# DM Système 3A – Simulation d'un réseau ferroviaire

## Principe

Le but de ce DM est de simuler un réseau ferroviaire durant une journée complète. Ce réseau ferroviaire est constitué de train en circulation sur plusieurs lignes. Ces trains desservent des gares. Des passagers présents sur la ligne veulent emprunter le réseau pour voyager dans un but touristique, c’est-à-dire qu'il parte d'un point quelconque du réseau pour une destination quelconque et proportionnel à leur capacité financière.

## Description des différents composants

### Réseau

Le réseau n'est pas représenté dans le code mais représente l'ensemble des trains, passagers et gares.

### Lignes

Une ligne représente un ensemble de Gares et de Trains. Dans le projet, **5 lignes** sontreprésentées : Frisson, Magie, Grand Tour, La citadine et La Paysanne. Chaque ligne comporte 5 trains et 5 gares. Afin d’assurer des éventuels échanges de passagers entre différentes lignes, certaines gares sont en commun avec plusieurs lignes (voir plus bas schéma 2).

### Trains

Un train est le seul moyen de locomotion des passagers entre deux gares. Dans le programme, un train est représenté par un thread. La structure thread permet aux trains d'être totalement indépendant les uns des autres et de «vivre » individuellement.

### Gares

Une gare représente l’ensemble des quais permettant d’accueillir les trains circulant sur la ligne, ainsi que l’ensemble des guichets présent dans ce lieu pour permettre au voyageur de payer son billet.

## Déroulement prévue

Suite à un manque de temps dû à un emploi du temps très chargé, soutenances de fin d’année, rapports et examens qui s’enchainent, nous n’avons pas pu implémenter toutes les fonctionnalités prévues. Voici le déroulement du programme prévu :

* Lors du démarrage du programme, une tableau de voyageur est créée en fonction des paramètres fournis (montant Max et montant Min).
* Créer une liste de voyageur en fonction des params fourni = >attribution d’un portemonnaie compris entre vMAX vMin.
* Créer une map avec les lignes qui comprennent les trains et des gares attritrés à chaque ligne. On est sensé répartir les voyageurs entre les différentes gares.
* Au moment de payer au guichet (voyageur), une carte personnelle est attribuée => liste de gare par lesquelles ils passent, la liste sera crée à partir d’une recherche parmi toute les gares pour constituer les changements de Lignes si nécessaire. Lorsqu’on établi sa liste il va payer en fonction du nombre de gare qu’il va traverser sur cette ligne avant de faire un changement et que l’étape se répètera à chaque fois à qu’il sort d’un train et il recommence à nouveau au guichet avec une nouvelle MAP.
* Regarder au niveau du voyageur si la prochaine gare à laquelle il doit aller est différente de la gare de la ligne. Si c le cas il va effectuer un changement pour aller à la ligne souhaitée.
* Répéter les étapes tant qu’il n’est pas arrivé à sa destination. On enlève le problème de sous parce qu’un voyageur ne peut aller qu’à des destination financièrement accessible.
* Un sémaphore est placée sur chaque guichet de gare et il y a entre 5 et 10 par gare en aléatoire. Les guichets fonctionnent donc de manière indépendants.
* Le maximum de train peuvent être dans une gare. Pas de limitation technique/semaphore.
* Lorsqu’un train est dans une gare et des voyageurs souhaitent monter dans le train. Un semaphore est utilisé pour réguler la montée, 10 passager au max. Car on a 5 wagons avec 2 portes par train.
* Les trains et voyageurs sont représentés par des threads qui vont donc communiquer entre eux pour savoir à quelles gares ils sont => grâce à une mémoire partagées (tas variable globale) accessible au thread train et voyageurs. Une vérification est faite pour que le voyageur qui voit un train de sa ligne ne le prenne pas si ce dernier ne va pas dans le sens souhaité (MAP).
* Un train attend 10 min simulée en gare.
* Chaque train est placé à des gares différentes en choisiant arbitrairement s’ils sont dans le sens aller ou retour. Arrivé en bout de lignes ils changent de direction
* Chaque heure simulée un bilan est rendu sur l’état de l’ensemble du réseau => Lignes => Gares dont la recette et le nombre de voyageur qui est passé par la gare. + trains

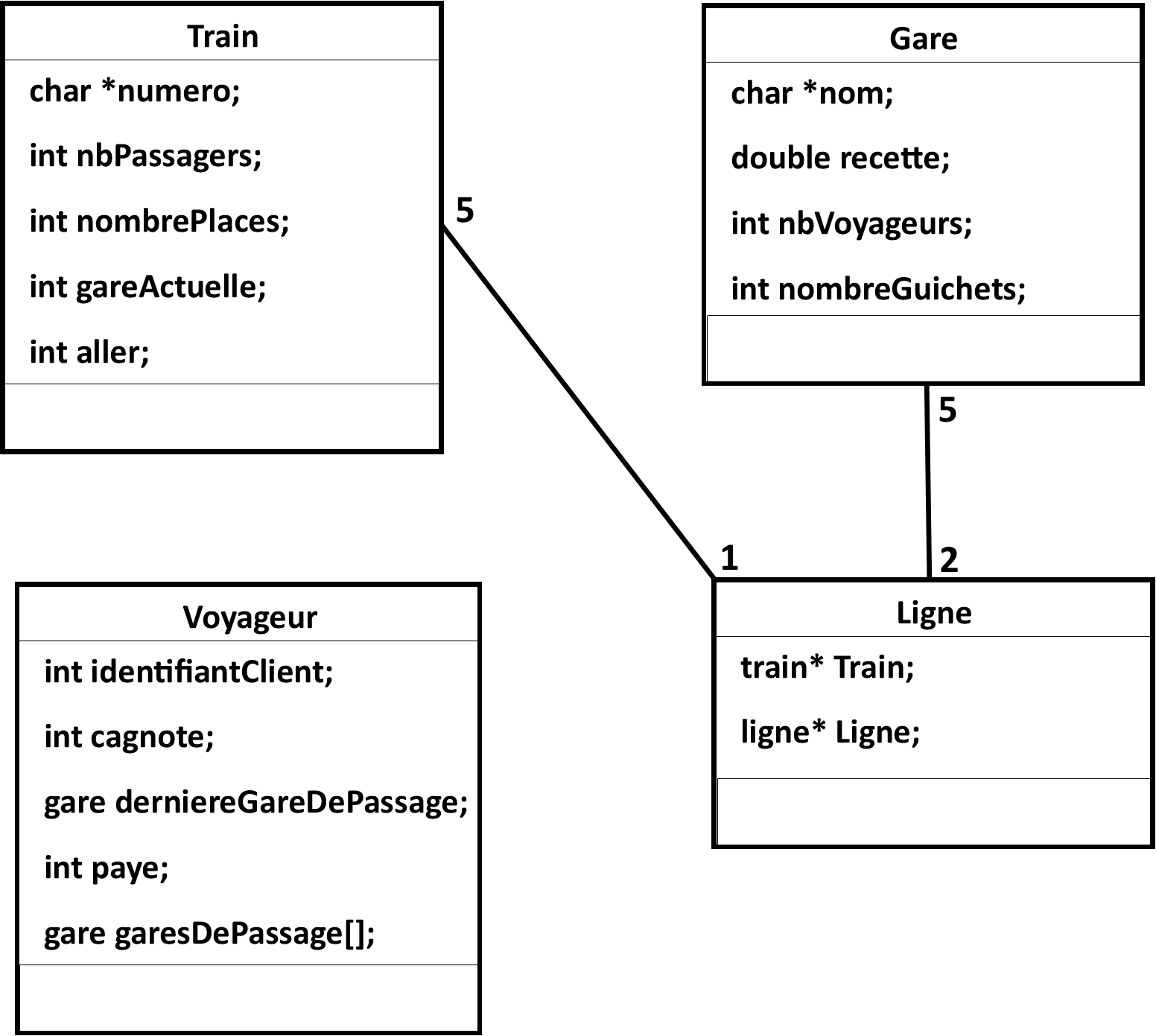
### Structures des composants

|  |  |  |
| --- | --- | --- |
| Nom | Structure | Explications |
| ligne | **struct** ligne  {  char \*nom;  train trains[5];  gare gares[5];  }; | Chaque ligne a un nom, un tableau de Train et de Gare de taille 5 |
| train | **struct** train  {  char \*numero;  int nbPassagers;  int nombrePlaces;  int gareActuelle;  *//1 si trajet aller 0 si retour*  int aller;  }; | Chaque train a un numéro généré de manière aléatoire, un nombre de passager, un nombre de place/capacité, l’indice de la gare où se situe le train ou la dernière gare visitée s’il est en déplacement, ainsi qu’un Int pour indiquer la direction du train sur la ligne. |
| gare | **struct** gare  {  char \*nom;  double recette;  int nbVoyageurs;  int nombreGuichets;  }; | Chaque gare a un nom qui lui est attribué depuis un tableau entré en dur contenant des noms de gares ; une recette pour avoir la recette en fin de journée, le nombre Voyageur, un nombre de guichet par gare qui est attribué de manière aléatoire entre 5 et 20. |
| voyageur | **struct** voyageur  {  int identifiantClient;  int cagnotte;  gare derniereGareDePassage;  int paye;  *//Si O non payé sinon billet payé*  gare garesDePassage[];  }; | Chaque voyageur a un identifiant client, une cagnotte avec un montant initiale assigné de manière aléatoire, une dernière gare qui est assigné de manière aléatoire ; payé qui est un booléen permettant d’avoir l’était de paiement du billet et le tableau gare de passage qui est rempli en fonction du montant disponible dans la cagnotte lors du paiement du billet. |

|  |  |  |
| --- | --- | --- |
| Nom | Structure | Explications |
| contenerTrainGares | **struct** contenerTrainGares  {  train\* Train;  ligne\* Ligne;  }; | Le contener est une structure qui contient un pointeur de train et un pointeur vers sa ligne. Ce contener sera envoyé à la fonction qui permet de faire avancer les trains. |

## Représentation des différentes structures

Ceci n’est pas un diagramme de classe en UML car il ne s’agit pas véritablement de classes. A prendre à titre représentatif des structures et de leurs dépendances.



## Problèmes rencontrés

* **Durée de simulation** : Une des consignes du DM est de simuler le réseau ferroviaire durant une « journée ». Pour des choix pratiques, nous posons qu’une **journée de simulation** correspond à **une minute en temps réel**. Une fois cette correspondance posée, nous avons besoin d’un ratio pour passer du temps réel mesuré au temps simulé. Ce ratio s’exprime en produit en croix :

Nous obtenons donc un ratio de **1./(24\*60)** après calcul complet**.**

Nous avons essayé une première fonction d’obtention du temps : clock(). Cette méthode nous donne bien le temps écoulé en seconde. Or, notre ratio étant très petit lors de la conversion temps réel vers temps simulé donc nous avions peu de précision sur les secondes simulées : les secondes mesurées étant multipliées par des nombres très grands, les secondes restent à 0.

Pour y remédier, nous utilisons une fonction de mesure de temps en microseconde, de sorte que nous ayons une précision à la seconde dans le temps simulé. Le code ci-contre donne le temps écoulé à une précision en microseconde.

* **Répartition des gares sur les lignes** : Afin, de permettre une situation de « réseau » avec différentes gares communes entre 2 lignes, il a été décidé d’avoir une gare en commun entre la Ligne N et la Ligne N+1. Afin, de répartir de manière équitable les 20 gares sur les 5 lignes, on a procédé à une répartition en chevauchement : la dernière gare attribuée à la ligne N est aussi la première gare attribuée à la ligne N+1.
* **Représentation des gares** : Afin, de réguler le trafic des trains dans une gare, au niveau des quais, nous avons besoin d’un semaphore. Les gares sont donc représentés avec des sémaphores qui seront réquisitionné lors du passage des trains dans la gare et l’embarquement/débarquement des passagers.
* **Représentation des trains**: Les trains doivent être autonomes et aller d’une gare à une autre de manière parallèle. Pour les représenter, nous avons choisi l’utilisation des threads. Chaque train à un thread pour qu’il puisse avancer.
* **Container pour les fonctions des threads des trains** : les trains vont être traités dans une fonction appelé lors de la création du thread. Pour fournir le train de la ligne et la Ligne à laquelle il appartient, nous avons créé une structure *contenerTrainGares* qui va permettre à la fonction *AvancementTrain* de pouvoir modifier les trains via des pointeurs.
* **Gestion payer billet voyageur**: Chaque

Gestion de trajet par voyageur création du parcours voyageur avec changement de ligne

## Description des fonctions

|  |  |  |
| --- | --- | --- |
| Nom | Structure | Explications |
| getMicrotime() | long double getMicrotime(){  **struct** timeval currentTime;  gettimeofday(&currentTime, NULL);  **return** currentTime.tv\_sec \* (int)1e6 + currentTime.tv\_usec;  } | Renvoie le Temps actuel en microseconds |
| ratioMinsEnMs() | */\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\*\* RATIO*  *\*\*Inputs : nombre de minutes de la vie réelle (float)*  *\*\*Ouputs : nombre de microseconde pour la simulation (float)*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  double ratioMinsEnMs(double nbMinutes){  **return** nbMinutes \* RATIO\_MINUTE \* 60 \* 1000000;  } | Dans un souci de simulation de la circulation sur une journée simulée, un ration a été calculé pour que le temps d’exécution d’une minute représente 1 journée simulée. |
| afficheTemps | */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\*\* affichage du temps*  *\*\*Inputs : nombre de secondes (float)*  *\*\* i (int) si 1 affiche l'heure, sinon affiche l'heure toute les heures.*  *\*\*Ouputs : Temps affiché (char\*)*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  void afficheTemps(double diff,int i){  */\*\* Code ICI \*\*/*  } | Affiche le temps en fonction de la durée *diff* entré en paramètre. Elle va donc décomposer le temps. |

|  |  |  |
| --- | --- | --- |
| nbAleatoire | */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\*\*Génère proprement un nombre aléatoire dans un interval*  *\*\*Inputs : debut intervalle (float), fin intervalle (float)*  *\*\*Ouputs : nombre aléatoire (float)*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  double nbAleatoire(double deb, double fin){  */\*\* Code ICI \*\*/*  } | Prends en paramètre deux nombre et renvoie une valeur double aléatoire dans cet intervalle. |
| numTrainRandom | */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\*\*Génère un char contenant le numéro aléatoire d'un train*  *\*\*Inputs :*  *\*\*Ouputs : nombre aléatoire entier sous forme de chaine de caractère de taille 6*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  char\* numTrainRandom(){  */\*\* Code ICI \*\*/*  } | Rempli un tableau de char\* de valeur aléatoires comprises entre le caractère ‘0’ et ‘9’ en ASCII. |
| trainDisp | */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\*\* affiche l'état du train*  *\*\*Inputs : Train*  *\*\*Ouputs :*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  void trainDisp(train Train){  */\*\* Code ICI \*\*/*  } | Affiche l’état d’un train en parcourant tout ses éléments membres. |
| gareDisp | */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\*\* affiche l'état de la gare*  *\*\*Inputs : Gare*  *\*\*Ouputs :*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  void gareDisp(gare Gare){  */\*\* Code ICI \*\*/*  } | Affiche l’état d’un train en parcourant tous ses éléments membres. |
| LigneDisp | */\*\*\*\*\*\*\*\*\*\*\*\**  *\*\* affiche l'état d'une ligne*  *\*\*Inputs : Ligne*  *\*\*Ouputs :*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  void LigneDisp(ligne Ligne){  */\*\* Code ICI \*\*/*  } | Affiche l’état d’une ligne en parcourant tous ses éléments membres. |
| payerBillet | */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\*\* file d'attente des passagers à la billeterie*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  void\* payerBillet (void\* infos) {  */\*\* Code ICI \*\*/*  } | Fonction appelé pour faire payer les billets aux passagers. |
| initSemaphoreGare | */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\*\* Initialise les semaphores des gares*  *\*\*Inputs :*  *\*\*Ouputs :*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  void initSemaphoreGares(){  */\*\* Code ICI \*\*/*  } | Initialise le tableau de sémaphore qui représente les gares. |
| TrainArriveGare | */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\*\* gère les entrées de gares*  *\*\*Inputs : infos => un train et sa prochaine destination de gare en char\**  *\*\*Ouputs :*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  void\* TrainArriveGare(void\* infos)  {  */\*\* Code ICI \*\*/*  } | Pour l’inst |