Développement du dictionnaire de mots croisés MCWiktionnaire/MCWikti

Guy Lapalme
RALI-DIRO
Université de Montréal
Février 2025

Ce texte décrit les motivations et l'organisation de l'application que j'ai développée au cours des dernières années pour m'aider à résoudre des problèmes de mots croisés, même si je ne l'utilise qu'en dernier recours. Je cherche toujours à résoudre une grille sans utiliser de ressources externes. En cas de *blocage*, j'utilise d'abord mes dictionnaires *papier* avant d'avoir recours à mon application ou à des recherches sur le web. Cette application gratuite ne recueille aucune information sur les usagers. Je n'ai jamais tenté de la vendre ou d'en faire la promotion, je la considère comme une opportunité d'apprentissage et d'expérimentation.

1. Motivations

Cette application publiée en version macOS (MCWitionnaire) et iOS (MCWikti) permet d'accéder à plus de 1,6 million de formes tirées du Wiktionnaire. Le sigle MC qui débute le titre des applications ne signifie pas *Macintosh* en abrégé, mais plutôt « *Mots Croisés* ».

Ce développement m'a permis d'apprendre le langage de programmation Swift. Les premiers essais de structuration du dictionnaire ont été faits en Swift 1.0. L'adaptation à travers les différentes versions de Swift n'a pas toujours été facile, car l'interface de programmation (API) pour accéder aux caractères d'une chaîne a changé deux fois au cours des premières versions de Swift. Heureusement, ce mécanisme est maintenant stabilisé…

J'ai pu aussi expérimenter le processus de développement et de publication d'une application macOS et iOS.

- En août 2019, ont été publiées des versions macOS et iOS développées respectivement avec AppKit et UIKit. Ces deux versions utilisaient sensiblement la même structure de programme, d'interface usager et de dictionnaire, mais chaque plateforme avait une application distincte. Ces versions utilisaient des `StoryBoard` avec des `IBOutlet` définissant l'état du système et des `IBAction,` les traitements associés.
- En janvier 2022, les deux applications ont été réunies en une seule en utilisant SwiftUI. Ceci a permis de réorganiser et d'unifier le code. J'ai toutefois constaté que l'adaptation à chaque plateforme n'est pas aussi directe que le prétend la publicité d'Apple. Il faut parfois utiliser des directives de compilation pour masquer ou afficher certains bouts de code qui dépendent du système cible. De plus, certaines particularités de AppKit/UIKit, notamment les *NSAttributedString* ou le contrôle de l'édition des champs de texte, n'ont pas (encore) leurs équivalents en SwiftUI. Il a donc fallu développer des ponts avec les *vieilles* interfaces de programmation. Mais la grande majorité de l'interface et de la logique de l'application sont communes aux deux plateformes.

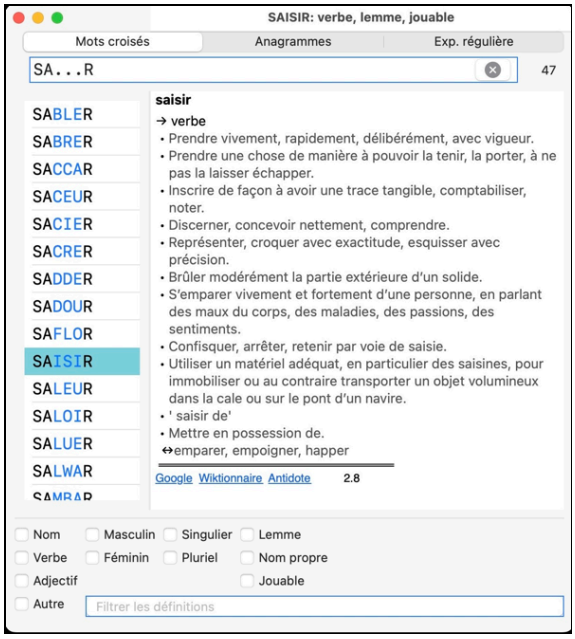
Le principal défi algorithmique et tout l'intérêt de ce projet est la recherche rapide de mots dans un dictionnaire de plus d'un million et demi de formes, en ne spécifiant que quelques lettres à certaines positions. Étant donné la vitesse des processeurs modernes, il est tout à fait possible d'effectuer une recherche avec une expression régulière en précisant les lettres connues aux endroits appropriés et en indiquant un `.` pour les lettres non spécifiées. Cette recherche ne prend que quelques secondes, ce qui est amplement suffisant pour un amateur de mots croisés qui souvent passe plusieurs heures à compléter sa grille!

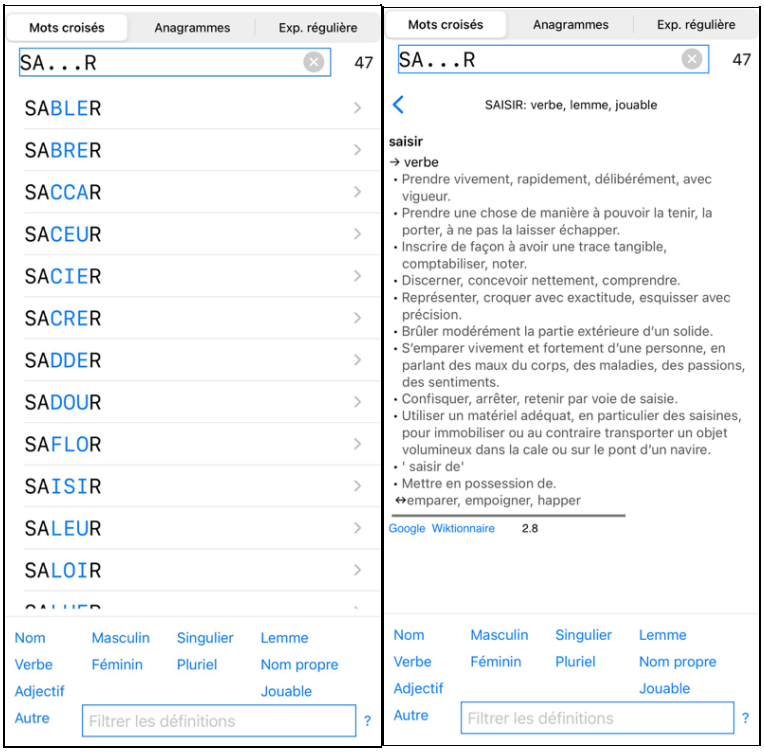
Or, il est possible d'être beaucoup plus *astucieux* en considérant l'ensemble des mots du dictionnaire comme un *langage* pour lequel on développe un automate minimal pour le générer. Il est ensuite possible de parcourir rapidement cet automate pour vérifier la présence de mots y compris lorsqu'une ou plusieurs lettres ne sont pas fixées. La méthode de création de cet automate sera présentée plus loin.

2. Présentation de l'application

La figure suivante montre l'interface de l'application selon le système d'exploitation:

- à gauche: celle sur macOS où on peut voir les formes possibles et la définition associée à une forme sélectionnée;
- à droite: deux images de l'interface iOS; la première montre l'interface de recherche initiale et la deuxième l'affichage de la définition de la forme sélectionnée (ici *SAISIR*) qui remplace la liste des formes. La flèche à gauche `<` permet de revenir à la liste des formes.





Les deux applications partagent les éléments d'interface suivants:

- Type de recherche: mode *Mots croisés*, *Anagrammes* et *Exp.[ression] régulière*;
- Un champ de texte dans lequel l'usager entre le modèle des formes à chercher; en mode *Mots croisés* ou *Anagrammes*, tout caractère non alphabétique est remplacé par un point; c'est surtout utile sur iPhone où le clavier n'affiche pas le point, il suffit de taper une espace pour obtenir un point. Les recherches se font sur la forme majuscule et non accentuée des mots. Ainsi, les noms *cote*, *côte* et *côté*, la forme conjuguée *cote* du verbe *coter,* ainsi que l'adjectif *coté* sont tous regroupés sous la forme *COTE*. Ce choix correspond à la convention habituelle dans les mots croisés français.
- Une liste de formes qui correspondent au modèle, le nombre de formes est affiché à droite du champ de texte.
- Lorsqu'une forme est choisie dans la version macOS, ses définitions (il peut y en avoir plusieurs) et ses synonymes sont affichés dans la partie droite de l'écran. Dans la version iOS, la définition est affichée dans une autre fenêtre (image complètement à droite) qui remplace la précédente. Ces comportements par `NavigationLink` sont ceux suggérés par Apple et adoptés par la majorité des applications macOS ou iOS.

  La définition est terminée par des liens qui permettent de lancer une recherche de la forme sur Google, dans le Wiktionnaire ou avec Antidote: en macOS ce sera sur le dictionnaire d'Antidote Web, en iOS, ce sera sur l'application Antidote iOS, si elle est présente sur l'appareil.

  En *double-cliquant* un mot dans une définition, la définition de ce mot est affichée. Ceci est très utile pour obtenir la définition d'un mot qui n'est indiqué que comme forme fléchie. Par exemple, la définition de *SAMIERS* est *Pluriel de samier* ; en double-cliquant *samier*, on apprend alors que c'est *une coquille du Sénégal*.
- La partie inférieure de l'écran est composée de filtres qui s'appliquent sur les caractéristiques des formes:
  - la partie du discours: *Nom*, *Verbe*, *Adjectif*, *Autre*;
  - le genre: *Masculin*, *Féminin*;

- le nombre: *Singulier*, *Pluriel*;
- le fait que cette forme est un *Lemme*, et non une forme fléchie, un *Nom propre* ou encore une forme *Jouable* en Scrabble: un jeu de lettres populaire dont le nom est déposé et pour lequel l'éditeur du dictionnaire de référence (*Officiel du Scrabble*) est très susceptible, c'est pourquoi « Scrabble » n'apparaît pas dans l'interface.
- Le contenu du champ de texte sous les filtres limite les formes à celles dont la définition ou les synonymes contiennent cette chaîne.

Sur macOS, l'activation d'un filtre se fait en cochant la case correspondante alors que sur IOS ce sont des boutons qu'on *active*. Aussitôt qu'un filtre est en fonction, la liste des formes possibles est limitée aux formes dont au moins une des acceptions respecte l'ensemble des contraintes. Tous les filtres sont réinitialisés lors d'une nouvelle recherche. Techniquement, ces filtres sont des `Toggle` de SwiftUI qui adapte la présentation à la plateforme.

3. Récupération des données

3.1. Extraction des formes

Wiktionnaire est un dictionnaire multilingue qui vise à décrire tous les mots de toutes les langues. Il en accès libre et sans contrainte d'utilisation à condition d'en identifier la source. En plus d'un accès interactif par le web, il est possible d'obtenir une *dump*, un fichier XML de plus de 7 gigabytes

À noter que nous nous limitons aux pages en français en ignorant les mots de moins de deux lettres ou comportant des caractères non alphabétiques, sauf des `'` et `-` à l'intérieur. En particulier, toutes les expressions ou mots composés contenant une ou plusieurs espaces (p.ex. *pomme de terre*) ne sont pas conservés. Les verbes réflexifs débutant par une voyelle sont conservés sans le `s'` initial (p.ex. on garde `ECLIPSER`, mais non `SECLIPSER`).

L'essentiel du fichier XML est composé d'une suite d'éléments `page` dont un court extrait de la structure pour l'entrée *saisir* est illustrée dans le prochain extrait de code.

```
<page>
    <title>saisir</title>
    <ns>0</ns>
    <id>175758</id>
    <revision>
      ...
      <text bytes="21968" sha1="3tgw14oxh7mzxmbzll88ec1ipoptzpb" xml:space="preserve">
== {{langue|fr}} ==
=== {{S|étymologie}} ===
...
=== {{S|verbe|fr}} ===
'''saisir''' {{pron|sɛ.zi‍ʁ|fr}} ou {{pron|se.zi‍ʁ|fr}} {{t|fr}} {{conjugaison|fr|grp=2}} {{lien pronominal}}
# Prendre [[vivement]], [[rapidement]], [[délibérément]], avec [[vigueur]].
#* {{exemple | lang=fr
 | Elle '''saisit''' le feuillage d'une façon très-singulière, ...
 | source={{w|Étienne Geoffroy Saint-Hilaire}}, ''[[s:Quelques ...
...
# [[prendre|Prendre]] une chose de manière à pouvoir la [[tenir]], la [[porter]], à ne pas la [[laisser]] [[échappe
...
==== {{S|synonymes}} ====
* [[emparer]], [[empoigner]], [[happer]]

==== {{S|antonymes}} ====
...
==== {{S|dérivés}} ====
...
    </text>
    </revision>
</page>
```

J'ai conçu un analyseur XML *en mode événementiel (SAX)* pour extraire les données de chaque balise `page`, sans stocker l'intégralité du fichier dans la mémoire. On ne traite que les pages dont l'élément `ns` indique 0 correspondant à une entrée publiée et qui ont une section débutant par `{{langue|fr}}`.

Dans une balise `page`, c'est la balise `text` qui m'intéresse, mais son contenu utilise un autre format: le *wikicode*. Des expressions régulières Swift ont été développées pour n'y traiter que les parties indiquant les parties du discours, les définitions et les synonymes. D'autres expressions *nettoient* le *wikicode* des définitions, notamment en extrayant les mots entourés de doubles crochets qui correspondent à des liens et en interprétant certains cas simples de HTML, par exemple les indices et les exposants.

Des expressions régulières extraient aussi des informations morphologiques associées à chaque forme encodées avec une chaîne de onze caractères. La table suivante en indique la valeur et la position lorsqu'une des acceptions des mots pour cette forme possède cette caractéristique; c'est une espace sinon. Comme le *wikicode* est par les contributeurs du Wiktionnaire, il peut y avoir des déviations par rapport au formalisme *officiel*, les expressions régulières doivent donc être *souples*.

J'explique à la prochaine section le calcul de l'indicateur `J` qui indique si cette forme est acceptable au Scrabble.

Lettre	Caractéristique	Position
N	Nom propre	0
n	nom commun	1
V	Verbe	2
A	Adjectif	3
-	Autre	4
m	masculin	5
f	féminin	6
s	singulier	7
p	pluriel	8
L	Lemme (c.-à-d. forme de base)	9
J	Jouable au Scrabble, il figure dans l'Officiel Du Scrabble 9	10

En janvier 2025, le parcours et l'analyse en Swift de ce fichier (7 GB) avec plus de 7 millions de pages, dont 1,8 million contiennent du français, nécessite environ une heure et demie sur un ordinateur portatif MacPro. Ce temps inclut la création de la base de données SQLite et de la liste des formes.

3.2. Sélection des mots valables au Scrabble

Cette liste n'est pas *officielle*, quoique très utilisable, car l'indicateur `J` n'est ajouté qu'aux formes déjà présentes dans MCWiktionnaire. Elle a été créée en plusieurs étapes avec des programmes de conversion ad hoc à partir des sources suivantes:

- Liste *officielle* des mots acceptables de 2 à 4 lettres au Scrabble en format texte:
- Liste officieuse grâce au site web Motscroisés.fr qui utilise une application web pour vérifier des mots valides au Scrabble, pour lequel il était possible de récupérer les listes de *n* lettres en format JSON avec la commande suivante: `https://scrabble.fr.wordsdb.ws/letters/wordlength/{n}/type/contain/offset/0/limit/100000`
- En 2024, j'ai mis ces listes à jour pour mieux refléter l'Officiel du Scrabble (ODS9) - 2024 en utilisant cette *liste complète des modifications apportées par l'ODS9*.

3.3. Ajout des symboles chimiques

Étonnamment, les symboles chimiques ne figurent pas tous dans le Wiktionnaire dans une section associée au français. Or ils apparaissent souvent dans les mots croisés. J'ai décidé de les ajouter à partir d'une liste tirée de Wikipédia pour lesquels, une définition standard a été concoctée.

3.4. Création des fichiers de données

Le résultat de ces extractions et opérations est un fichier JSON dont une partie est présentée ici. À chaque forme est associée une chaîne avec des indicateurs dans l'ordre indiqué dans le tableau précédent.

Par exemple, *SAISIES* peut être un nom féminin pluriel, un adjectif féminin pluriel, une forme d'un verbe et elle est *jouable* au Scrabble, alors que *SAISIR* est un lemme, un verbe jouable au Scrabble .

```
 "SAISIES":" nVA  f p J",
 "SAISIESARRETS":" n    f p  ",
 "SAISIESATTRIBUTIONS":" n    f p  ",
 "SAISIESBRANDONS":" n    f p  ",
 "SAISIESEXECUTIONS":" n    f p  ",
 "SAISIESGAGERIES":" n    f p  ",
 "SAISIESREVENDICATIONS":" n    f p  ",
 "SAISIMES":"  V     p J",
 "SAISINE":" n    fs LJ",
 "SAISINES":" n    f p J",
 "SAISIR":"  V      LJ",
 "SAISIRA":"  V    s  J",
```

Les définitions sont conservées dans un fichier SQLite contenant une structure JSON pour chaque forme. Ces structures JSON sont créées et parsées automatiquement par le compilateur Swift, car elles répondent au protocole Codable.

Voici la définition associée à *SAISIR* dont l'affichage apparait dans les figures de la section de présentation.

```
{"definitions": [{
    "mot": "saisir",
    "categories": ["verbe"],
    "defs": [
        "Prendre vivement, rapidement, délibérément, avec vigueur.",
        "Prendre une chose de manière à pouvoir la tenir, la porter, à ne pas la laisser échapper.",
        "Inscrire de façon à avoir une trace tangible, comptabiliser, noter.",
        "Discerner, concevoir nettement, comprendre.",
        "Représenter, croquer avec exactitude, esquisser avec précision.",
        "Brûler modérément la partie extérieure d'un solide.",
        "S'emparer vivement et fortement d'une personne, en parlant des maux du corps, des maladies, des passions, c
        "Confisquer, arrêter, retenir par voie de saisie.",
        "Utiliser un matériel adéquat, en particulier des saisines, pour immobiliser ou au contraire transporter un
        "' saisir de' ",
        "Mettre en possession de."
    ],
    "syns": [{
          "sens": "",
          "syns": ["emparer","empoigner","happer"]}]
  }]
}
```

Une transformation XML avait aussi été développée pour créer un format XML compatible avec l'application *Dictionnaire* de macOS. Cette transformation n'est plus entretenue depuis que la base de données SQLite a été implantée.

4. Algorithmes de recherche

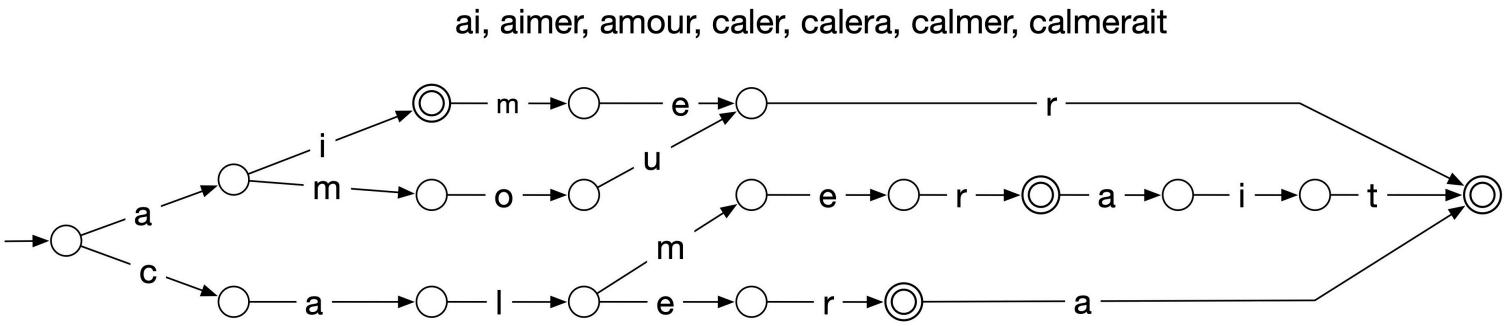
4.1. Automate des formes

Afin d'accélérer l'accès aux formes à partir de la spécification de quelques lettres à certaines positions, j'ai implanté une structure d'automate minimal pour le langage correspondant à l'ensemble des formes acceptables. La méthode de création de l'automate est celle de Maurel et Guenthner (2005). Dans la littérature, ce type d'automate est un *Directed Acyclic Word-Graph (DAWG)* décrit initialement par Blumer *et al.* (1983), utilisé par Appel et Jacobson (1988) dans leur programme pour jouer au Scrabble.

L'algorithme construit un trie pour compresser les préfixes communs des mots, mais aussi un autre pour les terminaisons communes, qui sont évidemment nombreuses dans un dictionnaire français comprenant toutes les formes fléchies.

La figure suivante montre la structure de l'automate à 17 états créée pour une liste de sept mots. Elle illustre le regroupement des préfixes et suffixes des mots. L'état initial marqué par la flèche entrante à gauche de la figure. Les mots sont obtenus en accumulant les lettres sur les arcs de tous les états depuis l'état initial. Les états *acceptants* qui correspondent à des formes figurant dans la liste sont marqués par un cercle double.

Les formes ainsi identifiées servent de clés pour accéder aux définitions dans la base SQLite.



Cette structure compresse très efficacement le dictionnaire: la dernière version du dictionnaire qui comprend plus de 1,6 million de formes, n'a qu'un peu plus de 225 000 états, dont la représentation JSON (non optimisée) utilise moins de 8 megs sur disque. Il faut environ une demi-heure sur un MacPro pour créer cet automate à partir de la liste des formes.

Cette structure d'automate permet une génération rapide de toutes les formes correspondant à certaines lettres à des positions précises. En mode *Mots croisés*, toutes les formes acceptables sont affichées immédiatement après chaque caractère entré par l'usager.

4.2. Recherche des anagrammes

La recherche d'anagramme est un parcours séquentiel des formes. On vérifie d'abord que la forme contient le même nombre de caractère que le motif. Ensuite, on trie les lettres du modèle et on tient compte de deux cas de figure:

- Toutes les lettres sont spécifiées: on compare les lettres du modèle avec les lettres triées de la forme;
- Il y a des points dans le motif: en ignorant les points, on vérifie que les lettres du modèle figurent parmi les lettres triées de la forme. Ce test est très rapide une fois triés les caractères des deux formes.

L'affichage des formes d'anagrammes un peu plus complexe, car il faut identifier les positions des points pour les colorer.

J'avais implanté une table qui listait toutes les anagrammes indexée par une clé formée des lettres triées. Ceci impliquait un fichier supplémentaire. C'était instantané pour les anagrammes complètes, mais le temps pour la recherche avec des motifs avec des points pouvait varier beaucoup en fonction de leur nombre. Comme la recherche de toutes les anagrammes est de l'ordre d'une seconde ou deux, c'est *plus qu'acceptable* pour le joueur de Scrabble moyen...

4.3. Recherche par expression régulière

Cette recherche est un parcours séquentiel de toutes les formes pour ne garder que celles qui concordent *entièrement* avec l'expression régulière spécifiée. Le seul défi en SwiftUI était de trouver une façon d'afficher un message d'erreur en cas d'expression régulière non légale. L'appariement entre ce modèle et toutes les formes ne prend que quelques secondes. Étant donné la nature imprévisible des patrons, les lettres ne sont pas colorées dans l'affichage, comme pour les recherches en type *Mots croisés* ou *Anagrammes*.

Ce type de recherche permet de trouver des formes avec des caractéristiques particulières: par exemple `[AEIOUY]+` pour les formes ne comportant que des voyelles.

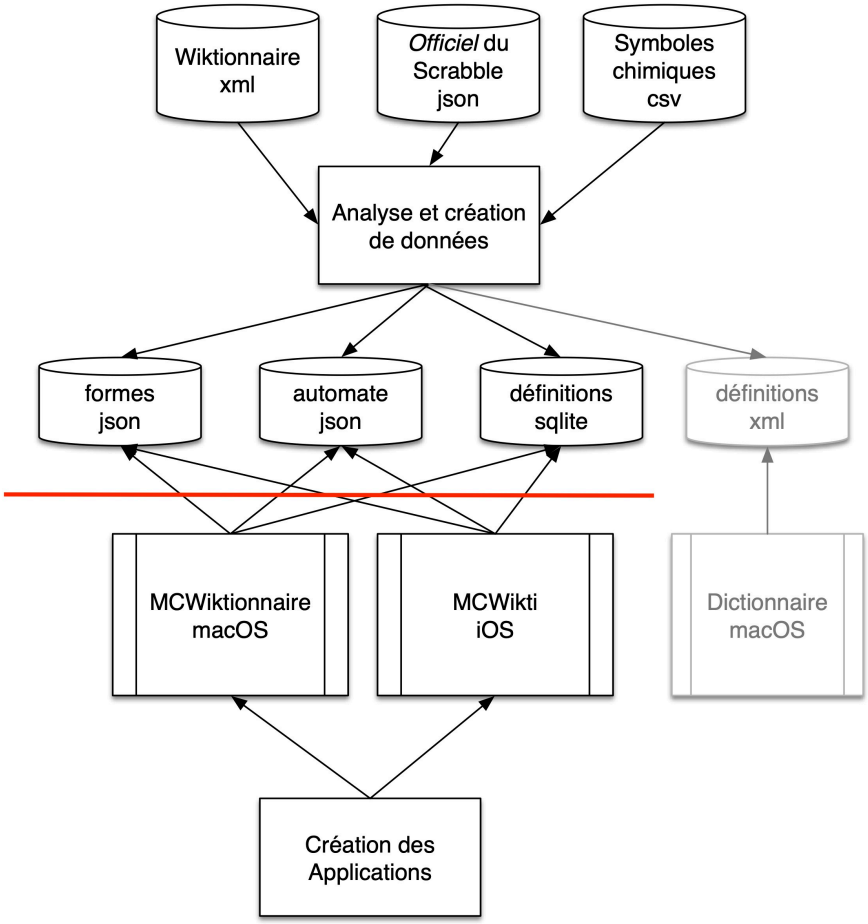
4.4. Filtres

La liste de toutes les formes identifiées par le motif original de recherche est conservée. L'affichage peut ensuite être limitée par l'application des filtres sur les caractéristiques, par exemple les parties du discours. On crée alors une expression régulière composée de 11 caractères avec la lettre correspondant à un filtre activé et un point sinon. La forme reste affichée si la chaîne de ses indicateurs concorde avec cette expression régulière.

Lorsqu'une chaîne est spécifiée dans la boite de texte dans la zone des filtres, la présence de cette sous-chaine dans la définition associée est vérifiée. Cette recherche ne tient pas compte de la casse ou des accents.

5. Création des applications

La partie supérieure de la figure suivante (au-dessus de la ligne rouge) schématise les étapes de récupération et d'organisation des données présentées aux sections précédentes pour créer un fichier de formes avec leurs indicateurs morphologiques, l'automate de recherche de formes ainsi que la base de données SQLite qui conserve les définitions. Le fichier de définitions en XML (indiqué en grisé) pour le dictionnaire macOS n'est plus généré depuis que les définitions sont affichées directement dans l'application. À noter que cette partie est exécutée une seule fois en amont des applications qui utilisent ces fichiers générés.



Les applications macOS et iOS, écrites en SwiftUI, partagent la grande majorité de leur code, car l'adaptation à chaque système et appareil (Mac, iPad ou iPhone) est faite *automatiquement* par le compilateur en spécifiant l'architecture cible.

L'application est organisée suivant les principes suggérés en SwiftUI par un empilage vertical (`VStack`) de vues (voir la figure ci-dessous) qui se mettent à jour en fonction du changement de certaines variables correspondant à l'état du système:

`RechercheView`: trois boutons mutuellement exclusifs (`Picker` en SwiftUI) qui indiquent le type de recherche à effectuer;

`FormeView`: champ de texte dans lequel l'usager tape sa requête, un point indique une position non spécifiée.

`ResultatsView` : les formes qui correspondent à la requête. Les lettres spécifiées apparaissent en noir alors que les lettres *libres* sont affichées en bleu.

`DefinitionView`: la définition, lorsque l'usager appuie sur un des résultats. En iOS sur iPhone, cette vue remplace la `ResultatsView`, alors qu'en macOS ou sur iPad cette vue est affichée à côté de la précédente.

`FiltreView` : ensemble boutons qui lorsqu'ils sont activés, limitent les formes dans `ResultatView` à ceux dont les indicateurs morphologiques correspondent à tous les boutons activés. Il y a également un champ de texte qui filtre les définitions associées aux formes; lorsque l'usager saisit une chaîne dans ce champ, seules les formes dont la définition comprend ce texte comme sous-chaîne restent affichés.

La définition est affichée avec des NSAttributedString pour créer des titres en gras, des listes à puces ainsi que les exposants et les indices à partir des balises `<sup>` et `<sub>`. On ajoute également des liens vers des ressources externes (e.g. Google, Wiktionnaire et Antidote). Lorsqu'un filtre de définition est spécifié, la partie qui correspond au texte du filtre est surlignée en jaune dans la définition.

5.1. Quelques particularités

5.1.1. Support des *vieux* iPhones

L'application iPhone supporte la version iOS 15 pour qu'elle reste disponible sur mon iPhone 6s, étant donné que je suis le principal, et probablement le seul, utilisateur de l'application... Ce choix implique que certaines *facilités* introduites dans iOS 16 ne sont pas disponibles, notamment les expressions régulières natives de Swift. Il a donc fallu continuer à utiliser NSRegularExpression, un formalisme un peu plus laborieux. L'analyse de la dump Wiktionnaire, effectuée en amont, par un programme distinct profite toutefois des expressions régulières *natives* de Swift.

Il en est de même pour l'affichage des définitions qui doit utiliser des NSAttributedString et non les AttributedString de Swift qui n'implantent pas encore toutes les possibilités des précédentes.

5.1.2. Affichage progressif des définitions

L'affichage des définitions associées aux formes est relativement complexe. Ii a fallu développer un chargement des résultats à la demande pour contourner le fait que lorsqu'il y a plusieurs milliers de résultats, Swift les calcule tous (y compris les définitions associées), même si seulement une faible partie est affichée. Le calcul de toutes les formes ne pose lui aucun enjeu.

Ce comportement ralentissait alors tout le système. Il faut dire que dans la majorité des cas *pratiques* pour les mots croisés, ceci n'arrive pas trop souvent, mais c'est tout de même assez embêtant, car l'affichage des résultats est fait au fur et à mesure de la frappe de l'usager.

L'algorithme garde l'indice du dernier résultat affiché et lorsqu'il apparait sur l'écran, on ne prépare que le prochain bloc de résultats qui pourra être affiché lors du déroulement. Tout fonctionne bien avec iOS, mais avec macOS le `.onAppear` a un comportement légèrement différent. C'est pourquoi on vérifie plutôt si le dernier affiché est parmi les derniers `delta` affichés plutôt que seulement le dernier.

Ceci a permis de simplifier le code en évitant d'utiliser une *background queue* qui, de toute façon n'améliorait pas beaucoup l'utilisabilité de l'interface. La documentation d'Apple suggère l'utilisation d'un `LazyVStack` pour résoudre ce problème. Je suis arrivé à faire fonctionner cette structure sous iOS, mais son comportement était erratique sous macOS, alors j'ai gardé la solution précédente.

6. Conclusion

Ce document a présenté les principes d'organisation de l'application de mots croisés que je développe depuis plus de dix ans maintenant. Il décrit l'état du système en 2025. Il passe sous silence certains essais infructueux et des *fonctionnalités* qui avaient été implantées lors des versions précédentes, mais qui ont été abandonnées, car elles ne se sont pas révélées très utiles; par exemple, la possibilité de récupérer les anciens modèle de recherche.

Cet exercice a été très intéressant et instructif, il y a en effet souvent loin de la théorie à la pratique. La publication d'une application implique plusieurs détails plus ou moins bien documentés: accès aux ressources intégrées à l'application, création des icônes de différentes grandeurs et résolution, utilisation du *debugger* sur le simulateur des appareils et la connexion à un iPhone *physique*, organisation des fichiers d'aide, gestion des certificats de développement, etc. À un certain moment, j'ai même dû discuter fortement avec Apple pour ne pas avoir à fournir mes coordonnées bancaires, car mon application est gratuite, *free as in free beer*, comme on disait dans les années 80.

6.1. Bibliographie

Appel, A. W. Appel, Jacobson, G. J. 1988. *The world's fastest Scrabble program*. Commun. ACM 31, 5 (May 1988), 572–578. https://doi.org/10.1145/42411.42420].

Blumer. A.. Blumer, J., Ehrenfeucht, A.. Haussler, D.. and McConnell, R. *Linear size finite automata for the set of all subwords of a word: an outline of results*. Theoretical. Camputer. Science. 21 (1983). 12-20.

Maurel, D., Guenthner, F., *Automata and Dictionaries*, Vol 6, Text in Computing, College Press, 2005.

6.2. Statistiques des versions

Le tableau suivant montre l'augmentation du nombre de formes disponibles dans l'application. Le passage à la version 2.0 a été un peu *compliqué*, ce qui explique que plusieurs versions (2.0 à 2.2) au début de 2022 avaient gardé la même source du Wiktionnaire.

Version	Date	Nb Formes	Nb États
2.8	Janv 2025	1 601 740	225 956
2.7	oct 2024	1 597 587	225 048
2.6	juin 2024	1 593 963	224 024
2.5	fév 2024	1 587 621	222 179
2.4	sept 2023	1 565 092	217 971
2.3	dec 2022	1 537 235	213 399
2.2	mars 2022	1 516 065	208 454
2.1	fév 2022	1 516 065	208 454
2.0	jan 2022	1 516 065	208 454
1.8	avr 2021	1 497 131	205 717
1.7	jan 2020	1 400 372	198 476
1.0	août 2019	1 387 089	196 896

Quelques travaux *antérieurs*:

- En 2007, j'ai implanté la méthode décrite par Maurel et Gunthner (2005) en *Ruby* pour la création d'un automate de mots avec la recherche avec des motifs. Quelques années plus tard, je l'ai ensuite portée d'abord en *python* et ensuite en Swift.

- En 2016, j'avais développé une première application macOS, *MCDico*, en utilisant comme source de données le GLAWI, une version structurée et normalisée en XML du Wiktionnaire. Comme cette ressource n'était plus mise à jour, j'ai décidé de créer mon propre analyseur de dump Wiktionnaire pour *MCDico*. Cette version n'a jamais été publiée sur le *AppStore*, je ne l'utilisais que sur mon ordinateur.
- En 2018, j'ai créé une interface web de recherche, toujours accessible, de cette ressource: MCDicoWeb. Toutefois, elle n'utilise pas la dernière version du Wiktionnaire et elle n'affiche pas les définitions des mots.