Dactylogame
Explication de code – Coté serveur

Introduction

On s’intéressera ici uniquement du côté serveur. Donc on ne va pas parler de certaines fonctions Javascript comme randomTexte(), et du CSS.

explicationsImportantes.txt :

« Jeu : Ecrire le plus vite possible. Nom : Dactylogame !!!!!!

Deux joueurs connectés simultanément via websocket..
Une course sur deux suites de noms communs aléatoires et différentes se lance, le joueur qui obtient le plus grand score (noms écrits correctement) gagne la partie.

Lors du lancement de la partie, le premier joueur doit attendre le second joueur (message d'attente)

Sur l'affichage, le joueur voit ce qu'il écrit, son score, et le score en temps réel de son adversaire.
Le chrono est affiché en haut de l'écran, et est le même pour les deux joueurs (30 secondes par défaut).

Lorsque le temps est écoulé, le joueur qui a le plus grand score gagne la partie.
Un bouton "Rejouer" permet de relancer une partie à l'aide d'un rafrachissement de la page. Les deux joueurs commenceront en même temps. »

Comme expliqué rapidement plus haut, j’ai utilisé fastapi et uvicorn pour gérer le serveur, et la transmission de données se fera via Websocket.

Explications de code important et problèmes rencontrés

L’idée principale de projet était simplement de faire jouer deux joueurs et qu’ils puissent voir leur progression en temps réel via fastapi Websocket, une version plus avancée de l’exercice 3 en protocole web.
Après l’initialisation de main.py (initialiser app, app.mount, app.get(«/»), on initialise les variables importantes :

main.py

```
ws_connexion = []
isLaunched = False
```

ws_connexion servira a stocker les joueurs qui se connecteront au serveur et isLaunched permet d’éviter tout lancement de script en double non voulu.
Ici, isLaunched évitera de lancer aux deux joueurs connectés d’exécuter la fonction javascript `launchGame()` deux fois

main.py

```
if not isLaunched:
    await player1.send_text(json.dumps({"type" : "start"}))
    await player2.send_text(json.dumps({"type" : "start"}))
    isLaunched = True
```

```
const ws = new WebSocket("ws://localhost:8000/ws");

ws.onmessage = function (event) {
  let json = JSON.parse(event.data)
            ...
  if (json.type == "start"){
   launchGame()
 }
```

Un peu avant cela, on doit vérifier qu'il y ait bel et bien deux joueurs connectés en même temps, et on peut vérifier cela avec la longueur de ws_connxexion:

main.py

```
import asyncio

while len(ws_connexion) < 2:
  print("En attente de connexion")
  await asyncio.sleep(3) # attente de 3 secondes
```

Je ne sais d'ailleurs pas si cette méthode est une bonne méthode pour attendre un adversaire.
Je n'ai aussi pas pu faire en sorte de laisser entrer uniquement 2 personnes, à cause de nombreux problèmes, comme lorsque l'on rafraîchit sa page et que le serveur considère la venu d'un troisième joueur alors que c'est la même personne. Une solution à cela serait de créer un système de compte pour chaque joueur, avec login et mot de passe.

On utilise ici asyncio pour laisser passer les requêtes sur le serveur grâce à cet await. En effet, mettre :

main.py

```
time.sleep(3)
```

nous fait comprendre qu'il y a un gros problème en rendant impossible la connexion à la page de joueurs après que le premier soit arrivé.

Lorsque deux joueurs sont connectés au même moment, on peut les assigne à des constantes

main.py

```
player1 = ws_connexion[0]
player2 = ws_connexion[1]
```

Ce qui nous permettra d'identifier qui a envoyé une donnée, et choisir manuellement qui doit recevoir une donnée.

Par exemple, si nous souhaitons savoir qui nous envoie un score égal à 2 après avoir reçu une donnée *data* :

main.py

```
data = await websocket.receive_text()
```

Remarque, après quelques tests non-concluant, je n'ai pas trouvé si c'était possible de faire de cette manière :

main.py

```
data1 = await player1.receive_text()
data2 = await player2.receive_text()
```

pour ensuite pouvoir savoir directement qui a envoyé quoi.

On se demande si le websocket qui envoie la donnéeest bien égal à player1 ou player2

main.py

```
if websocket == player1:
      await player2.send_text(json.dumps({"type" : "counter", "counterAdv" :
      data["score"]}))
else:
      await player1.send_text(json.dumps({"type" : "counter", "counterAdv" :
      data["score"]}))
```

Puis on dit à l'autre joueur qu'on a reçu le score de l'adversaire en utilisant

main.py

```
await player1.send_text(json.dumps({"type" : "counter","counterAdv" : data["score"]}))
```

Note : Le score d'un joueur est envoyé à son adversaire à chaque fois qu'il entre un mot complet, et l'envoie en appuyant sur « espace » ou « entrer »

static/script.js

```
function taptap(){ // à chaque fois que l'utilisateur appuie sur une touche
  if(event.key == ' ' || event.key == 'Enter'){
            ... //scoreJou ++ si le mot est correct
            ... //erreurJou ++ sinon (erreurJou ne sert à rien dans le jeu)
    ws.send(JSON.stringify({type: "score", score: scoreJou}))
  }}
```

Pour terminer, pour rejouer une partie, le serveur réinitialise le tableau ws_connexion et les deux joueurs rafraîchissent automatiquement la page, lorsqu'un des deux appuit sur le bouton « rejouer ».

static/script.js

```
function rejouer(){
 ws.send(JSON.stringify({type: "rejouer"}))
}
```

main.py

```
elif data['type'] == "rejouer":
 for conn in ws_connexion:
   await conn.send_text(json.dumps({"type" : "rejouer"}))
 ws_connexion = []
 isLaunched = False
```

static/script.js

```
ws.onmessage = function (event) {
  let json = JSON.parse(event.data)
       ...
       ...
  if (json.type == 'rejouer'){
   window.location.reload()
  }
};
```

Sanna Thomas

J'espère que ce rapport d'entraînement a été assez clair et propre