Ewen CHAUDEMANCHE
Bastien Bodineau

Ruzzle Solver

Explication :

La version initiale :

Le programme parcourt la grille entière, pour chaque lettre on ouvre le dictionnaire correspondant, le dictionnaire ayant été au élaguer pour retirer les mots contenant des lettres ne se trouvant pas dans la grille.
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().
Ce traitement est long car le nombre de chemin possible sur la grille est de :

$$\frac{16!\times15!\times14!\times13!\times12!\times11!\times10!\times9!\times8!\times7!\times6!\times5!\times4!\times3!\times2!}{16!+15!+14!+13!+12!+11!+10!+9!+8!+7!+6!+5!+4!+3!+2!}$$ possibilités

La version finale :

Afin de palier au défaut de la version initiale, une seconde version a été réalisée.
La grille est initialisé avec les données entrées par l'utilisateur soit en argument du programme, soit en les saisissant lorsque cela lui est demander.
Une fois la grille complète le solver est lancé, le module pioche un mot dans le dictionnaire, 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 le score engrangé par le mot ainsi que les bonus qui sont utilisés
Tout ce traitement étant réalisé par findWord().
Si la totalité des lettres du mot on été trouvées, on ajoute le mot et son score dans une liste doublement chainée.
Finalement on affiche la liste avec printList() dans le programme principal (ruzzle.c).

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 ~600-900 ms alors que la version initiale le faisait en ~2min – 25 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.
Une autre branche à alors été créer pour la mise en place de la documentation et des tests unitaire

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

Élève :	Chaudemanche Ewen	Bastien Bodineau
README.md :	60 %	40 %
Documentation :	60 %	40 %
Git :	90 %	10 %
Test Unitaires	0 %	100 %
GDB*	0 %	0 %
Programme	75 %	25 %
Compte Rendu	95 %	5 %
Makefile**	40 %	60 %

* GDB n'a pas été utilisé en l'état durant ce projet. Cependant, le debugeur de CodeBlocks lui à tourné à plein régime

** Les Makefiles originaux ont été fourni par Bastien Bodineau et on été remplacé par des makefile générer à partir du projet CodeBlocks.

Le projet a été mené à bien, malgrés les difficulté à utiliser git pour une grande moitié du projet puisque Bastien apprenait a l'utiliser pendant la durée du projet.