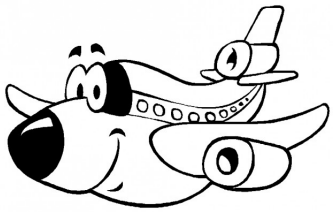

Dossier Conception Implémentation SMAirport

BALLEJOS Lilian, CHARPIN Etienne et DENIZOT Hugo
ISIMA INP
Année Universitaire 2022/2023

Enseignant référant : Mr HILL David
Date du rendu : 17 Avril 2023

ISIMA
1 rue de la Chebarde - TSA 60125 - CS 60026 - 63178 Aubière CEDEX
Tel : 04 73 40 50 00
Site web : isima.fr

Sommaire

1 Introduction

Nous allons vous présenter aujourd'hui dans ce dossier la conception programme de **SMA** (simulation multi-agents).

Il s'agit d'une simulation qui est composée de douaniers et de clients lambdas qui interagiront les uns avec les autres dans un aéroport.

Nous nous sommes concentrés sur les différents enregistrements que les clients doivent passer, notamment les files d'attente, les fouilles de sécurité et la déposition des bagages. Nous allons également prendre en compte les actions possibles pour les différents agents, comme les douaniers qui peuvent fouiller les clients et les arrêter s'ils ne sont pas en règle. L'objectif de ce projet est d'analyser les différents processus et les interactions entre les agents pour comprendre comment ils contribuent au bon fonctionnement de l'aéroport.

Remarque :
Notre projet est disponible sur le Gitlab de l'ISIMA INP en protection *internal* ce qui signifie que pour y accéder il faut être connecté à celui-ci.
Retrouver notre projet en suivant ce lien : ici

2 Choix du langage

Nous avons décidé de coder ce programme en **C++** pour deux raisons :

— Ce langage est performant car plus proche de la machine et sera donc plus efficace et plus rapide pour effectuer une SMA (des programmes qui sont parfois très lourd)
— Cela nous permettra de nous améliorer dans ce langage que nous étudions en ce moment en cours et sur lequel nous serons évalués.

3 Objectifs

Nous sommes dans la zone d'entrée d'un aéroport.
L'agent "Client" se déplace entre les différentes parties de l'aéroport et interagit avec celles-ci .

L'agent douanier lui interagit avec les clients en les fouillant et les arrête si ceux-ci ne sont pas en règle.

4 Implémentation

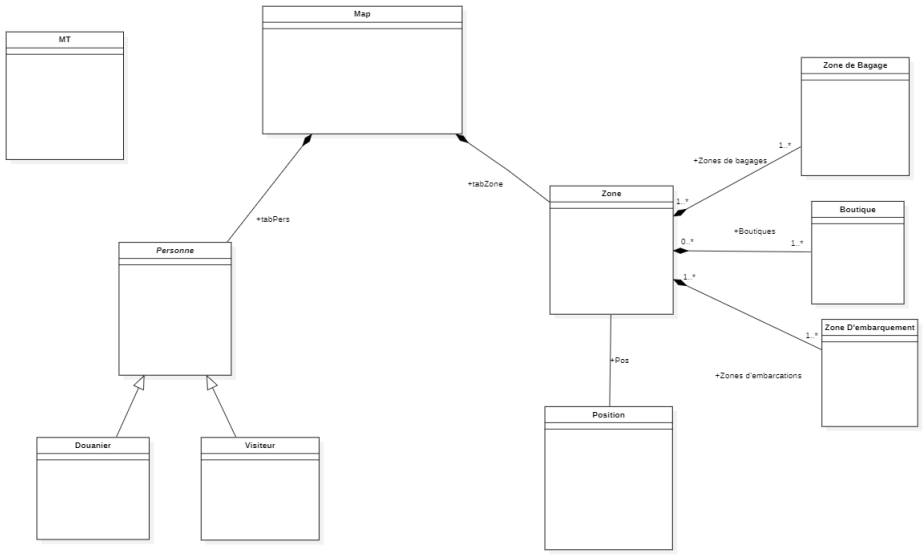

FIGURE 1 – Diagramme de classe de l'application

Pour ce qui est de l'implémentation de notre code, nous avons débuté par le code de nos classes mères. Nous avons dans un premier temps implémenté les classes Zone et Personne pour permettre de les faire hériter sur ZoneBagage, ZoneEmbarquement et Boutique pour l'un et Douanier et Visiteur pour l'autre. Une fois les prototypes de ces classes définis, nous avons choisi de coder la carte de notre aéroport qui se trouve être une grille de 20 cases par 20. Ensuite nous avons pris le temps de coder les déplacements des différentes personnes de notre simulation. Un douanier se déplace alors possiblement à chaque itération d'une case dans tous les sens et cherche si il possède des visiteurs à une case de lui. Si il possède des visiteurs dans son entourage, il en choisit 1 aléatoirement et a une certaine possibilité de le contrôler. Si il est en contrôle, il bloque ses déplacements pendant 2 itérations ainsi que celles du visiteur.

Concernant l'implémentation de la gestion du tirage aléatoire, nous avons de nouveau utilisé les classes de Mersenne-Twister afin de travailler avec un générateur de nombre pseudo aléatoire efficace.

Pour le déplacement du visiteur, le principe est sensiblement le même. Comme nous assignons à sa création un temps avant envol, nous avons considéré que le visiteur peut aller visiter une boutique si le temps lui permet, puis devra déposer ses bagages avant d'aller à la zone d'embarquement. Dans sa course, le visiteur peut donc être contrôlé par un douanier ce qui peut potentiellement lui faire rater son avion. C'est cette statistique que nous étudieront dans nos résultats en fin de rapport. Lorsque le visiteur possède un temps suffisant pour aller à une boutique, nous tirons aléatoirement une boutique vers laquelle le visiteur se déplacera.

Sur l'image ci-dessous, le X représente un visiteur, le D un douanier, le B une boutique, le L un zone de dépot de bagages et enfin le Z pour la zone d'embarquement.

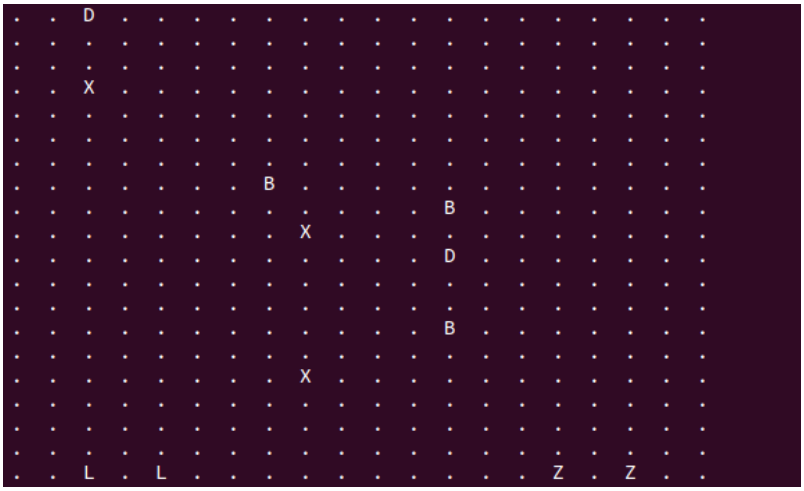

FIGURE 2 – Représentation graphique de l'application

5 Patrons de conception

Pour ce projet, nous devions choisir et implémenter un ou plusieurs patrons de conception nécessaire à notre développement. Nous avons choisi par nécessité le patron de conception Singleton.

5.1 Singleton

5.1.1 Présentation

Le Singleton est un patron de conception de création qui garantit que l'instance d'une classe n'existe qu'en un seul exemplaire, tout en fournissant un point d'accès global à cette instance.

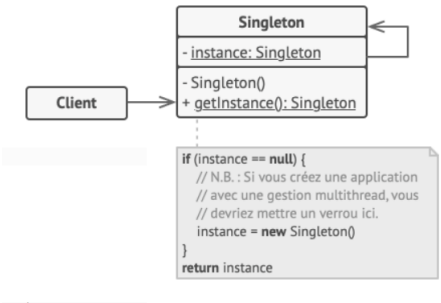

FIGURE 3 – Diagramme de classe du singleton

5.1.2 Implémentation

Dans notre projet, après la mise en place de la conception, nous nous sommes rendu compte de la nécessité d'implémenter un objet Map représentant la carte de l'aéroport. Cette carte se devait d'être disponible pour l'ensemble des agents ainsi qu'aux objets la demandant. En c++, afin de déclarer un objet global, nous devons tout d'abord le déclarer dans la fonction main afin de la créer puis il nous faut ensuite déclarer dans chaque classe nécessitant la Map l'intitulé **extern Map m**.

```
using std::endl;
using std::cout;

Map m{};

int main(int, char **) {
```

FIGURE 4 – Déclaration de notre Map

6 Résultats

Afin d'ajouter une utilité à cette simulation multi-agent, nous avons implémenté un petit programme annexe afin de vérifier sur une journée le nombre de personnes ratant leur avion. Nous mettons de coté ceux qui le rate à cause d'un contrôle de douanier. Pour ce faire, nous avons implémenté une journée de 86400 itérations dans un aéroport comportant 2 douaniers 3 boutiques ainsi que 2 zones de bagages et 2 zones d'embarquement. Nous afficherons alors le nombre de personnes ratant leur avion sur le nombre total de personnes venues dans l'aéroport puis nous isolons le nombre de contrôle ainsi que le nombre d'avions ratés à cause d'un contrôle.

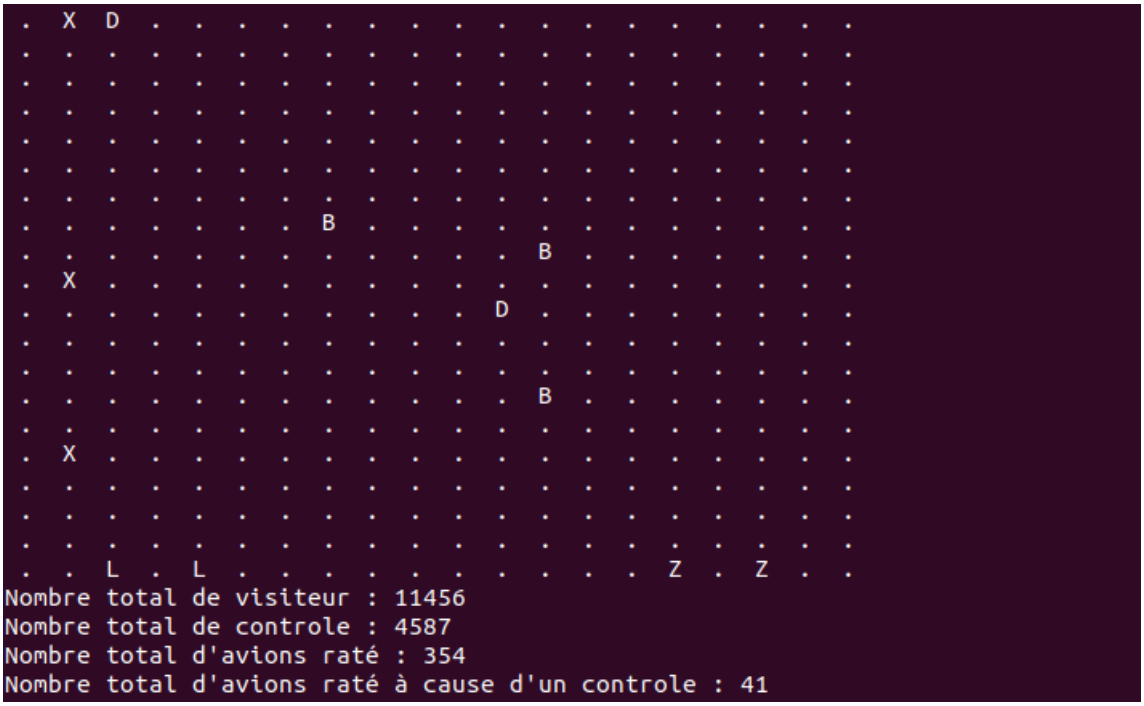

FIGURE 5 – Affichage d'une itération type

Nous voyons ci-dessus l'affichage d'une itération type de notre simulation. Pour obtenir des résultats cohérents, nous faisons 100 itérations de cette simulation et calculons alors la moyenne.

A la fin de nos itérations, nous voyons qu'en moyenne, 321 personnes ratent leur avions sur 11632 dont 43 à cause d'un des 4613 contrôles de la journée ce qui correspond à 0.02 et 0.003 pour-cent. Si nous faisions cette simulation pour le compte d'un aéroport, nous pourrions conclure que les douaniers sont responsables d'un nombre assez important de manque d'avions dans un journée type avec ces paramètres.

7 Changement de Conception

Malgré une analyse et une conception poussée en aval de notre SMA, nous avons tout de même légèrement modifié nos diagrammes prévisionnels que nous allons vous présenter ci-dessous.

La première modification concerne le diagramme de gantt de notre projet.

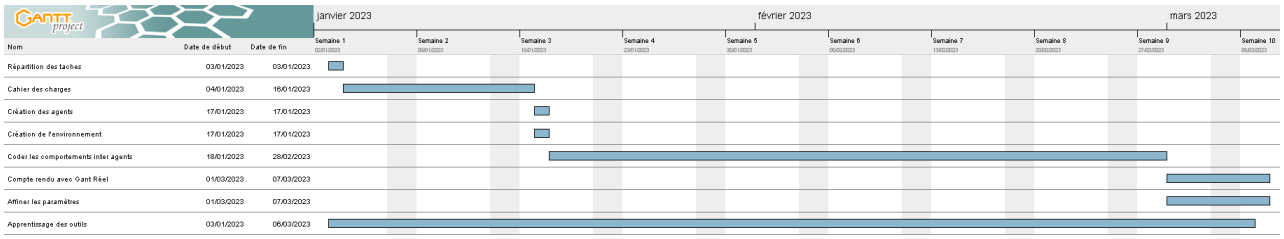

FIGURE 6 – Diagramme de Gantt prévisionnel

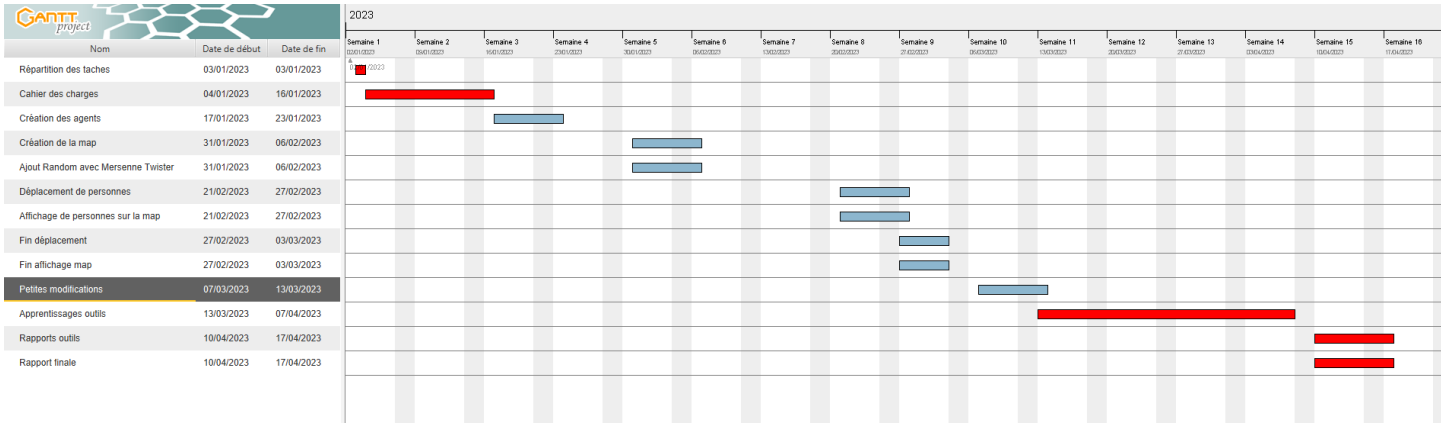

FIGURE 7 – Diagramme de Gantt réel

Comme nous pouvons le voir en comparant les deux diagrammes ci-dessous, nous avions un peu sous-estimé le temps que nous possédions pour pouvoir développer ce projet. Pour résoudre ce problème, nous avons étendu les taches de notre projet jusqu'à la date limite. Nous avons aussi séparer la plupart de nos taches pour pouvoir gagner du temps en se les répartissant de manière optimisée. Enfin, nous avons concentré notre temps à la gestion de nos outils à étudier après avoir fini le code de notre simulation multi-agent.

Pour ce qui est du diagramme de classe et d'utilisation de notre conception, ils se sont révélés être plutôt précis par rapport à notre conception final. La seul différence étant la gestion de l'aéroport qui a évolué d'une classe à un objet carte disponible pour l'ensemble des objets.

8 Conclusion

Pour conclure sur ce projet de simulation multi-agent, nous avons eu la possibilité de travailler et améliorer de nombreuses compétences tout au long de notre projet. La partie analyse et conception nous a permis de travailler et de mettre au gout du jour notre vision de la représentation UML par la création de diagramme de Gantt, de classe mais aussi de cas d'utilisation. Ainsi nous avons pu nous donner quelques indications représentant un cahier des charges sur un projet ou nous étions le concepteur et le client. Ensuite, lors de l'implémentation, nous avons eu la chance de travailler en c++ ce qui nous a donné un temps d'avance sur les révisions de la matière qui était présente au second semestre. Enfin, nous avons tous eu la chance de pouvoir découvrir et d'approfondir un outil d'aide à la conception de tels programmes.

Table des figures