IN104

Projet Wordle

Thibault MOUGIN

12/05/2022

1 Objectif et fonctionnalités

On cherche initialement à programmer en C et en console le jeu Wordle (cf `https://www.nytimes.com/games/wordle/index.html`) avec pour fonctionnalités le choix du nombre de lettres du mot, le nombre d'essais, la langue. Ceci est réalisable à l'aide du fichier *setup.c* à l'aide d'un dictionnaire sous format txt fourni initialement : on utilisera cependant tout le long un dictionnaire français, des mots de 5 lettres et 6 essais au maximum.
On privilégiera plutôt la mise en place du jeu dans la console avec la possibilité d'activer un algorithme de résolution que l'on surnommera "IA" (pour intelligence artificielle... même si il n'en est rien?..). Après différentes tentatives d'algorithmes de résolution, on finira par combiner un choix basé sur la fréquence des lettres des mots éligibles à être le prochain essai (pour les 3 premiers essais) et un algorithme cherchant à limiter à l'avance le nombre de mots possibles pour les 3 essais suivants.

2 Architecture et structures

Le programme se divise en 3 fichiers c : *wordle.c*, le "main" que l'on compile ; *ai.c*, regroupant les fonctions déterminant le meilleur mot à proposer à chaque essai (au sens qu'il garantit que l'on trouve le mot le plus vite, dans l'idéal en moins de 6 essais) ; et le fichier *utils.c*, où sont définies toutes le fonctions dites "utilitaires" : longueur d'un mot, accès au dictionnaire, vérification de la présence d'une lettre dans un mot, d'un mot dans le dictionnaire, fonction de suppression des accents dans un chaine de caractères, écriture dans la console du bon placement des lettres des essais... Pour obtenir les fichiers texte stockant les mots, on a créé un fichier *setup.c* contenant les fonctions nécessaires au choix des mots que l'on cherche à deviner dans le jeu
Les mots que l'on utilise sont ceux du fichier *dicotrie.txt*, obtenu à l'aide d'un dictionnaire français que l'on a trié grâce à la fonction *tridico* du fichier *setup.c* pour ne garder que ceux de 5 lettres. Puisque Wordle supprime tous les accents présents dans les mots, à chaque fois qu'on lis un mot du dictionnaire pour le comparer avec un essai, on supprime les accents de ces mots à l'aide la bibliothèque *unac*. Au vu de la quantité de caractères spéciaux présents dans les mots du dictionnaire, on a préféré cela à une fonction "faite main".
Le jeu se joue dans la console : l'utilisateur indique par "o" ou "n" si il souhaite être aidé par l'IA puis on choisit au hasard un nombre r entre 1 et 6145 (la longueur du dictionnaire), on définit la solution comme étant le r-ième mot du dictionnaire. *scanf* nous fournit l'essai de l'utilisateur dont on vérifie la longueur à l'aide de la fonction *lon* (facilement paramétrable mais on ne jouera qu'avec des mots de 5 lettres) et on s'assure qu'il est bien dans le dictionnaire à l'aide de la fonction *verifdico* (retourne un booléen), qui est simplement une recherche linéaire dans *dicotrie.txt* (un seul appel par essai donc une dichotomie n'apporte pas grand chose). Si l'essai est le mot recherché, le jeu est terminé, sinon on appelle la fonction *hint* (puis l'IA si celle-ci est activée) qui prend un essai et la solution pour retourner une chaîne de caractères de la forme par exemple *"xX..x"*, les *"."* étant les cases grises du jeu Wordle, les *"x"* les cases oranges et les *"X"* les cases vertes.

3 Principe de l'IA

On cherche à déterminer le mot choisi au hasard en moins de 6 essais. Pour ce faire, on crée une structure *pos* destinée à stocker les mots possibles et leur nombre après chaque essai d'après l'indice fourni par *hint*.

```
1 struct pos{
2     int* possibles;
3     int size;
4 };
```

Par soucis de rapidité, on stocke dans la liste *possibles* les indices des mots dans le dictionnaire (auquels on accède facilement grâce à la fonction *readnthline*). Aussi, la taile *size* de cette liste est une valeur que l'on cherche à faire diminuer à chaque essai. Ainsi, on cherche dans les trois premiers essais à réduire la taille de cette liste en obtenant un maximum d'informations sur le mot recherché par une analyse de la fréquences des lettres parmi les mots possibles. On crée alors une fonction *freqscore* qui pour un mot donné renvoie :

$$\sum_{i=1}^{5} \text{freq(mot[i])/nb d'apparitions(mot[i])}$$

où la fréquence d'une lettre est celle de son apparition en français (en %, arrondie à l'unité). On choisit alors le mot possible ayant le plus grand score car il est susceptible de nous donner le plus d'informations. Cette méthode nous fournit également le premier mot à essayer pour l'IA qui est *"saine"*. Evidemment, à chaque essai, on supprime de la liste notre essai, pour garantir la stricte décroissance de sa taille. Au bout de 3 essais, on a généralement une liste de mots possibles de taille inférieur à 800, ce qui nous permet d'appliquer la deuxième partie de l'IA.

Pour les 3 essais suivants, on parcourt la liste des mots possibles pour supposer que chaque mot est la solution. A partir de chaque mot, on détermine quel serait la taille de la liste des mots possibles pour tous les essais possibles (d'après l'indice fourni par *hint*). On choisit le mot pour lequel cette taille est minimale dans le pire des cas, ie. pour l'essai diminuant le moins la taille de *possibles*.

Exemple (cas où la taille de la liste des mots possibles est de 3) :

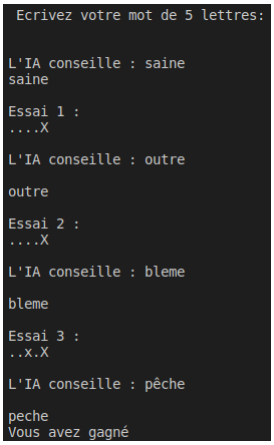
Essai / Sol° supposée	possibles[0]	possibles[1]	possibles[2]
possibles[0]	1	2	3
possibles[1]	2	1	3
possibles[2]	3	2	1

Si la solution était *possibles[1]* et qu'on essayait *possibles[0]*, on aurait ensuite 2 mots encore possibles. Si on essaie ce mot et que la solution était possibles[2], 3 mots seraient encore envisageable : c'est le pire des cas pour cet essai. Le mot à essayer que notre algorithme retournera sera alors *possibles[1]* car dans le pire des cas, on aura seulement 2 mots encore possibles contre 3 pour les autres essais.

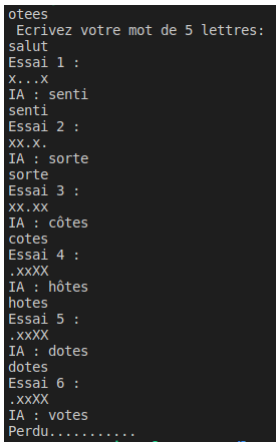
Cet algorithme est en $O(n^2)$, où n est la taille de la liste des mots possibles, c'est pour cette raison qu'on cherche dans un premier temps à diminuer n. En effet, pour ~ 800 mots il prend déjà 1-2 secondes, alors pour 6145...

4 Résultats et pistes d'améliorations

L'analyse fréquentielle des lettres permet environ 1 fois sur 3 de trouver la solution en 3-4 essais :



On trouve le bon mot en moins de 6 essais environ 90% du temps (sur 100 parties jouées par l'IA). La stricte décroissante de la taille de *possibles* garantit toujours la réussite de l'IA mais on dépasse dans certains cas les 6 essais, ici avec *ôtées* comme solution, (l'IA conseille *otees* à l'essai 7) :



Cet échec est essentiellement dû au fait que l'on se retrouve avec tous les mots qui s'écrivent en *"-otes"* dans la liste des mots possibles, ce qui naturellement nous fait perdre plusieurs essais. Une façon efficace d'empêcher cela d'arriver est d'appliquer la deuxième partie de notre IA récursivement, c'est-à-dire ne pas se contenter de calculer la taille de *possibles* après un seul essai. De cette manière, on se serait rendu compte que jouer *senti* nous mène à avoir à considérer tous les mots en *"-otes"*, qui sont trop nombreux.