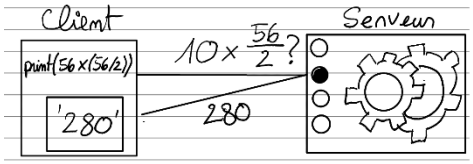
Présentation de la SAE3.02



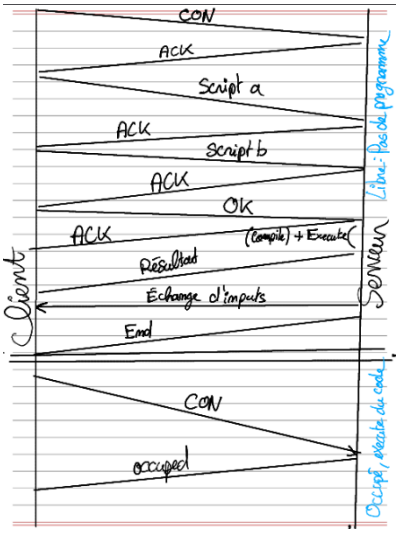
Introduction

Duarnt ces deux derniers mois, ce projet m'a permis de concevoir un système de calcul client/serveur avec une interface graphique. Je vais dû utiliser le module PyQt qui offre la possibilité de développer une interface structurée dans une Class. L'affichage ne devra jamais rester bloquer lorsque par exemple, je téléverse un fichier. Elle devra aussi prendre quelques fonctionnalités en charge, voir plus bas.

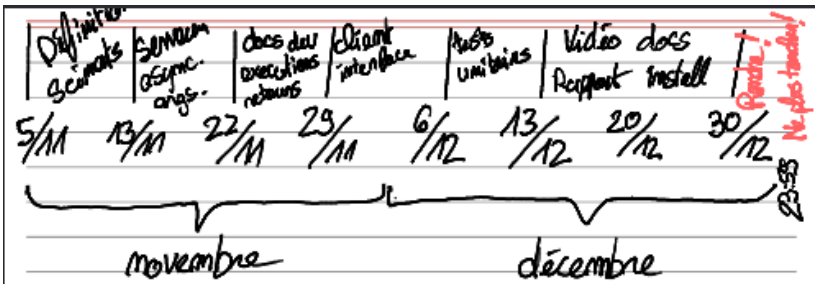
Analyse du besoin

J'ai réalisé des dessins afin de visualiser et planifier mon développement

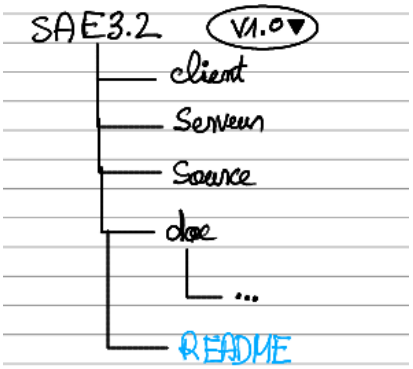
J'ai dessiné le protocole de communication



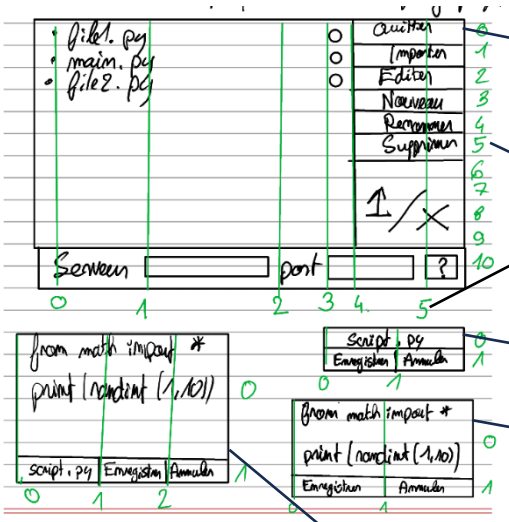
Je me suis donné des missions délimité dans le temps



J'ai défini la structure de mon arborescence



J'ai fait des croquis détaillés pour les interfaces



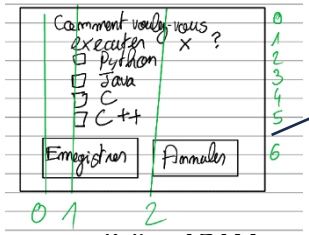
Fenêtre principal

Grâce aux indications en vert je repère facilement les coordonnées vertical et horizontal

Fenêtre renommer

Fenêtre éditer

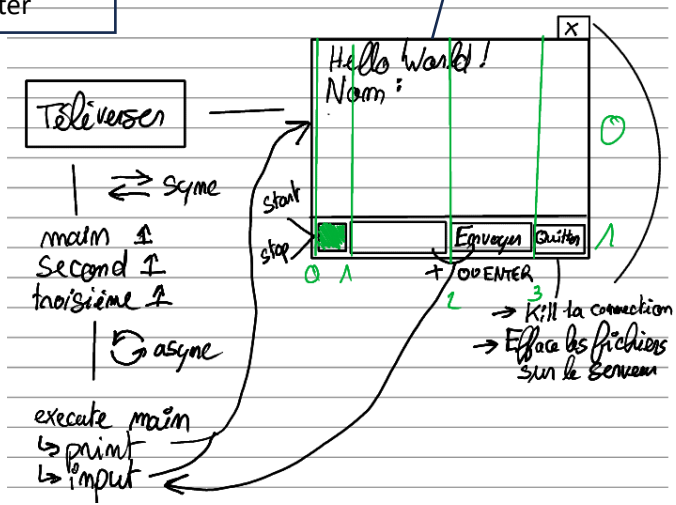
Fenêtre nouveau fichier



Fenêtre choisir langage

Cette version permettait d'envoyer des inputs et de redémarrer le script

La V1 abandonné de la fenêtre console



Kylian ADAM

Présentation de la SAE3.02

Fonctionnalités

Le client sera une interface graphique, intuitif avec facilité d'usage. On pourra y lister les scripts qu'on souhaite envoyer, les corriger aussi si besoin. Le serveur va récupérer les fichiers du client et les stocker, il va executer le script choisit par le client. Le serveur prend en charge 4 langages de programmations : Python, C, C++ et Java. Si je donne un fichier avec une extension qui ne correspond pas à l'un des quatres, alors le client va me demander d'en choisir un parmis les quatres.

J'ai également deux fonctionnalités supplémentaires. Ce système est capable de démarrer autant de serveur que l'utilisateur à besoin, tant qu'on lui défini son propre port TCP. Egalement, le client proposera un moyens d'éditer temporairement les scripts avant qu'ils soient envoyés, c'est à dire que le fichier qui a été utilisé pour importer le script ne sera pas modifié par cette édition.

Dans ma vidéo de démonstration, on peut sur voir sur Wireshark que le traffic réseau entre le client et le serveur se fait en clair. Il vaudrait mieux utiliser des scripts qui ne sont pas sensibles.

Solutions envisagés / mises en oeuvres

Argument du port

Au lancement du serveur, l'administrateur a la possibilité de définir le port d'échange du serveur.

```python
parser = argparse.ArgumentParser()
parser.add_argument('-p','--port',type=int,required=False,help="Spécifiez le port à utiliser.")
args = parser.parse_args()
if args.port != None:
    if args.port >= 0 and args.port <= 65535:
        port = args.port
    else:
        print(f"\033[31mLe port doit être entre 0 et 65535!\033[0m")
        return
```

L'argument **-h** permet d'obtenir l'aide :

```
usage: server.py [-h] [-p PORT]

options:
  -h, --help       show this help message and exit
  -p, --port PORT  Spécifiez le port à utiliser.
```

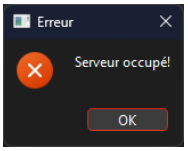
L'argument **-p** permet de définir le port. Il doit être entre 0 et 65535.

server.py -p 18 →

```
Serveur démarré sur le port 18 et en attente de connexions...
```

Serveur occupé

Le client ne reste pas connecté au serveur, il échange jusqu'à obtenir le résultat de l'execution. Mais si un client tenterai de se connecter au serveur alors qu'il execute toujours, alors le serveur renvoit le message

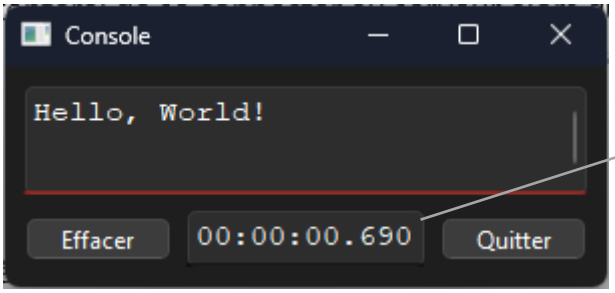


Alors, l'utilisateur pourra directement essayer un autre serveur.

Chronomètre

La fenêtre console ne se bloque lors des différents traitements, il affiche un chronomètre en direct, qui s'arrête une fois les informations retournées par le serveur

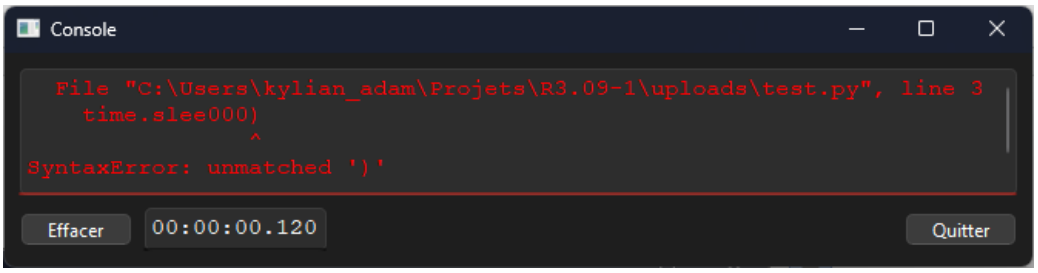
Présentation de la SAE3.02



La durée varie selon le langage, la distance du serveur, et la file d'attente pour la thread, le script lui même

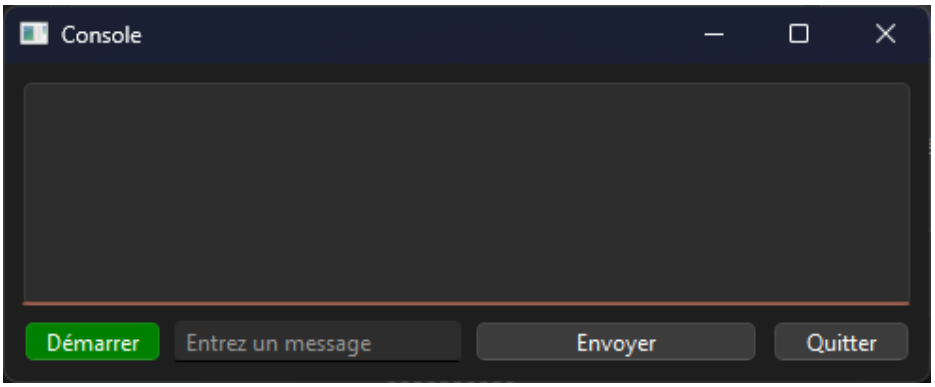
Le serveur renvoie deux informations: Le type et le texte détaillant le retour.

Le type de retour stdout ou stderr, selon lequel on change la couleur du texte. Blanc comme en haut pour stdout et rouge pour stderr comme ci-dessous.



Console Avancée

J'aurait voulu faire une interface différente pour ma console, en proposant aussi la possibilité de démarrer le programme séparrément, puis de l'arrêter, même de le redémarrer. Et puis égallement de mettre un champ pour envoyer des messages lors de l'execution, pour répondre aux inputs dans un script. L'interface que j'ai développé était celle-ci:



Mais c'était difficile à mettre en place, j'ai préférer mettre en priorité les fonctionnalités importantes et veiller à la qualité de l'échange, donc je n'ai pas pu réaliser cet ajout.

Conclusion

J'ai réussi à concevoir mon premier système client/serveur. Je n'ai pas totallement respecté mon protocol initiale ; lorsque le client envoie un fichier il était nécéssaire de prévoir des solutions de vérification côté serveur. Donc avant d'envoyer le script, j'envoie le nom du fichier puis sa taille en octet, et comme ça le serveur écoutera par paquet de 1024 octets jusqu'à ce qu'il reçoive la taille totale. Ainsi il pourra sauvegarder les données dans un fichier stocké dans le répertoire upload.

Présentation de la SAE3.02

Pour éviter d'emagasiner trop de fichier, j'ai prévu qu'à l'arrêt du serveur, tous les fichiers du répertoire upload soient supprimés. Si j'aurai voulu les garder, j'aurai dû concevoir un système de compartimentation des fichiers qui seront accéssible avec un protocol type AAA. Aussi prévoir un délai d'inactivité après lequel le compte de l'utilisateur sera supprimé. Ça rend rend la solution rapidement plus compliqué.

Finallement il y a beaucoup d'autres améliorations qui sont implémentables, mais comme toujours, c'est le temps qui manque pour réaliser toutes mes idées. Je reste vraiment fier de cette première version. Je pense que je me suis accordé entre 60 et 70 heures complètement concentré dans ce projet.