CY Cergy Paris Université

RAPPORT

pour le projet d'intégration
Licence 3 Informatique

sur le sujet

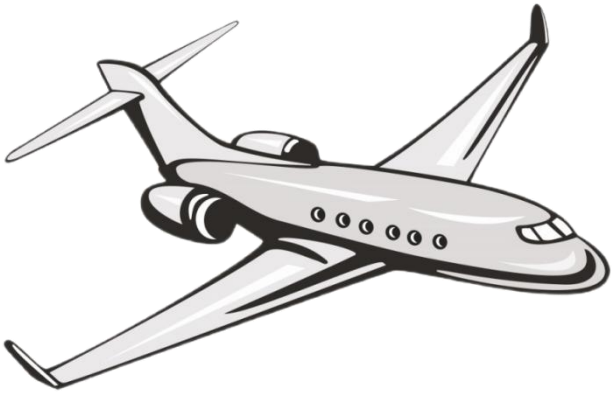
Contrôle du Trafic Aérien

rédigé par

CHEN Junjie, FENG Jiaqi, GUNDUZ Maxime

encadré par

Encadrant de gestion de projet : LIU Tianxiao



Année 2022/2023

Catégorie

Liste des tableaux

Remerciements

Tout d'abord, nous tenons à remercier très sincèrement et à témoigner toute notre reconnaissance à notre professeur de gestion de projet, Monsieur LIU Tianxiao, qui a supervisé ce projet.

Nous remercions également Monsieur LIU Tianxiao de nous avoir appris les bonnes méthodes pro- fessionnelles via ses explications et ses réponses à nos questions durant notre première année à Cergy Paris Université. Nous le remercions une nouvelle fois pour nous avoir proposé les bases de projet pour réaliser cela (le projet Train et le projet Aircraft).

Nous tenons également à remercier tous les membres de notre équipe qui ont contribué à ce projet.

Nous remercions également notre formation et Monsieur LIU Tianxiao de nous avoir proposé ce projet intéressant qui va enrichir notre profil au sein des entreprises.

1 Introduction

Dans le cadre de notre troisième année de Licence en Informatique à Cergy Paris Université, nous avons eu à choisir et à réaliser un projet d'intégration d'une durée d'environ 4 mois (du 24/11/2022 au 09/04/2023) en groupe de 3 personnes.

Notre projet d'intégration consiste à développer un programme qui simule un radar aérien sur un espace aérien en utilisant JAVA. Cet espace aérien contient des aéroports, un relief et des avions re- présentés par des Threads.

L'objectif primordial du projet est de réaliser de la programmation multithreading avec des ressources partagées (zone aérien).

1.1 Contexte du projet

Comme énoncé précédemment, le projet se base sur la conception d'une simulation d'un radar aérien avec un temps d'exécution infini. Le radar met en place des aéroports et des avions qui sont représentés par des threads. Les avions ont une certaine vitesse et une altitude et réalisent des dépla- cements entre eux de manière aléatoire.

L'utilisateur pourra interagir avec un bouton qui permettra de provoquer une situation d'urgence sur un avion sélectionné aléatoirement, et l'avion aura pour mission d'atterrir à l'aéroport le plus proche le plus rapidement possible.

L'utilisateur pourra également quitter le programme avec le bouton "Quitter".

1.2 Objectif du projet

La réalisation de ce projet a pour but les éléments suivants :

- La conception d'une carte.
- L'implémentation des aéroports.
- L'implémentation des avions avec des déplacements.
- L'implémentation de la fonctionnalité de cas d'urgence.

1.3 Organisation du rapport

Ce rapport a pour but de vous expliquer la réalisation de notre programme informatique. Nous commencerons par vous présenter les spécifications de notre programme, avec les fonctionnalités atten- dues et les contraintes du projet. Nous nous attarderons ensuite sur la partie technique en présentant l'architecture du programme et la conception des différentes fonctionnalités. Nous vous présenterons aussi une section avec un manuel utilisateur. Après avoir présenté tout cela, nous vous indiquerons la répartition des tâches et nous ferons le bilan du résultat final.

2 Spécification du projet

Nous avons présenté l'objectif du projet dans la section 1. Dans cette section, nous présentons la spécification de notre logiciel réalisé. Ceci correspond principalement au document de spécification du projet (cahier des charges).

2.1 Outils et langages

Pour la réalisation de ce projet, nous avons utilisé :

- Langage de programmation : JAVA avec la bibliothèque graphique Swing (Version Java SE 20).
- Platemeforme de développement : Eclipse IDE.
- Gestion des versions : Github.
- Outils de communication : Discord et Microsoft Teams.

Notre programme est conçu pour être exécuté sur Windows, Linux et macOS.

2.2 Notions de base et contraintes du projet

2.2.1 Fonctionnement général du logiciel

Notre objectif a été de concevoir les fonctionnalités courantes d'un radar aérien, qui consiste à afficher le trafic aérien d'une zone donnée. L'utilisateur pourra regarder cette simulation fictive autant de fois qu'il le souhaite, dans laquelle des avions se déplacent d'un aéroport à un autre. L'utilisateur pourra interagir uniquement avec un bouton lui permettant de déclencher un problème durant le vol d'un avion, nécessitant un atterrissage rapide. Il est donc intéressant pour l'utilisateur de voir la réaction des avions à l'appui de ce bouton, plusieurs fois. La conception de ce programme nécessite l'utilisation du modèle en cascade. En effet, sans concevoir la carte, il nous est impossible de concevoir les avions et leur déplacement, et sans leur déplacement, il nous est impossible de concevoir la gestion des cas d'urgence.

2.2.2 Notions et termonologies de base

Carte

Notre carte représente une zone aérienne composée de villes et de reliefs tels que des montagnes, des mers et des forêts. Chaque ville possède un aéroport capable de faire décoller et atterrir des avions automatiquement.

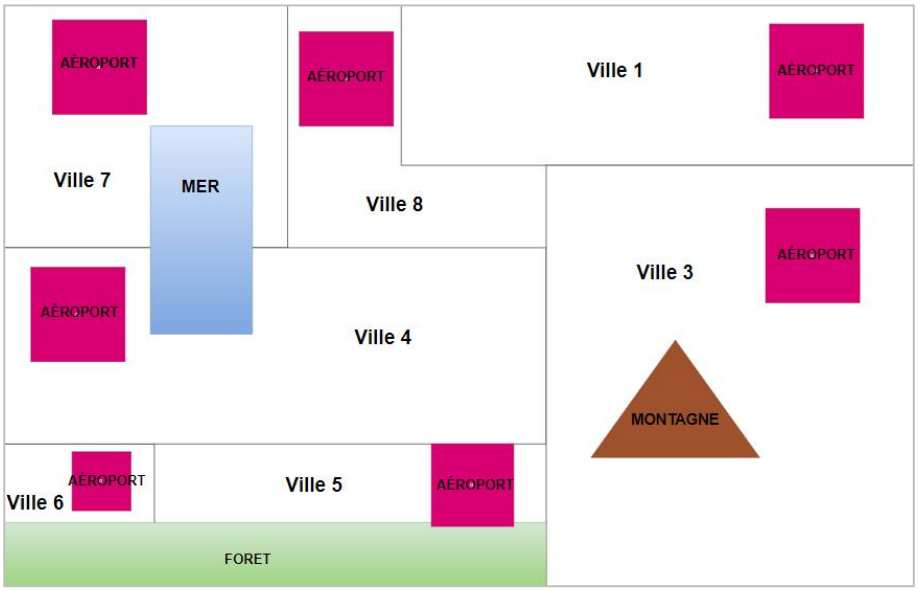


Figure 1 – Schéma illustrant le radar

Les avions

Les avions permettent de relier automatiquement les villes entre elles au sein de la zone aérienne. Notre programme regroupe 10 avions qui assurent continuellement la liaison entre les différentes villes. Un avion est caractérisé par un identifiant qui permet de l'identifier, une vitesse, une altitude et des coordonnées sur la carte.

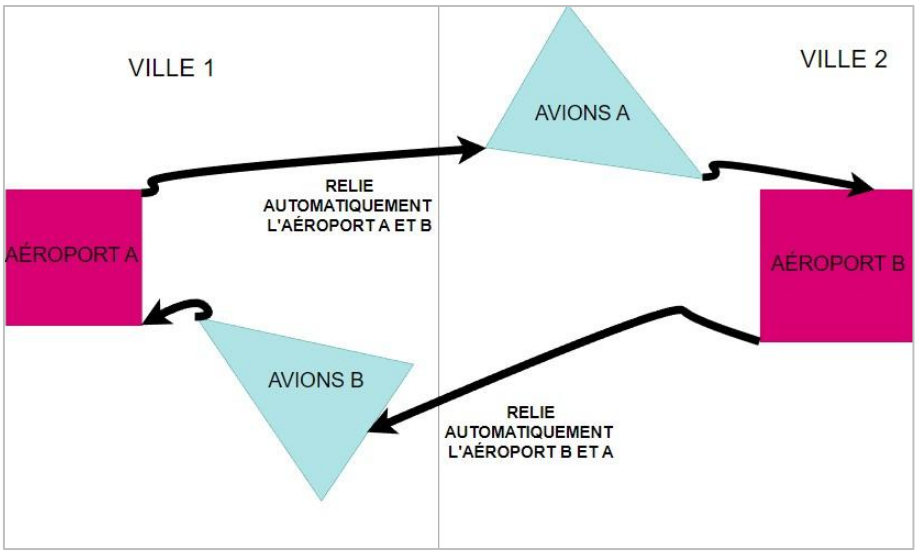


Figure 2 – Schéma illustrant les vols entre les aéroports

Les cas d'urgence

Les situations d'urgence sont des cas provoqués par l'utilisateur du programme, dans lesquels l'avion prend une couleur rouge et doit dévier vers l'aéroport le plus proche pour pouvoir atterrir le plus rapidement possible. Une fois l'avion atterri en toute sécurité, cet avion est remis en bon état et redécolle vers une autre ville.

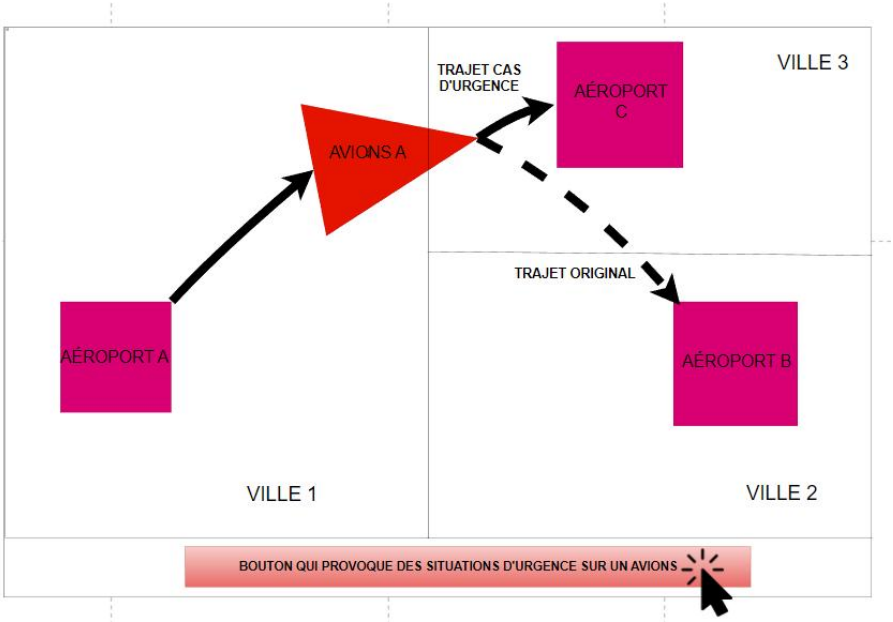


Figure 3 – Schéma illustrant un cas d'urgence

2.2.3 Contraintes et limitations connues

Premier paragraphe Pour réaliser ce projet, nous avons dû respecter plusieurs contraintes, dont la plus importante était le respect du planning et des délais. Nous avons donc réparti les missions à réaliser entre les membres du groupe, en fonction des compétences et des disponibilités de chacun. Nous avons également travaillé ensemble pour améliorer le programme et le présenter aux dates fixées.

Nous avons rencontré des difficultés pour représenter les avions avec du multithreading et la logique de ressources partagées, car nous n'avions pas d'expérience préalable dans ce domaine. Nous nous sommes appuyés sur une base de code fournie par notre professeur (projet train) qui nous a servi de modèle pour implémenter cette fonctionnalité.

Notre situation d'étudiants nouvellement arrivés à Cergy Paris Université nous a privés des séances de Génie Logiciel qui nous auraient permis d'apprendre les bonnes méthodes de travail et de présentation. Nous avons donc bénéficié du soutien de M.LIU et des conseils de nos amis de L3 en L3 qui ont suivi ce cours.

Un autre obstacle auquel nous avons dû faire face a été l'abandon d'un de nos camarades en cours de projet. Conscients du manque de temps pour mener à bien notre projet universitaire, nous avons décidé de nous concentrer sur les fonctionnalités principales. Nous avons ainsi réussi à créer un simulateur de radar aérien automatique qui remplit les critères essentiels.

2.2.4 Fonctionnalités attendues du projet

Fonctionnalités du programme

Un radar aérien permet de visualiser une zone aérienne avec des villes, des aéroports et des avions qui les relient. Notre objectif est de créer une simulation fictive de ce radar, dans laquelle l'utilisateur peut déclencher un problème technique sur un des avions. Ces éléments constituent les

fonctionnalités les plus essentielles pour notre programme, à savoir :

- Permettre de visualiser une zone aérienne,
- Permettre de visualiser des avions,
- Automatiser le déplacement des avions,
- Permettre d'interagir avec un bouton pour provoquer des problèmes techniques.

3. Conception et réalisation du projet

3.1 Architecture globale du logiciel

Les spécifications du projet ayant été décrites dans la section 2, cette section se concentre sur la dimension technique de notre logiciel.

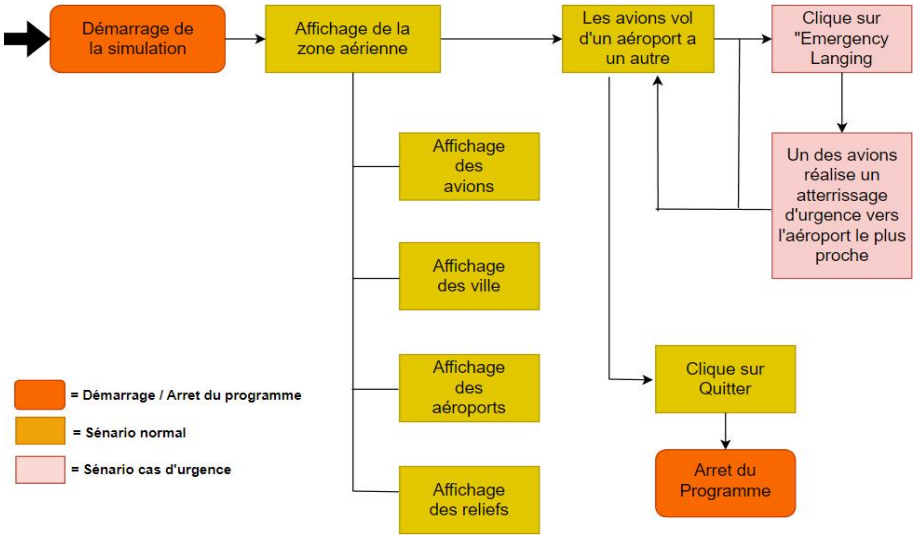


Figure 4 – Schéma illustrant l'Architecture globale du logiciel

Quand un utilisateur lance notre programme, le programme affiche une carte représentant les villes, les reliefs, les aéroports et les avions qui décollent automatiquement vers d'autres aéroports pour effectuer leur trajet. Quand un avion atterrit, il repart vers un autre aéroport de façon continue. Si l'utilisateur clique sur le bouton "Emergency Landing", il provoque une situation d'urgence sur un des avions qui doit atterrir en priorité. Ce processus se répète à chaque fois que l'utilisateur appuie sur le bouton. Le programme continue jusqu'à ce que l'utilisateur arrête la simulation en cliquant sur le bouton quitter.

3.2 Conception des classes de données

Cette sous section se base sur les classes les plus prépondérantes et ainsi l'ensemble des diagrammes UMLs (sans attirbuts et sans méthodes).

Classes	Responsabilités
GameConfiguration.java	Classe gérant les datas, comme la taille de block, la taille de fenêntre et vitesse de l'avion
Map.java	Représentant la classe de map, contenant les informations sur le map
City.java	Représentant la classe de villes, contenant les informations sur le ville (espace aérien, nom, couleur)
Block.java	Représentant la classe de blocks, contenant les informations sur le block (ligne, colonne)
Mountain.java	Représentant la classe de montagnes, contenant les informations sur le montagne (altitude, nom)
Airport.java	Représentant la classe d'aéroports, contenant les informations sur l'aéroport (nom d'aéroport,

	position)
Avion.java	Représente la classe d'avions, contenant les informations sur l'avion (speed, aéroport départ, aéroport d'arriver, altitude)
MobileElement.java	Permet d'obtenir la position de l'objet (avion, aéroport, montagne) courant
ATCDispaly.java	Permet d'appeler les fonctions dans la classe PaintStrategy et d'afficher tous les composants
PaintStrategy.java	Récupère les images nécessaires, classe implémente les fonctions de dessin
MainGUI.java	Gère l'affichage de ATC
MapBuilder.java	Permet de diviser les villes, et initialiser la carte, les aéroports et les montagnes
SimulationUtility.java	Permet de produire un l'avion, l'aéroport et le montagne automatiquement, encapsule diverses fonctions d'outils comme readImage, rafraîchissement du thread
MobileElementManager.java	Classe gérant les avions et les blocks, permet de produire un nombre spécifié d'avions et ouvre la mémoire partagée
BlockManager.java	Détermine si l'avion peut entrer dans le zone de vol partagé
AvionManager.java	Le noyau du contrôleur, permet de contrôler les mouvements des avions
TestATC.java	Permet de lancer la programme

Tableau 1 – L'ensemble de nos classe avec leur fonctionnalité

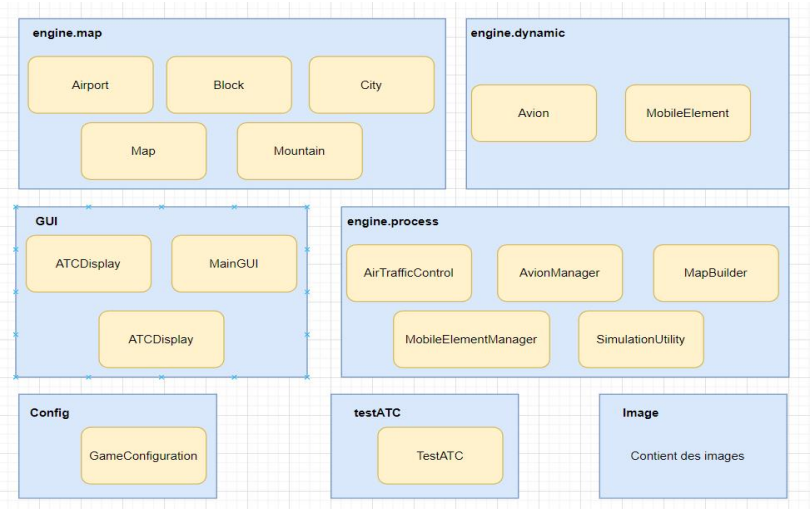


Figure 5 – Schéma illustrant une vue d'ensemble des classe

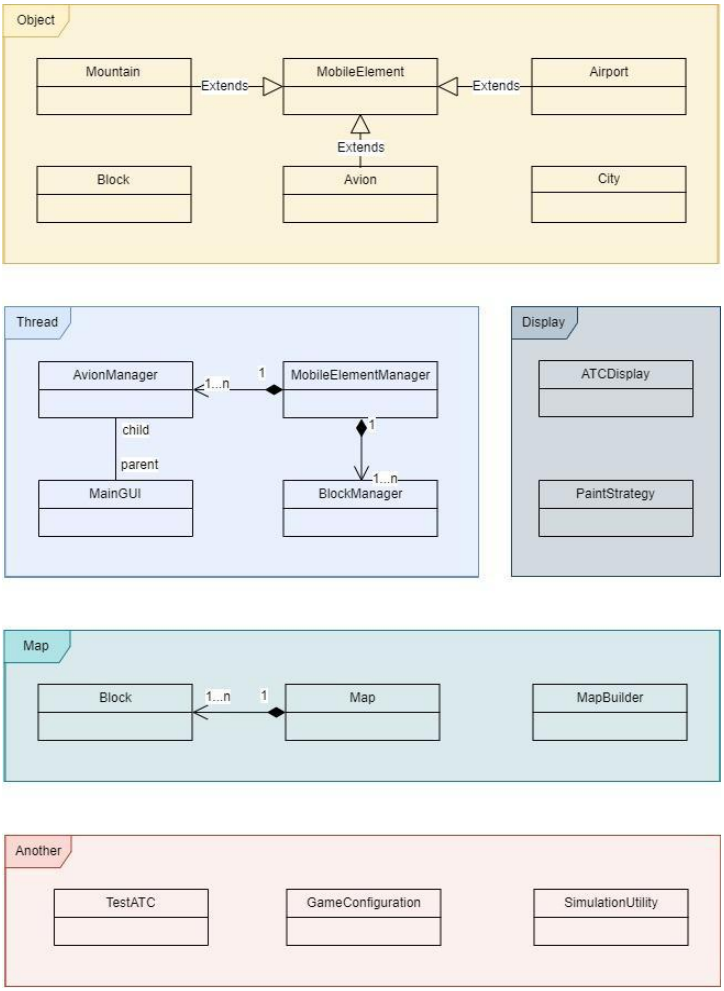


Figure 6 – Diagramme UML

3.3 Conception des traitements (processus)

Dans cette sous-partie, nous allons voir les principaux éléments fonctionnels du logiciel, notamment la conception de la carte, des avions et des cas d'urgence.

3.3.1 Conception de la carte et des zones partagées

La carte est un élément indispensable de notre simulation. Pour concevoir la carte, nous nous sommes inspirés de la logique d'un des programmes vus pendant notre formation (Programme "Aircraft" de Monsieur Liu). Nous avons programmé une classe "Block.java" qui permet de créer des blocs. Ces blocs réunis forment des zones aériennes, et ces zones aériennes forment notre carte. La logique des blocs permet également de créer sur la carte des composants tels que des reliefs et des aéroports. La particularité de ces composants est notamment qu'ils sont des blocs avec une taille plus élevée et des caractéristiques visuelles particulières.

Figure 7 – Schéma représentant la composition de la carte

Les zones sont formées de blocs et sont essentielles dans notre projet. Pour créer ces zones, nous avons découpé la carte en plusieurs zones de 50*50, qui servent de variables partagées pour que tous les avions puissent les consulter et les utiliser pendant leur vol. Chaque avion en vol vérifie s'il y a des avions dans les zones devant lui et à sa gauche et à sa droite. S'il y a des avions, il réduit sa vitesse à l'avance. Si un autre avion n'est pas sur la même trajectoire que l'avion actuel, l'altitude de l'un des avions est modifiée. De plus, nous avons défini un temps pour ralentir et descendre en altitude, afin de laisser passer l'avion de devant et de continuer pendant un certain temps, créant ainsi une certaine distance. C'est-à-dire, après le temps d'attente, l'avion reprend sa vitesse et son altitude initiales, garantissant une distance de sécurité entre les avions en vol. Cela permet d'éviter la plupart des problèmes de collision.

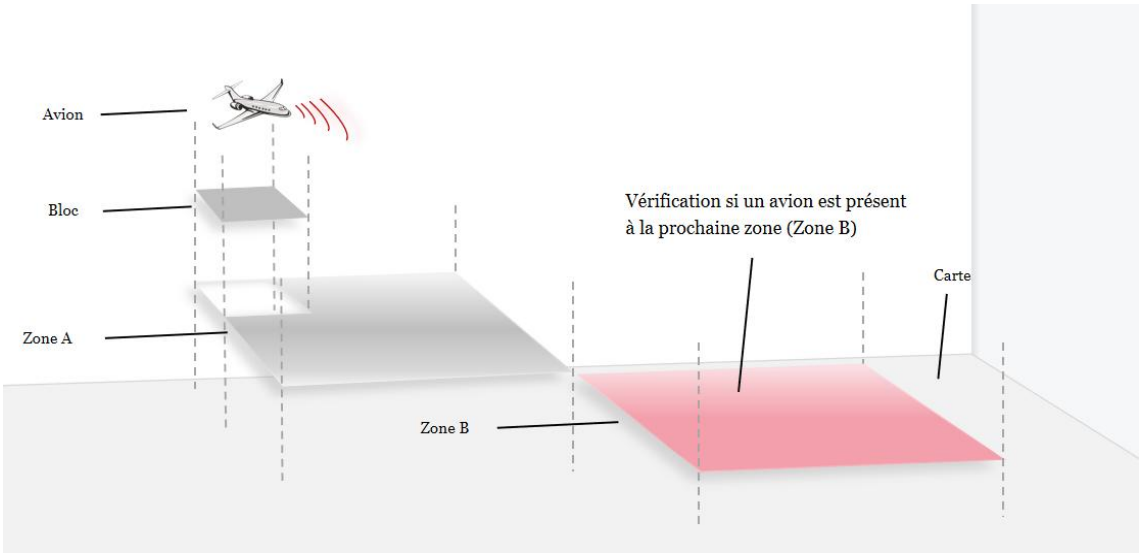


Figure 8 – Schéma représentant l'analyse d'une zone partagée

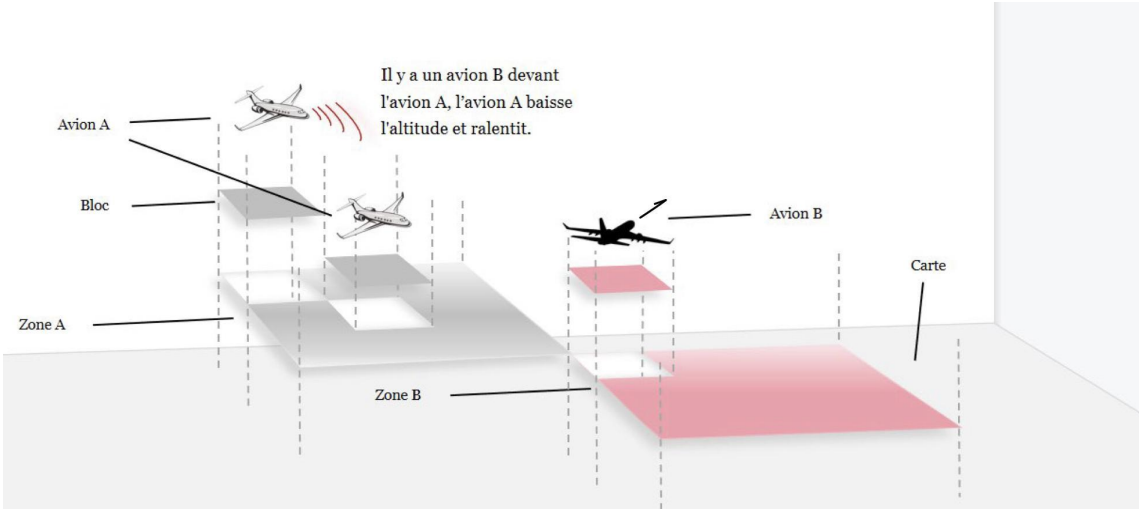


Figure 9 – Schéma représentant l'analyse d'une zone partagée

3.3.2 Conception des avions et les cas d'urgence

Les avions sont les éléments essentielle a notre simulation, elle réalise des déplacements entre différentes ville de maniéré automatique. Durant leurs vols, les avions peuvent tomber en panne et avoir une représentation spéciale, comme nous pouvons le voir ci-dessous.

Avions avec un problème

Avions sans problème

Pour créer ces avions, nous avons commencé par définir le nombre d'avions à partir du fichier de configuration GameConfiguration.java dans mobileElement.java. Puis, nous avons créé un avionManager pour chaque avion et nous les avons stockés dans une liste chaînée. Nous avons ensuite utilisé mainGui.java pour générer un thread par avionManager en multithreading. Chaque avionManager, qui héritait de la classe Thread, gérait un avion en stockant son heure de décollage, sa vitesse initiale, son point de départ et sa destination.

Dans la méthode run() de l'avionManager, nous avons contrôlé les actions de vol de l'avion, comme le changement d'altitude et de vitesse.

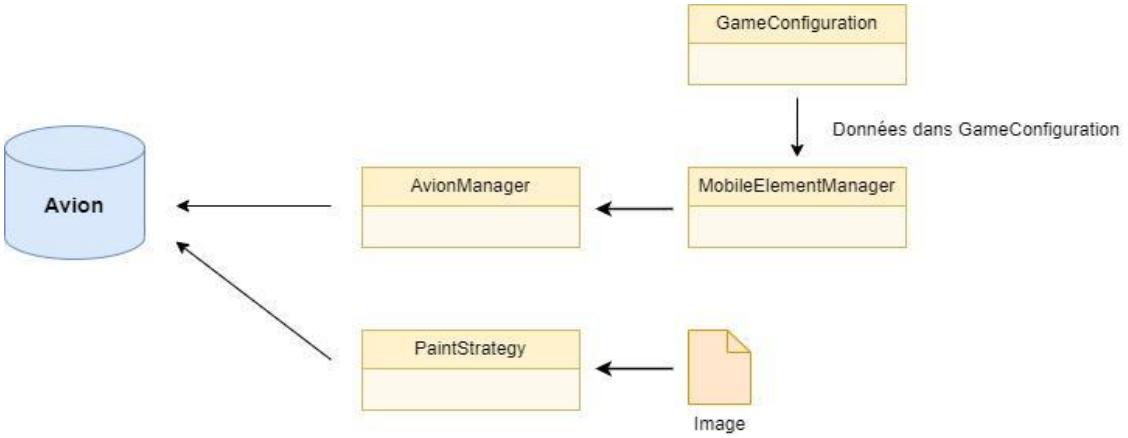


Figure 10 – Schéma représentant la création d'un avions

Pour le mouvement des avions, nous calculons les coordonnées horizontales et verticales des points de départ et de destination de l'avion et utilisons un algorithme pour modifier les coordonnées actuelles de l'avion en fonction de sa vitesse, réalisant ainsi le mouvement de l'avion. Par exemple, comme le montre l'image, lorsque l'avion se déplace vers la droite, nous modifions la position de l'avion en fonction du changement de blocs. Nous divisons la carte en blocs, dans le but de rendre le vol de l'avion plus fluide. Lorsque nous augmentons la vitesse de l'avion, nous modifions les blocs de déplacement de l'avion. Comme on peut le voir sur la figure 11, l'avion passe du déplacement d'un bloc à celui de deux blocs, ce qui permet de réaliser le changement de vitesse de l'avion. Pour les déplacements latéraux de l'avion, nous utilisons un algorithme qui modifie simultanément les coordonnées horizontales et verticales, permettant aux avions de se déplacer en diagonale.

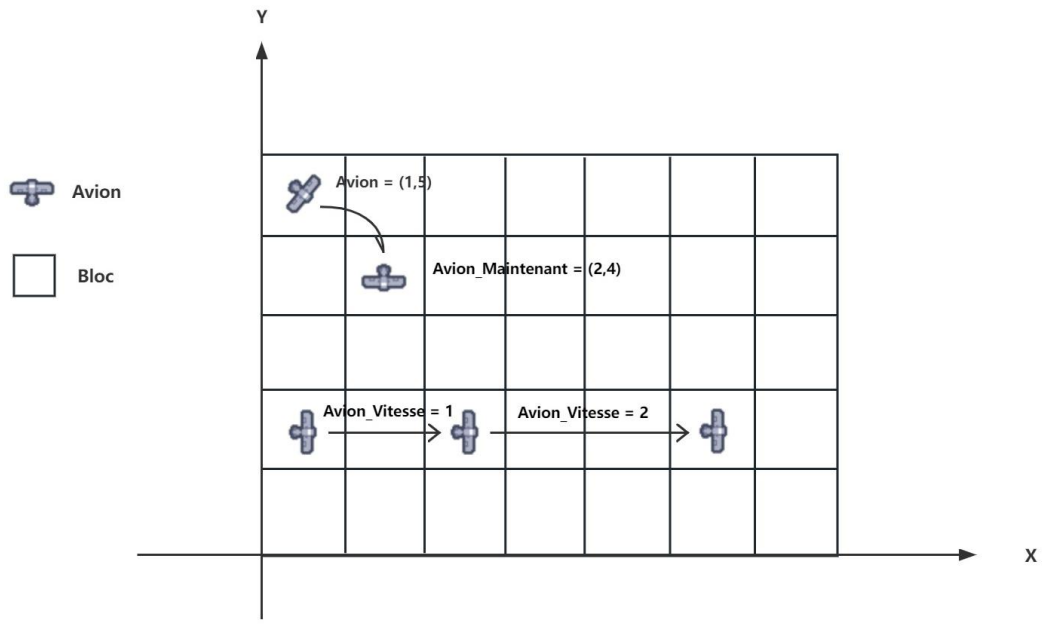


Figure 11 – Schéma déplacement des avions

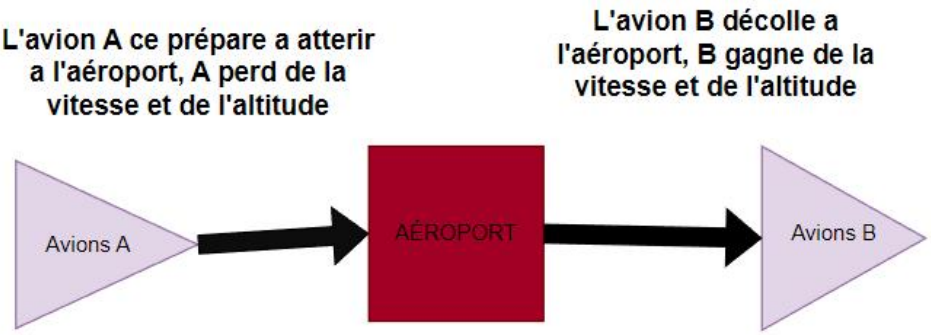


Figure 12 - illustant le décollage et l'atterisage d'un avion

Lorsque l'avion est sur le point d'arriver à l'aéroport (à 120 blocs de distance), afin de garantir un réalisme accru, nous réduisons progressivement la vitesse de l'avion et diminuons progressivement son altitude. Lorsque l'avion quitte l'aéroport, nous augmentons progressivement son altitude. L'altitude maximale de vol de l'avion est de 10 000, et l'avion se déplace en blocs par unité de temps.

Nous avons développé une fonctionnalité qui permet de simuler des cas d'urgence sur des avions dans notre programme. Le seul moyen d'interaction avec le programme est le bouton "Emergency landing". Lorsqu'un utilisateur clique sur ce bouton, un avion est sélectionné aléatoirement parmi ceux de la simulation. Sa couleur passe au rouge et il prend un statut particulier. La sélection de l'avion est effectuée dans la classe "MainGUI", tandis que le changement de couleur est géré par la classe "PaintStrategy". Une fois l'avion sélectionné, l'aéroport le plus proche est localisé en comparant les coordonnées de l'avion et celles des aéroports de la simulation. L'avion se dirige alors vers cet aéroport pour effectuer un atterrissage d'urgence. Toute cette logique est implémentée dans la classe "MainGIU".

Le schéma ci-dessous permet de voir les classes impliquer dans la création de panne pour la création de panne.

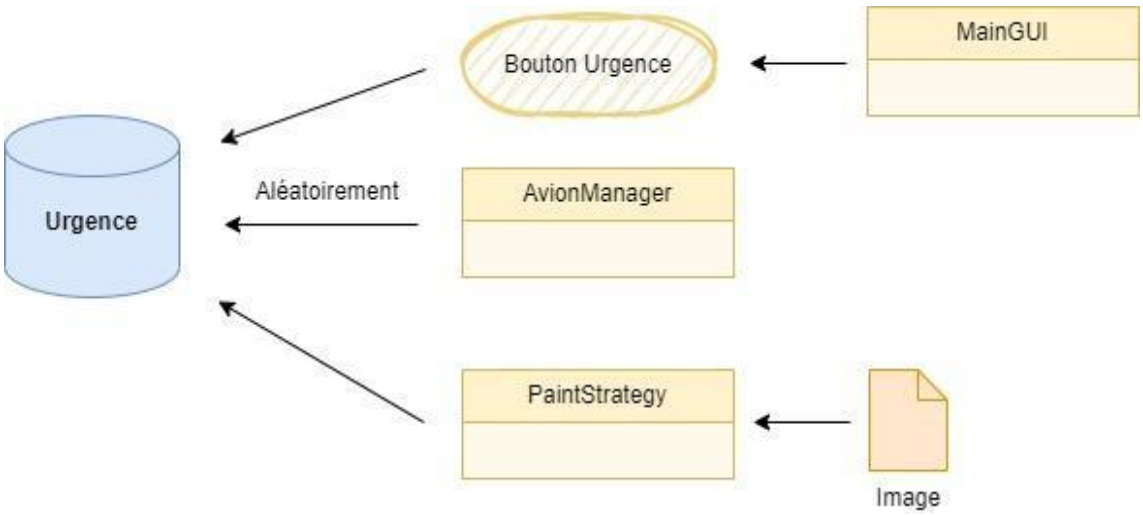


Figure 13 – Schéma représentant les classes impliquer

3.4 Conception de l'IHM graphique

Notre programme regroupe juste une fenêtre IHM divisée en 3 paties qui sont les suivantes.

- Une partie permet de visionner la simulation.
- Une partie qui permet de consulter les informations sur les vols.
- Une partie qui regroupe 2 boutons cliquables.

Vous pouvez retrouver ces 3 parties dans le schéma ci-dessous :

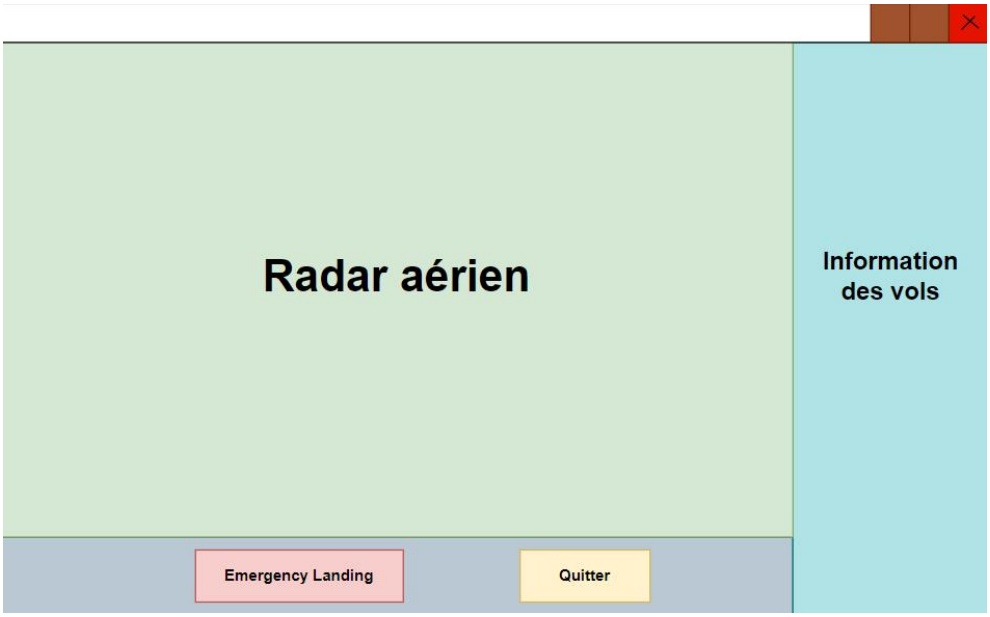


Figure 14 – Schéma illustrant l'IHM graphique de notre programme

4 Manuel utilisateur

Dans cette partie, nous verrons comment utiliser notre programme.

4.1 Exécution du programme

Les prérequis :

Avoir installé le JDK qui va permettre de compiler et d'exécuter.
Avoir un IDE Java (Eclipse).
Vous pouvez aussi directement cliquer sur le fichier .exe contenu dans notre archive compressée.

Les étapes :

- Assurez-vous d'avoir placé le projet dans votre workspace et d'exécuter l'IDE Eclipse.
- Allez dans la rubrique "File", puis cliquez sur "Open Projects from File System...".
- Sélectionnez "Directory" et sélectionnez le projet, puis cliquez sur "Finish".
- Allez dans le répertoire "/src/TestATC/TestATC.java", puis cliquez sur la rubrique "Run" et exécutez le projet en cliquant sur "Run".

4.2 Interaction avec le programme

Comme énoncé jusqu'à présent, la simulation est complètement automatique. L'utilisateur ne peut cliquer que sur ces deux boutons ci-dessous.

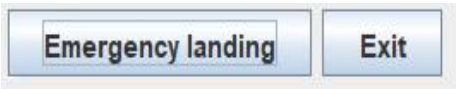


Figure 15 – Image représentant nos boutons

Ceci est une simulation automatique, seul le bouton "Emergency landing" permet réellement à l'utilisateur d'interagir via une souris.

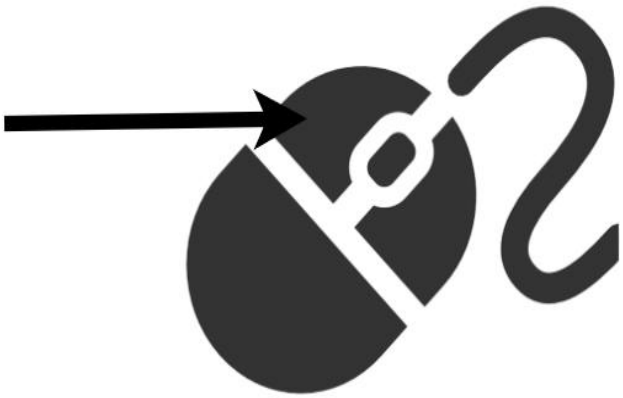


Figure 16 – La touche de la souris pour cliqué sur les boutons

La figure 14 présente une carte et des éléments essentiels et ainsi le panneau d'info.

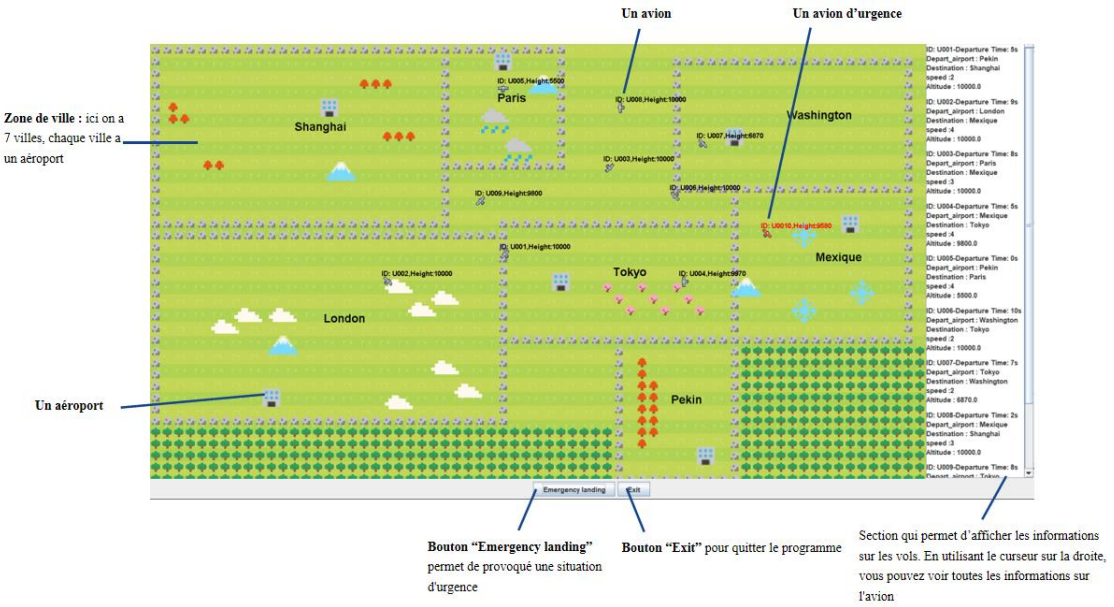


Figure 17 – Image illustrant l'interface utilisateur

5 Déroulement du projet

Dans cette section, nous décrivons comment le projet a été réalisé en équipe.

5.1 Réalisation du projet par étapes

5.1.1 Diagramme de Gantt

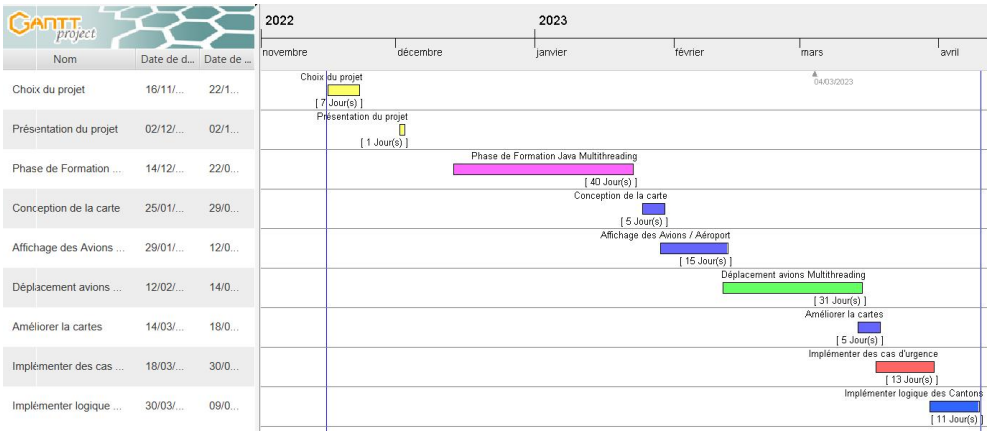


Figure 18 – Diagramme de Gantt

Vous pouvez retrouver ici un diagramme de Gantt qui regroupe toutes les phases importantes pour réaliser ce projet. Nous rappelons que nous avons suivi un modèle en cascade pour réaliser ce projet.

5.1.2 Répartition des Tâches

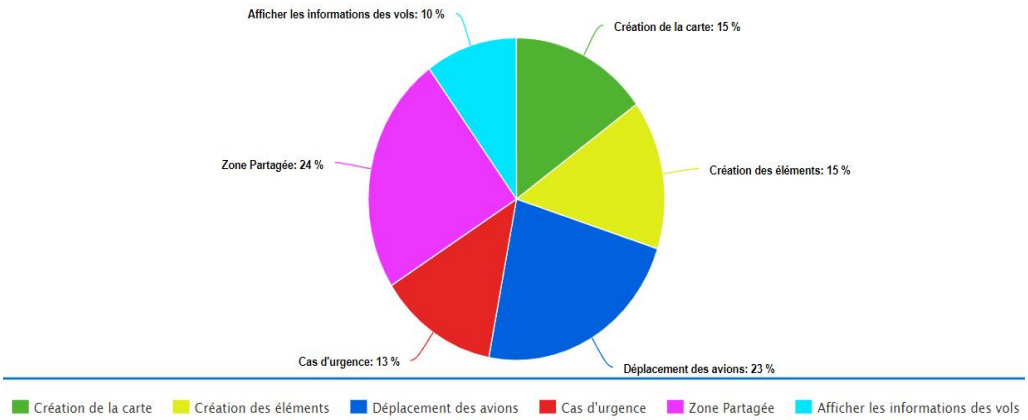


Figure 19 – Diagramme en camanbert illustrant la répartitions des tâches

La figure 19 représente la répartition des tâches de chacune des grandes étapes du projet.

5.2 Répartitions des tâches entre membres de l'équipe

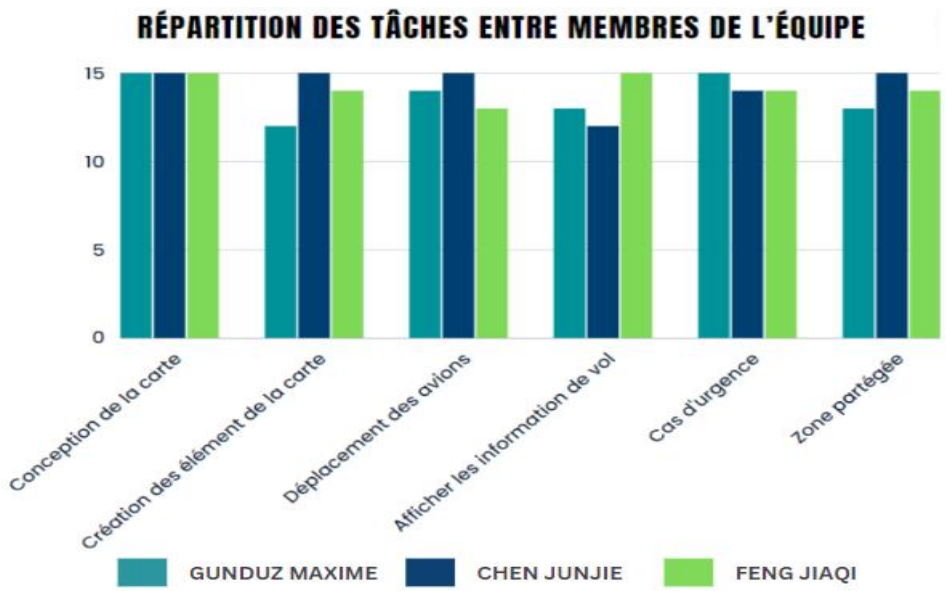


Figure 20 – Diagramme en barre illustrant la répartition des tâches entre membres de l'équipe

La figure 20 regroupe la répartition des tâches entre membres de l'équipe.

6 Conclusion et perspectives

Dans cette section, nous résumons la réalisation du projet et nous présentons également les extensionset améliorations possibles du projet.

6.1 Résumé du travail réalisé

Résumé du programme :

Notre programme est une simulation d'un radar aérien qui affiche les déplacements des avions entre les différents aéroports de manière automatique. L'utilisateur n'a pas besoin d'intervenir sur la simu- lation, sauf pour provoquer des cas d'urgence et observer le comportement des avions en situation d'urgence.

Les défis :

La gestion des collisions d'avions est un des problèmes que nous avons dû résoudre dans notre projet. Nous avons utilisé la classe "blockmanager" pour assurer qu'il n'y ait qu'un seul avion par zone de vol. Nous avons aussi donné à chaque avion la capacité de détecter les zones adjacentes à sa position. Ainsi, l'avion peut adapter sa vitesse s'il perçoit un autre avion devant ou sur le côté. Cependant, ce mécanisme a une faiblesse : lorsque deux avions provenant de directions différentes se dirigent vers la même zone libre, et qu'ils ont la même vitesse et la même distance à la zone libre, ils ont du mal à détecter ou à évaluer correctement si la zone libre est libre ou non.

Pour pallier ce problème, nous avons augmenté la portée de détection d'un avion, en lui permettant de détecter non seulement la zone devant lui, mais aussi la zone suivante. De plus, lorsque des avions d'autres directions se disputent la même zone, nous modifions l'altitude de l'avion pour prévenir les collisions. Cependant, malgré ces améliorations, il arrive parfois que l'avion connaisse des saccades en vol, peut-être à cause de problèmes systématiques lors de l'appel à wait() ou de l'évaluation logique. Ce problème n'est pas encore résolu.

6.2 Améliorations possibles du projet

Nous avons pu penser aux améliorations suivantes :

- Implémentation d'une fonctionnalité de sauvegarde et de chargement, d'une sauvegarde.
- Implémentation d'un bouton de vitesse pour la simulation.
- Ajout des effets sonores.

- Ajout de la possibilité à l'utilisateur de déterminer le nombre d'avions présents.
- Ajout de la possibilité à l'utilisateur de déterminer la position des aéroports.
- Permettre à l'utilisateur de prendre le contrôle d'un avion de la simulation.
- Permettre à l'utilisateur de choisir une carte.
- Permettre à l'utilisateur de choisir le design des avions.
- Prise en charge d'autres véhicules volants comme des hélicoptères.

Références

Nous avons utilisé plusieurs références pour réaliser ce programme que nous allons citer ici.

Projet « Train » de M.Liu : https://depinfo.u-cergy.fr/~tliu/gl/train.zip

Projet « Aircraft » de M.Liu : https://depinfo.u-cergy.fr/~tliu/ens/gpi/aircraft.zip