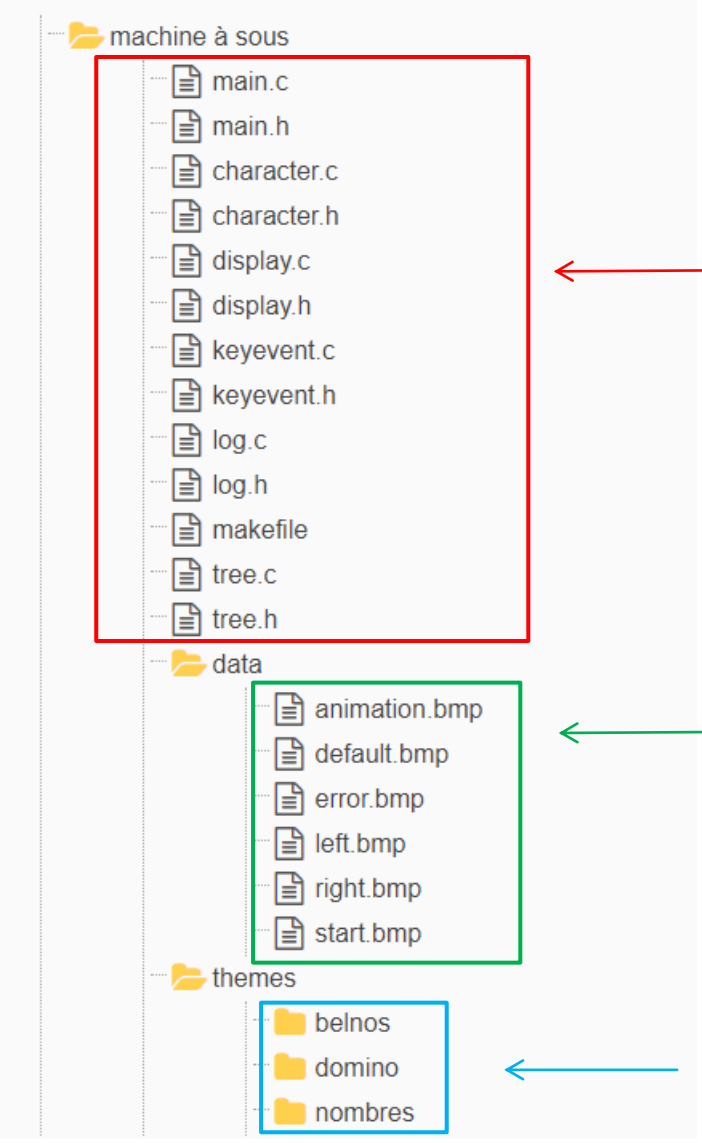
Comme vous pouvez le constater, elle rassemble la représentation des caractères qui la compose (présence d'un pointer vers une structure line) ainsi que l'arbre stockant les combinaisons (présence d'un pointer vers une structure de type de tree). D'autres informations sont également présentes : l'argent de l'utilisateur et s'il a choisi d'activer les animations. Toutes ces informations sont chargées

dans la structure dans le menu principal (plus précisément lorsque que l'on choisit le thème de la roue du casino).

IV- Chargement d'un thème

Dans cette partie, nous allons étudier comment les différents thèmes de roue de casino sont stockés.

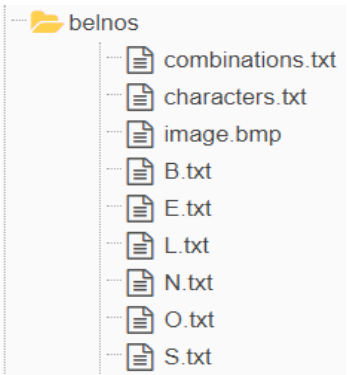
Tout d'abord voici un petit aperçu de l'arborescence des fichiers du projet :



Dans le répertoire courant on retrouve les fichiers du programme

Le dossier data regroupe les images et le son du programme ainsi que le fichier log et le tableau de combinaisons du thème sélectionné.

C'est dans le dossier themes que sont stockés les différents thèmes des roues. Chaque thème est stocké dans un dossier différent. Le nom du thème est donné par le nom du dossier. Nous nous intéresserons particulièrement à ce dossier.



Chaque thème suit les mêmes caractéristiques dans l'organisation des fichiers. Nous allons le

découvrir en analysant le thème belnos.

Voici le contenu du dossier :

Tout d'abord le fichier « `combinations.txt` » (obligatoire) stocke sans surprise les combinaisons gagnantes de la roue. En effet cela ne sert à rien de noter toutes les combinaisons possible puisque qu'il suffit d'attribuer un gain nul lorsque la fonction `tree_get_value` retourne `NULL`.

La syntaxe à l'intérieur du fichier est la suivante :

[Combinaison de 3 lettres] [gain associé]

Pour ajouter une autre combinaison, il suffit de passer à la ligne puis d'écrire une autre combinaison et son gain en suivant la syntaxe. Le chargement de ce fichier est assuré par des fonctions contenues dans « tree.c »

Le fichier « `characters.txt` » (obligatoire) liste les caractères de la roue du casino.

Sa syntaxe est la suivante :

[nom du fichier] [caractère associé]

Comme pour « `combinations.txt` » il suffit de passer à la ligne et répéter la syntaxe pour ajouter un caractère.
Pour chaque caractère le programme va lire sa représentation dans le fichier indiqué. Notez que la représentation des caractères dans un fichier a pour dimension 10x10 pour faciliter la tâche. Le programme réalisera un agrandissement par 4 en largeur et par 2 en hauteur pour le mettre aux bonnes dimensions. Pour dessiner le caractère, il suffit de mettre des espaces pour laisser du blanc et le caractère associé pour afficher des pixels orange. Il est important de mettre un retour à la ligne à la fin de la représentation pour que la dernière ligne soit prise en compte.

Exemple avec le fichier `S.txt` associé au caractère S :

```
 1  SSSSSSSS
 2 S
 3 S
 4 S
 5  SSSSSSS
 6         S
 7         S
 8         S
 9         S
10 SSSSSSSS
11
```

Le chargement des caractères est assuré par des fonctions contenues dans « `character.c` »

Le fichier « `image.bmp` » est facultatif. Il permet de mettre une image de présentation du thème. S'il n'est pas présent l'image de présentation sera remplacé par l'image « `default.bmp` » présente dans le dossier data.

En cas d'échec du chargement du thème l'image « `error.bmp` » sera affiché et nous expliquerons ci-après comment trouver facilement l'origine de l'erreur.

V- Gestion des erreurs

Lors de l'exécution du programme un fichier de logs nommé « `machine.log` » est généré dans le répertoire « data » du programme. Ce fichier contient des informations sur les fonctions exécutées. Le fichier peut être consulté pendant l'exécution du programme en sélectionnant le choix « consulter le fichier log » depuis le menu principal.

Voici un exemple :

```
16 2020-05-19 00:24:11> Tentative de fermeture du paquet actuel
17 2020-05-19 00:24:11> Chargement du paquet:nombres
18 2020-05-19 00:24:11> 90 combinaison(s) ont bien été chargé(es) (combinations.txt)
19 2020-05-19 00:24:11> le caractère 1 contenu dans 1.txt a bien été chargé (10 ligne(s) détectée(s))
20 2020-05-19 00:24:11> le caractère 3 contenu dans 3.txt a bien été chargé (10 ligne(s) détectée(s))
21 2020-05-19 00:24:11> le caractère 4 contenu dans 4.txt a bien été chargé (10 ligne(s) détectée(s))
22 2020-05-19 00:24:11> le caractère 6 contenu dans 6.txt a bien été chargé (10 ligne(s) détectée(s))
23 2020-05-19 00:24:11> le caractère 7 contenu dans 7.txt a bien été chargé (10 ligne(s) détectée(s))
24 2020-05-19 00:24:11> Erreur: Impossible de charger le caractère 8 contenu dans 10.txt
25 2020-05-19 00:24:11> 5 caractère(s) ont été chargé(s) avec succès
26 2020-05-19 00:24:11> Affichage avec succès du bitmap image.bmp
27 2020-05-19 00:24:11> Le paquet nombres a bien été chargé
```

Le fichier log permet de dater et de décrire les évènements et erreurs du programme. Ici on peut voir en ligne 24 qu'il y a une erreur lors du chargement du paquet. Seuls 5 caractères sur les 6 ont pu être chargés. La raison de l'erreur est bien indiquée. Il manque sûrement le fichier `10.txt` dans le répertoire nombres.

Les fonctions permettant de générer un fichier log sont présentes dans « `log.h` » et « `log.c` ».

VI- Affichage des images bitmap

L'ajout d'une fonction qui décode les bitmap a simplifié grandement l'affichage du programme. En effet, cette fonction permet de remplir des zones entières de l'écran avec une image. La fenêtre de lancement du programme est entièrement une image, de même que le menu qui demande si l'utilisateur veut activer ou non les animations. On peut encore citer les flèches du menu principal.

Pour créer une telle fonction, il a fallu lire des articles sur le format bitmap et sur les astuces pour le décoder facilement en C. Un bitmap est composé d'une partie appelé « header » qui contient toutes les informations sur l'image et d'une partie « data » qui contient les pixels de l'image. Pour lire un fichier bitmap, il faut l'ouvrir en mode « lecture binaire ». Lors de la lecture il faut veiller à ne pas se décaler d'un seul octet dans la lecture du « header » afin de ne pas fausser toutes les informations lors de la lecture. Il faut également retourner l'affichage verticalement car sur le terminal l'axe des ordonnées est orienté vers le bas.

Voici les structures qui rassemblent toutes les données du « header » :

```c
typedef struct BitMapHeader{ //en tête d'un fichier bitmap (taille:12 octets)
    uint32_t FileSize;
    uint16_t Reserved1;
    uint16_t Reserved2;
    uint32_t Offset;
}BitMapHeader;

typedef struct BitMapInfoHeader{//partie de l'en tête qui contient les infomations de l'image (taille:40 octets)
    uint32_t Size;
    uint32_t Width;
    uint32_t Height;
    uint16_t Planes;
    uint16_t BitsPerPixel;
    uint32_t Compression;
    uint32_t ImageSize;
    uint32_t HorizontalResolution;
    uint32_t VerticalResolution;
    uint32_t ColorUsed;
    uint32_t ImportantColors;
}BitMapInfoHeader;
```

La taille pour stoker chaque propriété de l'image est précise. On utilise des entiers non signé de taille 8 bits (1 octet), 16 bits (2 octets) et 32 bits (4 octets) dont le type est défini dans l'en-tête « `stdint.h` ». La fonction se contente d'afficher le bitmap au fur et à mesure qu'elle le lit. Elle ne le stocke pas en mémoire. La fonction est présente tout en bas du fichier « `display.c` ».

VII- Gestion des évènements claviers

La seule manière pour l'utilisateur de communiquer avec le programme est d'utiliser le clavier, car nous sommes sur terminal. Dans le programme, vous remarquerez qu'à aucun moment on ne demande de saisir du texte à l'utilisateur. En effet, c'est pour éviter de devoir gérer toutes sortes d'erreurs en cas de mauvaise saisie ou encore de devoir vider le buffer.

Pour gérer les évènements clavier, le programme utilise la structure `KeyEvent` défini dans « `keyevent.h` » ainsi que les fonctions dans le fichier « `keyevent.c` » :

```
typedef struct KeyEvent{
    char key;
    struct KeyEvent (*action)();
    void *arg;
}KeyEvent;
```

Cette structure permet d'associer à une touche une action avec un argument. L'action est stockée sous la forme de pointer sur une fonction.

A partir de cette structure le programme va utiliser une fonction (bloquante) `WaitForKeyEvent` dont le pseudo-code est le suivant :

```
EventList est une liste d'évènement clavier de taille n
n et c sont des entiers

Fonction WaitForKeyEvent(EventList,n)
|       Tant Que Vrai faire
|       |       Attendre qu'une touche soit pressé et affecter son numéro à c
|       |       Pour i allant de 0 à n exclue faire
|       |       |       si la touche associée à EventList[i] est égale à c alors
|       |       |       |       Retourner EventList[i]
|       |       |       Fin si
|       |       Fin pour
|       Fin Tant Que
Fin Fonction
```

Cette fonction ne s'arrête pas tant que l'utilisateur n'a pas pressé une touche faisant partie de la liste d'évènement clavier.

En C, on utilise la fonction `getchar` pour récupérer une touche du clavier. Pour éviter de devoir appuyer sur la touche [entrer] après chaque touche, le programme va changer certains paramètres du terminal. Le terminal passe en mode ligne et n'affiche pas les touches pressées (pour des raisons d'esthétiques).

Ces changements sont effectués grâce à la ligne de code :

```
system("stty raw -echo"); //changement des paramètres de la fenêtre
```

Pour rétablir les paramètres par défaut, on utilise la ligne :

```
system("stty cooked echo"); //on rétablit les paramètres précédents de la fenêtre
```

Toute la structure principale du programme est basée sur la structure KeyEvent. Chaque menu (au niveau de l'affichage) est en fait une fonction retournant un évènement clavier. Cet évènement stocke une action qui est dans ce cas le pointer vers la fonction du menu suivant. Ainsi lorsque qu'une fonction se termine, une autre est relancé immédiatement. Pour être plus simple, chaque fonction va retourner la fonction suivante à exécuter.

Voici le pseudo-code qui permet de gérer l'enchainement des fonctions :

```
Event est un évènement clavier.

Associer à l'évènement Event l'action MenuPrincipal et l'argument 0.
Tant Que l'action associée à Event est valide faire
|       Exécuter la fonction Event.action avec comme argument Event.arg
|       Associer la valeur de retour de la fonction à Event
Fin Tant Que
```

VIII- Réalisation du projet

Nous avons commencé par programmer l'affichage des caractères et l'animation. C'est sur cette partie que nous avons passé le plus de temps car il fallut résoudre tous les problèmes liés à l'affichage, puis trouver un modèle adéquat pour représenter l'animation d'une roue. Ensuite, nous avons rapidement créé la structure pour stocker les combinaisons. Puis nous avons été confrontés au problème de l'organisation du stockage des thèmes. Il a fallu trouver une solution pour ne pas surcharger les dossiers du projet et créer des fonctions pour lire dans les dossiers et fichier.

Pour gérer les dossiers et fichiers nous avons notamment utilisé ces fonctions :

```
DIR *opendir(const char *dirname); //ouvre un répertoire
int closedir(DIR *dirp); //ferme un répertoire
struct dirent *readdir(DIR *dir);//lit les entrées dans un répertoire
FILE *fopen(const char *filename, const char *mode);//ouvre un flux (fichier)
int fclose(FILE *stream); //ferme le flux
char *getcwd (char *buf, size_t size); //permet d'obtenir le chemin du répertoire courant du programme
int chdir(const char *path); //change de répertoire courant
```

Ces fonctions sont déclarées dans « `stdio.h` », « `dirent.h` » et « `stdlib.h` ».
Nous avons ensuite créé la fonction d'affichage des bitmaps. Ensuite nous avons créé une fonction permettant d'afficher les nombres en plus grand via un affichage de style 7 segments.

Ensuite, nous avons mis en place la détection des touches. Enfin, nous avons rassemblés chaque morceau de code pour former le projet. Les fichiers « `log.h` » et « `log.c` » ont été rajouté ensuite pour faciliter la création des thèmes.

Difficultés rencontrées

David :

Au début Guillaume m'a aidé à monter en compétences sur le langage C afin de pouvoir réaliser en binôme le projet. J'ai dû d'abord me renseigner et faire des cours sur internet afin de comprendre les idées de Guillaume ce qui a vraiment été une expérience enrichissante pour moi. Ma principale difficulté était de savoir comprendre sur le vif et d'apprendre rapidement toutes les nouvelles notions que nous avons abordées dans ce projet hors des cours.

Guillaume :

J'ai pu faire quelques découvertes durant ce projet notamment avec la manipulation des dossiers qui m'a posé quelques problèmes au début.
La création de la fonction d'affichage bitmap n'a pas été facile. Il a fallu comparer pas mal de sources pour se documenter. J'étais plutôt à l'aise en c car je pratique ce langage de manière autonome chez moi depuis le début de l'année.

Améliorations

Comme vous avez pu le constater, la structure `Tree` fonctionne bien dans notre cas. Cependant son organisation est assez complexe à mettre en place et la longueur de l'arbre dépend de la longueur de la clé. Après avoir pris un peu de recul sur le projet, nous nous sommes aperçus qu'un arbre binaire de recherche aurait été plus adapté et plus simple dans notre cas.

Conclusion

Ce qui aura été le plus compliqué durant ce projet est de structurer de manière efficace les données. Ce sont clairement les limites du langage C. Nous sommes limités aux structures et nous ne pouvons pas utiliser des classes qui auraient été pourtant très utiles. Cela entraine la création de fonctions à la syntaxe répétitive mais surtout l'obligation d'utiliser un grand nombre de variables dans les fonctions principales.

Ce projet aura été formateur, car il nous aura permis de pratiquer le langage C, et d'approfondir nos connaissances.

Sources

Lecture d'un bitmap :
https://stackoverflow.com/questions/50090500/create-simple-bitmap-in-c-without-external-libraries
http://www.di.unito.it/~marcog/SM/BMPformat-Wiki.pdf

Parcours d'un dossier :
https://www.geeksforgeeks.org/c-program-list-files-sub-directories-directory/

Fonctions des librairies standards en C :
https://www.tutorialspoint.com/c_standard_library/
http://manpagesfr.free.fr/man/man3/setlocale.3.html
https://stackoverflow.com/questions/8681623/printf-and-wprintf-in-single-c-code

Tutoriels C :
https://www.geeksforgeeks.org/data-structures/linked-list/
https://openclassrooms.com/fr/courses/19980-apprenez-a-programmer-en-c

« Escape sequences » sur le terminal :
https://stackoverflow.com/questions/4842424/list-of-ansi-color-escape-sequences
https://superuser.com/questions/668813/change-the-size-of-os-x-terminal-using-keyboard

Debugger C en ligne :
https://www.onlinegdb.com/

Annexe

Aucun thème n'est codé en dur dans notre programme. Ce qui signifie qu'on peut à tout moment ajouter un nouveau thème sans modifier le programme.

Pour cela il suffit de suivre ces étapes :

-créer un nouveau sous-dossier dans le répertoire «`themes`» sous le nom du nouveau thème. Si on relance le programme, on verra apparaître ce nouveau thème dans le menu principal avec un message d'erreur.

-créer deux fichiers sous le nom de «`combinations.txt`» et «`characters.txt`».

-créer tous les fichiers contenant le dessin des caractères (voir la section chargement d'un thème).

-remplir le fichier «`characters.txt`» avec les fichiers représentant les caractères et leur caractère associé.

-remplir le fichier «`combinations.txt`» avec les combinaisons souhaitées et un gain ne dépassant pas 1000.

-vous pouvez si vous voulez ajouter une image bitmap sous le nom de «`image.bmp`» de dimension 43x26 pixels.

-si toutes les étapes ont été respectées, le thème devrait être chargé correctement dans le menu principal. En cas d'erreur, consulter le fichier log.