QTRAINSIM

Simulateur de trains électriques

Documentation accompagnant la version 1.0, publiée le 9 septembre 2011.

Développeur : Jérémie Ecoffey

Superviseur : Yann Thoma, Professeur à la HEIG-VD

# Introduction

Ce document vise à présenter le logiciel QTrainSim. Ce logiciel a été développé par Jérémie Ecoffey, sous la supervision de Yann Thoma (Professeur à la HEIG-VD). Il est utilisé dans le cadre des cours de programmation concurrente, en complément de maquettes de trains réelles.

## But du Logiciel

Pour certains laboratoires de programmation concurrente, usage est fait de maquettes de train permettant une illustration pédagogique de certains concepts, tels l’exclusion mutuelle ou la section critique. Le nombre d’étudiants ne permettant pas que chacun ait accès aux maquettes pendant la préparation du laboratoire, un simulateur est fourni. Il utilise la même API que le système de contrôle des maquettes.

QTrainSim vient en remplacement de TrainSim. TrainSim fut créé en 1989 dans la même optique, et différentes versions se sont succédées pour répondre aux évolutions technologiques et pédagogiques. Les couches rédactionnelles successives en ont rendu l’usage et la maintenance difficile. De plus, TrainSim n’était pas portable, et exigeait des étudiants l’installation d’une machine virtuelle Linux, ce qui rendait son utilisation parfois difficile pour des étudiants ayant des machines peu puissantes.

QTrainSim reprend le cahier des charges de TrainSim, en ajoutant la portabilité, ainsi que certaines facilités d’interface. La rétrocompatibilité devait être assurée, notamment avec les fichiers représentant les maquettes. Les travaux présentés par les étudiants dans années précédentes doivent également être utilisables, moyennant quelques adaptations mineures.

Enfin, la compatibilité avec le système de contrôle des maquettes est respectée par l’usage de la même API.

## Lectorat du présent Document

Ce document s’adresse aussi bien aux étudiants utilisant QTrainSim dans le cadre d’un laboratoire, qu’aux développeurs futurs chargés de sa maintenance. Il décrit sa structure générale, son utilisation et les spécificités auxquelles il faut prêter attention.

# Utilisation dans le cadre d’un laboratoire

## Généralités

Pour programmer le comportement des locomotives dans QTrainSim, vous utiliserez le langage C. Il est possible de créer des objets en C++ (le simulateur lui-même est en C++), et même d’utiliser la librairie Qt. Toutefois, le projet est structuré de telle manière que l’utilisation la plus simple consiste en la complétion en C de la méthode run() utilisée par le thread User dans main.cpp. Cela peut impliquer la déclaration de variables hors de cette méthode, voire même la création d’autres threads.

En aucun cas il n’est nécessaire, ni même souhaitable, que vous modifiiez les autres fichiers du projet. Si vous le faites, sachez que votre programme pourrait ne pas fonctionner sur la maquette physique. A vos risques et périls…

Vous trouverez toutes les fonctions nécessaires dans l’API (voir ctrain\_handler.h). Ces fonctions peuvent être appelées directement et sans préfixe depuis main.cpp. Lisez attentivement les commentaires de chaque fonction.

## Présentation de l’interface

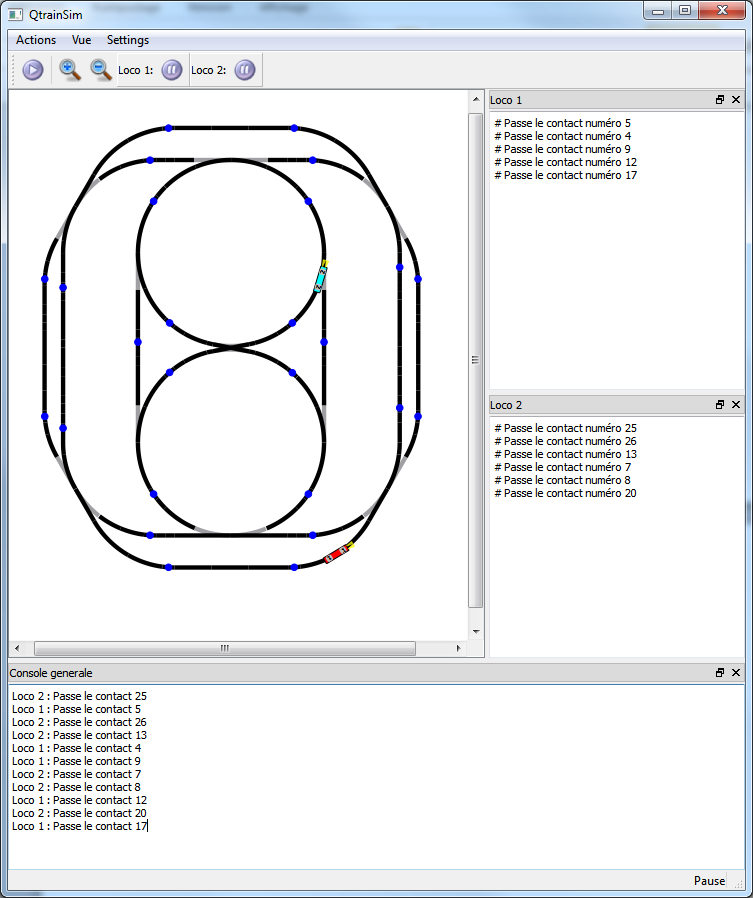


Figure : Interface utilisateur de QTrainSim

L’interface de QTrainSim (voir ci-dessus) est très simple et fonctionnelle, et s’adapte à vos besoins.

La maquette est schématisée par une vue en deux dimension. Les voies variables (aiguillages, aiguillages enroulés, aiguillages triples, traversées-jonctions) sont représentées avec une voie en noir (qui indique l’orientation actuelle de la voie) et une ou plusieurs voies grisées. Il est possible de modifier l’orientation d’une voie variable en cliquant directement sur celle-ci.

La ou les locomotives sont représentées par un rectangle de couleur portant le numéro de la locomotive aux deux extrémités. Deux triangles jaunes représentent les phares, et indiquent la direction de la locomotive. Les couleurs des locomotives sont générées dynamiquement de manière à ce qu’elles soient le plus distinguables possible.

Les contacts sont indiqués par des disques bleus.

La vue de la maquette peut être manipulée (zoom, rotation…) en passant par le menu « Vue ». Des boutons de zoom (« in », un loupe avec un + ; « out » une loupe avec un -) sont également disponibles directement dans la barre d’outils.

Le menu « Vue » vous permet également d’afficher ou cacher les numéros des contacts, ainsi que les numéros des aiguillages.

La console générale offre un journal des informations de toutes les locomotives dans l’ordre chronologique.

Chaque locomotive est également dotée de sa propre console (sur la droite de la fenêtre). Ces consoles peuvent être affichées ou cachées en passant par le menu « Vue -> View Loco Logs ».

La barre d’outils vous offre également un bouton permettant la mise en marche et la mise en pause de la simulation, ainsi que des boutons permettant l’arrêt d’une locomotive indépendamment des autres. Cela est notamment utile pour arrêter une locomotive dans une section critique, afin de vérifier qu’elle est bien respectée.

Enfin, le menu « Settings » vous permet d’activer ou désactiver l’inertie (« Inertia »). En pratique, activer l’inertie vous permet une simulation plus fidèle aux conditions réelles. Vos locos changeront alors leurs vitesses de manière progressives. Cela est notamment important dans la gestion des distances de freinage.

Notez tout de même que l’inertie dans le simulateur n’est pas nécessairement la même que celle des locomotives sur les maquettes réelles. Prévoyez donc un peu de marge…

## Guide des bonnes pratiques

Avant toute chose, lisez attentivement les commentaires des fonctions fournies par l’API. Cela vous permettra de construire plus efficacement la structure de vos threads.

Pour assurer le bon fonctionnement du simulateur, la fonction main doit *au minimum* correspondre à ceci : (Evidemment, il faut remplacer NOM\_DE\_LA\_MAQUETTE) par le nom adéquat…)

int main(int argc, char \*argv[])

{

QApplication app(argc,argv);

init\_maquette();

selection\_maquette("NOM\_DE\_LA\_MAQUETTE");

UserThread user;

if (!user.initialize()) {

exit(0);

}

user.start();

return app.exec();

}

Dès lors, en incluant votre code dans la méthode run() de la classe UserThread (directement dans main.cpp), vous n’aurez pas de problème (En tous cas pas de problème lié au simulateur). Evidemment, vous pouvez déclarer d’autres threads. Veillez toutefois à les créer depuis le run() de la classe UserThread.

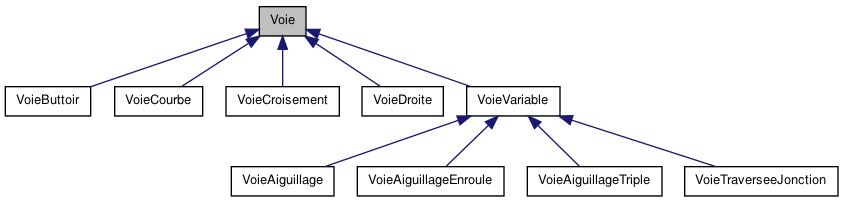
# Guide de Maintenance

Le guide ci-dessous ne se veut pas exhaustif. Il contient simplement les indications jugées utiles aux évolutions futures du logiciel. Si ce guide ne suffit pas, les fichiers eux-mêmes sont porteurs de commentaires qui devraient vous aider.

A notre sens, les évolutions possibles du logiciel tiennent essentiellement à l’ajout possible de nouvelles voies. C’est pourquoi nous nous concentrons essentiellement sur ces éléments.

## Aperçu de la hiérarchie des classes

Les voies sont hiérarchisées de la manière suivante :



1. Figure : hiérarchie des classes de voies (héritage)

Voie et VoieVariable sont des classes abstraites.

Il s’agit là du seul héritage complexe du projet. Tous les autres éléments existent pour eux-mêmes.

## Format de fichier

Les fichiers décrivant les maquettes sont au format txt. Ils commencent généralement par une courte introduction indiquant de quelle maquette il s’agit. Cette partie n’est pas prise en compte par le chargeur de fichiers.

Le chargement commence effectivement par une ligne constituée uniquement d’un entier, indiquant le nombre de voies. Les lignes suivantes décrivent chacune une voie et sont structurées comme suit :

no\_voie no\_ref nos\_voies\_voisines [argument facultatif]

*NB : les éléments sont séparés par des espaces, et non par des tabulations.*

no\_voie est explicite. no\_ref est le numéro de référence Märklin de la voie. nos\_voies\_voisines est la liste des numéros de voies étant directement lié à cette voie. Pour certaines voies, la mention de la direction est ajoutée (voies courbes, aiguillages, etc…).

L’ordre des éléments dans la liste des voies voisines est important dès qu’il s’agit d’une voie ayant trois ou plus extrémités.

* Pour les aiguillages, l’ordre est le suivant : voie connectée à l’extrémité « commune » (c’est-à-dire celle qui donne accès aux deux autres), voie connectée à l’extrémité en bout de droite, voie connectée à l’extrémité en bout de courbe.
* Pour les aiguillages enroulés, l’ordre est le suivant : voie connectée à l’extrémité « commune », voie connectée à l’extrémité intérieure (située à l’intérieur de la courbure), voie connectée à l’extrémité extérieure.
* Pour les aiguillages triples, l’ordre est le suivant : voie connectée à l’extrémité « commune », voie connectée à l’extrémité en bout de droite, voie connectée à l’extrémité en bout de courbe gauche, voie connectée à l’extrémité en bout de courbe droite.
* Pour les croisements, de même que pour les traversées-jonctions, l’ordre est le suivant : voie connectée à gauche d’un des angles aigus, voie connectée en face de la première, voie connectée à droite de la première, voie connectée en face de la troisième.

4

3

2

1

1. Figure : illustration de l'ordre de liaison d'un croisement ou d'une traversée-jonction.

Si le numéro de référence Märklin devait ne pas être reconnu par le chargeur de fichiers, deux cas peuvent se présenter.

Le premier cas, le plus simple, est que votre numéro fait référence à un type de voie étant déjà implémenté. Auquel cas, il suffit d’ajouter sa référence dans le fichier texte « infosVoies.txt », en respectant la structure prévue :

no\_ref type arguments

Les arguments sont (dans cet ordre) :

* Pour une droite : la longueur en mm.
* Pour une courbe : l’angle en degré et le rayon en mm.
* Pour un aiguillage : l’angle de la courbe en degrés, le rayon de la courbe en mm et la longueur de la droite en mm.
* Pour un aiguillage enroulé : l’angle des courbes en degrés, le rayon des courbes en mm et la distance entre les deux centres de rotation.
* Pour un aiguillage triple : l’angle des courbes en degrés, le rayon des courbes en mm et la longueur de la droite en mm.
* Pour un croisement : l’angle aigu entre les droites en degrés et la longueur des droites en mm.
* Pour une traversée-jonction : l’angle des courbes (qui est également l’angle aigu entre les droites) en degrés, le rayon des courbes en mm et la longueur des droites.
* Pour un buttoir : la longueur de la voie.

(Notez que l’aiguillage enroulé tel qu’implémenté implique que le rayon et l’angle des courbes soient les mêmes. C’est le cas pour le matériel Märklin, mais pas nécessairement pour d’autres marques. Le cas échéant, il sera nécessaire de créer une nouvelle classe.)

Le second cas qui peut se présenter est que la voie désirée ne correspond pas à une voie implémentée. Dès lors, il faudra créer la classe correspondante qui héritera soit de Voie (pour les voies statiques) soit de VoieVariable. Cela implique également des adaptations dans le chargeur de fichiers.

Lors de la création d’une nouvelle voie, il y a quelques points clés à prendre en compte : Les angles sont stockés en degrés, arrondis au quart de degré. Pour cela, ils sont accessibles au travers de fonctions ad hoc. Il est également important de noter que les coordonnées sont calculées en référentiel mathématique (angle 0 à 3 heures, x positif vers la droite, y positif vers le haut), mais que l’affichage se fait en référentiel d’écran (c’est-à-dire que l’axe y est inversé). Cela devient particulièrement clé quand il s’agit de calculer des angles à partir de coordonnées (fonction atan2 de math.h). Prêtez-y attention, et inspirez-vous des classes préexistantes.

De plus, chaque voie doit comporter une fonction correctionPosition(…), qui permet de « tordre » légèrement les voies pour qu’elles s’ajustent lors de leur pose. En clair, lorsqu’une voie est posée, et qu’elle n’arrive pas exactement en face de sa voisine déjà posée, les deux voies sont modifiées afin qu’elles se rencontrent à mi-chemin. Cette correction n’est pas très complexe pour une voie droite. Pour une courbe par contre, elle implique le calcul d’un nouveau rayon, et d’un nouveau centre de rotation. L’angle est par contre conservé.

Evidemment, lors d’une correction, les autres points de liaisons des voies ne doivent pas être modifiés.

Là encore, inspirez-vous de ce qui existe déjà.

# Conclusion

QTrainSim est prêt à être utilisé dans le cadre pour lequel il est prévu, c’est-à-dire les cours de Programmation Concurrente. En l’état, il a été testé avec succès sur Windows (seven) et Linux (Ubuntu 11.04). Les objectifs fixés (rétrocompatibilité, gestion de l’inertie, maintien des fonctionnalités déjà présentes, multi-plateforme) ont été atteints.

La version actuelle demanderait encore quelques améliorations :

* Les numéros de contacts et des aiguillages pourraient être plus soignés.
* L’alerte proximité est actuellement calculée en projetant les trajets des locomotives dans les deux à trois secondes. Idéalement, il faudrait utiliser la notion de segment (en associant les segments incompatibles entre eux) afin d’être plus précis.
* Quand une locomotive arrive en buttée, elle est actuellement simplement stoppée. Il semble plus correct qu’elle déraille.