Rapport projet de simulation d'aéroport

Chuzel Philippe Duverger David Le Gallic Maël

28 janvier 2016

Table des matières

Chapitre 1

Objectifs de l'étude

La présente étude a pour objet de dimensionner de façon optimale de futures évolutions à l'aéroport de Brest. L'objectif de ce travail va donc consister à simuler un aéroport et voir, en fonction des différents paramètres de la simulation, si la configuration actuelle permet un fonctionnement acceptable.

La liste des différents paramètres est placé dans un fichier texte qui peut être modifiable afin de tester différentes configurations et le résultat sera placer dans un fichier excel.

Nous allons donc faire varier les paramètres suivants :

— Une ou deux pistes présentes sur l'aéroport.

— 4,6 ou 8 gateWays pour les avions.

— 2 ou 4 taxiways selon si on a une ou deux pistes

Plusieurs autres paramètres comme la météo, les jours de Weekend, les périodes de pointes seront également pris en compte.

Chapitre 2

Analyse du problème

La modélisation qui nous est ici imposer est un modèle de simulation évènementielle et donc nous avons donc implémenté l'aéroport de telle sorte que chaque étape de l'atterrissage et du décollage soit représenté comme un évènement.

Il y aura donc les évènement suivant :
— ArriverAvionEvent.
— Ouverture.
— Fermeture.
— DemandeAttEvent.
— ApprocheEvent.
— AtterrissageEvent.
— RoulementArrEvent.
— LiberationPisteEvent.
— DemandeGateWayEvent.
— DebarquementEvent.
— LiberationTaxiWayInEvent.
— EmbarquementEvent.
— DemandeTaxiWayOutEvent.
— RoulementDepEvent.
— LiberationGateWayEvent.
— DemandeDecEvent.
— DecollageEvent.
— FinDecollageEvent.

Pour certaines étapes, l'avion va soit demander l'accès à des ressources de l'aéroport ou va en libérer et ces évènement sont vitaux pour permettre la régulation de l'aéroport.

Le cahier des charges impose beaucoup d'éléments sur la simulation mais un point reste soumis à discution, à quels endroits un avion peut attendre ? Nous avons décider par conséquent de faire une simulation avec les points d'attente suivant :

— Un avion peut attendre une première fois au moment de l'atterrissage si la piste ou le taxiWay d'entrée sont indisponibles. Il peut être donc stocker ici dans une première liste d'attente.
— Dans le cas où aucune gateWay est disponible, l'avion peut attendre sur un taxiWay et être stocker dans une seconde liste.
— Si l'avion souhaite aller sur le taxiWay de sortie, il peut également attendre dans un gateWay.
— Dans le cas symétrique, si un avion souhaite décoller et que la piste n'est pas disponible, on peut attendre sur le taxiWay de sortie.

Ainsi, il ne reste que à voir comment représenter les entités et les classes qui vont être utilisées par la suite dans la simulation.

Nous avons donc modélisé les entités suivantes :
— Une entité avion.
— Une entité taxiWayIn pour l'entrée des avions.
— Une entité taxiWayOut pour la sortie des avions.
— Une entité gateWay.
— Une entité piste.

Enfin, nous avons créer deux classes :

— Une classe aeroMonitor qui va lancer toute la simulation et va fixer les différents paramètres de l'aéroport, sa date de départ et sa date de fin.
— Une classe aeroEngine qui va contenir toutes les listes d'entité et d'évènements et les différentes fonctions qui régisse le fonctionnement de l'aéroport.

Chapitre 3

Modélisation du système

Lors de la réalisation des différents projets de cette U.V., nous avions implémenter plusieurs classes génériques à partir desquelles nous avons réaliser une structure d'héritage et de polymorphisme afin de ne pas avoir à refaire ce travail par la suite. Ces classes sont ISimEngine, ISimEvent, ISimEntity, ISimMonitor, basicIrecordable et basicLogicalDateTime.

Ainsi, nous aboutissons à un ensemble de classe dont une représentation est donnée ci-dessous.

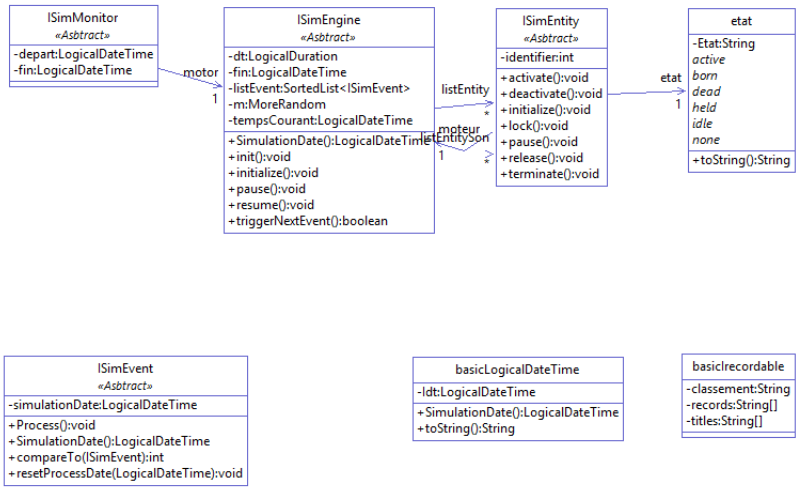


FIGURE 3.1 – Ensemble des classes génériques utilisées pour la simulation de l'aéroport.

La classe ISimEntity dispose d'un entier "identifiant" qui est très utile pour pouvoir dissocier deux entités dans la simulation. On indique également dans quel moteur cette entité travaille et en fin elle a un état qui permet de savoir si elle est utilisable lorsqu'un évènement essaie d'accéder à une ressource de la simulation.

La classe ISimMonitor initialise et instancie toutes les entités et le moteur de la simulation. C'est également ici que l'on donne la date de départ et de fin de la simulation.

La classe ISimEvent est la classe qui va permettre la création des évènements. La méthode process() est la méthode qui sera utilisée une fois que l'évènement se produira à la date simulationDate. La fonction compareTo() permet de comparer la date d'exécution de plusieurs éléments et est utilisé afin de trier la liste d'évènement du moteur de simulation.

La classe ISimEngine permet la création du moteur de simulation. Le moteur instancie un objet de génération de nombre aléatoire, une date de fin de simulation, le temps courant de la simulation et une sortedList qui contient l'intégralité des évènements de la simulation. Il y a deux méthodes principales :

— La méthode init() qui va initialiser toute les variables de la simulation et qui doit être redéfinie pour chaque modèle de simulation.
— La méthode triggerNextEvent() qui va chercher l'évènement le pus récent, changer le temps courant de la simulation et lancé la méthode process() de cet évènement.

Enfin, les deux classes basicIrecordable et basicLogicalDateTime permettent l'utilisation du logger afin d'écrire dans le fichier excel.

3.1 Quelles entités faut-il implémenter ?

Dans cette partie nous allons expliquer comment nous avons implémenté nos entités et pourquoi nous avons décidé de placer certains attributs à ces objet.

3.1.1 AeroportEntity

Cette classe hérite de tous les attributs de la classe ISimEntity qui permet d'établir l'état de l'entité (savoir si elle est active ou utilisé par un autre évènement à un instant t) et lui donne un identifiant unique. Nous avons créer cette seconde classe afin de créer un attribut classeIntAero qui permet de différencier toutes les entités entre elles.

Avec cette outil, on peut très facilement travailler sur ces entités pendant toute la simulation.

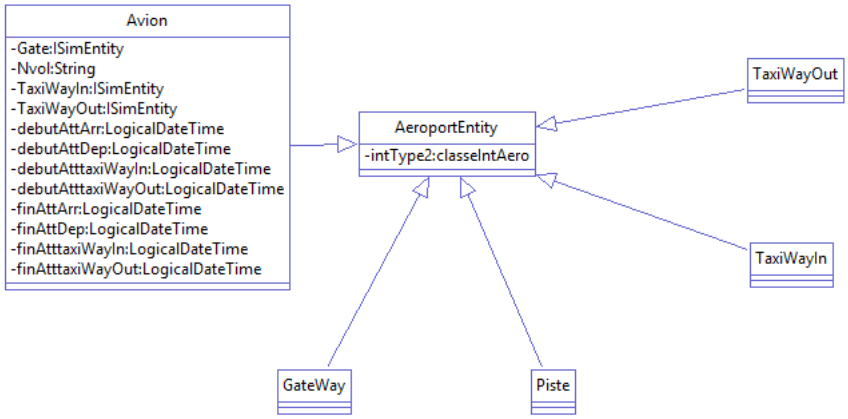


FIGURE 3.2 – Ensemble de la structure d'héritage pour les différentes entités de l'aéroport.

3.1.2 Avion

La classe Avion est la seule entité à qui nous avons dû donner d'autres attributs pour faire fonctionner la simulation. Nous devons savoir quand l'avion commence et fini d'attendre à chaque étape du processus et nous devons également savoir quel gateWay et quels taxiWays il va occuper/libérer afin de pourvoir libérer ces entités une fois qu'elle n'est plus utiliser par l'avion. De plus, le cahier des charges nous oblige de donner un numéro de vol unique à chaque avion.

Au final, nous avons donner les attributs suivant à cette classe :

— Une chaine de caractère qui va correspondre au numéro de vol.
— des entité Gate et TaxiWay qui correspondent aux entités utilisées par l'avion.
— 8 dates qui vont correspondre au début et fin d'attente aux différents endroits d'attente de la simulation.

3.1.3 TaxiWay / GateWay / Piste

Pou réaliser ces derniers objets, le travail fut considérablement plus simple. Étant donné qu'on peut savoir si un objet est disponible ou pas pour la réalisation d'un évènement via le paramètre "état", il nous suffit de lui affecter le bon type via la bonne méthode issu de la classe mère AeroportEntity.

3.2 Quels évènements faut-il implémenter ?

Ici, il nous a uniquement fallu réutiliser la classe ISimEntity afin de réaliser cette partie du travail.Tous les évènements de l'aéroport hérite de cette classe et la figure 3.3 montre comment les évènement s'enchainent.

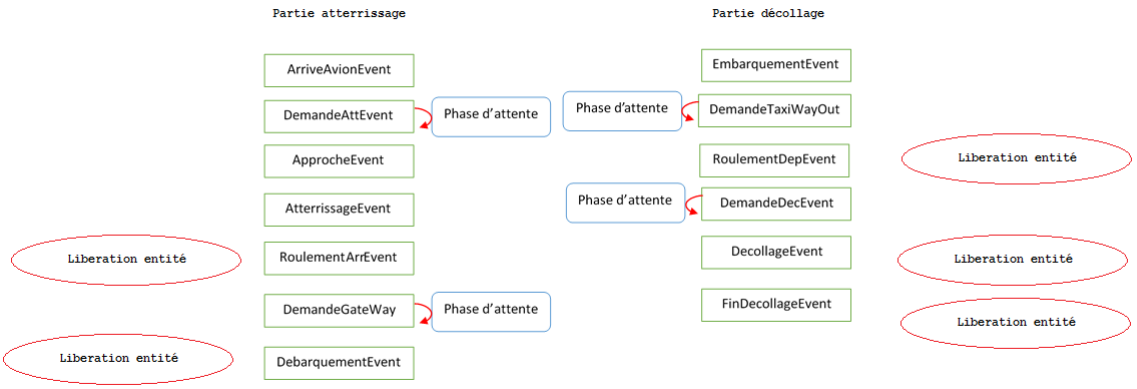


FIGURE 3.3 – Évènements permettant la réalisation d'un processus d'un avion au cours de la simulation.

3.2.1 ArriverAvionEvent

Cette classe va permettre la création des avions pendant tout le long de la simulation 3.4. Nous avons implémenter par conséquent cette classe de telle manière que d'une part elle créer un avion avec un identifiant unique qui sera utilisé pendant tout le processus et d'autre part qu'elle programme l'arrivée du prochain avion en fonction de la plage horaire.

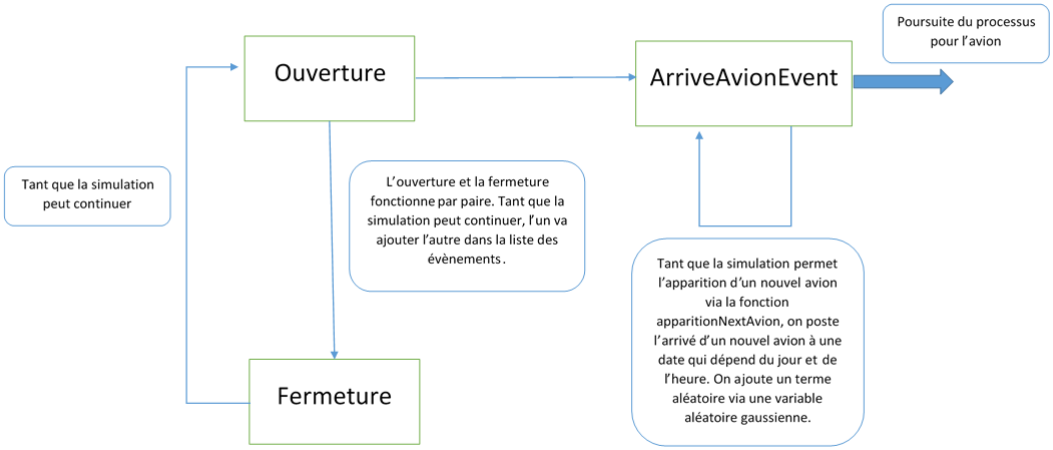


FIGURE 3.4 – Fonctionnement de l'ouverture et la fermeture de l'aéroport et la création des nouveaux avions.

3.2.2 Ouverture

L'évènement ouverture est mis en place à chaque début de journée et va lancer la création d'un premier avion avec l'ajout d'un event ArriverAvionEvent à la liste d'évènement à la date d'ouverture. A chaque ouverture, on fait un test pour savoir si il pleuvra ou pas pendant la journée et on incrémente le jour de la semaine.

De plus, elle va insérer un évènement fermeture qui va indiquer quand l'arrivée d'avion s'arrête3.4.

3.2.3 Fermeture

L'évènement Fermeture est en quelque sorte le complémentaire de l'évènement ouverture. Lorsque l'événement fermeture arrive, il indique à la simulation la fin des arrivées d'avion et insère un nouvel évènement Ouverture si la date ne dépasse pas la date de fin de simulation3.4.

3.2.4 DemandeAttEvent

Cet évènement est un des premier points importants de la simulation. Il doit vérifier ici la disponibilité de la piste et du taxiWay pour l'attérissage de l'avion.

Si les conditions sont réunies , il va ajouter un événement ApprocheEvent. Dans le cas contraire, l'avion sera placé dans une liste d'attente et attendra qu'un avion envoie un évènement de libération de piste pour refaire une demande.

La figure suivante montre comment se passe les évènement de demande dans la simulation 3.5.

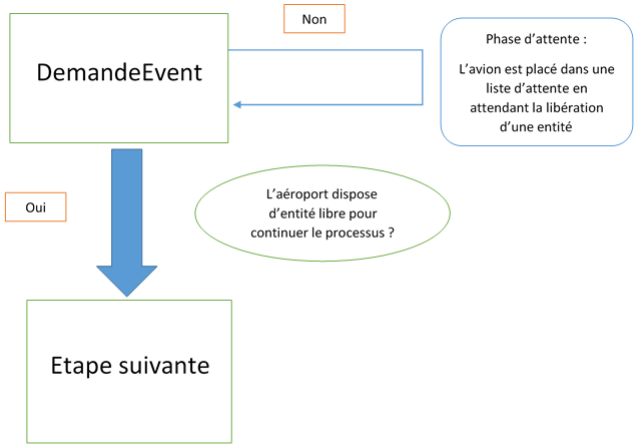


FIGURE 3.5 – Réalisation d'une demande d'entité dans la simulation et les évènement résultant.

3.2.5 ApprocheEvent

L'évènement Approche correspond à la phase d'atterrissage de l'avion une fois qu'il a trouvé une piste et un texiWay d'entrée de libre. Le temps d'approche est compris entre 2 et 6 minutes et si il y a de la pluie, le temps trouvé est doublé. L'évènement Atterrissage est donc posté avec ce délai.

3.2.6 AterissageEvent

L'évènement AterissageEvent modélise l'atterrissage de l'avion. Il prend dans tout les cas deux minutes à se réaliser.

Il va donc poster l'évènement RoulementArrEvent avec un retard de deux minutes.

3.2.7 RoulementArrEvent

L'évènement RoulementArrEvent est un évènement simple qui modélise la phase de roulement jusqu'aux gateWay. Nous avons considérer que l'avion peut attendre sur le TaxiWay d'entrée et de sortie. De plus, la piste est maintenant libre et on va poster un nouvel évènement LibérationPisteEvent

Il va également ajouter le prochain évènement DemandeGatWay avec le retard de 2 à 6 minutes.

3.2.8 LiberationPisteEvent

Cet évènement est très important pour l'exécution du programme. Il est appelé quand un avion a fini d'utiliser une piste et il va chercher les avions qui sont dans une phase d'attente d'atterrissage ou de décollage, prendre celui qui est le plus prioritaire et va poster une nouvelle demande d'atterrissage ou de décollage.

La figure suivante montre comment la libération d'une piste fonctionne 3.6 :

3.2.9 DemandeGateWayEvent

Cet évènement doit vérifier ici la disponibilité d'un gateWay pour l'avion.

Si les conditions sont réunies , il va ajouter un événement DebarquementEvent. Dans le cas contraire, l'avion sera placé dans une liste d'attente et attendra qu'un avion envoie un évènement de libération de gateWay pour refaire une demande.

3.2.10 DebarquementEvent

L'évènement DebarquementEvent est un évènement qui modélise le débarquement des passagers.

Il va ajouter le prochain évènement EmbarquementEvent qui va prendre une quarantaine de minute. De plus, le taxiWay est rendu libre et un évènement LiberationTaxiWayOut est posté.

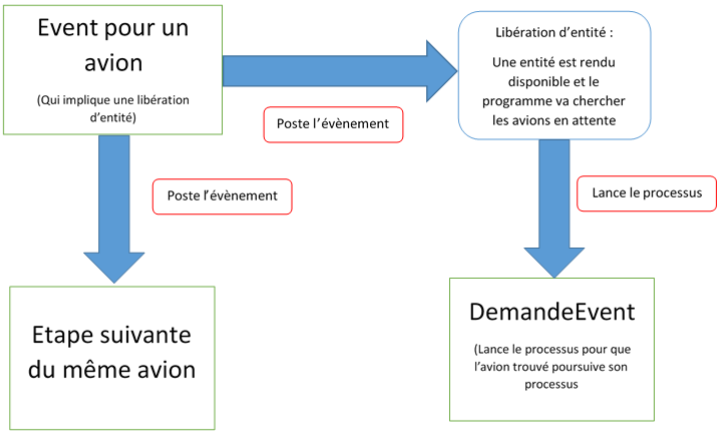


FIGURE 3.6 – Événements résultants suite à une libération d'entité par un avion.

3.2.11 LiberationTaxiWayInEvent

Cet évènement est appelé quand un avion a fini d'utiliser un taxiWay d'entrée et il va chercher les avions qui sont dans une phase d'attente d'atterrissage, prendre celui qui est le plus prioritaire et va poster une nouvelle demande d'atterrissage.

3.2.12 EmbarquementEvent

L'évènement EmbarquementEvent est un évènement simple qui consiste uniquement à ajouter un temps de retard qui traduit l'embarquement de l'avion.

Il consiste uniquement à ajouter le prochain évènement DemandeDecEvent avec le retard demandé par le cahier des charges.

3.2.13 DemandeTaxiWayOutEvent

Cet évènement est un des points névralgiques de la simulation. Lorsqu'un avion souhaite décoller comme lorsqu'il souhaite atterrir, il doit vérifier la disponibilité du taxiWay.

Cette évènement va d'abord voir quelles sont les entités disponibles, les stocker dans une variable temporaire et si toutes les conditions sont réunies, il va ajouter un événement RoulementDepEvent. Dans le cas contraire, l'avion sera placé dans une liste d'attente et attendra qu'un avion envoie un évènement de libération de TaxiWay de sortie pour refaire une demande.

3.2.14 RoulementDepEvent

L'évènement RoulementDepEvent va uniquement traduire le roulement de l'avion pour le décollage.

Il va ajouter le prochain évènement DecollageEvent avec un retard compris entre 2 et 5 minutes.

Il va également poster un évènement liberationGateWay pour signaler la libération de la gate occupée et la libérer.

3.2.15 LiberationGateWayEvent

Cet évènement est appelé quand un avion a fini d'utiliser un gateWay et il va chercher les avions qui sont dans une phase d'attente pour entrer, prendre celui qui est le plus prioritaire et va poster une nouvelle demande de gateWay.

3.2.16 DemandeDecEvent

Lorsqu'un avion souhaite décoller comme lorsqu'il souhaite atterrir, il doit vérifier la disponibilité d'une piste.

Cette évènement va d'abord voir quelles sont les pistes disponibles si toutes les conditions sont réunies, il va ajouter un événement DecollageEvent. Dans le cas contraire, l'avion sera placé dans une liste d'attente et attendra qu'un avion envoie un évènement de libération de piste pour refaire une demande.

3.2.17 DecollageEvent

L'évènement DecollageEvent est également un évènement simple qui traduit le décollage de l'avion.

Il va donc ajouter l'évènement FinDecollageEvent avec un retard demandé par le cahier des charges qui est de 3 minutes.

3.2.18 FinDecollageEvent

Cet évènement est la dernière étape du processus. Une fois que l'avion a décollé, il est supprimé de la liste d'avion et ses données importantes sont inscrites dans le Logger.

De plus, un évènement LiberationPisteEvent est ajouté dans la liste d'évènement pour indiquer que la piste et le taxiway emprunter sont de nouveau disponible.

3.3 Quelles entités de contrôle ?

Les différentes classes ci-dessous régulent toute la simulation. Le moteur va être là pour instancier les entités et implémenter une partie des fonctions qui servent à l'interaction des évènement et des entités entre eux. Le moniteur est la classe qui sert à lancer la simulation et la classe globalVariables sert à la lecture des données de la simulation.

3.3.1 AeroEngine

Cette classe instancie toute les entités de la simulation. Les différentes entités sont placé dans une seule liste global listEntityAero et les avions sont placés dans une liste distincte listAvion. C'est également ici que sont placées les listes d'attente qui servent à la régulation des processus des avions. C'est également ici que nous gérons l'information sur la pluie et le jour de la semaine.

Cette classe a trois principales :

— La méthode init() qui va initialiser toutes les entités et les placer dans les différentes listes dédiées.

— La méthode appartionNextAvion()qui calcul l'horaire d'arrivé du prochaine avion et qui est utilisée dans l'évènement arriverAvionEvent.

— La méthode RevisionOuverture qui regarde la première date d'ouverture et si elle se produit après l'heure d'ouverture de l'aéroport, elle va forcer la simulation à commencer le lendemain.

3.3.2 globalVariables

Cette classe permet la lecture du fichier texte qui contient les différents paramètres de la simulation. Elle instancie tous les paramètres de la simulation et Elle possède deux fonctions :

— Une fonction readScript() qui va lire le fichier texte en paramètre et qui contient tous les paramètres de la simulation.

— Une fonction readLine() qui va lire une ligne du fichier texte et en fonction du premier paramètre lu, elle va initialiser une des variables de la classe qui servira ensuite à la modélisation de l'aéroport.

Elle est lancée en début de la simulation afin d'initialiser la simulation. De plus, c'est dans ce fichier texte qu'est placé le titre du logger qui contiendra les informations de la simulation.

3.3.3 AeroMonitor

C'est dans cette classe que la simulation est initialisée.

Cette classe va instancier le moteur de simulation avec ses différents entités. Son unique méthode run() va initialiser le logger et lancer la simulation tant qu'il y a des évènements restant dans la liste ou que la date de fin de simulation n'est pas atteinte. Enfin, c'est dans cette classe que se trouve le main de la simulation. Il fait d'abord appel à la classe globalVariables afin d'initialiser les variables de la simulation et ensuite initialise le moniteur avec une date de début et de fin et enfin il fait appel à la méthode run().

Chapitre 4

Implémentation du modèle

Description du code, du fonctionnement du moteur de simulation, des générateurs aléatoires. Ceci peut être aidé par le placement de commentaires pertinents dans le code. Manuel utilisateur : description du fonctionnement, paramétrage. Le code source, fichiers de données et exécutables seront fournis sous forme électronique dans des sous-répertoires src, data, bin respectivement.

4.1 Génération des nombre aléatoire

Nous devons à plusieurs reprises générer des nombres aléatoires au cours de la simulation. A cette fin, nous avons décider de placer dans le moteur de simulation un générateur de nombre aléatoire et il est utilisé ainsi :

- **Génération d'un nouvel avion** : dans ce cas, nous avons décider que l'horaire d'arrivé serait obtenu en fonction de la plage horaire et du jour de la semaine. Dans le cas où nous sommes en période de pointe en semaine, les avions arrivent en moyenne toute les 10 min. Le temps d'apparition du prochaine avion est donc obtenu grâce à une variable aléatoire de moyenne 10 et d'une certaine variance qui est pour nous de 2 minutes. Lorsqu'un avion commence sont processus d'atterrissage décollage, il génère donc l'évènement ArriverAvionEvent à cette date obtenue.
- **Durée d'un évènement** : Souvent, un évènement peut durée entre X et Y minute. Pour générer l'horaire du prochain évènement, on va générer un temps qui vaut X plus une variable aléatoire entre (Y-X) qu'on génère en utilisant la fonction nextlong() et en prenant la valeur absolue modulo (Y-X).

4.2 Utilisation de liste d'attente

Afin de pouvoir gérer l'attente des avions dans certaines étapes du processus d'atterrissage décollage, nous avons créer 4 arrayList d'avion qui peuvent être utilisées comme suit :

- Si un avion doit être mis en attente, il est ajouté à la liste d'attente correspondante.
- Si une entité se libère et qu'il est prioritaire, il est sélectionné et si les conditions sont remplies, il passe à la prochaine étapes du processus. Il est enfin enlevé de la liste.
- Dans le cas contraire, il reste en attente et donc dans la liste.

Étant donné que les arrayList de l'API java fonctionne selon le principe du add last, il nous suffit de chercher le premier élément de la liste pour avoir l'avion qui est le plus longtemps en attente.

4.3 SortedList

Cette liste qui nous a été fournie nous permet de gérer de façon assez simple la boucle de sortie des évènements de la simulation. le principe de cette liste est quelle va trier automatiquement ordonner ses éléments en fonction d'un attribut des entités présentes. Pour nous cet élément correspond à la date d'exécution de l'évènement et pour comparer deux dates, nous avons implémenté la fonction compareTo() qui va permettre à la liste d'ordonner la liste en plaçant l'évènement le plus récent en tête de la liste.

4.4 Gestion des priorités

Lorsqu'une piste se libère, l'évènement LiberationPisteEvent va chercher les avions en attente pour le décollage et l'atterrissage. Il est nécessaire de donner une priorité à l'un des deux. Étant donné que le nombre de gateWay disponible est limité, nous avons implémenter le code de tel sorte qu'il regarde d'abord si un avions

peut décoller et ensuite il regarde si un autre avion peut atterrir de tel sorte que le décollage soit prioritaire. Cela a un lourd impact sur le résultat final.

Chapitre 5

Compte-rendu de vérification et validation

5.1 Principaux résultats

Une fois que tout le programme est fonctionnel, on observe les résultats suivants :

— Étant donné que nous avons donné la priorité au décollage, on observe que les résultats sont principalement concentrés à l'atterrissage. Le temps d'attente au décollage, pour l'acces au gateWay et au taxiWay de sortie sont très faibles et dépasse presque jamais 5 minutes.

— On observe un temps de retard est d'autant plus grand que le nombre de gateWay est faible. Néanmoins, si on a plus de 6 gateWay, on observe que le retard ne diminue plus.

— Etant donné que le temps d'occupation d'un taxiWay est compris entre 6 et 13 minutes et que le temps pour le décollage est de 3 minutes, il est normal que les retards soient concentrés à l'atterrissage.

Les premiers résultats demandés sont affichés dans la partie suivante.

5.2 Fonctionnement du programme

5.2.1 Avantage du modèle

Les principaux avantages de ce modèle sont :

— Une réutilisation assez simple de ce modèle à tout type de simulation évènementielle. La logique reste la même et la technique de demande d'accès à une ressource et de la libération de cette dernière permet de bien voir ce que fait le programme.

— On peut facilement avoir accès à une information en changeant juste un champs de donné soit sur une entité soit en changeant le process sur un évènement.

— Le logger est très facile à utiliser. Il suffit d'initialiser un objet BasiIRecordable et donner les champs nécessaires. Nous avons néanmoins changer le code source du logger afin de supprimer les trois premières colonnes que nous n'utilisons pas.

— Pour utiliser le simulateur, nous utilisons un fichier texte qui contient toutes les information sur la simulation et également le nom que le logger aura. Il est assez simple d'utilisation et un exemple est fourni en annexe. La syntaxe de base est très simple et on peut facilement changer un paramètre pour faire varier des constantes de la simulation.

5.2.2 Problèmes rencontrés

Nous avons un seul problème lors de la réalisation du code mais ce dernier nous a pris beaucoup de temps. Parfois, une piste et un taxiWay d'entrée pouvait se libérer en même temps. Dès lors, un avion pouvait percevoir ces deux évènement, poster deux évènement de demande de ressource, et dans un premier temps continuer son processus et en même temps voir que les ressources de l'aéroport sont occupées et être placé dans la liste d'attente. L'avion pouvait donc être dans deux états différents.

Pour résoudre ce problème, nous avons implémenté dans le processus de chaque évènement de libération de ressource l'algorithme suivant :

— trouver l'avion en attente prioritaire.

— lancer directement le processus de demande de ressource et donc ne pas poster un évènement de demande.

— éliminer l'avion de la liste d'attente

Ainsi, lorsque deux libération d'entités se produisent en même temps, la première demande traite la demande du premier avion prioritaire, l'enlève de la liste et la seconde va donc prendre un autre avion et voir que les entités ont déjà été alloué et va donc laissé l'avion dans la liste d'attente.

Chapitre 6

Résultats de la simulation

Résultats obtenues avec les premières simulations
Nous avons simulé l'aéroport de Brest sous différentes conditions afin de déterminer si notre simulation est cohérente avec les données fournies en entrée.

6.1 Résultat de la configuration : 1 piste, 1 taxiWay d'entrée et de sortie et 4 gateWays

Dans cette configuration, nous arrivons aux résultats suivants :

Durée moyenne d'attente pour la phase d'arrivée
20,9 min
Durée moyenne d'attente en TaxiWayIn
2,4 min
Durée moyenne d'attente en TaxiWayOut
0 min
Durée moyenne d'attente pour la phase de départ
0,9 min
Durée moyenne d'attente
24,2 min

FIGURE 6.1 – Temps d'attente pour la configuration 1 piste, 1 taxiWay d'entrée et de sortie et 4 gateWays.

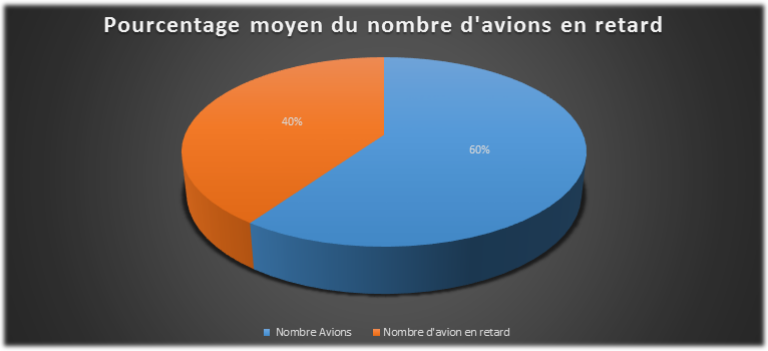


FIGURE 6.2 – Pourcentage d'avion en retard pour la configuration 1 piste, 1 taxiWay d'entrée et de sortie et 4 gateWays.

Le pourcentage de retard est ici énorme mais s'explique facilement par le fait que nous avons mis le nombre minimum de Gates.

6.2 Résultat de la configuration : 1 piste, 1 taxiWay d'entrée et de sortie et 6 gateWays

Dans cette configuration, nous arrivons aux résultats suivants :

Durée moyenne de la phase d'arrivée
2,4 min
Durée d'attente moyenne en TaxiWayIn
0 min
Durée d'attente moyenne en TaxiWayOut
0 min
Durée moyenne de la phase de départ
0,2 min
Durée moyenne d'attente
2,5 min

FIGURE 6.3 – Temps d'attente pour la configuration 1 piste, 1 taxiWay d'entrée et de sortie et 4 gateWays.

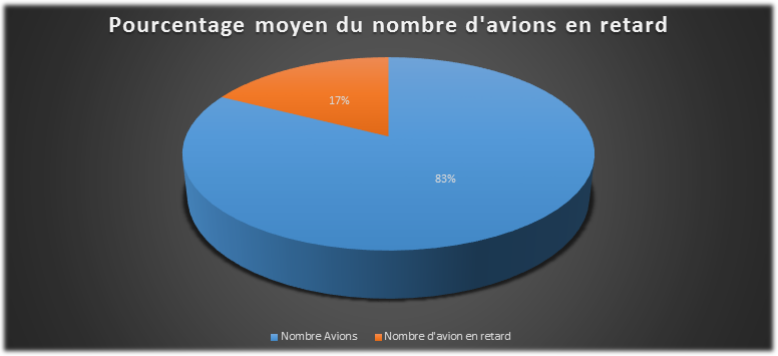


FIGURE 6.4 – Pourcentage d'avion en retard pour la configuration 1 piste, 1 taxiWay d'entrée et de sortie et 4 gateWays.

Nous observons que le temps d'attente chute très rapidement. Néanmoins, on observe encore que les avions doivent encore attendre longtemps en l'air avant de pouvoir se poser.

6.3 Résultat de la configuration : 1 piste, 1 taxiWay d'entrée et de sortie et 8 gateWays

Dans cette configuration, nous arrivons aux résultats suivants :

Durée moyenne de la phase d'arrivée
2,3 min
Durée moyenne d'attente en TaxiWaysIn
0 min
Durée moyenne d'attente en TaxiWayOut
0 min
Durée moyenne de la phase de départ
0,2 min
Temps moyen d'attente
2,5 min

FIGURE 6.5 – Temps d'attente pour la configuration 1 piste, 1 taxiWay d'entrée et de sortie et 4 gateWays.

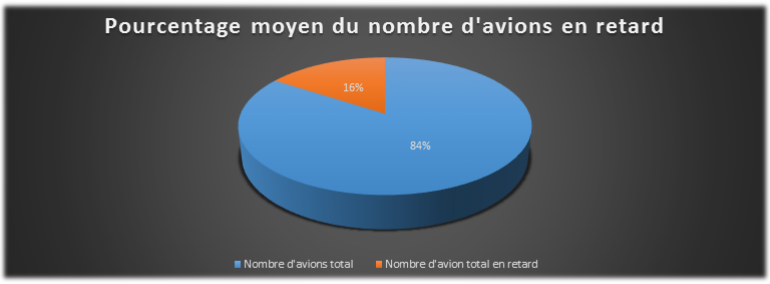


FIGURE 6.6 – Pourcentage d'avion en retard pour la configuration 1 piste, 1 taxiWay d'entrée et de sortie et 4 gateWays.

Les résultats restent quasi similaires que précédemment.

6.4 fréquentation horaire

On observe la fréquentation horaire suivante pendant la simulation :

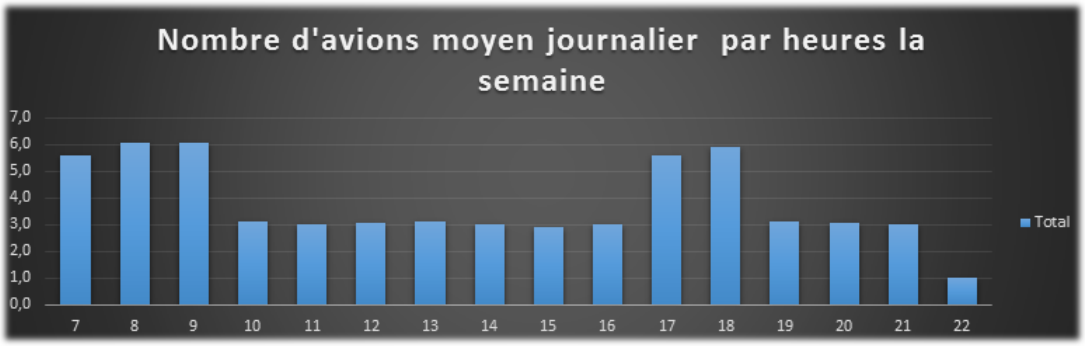


FIGURE 6.7 – Frequentation horaire en semaine.

On observe bien que la fréquentation horaire correspond bien au cahier des charge. Notre modélisation d'apparition d'avion fait que un avion peut parfois notifier son arrivé un peu après 22h ce qui explique la dernière colonne. Ce point mis à part, la simulation marche très bien.

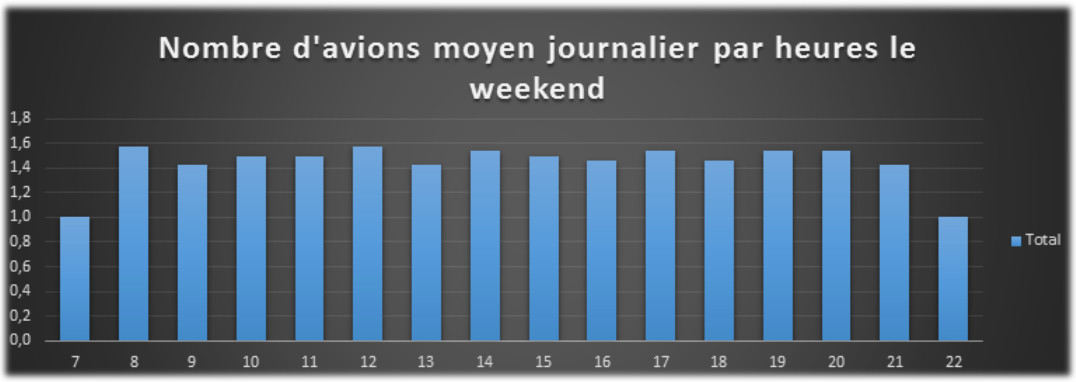


FIGURE 6.8 – Frequentation horaire en WE.

6.5 Validation du simulateur

Nous pouvons constater que le pourcentage de retard est beaucoup plus élevé avec 4 Gates qu'avec 6 ou 8 Gates car il y a moins d'avions qui peuvent débarquer les passagers simultanément. De plus, la différence de retard entre 6 et 8 Gates est quasi négligeable. Ceci peut s'expliquer par le fait que nous n'avons ici seulement que 1 TaxiWay d'entrée. En ce qui concerne les fréquentations de l'aéroport, elle est beaucoup plus importante sur les plages horaires 7h-10h et 17h-19h la semaine, qui correspondent aux heures de pointes et elle est bien étalée sur toute la journée les weekends. Pour finir sur les vérifications, nous avons constaté que l'attente était beaucoup plus centrée sur la phase d'arrivée. Ceci peut s'expliquer comme dit précédemment par le fait que les avions prêt à décoller sont prioritaires sur les avions en phase d'atterrissage. De plus, le fait de n'avoir qu'un seul TaxiWay d'entrée augmente considérablement cette phase d'attente. Le temps d'attente avec 6 et 8 Gates et très inférieur à celui avec 4 Gates. Les résultats paraissent donc cohérents en fonction des résultats souhaités. Pour la validation, nous remplissons un tableau des résultats attendus et regardons si les résultats obtenus sont cohérents ou non.

✓ **Validation du simulateur :**

	Cohérent	Moyennement cohérent	Pas cohérent
Pourcentage moyen de retard	✗		
Fréquentation moyenne la semaine	✗		
Fréquentation moyenne le weekend	✗		
Temps d'attente moyen	✗		

FIGURE 6.9 – Récapitulatif de la simulation.

Chapitre 7

Analyse des résultats

7.1 Quelle configuration optimale ?

Avec les trois simulations précédentes, nous avons obtenus les résultats suivants :

	4 Gates	6 Gates	8 Gates
Retard (%)	40%	17%	16%
Temps retard moyen (min)	21.8 min	2.5 min	2.5 min
Temps d'attente moyen (min)	24.2 min	2.5 min	2.5 min

FIGURE 7.1 – Récapitulatif des essais.

Nous constatons que le pourcentage de retards, le temps moyen des retards et le temps moyen d'attente sont très élevés pour un aéroport avec seulement 4 Gates. Si nous passons à 6 Gates ces chiffres sont largement diminués et reste raisonnables. De plus, il n'y a pas de grand changement avec les chiffres obtenus avec 6 et 8Gates. Nous supprimons donc le fait d'avoir que 4 Gates. Il faut donc avoir 6 Gates minimum. Nous avons pu remarquer précédemment que c'est la phase d'arrivée qui génère le plus d'attente. C'est pourquoi, nous allons rajouter un TaxiWay d'entrée et refaire les simulations pour 6 et 8 Gates.

Voici les résultats pour la configuration avec 6 gateWay et 2 taxiWay d'entrée :

Nombre de piste	Nombre de GateWay	Nombre de TaxiWay d'entree	Nombre de TaxyWay de sortie
1	6	2	1

Nombre d'avions	4449
Nombre d'avion en retard	376
Temps moyen d'attente	1,2 min
Temps moyen de retard	0,8 min

FIGURE 7.2 – résultats pour la configuration avec 6 gateWay et 2 taxiWay d'entrée.

Et voici les résultats pour la configuration avec 8 gateWay et 2 taxiWay d'entrée :

Nombre de piste	Nombre de GateWay	Nombre de TaxiWay d'entree	Nombre de TaxyWay de sortie
1	8	2	1

- **Retards moyen sur 3 mois :**

Nombre d'avions
4456
Nombre d'avion en retard
264
Temps moyen de retard
0,4 min
Temps moyen d'attente
0,5 min

FIGURE 7.3 – résultats pour la configuration avec 8 gateWay et 2 taxiWay d'entrée.

Les nouveaux résultats que nous pouvons comparer sont les suivants :

6 Gates	1 TaxyWayIn	2 TaxyWayIn
Retard (%)	17%	8%
Temps retard moyen (min)	2.5 min	0.8 min
Temps d'attente moyen (min)	2.5 min	1.2 min

8 Gates	1 TaxyWayIn	2 TaxyWayIn
Retard (%)	16%	6%
Temps retard moyen (min)	2.5 min	0.5 min
Temps d'attente moyen (min)	2.5 min	0.4 min

FIGURE 7.4 – récapitulatif des deux essais.

Nous remarquons que le fait de mettre 2 TaxiWay d'entrée réduit encore le retard et l'attente. Un aéroport avec 8 Gates et deux TaxyWays est plus performant mais un aéroport avec 6 Gates et deux TaxyWays suffit amplement et pour un prix moindre. Le fait de mettre 3 TaxyWays d'entrée ne réduit pas d'avantage le retard et l'attente.

Un aéroport avec 6 Gates et 2 TaxyWays d'entrée serait donc le mieux du point de vue des retards et du budget.

7.2 Quels résultats obtenons nous ?

7.2.1 Récapitulatif des résultats sur trois mois

Nous allons donc présenter les différents résultats obtenus pour cette configuration optimale selon nous.

➢ **Retards moyen sur 3 mois :**

Nombre d'avions
4449
Nombre d'avion en retard
376
Temps moyen d'attente
1,2 min
Temps moyen de retard
0,8 min

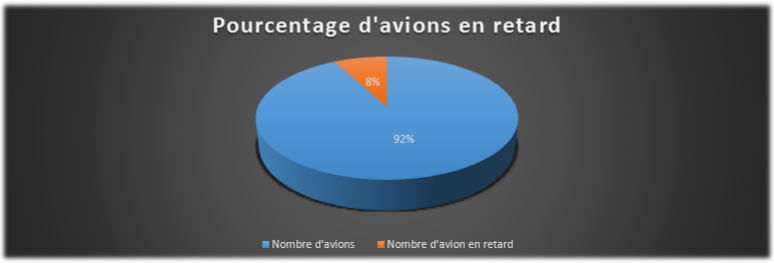
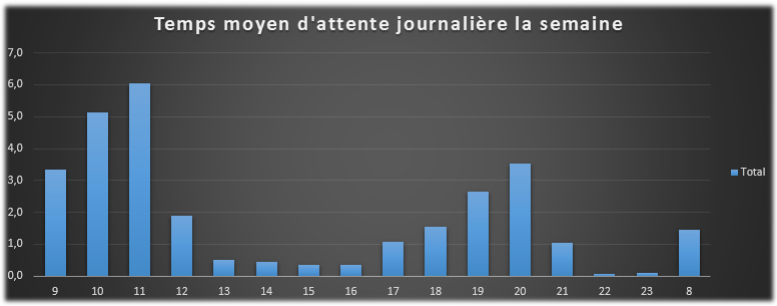


FIGURE 7.5 – récapitulatif des trois mois de simulation.

➢ **Durée moyenne d'attente journalière la semaine:**



➢ **Durée moyenne d'attente journalière le weekend:**

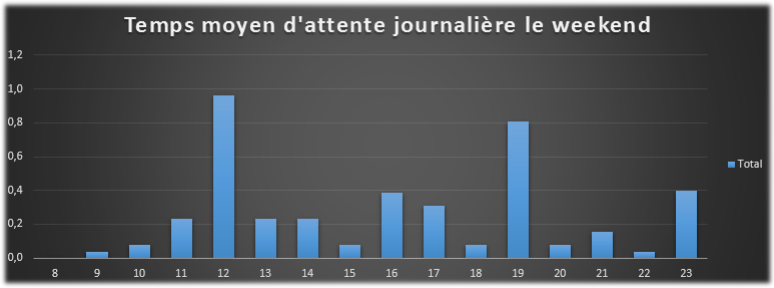


FIGURE 7.6 – Retard sur 3 mois de simulation.

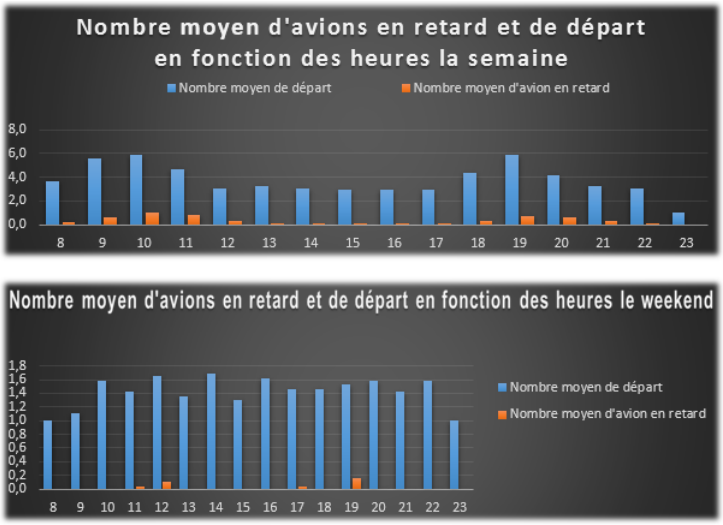


FIGURE 7.7 – Proportion d'avion en retard en fonction de l'heure.

7.2.2 Temps de la phase d'atterrissage et de décollage.

La durée d'une phase d'arrivé est la différence entre le moment du début du débarquement et le moment de l'approche de l'avion tandis qu'une durée de phase de départ correspond à la différence entre le moment de la fin du décollage et le moment de fin d'embarquement.

Nous obtenons les résultats suivants :

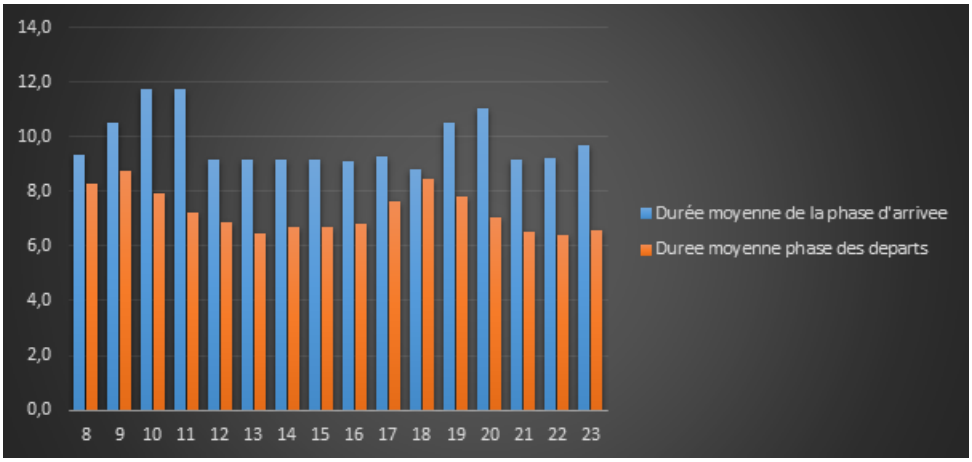


FIGURE 7.8 – Durée moyenne de la phase d'atterrissage et de décollage par tranche horaire pendant la semaine.

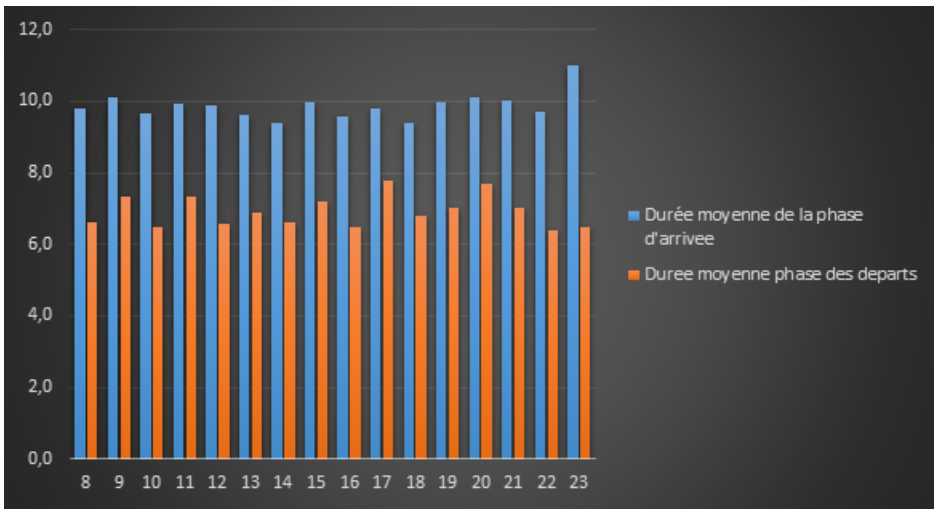


FIGURE 7.9 – Durée moyenne de la phase d'atterrissage et de décollage par tranche horaire pendant le Week-End.

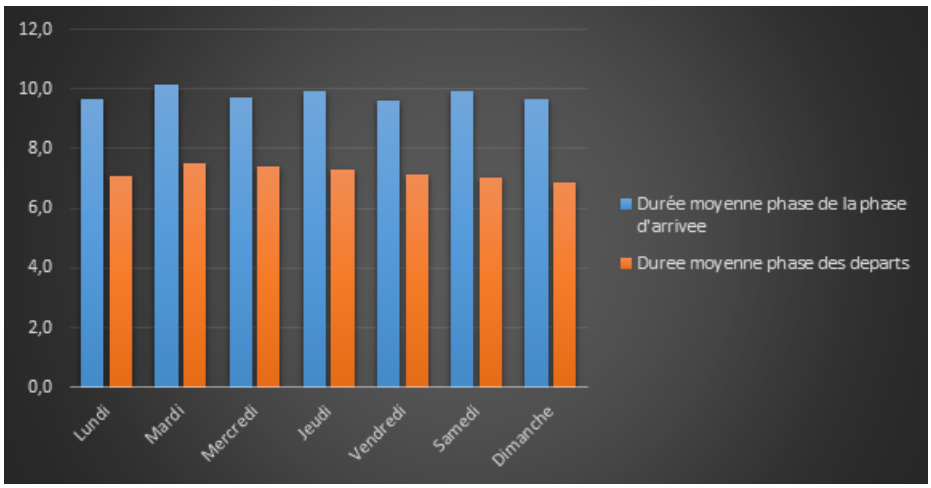


FIGURE 7.10 – Durée moyenne de la phase d'atterrissage et de décollage selon le jour de la semaine.

Nous pouvons constater que les durées moyennes des phases d'arrivée et de départ sont les plus importantes lors des heures de pointe la semaine et que la durée moyenne des phases de départ est toujours inférieure à la durée moyenne des phases d'arrivée. Ceci s'explique par le fait que les décollages sont prioritaires et que le cahier des charges impose un temps de décollage plus faible que le temps d'atterrissage.

7.3 Conclusions

Avec tous ces résultats obtenus pour une simulation avec 6 Gates et 2 TaxiWays d'entrée, nous constatons beaucoup de choses. A première vue l'aéroport semble très bien fonctionner sur 3 mois avec des temps de retard et d'attente relativement faible de l'ordre de 1 minute. La fréquentation reste en moyenne plus élevée la semaine que le weekend avec une hausse pendant les heures de pointes (7h-10h, 17h-19h). Nous observons également que les temps d'attente ne sont plus situés majoritairement lors de la phase d'arrivée. En effet, les différentes phases d'attentes ont maintenant un pourcentage quasi similaire ce qui permet d'étaler les attentes. Le temps d'attente et les retards sont plus importants pendant les heures de pointe la semaine mais restent très corrects.

Nous pouvons en conclure que le fait d'avoir un aéroport avec 6 Gates et 2 TaxiWays d'entrée serait optimal du fait de son bon fonctionnement sur 3 mois, de son temps d'attente et de retards moyen très faible, et de son prix plus faible par rapport à un aéroport avec 8 Gates et 2 TaxiWays d'entrée qui au final n'apporte quasiment aucune différence.

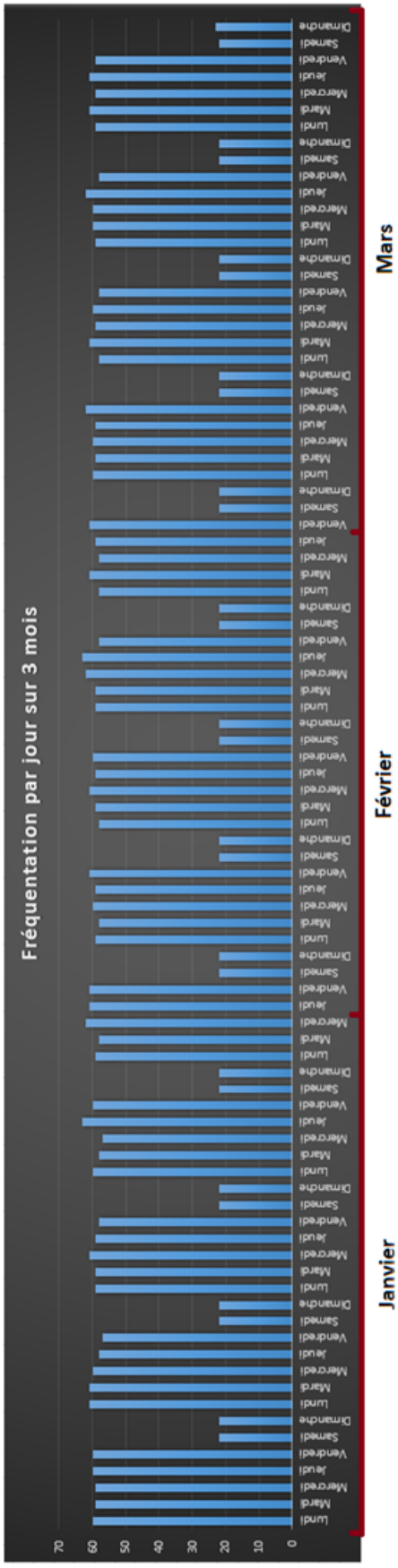


FIGURE 7.11 – Fréquentation sur trois mois.

Chapitre 8

Perspectives d'évolutions

Nous avons obtenu avec ce projet un simulateur assez performant et simple de compréhension. La réponse que nous avons proposé pour répondre à la problématique du sujet repose sur la théorie de la contrainte ou la théorie de la congestion.

Dans tout processus qui implique un flux d'entrée qui subit un ensemble de transformation ou suit une succession d'étape afin de générer un flux de sortie, on observe que il y a un élément qui à le rôle de goulot dans la chaine. Ici, on n'a pu voir que d'abord ce sont les gateWay qui occupent le rôle de goulot dans la simulation avec 4 gateWay. Néanmoins, si on a plus de 6 gateWay, le retard ne diminue plus car le goulot s'est déplacé vers les taxiWay d'entrées. De même, on pourrait continuer avec la configuration avec 6 gateWay et deux taxiWay d'entrée car le goulot est maintenant au niveau des pistes d'atterrissage.

On peut donc en conclure que le principe de la modélisation est bon car on voit bien quels sont les endroits qu'il faut modifier pour dimensionner l'aéroport et voir comment on peut l'améliorer.

Néanmoins, la simulation ne prend pas en compte quantité de détails qui sont important :

— les avions ne vont jamais prendre le même temps pour faire le débarquement, l'embarquement et faire le ravitaillement.

— Les différents temps d'exécution de chaque étape de l'atterrissage et de décollage dépendent a priori du modèle d'avion.

— les avions ne vont pas forcément atterrir et décoller le même jour.

Cette liste est clairement non exhaustive et beaucoup d'entre eux sont difficiles à implémenter. Néanmoins si on souhaite avoir un modèle pertinent, il est nécessaire de demander au client quelles informations sont pertinentes pour faire la modélisation.

Table des figures