IN104

Rapport Projet Wordle

Aymeric Lévy-Dauchez Julien Namazi

Mai 2022

Table des matières

1 Introduction

L'objectif de ce projet est d'une part, d'implémenter le jeu Wordle et d'autre part d'essayer d'établir une stratégie de jeu qui permette à l'ordinateur de jouer le mot optimal à chaque tentative en s'aidant de la théorie de l'information.

2 Implémentation de Wordle en C

Dans cette partie, nous détaillons les différentes étapes qui ont jalonné notre implémentation du jeu.

2.1 Comparaison du mot entré par le joueur avec le mot à trouver

La première étape de l'implémentation est d'être capable de comparer le mot à trouver avec le mot entré par le joueur et de lui indiquer dans la console son résultat. Fixant un caractère du mot saisi par le joueur, on regarde si on retrouve ce caractère dans le mot à deviner et on stocke les indices de coïncidence dans un tableau.

S'il y a au moins une coïncidence entre le caractère considéré et le mot à deviner, alors il faut déterminer si ce caractère est bien placé.

Pour ce faire, on va tout d'abord chercher dans le tableau de mémoire si l'indice de la lettre considérée a été gardée en mémoire. Si c'est le cas, cette lettre est bien placée!

Les cas pathologiques apparaissent alors : il s'agit des mots contenant plusieurs fois la même lettre. Pour pouvoir traiter cela, il est essentiel d'avoir une fonction qui dénombre les occurrences d'une lettre dans le mot saisi et le mot cherché, c'est le travail de la fonction occurrence.

Nous avons ensuite construit notre code en réfléchissant à partir d'exemples, pour aboutir à la formulation suivante. Si la lettre considérée n'est pas bien placée, quatre possibilités sont à envisager :

- La lettre en question n'est pas dans le mot : ce cas est traité à part, il s'agit simplement du cas où le nombre de coïncidences k est nul.

- La lettre en question apparaît autant de fois dans le mot saisi que dans le mot à trouver : dans ce cas, elle est nécessairement mal placée.

- La lettre en question apparaît plus de fois dans le mot saisi que dans le mot à trouver. Dans ce cas, on a un problème : il faut pouvoir afficher que l'une des deux lettres est incorrecte et l'autre mal placée pour que le joueur ne se méprenne pas sur le nombre d'occurrences dans le mot à deviner. Afin de traiter ce problème, nous affectons le caractère '0' à la place de cette lettre dans le mot saisi par le joueur, et affichons que la lettre est placée, ceci tant que le nombre d'occurrences de cette lettre dans le mot saisi n'est pas égal au nombre d'occurrence de cette lettre dans le mot à deviner. Ainsi le nombre d'occurrences de cette lettre dans le mot saisi est voué à diminuer jusqu'à égaler le nombre d'occurrences de la lettre dans le mot à trouver. Par conséquent, les lettres mal placées seront toujours affichées en fin de mot, ce qui n'est pas gênant.

-La lettre en question apparaît moins de fois dans le mot saisi que dans le mot à trouver : pas de problème, la lettre est simplement mal placée.

Nous avons testé notre algorithme de jeu sur des parties que nous avions effectuées sur Wordle, et les résultats se sont avérés concluants.

On fournit ci-dessous un schéma de l'architecture de l'algorithme de jeu, illustrant de manière visuelle le texte qui précède.

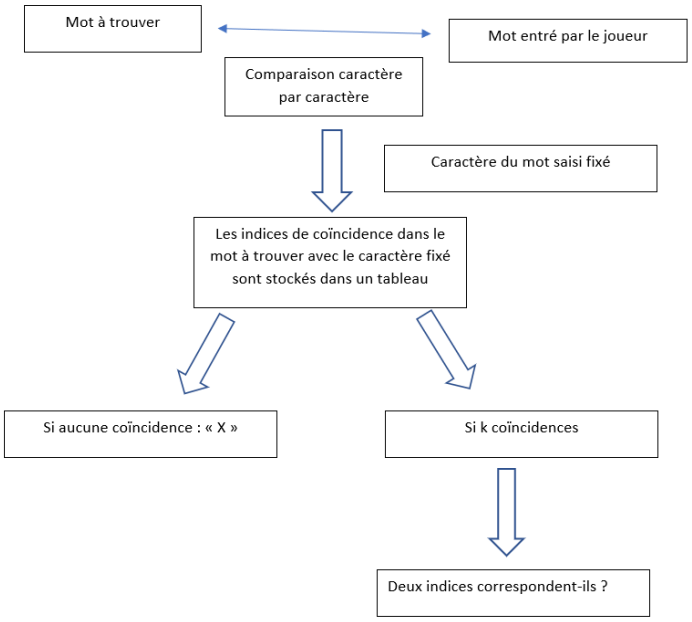

FIGURE 1 – Structure du code de jeu Wordle

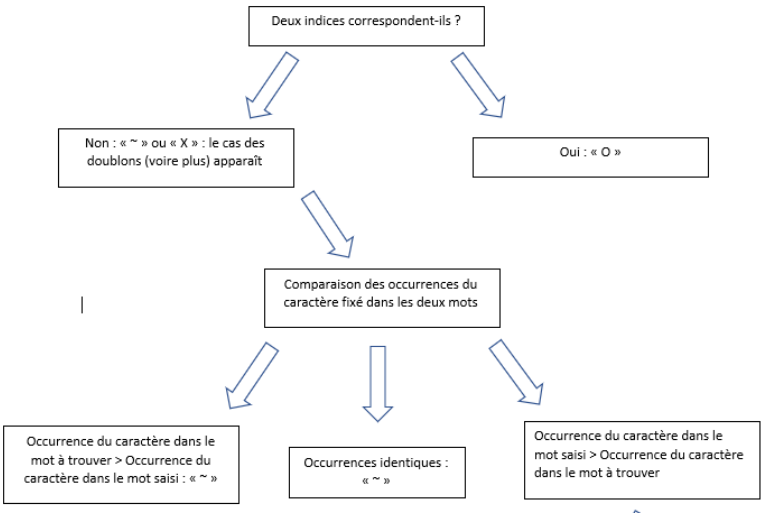

FIGURE 2 – Structure du code de jeu Wordle

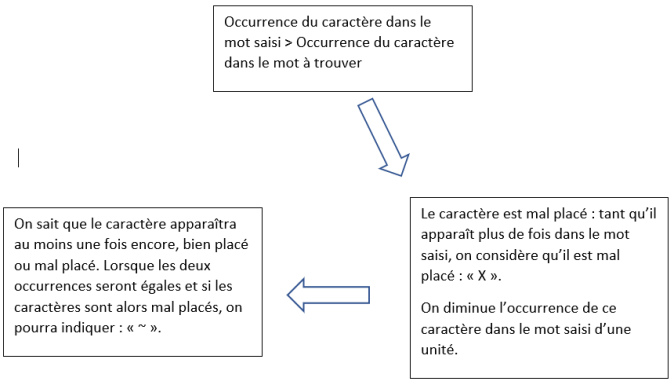

FIGURE 3 – Structure du code de jeu Wordle

2.2 Chargement du dictionnaire et choix du mot à deviner

A présent que l'on peut jouer à Wordle, il faut pouvoir piocher dans une base de données pour choisir le mot à deviner.

La première étape est donc de choisir le dictionnaire où l'on sélectionne les mots à trouver. Clairement, il ne faut pas que celui-ci contienne un nombre trop important de mots, sous peine d'essayer de faire deviner au joueur des mots qu'il ne connaît pas.

Dans un premier temps, nous avions imaginé récupérer plusieurs romans, articles, ... et faire des statistiques sur la fréquence d'apparition des mots. Au-delà d'un seuil fréquentiel arbitraire, nous aurions pu constituer notre dictionnaire avec des mots raisonnablement communs. Cependant, il aurait fallu utiliser une grande variété de textes pour éviter une sur-représentativité de certains termes liée au sujet des textes choisis, et cela aurait donc demandé un temps de calcul et de recherche considérable.

Nous nous sommes donc limités à un dictionnaire pour enfants contenant 835 mots basiques.

Dans le fichier dico.c, nous avons alors écrit deux fonctions : la première permet de lire ce dictionnaire pour le transformer en un tableau de chaînes de caractères ; et la deuxième permet de sélectionner un mot de façon aléatoire dans ce tableau.

2.3 Vérification de la validité du mot saisi par le joueur

Pour finir, il faut s'assurer que le mot entré par le joueur est correct dans la langue de jeu, pour restreindre un minimum les possibilités de jeu. Pour cela, nous avons opté pour une recherche dichotomique dans le tableau définir par la fonction de lecture du fichier. En effet, si celui-ci est trié par ordre alphabétique, la recherche dichotomique a alors une complexité temporelle meilleure (en $O(log_2(n))$ qu'une recherche séquentielle (en $O(n)$).

Toutefois, nous ne pouvons utiliser comme dictionnaire de vérification le dictionnaire pour enfants utilisé pour choisir le mot à trouver, car cela restreindrait de façon trop importante les possibilités de jeu. Nous avons alors opté pour un dictionnaire de Scrabble, contenant 386264 mots, constituant par conséquent une base de données plus que raisonnable pour la validation d'un mot.

Nous avons rencontré dans cette partie plusieurs difficultés liées aux dictionnaires : ceux-ci n'étaient en effet pas triés par ordre alphabétique, et contenaient des lettres majuscules.

2.4 Synthèse

Le principe des algorithmes nous est venu relativement rapidement, en revanche nous avons manqué de rapidité dans l'implémentation, notamment à cause d'erreurs de gestion de la mémoire.

Par conséquent, malgré le temps considérable passé sur cette partie, nous nous avons très certainement amélioré notre compréhension des mécanismes de gestion de la mémoire, des manières optimales de débugger et des erreurs fréquentes à repérer rapidement.

3 Implémentation d'une intelligence artificielle pour Wordle

3.1 Explication de la démarche

3.1.1 Contexte

Maintenant que nous avons codé le jeu Wordle en C, il est légitime de se demander s'il est possible de coder une intelligence artificielle sur Wordle. Mais que ferait concrètement une IA sur ce jeu ? Elle devrait être capable de jouer seule au jeu et de trouver le mot à déterminer en 5 ou 6 étapes. Pour coder cela, il va donc falloir dans un premier temps déterminer un processus pour chercher un mot dans le dictionnaire. On pourrait penser à prendre un mot aléatoire du dictionnaire mais évidemment nos chances que l'IA trouvent le bon mot seraient quasiment nulles.

Ainsi il va falloir trouver une propriété discriminante sur chaque mot du dictionnaire pour pouvoir déterminer s'il est pertinent de jouer un mot plutôt qu'un autre. On va introduire le concept d'entropie.

3.1.2 Notion d'entropie

Disons que le mot à deviner soit $chien$ et que nous décidons de jouer le mot $wagon$. Évidemment c'est une très mauvaise stratégie si l'on veut gagner en un nombre de coups minimal, mais qu'est-ce qui la rend mauvaise ? w est une lettre très rare (0.17 % des mots de notre langue contiennent cette lettre), on a donc peu de chance de la retrouver dans le mot à deviner. La lettre g au milieu d'un mot n'est pas chose commune non plus.Dans le cas du mot chien (mot très commun qui serait typiquement proposé comme mot aléatoire par Wordle) on aurait 4 cases grises et une case orange à la fin. Si on comptait le nombre de mots qui contiennent un n comme dernière lettre et pas $wago$ avant, cela ne nous réduirait que très peu le nombre de mots que nous pourrions jouer.

Au vu de l'exemple précédent, on comprend qu'il est important de trouver les mots qui conduiront à un nombre de mots encore possibles après la saisie qui soit minimal. Notons N le nombre de mots parmi une liste de mots données (on verra plus tard que cette liste prend en compte tous les résultats des coups précédents) et T_m le nombre total de mots qu'il sera possible de jouer après avoir jouer le mot m, en respectant les informations fournies par Wordle.

On note :

$$p_m = \frac{T_m}{N}$$

Ainsi, plus ce rapport p_m est important, moins il sera pertinent de jouer le mot associé car cela signifierait que le mot m nous conduit à un grand nombre de possibilités de mots encore possibles. On note E_mot l'entropie d'un mot parmi une liste de mots, et on écrit :

$$E_{mot} = \sum_{m=1}^{3^5} -p_m log_2(p_m)$$

Quelle est la signification de cette somme ? Prenons un mot d'une liste de mots (qui au premier coup sera notre

dictionnaire). En considérant des mots de 5 lettres, après avoir joué ce mot il existe 3^5 résultats possibles : chaque résultat est une combinaison des couleurs grises, oranges ou vertes (en reprenant le code couleur du jeu Wordle). En superposant le mot que l'on a joué et un de ses résultats, cela forme une contrainte sur le mot à deviner au tour suivant. Cette contrainte conduit à un certains nombres de mots qui restent possibles à jouer. Ainsi, pour un mot fixé on calcule pour chacune des 3^5 combinaisons possibles de résultats le nombre de mots qui vérifient la contrainte posée par le résultat obtenu pour ce mot. En passant par un logarithme en base 2 pour déterminer le nombre de bits d'information (une information est un des p_m) que chaque p_m représente, on en déduit par somme l'entropie d'un mot.

3.1.3 Détermination du mot à jouer à chaque tour

Nous avons à notre disposition une liste des mots possibles à jouer (prenant en compte les résultats des tours précédents). Le mot à jouer sera alors celui qui aura l'entropie maximale parmi tous les mots de cette liste de mots. En effet, ce mot sera alors celui qui, en moyenne, conduit à un nombre minimal de possibilités après l'avoir joué. Dans notre code, il va donc falloir écrire un algorithme qui à partir d'une liste de mots nous renvoie le mot qui a l'entropie maximale.

3.1.4 Vue d'ensemble de l'algorithme à écrire

Le programme détermine le mot aléatoire que l'IA va devoir deviner. Au premier tour cette dernière a à sa disposition le dictionnaire français. Elle va alors déterminer le mot d'entropie maximale parmi ce dictionnaire (le premier mot joué par l'IA sera alors le même à chaque partie). Ce mot est le premier coup de l'IA. En utilisant le code précédent (Wordle pour un joueur humain) on rentre le coup de l'IA comme si c'était celui de l'humain et le programme nous renvoie le résultat (une des 3^5 combinaisons de couleurs possibles (remplacées ici par des X, $\sim$ou O).

A partir de ce résultat l'IA calcule le nouvel ensemble de mots possibles en prenant en compte ce premier tour. Par exemple si on a une croix à la place d'une lettre a notre intelligence artificielle va éliminer tout les mots qui contiennent un a à cette position (ou éliminer tous les mots qui contiennent la lettre a si cette dernière n'a qu'une seule occurrence dans le mot). Parmi ce nouvel ensemble de mots, notre IA recalcule de nouveau le mot d'entropie maximale. Ce mot sera celui joué par cette dernière, et de nouveau le jeu nous renvoie une combinaison de X, $\sim$ou O. On recommence cela jusqu'à ce que le bon mot soit trouvé (le résultat est alors $OOOOO$) ou alors que le nombre d'étapes maximales soit dépassé.

3.2 Les différentes étapes de l'implémentation

3.2.1 Obtenir les 3^5 combinaisons possibles de résultats

On aimerait obtenir un tableau dont chaque élément est une des 3^5 combinaisons de X, $\sim$, ou de O. On sait qu'on a unicité de l'écriture en base 3 des entiers en base 10. En associant 0 au symbole X, 1 au symbole $\sim$et 2 au symbole O on exprime chacun des entiers de 1 à $243 = 3^5$ en base 3. A chacune de ces écritures en base 3 on associe alors une unique écriture en terme symboles.
Par exemple l'entier 226 s'écrit 22101 en base 3 et donc s'écrit $OO\sim X\sim$.

3.2.2 Les différentes structures utilisées

Nous devions ensuite trouver un moyen d'associer à chaque mot une liste d'informations (la liste des p_m) et ensuite d'associer à chaque mot de notre liste de mots son entropie. Pour cela nous avons créer les structures $information$ et $entropies$.

La structure $information$ contient un tableau de double (la liste des informations p_m définies précédemment) ainsi qu'un type char* qui est le mot associé à toutes ces informations.
La structure $entropies$ est une liste chaînée dont chaque maillon contient un mot de la liste de mots (type char*), une entropie associée calculée à partir de la structure $information$ et un pointeur vers le maillon suivant.

3.2.3 Détermination de la nouvelle liste de mots possibles

Après avoir joué un mot, une combinaison de symboles est affichée. Il va falloir déterminer le prochain mot à jouer et pour cela il faut déjà déterminer la nouvelle liste de mots associées à ce résultat précédent.

On compare donc chaque mot de l'ancienne liste de mots avec la combinaison de symboles précédente et le mot joué précédemment (c'est-à-dire le maximum d'entropie de l'ancienne liste de mots). Pour cela on regarde en premier sur chaque lettre verte de l'ancien mot (donc correspondant à un O) si il y a bien exactement la même lettre au même endroit pour chaque mot de la liste. Si oui on passe au test suivant. Si une lettre de l'ancien mot est orange (donc correspondant à un $\sim$) on vérifie si le mot de l'ancienne liste que l'on teste ne contient pas cette lettre exactement à la même position. Enfin on vérifie pour les lettres indiquées en gris (donc correspondant à un X) qu'elles ne sont pas dans le mot de l'ancienne liste sur lequel on effectue tout ces tests, si jamais c'est le cas on regarde si le nombre d'occurrence de cette lettre dans l'ancien mot joué est plus grande que 1. Par exemple, si le mot à deviner est "parer" et que l'IA joue "paree" le deuxième e sera en gris mais pour autant il y a bien la lettre 'e' dans parer.

Si un mot de l'ancienne liste de mots vérifie ces conditions, on peut le rajouter dans la nouvelle liste et on passe aux mots suivants, sinon on passe directement au mot suivant.

3.2.4 Le point clé : déterminer le mot avec la plus grande entropie

Illustrons le processus effectué par l'IA pendant une étape du jeu.

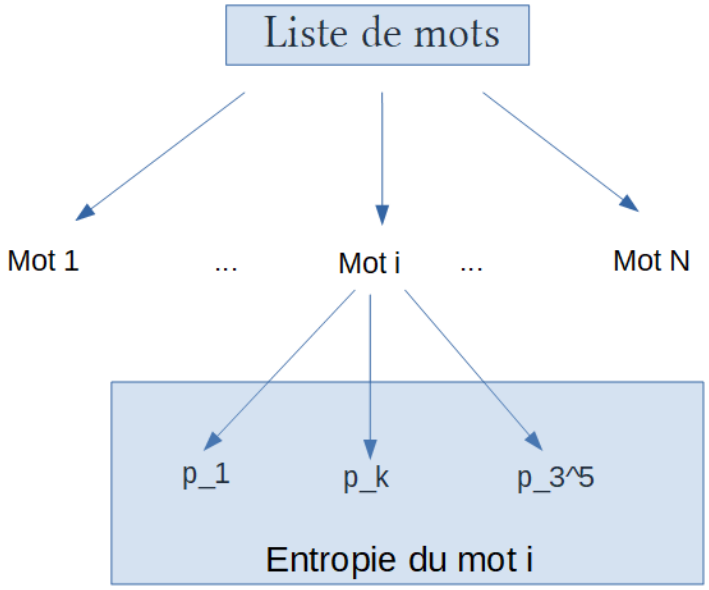

FIGURE 4 – Illustration du principe de l'algorithme

On dispose d'une liste de mots. Pour chaque mot de cette liste on détermine sa liste d'informations associées : l'ensemble des p_k. On a crée pour chaque mot les structures $informations$ associées contenant le tableau des p_k (et le mot associé). A partir de cela on rentre dans chaque maillon de la structure $entropies$ un mot de cette liste et l'entropie associée. Pour terminer on effectue un parcours de cette structure pour déterminer le maximum de chacune des entropies.

Le prochain coup que l'IA joue est le mot qui correspond à l'entropie maximale précédemment déterminée.

3.3 Synthèse

Cette partie de notre projet aura été très intéressante. Nous avons été contraint de développer des tactiques de résolution plus avancés pour pouvoir implémenter un tel programme. Bien sûr cela n'aurait pas été possible sans la

vidéo "Solving Wordle using information theory" de 3Blue1Brown qui nous a donné le cadre théorique de cette étude. La partie intéressante a été la gestion de nouvelles structures pour pouvoir coder un tel problème. Cependant notre programme compile mais au moment de l'éxécution affiche une erreur de segmentation. Nous avons fait tout notre possible pour essayer de résoudre cela en utilisant les fonctions de débuggage de VS Code.

4 Conclusion

Ce projet nous a permis de construire un programme de bout en bout, avec toutes les subtilités et difficultés que cela implique. Nous avons ainsi pu mettre en pratique la théorie et les conseils vus en cours, et faire l'expérience de certains désagréments que nous aurions pu éviter en testant plus régulièrement nos fonctions, en prêtant une grande attention à la gestion de la mémoire, etc ... Ce projet nous a également permis de découvrir la théorie de l'information, concept fort utile pour traiter de tels problèmes.

Au-delà de l'erreur de segmentation qui reste à corriger, de nombreuses améliorations sont possibles. Par exemple, à chaque tour le programme calcule les 3^n combinaisons (où n est la taille du mot à deviner) que l'on pourrait obtenir pour chaque mot joué au tour suivant. En tenant compte des informations obtenues aux tours précédents, on pourrait mieux optimiser le programme d'un point de vue de la complexité temporelle ; on pourrait également, lors du test de la validité d'un mot compte tenu des résultats précédents, vérifier qu'une lettre mal placée se retrouve dans le mot, ce qui réduirait encore plus l'ensemble des mots possibles et donnerait un calcul de l'entropie plus fiable, etc ...