Rapport du Projet d'Algorithmique 2007-2008
Gestion du trafic aérien d'un aéroport

Laura Liu
Rémy Tuyéras

10 janvier 2008

1 Présentation du Projet

Le projet consiste en la réalisation d'un logiciel de simulation de la gestion du trafic d'un aéroport. L'utilisateur peut agir sur la réalisation des évènements via des modules de création de compagnies et de mise en décollage ou atterrissage des avions, mais doit penser ses actions avec les règles de priorité régissant l'orchestration du trafic aérien.

Le logiciel comporte de même une interface graphique qui illustre la dynamique des échanges aériens ainsi qu'un module effectuant l'historique de tous les évènements déclarés dans un fichier de sauvegarde nommé `aeroport.log`.

Enfin, l'utilisateur a accès, via l'interface graphique, à un affichage ordonné des sept évènements à venir ainsi que des trois évènements effectivement réalisés pour chaque piste.

2 Description du Programme

Le programme est séparé explicitement en trois chapitres : un premier chapitre présente les différentes structures utilisées, un deuxième expose toutes les fonctions intermédiaires, et un troisième et dernier chapitre consiste en l'implémentation du *main*.

2.1 Présentation des structures du programme

Les Structures `Couple` et `Quadruplet` facilitent la manipulation d'objets graphiques et permettent une utilisation aisée des principes de la géométrie euclidienne.

La Structure `horloge` facilite le calcul de l'heure courante du jeu en différentiant le minutage, calculé modulo 60, et l'heure stricte, calculée sur un modulo 24.

La Structure `Decors` permet un affichage organisé du décors en comprenant une zone du tableau blanc comme un rectangle (*cadre*) pouvant être colorié d'une certaine couleur (*color*) et accueillir du texte (*nom*) défini par sa couleur (*encre*), sa police (*font*) et son emplacement dans cette zone (*centrage*).

La Structure `rangement` permet d'organiser l'affichage des avions de manière la plus réaliste possible en spécifiant un emplacement géométrique (*place*) et en précisant si celui-ci est occupé ou non (*etat*).

La Structure `avion` a subi des améliorations pour ne pas dire des modifications par rapport à la structure de base proposée. On peut distinguer deux types de champs : le premier est utile à la cohésion des échanges aériens, alors que le deuxième sert à l'affichage graphique de l'avion.

Un avion est en premier lieu affilié à une compagnie par un pointeur de type `Compagnie *`, et est défini par les champs renseignant sur sa capacité (*consommation*), sa dimension qualitative (*heure de décollage* et *identifiant*) et sa dimension quantitative (*carburant*).

Pour ce qui des champs de deuxième type, l'avions est repéré graphiquement par un couple de coordonnée définissant son emplacement courant (*centre*) ainsi que la direction dans laquelle il est orienté (*axe*). Par la suite le graphisme de l'avion sera géré en *coordonnées polaires* à partir de ce centre et de cet axe : ces derniers constitueront respectivement le centre et l'axe des ordonnées du repère de l'avion.

Le *mode* est un outil utilisé pour le coloriage du graphisme ainsi que pour informer de manière plus précise de la piste à laquelle est affilié l'avion (voir aussi les commentaires du programme).

Enfin, le champ *info* est une information comparable aux données fournies par l'historique. Il sert à l'affichage, au niveau de l'interface graphique, de toutes les informations concernant l'avion et est mis à jour en temps réel (carburant, état d'urgence etc.). Les champs *forme*, *ombre*, *grd* et *elgt* servent à calculer respectivement la forme de la carlingue (tableau de type `XPoint` utilisé avec `draw_filled_polygone`), la forme de l'ombre, le grandissement du graphisme et l'éloignement de l'ombre.

La Structure `compagnie` constitue l'objet informatique définissant une compagnie. Elle définit cette dernière surtout sur le plan qualitatif (*nom* et *acronyme*).

2.2 description des principaux algorithmes

On s'efforce de présenter dans cette sous-section les points clefs du programme en s'attardant davantage sur l'esprit plutôt que sur le détail informatique.

Gestion de la dynamique graphique Une certaine esthétique a été donnée au logiciel, d'un point de vue graphique, par l'intrégration de cinématiques sur les avions. Le fonctionnement de ces animations ne réside essentiellement que dans deux fonctions : `Affiche_Avion` et `Cinematique_Avion`.

`Affiche_Avion` Cette fonction calcule la forme de l'avion tout comme son ombre par de simples opérations de translations et de rotations respectivement par rapport à son *centre* et à son *axe*. On fournit alors à la fonction un tableau de coordonnées polaires permettant de calculer chaque point du polygone censé représenter l'image de l'avion. Ainsi, pour calculer un point du polygone, on applique dans un premier temps une rotation à l'*axe* de l'avion, puis on calcul quel devra être l'éloignement de ce point par rapport au *centre*, selon le nouvel axe obtenu après rotation. Ensuite, il ne reste plus qu'à afficher le polygone de l'ombre puis de la forme de l'avion avec un léger décalage (pour donner une impression de perspective) ainsi que les informations du champ *info* de l'avion au-dessus de son image.

`Cinématique_Avion` Cette fonction ne fait que réutiliser la fonction `Affiche_Avion` en donnant l'impression que l'avion tourne et avance par des affichages successifs (boucles) qui peuvent faire penser aux méthodes utilisées dans les dessins animés. On choisit d'utiliser la géométrie complexe pour le calcul du déterminant et du produit scalaire. Ces deux quantités fournissent des informations précieuses sur le sens des rotations.

Gestion de l'organisation graphique En plus d'afficher des avions, on veut pouvoir les ordonner de manière logique. Ainsi grâce aux fonctions de répartition `Ranger_Liste_Avions_A`, `Ranger_Liste_Avions_E` et `Ranger_Liste_Avions_Q`, on peut ranger les avions respectivement à l'atterrissage, à l'embarquement et en bout de piste de manière la plus réaliste possible. Sont listés, ci-dessus, les particularités de chaque fontions.

`Ranger_Liste_Avions_A` Cette fonction n'a pas présenté de grandes difficultés dans le sens où l'on a, à chaque fois, redistribué les places pour les huit premiers avions des listes d'attente en atterrissage.

`Ranger_Liste_Avions_E` Cette fonction doit pouvoir laisser en place tout avion n'ayant pas quitté l'embarquement et attribuer les places libérées aux prochains avions créés.

`Ranger_Liste_Avions_Q` Cette fonction doit quant à elle pouvoir faire avancer en bout piste tout avion venant d'entrer dans la queue afin d'occuper les premières places comme dans une véritable file d'attente.

Les fonctions de saisie et d'affichage de données Afin de ne pas casser le rythme de gestion de l'aéroport, on opte pour une saisie et un affichage des données par l'interface graphique. Au total, sept fonctions différentes de saisie ont été nécessaires pour pouvoir véhiculer tout les types d'information entre l'utilisateur et le programme. Toutes ont leur particularité et puisent leurs informations de manières différentes mais se construisent sur un même schéma :

- **Affichage de la consigne d'utilisation du module**
  Les fonctions commencent toutes par une invitation à parcourir une liste d'information via des boutons permettant d'évoluer dans celle-ci ;

- **Affichage de l'information**
  On transmet par la suite l'information à l'utilisateur en l'affichant par l'interface graphique (utilisation de la fonction `draw_string`). Cette information est, selon la fonction, puisée dans un fichier, une liste ou encore calculée par la fonction même. Il arrive que dans certaines fonctions, il faille traiter à part le cas d'une information vide (c'est le cas de `Voir_Etat_Avions_Compagnie` et `Voir_Historique`) ;

- **Implémentation des boutons de défilement**
  Enfin, il faut permettre à l'utilisateur de pouvoir parcourir l'ensemble des informations qu'il est autorisé à consulter à partir des boutons. Selon la nature de la source dans laquelle est puisée l'information (*fichier*, *liste chaînée*, *liste doublement chaînée*), il a fallu gérer l'incrémentation de diverses manières (pour ne pas se retrouver avec une lecture d'un pointeur `NULL` dans le cas des listes, par exemple).

L'affichage des évènements sur l'écran de contrôle Cette partie fut la plus difficile sur tous les points de vue :
- pouvoir afficher les informations de deux pistes différentes dans une seule et même chaîne ;
- pouvoir conserver l'information d'un évènement d'une première piste indépendamment des autres pistes, i.e. le défilement d'informations pour une piste ne doit pas occasionner le déplacement des informations des autres pistes, alors que les informations de chaque piste sont regroupées dans des chaînes communes ;
- se procurer l'information d'un évènement passé ;
- prévoir, dans la logique des priorités imposées, les évènements futurs probables ;

Le mode `normal` et le mode `test` Voici une innovation de notre part qui est justifiée pour diverses raisons. Tout d'abord, nous ne fournissons pas à l'examinateur un ensemble de fichiers tests, comme cela a été suggéré dans l'énoncé, mais une possibilité d'évaluer les performances du programme via un mode `test` incorporé dans la commande de lancement.

Cette initiative a été motivée par le fait que cela permettait une plus grande plage d'évaluation à l'examinateur. De plus, il aurait été voulu par l'énoncé que l'on puisse créer des avions en décollage ou à l'atterrissage à partir de fichiers, afin de faciliter l'évaluation de cas particuliers (crash, mise en liste noire... etc.). Cependant, cette possibilité n'était pas envisageable dans le sens où un avion doit obligatoirement être affilié à une compagnie qui est définie par son *acronyme* et son *nom*. Or la mise en forme des fichiers de sauvegarde était imposée et seule l'information de l'acronyme de la compagnie nous était fournie : information ne nous renseignant pas sur le nom propre de celle-ci, utile à sa définition.

Aussi, peut-on nous opposer le fait que la façon dont nous avons implémenté notre logiciel pouvait contourner ce problème en recherchant dans le fichier `LstComp.txt` le nom de la compagnie auquel a été associé l'acronyme. Cependant, un problème survenait : cette possibilité n'était envisageable qu'à la condition de traiter une multitude de cas, beaucoup trop nombreux, étant donné que le nom des compagnies sont des groupes nominaux d'éléments en nombre aléatoire. On peut d'ailleurs se rendre compte des cas qu'il aurait fallu traiter dans la fonction `Demande_Comp`.

Pour comprendre l'utilité du mode `test`, il faut avant tout comprendre celle du mode `normal`. Le mode `normal` impose à l'utilisateur de démarrer le simulateur sans aucun avion sur la piste ni dans le ciel. Il devra alors pour commencer appuyer sur le bouton `DEBUTER`. Le mode `test` permet, quant à lui, une création préalable d'avions et de compagnies avant le lancement du logiciel (via le bouton `DEBUTER`), autorisant à mettre en scène n'importe quelle situation souhaitée par l'utilisateur. Ainsi ce n'est pas un ensemble limité de tests que nous proposons, mais bien un nombre indéterminé, laissé à l'imagination de l'utilisateur.

3 Problèmes et difficultés rencontrés

Excepté le fait que l'environnement linux utilisé dans les salles de TP ne reconnaissait pas le caractère `'\n'` au même titre qu'un caractère quelconque - ce qui nous a forcé à utiliser le caractère de reconnaissance de fin de ligne `'/'` - nous avons à quelques détails près abouti au résultat escompté.

Par ailleurs on peut souligner le fait que la réalisation des graphismes générait beaucoup d'informations pour la variable du tableau blanc (`MlvType`),

c'est pourquoi nous avons systématiquement alloué dynamiquement les variables de ce type, afin de pouvoir les libérer de leur contenu à des moments clefs de l'évolution du programme (par exemple, à chaque fois que l'utilisateur entre dans un module de création etc.).

4 Conclusion

Quand bien même n'avons nous pu réaliser ce que l'énoncé semblait sous-entendre (après les précisions du chargé de TP) au niveau de la création d'avion à partir de fichiers de sauvegarde, nous proposons une gestion graphique des données et de la dynamique du trafic permettant à l'utilisateur de mieux appréhender l'évolution des évènements, ce qui n'était pas forcément le cas pour une programmation en mode texte. Aussi fournissons-nous à l'utilisateur deux options complémentaires (`normal` et `test`) permettant de réaliser une grande variété de situations. Enfin, en perspective d'améliorations, on pourrait penser à une création aléatoire d'évènements indépendante de l'utilisateur, ce qui rendrait la simulation peut-être davantage réaliste.