Compte rendu du projet

Simulateur de fonctionnement d’un système à carburant d’un avion

Weng Marc
Richard Clément

Année universitaire : 2021-2022

Introduction

Dans le cadre du projet de l'UE IN407, on nous a demandé de réaliser un projet, qui a pour but de développer un simulateur d'un système à carburant d'un avion.

Dans ce compte rendu, nous allons simplement décrire et expliquer les différentes méthodes (fonctions) utilisées. Chaque méthode possède aussi une documentation, cette documentation est contenu dans le fichier où est implémenté la méthode.

Un Makefile est aussi disponible, même si le fichier contient seulement une commande pour exécuter le code et une commande pour supprimer le dossier "__pychache__".

Sommaire

Dans le fichier : login.py

Ce fichier possède un total de 2 méthodes, l'utilité de ce fichier est de créer une interface de connexion pour l'utilisateur.
Ce fichier est en lien avec le fichier "logs" qui regroupe tous les utilisateurs qui ont déjà été enregistré, les informations inscrites sont sous la forme de :

- Nom d'utilisateur - Mot de passe - Score maximal - série de note des sessions précédentes

La méthode switch_game

- But : Cette méthode permet de faire la transition entre l'interface de connexion et l'interface de la simulation.

- Principe : On teste si le nom et le mot de passe entrés par l'utilisateur n'est pas vide. Si ce n'est pas cas alors :
    - On ouvre le fichier "logs" afin de vérifier si l'utilisateur actuel a déjà utilisé le simulateur à l'aide de son nom et du mot de passe qu'il a donné :
        - Si l'utilisateur est connu, on récupère le score maximal qu'il a pu faire durant toutes ses précédentes sessions pour l'enregistrer un objet de la classe Player, qui a été créé durant l'ouverture du fichier.

        - Si l'utilisateur n'est pas connu, on inscrit son nom et son mot de passe à la suite des autres utilisateurs, et en initialisant son score maximal à 0.

    - On ferme la fenêtre de connexion pour ouvrir la fenêtre du simulateur.

  Si le nom et le mot de passe entrés par l'utilisateur est vide, on affiche un message d'erreur pour l'utilisateur.

La méthode login_screen

- But : Cette méthode permet la mise en place de l'interface de connexion avec l'aide de la bibliothèque Tkinter.

- Principe : La fenêtre possède 2 parties, l'un étant utilisé pour l'affichage de texte, et l'autre pour les entrées :
    - La partie "entrées" possède 2 espaces de saisie pour que l'utilisateur puisse y mettre, dans l'ordre, son nom puis son mot de passe.

    - Enfin un bouton de connexion est placé juste en dessous.
      Ce bouton est relié à la méthode switch_game.

  Un canvas est situé en dessous du bouton de connexion.
  À l'initialisation, le texte, que ce canvas affiche, est vide.

Dans le fichier : game.py

Ce fichier possède un total de 3 méthodes, une grande partie du code concerne la création de l'interface de la simulation.

La méthode save

- But : Cette méthode sert à sauvegarder les informations de l'utilisateur à la fin de la simulation.

- Principe : On met à jour le fichier "logs" en sauvegardant l'ancienne version pour y rajouter par dessus les nouvelles données, puis on supprime la version actuelle du fichier pour en créer un nouveau derrière avec les données qui ont été mise à jour :
    - On ouvre d'abord le fichier "logs" en lecture, puis on crée une chaîne de caractères vide, qui sera utilisée pour créer un nouveau fichier "logs".

    - On récupère toutes les lignes dans la variable créée, et au moment où il y a similitude par rapport au nom et au mot de passe de l'utilisateur, on effectue la modification.

    - On ajoute à la fin de la ligne, qui concerne l'utilisateur, une note sur 10 par rapport au score qu'il a obtenu durant cette session, puis on rajoute le reste des lignes.

    - On supprime l'ancien fichier "logs" à l'aide de la méthode remove, contenu dans la bibliothèque os.

    - Enfin, on crée le nouveau fichier "logs".

La méthode closing

- But : Affiche une boîte de message pour confirmer la fermeture de la fenêtre de la simulation.

- Principe : Si l'utilisateur confirme la fermeture, la méthode save est lancée, puis l'interface de la simulation se ferme et le programme prend fin.

La méthode game_screen

- But : Création de l'interface de la simulation.

- Principe : L'interface est divisé en 2 parties :
    - Une partie est constituée du système de l'avion, ainsi que des boutons qui ont pour principe de provoquer des événements, comme la panne d'une pompe, le meilleur score et le score actuel sont aussi affichés.

    - L'autre partie est réservée pour les boutons de commande, comme par exemple, l'ouverture ou la fermeture d'une vanne.

  Durant la création des différents canvas, frame et label, on crée aussi les classes associés aux différentes parties du système telles que les pompes, les vannes, les réservoirs, ou bien les moteurs, les tuyaux ne possèdent pas de classe, car ils n'ont pas besoin d'être modifiés.

  Quand tous les objets ont bien tous été créés, on instancie alors une nouvelle variable de la classe Simulation, qui regroupe tous les objets créés jusqu'à maintenant pour le système, ainsi que le l'utilisateur (classe Joueur).

  Concernant les boutons, ils sont reliés par des méthodes qui seront décrites plus tard, mais pour faire court, les méthodes utilisées permettent de changer l'état de l'objet et redessiner le canvas ou bien changer le texte.

  À la fin de la méthode, la dernière ligne sert à détecter quand l'utilisateur appuie sur le bouton "x" pour fermer la fenêtre, cela déclenche alors la méthode closing.

Dans le fichier : playerCommands.py

Ce fichier possède un total de 14 méthodes, elles sont utilisées lorsque l'utilisateur actionne un bouton depuis l'interface de la simulation. Étant donné du caractère répétitif de certaines commandes, on s'attardera seulement sur les méthodes qui regroupent d'autres méthodes du fichier ou bien les méthodes "importantes".

La méthode update_powered_valve

- But : Ajoute ou retire les réservoirs qui alimentent, ou non, les moteurs, en fonction de l'état qui est donné en argument..

- Principe : La méthode utilise le nom de la vanne, qui est passé en argument, afin d'ajouter, ou de retirer, le nom du réservoir de la liste, cette liste est l'une des variables d'instance de la classe Motor.

  Ensuite, la méthode fait appel à une autre méthode pour mettre à jour le canvas qui possède le texte affichant les noms des réservoirs qui l'alimentent.

La méthode interact_valve

- But : Met la vanne en position ouverte ou fermée.

- Principe : À l'aide du nom de la vanne passé en argument, la méthode va tout d'abord mettre à jour la variable "état" dans la classe de la vanne en question, puis il modifie le canvas de la vanne pour la mettre dans la position ouverte, ou bien fermée.

  En fonction de la vanne, la méthode se comportera différemment, en effet :

  - Si la vanne est l'une des vanne reliant les réservoirs :
    - Elle met à jour la quantité de carburant dans les réservoirs, en fonction de la vanne, et si le réservoir est vidé, alors, elle mettra à jour l'état du réservoir, son canvas, ainsi que sa quantité de carburant à l'aide la méthode not_empty_tank.

  - Si la vanne est une autre vanne que celle reliant les réservoirs:

- Elle mettra à jour la liste des réservoirs alimentant le moteur, en fonction de la vanne, en appelant la méthode update_powered_motor.

- Elle vérifiera ensuite si toutes les pompes de tous les réservoirs sont en marche, et si c'est le cas, alors, une boîte de message s'affiche puis des points seront retirés à l'utilisateur, une mise à jour du canvas possède le texte du score sera effectuée.

La méthode interact_pump

- But : Met la pompe secondaire en état de marche ou d'arrêt.

- Principe : À l'aide du nom du réservoir passé en argument, la méthode va d'abord :
    - Vérifier l'état de la pompe secondaire afin de voir s'il faut le mettre en état de marche ou d'arrêt.

    - Ensuite, elle vérifie si le réservoir est vide, si c'est le cas, alors, une boîte de message s'affiche pour signaler que la priorité est l'équilibrage des réservoirs, et non, la mise en marche ou l'arrêt de la pompe secondaire, des points seront ensuite retirés à l'utilisateur, une mise à jour du canvas possède le texte du score sera effectuée.

  En fonction de l'état actuel de la pompe secondaire, on utilisera la méthode pour activer la pompe ou l'arrêter, la méthode possède bien-sûr une partie pour manier le canvas pour rendre lisible l'état de la pompe.

La méthode broke_first_pump

- But : Mettre la pompe principale en panne.

- Principe : Mets la pompe principale en panne, puis modifie le canvas pour que l'utilisateur aperçoive que la pompe est en panne.

  La méthode lance la méthode two_pump_broken.

La méthode broke_second_pump

- But : Mettre la pompe secondaire en panne.

- Principe : Mets la pompe secondaire en panne, puis modifie le canvas pour que l'utilisateur aperçoive que la pompe est en panne.

  La méthode lance la méthode two_pump_broken.

La méthode two_pump_broken

- But : Actionne automatiquement la vanne en fonction du réservoir où les 2 pompes sont en panne.

- Principe : La méthode vérifie si les 2 pompes d'un réservoir sont en panne à chaque fois que l'une des 2 méthodes décrites au-dessus est utilisée.
  Si les 2 pompes sont en panne, l'utilisateur se voit retirer des points, puis en fonction du réservoir, une vanne est mise en position ouverte.

La méthode empty_tank

- But : Active la vidange du réservoir

- Principe : Met le réservoir dans l'état "vidé", et met à jour sa quantité de carburant à 0.
  La méthode modifie ensuite le canvas.

La méthode not_empty_tank

- But : Met le réservoir dans l'état différent de "vidé", et met à jour sa quantité de carburant en fonction de la variable passée en argument.

- Principe : En fonction de la vanne actionnée et de l'état initial du réservoir :
  - Si le réservoir est vide, alors, l'utilisateur se voit gagner des points, puis la méthode met le réservoir dans l'état "plein" (pas vraiment plein).

  La méthode met à jour la quantité des réservoirs, et modifie leur canvas

Dans le fichier : draw.py

Il n’y aura pas de détail concernant les méthodes dans ce fichier, la totalité des méthodes du fichier étant des méthodes pour modifier le canvas de l’objet en question, pompe, réservoir, vanne, ou bien moteur.

Dans le fichier : system.py

Ce fichier ne possède que des définitions de classe.
Étant donné que ces classes ne possèdent rien de particulier, je ne vais pas vous les décrire en détail. Ces classes ont tous un héritage simple de la classe object, elles n'ont pas de surcharge, elles possèdent tous un accesseurs et un mutateur pour chacune de leurs variables d'instance.

- Class Pump (object)

Variable d'instance	Type	Description
id	int	Un identifiant, utilisé pour les canvas
state	int	La valeur de l'état de la vanne : - -1, quand la pompe est en panne - 0, quand la pompe est à l'arrêt - 1, quand la pompe est en marche

- Class Tank (object)

Variable d'instance	Type	Description
canvas	Canvas	Le canvas du réservoir
id	int	L'identifiant, utilisé pour le canvas
first_pump	Pump	La pompe principale
second_pump	Pump	La pompe secondaire
quantityMax	int	La quantité maximale, valeur de départ
quantity	int	La quantité de carburant actuel
color	str	La couleur utilisé pour le réservoir
state	int	La valeur de l'état de la vanne : - 0, quand il est vide - 1, sinon

- Class Valve (object)

Variable d'instance	Type	Description
id	int	Un identifiant, utilisé pour les canvas
state	int	La valeur de l'état de la vanne : - 0, quand la vanne est fermée - 1, quand la vanne est ouverte

- Class Motor (object)

Variable d'instance	Type	Description
id	int	Un identifiant, utilisé pour les canvas
canvas	Canvas	Le canvas du réservoir
powered	list	Une liste des réservoirs qui l'alimente

- Class Player (object)

Variable d'instance	Type	Description
user	str	Le nom de l'utilisateur
pwd	str	Le mot de passe de l'utilisateur
max_score	int	Le meilleur score
score	int	Le score actuel
scoreCanvas	Canvas	Le canvas des scores
action	int	Le nombre d'action effectuée
good	int	Le nombre de bonne action effectuée

- Class Simulation (object)

Variable d’instance	Type	Description
player	Player	L’utilisateur
tank1	Tank	Le réservoir 1
tank2	Tank	Le réservoir 2
tank3	Tank	Le réservoir 3
valveT12	Valve	La vanne T12
valveT23	Valve	La vanne T23
valve12	Valve	La vanne 12
valve13	Valve	La vanne 13
valve23	Valve	La vanne 23
motor1	Motor	Le moteur 1
motor2	Motor	Le moteur 2
motor3	Motor	Le moteur 3