**Projet C++ IN505:**
**Simulateur de fonctionnement d'un système carburant d'un avion**

Hugo Tedeschi
Amir Hammad

## Introduction:

Le simulateur aura pour but d’alimenter en carburant un avion. Et il sera ensuite possible au pilote de s’exercer pour son apprentissage.

Dans la première partie de ce projet, un système carburant d’un avion serra en mémoire afin de lier les different oject du système.

Le système carburant d'un avion alimente ses moteurs en carburant.
Il est composé de:
plusieurs réservoirs,
d'un système de pompes,
de vannes,
de moteurs.

Des pannes ont été ajouter pour dynamiser la simulation et entrainer le pilote en condition réel, par exemple:
la panne d'une pompe pendant le vol ou un réservoir qui se vide.

Pour assurer l'acheminement du carburant à tous les moteurs pendant le vol, il est possible de modifier le système pour mettre en marche une pompe de secours ou alimenter un moteur par un autre réservoir.

Le pilote a accès au système carburant pendant le vol et peut faire des modifications à sa façon.

Il peut actionner pompes de secours, ouvrir ou fermer les vannes pour changer l'acheminement du carburant.

Pour éviter les erreurs au cours d'un vol, le pilote a besoin de s'entraîner.

## Description du système:

Une classe abstraire Element permet de définir des comportements dont l'implémentation se fait dans ses sous-classes.

### Objets:

- Moteur
- Reservoir
- Vanne
- Pompe

**Réservoir:**
Le réservoir possède 2 attributs:
Le volume du réservoir.
L'état du réservoir, vide ou s'il lui reste du carburant.

```
class Reservoir: public Element              // tank1 2 3
{
    private:
        bool plein;
        int volume;   // correspond au volume du reservoir.
    public:
        Reservoir();
        Reservoir(int etat, int volume);
        bool getPlein();
        void setPlein(bool plein);
        ~Reservoir();
        void description();
};
```

**Pompe:**
La pompe possède 2 attributs:
L'état de la pompe Active ou inactive.
Le second état de la pompe Fonctionnel ou Défaillant .

```
class Pompe: public Element         // P1 et P2 dans Reservoir
{
    private:
        //Reservoir reservoir;    //reservoir d'ou provient le carburant
        bool fonctionnel;       // booleen indiquant sir la pompe est en panne ou pas

    public:
        Pompe();
        Pompe(const int etat, const bool fonctionnel);
        bool getFonctionnel();
        void setFonctionnel(bool fonctionnel);
        void checkFontionnelle();
        //~ Pompe(const Pompe &p);
        ~Pompe();
        void description();

};
```

**Vanne:**
la vanne possède un seul état Ouvert ou Fermer.

```
class Vanne: public Element // (VT12,V13,V23),(VT12 ,VT23)
{
    private:
    public:
        Vanne();
        Vanne(int etat);
        void description();
        ~Vanne();
};
```

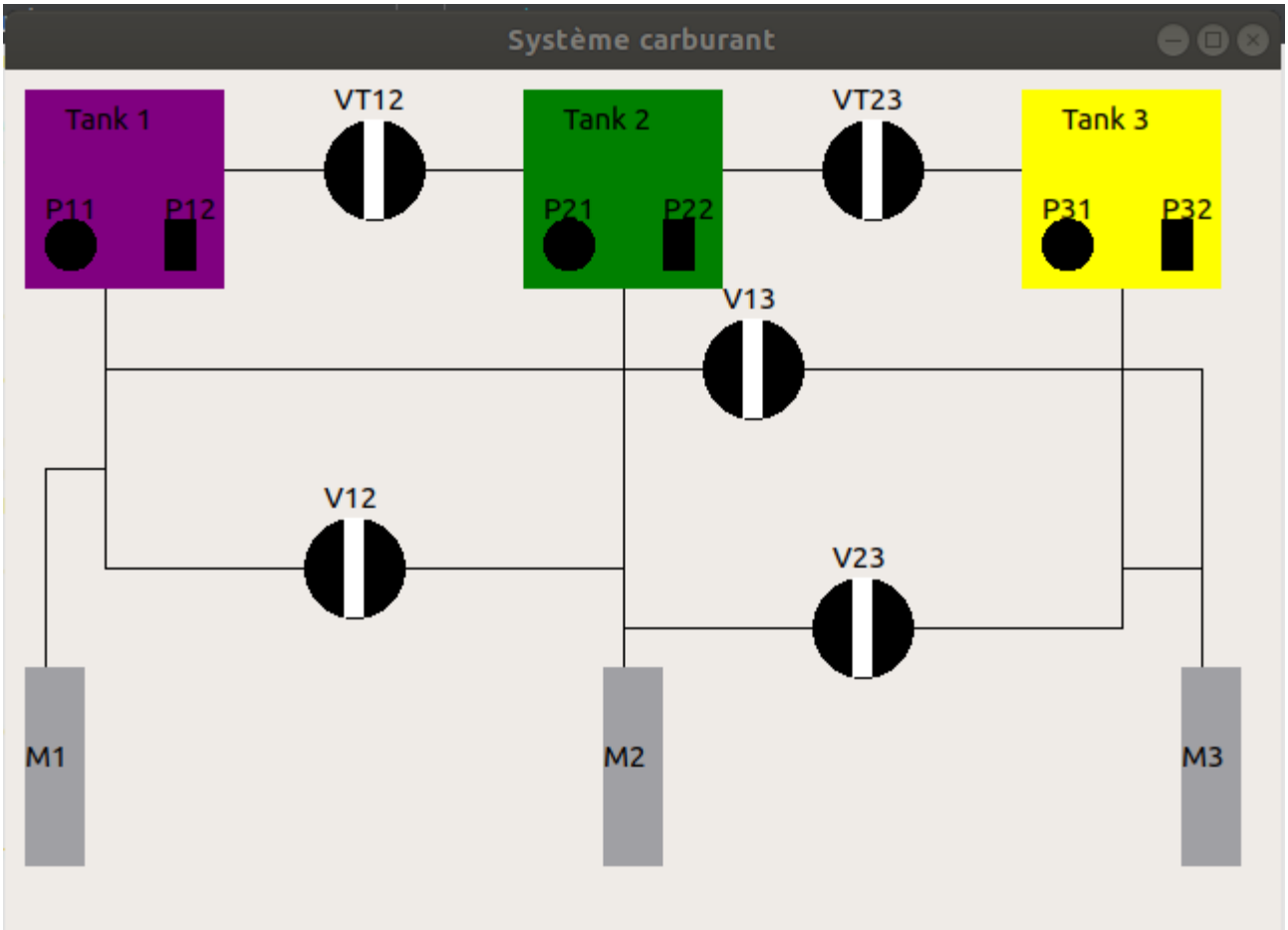
**Moteur:**
le moteur est associe a un réservoir qui l'alimente ainsi que la pompe.

```
class Moteur:public Element                // M1 M2 alimenter
{
    private:
        Pompe pompe;
        Reservoir reservoir;
        int etat;
    public:
        Moteur();
        Moteur(int etat);
        //~ Moteur(const Moteur& m);
        Reservoir getReservoir();
        Pompe getPompe();
        void setReservoir(Reservoir R);
        void setPompe(Pompe P);
        void description();

        ~Moteur();

};
```

**Interface:**
Nous disposons aussi d'interface graphique représentant le système de carburant et des boutons que le pilote utilisera pour s'entrainer.



L’interface graphique du tableau de bord du pilote permet au pilote de gérer le système carburant de l'avion ainsi que déclencher des événements de pannes du système.

| | |
|---|---|
| VT12, VT23, V12, V13, V23, | Ouvir et fermer les vannes |
| P12, P22, P32, | Démarrer ou arrêter les pompes de secours |
| p_P11, p_p12 , etc | Panne des pompes |
| v_R1, v_R2, v_R3 | Vidange des réservoirs. |


L’initialisation et les modifications du systeme de carburant ce font dans le fichier TableaudeBord.

Pour chaque variable d'instance, une méthode getter renvoie sa valeur tandis qu'une méthode setter définit ou met à jour sa valeur.

```
class TableauBord
{
    private:
        Vanne V12, V13, V23, VT23, VT12;
        Pompe P1, P2, P3, S1, S2, S3;
        Reservoir R1, R2, R3;
        Moteur M1, M2, M3;
```

Chaque clic sur un bouton agit directement sur le système grace aux accesseurs et mutateurs.
Puis nous en cliquant sur l'interface graphique du système nous actualisons la page en la redessinant l'interface du système de carburant.

```
class graphique : public QWidget
{
    Q_OBJECT
public:
    explicit graphique(QWidget *parent = nullptr);
protected:
    void paintEvent(QPaintEvent* e);
    void mousePressEvent(QMouseEvent *event);
};
```

```
class Widget : public QWidget
{
    Q_OBJECT

public:
    Widget(QWidget *parent = 0);
    ~Widget();
private slots:
    void VT12clic();
    ...
private:
    QPushButton* VT12;
    ...
```

Le pilote pourra s'exercer à grace aux propriétés, les pannes de pompes ou les vidanges manuelles ainsi que les exercices proposés.
Ensuite une note serra attribuer une fois les exercices finis.

Pour finir un systeme d'authentification entre Administrateur et Pilot aura lieu a chaque lancement de simulation.