République Algérienne Démocratique et Populaire Ministère de l'enseignement Supérieur et de la Recherche Scientifique Université des Sciences et de la Technologie Houari Boumediene

Rapport du projet de Programmation réseau

Nom et Prenom :

BAROUD YASMINE

DJEKOUNE IMAD EDDINE

SOUIDI MOHAMED AMINE

SOUAYEH ABDELKADER

Table des matières

Table des figures

1 Introduction

La gestion de l'aéroport comprend toutes les activités nécessaires pour garantir le bon fonctionnement de l'aéroport, telles que la coordination des déplacements des voyageurs dans le monde entier, la gestion des passages, le service d'import/export, la sécurité et la planification des vols, etc. Notre travail se concentrera sur la gestion de plannification des vols.

Dans ce rapport, nous allons décrit les étapes de développement de notre application Client/Serveur basée sur un système réparti pour permettre la communication et l'échange d'information entre les avions(clients) et le tour de controle(serveur) du système. Nous allons examiner les méthodes mises en place pour gérer l'acheminement et de la surveillance des mouvements des avions à l'intérieur de l'espace aérien de l'aéroport, ainsi que les amélioration opté pour un meilleur résultat de fonctionnement.

2 Modélisation

Les principaux acteurs de notre application sont : Client et Serveur. Le client peut étre un seul client ou plusieur.

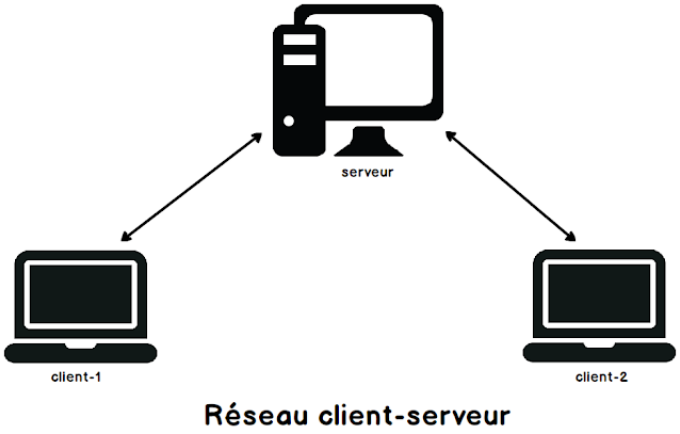

FIGURE 1 – Exemple d'un réseau de communication entre 2 client et un serveur.

Client : Le client réprésente l'avion. Son role consiste à :

— Créer l'avion en spécifiant sa référence(modèle), la capacité de son réservoir et aussi son état. La création de l'avion peut se faire en deux maniere :
  — Saisir directement les attributs de l'avion sans visualiser son modèle en 3D.
  — Création de l'avion en choisissant un modèle parmi ceux proposés dans la liste des modèles de l'application et le visualiser en 3D, saisir le reste des attributs.
— Consulter la liste des vols.
— Visualiser le vol des avions qui les a crée en temps réel.

Serveur : Le serveur représente le tour de controle de l'application. Il est chargé de gerer l'enregistrement des avions, programmation des vols ainsi que l'acheminement des avions tout au long le trajet et ce afin d'assurer le bon fonctionnement de l'aéroport.

1. Enregistrement des avions
   — A chaque arrivé d'un avion, ce dernier doit s'enregistrer dans un annuaire gérer par le tour de controle. Ce dernier doit connaitre à temps réel l'état de chaque avion qui peut prendre les valeurs suivantes :
     — **Active** : l'avion en trajet.
     — **Standby** : l'avion se prépare pour un vol.
     — **Idel** : l'avion est en repos.
     — **New** : une nouvelle entité qui a besoin d'un numéro de référence.
     — **Broken** : l'avion est en panne.
   — Modifier ou supprimer les avions créés par le client.
2. Programmation des vols
   — Programmer un Vol. Il est défini par :
     — L'heure de départ et d'arrivée.

- Le numéro du vol.
- La référence de l'avion.
- L'aéroport de départ et d'arrivé.
- Le trajet à suivre (les différentes stations à visiter).
- Type de vol : directe ou avec escale.
- Vitesse.
- Un identifiant(Id) unique du vol.

3. L'acheminement des avions
   - L'acheminement des avions est assuré via un échange des messages périodiques entre l'avion (client) et le tour de controle (serveur). Le tour de controle utilise un radar pour connaitre la position et l'état de ses avions en service en temps réel. Ce dernier affiche en temps réel l'état des vols sous différents couleur selon leur situation comme suit :
     - Un vol dans état normal : l'avion est représenté par un avion vert.
     - Un Vol en catastrophe : l'avion est représenté par un avion rouge. On peut distinguer deux cas :
       - Le réservoir n'est pas suffisant et le tour de controle n'a pas pu trouver une station proche pour atterrir l'avion.
       - Deux avions entre en collisions.
     - Un vol en danger c'est a dire l'avion est représenté par un avion jaune. On peut distinguer deux cas :
       - Le réservoir n'est pas suffisant : dans ce cas, le tour de controle va consulter la liste des aéroports triés de plus proche aéroport au plus loin aéroport par rapport a la position réel de l'avion pour ganger du temsp et du carburant. il va reagir en verifiant certains conditions afin de réacheminer l'avion vers l'aéroport choisie pour remplir le réservoir et continuer son chemin vers la station d'arriver.
       - Deux avions se rapprochent : dans ce cas, chaque avion est en contacte avec le tour de controle, ce dernier prend la responsabilité de decider selon la situation des avions la méthode appliqué si l'avion sera reacheminer ou changer sa vitesse pour eviter la collision. nous allons detaillé ces deux situations dans la section suivante.

Station : Une station représente un aéroport de départ, d'arrivé ou un aéroport intermédiaire. Les stations sont définies exclusivement par le tour de controle. Un avion n'a pas le droit d'atterrir dans une station sans l'accord du tour de controle. Chaque station possède une capacité bien définie en termes d'avion et carburant.

3 Conception

Dans cette partie, nous alons présenté une étude conceptuelle menée pour le développement de notre application qui se base sur une approche réel afin de garantir un fonctionnement efficace de l'aéroport.

3.1 Diagramme de cas d'utilisation

Celui-ci est utilisé afin de donner une vision globale du comportement fonctionnel du l'application en spécifiant les fonctionnalités principales et les interactions entre les clients et le tour de controle (serveur). La figure suivante illustre le diagramme de cas d'utilisation de l'application.

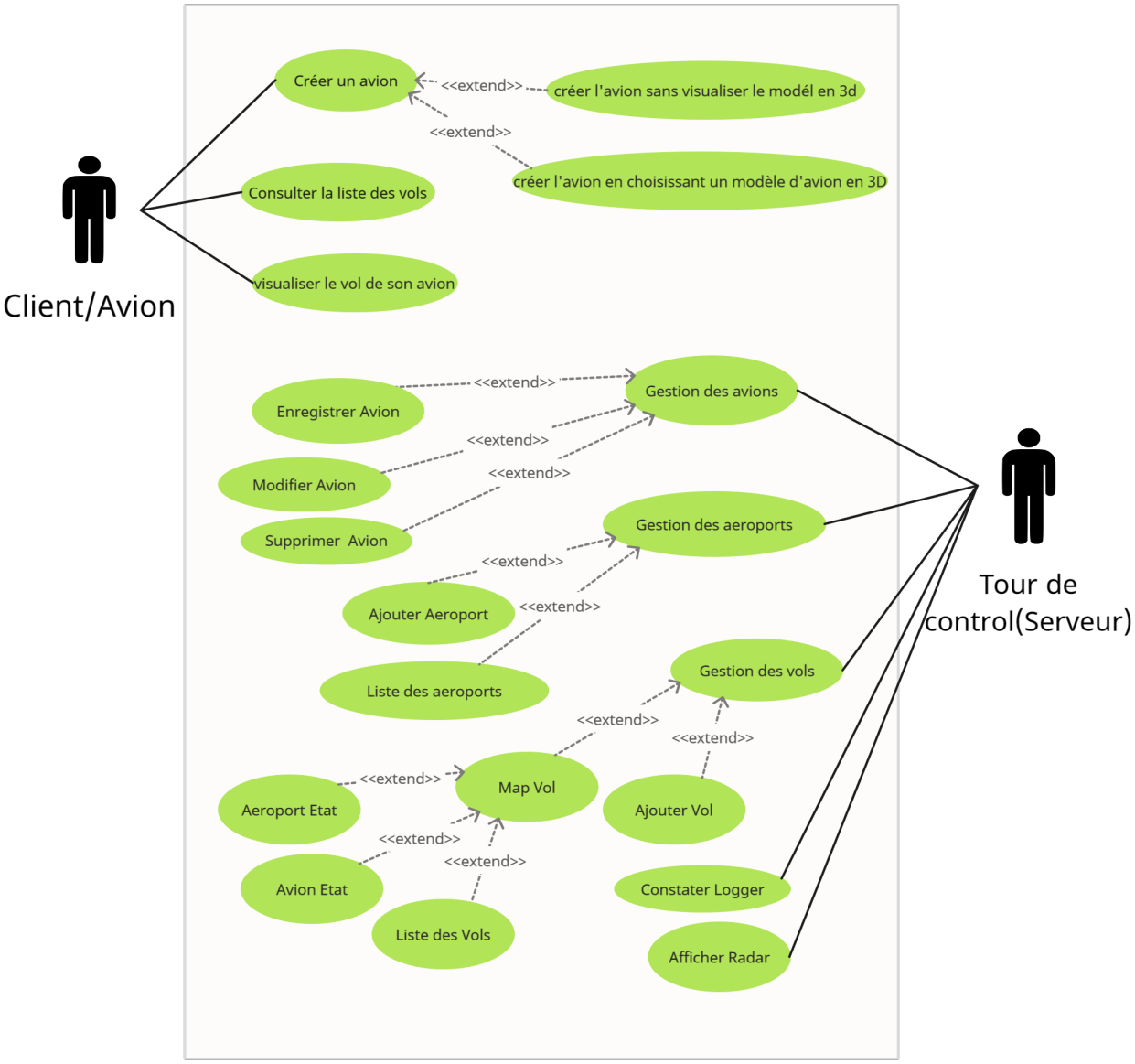

FIGURE 2 – Diagramme de cas d'utilisation des deux acteurs client/serveur.

3.2 Diagramme de classe

Le diagramme de classes est une vue statique de la structure interne de l'application, des éléments qui le composent et leurs relations. La figure suivante montre le diagramme de classes de l'application.

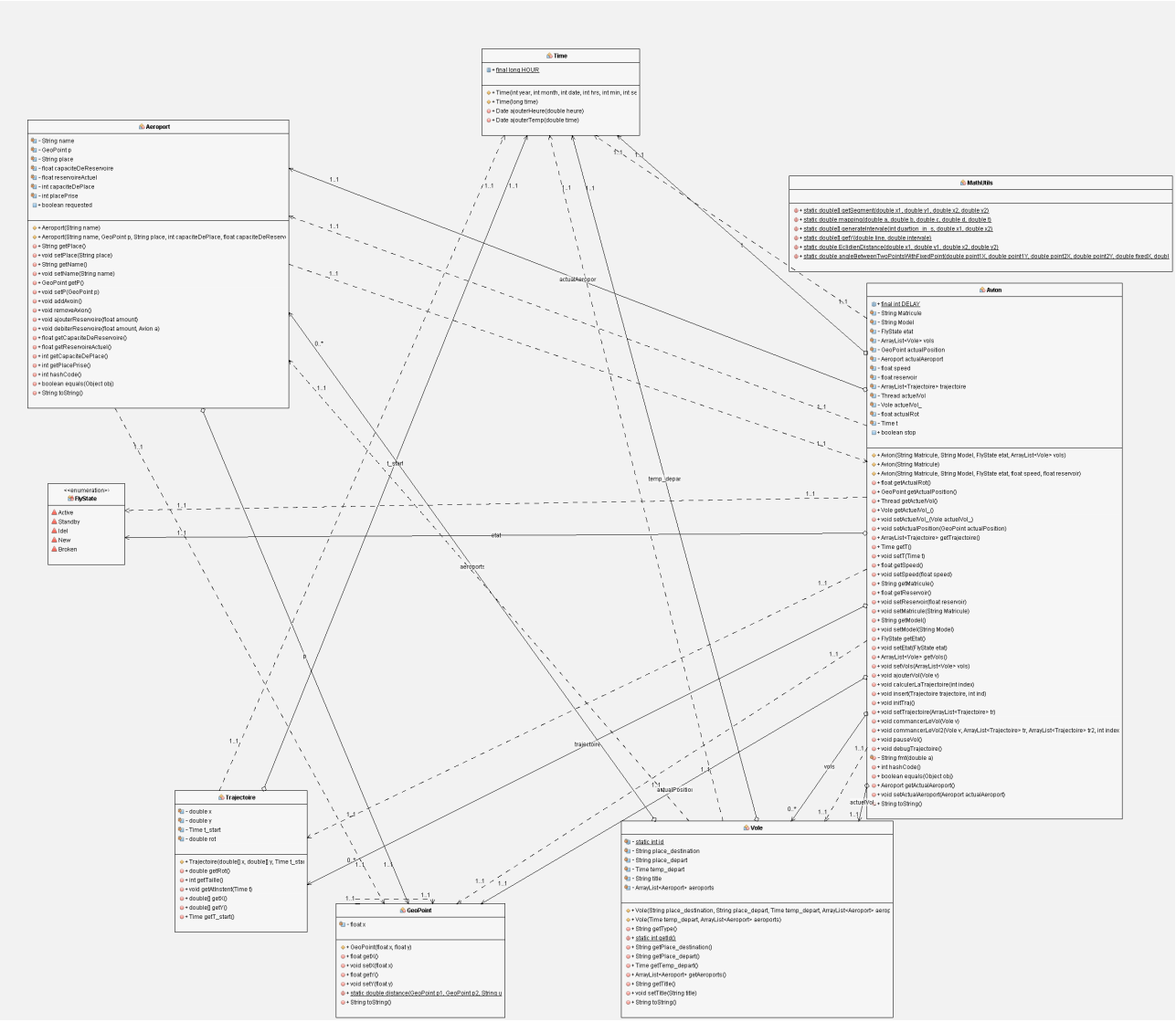

FIGURE 3 – Diagramme de classe de l'application

3.3 Structure de Données

Comme structure de données, nous avons utilisé le type ArrayList partagable avec le client et le serveur pour les raisons suivant :

- Les ArrayList sont des structures de données dynamiques, ce qui signifie qu'elles peuvent facilement s'adapter à des modifications de taille pendant l'exécution du programme. Cela permet de facilement ajouter ou supprimer des éléments sans avoir à redimensionner manuellement la structure de données.
- Accès direct aux éléments : Les ArrayList permettent un accès direct aux éléments en utilisant des index, ce qui permet de récupérer ou de mettre à jour rapidement un élément spécifique.
- Intégration facile avec d'autres classes Java : Les ArrayList étendent la classe AbstractList, qui fournit une implémentation commune pour les classes de listes en Java. Cela permet de facilement utiliser des ArrayList avec d'autres classes java.
- Mémoire efficace : Les ArrayList utilisent un tableau interne pour stocker les données, ce qui permet de conserver de la mémoire en comparant à d'autres structures de données.
- En général, les ArrayList sont une option pratique pour les projets Java qui nécessitent des structures de données flexibles et efficaces pour gérer des collections d'objets et l'interface graphique s'adapte da maniére efficace au changement de cette liste.

3.4 Solution pour la résolution des collision

Pour assurer la sécurité des avions, nous avons proposé une idée dont le principe est que le controleur doit planifié le trafic pour qu'il soit distant 10 minute chacun. C'est a dire, quand un avion est affronché un point A, le prochaine avion ne peut pas passé ce point avant 10min, si je garde la vitesse de l'avion, ils vont garder l'espacement jusqu'au bout de trajectoire. ceci deminuer le taux du risque que les avions tombent en collision.

Aussi, nous avons examiner deux situations possible que les avions peuvent se rencontrer au sein du vol.

situtaion 1 : Deux avions sur le méme trajectoire mais dans le sense inverse

Pour eviter ce problème, le tour de controle a la responsabilité de choisir quel avion sera réacheminer un petit peu sur son trajectoire et quand l'autre avion passe et il assure qu'il n ya pas d'autre collision, l'avion reviens a son trajectoire normal.

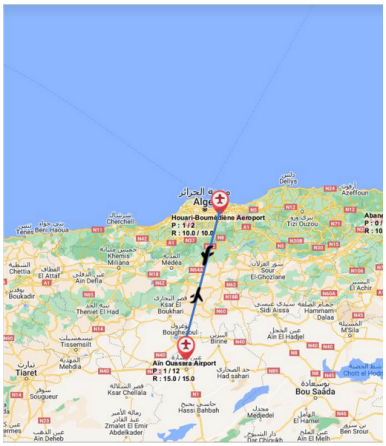

FIGURE 4 – Deux avions sur la méme trajectoire dans le sens inverse.

situtaion 2 : Deux Avions passent sur le même point (x,y) et en méme temps

si une avion A et une avion B seront rencontré sur le même point (x,y) et en même temps, mais ils ont des trajectoires différents, dans ce cas, le tour de controle va chosir l'avion qui va augmenter sa vitesse et l'autre avion qui va ralentir sa vitesse afin de ne pas se rencontrer sur le même point dans le même temps.

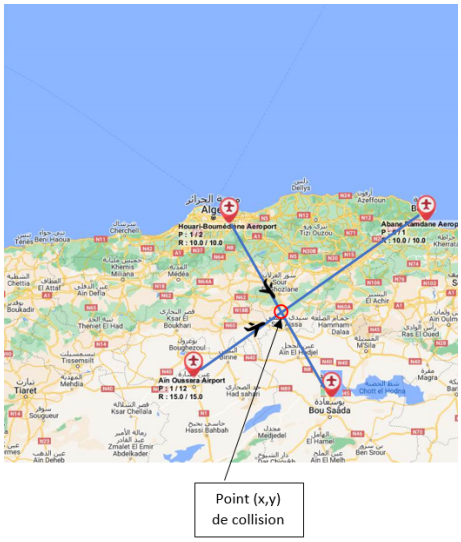

FIGURE 5 – Deux avions avec des trajectoire différents passent sur le même point (x,y) et en méme temps.

Implementation

Algorithm 1: Collision(avion A, avion B)

```
if collision then
    if (situation == situation 1) then
        reacheminer_Avion(A);
        collision=false;
    end
    if (situation == situation 2) then
        Augmenter_vitesse(A);
        Ralentir_vitesse(B);
        collision=false;
    end
end
```

3.5 Simulation

Chaque avion et un vol est caractérisé par ses attributs, ses propres informations et à partir de ces informations, nous pouvons faire la simulation du traffic des avions. Nous allons présenté dans cette partie la méthode qu'on a utilisé pour récupérer les positions des avions en temps réel. Nous avons utilisé une map pour donner un aspect réel à l'application.

1. L'ajout de la map
   L'ajout du map est fait en 3 etapes :
   (a) étape1 : télécharger la map puis sauvegarder la map dans une image sous format png.
   (b) étape2 : lire l'image dans le programme. Nous avons dévisé l'image a un ensemble des grid, chaque grid contient 4 fragement de 1024×1024. Ceci afin de faciliter le deplacemement la map et eliminer le cas ou la map bouge lentement, par exemple, si l'utilisateur déplace une région de la map vers la gauche, l'image qui se trouve à gauche est contenante dans la grid, donc, elle sera affiché directement.
   (c) étape 3 : Transformer les coordonnées des positions de latitude, longitude vers x,y et vice versa.
2. Récuperer les coordonnées des positions réels des avions
   (a) calculer le temp d'arrivé
   on sais que :

$$v = d/t \tag{1}$$

   où :
   v : la vitesse.
   d : Distance (la distance c'est la distance geodesic).
   t : Temps.
   d'où

$$t = d/v \tag{2}$$

   donc

$$TempPris = distance(départ, arrivee)/v \tag{3}$$

   et on a

$$temp\_d'arriver = temp\_de\_depart + TempPris \tag{4}$$

   (b) calculer la trajectoire que lavion doit suivre
   Si une avion part d'un point A(x,y) vers le point B(x,y), elle fera une trajectoire linéare qu'on peut la calculer comme suit :

$$Trajectoire(x) = a * x + b \tag{5}$$

où

$$a = (B.y - A.y/B.x - A.x) \tag{6}$$

et

$$b = B.y - a * B.x \tag{7}$$

Pour avoir la trajectoire a chaque instant t,il suffit juste de faire un changement d'espace comme suit :
changement d'éspace (Mapping) [A.x , A.y] –> [t_départ , t_arriver]
On obtiendra la trajectoire(t), qui donne a chaque instant t la position de l'avion.

(c) transformations des coordonnées latitude et longitude vers les coordonnées x et y
pour faire la tranfsormation, nous avons appliqué les formules suivantes :

$$x = \left\lfloor \frac{256}{2\pi} 2^{\text{zoom level}} (\lambda + \pi) \right\rfloor \text{ pixels}$$

$$y = \left\lfloor \frac{256}{2\pi} 2^{\text{zoom level}} \left(\pi - \ln\left[\tan\left(\frac{\pi}{4} + \frac{\varphi}{2}\right)\right]\right) \right\rfloor \text{ pixels}$$

où :
λ : longitude en radians.
φ : geodetic latitude en radians.

3.6 Solution de réservoir insufisant

Chaque avion posséde une quantité de réservoir, et cette quantité sera diminué tout au long du trajet. Si l'avion a atteint un seuil = 20% de réservoir(carburant) réstant, dans ce cas, l'algorithme qui permet de résoudre le probléme du réservoir insuffisant sera déclenché. L'idée du base de cet algorithme est comme suit :
Vérifer si le réservoir réstant est siffisant pour arriver à la déstination ou non.
Si oui, alors continuer le vol jusqu'a la station d'arrivé et remplir le réservoir dans cette station.
Si non, Calculer la distance géodésique de la position d'avion où elle a atteint le seuil minimale avec tout les aéroports qui se trouvent dans la liste des aéroports.
Trier les aéroports de plus proche au plus loin et chercher les aéroports les plus proches en vérifiant les conditions suivants :
Vérifier si l'aéroport de la liste trié contient du reservoir ou non, si oui alors vérifier s'il posséde une place libre pour l'avion ou non
S'il ne posséde pas de place libre, vérifier le deuxième aéroport s'il contient du carburant.
Si oui, vérifié s'il a de place libre pour l'avion, si oui alors l'avion sera réacheminer vers cet aéroport.
Sinon, si les deux aéroports ne posséde pas du place libre, réacheminer l'avion vers le plus proche aéroport et faire un tour de cercle en attendant une place se libére en activant l'algorithme de détection des collision et ce afin d'éviter que deux avion tombent en collision s'ils tournent sur la même trajectoire circulaire. Tout ça sera accompagné par l'affichage de l'information pour liberer une place, s'il ya de collision ou pas, ainsi que l'affichage des aéroports avec ses informations.

4 Présentation de la méthode de communication utilisé

Dans le cadre de notre projet de gestion de vols dans un aéroport, nous avons utilisé la méthode de communication RMI (Remote Method Invocation) en Java pour établir une communication entre le serveur (représentant le tour de contrôle) et les clients (représentant les avions). RMI est une technologie Java qui permet aux programmes Java d'appeler des méthodes sur des objets qui sont situés sur des machines distantes. Il utilise des protocoles de communication standard pour transmettre les données et les messages entre les différentes machines. En utilisant RMI, nous avons pu définir des interfaces pour les objets sur le serveur qui décrivent les méthodes qui peuvent être appelées à distance. Les clients peuvent alors obtenir une référence à ces objets à distance via un registre RMI et les utiliser comme s'ils étaient des objets locaux. Cela a permis une communication efficace entre le serveur et les clients, où les avions pouvaient enregistrer leur état, planifier des vols et informer le serveur de leur position en temps réel. En utilisant RMI, nous avons pu gérer les problèmes de fiabilité en utilisant des mécanismes de gestion des exceptions pour gérer les erreurs de communication. Enfin, RMI a permis une scalabilité efficace en permettant l'ajout de nouveaux clients sans perturber le fonctionnement du système existant. En somme, RMI est un outil puissant pour la communication distante entre les objets Java, et il a été un choix efficace pour notre projet de gestion de vols dans un aéroport

5 Les Outils de programmation utilisé

Dans notre projet de gestion de vols dans un aéroport, nous avons utilisé le langage java comme langage de programmation. Nous avons utilisé aussi la bibliothèque LWJGL (Lightweight Java Game Library) pour afficher les avions en 3D. LWJGL est une bibliothèque open-source qui fournit des fonctionnalités de bas niveau pour les développeurs de jeux et d'applications graphiques en Java. Elle permet l'accès aux fonctionnalités de bas niveau de la carte graphique, telles que l'accélération matérielle, les shaders, les textures, les modèles 3D, etc. En utilisant LWJGL, nous avons pu créer une visualisation 3D des avions qui était non seulement réaliste, mais également interactive. Nous avons utilisé LWJGL pour afficher les avions en 3D en utilisant des modèles 3D, des textures et des animations, et pour gérer les interactions avec les avions via la souris et le clavier. Cela a permis une meilleure compréhension de l'état et de la position des avions pour l'utilisateur. En utilisant LWJGL, nous avons également pu rendre notre projet plus attrayant en termes de visuels, en offrant une meilleure expérience utilisateur. En utilisant les capacités graphiques de LWJGL, nous avons pu créer une visualisation 3D de l'état des vols qui est facile à comprendre, interactive et attrayante. En somme, LWJGL était un choix efficace pour notre projet de gestion de vols dans un aéroport en raison de sa flexibilité, sa puissance et sa facilité d'utilisation pour les développeurs.

6 Implémentation

Dans cette partie, nous allons présenter les dofférentes interface de notre application de gestion de vols dans un aéroport. Nous allons utilisé deux interface graphique différents. Une pour le client et une pour le serveur.

FIGURE 6 – Interphace graphique de la création d'avion.

FIGURE 7 – Interface graphique de consultation des avions crée par le client.

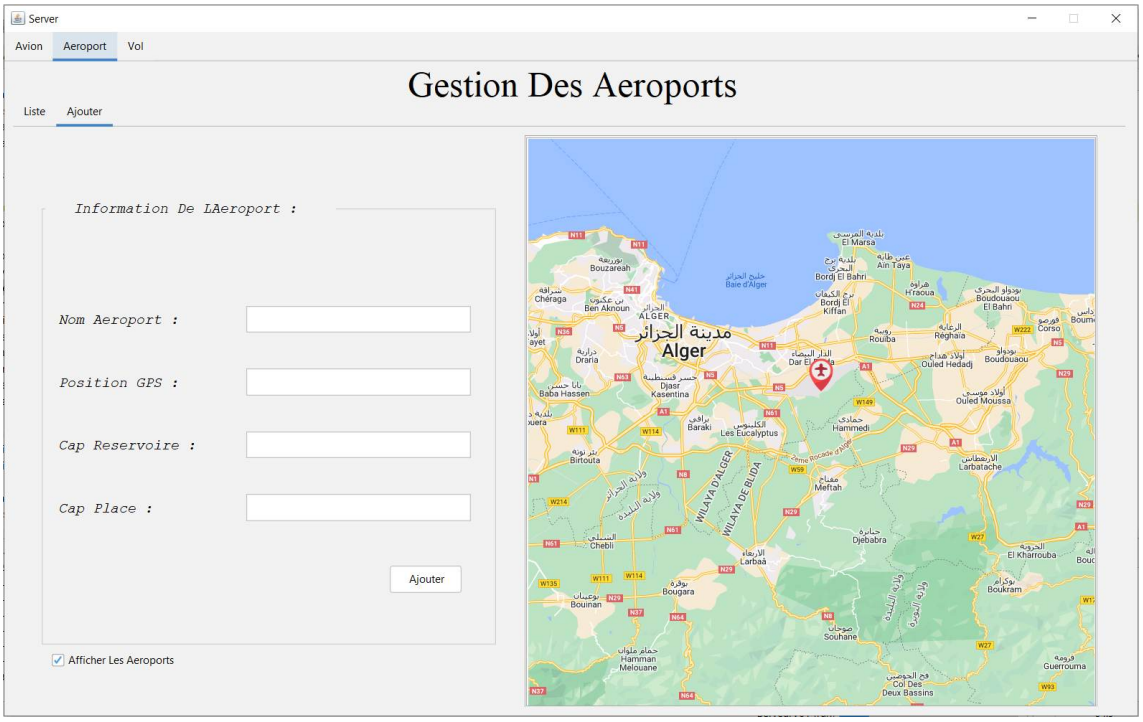

FIGURE 8 – Interface graphique de l'ajout d'un aéroport.

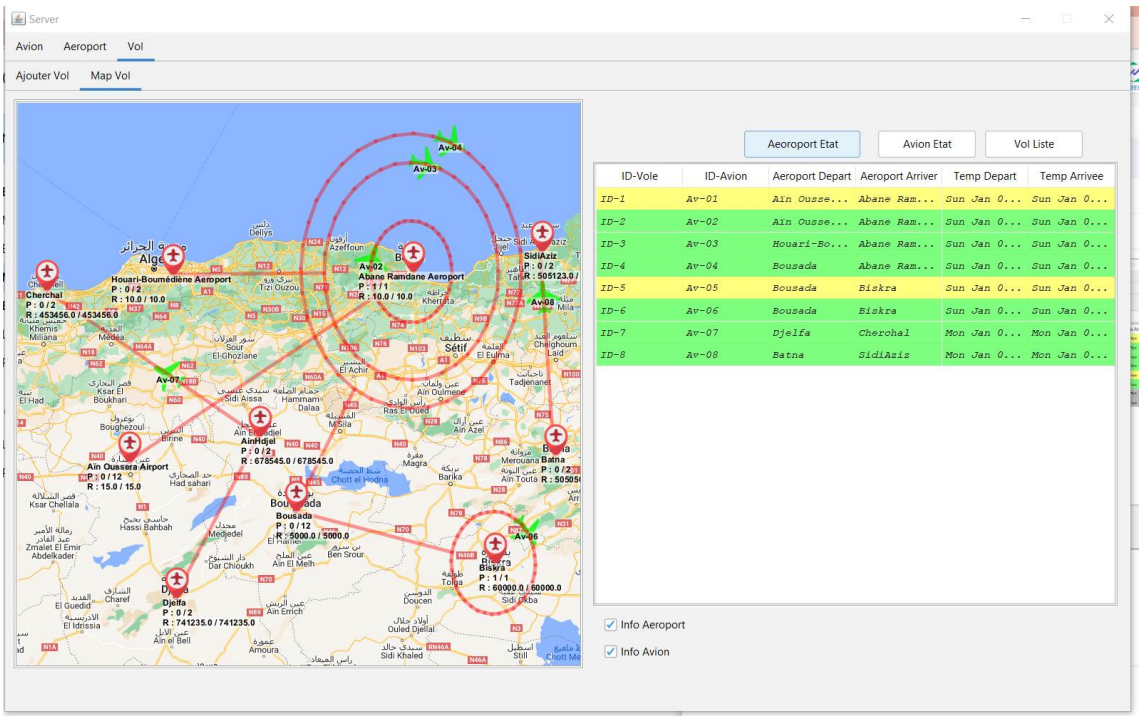

FIGURE 9 – Interface graphique de gestion des vols.

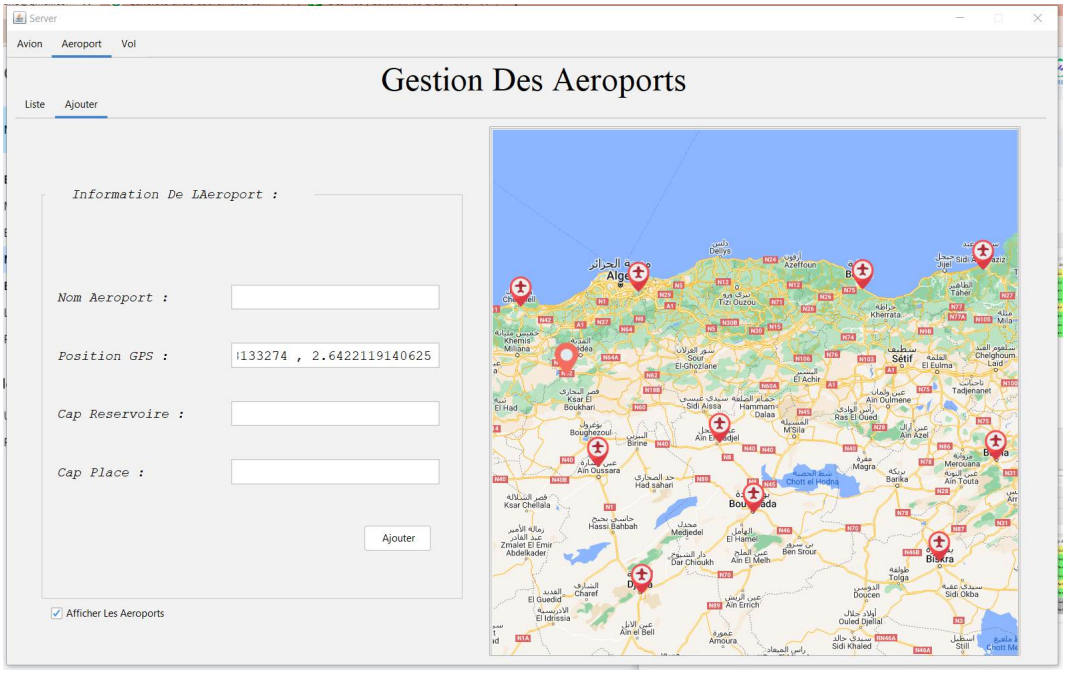

FIGURE 10 – Interface graphique de gestion des aéroports.

7 Conclusion

En conclusion, nous avons réussi à fournir une solution efficace pour la planification et la coordination des mouvements des avions à l'aéroport. En utilisant les fonctionnalités de Java, telles que les ArrayList, et en s'appuyant sur les normes et les règlementations de l'aviation, le système a permis une gestion efficace des informations de vol, une coordination précise des mouvements des avions, et une amélioration de la sécurité pour les vols des avions pour eviter les collisons. En utilisant des algorithmes de planification de vols, des systèmes et des outils de communication de RMI, le projet a réussi à améliorer l'efficacité de la gestion des vols de l'aéroport. En fin, ce projet a démontré l'importance de la technologie pour améliorer les processus de gestion d'un aéroport et la nécessité de collaboration entre les différents acteurs pour garantir un fonctionnement optimal de l'aéroport.