Rapport d'IN104

Projet Wordle

Mahé de La Villemarqué - Maxime Bouvrot

I. Présentation

Ce projet consistait à coder le jeu Wordle en C. Le projet s'articulait en 2 étapes, d'abord coder le jeu, puis coder un programme faisant jouer l'ordinateur de manière optimale.

II. Structure du programme

Le Jeu

Voici les étapes codées dans le main du fichier jeu.c:

- Ouvrir le dico
- Déterminer un nombre aléatoire nb_aléatoire compris entre 0 et le nombre de ligne du dico.
- Choisir un mot aléatoire de 5 lettres dans le dico qu'on appellera mot mystère (le mot sur la ligne nb_aléatoire. Si celui-ci ne fait pas 5 lettres, descendre à la ligne suivante jusqu'à tomber sur un mot de 5 lettres.)
- Tant que l'utilisateur n'a pas épuisé ses 6 essais ou qu'il n'a pas trouvé le mot mystère, demander à l'utilisateur de saisir un mot de 5 lettres.
- Vérifier que le mot est bien dans le dico et fait bien 5 lettres.
- Effectuer la comparaison entre le mot mystère et le mot utilisateur.

La comparaison s'effectue selon le schéma suivant :

On créé un tableau de 5 entiers initialisés à 0.

On commence par traiter les lettres vertes. On créé une copie du mot dans laquelle les lettres vertes sont remplacées par des espaces, qu'on nomme mot_jaune (il n'y a plus que des espaces ou des lettres jaunes ou noires dedans).

On traite ensuite les lettres jaunes. On créé une copie du mot_jaune où les lettres vertes sont remplacées par des espaces, qu'on nomme mot_noir.

On complète pour chaque étape le tableau : 2 si vert, 1 si jaune, 0 si noir.

- Afficher le résultat. Par exemple si l'utilisateur rentre le mot "aimer" :

  A lettre jaune

  I lettre verte

  M lettre noire

  E lettre noire

  R lettre noire

Le programme s'arrête si l'utilisateur a épuisé ses 6 essais ou s'il a trouvé le mot mystère.

L'ordinateur optimal

L'ordinateur marche de manière similaire excepté au niveau du mot donné par l'utilisateur.

Lorsque l'ordinateur doit rentrer un mot, il a à sa disposition le tableau des comparaisons et l'ancien mot donné et peut donc en déduire le coup optimal à jouer.

Le but était de partir d'un ordinateur relativement basique pour aboutir à un ordinateur de plus en plus intelligent.

Le premier ordinateur (bot_1) renvoyait juste un mot aléatoire au hasard (et n'aboutissait jamais car il y a + de 60000 mots dans notre dictionnaire.

Le deuxième ordinateur (bot_2) utilisait la structure qui suit :

Entrée : le tableau des comparaisons, "mot_vert" qui sert à stocker les lettres vertes et leur position", "mot_utilisateur" qui sert à connaitre l'ancien mot donné par l'ordinateur et le dictionnaire.

Le tableau des comparaisons est utilisé pour modifier mot_vert et ajouter les nouvelles lettres vertes trouvées

On parcourait ensuite le dictionnaire pour trouver un mot compatible qui était après renvoyé par l'ordinateur.

Une troisième version de l'ordinateur (bot_3) est de partir d'un mot optimal en anglais "soare", pour trouver au plus vite une lettre verte puis on traite comme avec le bot 2.

Enfin, la version 4 (bot_4) prend aussi en compte les lettres noires en créant un "mot_noir" composé de toutes les lettres noires.

Pistes d'amélioration

Nous ne sommes pas parvenus à traiter efficacement les lettres jaunes dans nos bots. La complexité du traitement des lettres jaunes est en effet supérieure, car il y a beaucoup de cas de figure, et cela s'ajoute aux vérifications des lettres vertes et noires. On aboutit à une succession importante de test avant de choisir le bon mot. Notre programme d'ordinateur serait pourtant bien meilleur s'il parvenait à choisir le mot optimal en prenant en compte les lettres jaunes.

III. Choix, difficultés, optimisation

Les premières difficultés sont apparues avec les ouvertures fermetures successives du dico. Nous avons dû bien comprendre les fonction fclose, feof et rewind. Mais le gros du programme réside dans la fonction comparaison qui réalise la comparaison entre le mot_utilisateur entré et le mot_mystere à deviner. Plusieurs stratégies sont possibles, et on a commencé par n'identifier que les lettres vertes et noires, ce qui revient à effectuer un test binaire. Nous nous sommes ensuite intéressés aux lettres jaunes, et il a fallu modifier toute la structure du programme. La question "Par où commencer ?" s'est alors posée. Nous avons fait deux choix structurants : traiter les lettres vertes, puis jaunes, puis noires, et traiter le problème lettre par lettre. Une fois la lettre traitée, celle-ci était retirée du mot et remplacée par un espace. Une fois toutes les lettres vertes retirées, on avait un nouveau mot facile à utiliser, de même longueur que le mot de base par souci de simplicité et où il n'y avait plus que des lettres noires et jaunes. Nous avons dû jouer avec différents types (int, char, char*,…) et nous avons parfois mis du temps à comprendre nos erreurs qui venaient souvent d'une mauvaise définition des types de variable.

Puis, au cours de nos tests, nous nous sommes rendu compte que le programme n'arrivait pas à faire la comparaison lorsque des lettres apparaissaient plusieurs fois dans le mot. Il a donc été décidé d'utiliser un tableau de 5 entiers remplis de la forme suivante : 0 si lettre noire, 1 si lettre jaune, 2 si lettre verte. En plus de nous permettre de détecter si une lettre double était verte ou jaune ou noire, cette manière de faire était visuelle et fut très pratique pour nos bots qui avaient là toutes les informations faciles d'accès.

Après l'implémentation de plusieurs bots, nous avons remarqué des erreurs récurrentes "CORE DUMPED segmentation fault" très problématiques. Nous avons alors sillonné notre programme pour découvrir des problèmes de free et malloc dus à des copies de mots définies par malloc.

Optimisation

Afin de prendre en compte la spécificité de chaque dictionnaire, nous avons implémenté une fonction qui va chercher le nombre d'occurrence de chaque lettre dans le dictionnaire, et nous donner les 5 lettres les plus fréquentes. On définit ensuite le mot optimal comme celui composé de ces 5 lettres. Ce n'est pas important qu'il existe dans le dictionnaire, car il sert à éliminer le plus de mot possible en un essai.