Rapport du Projet d'Informatique :

La Machine à Inventer des Mots
(Fabricationcepteur de Syntrepèteries et Localemboules)

LARREGLE Théo
LEFLOCH Thomas

10/12/2018

Table des matières

1 Introduction

Sur le blog "Sciences étonnantes", un sujet a été abordé à propos d'un algorithme capable de générer des mots aléatoires. Ces mots sont conçus en fonction d'une langue afin que les mots générés aléatoirement y ressemblent le plus possible.
Ici, le but du projet est de créer un programme qui sera capable de générer des mots, ou des phrases simples en langue française ou même toute en une autre langue, en se basant uniquement sur le dictionnaire utilisé.
Pour la petite anecdote, le nom '*Fabricationcepteur de Syntrepèteries et Localemboules*', que nous avons donné au projet a été créé à l'aide du programme et des trigramme générés via un dictionnaire contenant les synonymes de 'Générateur' et 'Mots'.

2 Développement

2.1 Gestion de la ligne de commande

Le premier problème que pose l'algorithme est en effet son exécution. La difficulté réside dans le fait que ce soit une commande, il faut donc gérer les paramètres entrés par l'utilisateur. Dans un premier temps, le but sera donc d'organiser le programme, puis de gérer les options qui permettront l'exécution.

2.2 La génération de mots aléatoires

Nous avons donc, comme le montre le titre, commencé d'abord par créer une fonction qui demande la taille souhaitée pour renvoyer une chaîne de caractères, c'est-à-dire le mot souhaité. Le retour est en widestring du fait que le string ne prend pas en compte les accents.
Nous avons donc créé la constante "*alphabet*" pour récupérer les caractères, puis nous avons généré une variable aléatoire qui va tirer une des lettres, et le fera le nombre de fois que l'utilisateur demande.

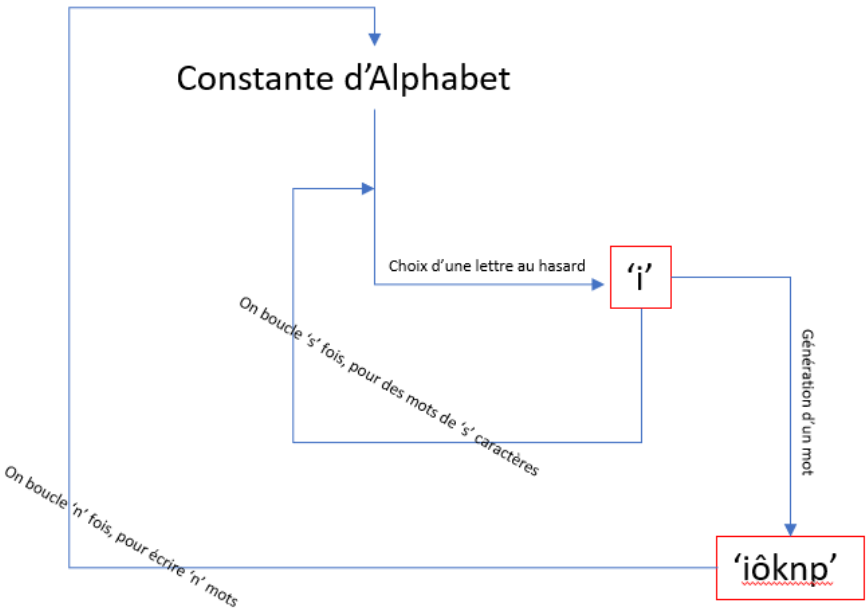

FIGURE 1 – Principe de fonctionnement du mode Aléatoire

2.3 La méthode par digramme

Présentation

Afin de générer des mots par 'digramme' (enchaînement de deux lettres), nous allons avoir besoin d'une fonction qui lit un fichier 'dictionnaire' contenant les mots souhaités ainsi que d'une fonction qui transforme ces mots en blocs de deux caractères. Ces caractères seront ensuite envoyés vers une troisième fonction qui va créer le tableau de probabilités avec ces lettres et enfin, une quatrième et dernière fonction se chargera d'assembler un mot avec ces probabilités.
Il faut préciser, ce qui sera aussi valable pour le trigramme, que deux fonctions ont été créé pour traduire les caractères en chiffre et inversement. En effet, comme le fournit le lexique, la constante *Widestring* comporte 43 caractères. La traduction consiste donc à trouver le rang correspondant aux variables et les traduire.

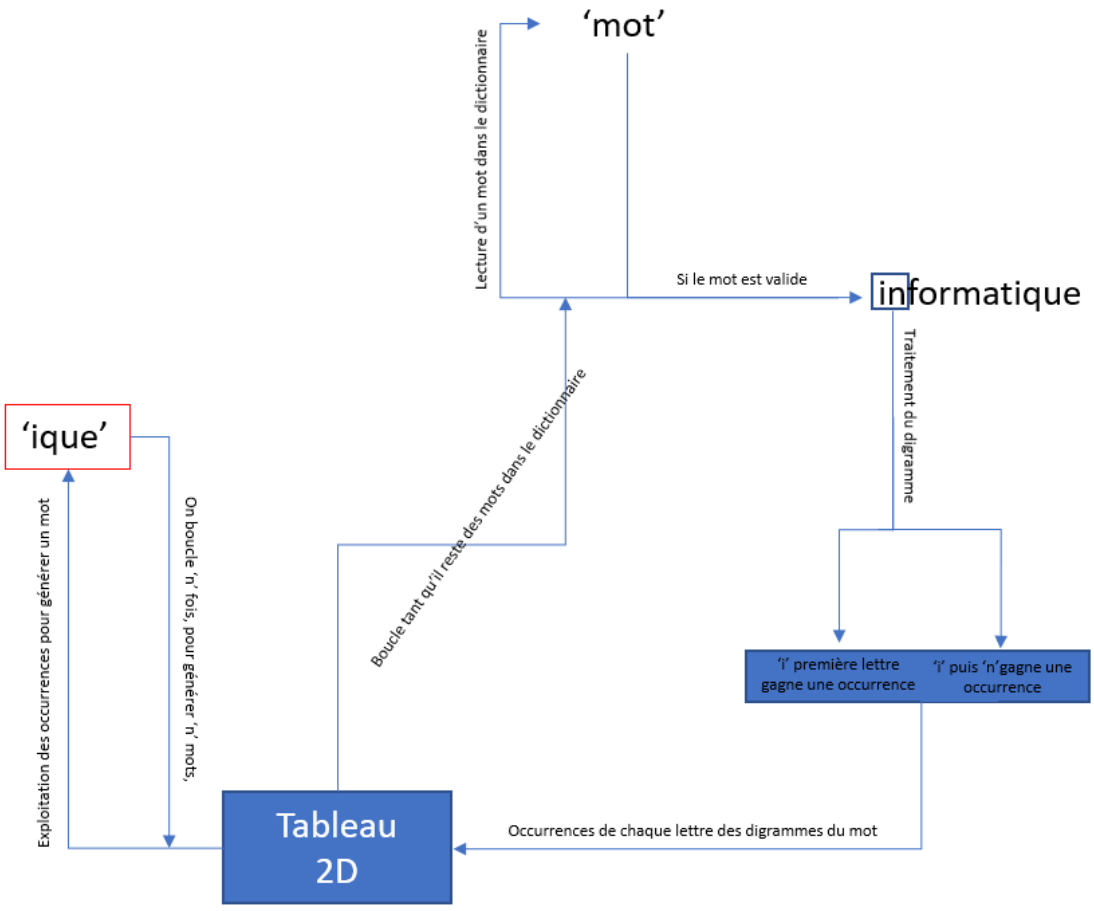

FIGURE 2 – Principe de Fonctionnement du Digramme

Le tableau

Comme pour le tableau présenté dans le sujet, notre tableau sera aussi composé d'une ligne et colonne 0. Elle nous permettra par la suite de récupérer les occurrences des lettres commençant et finissant un mot. Cela nous permettra de calculer selon le lexique la fréquence à laquelle la lettre à laquelle commence le mot et celle qui la termine. Pour les inscrire, nous faisions de simples test de la position du curseur (i = 1 et i = length(mot)) et nous placions la première et la dernière lettre à la case respectivement aux cases tab[0 ;lettre] et tab[lettre ;0]. Le reste sera traité normalement de la façon suivante : tab[lettreavant ; lettreaprès] += 1 jusqu'à ce que le mot soit entièrement compilé dans le tableau.

Les probabilités

Une fois donc le lexique terminé, nous nous retrouvons avec un tableau entièrement rempli. Il faut donc réaliser les probabilités de façon à pouvoir tester la probabilité de chaque case et voir si elle correspond au nombre tiré (la création du mot sera expliqué plus tard.). Nous avons choisi de créer des intervalles pour remédier à cela. Nous avons une fonction qui cumule le nombre dans chaque case en augmentant le nombre. Si l'occurrence de la case est nulle elle n'inscrit rien sinon elle attribue l'occurrence de la case avec ceux cumulés avant. Ainsi, nous obtenons des nombres dans un ordre croissant, représentatifs des intervalles et chacun agissant comme une borne.

La génération du mot

Les probabilités complétées, il faut maintenant créer le mot. La création du mot comporte deux étapes : la première lettre créée, et la construction du mot avec sa condition d'arrêt. Le mot, comme expliqué précédemment, est censé s'arrêter au moment où la deuxième coordonnée est égale à zéro, qui sera la condition d'arrêt de la boucle. Pour la première lettre, il suffit de choisir un nombre au hasard à partir de l'occurrence maximale, puis tester depuis 1 jusqu'à ce que le nombre tiré soit inférieur à la case, cela voudra donc dire que la case correspondante, puis les fonctions se chargeront de traduire l'indice du tableau, c'est-à-dire la case à l'axe y, et le récupère pour le comparer. La première lettre est ainsi donc créée.
Ensuite, la fonction entre enfin dans la boucle qui ne s'arrêtera que lorsque nous l'avons expliqué précédemment.

C'est donc ainsi que les mots en digramme sont créés. Nous allons maintenant aborder la création du trigramme, dont le concept reste similaire mais différent, notamment dans la création du mot.

2.4 La méthode par trigramme

Afin de générer des mots par 'trigramme' (enchaînement de trois lettres), nous avons commencer par utiliser les mêmes fonctions principales que le digramme, mais en passant sur une base de trois caractères au lieu de deux. Le mot sera alors encore plus proche de n'importe quel mot réel de la langue française puisque les enchaînements de trois lettres sont plus fidèles à la langue que ceux de deux lettres.

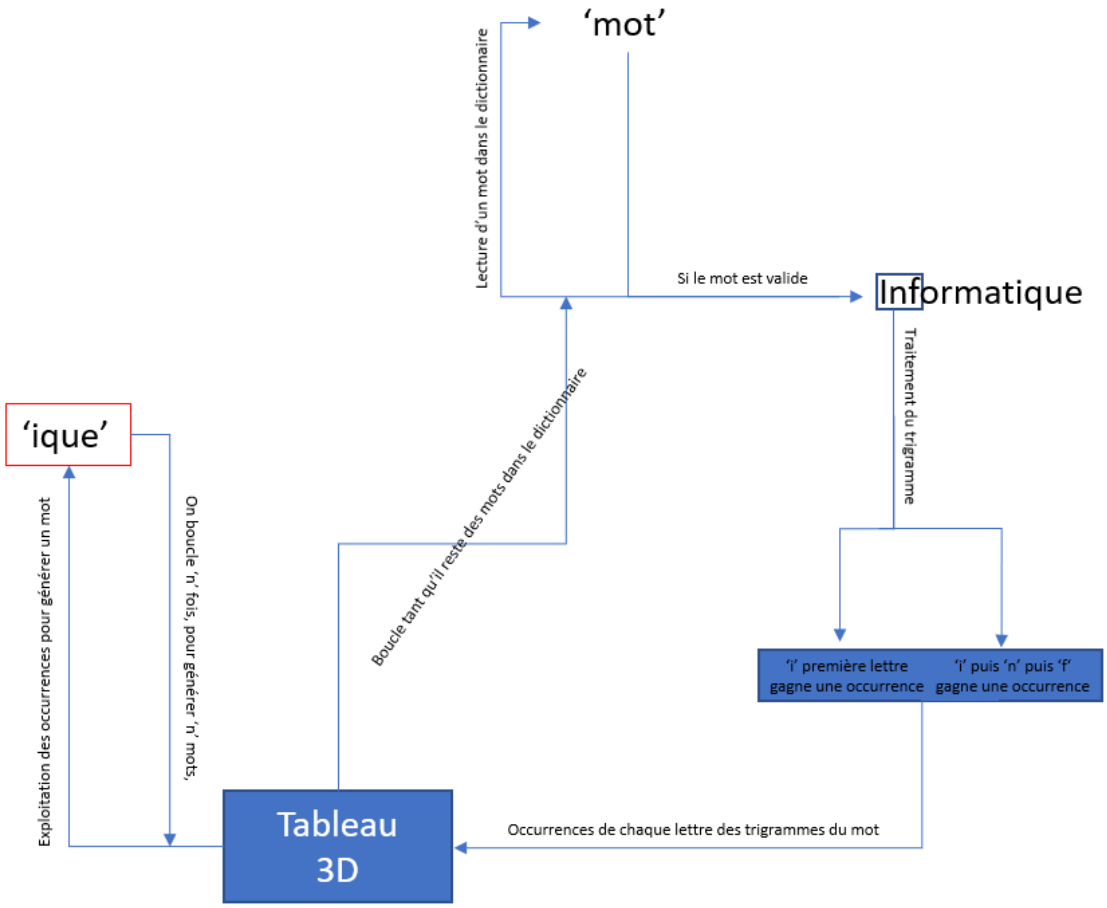

FIGURE 3 – Principe de fonctionnement du Trigramme

Le tableau

Seulement, il s'est posé un problème de définition pour l'inscription des occurrences selon le premier, le deuxième et le dernier caractère.
En effet, l'emplacement des occurrences posaient problème, du fait que l'emplacement de ses caractères se confondaient, on ne pouvait intégrer comme dans le digramme 2 des positions de caractères. C'est pourquoi nous avons opté pour créer un deuxième tableau où nous insérerons seulement les premières lettres des premiers caractères. La contrainte qui suit est donc de renvoyer deux tableaux en même temps, nous avons simplement créé un enregistrement contenant en variables les deux tableaux en même temps. Cela nous permet donc d'avoir un accès constant aux deux tableaux facilitant leur utilisation dans les fonctions.
Le domaine de définition du tableau à 3 dimensions diffère légèrement du premier tableau vu que le deuxième axe (Soit y dans tab[x ;y ;z]) est défini de 1 à 43 (au lieu de 0 à 43) car les difficultés rencontrées précédemment nous ont faits prendre cette décision.

Les probabilités

Les probabilités se déroulent avec le même procédé, vu que la méthode de calcul n'a pas besoin d'interférer sur le nombre de dimensions. Il a juste fallu ne pas oublier aussi de compléter le deuxième tableau avec cette méthode. Les deuxièmes lettres des mots étaient stockées en tab[0,lettre1,lettre2], le premier 0 indiquant simplement que c'est le début du mot. Les dernières lettres sont stockées dans tab[lettre1,lettre2,0], ce 0 indiquant l'inverse du premier cité.

La génération du mot

La première lettre sera donc calculée dans le premier tableau, celui à une dimension, identiquement au digramme, pour ensuite entrer dans la boucle. La boucle agit, au-delà de la condition d'arrêt qui reste la même, de deux manières différentes : la première est lorsque le mot n'est que d'une seule lettre, il s'agit de l'exception de la boucle, car cela n'apparaît qu'une seule fois ; le deuxième apparaît alors pour le reste de la boucle, et calcule normalement pour compléter le mot, en ayant comme conditions de la lettre créée les deux dernières lettres.

Optimisation

Le principal problème lorsque le programme a commencé à fonctionné est le temps qu'il mettait à charger le lexique, de plus des erreurs de segmentation sont apparues nous empêchant de poursuivre la lecture du texte. Nous avons découvert après de nombreux tests, que c'est donc comme l'explique simplement sa définition, une erreur qui se produit lorsque le programme essaie d'employer de la mémoire là où il n'a pas le droit de l'utiliser. Autrement dit, le programme consomme trop de mémoire pour être employé correctement. La source de ce problème provient que la variable *overtableau* est trop lourde, du fait que nous envoyons les deux tableaux en même temps. Nous avons simplement séparé les deux dans l'envoi du tableau, cela a donc considérablement baissé le temps que mettaient les fonctions à charger des tableaux inutilement.

2.5 Étendue du dictionnaire et gestion des erreurs

Pour répondre au projet, nous devons utiliser une constante de caractères 'alphabet', composée des lettres suivantes : 'abcdefghijklmnopqrstuvwxyzàâéèêëïîôùûüßæç-'.
Or dans les dictionnaires utilisés ou même ceux d'autres langues, il existe d'autres caractères comme le 'ö' que l'on retrouve dans 'maelström' par exemple.
Il a alors fallu que nous pensions à une méthode pour empêcher l'envoi de ces caractères dans la fonction de calcul des probabilités et ainsi éviter les erreurs. Pour cela, nous avons comparé chacun des caractères de chaque mot avec chacun des caractères de la constante 'alphabet', dans le cas où il y a une erreur, c'est-à-dire que le caractère n'est pas présent dans la constante, ce mot est alors ignoré et un nouveau mot est pris dans le dictionnaire.

2.6 Génération de Phrases

La génération de phrases se comporte de plusieurs étapes pour pouvoir créer une véritable phrase qui ressemble, structuralement et grammaticalement à du français.
Bien sûr, une fonction *"Majuscule"* est ajoutée pour faire débuter la phrase, et un point est ajouté automatiquement en fin de phrase, conformément à la structure de la langue.

Création du sujet

La création du sujet est basée sur un pseudo-aléatoire ; en effet, nous créons un nombre qui est tiré entre 0 et 100, ce qui fait une chance sur deux, si le nombre est en-dessous de 50 alors il tire un pronom aléatoire parmi la liste *Pronom.txt*. Le cas échéant, il créera grâce à la liste des noms propres un aléatoirement en trigramme, pour rester le plus fidèle possible.

Conjugaison et Accords

Il faut spécifier que les verbes que l'on conjugue sont seulement au premier groupe et deuxième groupe, la liste inclue contient des verbes finissant en -er et ceux finissant en -ir. Le verbe ainsi généré passera dans une fonction qui testera le mot et sa terminaison, et supprimera l'infinitif pour le conjuguer en fonction du sujet.

Affichage des phrases

Sur la même longueur d'onde que la création du sujet, une variable aléatoire sera tirée pour pouvoir afficher des phrases standards. Le premier cas est la création d'un sujet avec verbe et un adjectif. Le second cas est simplement sujet, verbe et adjectif. Deux types de phrases sont proposés : avec un groupe nominal sujet et avec un groupe nominal COD, que nous avons ajouté en bonus.

Gestion de l'erreur de segmentation

Les erreurs de segmentation présentées précédemment ont continués malgré la démarche suivie. Nous avons ainsi remarqué que la mémoire alloué au programme était limité et que la taille conséquente de nos tableaux (faute d'optimisation) dépassaient cette limite. Pour remédier à cela, nous avons donc décidé de supprimer dans la génération de phrases un tableau pour alléger la mémoire et permettre à la fonction de continuer sa démarche. Le choix à donc été de supprimer le tableau des adverbes. Nous ne pouvons donc plus créer de phrases avec des adverbes, mais ces derniers sont invariables et ne représentent donc pas le travail fait sur l'accord sur le genre et le nombre des mots, avec la phrase générée obtenue. Néanmoins, les algorithmes de gestion des adverbes seront laissés en commentaire afin que le travail qui à été fait puisse être observé, même si ils ne seront pas pris en compte dans la notation.

3 Conclusion

Nous avons donc réussi à créer des mot avec des enchaînements en digramme et en trigramme, avec l'utilisation des probabilités et des tableaux fixes remplis selon les occurrences, puis en posant une condition d'arrêt adéquat. La génération des phrases est presque un succès. Nous n'avons pas été capables de pouvoir créer des adverbes à causes des erreurs de segmentation, faute d'optimisation.