Université Paris Cité

# Compléments en Programmation Orientée Objet
# Projet : Analyseur de texte et optimisateur de disposition clavier

*Projet à réaliser en binôme, à rendre sur Moodle, avant la date indiquée sur Moodle.*
*Le langage doit être Java (version 21).*
*Le code doit être écrit par vous, mais votre programme peut dépendre de bibliothèques tierces (indiquez clairement lesquelles).*

Ce projet consiste en deux (ou trois) logiciels partageant certaines structures de données :

1. un analyseur de fréquence de suites de caractères dans un corpus de textes donné
2. un évaluateur de disposition de clavier
3. (s'il vous reste du temps) un optimiseur de disposition de clavier

## 1 Contexte

Dans ce sujet, on parle de dispositions *fonctionnelles* de clavier, à savoir l'association de touches physiques à des fonctionnalités (notamment des caractères). Des discussions existent aussi à propos des dispositions géométriques (placement physique des touches), mais c'est hors-sujet ici.
Les raisons qui font que les dispositions de clavier classiques (QWERTY, AZERTY, QWERTZ, ...) sont telles qu'elles sont, sont variées [1], mais la plupart d'entre elles ne sont plus d'actualité aujourd'hui (faciliter le travail des télégraphistes, empêcher les tiges des machines à écrire de se gêner en s'entrecroisant, ...).
Or, dès le début du XXe siècle [2], on a compris qu'on pouvait proposer des dispositions plus ergonomiques en se basant sur des études statistiques [3]. Cette idée connait récemmment un regain de popularité [4] ; cela s'explique sans doute par les capacités de calcul désormais à disposition de tout le monde ainsi que par la démocratisation d'internet, qui a facilité l'échange de nouvelles idées au sein de communautés d'enthousiastes. Aujourd'hui, il ne se passe pas un mois sans l'annonce d'une nouvelle proposition. [5]
Les concepteurs de nouvelles dispositions procèdent généralement étape par étape, introduisant une nouvelle modification, puis en évaluant le résultat selon des critères objectifs chiffrés, pour valider ou invalider le changement, puis bis repetita.
Ainsi, il est crucial de pouvoir exécuter automatiquement et efficacement l'évaluation d'une disposition. C'est ce que le sujer vous propose d'implémenter.

[1] `https://repository.kulib.kyoto-u.ac.jp/dspace/bitstream/2433/139379/1/42_161.pdf`, lecture très intéressante !

[2] Notoirement disposition Dvorak, en 1936, pour l'Anglais. Mais d'autres propositions ont été faites pour d'autres langues. On peut notamment évoquer le F-keyboard turc, qui a été, depuis 1955, le clavier turc officiel, avant d'être supplanté par une variante de QWERTY.

[3] Mais le poids de quelques décennies d'habitudes avaient déjà rendu le changement impossible.

[4] Difficile de dire qui a commencé. La disposition Bépo, pour le français, aurait été essentiellement finie en 2005 (mais Dvorak-fr avait été proposée en 2002, sans toutefois obtenir une grande notoriété). Colemak, pour l'Anglais a sorti sa première version en 2006. Celle-ci a eu un grand impact pour la suite, notamment sa variante Colemak-DH, qui semble être aujourd'hui la disposition ergonomique la plus populaire.

[5] dispositions récentes populaires, pour l'Anglais : Canary, Graphite, Gallium, Sturdy, ... ; pour le Français : Optimot, Ergo-L, Erglace, ...


FIG. 1 – Affectation des touches de clavier aux différents doigts (par couleur).

## 2 Vue d'ensemble de l'évaluation

Une disposition est évaluée sur des statistiques, par rapport à un usage donné et à une fonction d'évaluation donnée. Ledit usage est matérialisé par la donnée d'un corpus, c'est-à-dire un ensemble de textes censé correspondre au genre de texte qu'on imagine typiquement tapés par l'utilisateur final du clavier. Si l'utilisateur a pour métier d'écrire de la documentation technique en Anglais, le meilleur corpus, c'est un ensemble de documents techniques en Anglais. Si c'est un écrivain francophone, le corpus devrait consister en de la littérature en Français. Si c'est un progammeur, le corpus devra comporter du code source.
Une fois le corpus choisi, on analyse les mouvements que devraient nécessiter l'écriture de ce corpus si celle-ci se faisait sur un clavier utilisant la disposition à évaluer.
Un mouvement est caractérisé par une propriété géométrique donnée sur une petite séquence de touches (en pratique : 1, 2 ou 3 touches). Il s'agit donc de compter les différents types de mouvements qui nous intéressent pour le corpus choisi.
La fonction d'évaluation consiste en la donnée de différents poids aux scores des différents mouvements.
L'évaluation se fait ainsi :

1. On fait le pré-traitement du corpus en comptant les occurrences de tous les $N$-grammes. Ce calcul n'a pas à être refait tant qu'on garde le même corpus.
2. On traduit les $N$-grammes en séquences de touches, en comptant les occurrences de chacun des mouvements.
3. On calcule le score final en faisant la somme pondérée des quantités calculées à l'étape précédente, normalisée par la taille du corpus.

Remarque : un $N$-gramme est une séquence de $N$ caractères consécutifs. En pratique, on ne considère que $N = 1, 2$ (bigramme) ou 3 (trigramme). [6]

## 3 Mouvements à considérer

La pertinence des mouvements définis ci-dessous prend son sens seulement si on part du principe qu'on utilise la technique dactylographique classique (permettant de taper efficacement à l'aveugle). Aini, chaque touche est affectée à un doigt unique (cf. Fig. 3), et chaque doigt a une unique position de repos (une touche sur la rangée alphabétique centrale, à savoir, en AZERTY, les touche Q, S, D, F et J, K, L, M).

[6] C'est le vocabulaire habituellement utilisé dans le domaine. Attention aux définitions incompatibles issues d'autres domaines ou communautés !

**« Mouvements » à une seule touche :** [7]

- Chaque touche prise isolemment est un mouvement. Dans la fonction d'évaluation, on peut lui associer un poids correspondant à sa difficulté d'accès afin de mesurer l'effort. Cependant, ce critère a été rendu plus ou moins obsolète par l'étude des bigrammes et trigrammes (la difficulté d'une touche n'étant pas intrinsèque).
- On peut compter comme un mouvement le fait de presser n'importe quelle touche affectée à une même main, afin de vérifier l'équilibre de l'effort. La valeur numérique à prendre en compte, et à minimiser, est la distance à l'équilibre absolu (50%–50%). Attention : ce qui est intégré dans la somme pondérée ce n'est pas directement les scores des deux mains, mais une distance (la valeur absolue de la différence entre les mains).
- Même chose pour la répartiton entre les doigts. Là encore, on injecte une distance à la somme pondérée (distance [8] à une répartition idéale à définir, qui prendra en compte que certains doigts, plus forts, doivent avoir une charge plus importante).

**Mouvements de longueur 2 :** [9]

Mauvais mouvements à minimiser :

- les mouvements à un seul doigt (SFB [10])
- les ciseaux, c'est-à-dire mouvements à une main faisant succéder la rangée inférieure et la rangée supérieure (dans un sens ou dans l'autre)
- les mouvements à extension latérale (LSB [11]), c'est-à-dire dont l'une des touches est sur une autre colonne que la position de repos du doigt concerné (p. ex. les touches T,G,B, Y,H et N pour l'index en Azerty).

À maximiser, le reste :

- les alternances main gauche/main droite,
- les roulements (un roulement est un mouvement réalisé par deux doigts différents d'une même main), on préfère notamment les roulements de l'extérieur vers l'intérieur (de l'auriculaire vers l'index). On note que les roulements à maximiser sont ceux qui n'appartiennent pas aussi aux mauvaises catégories (SFB, LSB ou ciseau).

Remarque : une fois qu'on a minimisé les SFB, les LSB et les ciseaux, les préférences varient entre favoriser les roulements au détriment des alternances (Colemak, Ergo-L) ou bien le contraire (Dvorak, Bépo). Cela correspond à deux philosophies différentes, qui conviennent plus ou moins en fonction des personnes.

**Mouvements de longueur 3 :** [12]

Mauvais enchainements à minimiser :

- les redirections, c'est-à-dire les mouvements à une main avec changement de direction : de l'intérieur vers l'extérieur, puis de l'extérieur vers l'intérieur, ou alors le contraire.
- en particulier : les mauvaises redirections, ne faisant pas intervenir l'index, considérées le pire enchainement possible.
- les « skipgrammes » sur le même doigt (SKS [13]), c.-à-d., comme un SFB, mais avec une touche de l'autre main entre les deux.

[7] Correspondant habituellement à l'écriture d'un caractère unique.

[8] On parle de la distance entre deux vecteurs : la répartition idéale et la répartition effective. On peut prendre la distance suivante : la somme des valeurs absolues des différences de chaque coordonnée.

[9] Correspondant la plupart du temps à un bigramme.

[10] same finger bigram

[11] lateral stretch bigram

[12] Correspondant la plupart du temps à un trigramme.

[13] same finger skipgram

# 4 Décompte des mouvements

Le pré-traitement consiste à compter les occurrences de chaque caractère (1-grammes), chaque bigramme et chaque trigramme du corpus et à stocker ce décompte, indépendant de la disposition, afin d'évaluer ensuite plusieurs dispositions.
Cependant, les mouvements sont en réalité définis en termes de séquences de touches, et non en terme de $N$-grammes. On confond souvent les deux, car c'est souvent la même chose [14], mais, parfois, il y a des différences :

- quand des caractères sont obtenus par l'appui préalable d'une touche morte (p. ex. : « û », obtenu par `[^, U]`)
- quand certains caractères sont obtenus par le maintien d'une autre touche (modificateur), exemple : en AZERTY le « . » s'obtient par `[Shift+;]`.

Pour évaluer une disposition, il faut donc convertir chaque $N$-gramme en séquence de caractères et regarder si la séquence obtenue est un mouvement qu'il faut décompter (auquel cas on ajoute au score de ce mouvement le score associé au $N$-gramme). Par exemple, le caractère « î » devient le bigramme `[^, I]`, qui est un LSB en AZERTY, donc on ajoure le score de « î » à celui des LSB. Le bigramme « îl » devient la séquence `[^, I, L]`, qui est une mauvaise redirection, on ajoute donc le score de « îl » à celui des mauvaises redirections. En revanche, le trigramme « île » est ignoré, car il deviendrait une séquence de longueur 4 alors qu'on ne considère aucun mouvement de longueur 4.
Ainsi, chaque critère évalué donne un résultat numérique. Une note globale de la disposition peut être calculée par une moyenne pondérée (le choix de la pondération est arbitraire et dépend de l'importance accordée à chacun des critères).

# 5 Comment améliorer une disposition

Du fait de l'explosion combinatoire, il est hors de question d'explorer tout l'espace des possibles (avec des techniques du type *backtracking*).
Différentes techniques sont envisageables, mais je vous suggère d'utiliser un algorithme génétique :

- On maintient un pool de dispositions candidates (toutes déjà évaluées et triées)
- À chaque itération, on crée un nouveau candidat, soit par mutation, soit par croisement.
- On évalue le candidat. Il prend sa place dans le pool s'il est meilleur que certaines des dispositions existantes, auquel cas, on supprime la moins bonne disposition.

**Mutation :** petit changement, qu'on pourrait obtenir en opérant quelques inversions (suggestion : essayez 3 pour commencer).

**Croisement :** on sélectionne deux parents et on les "mélange". Une possibilité :

1. choisir les parents A et B au hasard dans le pool
2. remplir, au hasard, la moitié de la disposition avec les mêmes choix que A ;
3. remplir tout ce qu'on peut du reste avec les mêmes choix que B (sans créer de doublons) ;
4. remplir tout ce qu'on peut du reste avec les mêmes choix que A (sans créer de doublons) ;
5. remplir au hasard avec les caractères qui n'ont pas encore été placés.

# 6 Contraintes techniques

**Configurabilité** Votre code ne contient pas les données, susceptibles d'être modifiées. Votre logiciel doit pouvoir fonctionner pour des configurations différentes, fournies via des fichiers.
Ainsi, concevez des formats de fichiers pour stocker :

[14] ... surtout en Anglais...

- les géométries de clavier (affecter une rangée, une colonne, un doigt, voire des coordonnées millimétriques à chaque touche physique du clavier considéré... en fonction de ce dont vous avez besoin pour détecter les mouvements pris en compte)
- les dispositions de clavier (associations touche/caractère [15] [16])

Utilisez une syntaxe existante, telle que JSON, YAML, ou TOML. N'hésitez pas à importer des bibliothèques existantes pour vous aider.

**Compatibilité** Vos logiciels sont bien entendu compatibles entre eux. Ils partagent des classes et utilisent les mêmes formats de fichiers.
Cela inclut les géométries et dispositions, mais aussi les fichiers contenant des statistiques de fréquence de N-grammes collectées (vous pouvez utiliser un des formats déjà cité, ou bien, plus simplement, le format CSV).
Bonus : permettre une compatibilité plus large en prévoyant l'import/export depuis et vers des formats déjà utilisés pour stocker les dispositions de clavier dans les logiciels existants (format `.klc` de MSKLC pour Windows, format `.ahk` de AutoHotKey, format `.keylayout` de macos, format XKB pour Linux, format `.toml` de Kalamine [17], ...).

**Modélisation objet** Pensez à introduire un type pour chaque concept. Privilégiez autant que possible les objets immuables. Pensez aux `enum`, `record` et autres `sealed class` ou `interface`. Programmez à l'interface !
Privilégiez une architecture favorisant la réutilisation de code, notamment entre les 3 logiciels demandés.

**Concurrence** Certains traitements demandés ici peuvent être parallélisés. Notamment :

- le pré-traitement du corpus (traiter chaque texte du corpus en parallèle et, à la fin de l'analyse d'un texre, ajouter le résultat à un accumulateur commun partagé)
- l'algorithme génétique (on peut en faire tourner plusieurs instances en parallèle, et de temps en temps fusionner leurs pools respectifs pour ne garder que les meilleurs candidats)

Une parallélisation bien faite rapportera des points.

**Testabilité** Tout ce que vous écrirez devra avoir été programmé de telle sorte à ce que le comportement soit testable.
Le découpage en méthodes doit être suffisamment fin pour permettre des tests unitaires pertinents.
Le découpage en classes et la programmation « à l'interface » doivent permettre d'effectuer facilement des tests d'intégration. Cela peut être fait en remplaçant n'importe quel composant par un composant factice implémentant la même interface.
Les tests devront être écrits, exécutables, et fournis avec le rendu.

# 7 Aide technique

- Utilisez un système de compilation tel que Maven ou Gradle. Vous faciliterez la gestion des dépendances.

[15] Commencez par une simple « map ». Mais attention, dans une disposition complète, il y a plusieurs « couches », c'est-à-dire plusieurs tableaux d'associations, correspondant à des contextes différents : mode `Shift`, mode `AltGr`, différents modes correspondant au fait que tell ou telle touche morte a été enfoncée ; n'essayez pas de tout prendre en compte dès le début !

[16] Gardez en tête que pour analyser une disposition, il faut réussir à trouver, pour un caractère donné, quelle touche ou séquence de touches permet de l'obtenir.

[17] `https://github.com/OneDeadKey/kalamine/`

– Les tests peuvent être écrits dans un *framework* tel que JUnit5 [18], mais ce n'est pas indispensable. En tout cas, il faut des tests.

Un exemple minimal sera mis à votre disposition sur Moodle.

# 8 Critères d'évaluation

– Le projet est rendu dans une archive `.zip`.
– Le projet doit contenir un fichier `README.md` indiquant comment compiler, exécuter et utiliser votre programme et ses tests, décrivant les fonctionnalités effectivement implémentées et expliquant vos choix techniques originaux, le cas échéant.
– Joignez aussi un ou des diagrammes de classe.
– La commande pour compiler ou exécuter doit être simple (s'aider de Gradle, Maven... voire de Makefile).
– La compilation selon ces instructions doit terminer sans erreur ni avertissement.
– L'exécution doit être correcte : elle respecte le cahier des charges et ne quitte pas de façon non contrôlée.
Les exceptions doivent être rattrapées. Dans tous les cas, si le problème n'est pas réparable, un message d'erreur doit être présenté à l'utilisateur. Aucune de ces erreurs ne doit être due à un problème de programmation (comme le fameux `NullPointerException`...).
– Le code respecte les conventions de codage.
– Le code est architecturé intelligemment. Notamment, il faut utiliser des patrons de conception vus en cours ainsi que les constructions les plus appropriées fournies par Java.
– Les classes et méthodes sont documentées (javadoc).
– Des commentaires expliquent les portions de codes qui ne sont pas évidentes.
– Les tests fournis doivent être aussi exhaustifs que possible.
– Un projet qui tenterait de tout faire mais dans lequel rien ne marche bien sera moins bien noté qu'une projet qui ne remplit que la première moitié des objectifs sans avoir de bug.
Concentrez-vous sur les premiers objectifs (analyseur statistique de corpus et évaluateur de disposition) avant de passer au dernier (algorithme génétique).

[18] `https://junit.org/junit5/`