Université Versailles Saint Quentin-En-Yvelines

UFR des sciences

Département Informatique

Simulation d'un système de carburant d'un avion de chasse

Présenté par :

- Ahmed Saadi
- Helmi Abed

Année Universitaire 2019-2020

1 Introduction :

Dans le domaine aéronautique et plus précisément pour un avion, il reste primordial de savoir gérer et planifier sa politique de carburant. Pour se faire, un système de carburant est identifié pour gérer les opérations de ravitaillement et de vidange, ainsi que les flux de carburant entre le moteur et les réservoirs pendant le vol. Ce système permet aussi de transférer le carburant d'un réservoir à l'autre en cas de nécessité

Dans ce cadre nous avons réalisé un projet dont le but est d'implémenter un simulateur pour ce système de carburant afin que les pilotes puissent s'entrainer avant d'aller sur le terrain. Pour ce faire nous avons opté pour le langage de programmation C++ pour diverses raisons tel que le fait qu'il soit un langage hybride donc offre plus de fluidité dans la programmation.

la réalisation et l'élaboration de ce projet a été répartie entre binôme.

Notre projet est divisé en trois modules : Interface, Modèle et Base de données.

2 Architecture du logiciel :

Notre projet se compose de trois modules principaux basés sur une architecture MVC (Modèle – Vue – Contrôleur), celle-ci est donc modulaire et rend le logiciel facilement maintenable car la logique du code est séparée en trois parties que l'on retrouve dans des fichiers différents et dont la gestion interne est entièrement autonome.

2.1 Le Modèle :

Cette partie gère les données de l'application stockées dans des fichiers ou entrées par l'utilisateur. Elle assure aussi la cohésion entre les différents composants du système carburant que nous allons décrire dans les lignes qui suivent. Ainsi donc le modèle est composé de deux parties majeures : le système carburant et la base de données.

2.1.1 Le système carburant :

C'est le module du programme qui modélise sous forme d'un modèle calculatoire de données compréhensible par l'ordinateur le système carburant d'un avion de chasse, ses états et ses fonctions. Il s'agit donc de la représentation sous forme de classes des composants comme les réservoirs, les pompes, les vannes. . . Toutes ces classes communiquent entre elles fournissant ainsi les fonctionnalités d'un vrai système d'alimentation selon le modèle qui a été demandé.

Se référant au paradigme de la programmation objet, nous avons pensé le Système Carburant comme étant lui-même un objet dit « central » composé de réservoirs, de moteurs, de Vannes, de pompes. La classe Avion permet donc de modéliser notre « objet central ». En suivant donc le même raisonnement, nous avons établis une hiérarchie entre les différents composants.

2.2 La base de données :

Ce module permet de gérer l'enregistrement des tentatives de gestion du système de simulation de carburant dans une base de données (SQLITE).

Cette opération débute par le fait de l'authentification du pilote dans le système via une interface de login. Par la suite toutes les tentatives seront enregistrées dans la table d'historique spécifique à chaque utilisateur.

Pour ces deux modules, un listing complet des méthodes de ces classes est donné dans la suite du rapport.

3 La vue :

Cette partie intègre un module important de l'application : l'interface graphique.

L'interface graphique assure l'interaction entre l'utilisateur et le modèle qu'il ne voit pas.

Notre interface graphique possède deux fenêtres principales qui s'affichent au moment de la simulation. Tout d'abord la première fenêtre (figure 1) affiche l'état du Système et de tous ses composants (état des vannes, des pompes, et des réservoirs. . .).

Elle permet la gestion de l'état de la simulation (mettre en pause, arrêter, refaire), la gestion des connexions et des services associés tels que la création d'un compte, l'affichage des notes de l'utilisateur...

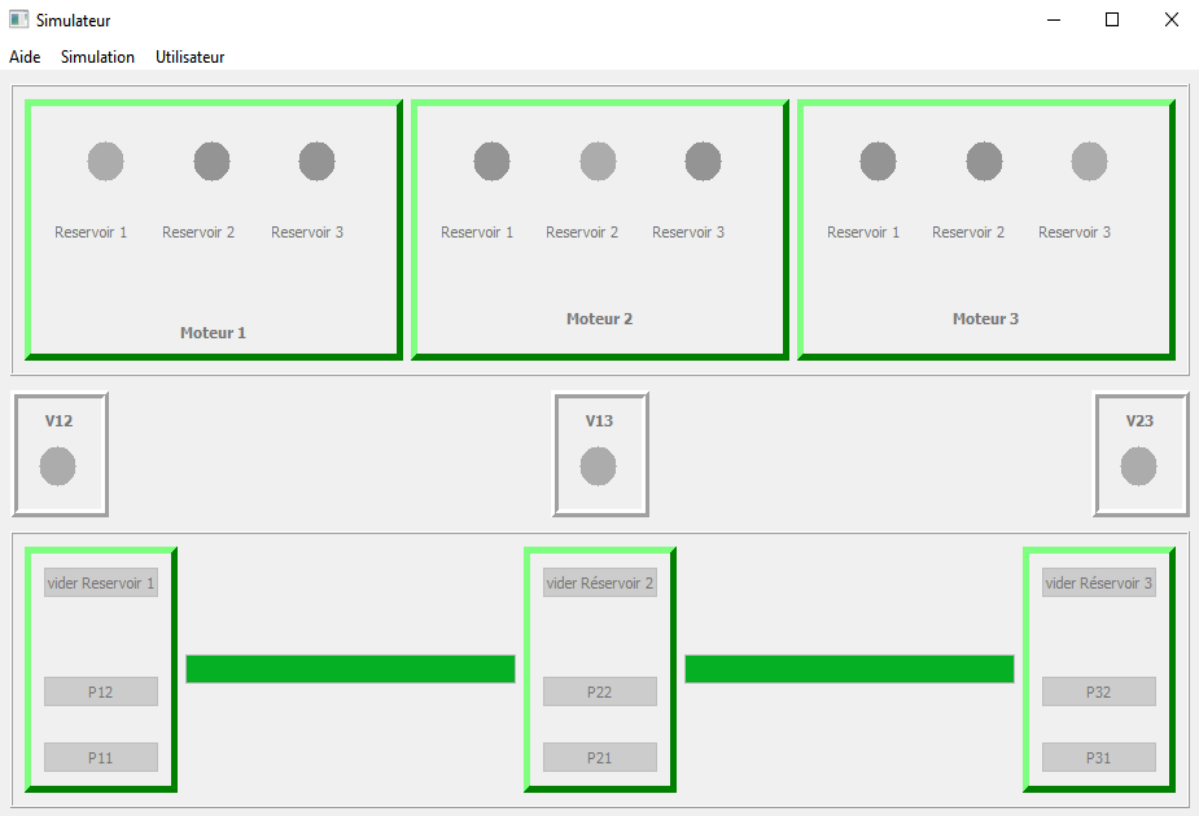

Figure 1: Fenêtre principale.

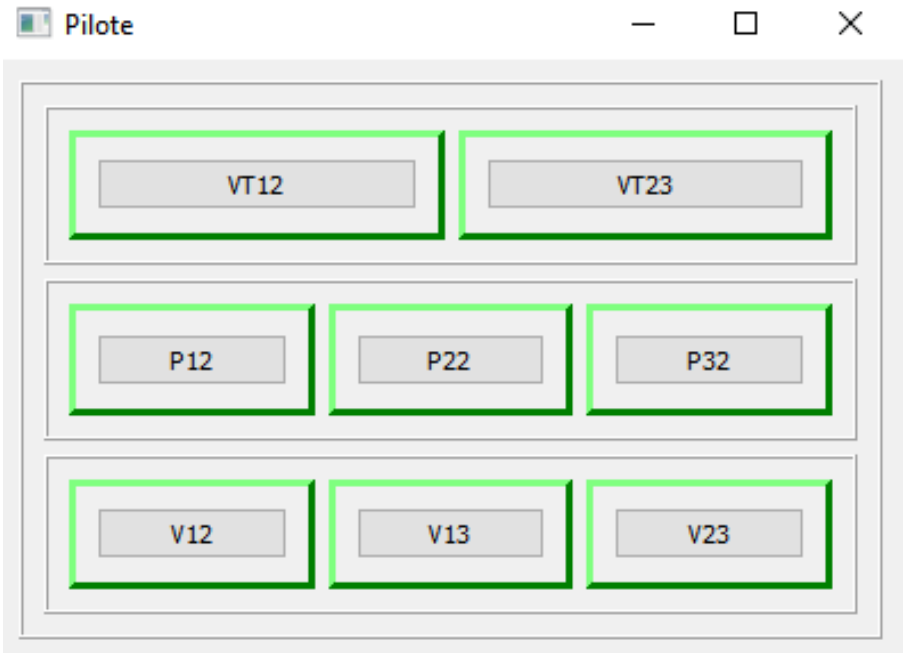

Figure 2 : Tableau de bord.

La seconde fenêtre (figure 2) représente le tableau de bord. Celui-ci contient les boutons gérant l'ouverture/fermeture des pompes et des vannes. Chaque action reportée entraine une modification du modèle.

5 Listing de quelque fonction de l'application :

5.1 Classe Pompe :

Cette classe représente les différentes pompes de l'Avion où chaque pompe alimente le moteur en carburant.

5.2.1 Propriétés :

etat: int: cette propriété représente les états des pompes qui individuellement chaque pompe spécifique peut avoir l'état 1 pour un fonctionnement normal , -1 pour représenter l'état en panne et 0 pour représenter l'arrêt de la pompe en question.

5.2.2 Fonctions membres :

Pompe::Pompe (int etats) : modifie l'état de la pompe.

Void Pompe :: panne() : met la pompe en panne .

void Pompe :: arret() : arrête la pompe .

ostream & operator << (ostream & os, Pompe p) : affiche l'état de la pompe.

Pompe :: ~Pompe() : détruit un objet pompe.

5.3 Classe réservoir :

Cette classe représente le réservoir d'un système carburant où chaque réservoir a deux types d'états : vidange et rempli. En effet s'il est alimenté par un réservoir voisin il peut être à la fois vidange et rempli. Cependant au moment où la pompe s'ouvre à nouveau entre les deux réservoirs il retourne aux états vidange et vide.

5.4.1 Propriétés :

rempli:bool: représente si le réservoir est vide ou non.

Pompe: p1 : cette propriété contient la pompe primaire du réservoir.

Fonction d'acces : Pompe::Pompe& getPompe1()

Pompe:p2 : cette propriété contient la pompe de secours du réservoir.

Fonction d'accès : Pompe::Pompe& getPompe2()

5.5 fonction membre :

Reservoir::Reservoir(bool remplissage) : met la pompe 1 en état marche et rempli le réservoir.

ostream & operator << (ostream& os, Reservoir r) : affiche l'état du réservoir (rempli/ vide).

void Reservoir :: vidanger() : vidange le réservoir.

bool Reservoir :: getRempli(): rempli réservoir en carburant.

QPalette Reservoir::getCouleur() : renvoi la couleur du réservoir.

5.6 classes avion :

Cette Classe stocke tous les éléments de l'avion (Réservoirs/vannes/pompes).

5.6.1 Propriétés:

Reservoir R[3] : chaque avion contient trois réservoirs.

Vanne v[5] : chaque avion a cinq vannes en totale.

Int moteur [3][3]: cette propriété représente le moteur (1ere dimension) et les réservoirs qui l'alimentent (2eme dimension).

5.6.2 fonction membre :

Avion::Avion(): un constructeur par défaut car un avion commence toujours dans un même état.

Void Avion::reset() : Remise à zéro l'état de l'avion lors d'une réinitialisation de l'utilisateur.

int Avion :: nbReservoirVidange(): Fonction renvoyant le nombre de réservoirs vidangés**.**

bool Avion :: peutFonctionner():: Renvoi un booléen indiquant s'il est encore possible de réparer l'avion sachant qu'il faut au moins trois pompes fonctionnelles et un réservoir non vidangé pour indiquer cet état.

bool Avion :: actionNecessaire():: Fonction indiquant s'il existe une action du pilote à faire pour revenir dans un état optimal.

5.7 Base :

Class identification : Cette Classe permet la connexion ou l'inscription d'une personne auprès de l'application. Si l'utilisateur va cocher la case Inscription et que l'identifiant n'est pas déjà pris ; un nouveau compte sera créer.

Figure 3 : fenêtre d'authentification.

void Identification::on_validation_accepted() : vérifie si l'utilisateur existe dans la base de donnée.

Conclusion :

L'objectif de notre projet a été atteint puisque nous sommes parvenu à implémenter différentes fonctionnalités du simulateur de système de carburant, et ce en présentant une application capable de générer des exercices pour le pilote afin de lui permettre d'enregistrer son historique ainsi que l'ensemble des notes qu'il a obtenu.

Cependant une authentification via un compte reste nécessaire pour accéder aux différentes fonctionnalités de l'application de Simulation .