Compte-rendu du projet de Concepts Avancés de Programmation

Mohamed Amine, PIRABAKARAN Thanushan

Mai 2022

Introduction :

Le but de ce projet est d'écrire un programme informatique permettant de développer un simulateur de fonctionnement d'un système de carburant d'un avion, ce simulateur aura aussi pour but d'entraîner l'utilisateur.

Prérequis / Configuration:

Le projet a été fait sur la machine virtuelle proposée par la fac sous python 3.8.5.
Les librairies importés tkinter, os et random principalement.

Composition du projet:

Le projet est composé de 5 dossiers et d'un README.

Le README.md contient nos prénoms et nos noms.

Le dossier data contiendra Log.txt prenant le nom et le mot de passe de chaques utilisateurs, le dossiers images contient toutes les images utilisées dans le programme,

Le dossier save contient deux fonctions : save et load permettant respectivement de sauvegarder notre partie et d'obtenir à partir d'un fichier une instance d'Avion utilisable dans le main.

Le dossier score contiendra les scores et la date de toutes les parties finis. Enfin modules contient tous nos fichiers ".py". Notamment notre main.py.

Démarrage du programme:

Structure de base:

La structure de base est composée de tous les éléments nécessaires dans un fichier graphique.py. Il est composé des vannes, réservoirs, moteurs et des pompes. Ces éléments sont stockés dans une classe nommée Avion et contenant la liste des éléments, soit des listes de réservoirs, moteurs, pompes et des vannes pour moteurs et vannes entre moteurs et pompes.

L'avion en détail:

La classe Avion est composée du décorateur @property afin de définir les noms et les états plus facilement.

Structure notable:

```
def alea_pannes(self):
        import random
        noms_pompes = [element._nom for element in self.pompes]
        noms_tanks = [element._nom for element in self.reservoirs]
        if random.random() < 0.50:
            self.panne(random.choice(noms_pompes))
        else:

self.panne(random.choices(noms_tanks,weights=[5,1,5],k=1)[0])
```

L'une des fonctions notables est alea_pannes dans la class Avion. Dans le else on prend l'un des tanks, on va definir 5 pour le poids de Tank1 et de Tank3 donc il y a 5 fois plus de chance d'avoir Tank1 ou Tank3 que d'avoir Tank2. Vu que le Tank2 est plus grand, il a donc moins de chance de tomber en panne.

Partie graphique :

- `class Root(Tk):` Classe qui crée la racine de l'interface graphique grâce à la librairie TKinter.
- `class Menu(object):` Classe qui crée l'interface du menu du simulateur, en créant un canvas sur la racine de la classe Root, sur lequel on ajoute des boutons, qui redirige vers d'autre interface grâce à des fonctions.Pour l'interface de la demo, on initialise `Avion`, `Illustration` et `Panneau`.
- `class Illustration(object):` interface du système carburant de la simulation, via un canvas, on crée des formes et les lignes pour créer cette interface grâce à la fonction `creer_formes`. Si tu veux expliquer les refresh je te laisse faire.
  La synchronisation de l'avion à l'interface graphique:
  Illustration contient des fonctions privées par exemple get_couleur qui retourne la couleur de la pompe que l'on veut en fonction de l'état, couleurs_pompes est donc une valeur global à la classe. Puis on l'actualise.
  La fonction `refresh(self, element_nom, etat):` Cette fonction prend un dictionnaire de fonction et auto gere pour ne pas se retrouver avec de multiples structures conditionnels du type "if elif elif...". On gagne du temps et un peu plus d'efficacité.

- `class Login(object):` Classe qui crée un système d'authentification, pour cela, une fonction `creer_dossiers` crée les dossiers nécessaires au bon fonctionnement de la class.
  La fonction `login` crée l'interface de connexion et contient aussi un bouton d'inscription, qui renvoi vers l'interface d'inscription.
  La fonction `connect` parcours le fichier où sont stockés tous les identifiants, et si un identifiants correspond, l'authentification est validé, sinon renvoi un message d'erreur.
  La fonction `registration_interface` crée l'interface d'inscription.
  La fonction `registration` stock le username et le mot de passe données et vérifie s'il n'est pas vide. Si oui renvoi un message d'erreur. Sinon parcours le fichier où sont stockés tous les identifiants, et si un identifiant correspond, renvoi un message d'erreur, sinon crée un compte avec ses identifiants.
- `class Historique(object):` Cette classe a pour but de créer deux boutons distincts pour afficher l'historique et une bouton aide qui a pour but de créer une légende des éléments présents dans le schéma. L'historique parcourt les éléments présents dans le fichier score, pour afficher le score et la date de l'utilisateur.

Conclusion:

Ce projet nous a beaucoup appris sur le langage Python en nous introduisant les notions de P.O.O. Ca nous a également appris à manipuler concrètement TKinter, voir les limites de ce dernier. Aussi de bien gérer notre temps et bien nous organiser en groupe pour le bon avancement du projet.

Ce qu'il manque:

- La gestion des sources des moteurs.
- Une meilleure gestion des modules utilisés ou bien l'utilisation d'un package.
- Des canvas plus malléables.
- Une gestion de la save et des loading plus efficace exemple un fichier pour la sauvegarde et le loading
- Makefile plus efficace avec par exemple un champ debug, install etc.
- Diminution des variables globales, utilisation de plus de classes etc.
- Animations de vannes, des réservoirs.