Keyboard Master

Par Noé LORRET-DESPRET, en CIR1

Description du jeu

Il s'agit d'un Type Racer, c'est-à-dire qu'il faut écrire le plus vite possible une phrase sans faire d'erreur. Le score final est le nombre de lettres (correctes) écrites par secondes (donc écrire plus vite sans faire de fautes = meilleur score).

Fonctionnement du jeu

Prérequis

Tout d'abord, vérifiez que vous avez installé :
- Python (3.10 ou supérieur)
- pip (Package Installer for Python)

Setup

Selon votre version de python, il faudra utiliser py, python ou python3, et soit pip soit pip3

Télécharger une release .zip du projet disponible **en cliquant ici**
Installer les packages nécessaires avec la commande : `` ***py -m pip install -r requirements.txt*** ``
Lancer le fichier principal avec la commande : `` ***py main.py*** ``

Vous pouvez tester le fonctionnement du programme avec la commande : ***py tests.py*** ``

Le code est linté selon pylint, si vous voulez le vérifier, installez pylint : `` ***py -m pip install pylint*** ``
Puis lancez la vérification : `` ***py -m pylint --rcfile=.pylintrc --recursive y .*** ``

Liste de fonctionnalités réalisées

Système de comptes
- Création / connexion
- Username unique
- Mot de passe stocké + crypté

Envoi/réception de requêtes au serveur
- Logique de connexion
- Logique de matchmaking
- Requête de récupération de phrase
- Envoi de donnés continus pour les matchs (update du score)

Database pour les phrases / matchs / comptes
- Interface Python pour communiquer avec le serveur…
- … qui communique après avec la base de donnés

Ecoute des touches appuyés et analyse + réaction
- Listener sur un second thread stoppable
- Logique en fonction de la phrase fetch
- Calcul du score

Fenêtre tkinter avancée (Classe via inheritance, tk.Frame, events…)
- Création d'une classe parent (qui hérite de tk.Tk)
- Création de frames enfants (qui héritent de tk.Frame)
- Lien parent / enfant (envoi d'events, states multi-frames…)

Liste de fonctionnalités à ajouter

Fonctionnalités à ajouter
- Sauvegarder meilleurs score des joueurs
- Affichage d'un leaderboard
- Skill based matchmaking (+ mmr)

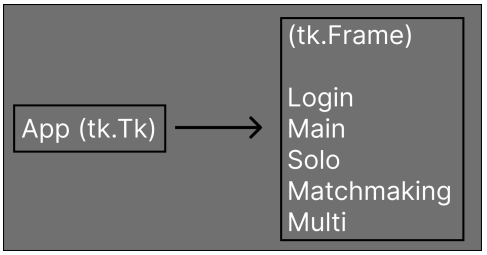
Optimisations dans le code Python
- Meilleur management des erreurs/crash
- Améliorer la fenêtre tkinter
- Uniformisation des techniques utilisées (events.bind vs self.variable dans le parent)

Description de l'algorithme

Ce projet se distingue en 3 parties distinctes :
- L'application Python
- Le serveur Javascript
- La base de données MongoDB

L'application Python est constituée d'une classe principale App (qui hérite de tk.Tk), et possède plusieurs frames (classes qui héritent de tk.Frame).

L'application gère les paramètres principaux, et permet de passer des variables / signaux entre les frames, et les frames permettent d'interagir avec l'app.

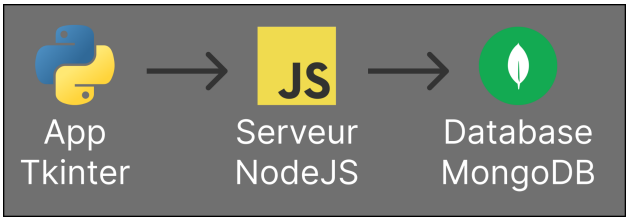


Les frames Solo et Multi contiennent aussi une grosse fonction qui est lancée dans un thread, afin d'écouter les touches pressées et réagir en fonction.

Multi et matchmaking lancent aussi d'autres fonctions (plus petites) dans des threads, afin de laisser l'application utilisable alors que l'on envoie des données au serveur.

Des fonctions supplémentaires permettent à l'app d'envoyer des requêtes au serveur NodeJS, qui gère la logique et la connexion à la database : cela permet de contrôler ce que les utilisateurs peuvent envoyer / demander à la base de données.

Le serveur possède une route api, qui exécute une fonction serverless Vercel, afin d'exécuter la requête de l'utilisateur et lui renvoyer des données.



La base de données est elle uniquement accessible par le serveur, pas de connexion directe des utilisateurs.

Répartition des rôles au sein du monôme

Projet
- Management Github : Noé
- Documentation & fichiers : Noé

Application Tkinter (Python)
- Design : Noé
- Management de la fenêtre : Noé
- Connexion au serveur backend : Noé
- Algorithmes simples : Noé
- Algorithmes complexes : Noé

Backend NodeJS (Javascript)
- Connexion à la database : Noé
- Fonction serverless principale : Noé
- Algorithmes simples : Noé

Base de données MongoDB (NoSQL)
- Administration & privilèges : Noé
- Design des schema : Noé
- Logique multi-documents : Noé

Discussion finale

C'était un projet pour moi optionnel, mais instructif. J'ai pu appliquer des concepts que je connais en théorie, mais je n'avais jamais réalisé (classes custom avec inheritance, pour faire des choses complexes en tkinter, module request pour connecter au serveur).
Sur la partie javascript, j'ai aussi eu l'occasion de pratiquer les fonctions serverless et les routes api que je n'avais encore jamais pu utiliser.
Côté MongoDB, j'ai découvert l'importance des schema pour les données, afin de pouvoir les manipuler / analyser facilement.
En conclusion, un projet instructif sur plusieurs langages (Python, Javascript), frameworks (tkinter, mongoose) et outils (fonctions serverless, database query, et autre).