# 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

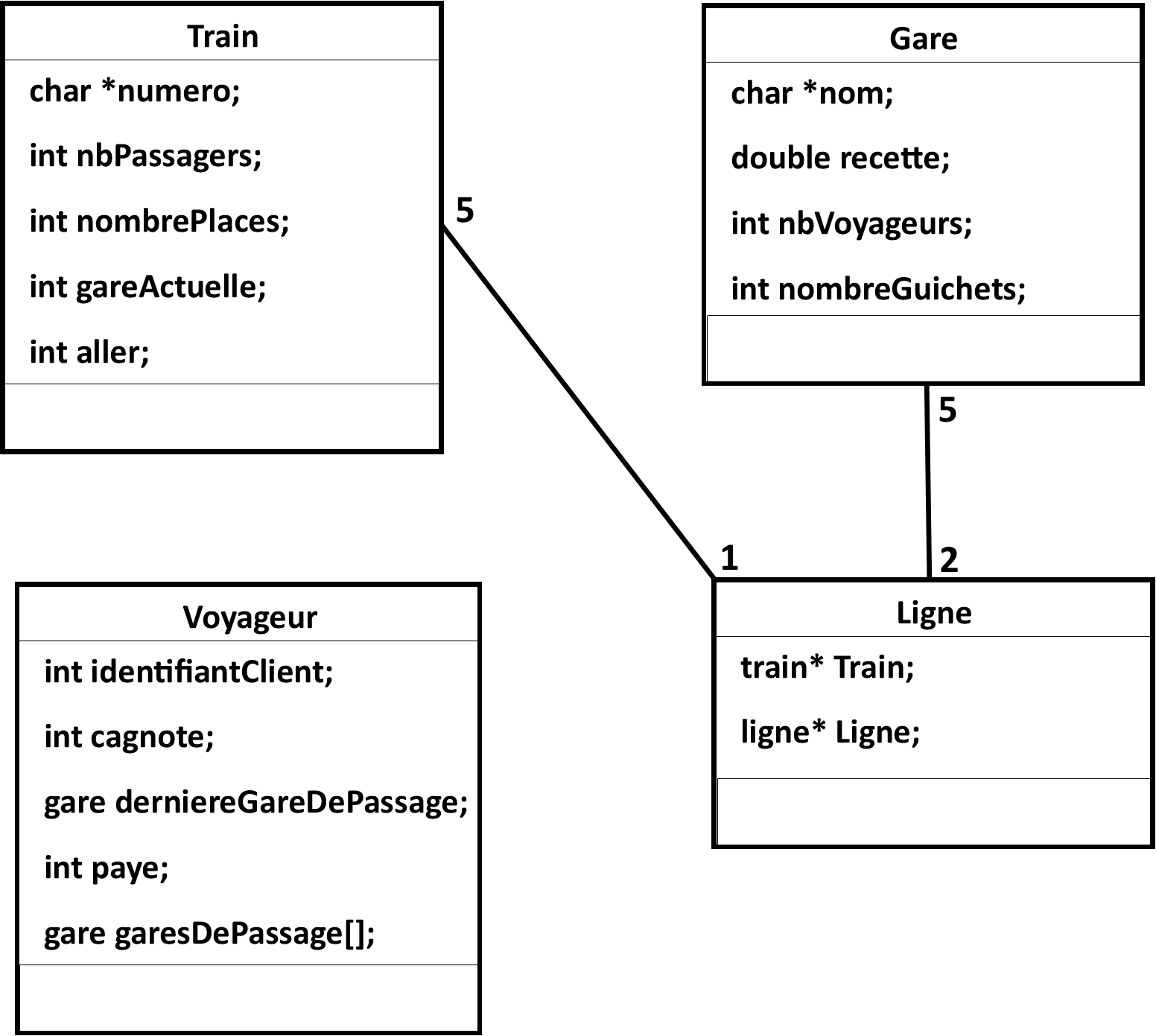
Une gare représente l’ensemble des quais permettant d’accueillir les trains circulant sur la ligne, ainsi que l’ensemble des guichets présent dans ce lieu pour permettre au voyageur de payer son billet.

### 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. |

## Diagramme UML



## 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 |

## Problèmes rencontrés

* Durée de simulation
* Chronomètres
* Contener pour la fction avance train