RAPPORT DE PROJET

Simulateur de fonctionnement d'un système carburant d'un avion

Réalisé par:

Yann Topilko TD1
Elias Djebbour TD3

SOMMAIRE

Introduction

Dans le cadre d'un projet scolaire, il nous a été demandé de réaliser un simulateur de de carburant. Pour cela, nous nous sommes répartis les tâches de manière efficace pour avancer réaliser ce projet.

I. Interface Graphique

Afin de réaliser ce projet, nous nous sommes servis du module tkinter. Ayant déjà travaillé avec celui-ci lors de précédents projets, il nous semblait évident de l'utiliser pour se sentir plus à l'aise lors de la réalisation de l'interface. Celle-ci se divise en trois interfaces distinctes. La première correspond à la page d'inscription/connexion ci-dessous (Figure 1). La deuxième correspond au tableau de bord du pilote (Figure 2). Et enfin le simulateur de carburant (Figure 3).

Figure 1

Figure 2

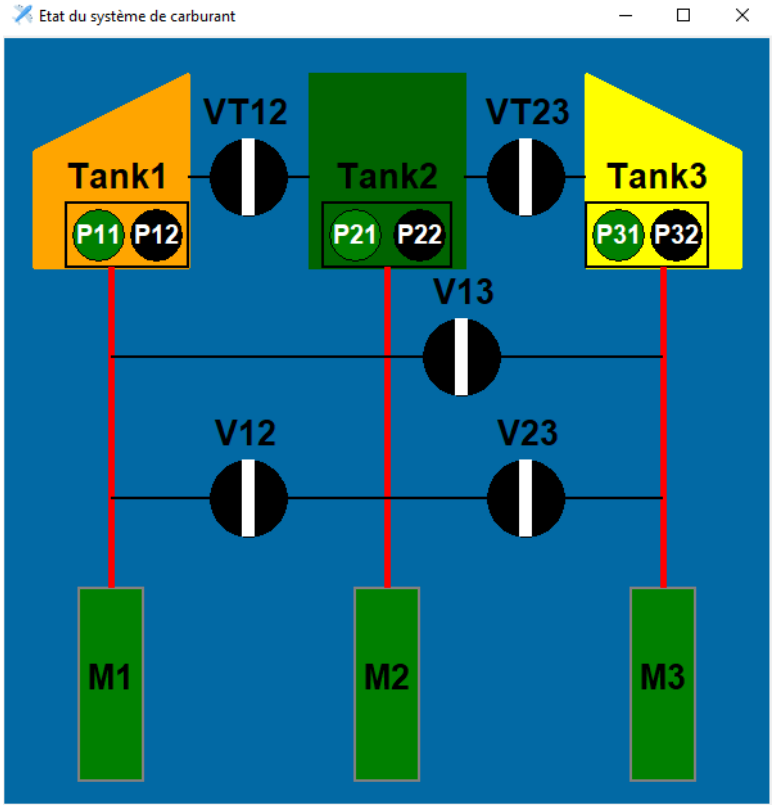

Figure 3

En ce qui concerne le système d'authentification, celui-ci va vérifier si le nom de l'utilisateur apparaît bien dans le dossier "Users", et si le mot de passe saisi correspond. Si c'est le cas, l'utilisateur se connecte.

Il existe 2 modes dans cette applications :

- Practice Mode : Ce mode permet de générer des pannes manuellement en cliquant sur les différents éléments de l'interface (Pompes et Tank)
- Simulation Mode : Ce mode est un exercice qui consiste à résoudre plusieurs séries de pannes. Pour chaque série de pannes, l'utilisateur gagne un point en fonction de si celle-ci est bien résolue. Ainsi les pannes ne peuvent être provoquées par l'utilisateur, mais par le système.

II. Structure de données

En ce qui concerne l'organisation des données, nous avons décidé de fonctionner par classe. Cela semblait le choix le plus efficace car cela nous permettait de créer des objets comme des vannes ou encore des pompes… Et ainsi de modifier leurs états en appliquant des méthodes dessus. Pour ce faire nous avons créé une classe mère "Composant" et toutes les autres classes à savoir "Pompes", "Vannes", "Tank", "Moteur" et "Flux" héritent de cette classe mère. Ainsi elles peuvent hériter du constructeur de la classe mère et ainsi alléger le code et le rendre plus lisible.

Pour chaque classe, des méthodes sont utilisables et elles permettent dans la très grande majorité des cas de changer l'état des composants. Le reste des méthodes permet de récupérer l'état du composant ou plus précisément pour la pompe de secours, de savoir quel moteur elle alimente.

En ce qui concerne l'état des composants, trois sont définis pour les pompes et deux pour les moteurs, vannes et tanks.
En effet, il y a l'état de marche qui s'applique à tous les composants et dans ce cas l'état vaut 1. Il y a aussi l'état de pause (ou fermeture / éteint) qui lui aussi s'applique aussi à tous les composants et dans ce cas l'état vaut 0. Et enfin l'état de panne qui ne s'applique uniquement sur les pompes et dans ce cas l'état vaut -1.
Bien-sûr, cela ne veut pas dire qu'un tank ne peut pas être "en panne" mais sa panne veut simplement dire qu'il est vide.

Pour se retrouver dans les flux nous avons décidé de les organiser de la manière suivante afin de savoir lesquels allumer ou éteindre (Figure 4).

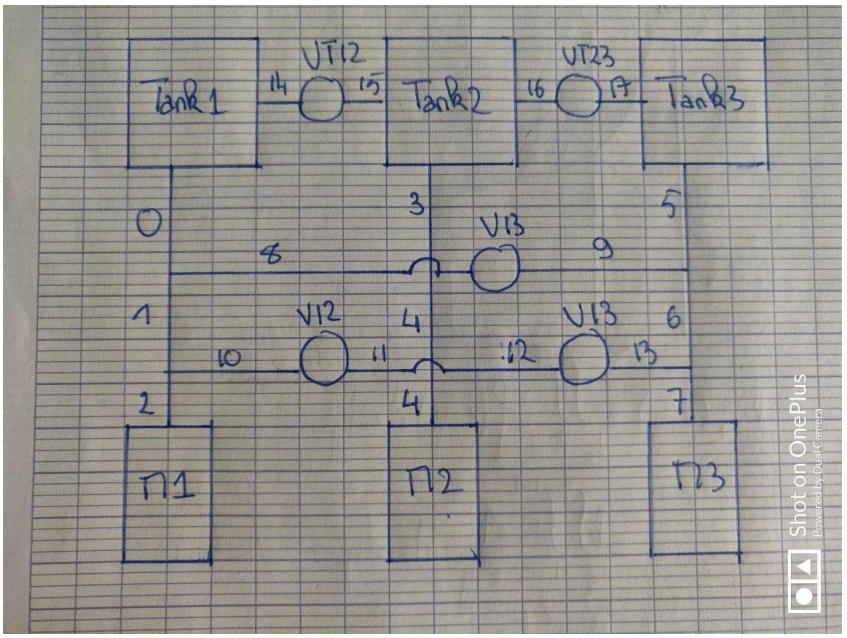

Figure 4

Ici, chaque flux possède un numéro qui correspond à son indice dans la liste des flux ainsi, lorsque l'on veut travailler avec les flux, il nous suffit de regarder ce schéma.
De plus si un flux est allumé plusieurs fois alors on lui met un compteur comme ça tant que celui-ci n'est pas à 0 alors on n'éteint pas le flux.

III. Simulation, checking et résolution

Dans cette dernière partie, nous traitons de la simulation et de comment les différents composants interagissent.
Dans le mode simulation, un fonction boucle est créée, et elle permet de générer à plusieurs reprises les séries de pannes tant que la simulation n'est pas finie. Pour savoir quand le cette fonction s'arrête, on regarde si la variable compteur valant de 0 au départ est différente du nombre de séries à résoudre et si oui alors le programme sort de cette boucle et affiche un pop up demandant si l'utilisateur veut voir son historique ou non (Figure 5). SI oui, l'historique est affiché (Figure 6)

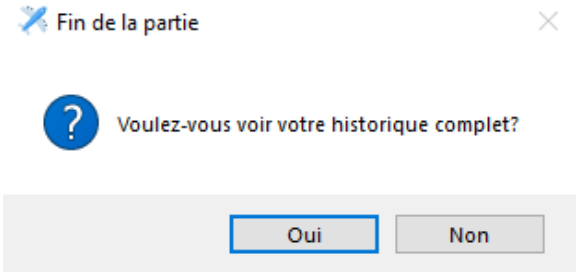

Figure 5

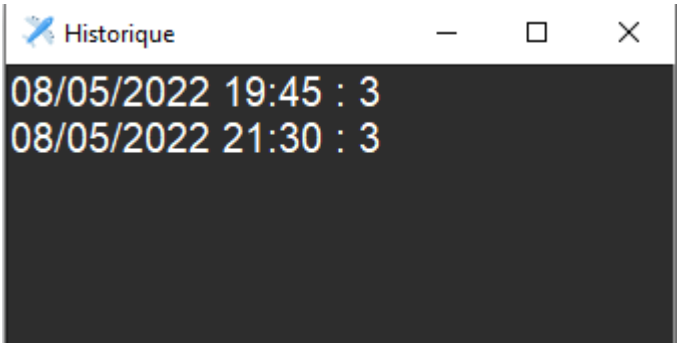

Figure 6

Avant même que la fonction boucle soit lancée, une fonction “set_système” est appelée et c’est celle-ci qui va définir l’état de base des composants. Puis un fois cela fait, elle va générer une panne et lancer la fonction boucle.

En ce qui concerne la panne, elle est générée de manière aléatoire toutefois, celle-ci doit respecter des critères. En, effet, pas plus de 3 pompes ne peuvent tomber en panne car dans le cas contraire, il serait impossible de régler le problème. Ainsi il est possible de générer entre 1 et 3 pannes de pompes (principale et de secours confondues). Et ce principe est le même pour les tanks. En effet , pas plus de 2 tanks ne peuvent tomber en panne car dans le cas contraire, nous ne pouvons plus permettre aux pompes d’alimenter les différents moteurs. Ainsi nous pouvons générer 1 ou 2 pannes de tanks. De ce fait, cela permet à l’utilisateur de rencontrer une multitude de cas afin d’être le mieux former en cas de situation réelle!

Comme dit précédemment, une fonction “boucle” est appelée juste après la génération de panne et permet donc de mettre en place la simulation. Cette fonction est appelée toutes les 500 millisecondes et possède deux comportements. Soit les trois moteurs sont alimentés (nous en reparlerons plus tard), soit ce n’est pas le cas et alors celle-ci appelle une fonction “checking”

Cette fonction checking est un “melting-point” de plusieurs autres fonctions. En effet, tous les 500ms elle va appeler 5 autres fonctions. La première étant "éteindre_flux_ moteur2" qui va avoir pour objectif de vérifier que si un moteur est éteint, tous les flux qui les relient aux pompes soit correctement éteint si nécessaire.
Prenons un exemple. Ici (Figure 7), la fonction "éteindre_flux_ moteur2" éteint le moteur “M1” ainsi les flux se situent entre lui et le Tank 1 car celui-ci est vide. Toutefois, si l'on venait à remplir le tank 1 ainsi que l’on activait la pompe de secours “P12” alors les liens s'allumeront et le moteur aussi.
Ce sont justement les fonctions “checkM1”, “checkM2” et “checkM2” qui vont s’occuper de ça. En effet, elles ont pour objectif de traiter l’intégralité des cas dans lesquels les moteurs doivent être allumés et de ce fait les bons flux.
La dernière fonction à être appelée est “allumer_vanne”. Celle-ci va seulement se contenter d’allumer les bon flux lorsque des vannes sont ouvertes.

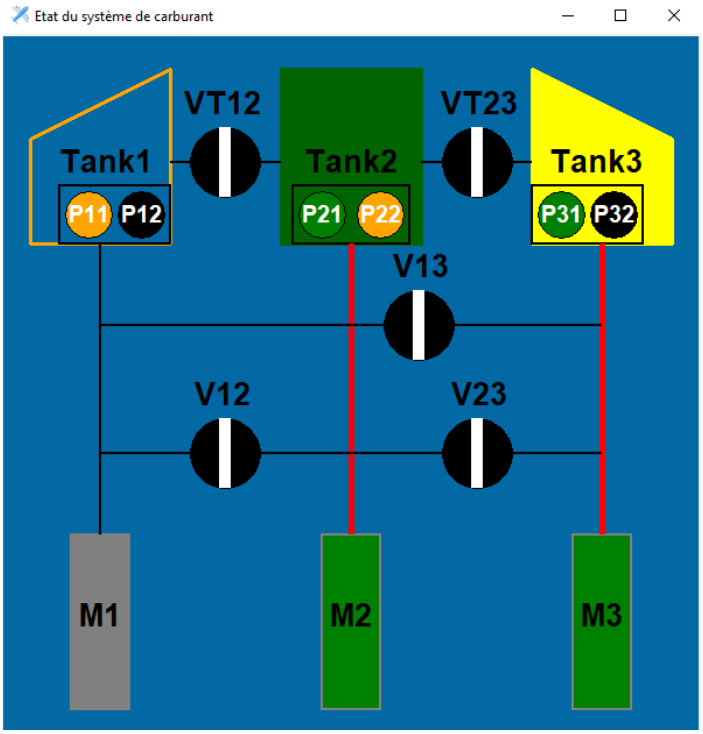

Figure 7

Maintenant que tous les moteurs sont allumés, “boucle” peut enfin rentrer dans son premier comportement et donc vérifier si les moteurs sont bien allumés.
Pour vérifier cela, une fonction “résolution” est appelée et elle va vérifier que si la pompe principale d’un moteur n’est pas disponible alors c’est la pompe de secours du moteur qui doit alimenter le moteur comme énoncé dans le sujet. Toutefois, si la pompe de secours est aussi en panne alors les autres pompes de secours pourront alimenter le moteur.
Si le moteur est “mal” alimenté alors la fonction retourne 0 ce qui correspond au nombre de points que l’utilisateur a gagné sur cette série de panne. Dans le cas contraire, il en gagne 1. Cette valeur est ajoutée à une variable score qui s’actualise à chaque séries de panne et qui est utilisé dans le message de fin d’exercice.

Conclusion

Au cours de ce projet, nous avons pu mettre en application plusieurs connaissances que nous avons apprises au cours de ce semestre, notamment l’implémentation des classes. De plus, cela nous a permis de nous documenter afin d’apprendre de nouvelles choses afin de réaliser ce que l’on souhaitait faire pour ce projet. En ce qui concerne les difficultés rencontrées, cela se résume principalement à la gestion de toutes les conditions qu’il fallait vérifier pour savoir s’il fallait allumer tel moteur ou tel flux etc.. Enfin en ce qui concerne les améliorations de ce code nous aimerions rendre celui-ci plus optimisé car nous avons l’impression que plusieurs conditions se ressemble et que nous pourrions gérer de cas d’un coup.