

Réalisation d’un réseau de Tramway



**Membres du groupe :**

**SCHALLER Elma, NORMANT Thomas, FOLCHER Lukas, PETITDEMANGE Gaulthier**

**Professeur tuteur :**

**CORDIER Frédéric**

**2019/2020**

**Sommaire :**

[**I.** **Introduction** 3](#_Toc36926469)

[1. Objectifs et cahier des charges 3](#_Toc36926470)

[2. Environnement de compilation 3](#_Toc36926471)

[**I.** **Fonctionnement de l’interface graphique** 4](#_Toc36926472)

[**II.** **Organigramme et contenu du programme** 7](#_Toc36926473)

[1. Organigramme des classes 7](#_Toc36926474)

[2. Description des classes 8](#_Toc36926475)

[a) Arret.h , ListeArret.h et ChainonArret.h 8](#_Toc36926476)

[b) Tramway.h, ListeTram.h et ChainonTram .h 9](#_Toc36926477)

[c) Ligne.h 10](#_Toc36926478)

[**III.** **Choix de l’organisation du programme** 11](#_Toc36926479)

[**IV.** **Contribution des étudiants** 12](#_Toc36926480)

[Répartition 12](#_Toc36926481)

[**V.** **Conclusion** 12](#_Toc36926482)

# **Introduction**

## Objectifs et cahier des charges

Les objectifs de ce projet consistaient à modéliser un réseau de tramway et à le gérer à travers une lecture de fichier.

Pour cela, il était demandé (et nécessaire) de gérer plusieurs informations, notamment :

* La gestion des arrêts (nom, distance qui les sépare, liste des trams qui se déplacent dessus).
* La gestion des tramways (durée d’arrêt sur une station, la ligne affectée, leur vitesse, leur distance minimale pouvant les séparer, leur sens ainsi que leur position sur le réseau).
* Certaines consignes : Dépassement de tram impossible, direction des trams qui changent au terminus, simulation en temps réel et affichage du réseau en 2D.
* Utilisation des listes chaînées et des méthodes liées.

## Environnement de compilation

Nous avons choisi de coder ce programme sous Code::Blocks puisque toutes les personnes du groupe maîtrisaient cet IDE, l’ayant vu et nous étant exercé dessus durant les cours.

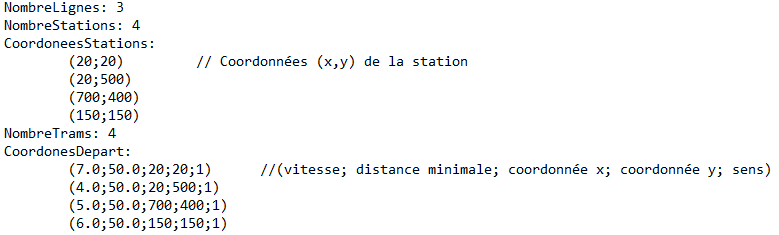
En ce qui concerne la bibliothèque graphique, nous avons choisi WinBGIm pour les mêmes raisons.

Le projet s’est accompagné de l’utilisation d’un gestionnaire de version, appelé GitHub, afin de mettre en commun nos travaux et de faciliter la communication. Nous avons aussi communiqué à travers Discord, que ce fut-ce en vocal ou à l’écrit.

# **Fonctionnement de l’interface graphique**

Notre programme démarre d’abord par la lecture d’un fichier texte afin de mettre en place l’interface graphique suivant les variables précisées. Ce fichier nommé « donneesTram » se présente de la manière suivante :

Tout d’abord, le nombre de lignes à placer. Puis, pour chaque ligne, le nombre de stations qui la composent, puis le nombre de tramways par station. On entre ensuite les coordonnées des stations suivies des variables concernant les tramways qui seront placés dessus. Dans l’ordre : leur vitesse, la distance minimale les séparant, leurs coordonnées de départ et leur sens.



Une fonction s’occupe alors de lire le fichier dans l’ordre et une autre fonction les affiche au lancement du programme. On place d’abord les arrêts d’une taille de trente sur trente pixels, puis les lignes qui vont relier les arrêts (en vert) et enfin les tramways qui seront représentés par un carré rouge pour un sens et vert pour un autre, de taille dix sur dix pixels.

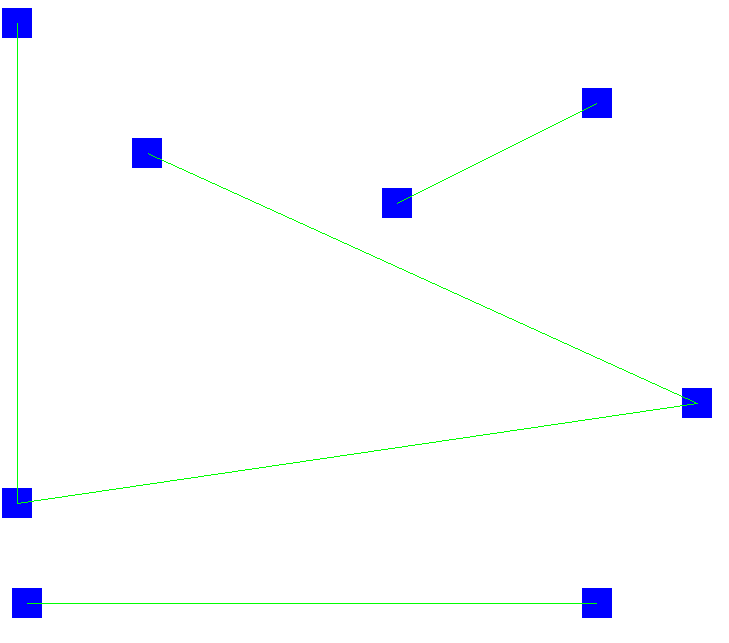


Figure  : Affichage des stations et des lignes les reliant.

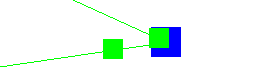
Après le lancement du programme, les tramways vont commencer à se déplacer à leur vitesse prédéfinie.



Figure  : Affiche des tramways sur le réseau.

Ils vont alors se conduire de différentes manières suivant la situation :

* Si le tramway s’approche trop d’un autre tramway, il va marquer une pause avant de continuer à avancer.
* Si le tramway arrive à une station, il va s’arrêter un temps avant de reprendre sa route.



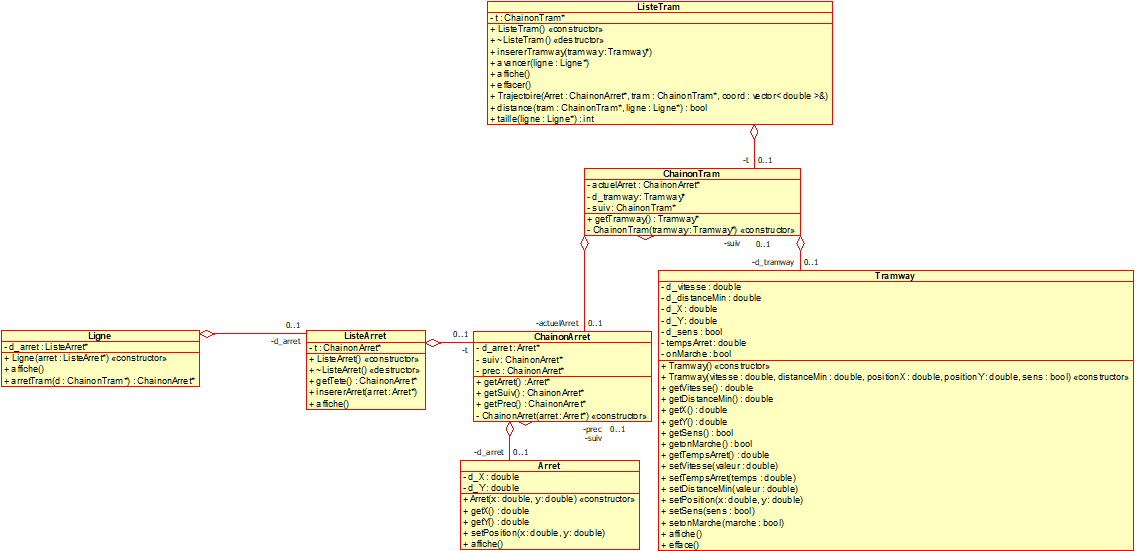
* S’il s’agit d’un terminus, il change de sens en changeant de couleurs. Sinon, il continue jusqu’à la prochaine station.



Les tramways se déplacent de la manière suivante : On définit leur future position, on efface le tramway et sa ligne et on les affiche à nouveau dans leur nouvelle position.

# **Organigramme et contenu du programme**

## Organigramme des classes[[1]](#footnote-1)



## Description des classes

### Arret.h , ListeArret.h et ChainonArret.h

#### Arret : un arrêt construit avec ses coordonnées

* Les fonctions « getX » et « getY » qui permettent de récupérer la composante x et y des coordonnées de l’arrêt.
* La fonction « setPosition » qui définit les coordonnées de l’arrêt.
* La fonction « affiche » qui affiche l’arrêt.

#### ChainonArret : un chaînon de la liste ListeArret qui est composé d’un pointeur d’arrêt

*+Classes amies : « ListeArret » et « Ligne ».*

* La fonction « getArret » qui retourne le pointeur de l’arrêt.
* Les fonctions « getSuiv » et « getPrec » qui retournent respectivement le chaînon suivant et le chaînon précédent.

#### ListeArret : liste chainée de ChainonArret

*+Classes amies : « ListeArret » et « Ligne ».*

* La fonction « ~listeArret » qui détruit la liste chaînée.
* La fonction « affiche » qui affiche les arrêts sur l’interface graphique.
* La fonction « insererArret » qui insère un pointeur d’arrêt dans la liste chaînée.

### Tramway.h, ListeTram.h et ChainonTram .h

#### Tramway : un tramway construit avec sa vitesse, sa distance minimale, ses coordonnées et son sens.

* Les fonctions « getVitesse », « getDistanceMin », « getX », « getY » et « getSens » qui retournent respectivement la vitesse, la distance minimale, la composante x et y des coordonnées et le sens du tramway.
* Les fonctions « setVitesse », « setDistanceMin », « setPosition » et « setSens » qui permettent respectivement de définir la vitesse, la distance minimale, la position et le sens du tramway.
* La fonction « setOnMarche » qui permet de définir si le tramway doit se déplacer sur « True », sinon sur « False » et la fonction « getOnMarche » qui récupère et retourne cette valeur.
* La fonction « setTempsArret » qui définit le temps de pause d’un tramway, et la fonction « getTempsArret » qui récupère et retourne cette valeur.
* Les fonctions « affiche » et « efface » qui respectivement affiche ou efface le tramway sur l’interface graphique (en rouge pour sens=0, sinon vert), sous forme d’un carré de 20 pixels.

#### ChainonTram : un chaînon de ListeTram qui est composé d’un pointeur de tramway

+Classe amie : « ListeTram ».

* La fonction « getTramway » qui retourne le pointeur d’un tramway.
* La fonction « actualArret » qui retourne le chaînon de l’arrêt d’où le tram vient.

#### ListeTram : liste doublement chaînée de ChainonTram

*+Classe amie : « ChainonTram ».*

* La fonction « insererTramway » qui insère un pointeur de tramway dans la liste chaînée.
* La fonction « avancer » qui fait avancer le tramway pointé. L’arrête s’il ne respecte pas les distances, change l’arrêt actuel dans ses variables.
* Les fonctions « affiche » et « efface » qui respectivement affiche ou efface la liste chaînée de tramway dans l’interface graphique.
* La fonction « trajectoire » qui gère la direction et le sens que va prendre le tramway.
* La fonction « distance » qui renvoie « Vrai » si le tramway respecte la distance avec le suivant, sinon « Faux ».
* La fonction « taille » qui renvoie la taille de la liste chaînée.

### Ligne.h

#### Ligne : liste chaînée de pointeurs lignes qui relient les arrêts.

* La fonction « affiche » qui affiche les lignes reliant les arrêts dans l’interface graphique.
* La fonction « chainonArret » qui renvoie le pointeur d’un chaînon d’arrêt.

# **Choix de l’organisation du programme**

Nous avons d’abord débuté une réunion afin de discuter sur la méthode que nous allions utiliser. Nous avons hésité entre l’utilisation de liste chaînées ou doublement chaînées, mais pour des raisons pratiques il fut évident que nous avions à choisir le doublement chaîné, pour les tramways en tout cas.

Durant tout notre projet, nous avons choisi de fonctionner par binôme afin d’intervenir sur les parties des uns et des autres lorsque nous bloquions. Nous nous sommes basés sur un système d’alternance où lorsqu’un élève du binôme s’arrêtait, l’autre reprenait au même endroit.

Nous avons d’abord décidé de confectionner le fichier qui allait être lu pour définir les bases de notre programme. Après avoir décidé de sa structure, nous avons utilisé une fonction afin de le lire.

Nous nous sommes ensuite penchés sur l’interface graphique. Il nous a fallu afficher d’abord les stations sous forme de carré (30 sur 30 pixels), puis les trams qui devaient être plus petit (pour les faire disparaître dans les arrêts, signifiant qu’ils y étaient entrés). S’en suivit les lignes, qui reliaient les arrêts d’une même liste chaînée entre eux.

Puis nous avons créé les différentes classes et méthodes en commençant par les listes de tram, les arrêts, les lignes. Nous avons principalement utilisé les pointeurs pour définir nos objets et y accéder plus facilement. Nous justifions l’utilisation d’une liste doublement chaînée pour les tramways par la facilité à accéder au tram précédent et pour pouvoir gérer les changements de sens d’un tramway sur une ligne.

Comme expliqué au début de ce document, l’évolution de la position d’un tramway se passe de la manière suivante : on efface le tramway ainsi que la ligne et on les affiche à leur position suivante. Il nous a fallu faire ce choix car se contenter uniquement de supprimer le tramway laissait une marque sur la ligne qui, à force, la faisait disparaître.

Nous avons aussi choisi de définir la taille des objets par des coordonnées [-15, +15] par exemple afin de les centrer par rapport à la ligne, sachant qu’un [0 ; +30] aurait tout à fait été possible mais aurait décalé nos objets.

Enfin, nous avons décidé d’aborder gestion de la distance minimale et des collisions en fonction des coefficients directeurs des lignes, puisqu’elles dépendent directement de la position d’un tramway par rapport à un autre. Il faut que le tramway précédent identifie qu’un tramway se situe devant lui et cela comprend plusieurs variables : la direction de la droite, le sens du tramway, mais aussi sa manière de changer de sens et de gérer la transition entre un bout de ligne et un autre lorsqu’il traverse un arrêt.

# **Contribution des étudiants**

### Répartition

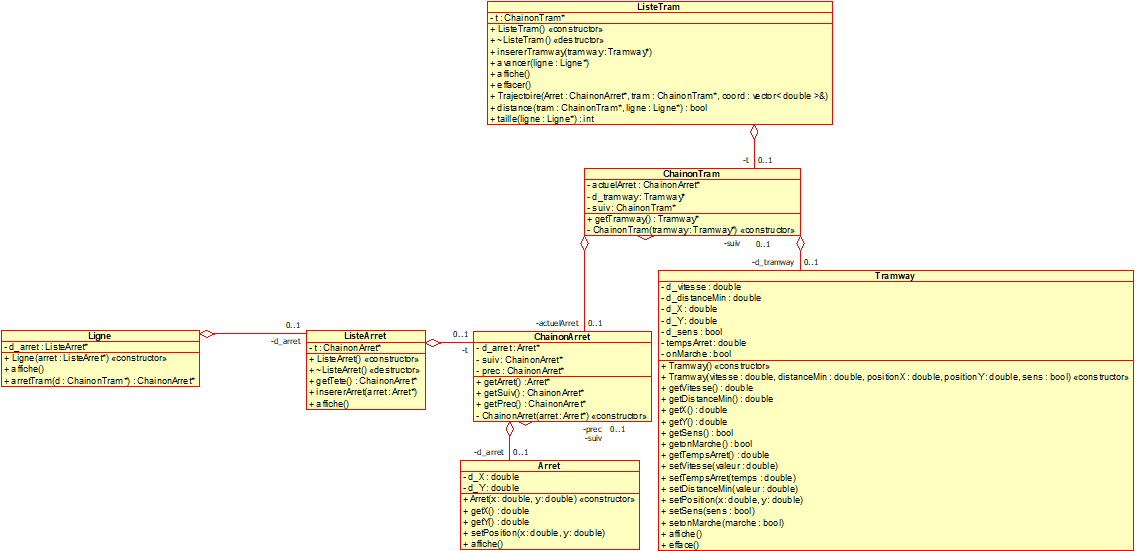
* Les classes « Listetram.h », « ChainonTram.h » et « Ligne.h » : le binôme Elma – Thomas.
* Les classes « Arret.h », « ChainonArret.h », « ListeArret.h », le fichier d’entrée et le « main.cpp » : le binôme Lukas – Gaulthier.
* Rapport de projet : tout le groupe.

# **Conclusion**

Le point le plus compliqué fut la gestion de collision et distance minimale, qui de notre point de vue utilisait beaucoup de variables. Après avoir tenté plusieurs approches, nous sommes arrivés à un projet correct mais néanmoins perfectible.

Pour conclure, nous tenions à remercier nos enseignants :

* Mr.Rivière et Mr.Schmitt pour leur apport en connaissance sur WinBGIm.
* Mr.Lepagnot pour son cours sur la gestion de projet.
* Mr.Maillot et Mr.Cordier pour les cours et travaux dirigés en structure de données, ainsi que pour ce projet qui nous aura permis de comprendre plusieurs points essentiels du programme.

**Annexe**

1. Disponible en Annexe à la fin du rapport [↑](#footnote-ref-1)