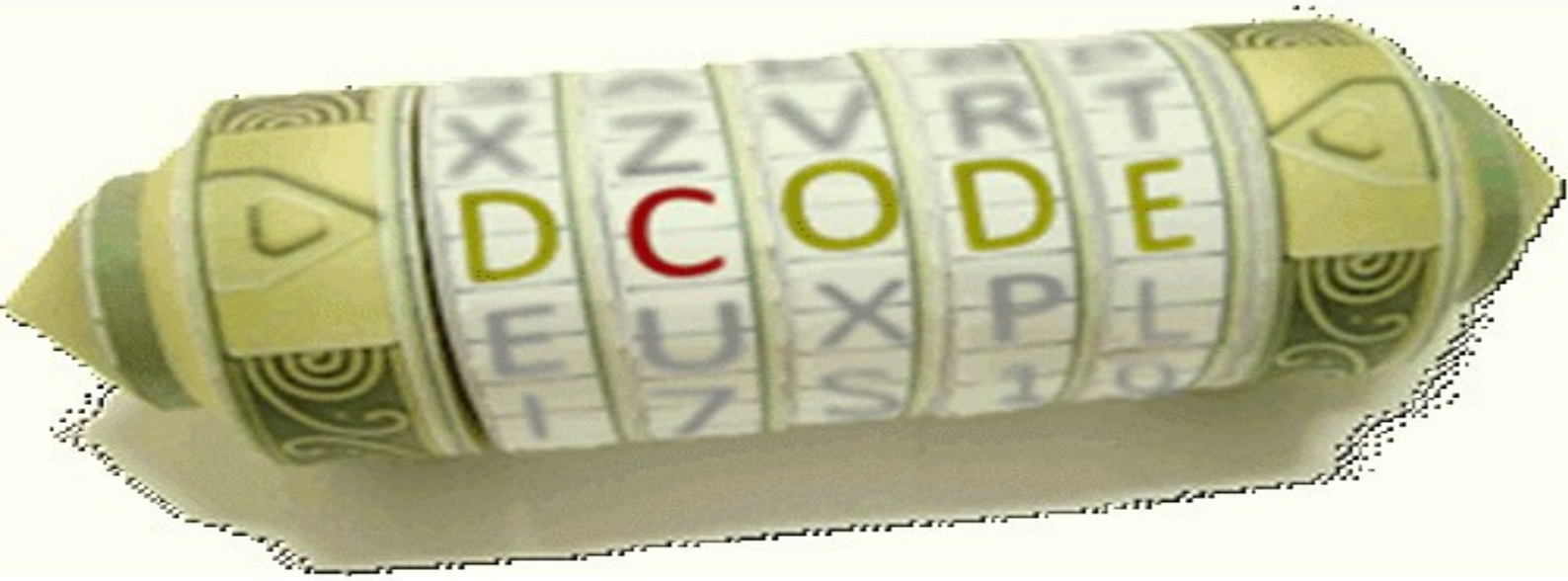
LES TROPHÉES NSI

Édition 2022

DOSSIER DE CANDIDATURE
PRÉSENTATION DU PROJET



Ce document est l'un des livrables à fournir lors du dépôt de votre projet : 4 pages maximum (hors documentation).

Pour accéder à la liste complète des éléments à fournir, consultez la page **Préparer votre participation**.

Vous avez des questions sur le concours ? Vous souhaitez des informations complémentaires pour déposer un projet ? Contactez-nous à **info@trophees-nsi.fr**.

NOM DU PROJET : CodeCraft

> PRÉSENTATION GÉNÉRALE :

Dans le cadre de notre année de terminale en spécialité nsi, nous avons du réaliser un projet en groupe sur un thème de notre choix. Nous avons donc réfléchis avec l'aide de notre professeur, à un projet qui nous intéresse. Pour finalement choisir un projet sur la création de mot par chaîne de markov. Après avoir réalisé notre projet, notre professeur nous a fais découvrir les trophées NSI auxquelles nous avons décidé de participer.
Le principe de markov, dont on étudieras ici les « chaînes de markov », est un modèle basé sur les probabilités. Il permet entre autre la prédiction du futur et donc d'anticiper les différents états d'une variable. Dans notre cas, nous allons voir comment utiliser ce principe pour la création de nouveau mots à partir d'un texte, ou d'un dictionnaire. Pour faire simple, les chaînes de markov nous permettront de déterminer la probabilité qu'une lettre tombe avec une autre et ainsi d'obtenir des mots cohérent lexicalement.
Nous verrons ensuite les nombreuses utilités de ses « nouveaux mots » et notamment dans le domaine de la sécurisation de mots de passe.

> ORGANISATION DU TRAVAIL :

[Maël] [célian]

problème1 : afficher la première lettre d'un mot au hasard dans le dico
--> reprise en main de la syntaxe + indices avec la liste contenant chacuns des mots

v1 : 2 lettres
lignes de codes a la suite (=! fonction)

[maël]
--> pour la première lettre :
- description : on transforme le fichier texte, modélisé par une chaîne de caractères en une liste avec tout les mots
- on choisit un mot au hasard
- on sélectionne la première lettre de ce mot
- on l'ajoute à notre nouveau mot (chaîne de caractères vide)
--> pour la deuxième lettre :
- On créé une liste vide (futur liste de lettre)
- on parcourt toute les lettre de chaques mots du dictionnaire
- si la lettre actuelle est la meme que notre première lettre, on ajoute la suivante dans notre liste
Ainsi, on obtient une liste de lettre qui peuvent suivre notre première lettre, (selon les mots du dico)
- on choisit enfin une lettre au hasard dans la liste
- on l'ajoute à notre nouveau mot

v2 : fonction_ordre
- on choisit la première lettre d'un mot au hasard dans l'ensemble du dico

- on met toute les lettres possibles qui peuvent suivre la lettre précédente et on en choisis une au hasard
- on s'arrête quand on tombe sur le caractère "\n"

+ ajout d'un module permettant d'empêcher l'ajout d'un caractère vide en fin de mot
[célian] test avec un dico anglais ; fonctionne parfaitement

##texte = open('english.txt', 'r',encoding = "latin1")
##dico = texte.read()
##texte.close()
ordre 1 : ok - sans problème

ordre 2 : problème de sauvegarde de variable résolu, on a encore un "out of range" ...
[célian] résolut le problème : inversion des variables --> tout fonctionne

ordre 3 : attention au départ !! la troisième lettre est trouvé sur le modèle de l'ordre 2 (faire une fonction)
[célian] a règle le prob --> start a adapter en fonction du degré
[mael] a mit ça en fonction
--> utiliser l'ordre de deg5 pour les 2,3,4 lettres est impossible : solution --> fonctions ordre1, ordre2, ordre3, ordre4

[mael] rajout des spécifications
ordre 4 et 5 --> sur le même principe

LES ÉTAPES DU PROJET :

Une première version :

Pour débuter notre projet, nous avons essayé de créer en quelques lignes un programme qui permet créer des mots de quelques lettres à partir d'un dictionnaire avec les mots en ligne, dans un fichier texte.
Il fonctionne sous le modèle suivant :
→ Pour la première lettre :
- on choisit un mot au hasard
- on sélectionne la première lettre de ce mot
- on l'ajoute à notre nouveau mot (chaîne de caractères vide)
→ Pour la deuxième lettre :
- On créé une liste vide (futur liste de lettre)
- on parcourt toute les lettres de chaque mots du dictionnaire
- si la lettre actuelle est la même que notre première lettre, on ajoute la suivante dans notre liste
Ainsi, on obtient une liste de lettre qui peuvent suivre notre première lettre, (selon les mots du dico)
- on choisit enfin une lettre au hasard dans la liste
- on l'ajoute à notre nouveau mot

→ Pour les autres lettres : On répète le programme autant de fois que l'on veut pour obtenir un mot de 3, 4, n lettres.
Dans cette première version, on est donc sur le modèle suivant : on choisit une lettre en fonction de la précédente, ce qui explique que les lettres ne se suivent pas vraiment, on n'arrive pas à obtenir des syllabes ou groupes de lettres cohérents.

Une Deuxième version :

Face à ce problème, nous avons donc du changer notre façon de voir les choses.
On introduit donc la notion d'« ordre », de façon à ce que une lettre soit choisie en fonction des n lettres précédentes, « n » étant un nombre fixe donné.
Cela permet d'avoir une vision plus large et d'avoir des lettres choisies en profondeurs. On aura ainsi des syllabes et groupes de lettres beaucoup plus cohérent et qui se rapprochent d'avantage de la langue française.
Par exemple, un mot d'ordre 3 sera un mot qui sera constitué de lettres choisies en fonctions des 3 précédentes.
On a donc créé des fonctions nommées *mot_ordre_1*, *mot_ordre_2*, etc … sur le principe suivant : (Jusqu'à l'ordre 5 pour notre part)

- Comme pour la première version, on initialise une première lettre, choisie de manière aléatoire et un deuxième, troisième, … cinquième en fonction du degré de la fonction (un fonction d'ordre 4 à besoin de 4 lettres initialement)
- Si lors de notre parcourt des lettres du fichier texte (dico) on tombe sur une bonne succession de lettres (pour un mot d'ordre 3 → trois bonne lettres qui se suivent), on ajoute la suivante à notre liste de lettre possibles et un en choisie ensuite une au hasard qu'on ajoutera à notre mot
remarque : cela nous permet d'intégrer la notion de probabilité de tel manière que lorsqu'on choisie une lettre (parmi notre liste de lettres possibles), une lettre qui revient plusieurs fois à d'avantage de chances d'être retenue ce qui nous amène vers des mots plus logiques
- On décale d'un rang chaque lettre stockées dans les variables
- On arrête la création du mot quand on tombe sur le caractère "\n", c'est à dire l'espace.

Une troisième version :

Dans une troisième version, nous allons donc apporter une modification à notre code source afin d'obtenir des mots de taille comprise entre 7 et 15. (on verra ultérieurement que cela est déjà grandement suffisant:).
Il nous a suffit de rajouter quelques indication :
- ne retient le mot que si sa taille est comprise entre 7 et 15, sinon il recommence la création d'un nouveau mot, et ainsi de suite … .
On en a donc finit avec le code source : on a un code source final qui permet de générer « nouveau mot » à partir d'un texte ou d'un dictionnaire avec la condition qu'il soit relativement grand.
Cela nous amène donc à la partie sécurisation et mot de passe …

> FONCTIONNEMENT ET OPÉRATIONNALITÉ :

Une première version :

On identifie un premier problème : Les mots sont difficilement prononçable et se rapproche très peu de ceux du dictionnaire et donc de la langue française ;

Une Deuxième version :

On identifie un second problème : Concernant le début de chaques fonction *mot_ordre* , nous avons remarqués que les premières lettres permettant le départ de la fonction étaient choisi sur le modèle de la version1 (une lettre en fonction de la précédente) ce qui ne rentre pas dans nos attendus initial.
→ Par exemple, un mot d'ordre 4, a ses quatres premières lettres qui semble choisies de manières aléatoire ce qui fausse le mot.
Une solution à ce problème : On construit des fonctions adaptées qui permettent de choisir les première lettres de manière plus rigoureuses : *premiere_lettre*, *lettre_suivant_deg2*, … , *lettre_suivante_deg5*.
Par exemple :
- *lettre_suivant_deg2* → permet de choisir un lettre en fonction des 2 premières
- *lettre_suivant_deg4* → permet de choisir un lettre en fonction des 4 premières.
Ainsi, chaque fonctions *mot_ordre* début avec une ou plusieurs lettres adaptés en fonction de son degré.
On a donc une version avec des programmes qui permettent d'obtenir des mots qui, plus on a une degré d'ordre élevé, « sonnent français ».
Quelques exemples de mots : 'autillage' ; 'récieux' ; 'drouillets' ; 'préhender'.
remarque, dans certains cas isolés, on recréé des mots déjà existants comme "Phillipe", "OTAN", "Languedoc".
On identifie un troisième problème : Pour aller vers notre idée de mot de passe, on problème se pose : certains de nos « nouveaux mots » sont trop courts et ne suffiraient pas à sécuriser correctement un compte, une adresse mail.

> OUVERTURE :

- génération de texte
-utiliser un livre pour générer des phrases aléatoirement qui paraissent française.

- Un lien avec la musique
-utiliser une partition pour générer une mélodie avec des suites de notes cohérente mais aléatoires.

- On a toujours dit que pour sécuriser un compte (Amazon, compte bancaire, Steam, …), il fallait un mot de passe dit « fort », c'est à dire un mot de passe avec : - au minimum 8 caractères, - des majuscules, - des minuscules, - des caractères spéciaux. Mais au fil du temps, on s'est rendu compte que cette méthode ne s'avère pas plus efficace qu'une autre.

- Un gros avantage des mots de passe à la manière « des chaînes de markov », est qu'il est assez simple de s'en souvenir puisqu'il « sonne français », mais tout tout aussi difficile à trouver pour un ordinateur.
Un exemple de mot de passe de 10 lettres (ici d'ordre 5) : salafrisse, qui se prononce facilement.

DOCUMENTATION

Pour se projet nous avons utilisé un dictionnaire français ainsi qu'un dictionnaire anglais.

Voici une capture d'écran de quelques mots créé avec notre projet :