SPEEDTYPER

Projet d'INF431

Mars 2018

Vincent DALLARD et Romain FOUILLAND

1 INTRODUCTION

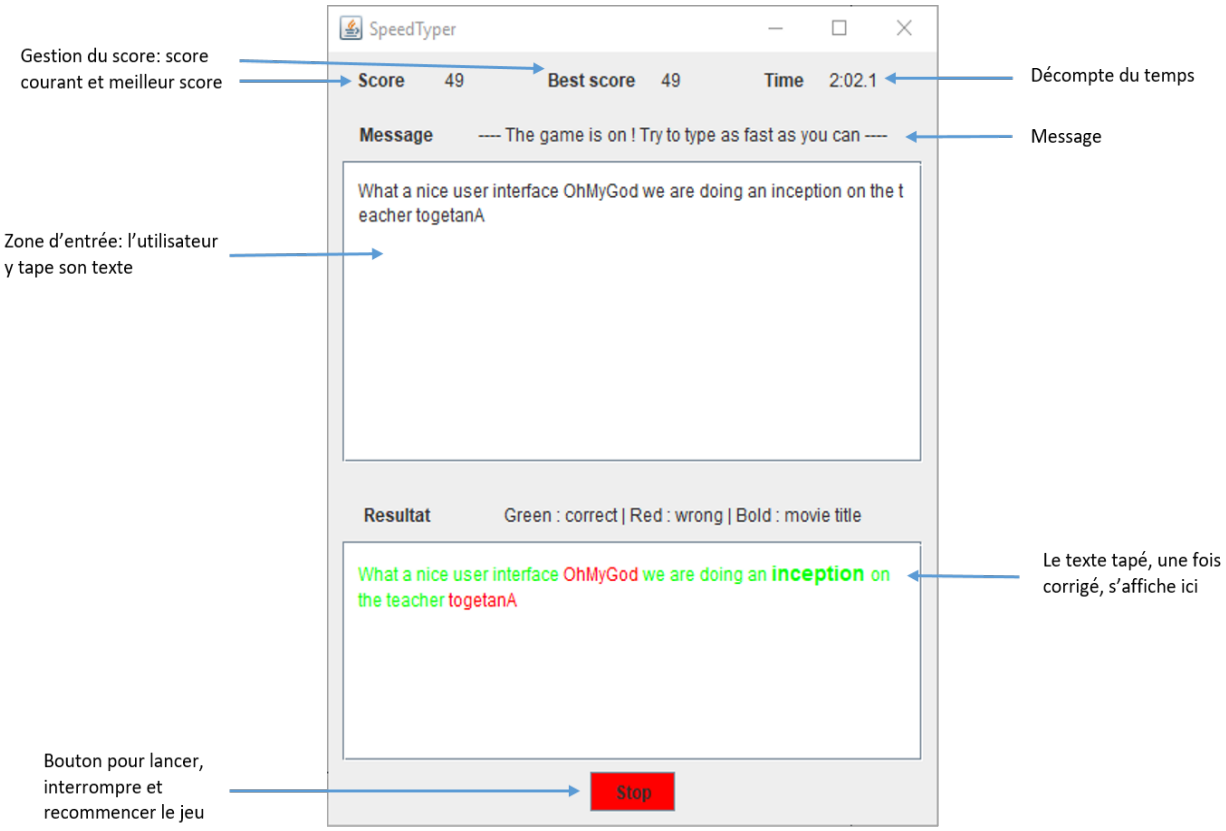
Le but du SpeedTyper est de taper le plus vite possible. Pour que le jeu soit pertinent, il faut donc que l'interface utilisateur (UI) soit la plus fluide possible. Ainsi, il faut gérer la vérification des mots et le décompte du temps en parallèle pour ne pas ralentir l'UI. De ce fait, un programme utilisant plusieurs threads est nécessaire.

Nous avons ainsi développée une première version naïve qui était gérée par l'UI. Afin d'améliorer les performances du jeu, nous avons dans une deuxième version utilisé des managers pour répartir les tâches et les nouveaux mots.

2 FONCTIONNEMENT

2.1 Description

La figure suivante est notre interface graphique finale.



Pour l'interface utilisateur, on a choisi d'utiliser un JPanel. Cependant, la fenêtre s'affiche différemment sur nos ordinateurs et il peut donc être nécessaire de la redimensionner un peu pour voir tous les champs.

Comme on peut le voir, la première ligne sert à gérer les données du jeu à savoir :

— Le score courant du joueur
— Le meilleur score qui est stocké dans un fichier en local pour être conservé entre les parties
— Le temps restant avant le début du jeu ou la fin de la partie (une fois celle-ci lancée)

La ligne suivante permet d'afficher des messages pour donner des informations au joueur (le jeu va bientôt être lancé, le jeu est fini, en cours ou encore le meilleur score a été battu).

Les deux zones de texte sont la zone d'entrée où l'utilisateur pourra taper de la façon la plus fluide possible et la zone de sortie où le texte corrigé s'affiche en couleur :

— gris en attente de correction
— vert si le mot existe en anglais
— rouge s'il n'existe pas
— gras et plus gros si c'est un nom des films célèbres

Enfin, un bouton permet de lancer ou arrêter la partie.

2.2 Fonctionnalités

Notre SpeedTyper permet d'interrompre et de relancer une nouvelle partie à tout moment en appuyant sur le bouton en bas.

Pour le test des mots, nous avons utilisé l'API Oxford Dictionnaries qui teste les mots en anglais avec une requête. Chaque mot correct rajoute sa longueur en points et l'enlève s'il n'existe pas.

Pour le test des noms de films, nous testons l'appartenance à une base de 250 films stockés dans un fichier texte. Un bon nom ajoute sa longueur au score (en comprenant les espaces pour compenser un titre avec que des mots faux).

3 IMPLÉMENTATIONS

3.1 Version naïve

Dans cette première version, le Thread Writer était chargé d'afficher l'UI et de gérer le texte tapé. Cependant, il devait aussi lancé le Timer et, à chaque fois qu'un nouveau mot était entré, il devait créer un Thread Checker qui devait tester ce mot.

Nous nous sommes dit que cette implémentation mettait trop de charge sur Writer et que cela pouvait rendre l'UI moins fluide que désirée.

3.2 Threads managers

3.2.1 • Chargement

Lors de l'ouverture du jeu, un Thread manager *EventDispatcher* est lancé et va s'occuper de lancer les bons Threads aux bons moments. Il commence par lancer l'UI et le Thread manager des entrées *InputManager*. Ensuite, il charge les noms de films depuis un fichier texte local. Enfin, *EventDispatcher* crée le Thread *Timer* (mais ne le lance pas encore) qui va gérer le temps. Une fois tous ses chargements effectués, *EventDispatcher* se met en attente de l'appui sur le bouton Go par l'utilisateur (par un *await* sur la variable de condition *goSignal*).

3.2.2 • Lancement du jeu

Lorsque l'utilisateur appuie sur Go, l'UI envoie un signal sur *goSignal* qui réveille *EventDispatcher*. Celui-ci lance alors le timer qui décompte 3 secondes avant de permettre au joueur de taper du texte. Ensuite, il lance *nbThreads* (que l'on a pris égal à 3) Threads *Checker* qui vont être chargés de vérifier la validité des mots tapés en parallèle. Enfin, *EventDispatcher* se rendort en attendant la variable de condition *overSignal*.

De son côté, *Timer* décompte 3 secondes avant de permettre à l'utilisateur de taper puis relance un décompte de la durée définie pour le jeu (dans *Timer.duration* en ms).

3.2.3 • Déroulement du jeu

Pendant toute la durée du jeu, le Thread *Writer* gère l'UI. En particulier, il empêche l'utilisateur de supprimer les mots déjà validés et récupère le dernier mot à chaque fois que l'utilisateur appuie sur la touche *espace*. Il écrit ce mot dans une *LinkedBlockingQueue* partagée avec *InputManager* qui lui va ensuite se charger de gérer les mots pour laisser *Writer* libre de s'occuper de l'UI.

À chaque fois qu'il récupère un mot, *InputManager* va :

— l'écrire dans une *LinkedBlockedQueue* partagée avec les Threads *Checker*. Cela réveille un Thread *Checker* qui va envoyer une requête à l'API pour tester la présence du mot. En fonction du résultat, il va mettre à jour le score et la couleur du mot dans la fenêtre de sortie.

— créer et lancer un Thread *MovieChecker*. Ce dernier va parcourir la liste des films pour voir si des films commencent par le mot tapé. Si cela n'est pas le cas, il s'arrête. Dans le cas contraire, il s'endort et attend que l'*InputManager* ait traité un nouveau pour se réveiller et traiter le titre correspondant à la concaténation des deux mots. Il répète ces actions jusqu'à tomber sur un des 250 titres ou jusqu'à n'avoir plus aucun des titres qui pourrait correspondre (il s'arrête).

3.2.4 • Fin

Deux scenarii sont envisageables pour finir le jeu :

— Soit le compte à rebours du *Timer* arrive à 0. Dans ce cas, il envoie un signal sur *overSignal* avant de mettre à jour le meilleur score le cas échéant. Ce signal réveille *EventDispatcher* qui va recréer un nouveau *Timer* et attendre un nouveau un appui sur le bouton Go.

— Soit le joueur appuie sur le bouton Stop. Dans ce cas, il ne compte plus écrire donc le Thread *Writer* lui-même peut se charger d'envoyer un signal *overSignal* et d'interrompre le *Timer*.

4
ANALYSE DE LA PROGRAMMATION CONCURRENTE

4.1 Verrous

Pour pouvoir mettre à jour en parallèle le score (par les *Checker* et les *MovieChecker*) et le texte de sortie (écriture du mot en gris, puis de sa correction par un *Checker* et éventuellement de sa correspondance avec un titre par un *MovieChecker*), il fallait verrouiller ces champs. On a pour cela utilisé le mot-clé *synchronized*.

Pour que l'*InputManager* puisse réveiller les Threads *MovieChecker* qui attendaient de nouveaux mots pour compléter leur titre, on a utilisé les mots-clés *synchronized, notifyAll* et *wait* sur la liste des mots traités par l'*InputManager*.

4.2 Signaux

Pour signaler aux Threads concernés que le jeu vient d'être lancé (pour lancer le décompte) ou qu'il vient de se finir (relancer *EventDispatcher*), on a utilisé un verrou et deux signaux *signalGo* et *signalOver* liés à ce verrou.

4.3 Sleep

Le *Timer* ne peut jamais être arrêté car il doit mettre à jour le temps toutes les 0.1s mais on a mis un *Thread.sleep(50) ;* tant qu'il restait plus de 300ms pour permettre d'éviter de l'attente active inutile.

4.4 LinkedBlockingQueue

On a utilisé une première *LinkedBlockingQueue* pour dialoguer entre l'UI et l'*InputManager*. Ce dernier dialogue avec les *Checker* avec une autre *LinkedBlockingQueue* car ces derniers sont en nombre fixe et ne meurent jamais pendant une partie.

Pour les *MovieChecker*, on a choisi d'en créer autant qu'il peut y avoir de titres possibles en parallèle et de les tuer que lorsqu'il est impossible que leur séquence de mots corresponde à un titre. De ce fait, ils sont *synchronized* sur la liste des mots traités par l'*InputManager* et celui-ci leur envoie une *String END_OF_GAME* pour les arrêter lorsqu'une partie est finie.