Ludivine BONNET
Cylia ADOUANE
TP :B

Compte-rendu : Projet C++

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

I) Introduction

Le projet consiste à réaliser un mini-simulateur de carburant d'un avion en langage C++. Cette application permet à un pilote de pouvoir tester sa capacité à réagir face à une série de pannes successives auxquelles il pourra être confronté lors d'un vol réel. Pour évaluer ses compétences, une note finale lui sera attribuée à chaque fin de test.
Le système contient trois réservoirs, chacun étant constitué de deux pompes. Quand le système fonctionne correctement, chaque pompe primaire alimente un moteur précis. La seconde pompe est une pompe de secours, elle n'a donc pas d'utilité si la première pompe n'est pas en panne.
Nous avons deux types de vannes : des vannes entre deux réservoirs , des vannes entre les pompes et moteurs.
Les vannes sont là pour aider le pilote à faire face aux divers pannes générées aléatoirement par le test.
Dans cette application, le pilote ne rencontrera pas de pannes au niveau des vannes et des moteurs.

Pour effectuer ce projet, nous avons d'abord schématisé le sujet afin de mieux visualiser ce qui était demandé et par la suite nous avons décomposé ceci sous forme de tâches pour répartir le travail.

En première partie, nous avons suivi la description du système carburant comme c'était écrit dans le sujet. Cela nous a permis de créer le squelette de l'application dont les différentes classes que nous avons besoin au départ.
Ensuite, nous avons commencé au fur et à mesure à faire la 1ère partie, après la 2ème partie, pour enfin finir avec l'interface graphique et connecter le tout ensemble.

II) Objets du système

Lorsque nous listons les objets qui composent le système, nous pouvons nous rendre compte qu'il existe 2 types d'objets :
- ceux ayant un état de type booléen (c'est-à-dire soit true soit false) comme par exemple une vanne qui est soit ouverte ou fermée.
- ceux ayant un état de type float (que l'on peut exprimer sous forme de valeur) comme par exemple les pompes qui ont trois états (en panne, en marche ou en arrêt). Ce sont des objets auxquelles nous ne pouvons pas attribuer seulement 2 états.

1) Etat_BOOL : classe mère

La classe ***Etat_BOOL*** est la classe mère de 3 objets : Vanne_PM, Vanne_RR, Moteur. Elle a juste un attribut ***etat*** qui est de type booléen et des méthodes pour récupérer l'état (***getEtat()***) et le modifier (***setEtat()***).

a) Vanne_PM : classe fille de Etat_BOOL

La classe ***Vanne_PM*** hérite de la classe Etat_BOOL. Elle représente une vanne qui relie 2 moteurs et 2 pompes. C'est pour cela qu'elle contient deux pointeurs de type ***Moteur*** et deux pointeurs de type ***Pompe***. Par exemple, la vanne V12 relie le réservoir T2 au moteur M1 et le réservoir T1 au moteur M2. L'état de type booléen représente ici l'ouverture de la vanne. Lorsque la vanne est ouverte (le flux de carburant ne circule pas), l'état est égal à true et lorsque la vanne est fermée (le flux de carburant circule), l'état est égal à false. Nous avons redéfini l'opérateur < < de manière à pouvoir observer l'état d'une pompe lorsqu'on utilise cet opérateur. Nous avons défini la classe ***System*** comme étant « classe friend » pour permettre au système d'avoir accès aux moteurs ainsi qu'aux réservoirs sur lesquels la vanne pointe.

b) Vanne_RR : classe fille de Etat_BOOL

La ***Vanne_RR*** représente une vanne qui relie 2 réservoirs. C'est pour cela qu'elle a en attribut 2 pointeurs de type ***Réservoir***. De même que la ***Vanne_PM***, son état true représente son état ouvert (c'est-à-dire que le flux de carburant ne circule pas) et son false représente son état fermé (le flux de carburant circule).

c) Moteur : classe fille de Etat_BOOL

Le ***Moteur*** représente un moteur du système. Son état false signifie qu'il n'est pas en marche, c'est-à-dire en panne et son état true signifie qu'il est en marche. Il possède un attribut nommé ***alim*** qui désigne la pompe qui alimente ce moteur. Cet attribut est de type pointeur.

2) Etat_FLOAT : classe mère

La classe ***Etat_FLOAT*** est la classe mère de 2 objets : Reservoir et Pompe. Elle est identique à Etat_BOOL à la différence que son attribut etat est de type float.

a) Reservoir : classe fille de Etat_FLOAT

C'est la classe qui représente un réservoir du système. Pour représenter le fait qu'un réservoir contient 2 pompes (une pompe primaire et une pompe secondaire), nous avons mis 2 pompes en attribut. L'état du réservoir représente le nombre de kérosène en unités arbitraires que contient le réservoir. Pour traduire le fait que le réservoir T2 est plus faible en capacité que les 2 autres réservoirs, nous avons attribué 9 pour l'état de T2 et 30 pour l'état des autres.

b) Pompe : classe fille de Etat_FLOAT

C'est la classe qui représente une pompe. L'état d'une pompe prend 3 valeurs 0 (ce qui signifie qu'elle est à l'arrêt), 1 (en marche) et 2 (en panne). Nous avons redéfini l'opérateur < < pour écrire l'état de la pompe (pompe à l'arrêt, en marche ou en panne).

III) Implémentation du système

La classe ***System*** contient tous les objets du système : les 3 réservoirs, les 2 vannes_RR, les 3 vannes_PM et les 3 moteurs. Elle contient indirectement la pompe primaire et la pompe secondaire de chaque réservoir car ces pompes sont stockées dans l'objet réservoir. Elle contient des méthodes qui sont les actions que le pilote peut effectuer :
- **actionner une Vanne_RR** (qui relie 2 réservoirs)
Cette méthode consiste à équilibrer le niveau des 2 réservoirs qui sont liés à la vanne en question. Elle vérifie également si la deuxième vanne a été actionnée, auquel cas où elle va rééquilibrer le niveau des 3 réservoirs du système.
- **actionner une Vanne_PM** (qui relie 2 pompes et 2 moteurs)
Cette méthode consiste, en cas de panne des 2 pompes du réservoir qui relie le moteur, à actionner la pompe de secours du deuxième réservoir qui relie la vanne et ainsi changer de pointeur alim(pompe qui alimente le moteur) qui deviendra cette pompe de secours.
- **mettre en marche la pompe de secours d'un réservoir**
Elle contient également une méthode qui consiste à remettre le système au point départ (c'est-à-dire quand le système fonctionne normalement sans pannes, sans pompes de secours en marche et avec toutes les vannes ouvertes).

L'opérateur < < fait appel aux opérateurs < < des objets contenus dans le système cités plus haut.

Nous avons défini les classes Panne et tableau_bord comme étant « friend ». Cela permet à la classe Panne de pouvoir injecter des pannes dans le système et au tableau_bord d'avoir accès aux différents objets que contient le système.

IV) Implémentation de la classe Panne

Différentes pannes sont possibles : la panne d'une pompe d'un réservoir, la panne de 2 pompes d'un même réservoir et la vidange d'un ou plusieurs réservoirs. Pour traduire cela, nous avons implémenter une classe ***Panne*** qui contient ces actions sous forme de méthodes. La classe Panne contient un pointeur sur un système et peut ainsi modifier ce système. Les différentes méthodes de cette classe sont les différentes pannes que l'on peut injecter au système et les différents exercices dont la série de test est constituée. Les méthodes sont la panne de chaque pompe du système, la vidange de chaque réservoir du système ainsi que les exercices. Les exercices sont :

1er niveau :
- exercice 1 : panne de la pompe primaire du réservoir T1.
- exercice 2 : panne de la pompe primaire du réservoir T2.
- exercice 3 : panne de la pompe primaire du réservoir T3.

2ème niveau :
- exercice 4 : réservoir T3 vide
- exercice 5 : réservoir T1 vide

3ème niveau :
- exercice 6 : on tire au sort parmi 3 pannes possibles : soit la panne de 2 réservoirs de T1, de T2 ou de T3, soit la panne des réservoirs T1 et T2.

Les exercices de niveau 1 valent 1 point si le pilote a effectué le bon geste sinon 0, de même pour les exercices de niveau 2 qui valent 2 points et de niveau 3 qui valent 3 points. Ceci fait un total de 10 points si le pilote a effectué un « sans fautes » à tous les exercices.

V) Sauvegarde des pilotes

Pour sauvegarder les différents pilotes du système et ainsi leur permettre de s'authentifier au système, nous avons créé une classe ***Pilote***. Cette classe contient l'id du pilote, son mot de passe ainsi que son historique de notes. En fonction du nombre de tests que le pilote a effectué, une note finale lui est attribuée. Celle-ci constitue la moyenne de toutes les notes qui ont été sauvegardées dans l'historique du pilote. Nous avons choisi de conserver l'historique dans une structure de type liste chaînée, ainsi on peut stocker autant de notes du pilote que l'on souhaite. Pour avoir un aperçu sur l'entraînement du pilote au cours du temps, nous avons rassemblé toutes les notes ainsi que la moyenne dans un fichier propre à chaque pilote qui sera nommé historique_piloteid.txt. Dans ce nom, id étant l'id du pilote.

VI) Interface graphique

Notre interface graphique a été réalisée à l'aide de Qt. Elle est composée de 2 fenêtres qui sont deux classes : dessin_system et tableau_bord.

1) Classe dessin_system

La classe ***dessin_system*** permet de représenter le système. Cette classe hérite de QgraphicsScene. Nous avons dessiné 3 rectangles qui constituent les 3 réservoirs, ils sont remplis en bleu lorsqu'ils ne sont pas vides et sont remplis de blanc lorsqu'ils sont vides. Nous avons dessiné les pompes des réservoirs, la pompe primaire de chaque réservoir étant toujours le plus à gauche dans un réservoir et la pompe secondaire le plus à droite. Une pompe en marche est représenté par un cercle de couleur vert, une pompe en panne par un cercle de couleur rouge et une pompe à l'arrêt par un cercle de couleur noir. Nous représentons également les différentes vannes du système. Celles-ci sont symbolisées par un cercle noir. Un rectangle blanc est dessiné par dessus. Celui-ci est dessiné à l'horizontal lorsque la vanne est fermée et à la vertical lorsque la vanne est ouverte.

2) Classe tableau_bord

Cette classe hérite de Qwidget et permet de faire apparaître tous les boutons sur lesquels un pilote pourra actionner. Pour permettre à un pilote de lancer un test, celui-ci doit d'abord d'authentifier. Nous avons enregistré dans notre système 2 pilotes (Max et Camille). Le mot de passe de Max est « ciel » et le mot de passe de Camille est « monde ». Lorsque l'on clique sur un de ces pilotes, le terminal demande d'entrer le mot de passe, si le mot de passe saisi est incorrect, le terminal affiche « mot de passe incorrect », sinon le pilote est connecté correctement (cette fonctionnalité a été implémentée dans le slot action_pilote1() pour Max et dans le slot action_pilote2() pour Camille). Lorsque l'authentification est réussie, la variable pilote_actuel de la classe prend le numéro du pilote connecté (1 pour Max et 2 pour Camille). Une fois connecté, le pilote peut lancer une série de tests et ce un nombre de fois illimité et s'il le souhaite afficher son historique de notes. Le pilote pourra agir sur les boutons du tableau de bord comme l'ouverture d'une vanne, la marche d'une pompe selon le type d'exercice. Quand le pilote a fini sa série de tests, il appuie sur le bouton « Fin Test ». Cette action va enregistrer automatiquement la nouvelle note obtenue à son historique. Il peut également se déconnecter en fin de test et revenir ainsi sur la fenêtre d'accueil et aussi appuyer sur le bouton « Quitter » pour quitter le système.

VI) Conclusion

Ce projet nous a permis de développer notre capacité à travailler en équipe, à s'organiser et à concevoir un projet. Il nous a permis également d'approfondir nos connaissances sur le langage C++ et d'apprendre à se servir d'une bibliothèque standard déjà présente (Qt).