

Projet de Programmation L2.1 : Simulation de gestion d'aéroport

Borges Daniel (dborges@etudiant.univ-mlv.fr)

Dinca Daniel (ddinca@etudiant.univ-mlv.fr)

5 janvier 2006

<p>Prochains atterrissages SWI545-A-----10-01</p>	<p>08:15 Nombre de pistes : 1</p> <p>Historique SWI223-A-----13-02 SWI112-D-0812----- TAP359-U-----01-02 AFR753-D-0811----- AFR414-D-0810----- AFR745-A-----03-01 AFR159-A-----07-02 JAL234-A-----03-01 AFR121-A-----07-02 KLM945-N-----45-05 SWI158-A-----06-02 SWI744-A-----11-03</p>
<p>Avions en bout de piste SWI336-D-0815-----</p> <p>Prochains décollages TAP127-D-0835----- SWI745-D-0900----- JAL001-D-0900----- AFR147-D-0901----- SWI654-D-0901----- AFR885-D-2200----- AFR111-D-0457-----</p>	<p>Continuer Ajouter un avion Charger un fichier Supprimer un avion Avion en urgence ! Ajout compagnie Ajout à Liste Noire Évènement aléatoire Status Historique Quitter</p>

1 Manuel d'utilisation

1.1 Fonctionnalités

Ce programme de simulation de gestion d'aéroports est capable de gérer :

- De multiples compagnies possédant plusieurs avions au décollage comme à l'atterrissage.
- Les priorités des avions en urgence sur les autres avions.
- Possibilité de mettre une compagnie sur liste noire ce qui a pour effet d'empêcher tous ses avions de décoller et fait immédiatement atterrir, sauf urgence, ses avions en vol.
- Historique des évènements.
- Gestion de plusieurs pistes d'atterrissages/décollage.
- Détection des crashes d'avions qui auraient tentés de rester en vol sans carburant.

1.2 Description de l'interface utilisateur

L'écran est divisé verticalement en deux. À gauche s'affichent les avions allant atterrir (en haut) et décoller (en bas). À droite s'affichent, de haut en bas : l'heure au format 24 heures, le nombre de pistes, l'historique des 12 derniers évènements dans le format d'affichage de `aeroport.log`, les boutons controlant les évènements.

1.3 Utilisation

L'utilisateur peut interagir avec le programme toutes les 5 minutes. Pour cela, il devra cliquer sur l'un des boutons du menu de droite puis remplir les informations demandées et en obtenir la sortie sur le terminal qui a lancé l'application.

Pour tester l'application, des fichiers de jeu d'essai sont disponibles dans `doc/jeu_essai/`.

1.4 Fichiers

1.4.1 Fichier de compagnie

Un fichier de compagnie est un fichier permettant d'ajouter des compagnies au lancement du programme, avec l'option `-c`. Ce fichier doit suivre une syntaxe précise.

Chaque ligne représente une compagnie, sauf les lignes commençant par `#` qui servent de commentaires dans le fichier. Chaque ligne de compagnie doit commencer par l'acronyme de la compagnie, 3 caractères de l'alphabet or accents et en majuscules, suivi d'un espace et du nom de la compagnie.

La lecture du fichier s'arrête si une erreur est rencontrée et la ligne ayant une erreur est affichée.

Exemple de fichier de compagnie :

```
# Compagnies françaises
AFR Air France
COR Corsair
ALN Air Linair
CCM CCM Airlines
# Compagnies étrangères
KLM KLM
JAL Japan Air Lines
TAP TAP Air Portugal
SWI Swiss
```

1.4.2 Fichier d'avions

Un fichier d'avions permet d'ajouter des avions pendant l'exécution du programme ou au démarrage en passant l'argument -a au programme. Ce fichier doit contenir un avion par ligne et peut contenir des commentaires si la ligne commence par #. L'ajout d'un avion se fait en suivant le format de l'historique et de `aeroport.log`.

Les 3 premiers caractères représentent l'acronyme de la compagnie, les 3 caractères suivant sont le numéro du vol. Un tiret est ensuite obligatoirement placé suivi de la lettre D ou A selon que l'avion doit décoller ou atterrir, puis un autre tiret, l'heure de décollage si décollage il y a, au format hhmm, l'heure étant au format 24 heures. Suit encore un tiret puis, dans le cas d'un avion qui va atterrir, la quantité de carburant restant dans l'appareil sur deux chiffres, suivi d'un tiret puis de la quantité de carburant consommée par minutes par l'appareil, toujours sur deux chiffres.

La lecture du fichier s'arrête si une erreur est rencontrée et la ligne ayant une erreur est annoncée.

Exemple de fichier d'avions :

```
# Décollages
KLM123-D-2000-----
TAP745-D-2030-----
AFR111-D-1900-----
CCM042-D-2100-----
# Atterrissages
JAL008-A-----20-02
SWI415-A-----30-05
COR774-A-----21-03
```

1.5 Options de la ligne de commande

Le programme possède plusieurs options de lancement. Pour afficher la liste de toutes les options, il suffit de lancer le programme avec l'option "-h" :

```
$ ./aeroport -h
Simulation de gestion d'aéroport
Options :
-a <nom fichier> : charge les avions contenu dans <nom fichier>
-c <nom fichier> : charge les compagnies contenues dans <nom fichier>
-h : affiche cette aide puis quitte
-n <nombre> : nombre de pistes de l'aéroport
-t <hh :mm> : change l'heure de début du programme
```

Attention, l'ordre de passage des arguments n'a pas de priorités, autrement dit, si un fichier d'avions est passé avant un fichier de compagnies, le fichier d'avions ne bénéficiera pas des noms de compagnies du fichier de compagnies. Aussi, si un temps de départ est spécifié alors que des avions ont déjà été chargés, il ne sera pas pris en compte.

1.6 Journal d'évènements

Après la fin du programme, un fichier aeroport.log est disponible contenant la totalité des évènements qui se sont déroulés pendant l'exécution du programme. Ce fichier est effacé au début du programme, il est donc conseillé de le sauvegarder sous un autre nom si c'est

2 Manuel du programmeur

2.1 Organisation des fichiers

Les fichiers de code C se trouvent dans le répertoire `src` et l'entête `aeroport.h` dans le répertoire `include`.

`aeroport.c` réalise les initialisations, applique les arguments de la ligne de commande, gère l'interaction avec l'utilisateur.

`affichage.c` contient toutes les fonctions d'affichage graphique avec la libMlv.

Dans `annexe.c` se trouvent les fonctions annexes du programme, c'est à dire les fonctions chargées de contrôler des données, d'obtenir des informations à partir des listes et de convertir des données.

`es.c` gère toutes les entrées sorties du programme : lecture et écriture de fichiers, détection de clics sur la fenêtre du programme, lecture et écriture du terminal pour l'interaction texte avec l'utilisateur.

`gestion.c` contient toutes les fonctions de gestion de l'aéroport : atterrissage en urgence ou non d'avions, crashes, décollages, etc.

`memoire.c` s'occupe des allocations et libération de mémoire pour les différentes listes chaînées du programme.

2.2 Boucle évènementielle

Après l'initialisation des variables et leur mise à jour suivant les arguments passés à la ligne de commande, une boucle principale est lancée qui ne s'arrête que quand l'utilisateur appuie sur le bouton Quitter du menu.

Cette boucle à 4 étapes :

- Affichage avec la fonction `affichage()`.
- Une pause de 0.1s pour que laisser un peu de temps pour voir l'évolution du programme à l'aide de la fonction `usleep()`.
- Si l'heure actuelle est multiple de 5, la fonction `evenement_utilisateur()` est appelée pour ajouter un évènement. La valeur de retour de cette fonction est utilisée pour savoir si la boucle principale du programme doit continuer ou non.
- `evenements()` est appelée pour calculer le prochain évènement et le réaliser s'il existe.
- Dernière étape, le temps, représenté par en minutes par un entier, est incrémenté sauf si sa valeur était supérieure à 1440 (soit 24 heures) auquel cas sa valeur repasse à 1

Toutes ces fonctions se trouvent dans `aeroport.c`, sauf `usleep()`.

2.3 Gestion des évènements

Le programme commence par mettre tous les avions dont l'heure de décollage est inférieure à $heure\ t + DELAI$ (défini dans `aeroport.h`) sur la piste. Pour cela, la fonction `gestion_piste()` est appelée, elle va placer les pointeurs de la structure `Queue` sur le premier et le dernier avion de la liste de décollage étant sur la piste.

Le programme va ensuite détecter les avions en attente d'atterrissage qui risquent de se

crasher en vérifiant que leur nombre de tours possible en vol, calculé par la quantité de carburant embarquée divisée par la consommation de l'appareil, est supérieur à 1. Si ce n'est pas le cas, l'avion est immédiatement placé en urgence. La fonction `detection_urgences()` s'occupe de réaliser cette tâche en écrivant dans le champ `heure_decollage` les caractères URGE, ce champ n'étant pas utilisé par l'avion puisqu'il ne va pas atterrir.

Les évènements s'enchaînent alors dans l'ordre suivant, pour chaque piste :

- Si un avion doit atterrir en urgence (son champ `heure_decollage` est égal à URGE), l'avion atterrit et la piste courante est déclarée comme utilisée. La fonction `atterrissage_urgence()` s'occupe de cette tâche.
- Si la piste courante n'est pas utilisée, un qu'un avion en attente de décollage est sur la piste, la fonction `decollage()` le fait décoller et met la piste comme utilisée.
- Si la piste courante n'est toujours pas utilisée, la fonction `atterrissage_noire()` fait atterrir le prochain avion d'une compagnie étant sur liste noire et dont un avion est encore en vol. Si tel est le cas la piste courante est marquée comme utilisée.
- Si aucun des évènements précédents ne s'est réalisé, alors la piste est libre pour faire atterrir un avion en attente d'atterrissage. La fonction `atterrissage()` s'en charge.

Ces priorités s'appliquent tant qu'une piste est libre et qu'il y a des évènements possibles. À la fin de ces évènements, la quantité de carburant des avions en vol est décrétementée par `gestion_carburant()` et s'ils n'ont plus de carburant, un crash est déclaré par la fonction `crash()`.

2.4 Liste noire

La liste noire est gérée par une liste simplement chaînée de pointeurs vers les structures des compagnies sur liste noire.

2.5 Historique

L'historique est géré par une liste simplement chaînée de tableaux de 20 caractères. Les 19 premiers caractères sont les informations sur un avion dans le format de `aeroport.log` (exemple : DLA121-D-1300-----) et le 20e est un caractère nul pour simplifier l'affichage de la liste sur le terminal.