Rapport de mini-projet de Base de l'Informatique Industrielle

Projet Booggle

Rappel du principe de ce projet :

Développer un programme en langage C qui permet de rechercher les mots dans une grille de jeu Boggle, qui s'appelle Booggle.

Nous devons d'abord créer une table 4 * 4 avec une lettre aléatoire dans chacune des cases. Les joueurs doivent rechercher dans cette matrice un maximum de mots (présents dans un dictionnaire de référence) obtenus par juxtaposition de cellules adjacentes (horizontalement, verticalement et en diagonale) de la matrice 1 et les inscrire sur leur feuille. La longueur du mot trouvé est différente, et le score du joueur est également différent.

Taille du mot	3	4	5	6	7	8 et plus
Points	1	1	2	3	5	11

Les règles sont les suivantes:

1. Un mot est valable s'il appartient au dictionnaire de référence.
2. Un mot est valable s'il est composé d'au moins 3 lettres.
3. Les accents et la casse des lettres ne compte pas.
4. Les noms propres, les noms étrangers, les abréviations, les mots composés avec un trait d'union ou apostrophe ne sont pas autorisés.
5. Les mots féminins, les pluriels et les conjugaisons sont autorisés.
6. Les réductions sont autorisées (ex : danse, dans et ans).
7. Seuls comptent les mots trouvés par un seul joueur.
8. Un mot apparaissant plusieurs fois ou ayant plusieurs significations ne compte qu'une fois.
9. Le nombre de points d'un mot est une fonction croissante de sa longueur.

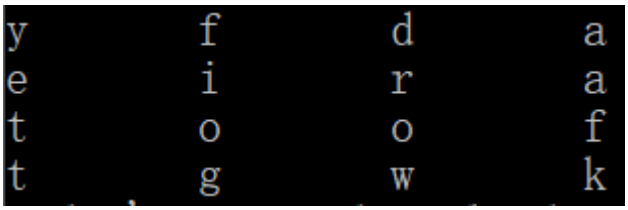
Réaliser

J'ai divisé ce projet en 4 parties:

1. Construire la grille
2. Recherche de dictionnaire
3. Scores statistiques
4. Recherchez toutes les possibilités dans la grille

Pour les trois premières parties, j'ai créé les fichiers .h et .cpp correspondants pour chaque partie. Les fichiers .h contient des déclarations de fonction, des définitions de structure, des définitions de constante et des références de fichier d'en-tête, les fichiers .cpp contient des fonctions utiles.

Pour la première partie, je définis une structure appelée grille_t et une fonction constructeur utilisé pour créer cette grille. Je génère aléatoirement des lettres en générant des nombres aléatoires de 0 à 25 et stockez-les dans une table de format char **.



Pour la deuxième partie, j'ai défini une structure 'mots' utilisé pour stocker les mots pour comparer avec le dictionnaire. En même temps, j'utilise la fonction 'comp2' pour ouvrir le dictionnaire et utiliser ‘strstr’ pour comparer avec le dictionnaire. S'il n'y a pas de chaîne correspondante dans le dictionnaire, 1 est défini sur myword.is_terminal pour quitter la comparaison. Sinon, une comparaison de longueur de chaîne est effectuée pour déterminer si le mot existe dans le dictionnaire. Si le mot existe dans le dictionnaire, vous n'avez pas besoin de continuer avec la comparaison, return isword = 1 en dehors de la boucle, sinon isword = 0. Ensuite, j'utilise "fonction compaire" pour déterminer s'il faut imprimer le mot (utilisé pour les tests), et j'utilise cette "fonction" pour retourner les mots qui peuvent être interrogés dans le dictionnaire.

Pour tester, j'ai testé la partie notation avec la partie dictionnaire ensemble.

Pour la troisième partie, j'ai conçu la fonction "notefunction ()" pour déterminer si le mot a été utilisé et pour calculer le score final. Selon les règles, les mots dont le nombre delettre est inférieur à 3 n'est pas compté, mais pour tester la seconde partie, je produis également ces mots avec une longueur inférieure à 3, mais pas de points.

Le programme continuera à récupérer les mots et à les comparer jusqu'à ce que l'utilisateur tape « quit »". Donc, le mot « quit » ne compte pas comme un mot valide.

Si le mot saisi ne figure pas dans le dictionnaire, imprimez le et imprimez "erreur". Si le mot peut être trouvé dans le dictionnaire et que le mot n'est pas réutilisé, imprimez « answer xx yes » et imprimez votre score. Ce dernier "used" est pour voir si le mot a été utilisé. Si used est 0, il n'est pas utilisé, sinon le mot a déjà été utilisé.

```
your word
nation
answer nation yes,your point is:3,used is 0
point
answer point yes,your point is:5,used is 0
groupe
answer groupe yes,your point is:8,used is 0
answer
answer erreur
si
answer si yes,your point is:8,used is 0
oui
answer oui yes,your point is:9,used is 0
aa
aa erreur
no
answer no yes,your point is:9,used is 0
et
answer et yes,your point is:9,used is 0
or
answer or yes,your point is:9,used is 0
ca
answer ca yes,your point is:9,used is 0
```

Si le mot a déjà été utilisé, imprimez « your word has been used...please try a new word ».

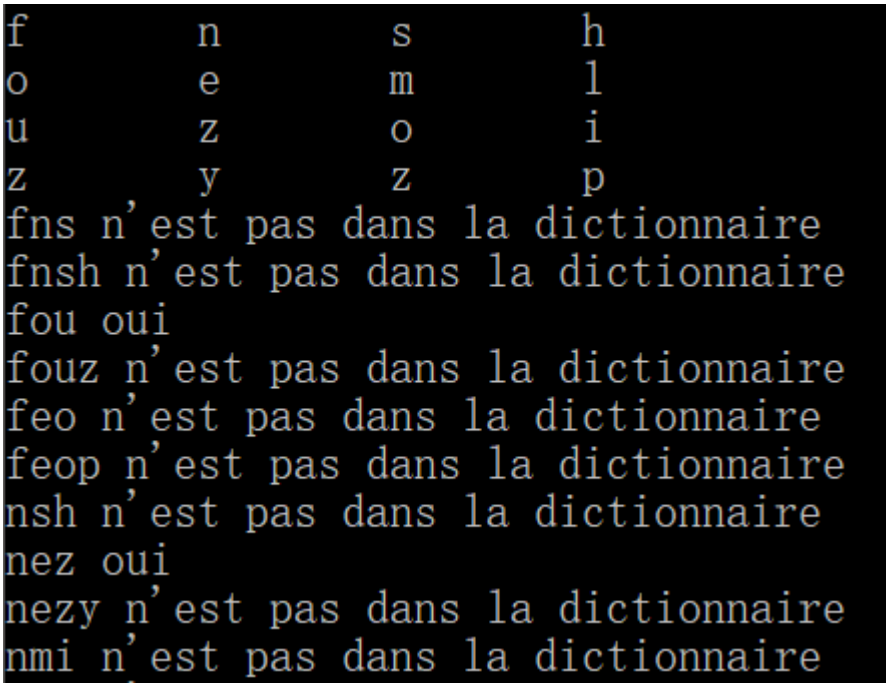
```
your word
nation
answer nation yes,your point is:3,used is 0
nation
your word has been used...please try a new word, used is 1
nation
your word has been used...please try a new word, used is 1
```

Si vous voyez le résultat imprimé est « can't open » alors le fichier du dictionnaire n'est pas dans le bon chemin.

```
w       f       q       b
l       v       r       g
i       f       q       l
e       w       i       y
can't open
```

Afin de trouver tous les mots possibles dans la grille, j'ai défini une structure dans le fichier noeud.h, qui s'appelle retourer. Lorsque retourner est utilisé pour renvoyer une série de chaînes, cette structure contient la valeur de la chaîne et le nombre de chaînes. Const int DX[ ], const int DY[ ] sont utilisé pour définir 8 directions. La fonction Search est utilisé pour déterminer quelle lettre dans la grille pour commencer. La fonction boucle utilise l'algorithme DFS pour rechercher tous les arrangements alphabétiques possibles dans la grille. Malheureusement, je ne sais pas comment utiliser l'algorithme DFS. Mon programme ne

fonctionne pas comme prévu après avoir rejoint la fonction de boucle. Maintenant, mon programme ne peut récupérer des mots que dans 8 directions et ne peut pas utiliser l'algorithme DFS.



Fonctionnement normal

Ce programme prend beaucoup de temps dans la recherche du dictionnaire en raison de la possibilité de considérer l'arrangement de toutes les lettres. Normalement, ce programme peut imprimer toutes les dispositions des lettres, et juger si elle apparaît dans le dictionnaire, puis entrez la réponse dans le fichier answer.txt. Ensuite, le joueur entre les mots saisis dans le clavier et ce programme les compare avec les réponses dans le fichier answer.txt. Si la réponse correspond à answer.txt, il entre dans le module de notation et le score final du joueur est compté jusqu'à ce que le joueur tape "quit".

Sommaire

Le cœur de ce programme est d'utiliser l'algorithme DFS. Après avoir fait les derniers changements, j'ai rendu l'affichage de "grille" plus beau. Et au dernier moment, j'ai trouvé une solution à l'algorithme DFS. Mais malheureusement, je n'ai pas assez de temps pour optimiser ce programme. Je présente une capture d'écran ici pour montrer le processus de ma recherche terminée. Cette section a été commentée dans le programme final. Ceci est fait par la 87ème ligne de code dans le fichier noeud.cpp. Mais pour rendre le programme plus concis, j'ai fini par le commenter. En outre, dans le processus d'interrogation du dictionnaire, j'ai profité de la structure arborescente. Afin de simplifier le temps de fonctionnement du programme, j'ai défini

le drapeau de recherche de fin : is_terminal. S'il y a plus de temps, je vais le finir mieux.