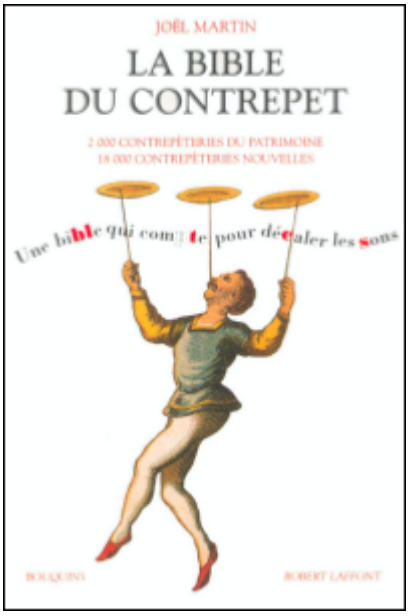
Département Informatique

Rapport de projet tuteuré

Contrepèterie
“L’art de décaler les sons”



Elaboré par : PERRET Louis, CHAUMONT Maël, DUTEYRAT Jules, DALLET Simon
Encadré par : Pascal LAFOURCADE

UNIVERSITÉ Clermont Auvergne

ACADÉMIE DE CLERMONT-FERRAND
Liberté
Égalité
Fraternité

Année Universitaire : 2021/2022

iut Université d’Auvergne
Clermont-Ferrand Aurillac Le Puy-en-Velay

Nous autorisons la diffusion de ce document au sein de l’intranet de l’IUT

Remerciements

En préambule, nous tenons à adresser nos sincères remerciements à notre professeur Monsieur LAFOURCADE sans qui ce projet n'aurait pas pu voir le jour. Nous le remercions notamment pour son temps et ses conseils formateurs et utiles qui ont pu nous aider à réaliser ce projet à un degré que nous n’avions pas imaginé au départ.

Nous tenons à le remercier également pour nous avoir prêté le livre de Joël Martin : La Bible du contrepet. Cet ouvrage nous a permis d’élargir notre connaissance à l’art des contrepèteries : sujet totalement nouveau pour la plupart des membres du groupe, mais aussi de tester et ajouter des contrepèteries sur nos réalisations.

Pour finir, nous souhaitons témoigner de notre reconnaissance pour les deux équipes qui ont contribué à ce projet avant nous. Elles nous ont permis de commencer à travailler sur des bases solides grâce à leurs rapports et à leurs réalisations.

Sommaire

Introduction

Dans le cadre du projet tuteuré de deuxième année, nous avons obtenu le projet “Contrepèteries, l’art de décaler les sons” de Monsieur LAFOURCADE..
Notre équipe est constituée de 5 étudiants : Maël CHAUMONT, Simon DALLET, Jules DUTEYRAT, Louis PERRET ainsi que Jules ROCHE qui a contribué au projet pendant le semestre 3.

Avant d’expliquer le sujet, il nous semble important de définir ce qu’est une contrepèterie. Voici la définition du Larousse : *“Inversion de l'ordre des syllabes, des lettres ou des mots qui, modifiant le sens, produit des phrases burlesques ou grivoises”*
Pour réaliser une contrepèterie, il faut donc, à partir d’une phrase, intervertir lettres et sons (et plus encore !) pour créer une phrase humoristique et vulgaire.

Les ouvrages de Joël Martin, un écrivain français, classent les contrepèteries selon quatres catégories :

Classiques	**Décadentes**	**Irrégulières**	**Enchevêtrées**
-Echange de consonnes	-Echange de sons, voyelles, groupes de mots	-Circulaire, à l’intérieur d’un même mot	-Cassure de mots, soudures entre deux mots….
L’art de dé**c**aler les **s**ons.	La **Ch**ine se dresse devant les Ni**pp**ons.	Cette p**ei**ne r**u**de et **i**nique	L’é**l**u se **c**asse

Le sujet du projet consiste à améliorer un site web ouvert à tous permettant de faire découvrir les contrepèteries aux utilisateurs, ainsi qu’une application sur terminal destinée uniquement au maître d’ouvrage reprenant les grandes lignes du site. Ces deux projets ont été réalisés par deux autres équipes avant nous.
Nous avons hérité de ce projet car, pour M. LAFOURCADE, ces deux réalisations pouvaient encore être améliorées, notamment en y ajoutant davantage de fonctionnalités pour les rendre les plus complètes possibles.

Ainsi, nous avons dégagé la problématique suivante: Comment, en se basant sur des réalisations préalables, pouvons-nous créer une application console et un site web permettant d’initier les utilisateurs à l’art des contrepèteries ?

Dans une première partie nous expliquerons notre démarche d’implémentation de toutes les nouvelles fonctionnalités, dans une deuxième partie, nous vous expliquerons comment nous les avons ajoutées aux supports déjà existants, puis pour conclure nous détaillerons notre ressenti et ce que nous a apporté ce projet au cours des 5 derniers mois.

Présentation synthétique du sujet de notre projet

Lorsque nous avons débuté ce projet, nous sommes partis d’une base existante : 2 anciens projets réalisés par d’autres équipes les années précédentes.
Tout d’abord un site, ayant été codé en Javascript, HTML et CSS par d’anciens étudiants de l’IUT (Adrien BLAY, Corentin CAMPIDELLI, Anthony COSTA, Pierre ERBLAND, Thomas GOUNON et Jean-Baptiste VAYSSADE).
Il est destiné à être utilisé par tous : chacun peut y accéder librement via son navigateur afin d'effectuer des recherches au sein d’un mot, ou encore améliorer son score sur un des jeux présents. Nous avons modifié ce site pour les besoins de ce projet, qui est disponible à l’url suivante : https://sancy.iut-clermont.uca.fr/~lafourcade/contrepeteries/fr/contrepeterie_irreguliere.html

Figure 1 : Page ‘Aide à la contrepèterie’ du site web

Nous avons de même récupéré un second projet : il s’agit d’une application sur terminal en Python, réalisée en 2020-2021 par Thomas COMBETTES et Corentin SABIER, des élèves en prép’ISIMA. Cette application est vouée à un usage plus personnel que le site. Elle est seulement destinée au maître d’ouvrage : cette application console possède des fonctionnalités plus poussées que le site web, notamment la recherche de contrepèteries dans une phrase ou encore des options de filtrages. De plus, il est possible de l’utiliser sans connexion internet.

```
Recherche des contrepétries possibles ...
Voici donc les lettres que l'on peut changer :
  'c'  'o'  'd'  'e'

 1   c - g    gode
 2   c - i    iode
 3   c - m    mode
 4   c - r    rode
 5   o - a    cade
 6   d - k    coke
 7   d - t    cote
 8   e - a    coda

0 = quitter l'aide,-1 revenir au début de l'aide
ou numéro de l'échange qui vous intéresse :
```

Figure 2 : Exemple de recherche au sein du mot ‘code’ avec l’application console

Par ailleurs, pour les besoins du projet, nous avons décidé de revoir la classification des contrepèteries. En effet, les catégories citées dans l’introduction, n’ont pas de logiques informatiques : les contrepèteries contenues dans une même catégorie nécessitent bien souvent des algorithmes totalement différents. C’est donc pour cela que nous avons séparés notre travail sur nos propres catégories. En particulier :

- Contrepèteries sur des sons : La **Ch**ine se dresse devant les Ni**pp**ons.
- Contrepèteries sur des lettres : L’art de dé**c**aler les **s**ons.
- Contrepèteries sur des coupures de mots : Le paysan **p**êche avec Sa**b**ine.
- Contrepèteries sur des liaisons de mots : L’**hi**rondelle n’a pas peur de la tr**a**que

Malgré tout, pour Mr. LAFOURCADE ces anciens projets étaient loin d’être parfaits, et de nombreux points restaient à améliorer : les fonctionnalités d’échanges de lettres/phonèmes permettant de réaliser nos contrepèteries étaient basiques (simple échange lettre à lettre sur le site, échanges 1 phonème-1 phonème présents pour l’application console mais rien de plus). Nous avons donc décidé d’élargir ces fonctionnalités, autant sur l’application console que sur le site, en y ajoutant de nouveaux types de contrepèteries ainsi que des fonctionnalités de filtrages que nous énumérerons dans les parties suivantes. Le style du site était aussi à revoir, son design général étant plutôt bon mais l’affichage des résultats et le placement des champs étaient très peu intuitifs pour l’utilisateur.

Pour résumer, notre sujet a donc été le suivant : multiplier les fonctionnalités de l’application sur terminal et du site, ainsi que d’améliorer l’aspect de ce même site.

Analyse du projet

En entrant sur le site web, un utilisateur peut avoir plusieurs idées en tête : il peut chercher des informations sur les contrepèteries, auxquelles il pourra accéder directement depuis la page d’accueil, ainsi que depuis la présentation du site et l’onglet de classification.
Il pourrait aussi venir sur ce site avec des intentions plus “ludiques”, voulant simplement jouer ou améliorer son score à un des 2 jeux présents.
Mais la plupart du temps, un utilisateur cherchera à trouver des contrepèteries, soit par rapport à un mot qu’il a en tête, soit dans une phrase entière. Il peut vouloir échanger des lettres ou des phonèmes (syllabes), selon des critères qu’il aura lui-même choisi ou pré-sélectionnés par défaut.
Dans le cas où l’utilisateur se servirait de l’application console, seul ce dernier cas d’utilisation est utile, car il s’agit de l’unique objectif de l’application console : trouver des contrepèteries, en choisissant parmi de nombreux critères plus poussés que sur le site.

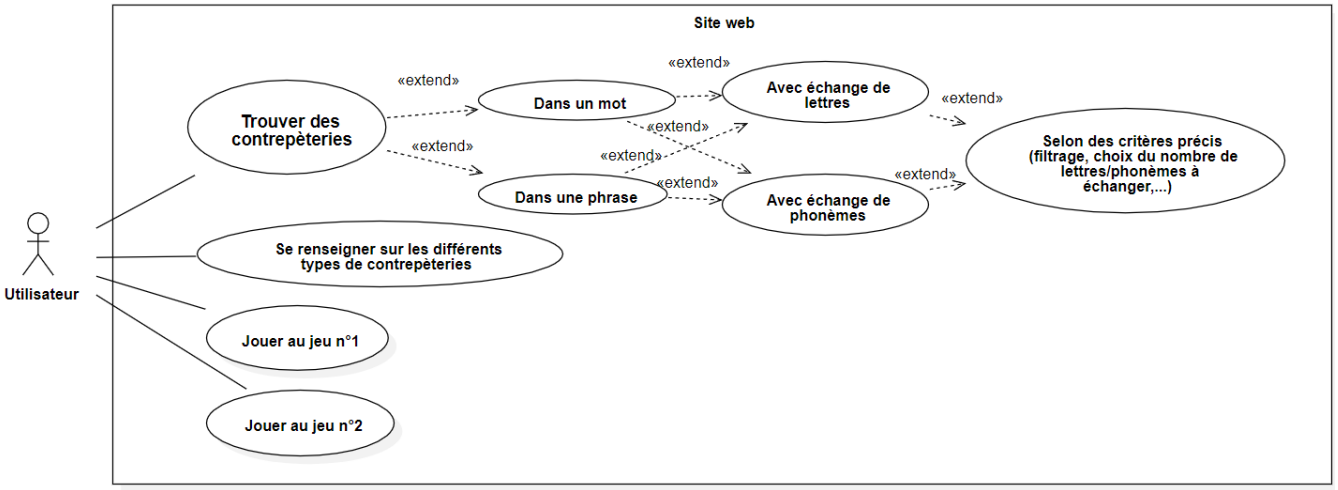


Figure 3 : Diagramme de cas d’utilisation du site web

Réaliser ces contrepèteries a été rendu possible grâce à des fichiers de données au format .csv, récupérés en partie grâce à un crawler (explication plus approfondie dans la partie sur les filtres).
À l’intérieur de ces fichiers *.csv**, on peut trouver différentes informations sur chaque colonne pour chaque mot. Premièrement, on a le mot écrit de toutes lettres puis, dans la colonne suivante, son écriture phonétique (alphabet phonétique international). Ensuite vient le genre du mot, c’est à dire masculin ou féminin en français, et non-genré pour l’anglais. Les informations suivantes sont la ou les classes grammaticales de chaque mot. Nous avons choisi d’en faire une liste car selon le contexte, un même mot peut avoir plusieurs significations et donc des classes grammaticales différentes.
Ex : “abandonné” peut être un verbe ou un adjectif selon le contexte.
Enfin, nous avons la pluralité c'est-à-dire “s” si le mot est singulier ou “p” s’il est pluriel, “sp” s’il est invariable.

Organisation

Pour réaliser le projet, étant donné les deux livrables attendus, nous avons créé, naturellement, deux sous-équipes pour s’organiser au mieux dans le travail. Une équipe de deux était destinée à travailler sur l’application console dans le langage Python (Louis PERRET et Jules DUTEYRAT)
Puis, l’autre équipe composée des autres membres, avait pour mission d’améliorer fonctionnellement et graphiquement le site web et d'implémenter des fonctionnalités basées sur le code Python une fois ces dernières réalisées. Les langages HTML et CSS ont été utilisés pour la partie graphique, ainsi que le JavaScript pour réaliser toutes les fonctionnalités.

Le Javascript, étant un langage de programmation côté client, a été un choix idéal puisque ce dernier permettait d’éviter une surcharge de requêtes auprès des serveurs de M. Lafourcade. Quant au python, c’était le seul langage que connaissaient les étudiants qui ont réalisé l’application console, il s’était donc imposé naturellement.

Comme nous avions un membre du projet *AJAC** (Jules ROCHE), nous avons fait le choix de lui confier l’amélioration du jeu présent sur le site web. En effet, il s’agit d’une tâche assez indépendante du reste du projet il était donc plus pratique pour l’ensemble de l’équipe qu’il se charge de cette tâche étant donné son départ de l’équipe au semestre 4.

En terme d’organisation générale, nous nous sommes retrouvés tous les lundi matins pendant un créneau de 4h pour mieux communiquer et avancer en binôme sur les tâches. De plus, le projet s'est réalisé sous le joug de la *méthode agile**. Pour ce faire, nous avons organisé des réunions avec M.Lafourcade ces mêmes lundi pour lui faire part de la progression et prendre note de ses retours et de ses indications. Pour permettre la meilleure organisation et le meilleur suivi possible, des comptes-rendus de nos travaux de la semaine étaient envoyés chaque dimanche soir (au préalable de la réunion du lundi) ainsi que des comptes-rendus de chaque réunion avec le professeur encadrant.

En dehors des créneaux en présentiel, nous avons effectué le travail sur notre temps-libre et échangé grâce à la plateforme Discord. Nous avons créé un groupe qui nous a permis d’échanger entre nous mais aussi avec M. LAFOURCADE de manière beaucoup plus efficace qu’avec des mails par exemple.
Nous avons retranscrit les tâches de chaque équipe ci-dessous, à l’aide d’un diagramme de Gantt.

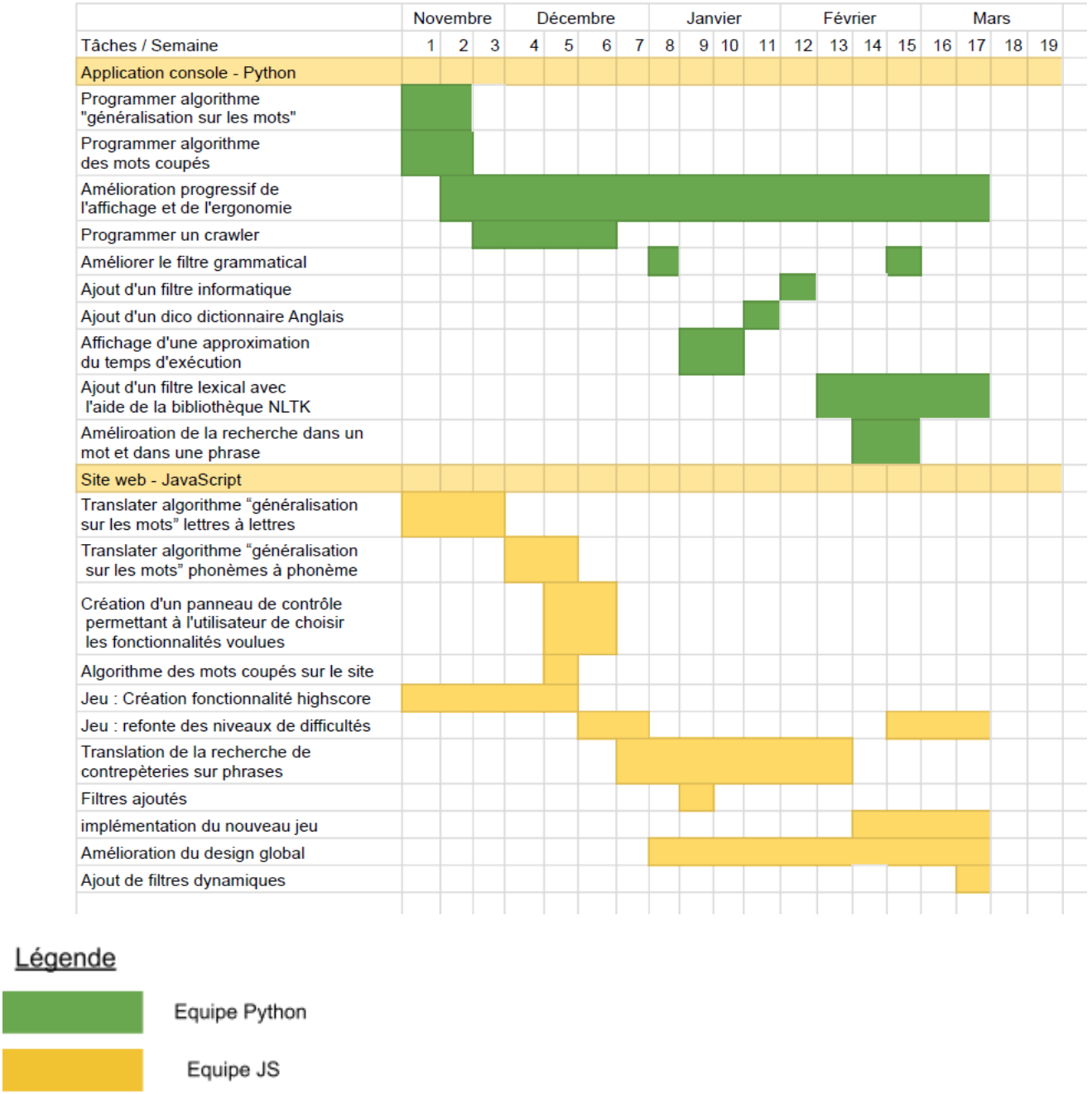


Figure 4 : Diagramme de Gantt

Sur ce diagramme de Gantt, seules les tâches principales sont répertoriées par souci de lisibilité. Nous l’avons découpé en disposant une colonne par semaine. Le projet durait 19 semaines mais nous avons arrêté la plupart de nos travaux à la 18ème. La dernière semaine a été consacrée essentiellement à la rédaction de ce rapport.

Généralement, l’équipe de l’application console développait les fonctionnalités puis l’équipe site web les intégrait au mieux sur le site, tout en développant les jeux et divers ajouts bien spécifiques au site web. Cette organisation nous a permis de développer beaucoup de fonctionnalités sur le projet tout en libérant du temps pour l’équipe du site. En effet, comme les algorithmes étaient imaginés par l’autre équipe, elle pouvait se concentrer plus efficacement sur les tâches supplémentaires que demandait le site Internet.

Réalisations

Ajout de types de contrepèteries

1. Les contrepèteries simples

Quand on pense contrepèterie, généralement la première pensée qui nous vient à l'esprit est une contrepèterie simple, soit un échange entre deux mots d'une lettre contre une lettre ou un phonème contre un phonème.

Exemple d'une contrepèterie simple avec lettres : **l**aver->**b**aver
Et avec phonèmes (=syllabes) : gât**eau**-> gât**é**

Pour réaliser une contrepèterie, nous avons besoin de 4 mots pour réaliser un échange : les 2 mots initiaux (ex: **c**ode -> **m**ode) et 2 mots finaux avec le même échange de lettres (tra**m** -> tra**c**).

C’est une manière de représenter l’échange que l’on souhaite effectuer pour pouvoir créer notre phrase. Nous l'appellerons tout au long du rapport un quadruplet.

En débutant ce projet, nous avons trouvé un algorithme permettant de trouver ces 2 mots initiaux. Cependant, ce dernier ne permettait que l’échange de 1 lettre - 1 lettre (réciproquement phonème). Nous avons débuté par une phase de généralisation de cet algorithme afin de pouvoir effectuer un échange de x lettre(s)/phonème(s) par y lettre(s)/phonème(s).

Pour ce faire, nous sommes partis de l’algorithme déjà présent, auquel nous avons ajouté des paramètres à prendre en compte : x qui représente le nombre de lettres/phonèmes à enlever, puis y qui représente le nombre de lettres/phonèmes à ajouter dans le mot de départ. Le but est de tester tous les échanges possibles dans lesquels nous remplaceront x par y, puis de tester si ce nouveau mot existe dans notre dictionnaire.
Exemple :

code - mode -> x=1 & y=1 car on remplace un ‘c’ par un ‘m’
code - diode -> x=1 & y=2 car on remplace ‘c’ par ‘di’
code - coq -> x=2 & y=1 car on remplace ‘de’ par ‘q’

```
fonction contrepeteriesSimples(mot, x, y, mode)
1    listeResultats <- initialisation
2    si mode est échange de phonème
3      alphabet <- récupérer alphabet de phonèmes
4      mot <- récupérer écriture phonétique du mot
5    sinon
6      alphabet <- récupérer alphabet des lettres
7    fin si
8    listeCouplesPossibles <- récupérer toutes les combinaisons possibles
                                de longueur y à partir de alphabet
9    pour chaque lettre dans mot
10       couple <- récupérer le couple de longueur x dans mot à partir de
                   lettre
11       s'il y a un couple
12             pour chaque combinaison dans listeCouplesPossibles
13                   nvtMot <- remplacer couple par combinaison dans mot
14                   si nvtMot existe et si couple ≠ combinaison
15                         ajouter nvtMot dans listeResultats
16                   fin si
17             fin pour
18       fin si
19     fin pour
20
21     retourner listeResultats
22   fin fonction
```

Figure 5 : Algorithme de la fonction contrepèteries simples

Le mode spécifie le type d’échange. Il peut correspondre à l’échange de lettres, dans ce cas nous avons besoin de l’alphabet basique (a,b,c,....), ou bien à l’échange de phonèmes, et dans ce cas nous avons besoin de l’alphabet phonétique correspondant à la langue dans laquelle on effectue la recherche ainsi que de l’écriture phonétique du mot pour pouvoir réaliser l’échange. Le fait de charger cet alphabet à chaque recherche permet de s'adapter dynamiquement au choix de l’utilisateur.

Une fois que nous avons récupéré les deux mots initiaux grâce à l’algorithme précédent, nous devons maintenant rechercher les deux mots finaux afin de pouvoir proposer tous les quadruplets possible.

Pour ce faire, nous allons parcourir notre dictionnaire de mots à la recherche de mots qui vont posséder le même échange de lettres/phonèmes. Dans l’exemple précédent, l’échange est un ‘c’ contre un ‘m’, nous allons donc parcourir notre dictionnaire à la recherche d’un mot qui possède un ‘c’ qui, quand on échange avec un ‘m’, donne un nouveau mot qui existe.

```
1 fonction rechercheQuadruplet(mot_origine, coupleX, coupleY motInitial)
2   listeResultats <- initialisation
3   pour chaque mot du dictionnaire
4     si le mot possède le coupleX
5       nvtMot <- échange du coupleX par le coupleY dans mot
6       si nvtMot existe et si nvtMot ≠ mot_origine
7         ajouter nvtMot dans listeResultats
8       fin si
9     fin si
10  fin pour
11  retourner listeResultats
21 fin fonction
```

Figure 6 : Algorithme de la fonction rechercheQuadruplet

Explication des paramètres :

- mot_origine : mot entré par l’utilisateur (‘code’ dans l’exemple)
- coupleX : couple de lettres/phonèmes d’origine (‘c’ dans l’exemple)
- coupleY : couple de lettres/phonèmes inséré (‘m’ dans l’exemple)
- motInitial : mot trouvé en échangeant coupleX par coupleY (‘mode’ dans l’exemple)

Une des réponses de cet algorithme si nous utilisons ‘code’ et ‘mode’ comme mot initiaux sera ‘trac’ et ‘tram’.

2. Les contrepèteries sur des phrases

La recherche dans les phrases constitue un point essentiel d’un projet traitant des contrepèteries. En effet, nombreux sont ceux qui se sont essayés à cet art sans succès. Un algorithme permettant de trouver une contrepèterie à partir d’une phrase saisie par l’utilisateur a donc toute son importance.

Exemple : Les femmes laissent leur **cœur** aux vain**cus**.

L’algorithme permettant de trouver la solution à cette contrepèterie était déjà présent et très fonctionnel sur le projet Python. Cependant, sur le site Web la page destinée à cette fonctionnalité était prête mais seulement un début d’algorithme était écrit. Nous avons fait le choix ici de repartir de zéro pour le code de cette page. Notre code est dans sa majorité une adaptation du projet Python, en JavaScript.

Comme le code provient d’un ancien projet, nous allons simplement rappeler le principe de l’algorithme de recherche. Après que l’utilisateur ait saisie une phrase, l’algorithme va tester tous les échanges possibles. C'est-à-dire qu’il va tester tous les échanges de lettres (ou phonèmes) pour chaque mots de la phrase, deux à deux.

```
1 fonction mainRecherchePhrase (phrase)
2       listeResultats <- intitialisation
3       pour chaque mot1 de phrase
4               pour chaque mot2 de phrase
 5                  nvPhrase <- fonction mixPhrase (mot1,mot2) -> test les échanges  possibles entre les
                                    deux mots (algorithme similaire à contrepteriesSimples)
6               fin pour
7               ajouter la nvPhrase à listeResultats
8       fin pour
9       retourner listeResultats
10 fin fonction
```

Figure 7 : Algorithme de la fonction mainRecherchePhrase

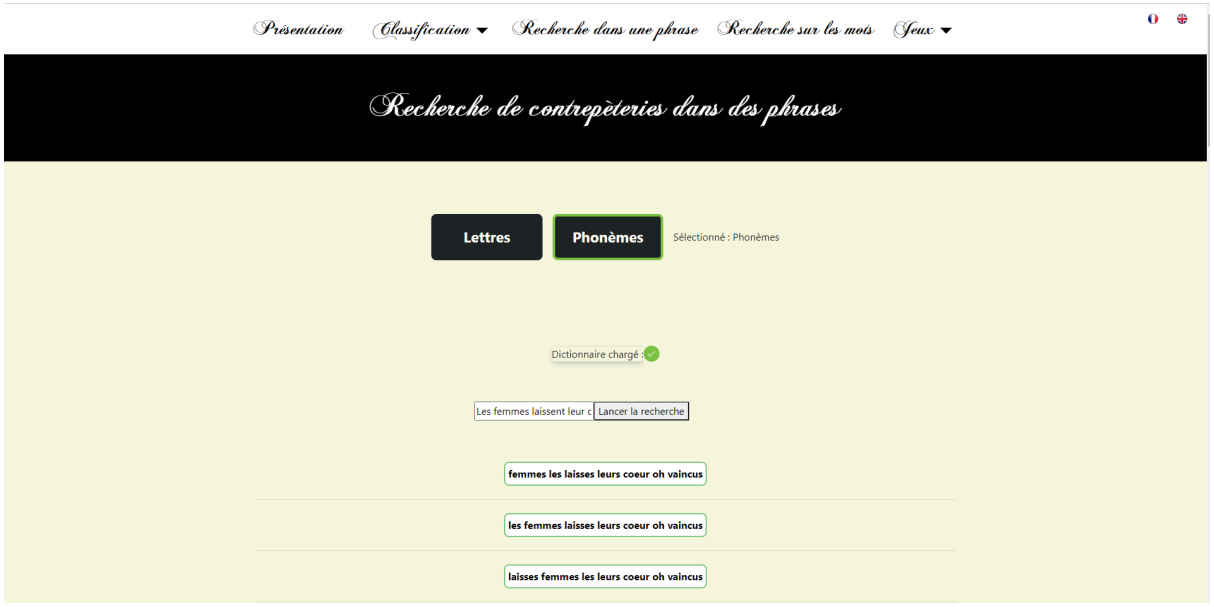


Figure 8 : Recherche de contrepèteries sur une phrase (Site Web)

Pour les résultats lors d’une recherche par phonème, nous proposons plusieurs possibilités d’orthographe. La recherche par les sons est souvent plus polyvalente mais pose des problèmes de résultats car plusieurs mots s’écrivent différemment mais se prononcent exactement pareil. Il est donc pertinent de proposer à l’utilisateur plusieurs possibilités d’orthographe du résultat.
Il s'agit de la fonctionnalité qui diffère le plus entre le site et l’application console. En effet, nous avons dû faire nos propres algorithmes pour s’adapter au mieux au web. Nous ne proposons pas un algorithme qui permet d’afficher toutes les possibilités d’orthographe d’une phrase. C’est un choix délibéré car faire un tel algorithme pourrait afficher plusieurs centaines de possibilités d’orthographe d’une phrase, ce qui est totalement impertinent que cela soit sur console ou sur une page web. Nous nous contentons de proposer seulement quelques résultats d’orthographe pour rester lisible pour l’utilisateur.

Figure 9 : Affichage des orthographes d’une phrase (Site Web)

3. Les contrepèteries enchevêtrées

Afin de proposer un maximum de résultats, nous nous sommes penchés vers les contrepèteries enchevêtrées. Ce type de contrepèteries consiste à l’ajout d’espaces et à la fusion de mots dans une contrepèterie.

Exemple : L’abeille coule -> la belle couille. Ici, il y a un regroupement de “l’” et “abeille” puis une séparation entre “la” et “belle”.

Afin de séparer les mots dans les phrases, une fonction récursive, c’est-à-dire qui s’appelle elle même jusqu’à une certaine condition d’arrêt, suivant l’algorithme suivant est utilisée sur les mots échangés :

```
fonction contrepeteriesEnchevetres(mot)
        listeResultats <- initialisation
        si mot est présent dans le dictionnaire
                ajouter mot à listeResultats
        fin si
        si la longueur du mot est de 1 lettre/phonème
                sortir de la fonction
        fin si
        pour chaque lettre de mot
                resultat1 <- rappeller la fonction pour le début du mot à la lettre
                resultat2 <- rappeller la fonction pour la lettre jusqu'à la fin du mot
        fin pour
        pour chaque résultat dans résultat1
                pour chaque résultat dans résultat2
                        on ajoute la concaténation des deux dans listeResultats
                fin pour
        fin pour
        retourner listeResultats
fin fonction
```

Ceci permet d’insérer tous les espaces possibles dans un mot et de vérifier si tous les fragments sont des mots qui existent.

Pour le regroupement de mots, il nous suffit de fusionner le mot actuellement sélectionné pour les échanges afin de considérer cette fusion comme un nouveau mot susceptible d’être échangé.

Il y a cependant une différence avec l’ajout d’espaces dans les mots. Dans ce cas, le nombre d’espace est limité à 3. Cela permet de meilleures performances sachant que le nombre de mots à vérifier (l'entièreté du dictionnaire, environ 300 000 mots) est bien supérieur.

Fonctionnalités de filtrage

Dans la partie précédente, nous avons détaillé les fonctionnalités permettant de rechercher des contrepèteries au sein d’un mot et d’une phrase. Ces dernières ont pour but de proposer un maximum de résultats à l’utilisateur.

Cependant, dans beaucoup de cas, nous pouvons nous retrouver avec quelques dizaines, voire centaines, de résultats.
Ou encore, les résultats ne sont peut-être pas assez pertinents pour l’utilisateur. Prenons comme exemple la contrepèterie suivante : “ôte ta lam**p**e que je **gu**ette”. Lorsqu'on effectue l’échange, les deux nouveaux mots doivent avoir la même classe grammaticale que les anciens (langue & lampe => nom, puis guette & pète => verbe). Il ne sert alors à rien de sortir des adjectifs ou encore des adverbes comme résultat puisqu’ils ne pourront pas être utilisés dans ce contexte. Il devient alors nécessaire de proposer des solutions permettant d’affiner ces résultats.

C’est pour cela que nous avons décidé d’implémenter différents filtres ayant pour but de proposer des résultats correspondant aux souhaits de l’utilisateur. Nous avons donc implémenté trois filtres :

-Filtre grammatical permettant de vérifier la classe grammaticale d’un mot
-Filtre par thème permettant de garder que des mots appartenant à un thème sélectionné
-Filtre linguistique permettant de sélectionner la langue dans laquelle effectuer les recherches
-Filtre syntaxical permettant de vérifier si une phrase possède une syntaxe correcte.

1. Programmer un crawler

Avant de commencer à implémenter les différents filtres énumérés plus haut, un problème s’est posé : notre dictionnaire n’était pas assez complet. En effet, pour pouvoir vérifier la classe grammaticale d’un mot, il faut pouvoir la stocker. Or, notre dictionnaire ne contenait que les mots suivis de leur écriture phonétique. Il a donc fallu dans un premier temps, enrichir ce dictionnaire.

Pour ce faire, nous avons implémenté un *crawler**, basé sur la technique du *scraping** afin de récupérer pour chaque mot d’une langue : son écriture phonétique, son genre, ses classes grammaticales ainsi que son nombre (singulier ou pluriel). Le scraping est une méthode qui consiste à parcourir un ou plusieurs sites web dans le but d’extraire des informations contenues dans ces derniers. Le crawler est le nom donné au programme qui utilise cette technique.

Ce crawler a été réalisé en python à l’aide des bibliothèques *Requests** et *BeautifoulSoup** qui permettent respectivement d'effectuer : une *requête HTTP** auprès d’un site web, puis de *parser** un fichier *HTML**.

Tout d’abord, nous avons recherché un site qui contenait toutes les informations dont nous aurions besoin. Ce site est *wiktionary*, un site internet définissant les mots de plus de 150 langues dont l’url est la suivante : https://www.wiktionary.org/ .
Dans le cas de notre projet, nous souhaitons créer un dictionnaire pour le français mais également pour l’anglais.

Une fois le site trouvé, nous pouvons commencer à programmer le crawler. Le principe du crawler sera le suivant :

-Effectuer une requête auprès du serveur qui aboutira par une réponse de sa part
-Récupérer les informations qui nous intéressent d’après sa réponse

À noter que la réponse est un fichier HTML.

La première étape s’effectuera via l’utilisation de la bibliothèque Requests. La deuxième étape, quant-à-elle, est plus longue à mettre en place. En effet, pour récupérer les informations qui nous intéressent, il faut être capable de les identifier au sein de la réponse. Le langage HTML est un langage de balisage, c'est-à-dire que les informations sont encapsulées dans des balises telles que <p>, ou encore <a>. Nous allons donc identifier nos informations cibles à l’aide de ses balises.

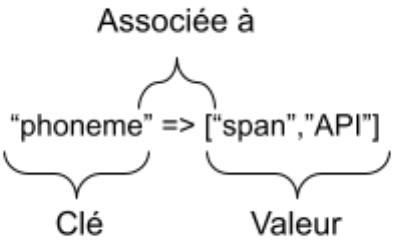
Voici un extrait du *code source** d’une page web après avoir recherché le mot “avion” sur le wiktionary français :

```
1 <p><b>avion</b>
2 <a href="/wiki/Annexe:Prononciation/fran%C3%A7ais" title="Annexe:Prononciation/français">
3 <span class="API" title="Prononciation API">\a.vjɔ̃\</span></a>
4 <span class="ligne-de-forme"><i>masculin</i></span>
```

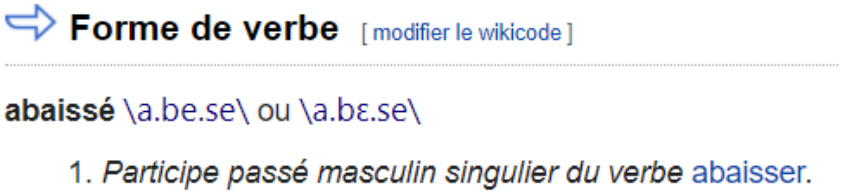
Comme on le constate à la ligne 3, l’écriture phonétique du mot ‘avion’ se situe dans une balise nommée ‘span’ qui possède une classe nommée ‘API’. C’est cette signature qui va nous permettre de récupérer l’écriture phonétique du mot avion.
Mais de manière plus générale, il va falloir rechercher, pour chaque information que l’on désire récupérer, la signature qu’il lui est attribuée à l’aide du code source.

Cette signature sera ensuite sauvegardée à l’intérieur d’une variable nommée dicoInfos. C’est un tableau associatif, c'est-à-dire, que pour une donnée en entrée elle va associer une à plusieurs valeurs. Sémantiquement, cette donnée en entrée est appelée clé et la donnée qui lui est associée est appelée valeur.
Pour dicoInfos, la clé est l’information à récupérer, et la valeur est la balise ainsi que la classe qui permettent de récupérer cette dernière.

Voici un exemple pour mieux comprendre :



Il ne reste plus qu’à vérifier si certaines informations ne seront pas à nettoyer, une fois qu’elles auront été récupérées. Par exemple, voici le résultat de recherche pour ‘abaissé’ :


Forme de verbe [modifier le wikicode]

abaissé \a.be.se\ ou \a.bɛ.se\

1. *Participe passé masculin singulier du verbe* abaisser.

Figure 10: Extrait de la page ‘abaissé sur wiktionary

Lorsque nous aurons récupéré la classe grammaticale de 'abaissé', nous aurons récupéré ‘Forme de verbe’. Même si cette appellation est correcte pour désigner sa classe grammaticale, nous souhaitons être synthétique dans les informations récupérées. De plus, ce n’est pas le seul cas de figure, il y a aussi ce préfixe qui est rajouté pour certains mots qui sont des noms, voire même des adjectifs. Nous devons alors trouver un moyen permettant de nettoyer nos réponses pour garder seulement ‘verbe’, ‘nom’ etc. C’est la dernière étape pour la mise en place du crawler. Il suffit de rentrer dans une variable, que l’on nommera infosAEnlever, les informations à enlever au sein de nos réponses.

```
1 fonction crawler(listeMot,urlSite,dicoInfos,infosAEnlever,fichierDest,isNom){
2       ouvrir le fichier nommé fichierDest
3       pour chaque mot contenu dans listeMot
4               url <- urlSite + mot
5               pageHTML <- effectuer une requête HTTP avec url
6               motInfos <- parser pageHTML à l’aide de dicoInfos
7               motInfos <- nettoyer motInfos avec infosAEnlever
8               si isNom == true
9                       motInfos <- ajouter ‘nom propre’ comme classe grammaticale
10              fin si
11              ecrire motInfos dans fichierDest
12              Si erreur
13                      afficher("informations inconnues pour ce mot")
14              fin si
15      fin pour
16 fin fonction
```

Figure 10: Algorithme du crawler

Description des paramètres de la fonction :

- listeMot => liste contenant les mots sur lesquels on veut utiliser le crawler
- urlSite => *url** du site sur lequel on va effectuer la requête au serveur (dans notre cas wiktionary)
- dicoInfos => tableau associatif contenant nos informations à récupérer
- infosAEnlever => variable contenant les informations à enlever
- fichierDest => fichier dans lequel on va écrire les résultats
- isNom => vaut vrai si on effectue une recherche sur des noms propres. Cela permet directement de mettre comme classe grammaticale “nom propre”.

2. Filtre grammatical

Une fois notre dictionnaire enrichi. Nous pouvons commencer à implémenter les différents filtres. Le premier filtre à avoir été implémenté est le filtre grammatical. Son objectif est de trier les résultats suivant leur(s) classe(s) grammaticale(s)..

Nous avons implémenté deux versions de ce filtre :

★ La première permet de garder les résultats qui possèdent la même classe grammaticale que le mot entré par l’utilisateur.
★ La seconde permet de garder les résultats qui possèdent la même classe grammaticale que celle sélectionnée par l’utilisateur.

Exemple :

Avec la première version du filtre :

- Mot entré par l’utilisateur : coder
- Classe grammaticale : verbe

➔ Donc les résultats proposés après la recherche de contrepèteries ne seront que des verbes.

- Mot entré par l’utilisateur : code
- Classes grammaticales : nom, verbe

➔ Donc les résultats proposés après la recherche de contrepèteries seront soit des verbes, soit des noms.

Avec la deuxième version du filtre :

- Mot entré par l’utilisateur : coder
- Classe grammaticale sélectionnée par l’utilisateur : adjectif

➔ Donc les résultats proposés après la recherche de contrepèteries ne seront que des adjectifs.

Pour implémenter ce filtre, il suffit de comparer les classes grammaticales entre le mot de départ (celui entré par l’utilisateur) ainsi que le mot trouvé lors de la recherche (recherche des mots initiaux ou des quadruplets).

La première étape est de vérifier quelle version a été sélectionnée par l’utilisateur (ligne 4 de l’algorithme) puis d’effectuer les tests. S’il y a concordance entre leurs classes grammaticales, alors on renvoie ‘vrai’ pour préciser que le résultat est validé par le filtre, sinon ‘faux’ sera retournée indiquant le contraire, et dans ce cas, le résultat ne sera pas gardé.

```
1 si filtre non activé
2       retourner vrai
3 fin si
4 si première version sélectionnée
5       pour chaque classe grammaticale du résultat
6       si le mot entré par l’utilisateur possède cette classe
7                       retourner vrai
8               fin si
9       fin pour
10 sinon
11      si le résultat possède la classe choisie par l’utilisateur
12              retourner vrai
13      fin si
14 fin si
15 retourner faux
```

Figure 11: Algorithme du filtre grammatical

3. Filtre par thèmes

Le deuxième filtre a avoir été implémenté est le filtre par thème. Il consiste à garder les résultats qui appartiennent à un thème donné.
Par exemple, si je sélectionne le thème vulgaire, tous les résultats seront filtrés et seuls les mots considérés comme étant grossiers ou vulgaires seront gardés.

Pour pouvoir réaliser ce filtre, nous devons posséder des dictionnaires de mots classés par thème. Si nous reprenons l’exemple précédent, pour qu’il puisse être réalisable, il nous faut un dictionnaire qui contient tous les mots vulgaires. Cependant, nous ne possédons pas un tel dictionnaire. Il a donc fallu dans un premier temps créer des dictionnaires par thème avant de pouvoir implémenter le filtre.
Pour ce faire, nous avons créé un petit crawler dont le but est seulement de récupérer les mots d’un thème indiqué. J’ai utilisé le même site, les mêmes bibliothèques et un algorithme similaire au premier présenté précédemment, c’est pour cela que l’algorithme ne sera pas détaillé ici, cependant il est disponible en annexe.

Une fois le crawler programmé, nous avons pu l’utiliser pour ajouter des thèmes d’après les souhaits du client : le thème vulgaire, non-vulgaire et le thème informatique. Le dictionnaire des mots vulgaires était déjà présent lorsque nous avons récupéré le projet, nous avons donc seulement eu besoin de récupérer le dictionnaire sur le thème de l’informatique.

L’algorithme est vraiment simple, si un mot est dans au moins un des thèmes sélectionnés par l’utilisateur, on renvoie vrai. De plus, pour gérer les filtres type ‘non-’, il suffit de vérifier si le mot n’est pas dans le dictionnaire du filtre associé. Dans le cas du filtre non-vulgaire, il suffit de retourner vrai si le mot n’est pas dans le dictionnaire vulgaire (ligne 7 de l’algorithme).

Il nous reste plus qu’à implémenter l’algorithme suivant :

```
1 fonction filtreTheme(mot, listeDico)
2       verif <- faux
3       si listeDico est vide
4               retourner true
5       fin si
6       pour chaque dico dans listeDico
7               si c'est un thème inverse
8                       si mot n'est pas dans dico
9                               verif <- vrai
10                              arrêter la boucle
11                      fin si
12              sinon
13                      si mot est dans dico
14                              verif <- vrai
15                              arrêter la boucle
16                      fin si
17              fin si
18      fin pour
19      retourner verif
20 fin fonction
```

Figure 12: Algorithme du filtre par thème

4. Filtre linguistique

Le troisième filtre à avoir été implémenté est le filtre linguistique. Il consiste à proposer la recherche de contrepèteries dans différentes langues.
Seul le français était disponible à l’heure actuelle. Cependant, M. LAFOURCADE souhaitait proposer des recherches de contrepèteries en anglais aux utilisateurs du site web, c’est pour cela que nous avons ajouté l’anglais comme langue disponible.

Pour pouvoir effectuer des recherches dans d’autres langues que le français, il suffit d’avoir le même dictionnaire que nous avons chargé, grâce au crawler expliqué dans la partie 1, mais dans la langue à ajouter.
Avec la manière dont ce dernier a été programmé, il n’y a pas nécessité d’en programmer un nouveau, puisqu’il suffit de lui passer en paramètre les bonnes informations qui correspondent à la langue anglaise. Typiquement l’url changera ainsi que les identifiants des informations à récupérer (dicoInfos).

De manière plus générale, le crawler programmé plus haut permet de récupérer l’écriture phonétique, le genre, la classe grammaticale ainsi que la pluralité d’un mot de n’importe quelle langue.

5. Filtre syntaxical

Le dernier filtre à avoir été implémenté est le filtre syntaxical. Son but est de vérifier si une phrase est française, c’est-à-dire si elle possède une syntaxe correcte.
Pour pouvoir le réaliser nous nous sommes basés sur la *programmation en langage naturel**. Ce type de programmation consiste à doter des programmes informatiques de la capacité de comprendre le langage humain.

Nous avons réalisé une première version de ce filtre seulement en python puisque ce dernier possédait une bibliothèque spécialement conçue pour ce type de programmation : NLTK. Elle possède toutes les techniques nécessaires pour pouvoir réaliser ce que nous voulons.

A. La tokenization

La tokenization (tokenization en anglais) est un processus qui consiste à diviser un texte en sous parties nommées token. Le but est de pouvoir transformer des données non structurées en données structurées. Concrètement, dans notre cas, cela consiste à travailler avec des morceaux de textes qui auront toujours autant de sens même s’ils ont été sortis du contexte de ce dernier.

Il existe deux types de tokenization :

- ➢ Tokenization par mots (word tokenize): Consiste à diviser un texte en mots -> Utile si nous voulons savoir la fréquence d’apparition des mots
- ➢ Tokenization par phrases (sentence tokenize) : Consiste à diviser un texte en phrases -> Utile pour vérifier le lien entre les mots

Exemple d’utilisation :

```
>>> exemple = "je regarde une jolie personne"
```

Lorsqu’on utilise la tokenization par phrase :

```
>>> sent_tokenize(exemple) #On applique la tokenization par phrase
["Je regarde une jolie personne"]
```

On remarque que le résultat est la phrase elle-même.
Alors que si on applique la tokenization par mots, on remarque que ce n’est pas chaque phrase mais bien chaque mot, un à un qui sont séparés :

```
>>> word_tokenize(exemple) #On applique la tokenization par mots
['je', 'regarde', 'une', 'jolie', 'personne']
```

Dans notre cas, nous avons utilisé la tokenization par mots puisque notre but est de vérifier si une phrase est française. Si nous demandons de séparer une phrase en plusieurs phrases, le résultat sera elle-même. Cette démarche ne nous apportera rien en plus contrairement à la tokenization par mots.

B. Part-of-Speech tag (POS Tag)

Après avoir segmenté notre phrase en mots, il faut dorénavant assigner un rôle à ces mots. Pour ce faire, nous allons leur attribuer un tag correspondant à leur rôle, autrement dit leur classe grammaticale, dans le contexte de la phrase grâce au Part-of-Speech tag.
Vous trouverez la liste des familles de tag possibles en annexe.

Voici un exemple d’utilisation :

```
>>> nltk.pos_tag(exemple_tokenize) #on applique le pos tag
[('je', 'PRON'), ('regarde', 'VERB'), ('une', 'DET'), ('jolie', 'ADJ'), ('personne', 'NOUN')]
```

Pour chaque mot, la fonction lui a ajouté un tag suivant son rôle au sein de la phrase. Il existe une très grande liste de tags suivis de leur définition, mais pour des raisons de pertinences nous ne mettrons qu’un extrait correspondant à l’exemple précédent :

Tags commençant par	Correspondent à
Det	Déterminant
Noun	Nom
Pron	Pronon
Verb	Verbe
Adj	Adjectif

C. Chunking

Une fois que nous avons divisé notre phrase en mots, puis associé à chacun d’eux un rôle. Nous pouvons désormais rechercher des patrons grammaticaux au sein de la phrase. Ces patrons grammaticaux représentent des règles de grammaire et notre but est de tester si une phrase possède un patron grammatical. Cette démarche est appelée chunking.

Le but est de définir toutes les règles grammaticales qui existent en français, puis si la phrase possède un des patrons définis, alors elle sera considérée comme étant correctement française.

Pour définir un patron grammatical, il faut se baser sur la syntaxe des tags proposés par la fonctionnalité POS-Tag ainsi que sur les *expressions rationnelles** nous permettant d’indiquer leurs fréquences d’apparitions. Une expression rationnelle est une chaîne de caractères permettant de définir un ensemble de chaîne de caractères.

?	Présent 0 à 1 fois
+	Présent 1 à n fois
*	Présent 0 à n fois

Figure 13 : Tableau des trois quantificateurs les plus répandus dans les expressions rationnelles

Voici comment définir la règle simple suivante : Déterminant - Nom - Adjectif
Un déterminant peut ne pas être présent mais le sera au maximum une fois, il y aura obligatoirement un nom et un adjectif sera présent au maximum 1 fois.
On va l’écrire de la manière suivante => *Patron : {<DET>?<NOUN><ADJ>?}.*
Les quantificateurs se placent toujours après le motif qu’il représente, ici, “<DET>?” signifie un déterminant qui se répète 0 à 1 fois.
Maintenant, nous pouvons mettre en place l’algorithme permettant de vérifier si une phrase est française:

```
fonction filtreSyntaxical(phrase)
        on définit nos patrons grammaticaux
        on parse la phrase pour essayer de retrouver des patrons
        si la phrase possède un patron
                on retourne vrai
        sinon
                on retourne faux
fin fonction
```

Figure 14 : Algorithme du filtre syntaxical

Par manque de temps, nous n’avons pas pu pousser ce filtre au maximum. A l’heure actuelle, il permet seulement de vérifier une règle simple : Pronom ou Déterminant + Nom - Verbe - Adjectif + Nom
Voici néanmoins un exemple de test réalisé pour vous montrez la première version de ce filtre :

```
Phrase : je regarde une jolie personne -> juste
Phrase : je une jolie personne -> pas juste
Phrase : je regarde une personne -> juste
Phrase : la personne regarde -> juste
Phrase : je regarde personne -> juste
Phrase : regarde une jolie personne -> pas juste
Phrase : une regarde jolie personne -> juste
```

Nous sommes partis d’une même phrase où nous avons inversé l’ordre des mots afin d’en produire de nouvelles. Il détecte correctement celles qui sont correctes de celles qui ne le sont pas.
Pour pousser le filtre à son maximum, il faut l’enrichir de règles grammaticales afin de pouvoir vérifier des phrases plus complexes.

Côté applicatif

1. Le site web

A- Arrivée sur la page d'accueil

Après connexion au site web, la première page vue par l'utilisateur est usuellement la page index.html, ce qui est aussi le cas pour notre site. Cette page contient la définition d’une contrepèterie et quelques exemples afin qu’un utilisateur ne connaissant pas les contrepèteries puisse se faire une idée du contenu et de l’objectif de notre site.
En haut de cette page d'accueil, un menu est présent, permettant à l’utilisateur d’accéder à plus de fonctionnalités qui seront décrites dans les parties suivantes. Un clic sur le lien “Présentation” dans le menu permet de revenir sur cette page d’accueil.
Le site contenait déjà toutes les pages d’informations sur les contrepèteries, nous n’avons pas pris de temps pour les retravailler. Nous avons préféré focaliser notre travail sur l’ajout de fonctionnalités.

B- Classification

La page “Classification” est une des pages que nous n’avons pas amélioré. Cependant, il nous semble bon de rappeler son utilité. Cette page sert à classifier les contrepèteries comme l’a fait Joël Martin dans ses ouvrages. Elle est en quelque sorte une “vitrine des contrepèteries” : de nombreux exemples y sont parsemés avec leur catégories ainsi que leurs solutions.

Figure 15 : Classification : contrepèteries classiques

C- Recherche sur les mots

En arrivant sur cette page, l’utilisateur peut tout d’abord choisir le type de contrepèteries qu’il souhaite réaliser : en lettres (choix par défaut) ou en phonèmes.
Dans le cas où cet utilisateur visite ce site pour la 1ère fois, il pourrait avoir besoin de quelques conseils ce pourquoi nous avons prévu un lien “Besoin d’aide?” dans le coin supérieur-droit de la page, permettant d’accéder à un tutoriel.
Plus bas, une icône ✓ à côté de “Dictionnaire chargé” permet de vérifier le bon fonctionnement du site et la progression du chargement des dictionnaires (4 différents dictionnaires sont chargés au lancement de cette page, un dictionnaire de mots, un de phonèmes, un de classes grammaticales et enfin un dictionnaire contenant les mots considérés commes vulgaires).

Figure 16 : vue globale de l’interface pour la recherche dans des mots

En dessous, un bouton de “Paramètres” est visible. Un clic dessus dévoile de nouveaux champs, modifiables par l’utilisateur. Tout d’abord, il est possible de remplacer le nombre de lettres à échanger dans le mot renseigné, puis le mot recherché.
*Exemple : en renseignant “co**lle**” avec **2** lettres dans mot renseigné **1** lettre dans mot recherché, on pourra obtenir “co**de**”.*

Il est aussi possible de choisir la longueur minimale et/ou maximale des quadruplets trouvés.
Exemple : avec 3 en longueur minimale et 5 en maximale, et en choisissant le couple ‘projet - promet’, on peut trouver les quadruplets ‘met - jet’.
Si on modifie la longueur minimale à 4, on ne trouvera plus ‘met - jet’ dans nos résultats puisque ces quadruplets font 3 lettres.

Une dernière fonctionnalité est présente dans cette zone de paramètres : il s’agit du filtrage. Différents filtres sont proposés, pouvant être activés/désactivés à volonté avant la recherche.
Les résultats ne correspondant pas aux filtres sélectionnés ne sont pas affichés à l’écran.
Le premier filtre visible est l’activation/désactivation des mots coupés, c’est-à-dire une contrepèterie avec insertion d’un espace (permettant de séparer 1 mot en 2).

Ensuite, l’utilisateur peut choisir d’afficher les mots possédant seulement les mêmes classes grammaticale, une fonction très utile lorsque l’on cherche à faire une contrepèterie puisque pour que la phrase après contrepèterie ait un sens, il est vital que les 2 mots partagent une même classe grammaticale.
Le dernier paramètre non cité est le réglage du filtre grossier. Ce filtre possède 3 configuration aux usages bien différents : seulement grossiers (seuls les mots vulgaires seront affichés), aucun mot grossier ou filtre grossier désactivé (pas de tri).

Bien sûr, il n’est pas nécessaire de remplir ces paramètres supplémentaires, et un utilisateur peut tout simplement lancer une recherche de contrepèterie sur un mot sans même ouvrir le menu de paramètres (dans ce cas la recherche prendra des paramètres par défaut).

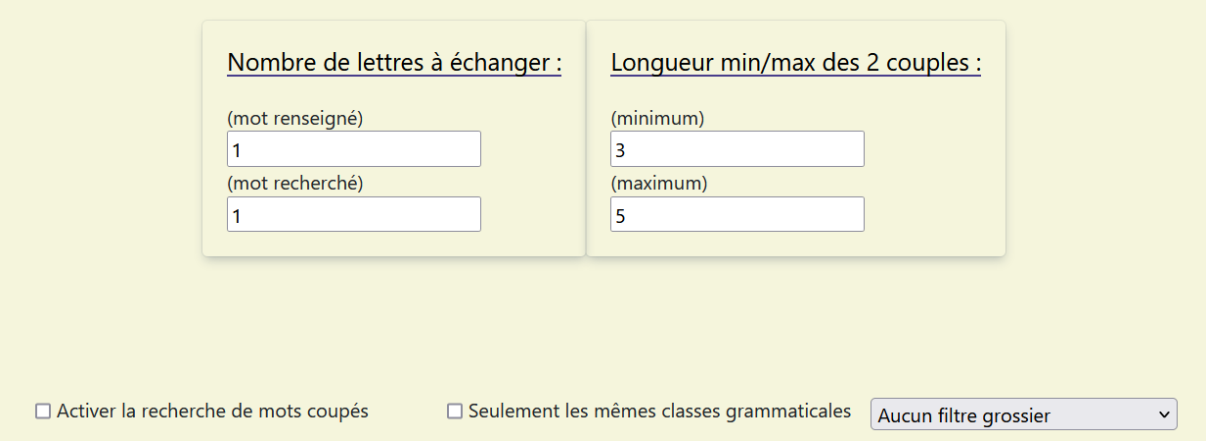


Figure 17 : vue du menu de paramètres

Note : une fonction de filtres dynamiques a également été implémentée. Ils sont utilisés lors de l’affichage des quadruplets et permettent de griser tous les résultats ne correspondant pas aux filtres dynamiques sélectionnés.

Toutes ces fonctionnalités ont été rajoutées par nos soins pour développer la recherche de contrepèteries qui n’était encore qu’une ébauche lorsque nous avons récupéré le projet. En effet, seul un champ et un bouton permettait de chercher une contrepèterie sur un mot. Il fallait charger le dictionnaire à la main, et seul l’affichage de couple était disponible (pas de quadruplets)

Figure 18 : Ancien site : recherche sur mots

D- Recherche sur les phrases

La page “recherche sur phrase” permet d’aider à trouver une solution d’une contrepèterie. L’utilisateur doit tout d’abord sélectionner un mode : “lettre” ou “phonème”. Ensuite, il doit rentrer sa phrase dans l’emplacement réservé puis cliquer sur “Lancer la recherche”. Comme il s’agit d’une des fonctionnalités les plus attendues de notre site web et qui sera sans doute l’une des plus utilisées, nous avons essayé de la rendre la plus simple possible pour tous les utilisateurs. Ici, il n’y a donc pas de paramètres et il est extrêmement simple d’effectuer une recherche.

Lorsqu’on choisit le mode “phonème”, le site travaille avec les sons : l’orthographe des résultats est souvent peu compréhensible. Pour régler ce problème, l’utilisateur peut cliquer sur le résultat pour afficher différents exemples d’orthographes. *(voir figure 9)*

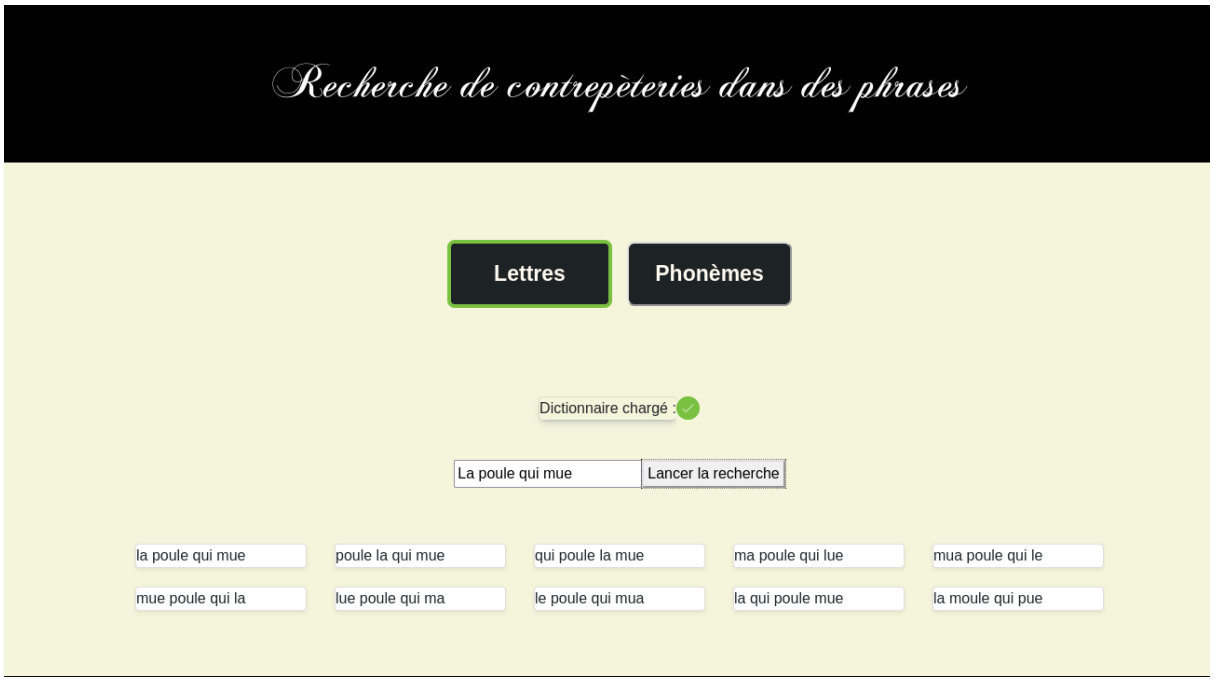


Figure 19 : Résultats de recherche sur une phrase

Il s’agit d’une fonctionnalité que nous avons reprise de zéro. L’ancienne page ne contenait que peu de code et le bouton de recherche provoquait un arrêt systématique du navigateur.

E- Trouver la contrepèterie !

Sur le site, 2 jeux sont maintenant disponibles avec différents modes. Le premier jeu “Trouver la contrepèterie !” était déjà présent. Nous l’avons amélioré sur plusieurs points : le design, les meilleurs scores et les difficultés. Ce jeu consiste, comme son nom l’indique, à trouver les contrepèteries. Tout d’abord il faut choisir son niveau de difficulté : “Facile”, “Moyen”, “Difficile”. Les contrepèteries sont plus difficiles à trouver selon les modes et le temps de réflexion est aussi diminué.
Une fois le mode choisi, il suffit de cliquer sur “Commencez la partie”. Après cela, le timer se lance et une contrepèterie apparaît. Pour jouer, il faut cliquer sur la syllabe/lettre de la phrase à échanger pour obtenir une contrepèterie. Lorsqu’on clique, l’élément sélectionné passe en bleu et si la réponse est la bonne, le score augmente et le jeu passe à la contrepèterie suivante.

Une fois toutes les réponses trouvées ou un échec sur un des niveaux, le jeu s'arrête et le score est stocké en local dans un classement.

Figure 20 : Exemple de partie sur “Trouver la contrepèterie !”

F- Quizz des mots

Le deuxième jeu présent sur le site a été développé entièrement par nos soins. Il reprend certains algorithmes (chronomètre) de l’autre jeu mais il s’agit d’un jeu innovant imaginé par notre équipe. Toujours sur le thème des contrepèteries, l’objectif est, pour un mot donné, de trouver un mot qui peut être une contrepèterie par un seul échange de lettres ou de sons.

Exemple : si le jeu nous donne le mot “**p**oule”, une réponse serait “**m**oule”.

Deux modes de jeux sont disponibles et suivent le même principe. Le premier est le mode “challenge”, ici 10 mots sont choisis aléatoirement et il faut trouver une contrepèterie pour un mot en 30 secondes puis l’on passe au suivant. Les séries de bonnes réponses rapportent plus de points. Après chaque round, un message de réussite ou d’échec s’affiche, et la liste des réponses possibles pour le mot précédent est indiquée.

Figure 21 : Exemple de partie challenge sur “Quiz des mots!”

Figure 22 : Exemple de partie challenge sur "Quiz des mots !”

Le deuxième mode est appelé “détente”, il y a toujours 10 mots mais ici il n’est plus question d’erreur ou de chronomètre. Le temps est illimité, le nombre de contrepèteries possible est indiqué pour chaque mot et l’on peut consulter à tout moment les réponses.

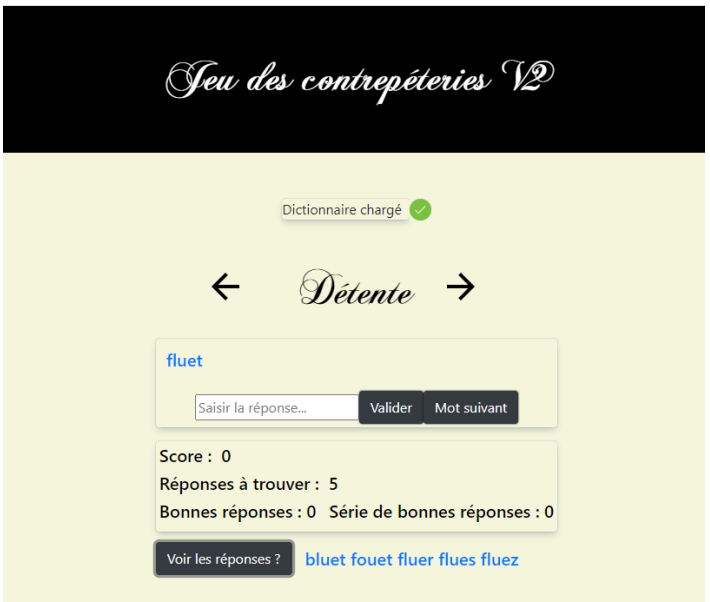


Figure 23 : Exemple de partie détente sur "Quiz des mots!”

Pour les deux modes, les différents mots à trouver ainsi que les réponses correctes du mot précédent sont cliquables et renvoient sur Wiktionary. Nous avons constitué ces jeux dans l’optique de s’améliorer et de découvrir la langue française de manière ludique.

2. L’application sur terminal

A- Le démarrage de l’application

Lors du démarrage de notre application, la première chose affichée est un choix vous permettant de choisir la langue dans laquelle on souhaite chercher les contrepèteries parmi anglais et français, ce qui n’était pas présent lorsque nous avons repris ce projet.

```
Choisissez la langue :

1 - fr

2 - en


Entré le numéro de la langue voulue :
```

Après cela, les dictionnaires de la langue en question chargent pendant quelques secondes. On arrive ensuite au menu principal de notre application séparant les fonctionnalités majeures de celle-ci.

```
Sélectionnez le mode que vous souhaitez :

a. Recherche de contrepèteries dans un mot
z. Recherche de contrepèteries dans une phrase
e. Configuration des filtres
r. Quitter
```

A noter que la navigation des menus (hormis le choix de la langue) se fait avec les premières lettres du clavier a,z,e… Le menu principal nous permet donc d’accéder au deux modes de recherche de contrepèteries. Celle dans les mots, qui compare un mot à tous les autres du dictionnaire, et dans les phrases, qui recherche les contrepèteries entre les mots d’une phrase.

B- La recherche dans les mots

Du côté de la recherche dans un mot, il nous est d’abord demandé celui sur lequel la recherche se portera.

```
historique :

1  :  code

2  :  bonjour

Pour utiliser l'historique, veuillez entrer le numéro du mot.
Pour effectuer une recherche rapide entre deux mots, séparez les avec un '/' (mot1/mot2).
Mot :
```

On peut voir la présence d’un historique que nous avons ajouté, permettant de sélectionner à nouveau les mots choisis précédemment.

On arrive ensuite au menu permettant de sélectionner différentes options pour la recherche dans les mots.

```
Sélectionner le type de recherche :
        a - recherche par lettre
        z - recherche par phonèmes
        e - recherche complète
        r - Retour au menu
Choix du mode :
```

La recherche des contrepèteries est séparée en deux catégories, les lettres et les phonèmes. Peu importe, le choix, la recherche sera effectuée en prenant en compte les contrepèteries coupées présentées précédemment, qui n'étaient pas présentes dans l’ancienne version. Nous avons cependant décidé d’ajouter la possibilité de rechercher les deux en même temps matérialisé par le mode ‘recherche complète’. Une fois une option sélectionnée on peut voir des nouveautés par rapport au projet précédent. On peut choisir nous même la taille des échanges dans la contrepèterie, tandis que ceux-ci étaient limités à 1 lettre auparavant. On remarque aussi la présence d’une prédiction du temps d’exécution fait grâce à un script shell prévenant l’utilisateur si la recherche demandée sera très chronophage. Il y a aussi une amélioration de l’affichage avec des menus plus clairs et un affichage page par page.

```
page 1/5


a: retourner au menu
z: sélectionner un nouveau mot
e: effectuer la recherche avec les lettres
r: page précédente
t: page suivante
ou numéro de l'échange qui vous intéresse :
```

Ici, les résultats ne sont pas montrés afin de gagner de la place.

C- La recherche dans les phrases

A l’inverse de la recherche dans les mots, la recherche dans les phrases demande d’abord quel type de recherche on veut effectuer avec, ici encore, la possibilité de rechercher à la fois avec les lettres et les phonèmes que nous avons ajoutés.

```
Voulez-vous échanger

                    a. Les lettres
                    z. Les sons
                    e. Lettres et sons
                    r. retour
```

Il nous est ensuite demandé de rentrer la phrase souhaitée puis, la recherche se lance de la manière vue dans la section dédiée.

Une fois la recherche effectuée, on retrouve un nouveau menu, similaire à celui de la recherche par mot.

```
page 1/625

Nombre de résultats pour les échanges avec les phonèmes :  31229

a - quitter l'application
z - revenir au menu principal
e - page précédente
r - page suivante
ou saisissez un des index pour obtenir toutes les orthographes :
```

Si l’orthographe d’un résultat semble faux, comme par exemple :

```
29  --> la bel couille
```

Il est désormais possible d’afficher différentes orthographes pour chacun des mots :

```
ou saisissez un des index pour obtenir toutes les orthographes : 29

Affichage des différentes orthographes. Une orthographe de chaque mot vous est proposée :
La bel couille
Là bêle couilles
Lacs bêlent couille
La bêles couille
La belle couille
La belles couille
```

Ici, on affiche une seule fois chacune des orthographes des mots au lieu de faire chacune des phrases possibles afin de conserver un nombre raisonnable de résultats. Sinon nous risquerions de nous retrouver avec un produit cartésien. On laisse donc à l’utilisateur le soin de choisir l’orthographe qu’il veut pour chacun des mots.

D- La configuration des filtres

Depuis le menu principal, il est possible d’accéder au menu de configuration des filtres. Il se présente comme ceci :

```
Voici votre configuration actuelle :
        0 FiltreGrammatical  -  Oui
        1 Themes  -  []
        2 MotCoupe  -  Oui
        3 EffacerComplétement  -  Oui
a: Retour au menu
Entrer le numéro de la configuration à changer :
```

Le premier filtre est le filtre grammatical. Son but est de conserver uniquement les résultats ayant du sens. Son fonctionnement est expliqué plus en détail dans la section dédiée. Ensuite, on a les thèmes. Ici, l’intérêt est de conserver uniquement les résultats ayant du vocabulaire appartenant à certains champs lexicaux. Nous proposons les filtres suivants : vulgaire uniquement, non-vulgaire uniquement et informatique. Le filtre des mots coupés permet de les désactiver si les recherches prennent trop de temps ou que le nombre de résultats est trop grand. Pour finir, on peut choisir l’option d’effacer les écrans précédents du terminal afin d’avoir un affichage plus propre, on ne peut cependant pas remonter le fil du terminal afin de retrouver les anciens résultats.

Bilan technique

Pour résumer, la demande initiale était de poursuivre le travail déjà entamé sur deux projets afin de proposer deux supports, une application console et un site web, permettant d’apprendre et de guider les utilisateurs dans la création et la recherche de contrepèteries.

Pour répondre à cette demande, nous sommes passés par deux processus.
Le premier était d’ajouter des nouveaux types de contrepèteries afin d’élargir au maximum les réponses proposées. Nous avons permis l’échange de plusieurs lettres par plusieurs lettres (respectivements phonèmes) pour la recherche dans un mot et dans une phrase. Les contrepèteries comprenant la césure d’un mot ainsi que la soudure entre plusieurs mots ont également été ajoutées.
Le second consistait à proposer des moyens de filtrer ces résultats, souvent trop nombreux et non raccord aux désirs de l’utilisateur. Ainsi, un filtre grammatical, permettant de filtrer par rapport aux classes grammaticales, un filtre par thème, permettant de filtrer suivant des thèmes ainsi qu’un filtre linguistique, permettant la recherche dans plusieurs langues ont été implémentés.

Vous pouvez retrouver le site amélioré à l’adresse suivante :
https://sancy.iut-clermont.uca.fr/~lafourcade/contrepetries-regroupees/contrepeterie/fr/aide_a_la_contrepeterie.html

Malgré les avancées que nous avons produites sur ces projets, il reste encore quelques points sur lesquels nous aurions aimé pouvoir travailler, voir finir, pour ceux que nous avons commencés si nous avions eu plus de temps. Notamment, l’utilisation de la bibliothèque NLTK pour implémenter le filtre syntaxical, est une tâche que nous aurions aimé approfondir au vue de la puissance de cette bibliothèque. C’est une des possibilités d’évolutions du projet dans le cas d’une reprise. L’autre possibilité d’évolution concerne l’implémentation de nouveaux types de contrepèteries. En effet, au départ du projet nous avions planifié l’implémentation des contrepèteries circulaires. Tâche, que nous avions dû oublier au profit d’autres.

Il reste encore plusieurs débouchés concernant une suite pour ce projet, pour lesquels nous espérons qu'elles trouveront preneur !

English Summary

What Was Already Done:

As said previously, in this project, we were given the task to upgrade two projects made last year and two years ago. During the 2019/2020 school year, a team of students from our IUT had a tutored project with a theme similar to ours. For their project about spoonerism, they began the creation of a website to showcase what a spoonerism is and a search engine for basic spoonerism. The website also had a small game around spoonerisms. Then, in 2020/2021 a team of students that used to be in prep’Isima was asked to continue the work previously made. Instead of upgrading the website, they decided to restart the project to make a python app that would lay on a better made code structure. This app gave less context about spoonerisms but had a more developed search engine.

What we added:

As we mentioned in the “Coté Applicatif” section, we improved those two projects quite a lot. First, we will talk about the python terminal app. We will go over it first because most changes that were initially added to the python program were then copied onto the website to keep the search up-to-date on both supports. Then, we will talk about other changes made to the website. Keep in mind that everything that applies to letters also applies for sounds.

The most notable change on the python app is the widening and complexification of spoonerism's research both between words and in sentences. First, in words, the research used to only be able to either search swaps between two single letters or between certain parts of the word for an unspecified amount of letters. Those two uses were handled separately and we decided to regroup them and change the searching method to something that can handle searching spoonerisms taking any number of letters in the original word and replacing them with any number of letters. Searching spoonerisms with several letters is now included in the regular search. In addition the research adds spaces in the newly created word, making spoonerisms forming two words possible.

We also greatly improved the algorithms searching for spoonerisms in sentences. As for the words, it used to only try swapping one letter for another and checking if the sentence was still formed of correct words. We added similar functionalities to the research in sentences to the one for words which are : making the research more general and adding spaces in words. However we added another functionality which is merging words together, creating new longer words instead of splitting them. We also explored and did some tests with NaturalLanguageToolKit aiming to keep phrases grammatically correct but did not have time to achieve anything with it. The tests are still in the projects for those who want to keep digging this idea.

As more general improvements, we added progress bars and runtimes predictions in the python app as well as modifying menus.

Now we will talk about the changes made to the website which aren't the same as those on the python app. Most of these changes are display and design changes. First of all, the main menu for searching spoonerisms in words has been completely reworked to allow the user to pick options for the search. The display of resultats was also changed from simple words to interactive buttons that can then display the other words allowing to complete the spoonerism.

example:

runny - babbit
bunny - rabbit

Dynamic options were also added to be able to search in the results. On top of that, a new game was added to test your skills in spoonerism.

Conclusion

Ce travail nous a tout d’abord permis de répondre à la problématique évoquée un peu plus haut dans ce rapport : comment, en se basant sur des réalisations préalables, pouvons-nous créer une application console et un site web permettant d’initier les utilisateurs à l’art des contrepèteries ?

Aujourd’hui, après avoir réalisé ce projet nous pouvons dire qu’il est tout à fait possible de créer des algorithmes réalisant ces contrepèteries, en utilisant Javascript et Python, mais aussi d’autres outils comme un Crawler web. En utilisant ces langages tout au long de ce projet, nous avons pu approfondir nos connaissances et compétences autant en Javascript qu’en Python, ce qui nous permet d’avoir aujourd’hui une longueur d’avance dans ces langages sur d’autres étudiants de notre promo.

Grâce à ce sujet plus qu’intéressant et une bonne cohésion d’équipe, nous avons grandement apprécié de travailler ensemble sur ce projet.
Les retours fréquents de M. LAFOURCADE concernant l’avancée du projet et les points à modifier nous ont permis une première expérience de projet en méthode “agile”, soit celle utilisée en entreprise.
Nous remercions encore une fois M. LAFOURCADE pour nous avoir suivis et conseillés lors de ce projet, et les anciennes équipes de l’IUT et de prép’ISIMA nous ayant précédés.

Nous espérons sincèrement que les utilisateurs apprécieront le site web et l’application console autant que nous avons aimé les développer.

À destination des équipes suivantes

Nous avons tenté de commenter au maximum les différents fichiers afin de les rendre plus facilement lisibles et compréhensibles. Il est probable que ce code, même commenté, reste assez complexe pour quelqu’un n’y étant pas habitué. N’hésitez pas à nous contacter à une des adresses mail suivantes : mael.chaumont15@gmail.com, simon.dallet63@gmail.com, jules.duteyrat@etu.uca.fr, louisperret0342@gmail.com

Bibliographie/Webographie :

- Documentation de la bibliothèque Requests : https://docs.python-requests.org/en/latest/
- Documentation de la bibliothèque BeautifulSoup : https://python.doctor/page-beautifulsoup-html-parser-python-library-xml
- Documentation de la bibliothèque NLTK : https://www.nltk.org/
- Livre de Joël Martin : *La bible du contrepet*
- Application mobile proposant une contrepèterie par jour : Contrepet.
  ->Nous a été utile pour tester nos algorithmes.

Lexique

Méthode agile : Méthodologie de gestion de projet basée sur la fixation d’objectifs à court terme, qui peuvent-être ajustés suivant les souhaits du client, mais aussi sur une relation forte entre l’équipe et le client.

.csv : Extension de fichier où les informations sont séparés par des virgules, dont le nom Comma-separated value.

AJAC : Élève redoublant qui doit repasser un semestre sur les deux de son année.

Crawler : Programme qui parcourt des sites web dans l’optique de récupérer et de stocker de l’information dans des fichiers. Dans la plupart des cas, c’est souvent un robot qui effectue cette recherche de manière autonome, et ce, sur plusieurs sites web. Par exemple, le robot de google

Requests : Bibliothèque python qui permet d’effectuer des requêtes HTTP. Plus précisément de pouvoir des requêtes mais également de traîter la réponse.

Beautiful Soup : Bibliothèque python qui permet de parser des fichiers sous format XML et HTML.

HTTP : Protocole de communication client-serveur utilisé sur le web.

Requête HTTP : Envoie d’un message d’un client à destination d’un serveur web distant.

Parser : Parcourir le contenu d’un fichier dans l’optique d’analyser sa syntaxe, voir d’en extraire certaines informations.

HTML : Langage informatique permettant de réaliser les interfaces de site web.

Code source : Texte qui présente les instructions composant un programme sous une forme lisible, telles qu'elles ont été écrites dans un langage de programmation.

Url : Chaîne de caractères qui identifie une ressource disponible sur le web

Programmation en langage naturel : Type de programmation visant à doter des programmes informatiques de la capacité de comprendre le langage humain. Elle représente également une branche de l’intelligence artificielle.

Expression rationnelle : Chaîne de caractères qui décrit, au travers d’une syntaxe spécifique, un ensemble possible de chaînes de caractères.

Annexe

```
1 fonction crawler(urlSite,dicoInfos,infosAEnlever,fichierDest){
2        listeResultat <- initialisation de la liste des résultats
3        pour chaque lettre de l’alphabet
4                url <- urlSite + lettre
5         pageHTML <- effectuer une requête HTTP avec url
6                motInfos <- parser pageHTML à l'aide de dicoInfos
7                motInfos <- nettoyer motInfos avec infosAEnlever
8                listeResultat <- ajouter les résultats de motInfos
9    fin pour
10   écrire listeResultat dans fichierDest
11 fin fonction
```

Figure 24 : Algorithme du crawler pour créer des dictionnaires par thème

Remarque : A la ligne 3, on concatène à l’url passée en paramètre chaque lettre de l’alphabet, afin de parcourir les différentes pages du site.
Si vous allez à cette url :
https://fr.wiktionary.org/w/index.php?title=Cat%C3%A9gorie:Lexique_en_fran%C3%A7ais_de_l%E2%80%99informatique&from=%22
Vous remarquerez qu’il y a index au début qui permet de classer les mots par ordre alphabétique. On se base donc dessus pour pouvoir tous les récupérer.

Part-of-speech	Equivalent français
Noun	Nom
Pronoun	Pronom
Adjective	Adjectif
Verb	Verbe
Adverb	Adverbe
Preposition	Préposition
Conjunction	Conjonction
Interjection	Interjection

Figure 25 : Tableau des familles de tags possibles avec POS-tag de NLTK