Cousin Brandon / Ngatchou Junior **20/11/2015**

Compte rendu Ruzzle :

1) Organisation du programme :

Le programme est conçu sous forme de plusieurs modules :

- **Un module pour l'affichage (display.c)**
- **Un module pour le tirage aléatoire (random.c)**
- **Un module pour le solver (solver.c)**
- **Un module pour faire une liste des mots trouvés et les trier (trie.c)**

- **Le programme principal (ruzzle.c)**

L'affichage :

Initialisation de la grille :
void initGrid(t_Case grid[N][N]) ;

Récupérer une case aléatoirement :
void getCase(t_Case grid[N][N], int line, int col, int nb_bonus[]) ;

Récupérer une case à partir du paramètre passé avec l'exécution du programme :
void getCaseFromStr(t_Case grid[N][N], int nb_bonus[], char gridStr[], int strIndex) ;

Affiche la grille :
void Grille(t_Case grid[N][N],int argc, char * gridStr[]) ;

Tirage aléatoire :

Tirage aléatoire d'une lettre :
char randChar() ;

Donne l'indice dans l'alphabet d'une lettre (de 0 à 25 pour utilisation avec une matrice) :
int getIChar(char c) ;

Tire une case aléatoirement :
int randCase() ;

Tire des bonus aléatoirement pour la grille :
void getBonus(char boCharL[], char boCharM[], int *nb_boL, int *nb_boM) ;

Solver :

Donne le total de points d'une case :
int totCase(t_Case Case) ;

Donne le coefficient multiplicateur du mot :
int mulWord(t_Case Case) ;

Récupère un mot dans le dictionnaire :
char* getWord(FILE * dict) ;

Cherche un point de départ dans la grille :
int searchStart(t_Case Case, char word[]) ;

Vérifie l'authenticité du mot formé à partir de la grille :
void confirmWord(char gridWord[], char word[], int pts) ;

Forme un mot à partir de la grille :
void formWord(t_Case grid[N][N], int i, int j, char word[]) ;

Cherche le mot tiré dans le dictionnaire dans la grille :
void searchWord(t_Case grid[N][N], char word[]) ;

Résoud la grille :
void Solver(t_Case grid[N][N]) ;

Liste des mots et tri :

Ajoute un élément dans la liste (mot + points) en triant à la volée :
void addElement(element * elem) ;

Affiche la liste :
void printList() ;

Supprime la liste :
void clearList() ;

Explication :

La version initiale :

Génère une grille aléatoire à partir du module d'affichage, le traitement est assez long (~30s) .
Le programme parcourt la grille entière, pour chaque lettre on ouvre le dictionnaire correspondant, le dictionnaire ayant été au préalable découpé en 26 dictionnaires (un pour chaque lettre).

Pour trouver les mots on parcourt les cellules voisines avec à chaque fois qu'une case correspond à la prochaine lettre du mot actuellement lu caractère par caractère dans le dictionnaire on continue le traitement sinon on revient en arrière.
Pour vérifier qu'un mot est incomplet on retourne 1 avec **searchWord()**, dans le cas d'un mot complet on retourne 2 sinon 0.
Le traitement étant réalisé par **findWords()**.

La version finale :

Afin de palier au défaut de la version initiale, une seconde version a été réalisée.
Cette nouvelle version permet deux choses, soit de rentrer en paramètre une grille saisie par l'utilisateur ou de générer une grille aléatoirement et la résoudre.

Pour cela par le biais du module d'affichage la grille qui est représentée sous une matrice de **Case (TAD)** est d'abord initialisée puis complétée par la fonction **getCase()** et **getCaseFromStr()** selon l'option choisie, aléatoire ou non.

Une fois la grille complète le solver est lancé, le module pioche un mot dans le dictionnaire avec **getWord()**, cherche un point de départ dans la grille, autrement dit la première lettre du mot pioché correspond à une lettre de la grille.
Dès qu'un point de départ est trouvé le solver tente de former le mot à partir de ce point en cherchant par tout les chemins possibles.
Pour cela on regarde caractère par caractère le mot pioché si la prochaine lettre est présente dans les 8 cellules voisines alors on se place sur chaque occurrence trouvée, on ajoute la lettre au mot en cours de formation puis on continue ainsi de suite par récursivité.

Durant ce processus on modifie les variables globales **totalW** et **boW** qui correspondent au nombre de points total et bonus du mot en cours de formation / formé.
Tout ce traitement étant réalisé par **formWord()**.

Durant la formation du mot on vérifie à l'aide de **confirmWord()** si le mot formé correspond à celui pioché si c'est le cas alors on calcule le total des points en prenant en compte les bonus du mots et ajoute l'élément avec **addElement()** dans la liste des mots.

Finalement on affiche la liste avec **printList()** dans le programme principal (**ruzzle.c**).

Mode d'emploi :

$ **./bin/ruzzleSolver**

Permet d'exécuter le programme en générant une grille aléatoire, les lettres sont tirées au hasard en prenant compte de leurs fréquences d'apparition dans la langue française.

On peut prédéfinir une grille à l'aide d'une chaine de 16 caractères.

Par exemple:

$ **./bin/ruzzleSolver adcksxirmdesuckh**

Génèrera la grille :

A D C K

S X I R

M D E S

U C K H

Dans le cas d'une chaîne trop petite un message d'erreur apparaîtra, si la chaîne est trop grande, la grille sera composée des 16 premiers caractères.

Les bonus lettres et mots sont tirés aléatoirement de manière à ce qu'il n'y ait pas trop de bonus.

Le résultat du solver se retrouve dans :

assets/listWords.txt

Bilan :

La principale difficulté dans ce projet a été de trouver un moyen d'optimiser au maximum le traitement.
La version finale permet de résoudre la grille en l'espace de **~100-130 ms** alors que la version initiale le faisait en **~30s – 1 min** ce qui était relativement important et ne répondait pas aux contraintes imposées.

Pour mettre en place cette nouvelle version une branche de développement a été crée sur le dépôt git afin d'éviter la régression sur la version déjà fonctionnelle, ainsi j'ai pu travailler sans interférer avec la version opérationnelle, une fois finis tout a été fusionné en prenant soin de ne pas supprimer des portions de code indispensable.

En ce qui concerne le projet, la répartition des tâches s'établit de la manière suivante :

Élève :	Cousin Brandon	Ngatchou Junior
README.md :	**99 %**	**1 %**
Documentation :	**100 %**	**0 %**
Git :	**99 %**	**1 %**
Tests Unitaires :	**100 %**	**0 %**
Gdb :	**100 %**	**0 %**
Programme :	**100 %**	**0 %**
Compte rendu :	**100 %**	**0 %**
Makefiles :	**100 %**	**0 %**

J'ai donc travaillé durant ce projet tout seul, me compliquant la tâche vu le temps initialement prévu pour un groupe de plusieurs personnes.

La seule chose que je pourrai ajouter c'est l'élaboration d'un arbre de recherche pour optimiser le traitement un maximum notamment sur des grilles plus importantes à ce niveau je ne vois pas d'autres améliorations à apporter.

Néanmoins grâce à ce projet j'ai pu faire face à la mise en place d'un vrai projet de A à Z, sa structuration, les problèmes rencontrés tels que les oublis d'instructions, la documentation et bien entendu le résultat final.

Pour voir le **git** créé pour ce projet :

https://github.com/DanAurea/Ruzzle