Université Paris Cité

Compléments en Programmation Orientée Objet
Projet : Dactylo-game

Projet à réaliser en binôme, à rendre sur Moodle, avant la date indiquée sur Moodle.
Le langage doit être Java (version 17 conseillée, mais si vous utilisez une autre version, signalez le clairement, idéalement dans votre configuration Gradle, sinon au moins dans le `README.md`).
Le code doit être écrit par vous, mais votre programme peut dépendre de bibliothèques tierces (celles indiquées dans le sujet ou des bibliothèques rendant un service équivalent).

Ce projet consiste en un jeu pour apprendre à taper au clavier et s'améliorer en se mesurant à d'autres joueurs.
Il s'agirait d'un hybride entre les logiciels d'entraînement dactylo classiques (comme sur les sites web Monkeytype [1], 10fastfingers [2], Dactylotest [3], KeyBR [4]...) et Tetris, notamment son mode multijoueur.

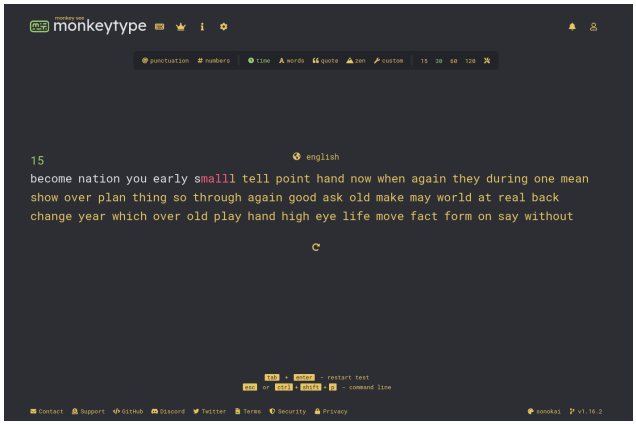

FIGURE 1 – Capture d'écran du site Monkeytype

1 Déroulement du jeu

En solo, mode normal : Le mode de base est similaire aux sites d'entraînement déjà cités : des mots apparaissent à l'écran, que le joueur est censé recopier au clavier sans faire d'erreur. Ces mots peuvent être aléatoires ou bien constituer un texte cohérent (issu de la littérature, d'un discours, d'un article de Wikipedia, ...).
Pendant la session, différentes indications graphiques renseignent le jouer sur sa situation (curseur dans le mot à taper, erreurs signalées en surbrillance dans le texte déjà tapé).
On a toujours une vision sur le futur proche : concrètement, il y a une zone tampon, une file, qui donne les prochains mots à taper ($\simeq$ 15 mots). Le mot à taper tout de suite est situé en tête de la file. Dès qu'un mot est validé (par appui sur la touche Espace), il est supprimé de la tête. La file est alors alimentée par un nouveau mot, ajouté à la fin.
À la fin de la session (délimitée par un temps ou bien par un nombre de mots), les statistiques sont affichées (vitesse en mots par minutes, précision, fluidité).

mot en cours, supprimé après validation par Espace → | lorem | ipsum | dolor | sit | amet | ← nouveau mot ajouté à la fin à chaque validation

En solo, mode jeu : on rajoute des mécaniques de jeu.

1. Pour chaque mot validé, on compte les erreurs (distance d'édition). Une vie est retirée pour chaque erreur. On perd la partie dès qu'on a 0 vies ou moins.
2. On monte d'un niveau tous les 100 mots tapés correctement. Le niveau change la vitesse du jeu (cf. règle d'après).

1. `https://monkeytype.com/`
2. `https://10fastfingers.com/`
3. `http://dactylotest.free.fr/dactylotest/`
4. `https://www.keybr.com/`

3. Les mots à taper arrivent, comme dans Tetris, à une vitesse imposée par le jeu. Ainsi, la file des prochains mots est alimentée par un nouveau mot toutes les $f(n)$ secondes , où f est une fonction décroissante du niveau courant n (prenons $f(n) = 3 * 0,9^n$, par exemple). Si la file était pleine, cela force la validation du mot courant (avec perte de vies si incorrect).
4. Pour permettre à un jouer de respirer après une pointe de vitesse, le fait de valider un mot ne provoque pas forcément l'ajout d'un nouveau mot à la file (contrairement au mode normal). On ne le fait que si la file n'était pas au moins à moitié remplie.
5. Certains mots, signalés en bleu, permettent de récupérer les vies s'ils sont correctement tapés du premier coup (on gagne un nombre de vies égal à la longueur du mot).

validation par Espace ou forcée (file pleine) → | lorem | ipsum | dolor | sit | amet | ← ajouts dûs au chronomètre ou à une file pas assez remplie

Mode jeu multi-joueur : Ce mode se joue nécessairement en réseau, à 2 joueurs ou plus.
Il est similaire au mode jeu solo, mais aucun mot n'est ajouté à la file à cause du chronomètre... car ils peuvent maintenant être envoyés par un adversaire !
En effet, certains mots, signalés en rouge, sont des opportunités pour piéger ses adversaires. Quand un joueur tape correctement du premier coup un mot en rouge, celui-ci est envoyé (sans sa couleur) à la file d'attente de tous les adversaires. Si la file est pleine quand un tel mot arrive, comme en jeu solo, la validation du mot en cours est forcée, occasionnant une perte de vies.
Un joueur qui n'a plus de vies est éliminé. Le dernier joueur survivant gagne la partie.
Remarque : il n'y a plus de notion de niveau. Le niveau est en quelque sorte imposé par l'adversité.

validation par Espace ou forcée (file pleine) → | lorem | ipsum | dolor | sit | amet | ← ajouts dûs au réseau ou à une file pas assez remplie

2 Contraintes techniques

Les statistiques Pour que tout le monde mesure la même chose et pour faciliter le test, nous précisons les définitions des 3 métriques utilisées pour mesurer les performances de frappe au clavier.

- Vitesse (MPM) : le nombre de caractères utiles, divisé par le temps en minute, divisé encore par 5 (ici, on considère par convention qu'un mot fait en moyenne 5 caractères).
- Précision (%) : le nombre de caractères utiles divisé par le nombre d'appuis de touche ($\times 100$).
- Régularité : c'est l'écart-type de la durée entre 2 caractères utiles consécutifs.

Dans les 3 cas, par « caractère utile », nous entendons tout caractère tapé non effacé qui au final appartient à un mot correctement tapé.

Réseau L'architecture sera client-serveur. Le serveur peut être un programme exécutable séparé ou le même programme que le client (un des joueurs hébergeant alors la partie pour les autres joueurs).
N'hésitez pas à discuter avec les étudiants d'autres binômes pour essayer de faire des clients et serveurs interopérables (même protocole). Cette démarche tend à augmenter la qualité des programmes.

Concurrence Il est logique que le logiciel en mode solo puisse s'exécuter entièrement dans le thread applicatif de JavaFX (ou du toolkit graphique choisi), c'est-à-dire sans création explicite de thread supplémentaire. En revanche, en mode multi-joueur, il faudra que le client créée un thread séparé pour gérer la connexion au serveur.
Le serveur, lui, utilisera un thread pool (1 thread par connexion client).
Dans tous les cas, prenez soin de vérifier la bonne synchronisation.

Paramétrabilité À chaque fois qu'un comportement du jeu (vitesse, fréquence des bonus, textes à taper, ...) est paramétrable par nature, il doit être considéré comme un paramètre de l'instance du jeu (on évitera d'écrire de multiples versions de la classe principale, chacune avec ses paramètres fixés dans le code de la classe). Permettez la création d'une instance d'un jeu à partir d'un objet matérialisant un paramétrage donné.
Même si vous n'implémentez pas toutes ces fonctionnalités, ayez à l'esprit que vous pourriez avoir un écran de choix des options et qu'un jeu d'options pourrait être sauvegardé, restauré, utilisé plus tard pour instancier un jeu, voire plusieurs jeux successifs... voire envoyé sur le réseau pour que toutes les instances sur le réseau aient les mêmes réglages.

Testabilité Dans tous les modes, 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 effectuer des tests unitaires.
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.
Notamment, écrivez des composants factices fournissant des séquences d'entrées écrites à l'avance dans un fichier (directement dans le `.java` ou fichier texte lu par le programme) en respectant le chronométrage spécifié dans le fichier (donc c'est une séquence d'évènements avec horodatage). Ces composants pourront remplacer et simuler le contrôleur de l'IG (entrées utilisateur) et le composant réseau (simulation du comportement des autres participants : serveur ou client selon qu'on soit en train de tester le client ou le serveur). Remarquez qu'il est ainsi possible de tester le mode multi-joueur sans avoir encore écrit l'implémentation de la partie réseau.
Les tests devront être écrits, exécutables, et fournis avec le rendu.

3 Autres aspects du jeu

Pour le reste soyez raisonnables. Tout en vous inspirant des sites web donnés en exemple, faites des choix guidés par l'objectif du logiciel. N'en faites pas trop pour l'IG, dès lors qu'elle est claire et pratique. Il est plus important de consacrer du temps pour améliorer la qualité architecturale du code.

4 Aide technique

- Utilisez un système de compilation tel que Maven ou Gradle. Vous faciliterez la gestion des dépendances.
- Interface graphique : JavaFX conseillé [5]. Utilisez le plugin JavaFX pour Maven ou Gradle, vous vous simplifierez la vie.
- Zones de texte « riche » (avec formattage : gras, souligné, couleurs,...) en JavaFX : la bibliothèque RichTextFX [6] devrait faire l'affaire.
- Réseau : vous pouvez utiliser une connexion TCP. Regardez la documentation des classes `java.net.Socket` et `java.net.ServerSocket`. Vous trouverez aussi sur votre moteur de recherche préféré plusieurs cours d'enseignants de l'UPC qui expliquent ces classes.
  Alternative possible à TCP brut : HTTP/WebSocket (via une bibliothèque tierce).
- Les objets passés sur le réseau peuvent être sérialisés en utilisant un format standard tel que JSON. Il est utile pour cela d'utiliser un bibliothèque telle GSON [7]. Alternatives : Protobuf, sérialisation native de Java.

5. `https://openjfx.io/`
6. `https://github.com/FXMisc/RichTextFX`
7. `https://github.com/google/gson`

  - Les tests peuvent être écrits dans un *framework* tel que JUnit5[8], mais ce n'est pas indispensable.

Un exemple minimal utilisant tout cela sera mis à votre disposition sur Moodle.

5 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.
  Notamment, en mode graphique, 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.
- Il y a 3 modes de jeu. Il est possible de faire un projet cohérent sans le mode multi-joueur. Cela rapporte moins de points, mais cela vous laisse plus de temps pour améliorer la qualité de ce qui a été codé.

8. `https://junit.org/junit5/`