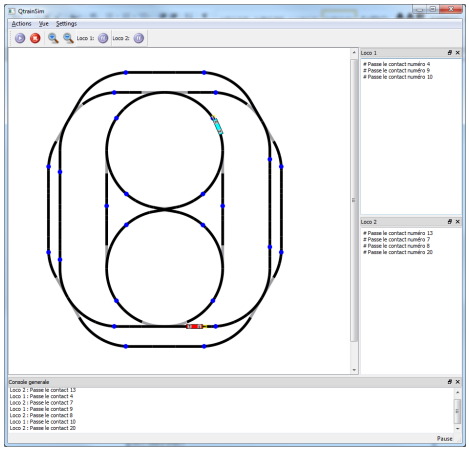
heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

ReDS
Reconfigurable & embedded
Digital Systems

QtrainSim

Kevin Georgy, Jéremie Ecoffey, Daniel Molla, Yann Thoma



Département TIC
Institut REDS
Version du 19 mars 2014

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale

Révisions

Date	Version	Ingénieur	Révision
17/01/09	v1.0	KGY	Version initiale
26/03/09	v1.1	KGY	Ajout des plans des maquettes et des fonctionnalités GUI
05/03/12	v1.2	DMO	Mise à niveau de la documentation pour l'application Qt
22/03/12	v1.3	YTA	Nouvelles fonctionnalités de l'application et refonte d'une partie du document
19/03/14	v1.4	YTA	Documentation de l'entrée clavier, passage à Qt5 et développement en C ou C++

Table 1 – Révisions

Mise en forme

- Les noms propres sont écrits en Petites Capitales
- Les fichiers, dossiers ou commandes sont en `chasse fixe`
- Les termes étrangers, les nouveaux termes ou les termes techniques sont en *emphasis*
- Le *listing* de code prend la forme suivante :
  - Pour des commandes ou un extrait de code source :

```
./configure
make -j8
make install
```

  - Pour du code sources :

```
1 #include <stdio.h>
2
3 int main(int argc, char ** argv)
4 {
5     print("Un exemple...\n");
6     return 0;
7 }
```

Table des matières

1 Introduction

QTRAINSIM est un programme de simulation des maquettes de train MÄRKLIN. Outre une utilisation en simulation pure, il peut également être exploité pour la commande de maquettes réelles, grâce à la librairie LIBTRAINSIM. Le développeur peut contrôler l'exécution des locomotives, via du code C ou C++.

Ce document indique comment installer le simulateur sur différentes plate-formes disponibles et comment l'utiliser.

Installation 2

2.1 Mise en place

Cette documentation se base sur l'environnement Qt SDK 5.0. A priori le code devrait être fonctionnel sur des version postérieurs du Qt SDK. Le simulateur est écrit en C++, en exploitant les différentes classes offertes par le Qt SDK.

Un projet démontrant un fonctionnement basique peut être récupéré sur le site web du reds : `http://reds.heig-vd.ch`. Le répertoire du projet contient :

```
/
└── QTrainSim
    ├── src ................. Contient les fichiers sources
    │   └── ...
    ├── images ............... Contient les images
    │   └── ...
    ├── sound ................ Contient les sons
    │   └── ...
    ├── Maquettes ........... Contient la description des maquettes
    │   └── ...
    ├── QTrainSim.pro ...... Fichier de description du projet Qt
    ├── qtrainsim.qrc ...... Fichier de ressources
    └── infosVoies.txt ..... Fichier de description des voies
```

Le projet peut être compilé en ligne de commande ou via QtCreator, l'IDE fourni avec l'environnement Qt.
En ligne de commande, une fois placé dans le répertoire `QTrainSim`, exécuter les commandes suivantes :

```
qmake # génération du Makefile
make  # compilation proprement dite
```

Par défaut la compilation se fait pour une simulation. Afin de compiler pour une exécution sur les maquettes réelles, la ligne de suivante doit être décommentée dans le fichier `QTrainSim.pro` :

```
1 CONFIG += MAQUETTE
```

Cette ligne permet de spécifier à l'application si elle doit utiliser la librairie du simulateur ou la librairie des maquettes physiques.
Les opérations de compilation peuvent également être exécutées grâce à l'environnement QtCreator. Il est conseillé de l'utiliser, notamment pour les possibilités de debug offertes. Le fichier de projet `QtrainSim.pro` peut être ouvert via QtCreator. L'application, après compilation, peut être lancée via l'interface de QtCreator, en mode *release* ou *debug*.
⚠ Attention, avec QtCreator, la compilation peut se faire en *shadow build* ou non. Par défaut c'est le mode *shadow build* qui est utilisé. Dans ce cas, il faut ajouter une étape de déploiement dans le panel *Projects*, sous l'onglet *Run*. Il faut simplement appeler

```
make install
```

au déploiement, afin que tous les fichiers maquettes soient copiés dans le répertoire de destination. L'autre option consiste à décocher l'option *shadow build* dans l'onglet *Build* du panel *Projects*.

Finalement, il est possible de développer en C ou C++. Pour un développement en C++, il faut travailler sur le fichier cppmain.cpp. Pour développer en C, il faut travailler dans cmain.c, et ajouter la ligne suivante dans le fichier `QtrainSim.pro` :

```
CONFIG += CDEVELOP
```

2.2 Écriture d'un programme

Les fonctions fournies par la librairie sont définies dans `src/ctrain_handler.h` (*listing* 2.1).

```
#ifndef H_CTRAIN_HANDLER

#define H_CTRAIN_HANDLER

/*
 * Fichier          : ctrain_handler.h
 * Auteur           : Kevin Georgy
 *
 * Date de creation : 4.2.2009
 * But              : Fournit les fonctions de controle du simulateur/maquette de trains.
 * Revision         : 27.3.2009 (CEZ)
 *                    27.4.2009 (KGY) Ajout du extern "CE pour les applications c++
 *                    08.9.2011 (Jeremie Ecoffey) Adaptation au nouveau simulateur.
 *                    15.2.2012 (YTA) Ajout des fonctions pour affichage de messages dans
 *                              la console generale et les consoles des locos.
 */

#ifdef __cplusplus
extern "C" {
#endif

// Vitesse a l'arret
#define VITESSE_NULLE 0

// Vitesse minimum
#define VITESSE_MINIMUM 3

// Vitesse maximum
#define VITESSE_MAXIMUM 14

// Numero max. d'aiguillage
#define MAX_AIGUILLAGES 80

// Numero max. de contact
#define MAX_CONTACTS 64

// Numero max. de loco
#define MAX_LOCOS 80

// Direction des aiguillages
#define DEVIE 0
#define TOUT_DROIT 1

// Etat des phares
#define ETEINT 0
#define ALLUME 1

/*
 * Initialise la communication avec la maquette/simulateur.
 * A appeler au debut du programme client.
 */
void init_maquette(void);

/*
 * Met fin a la simulation. A appeler a la fin du programme client.
 */
void mettre_maquette_hors_service(void);

/*
 * Realimente la maquette. Inutile apres init_maquette().
 */
void mettre_maquette_en_service(void);

/*
 * Change la direction d'un aiguillage.
 *   no_aiguillage : No de l'aiguillage a diriger.
 *   direction     : Nouvelle direction. (DEVIE ou TOUT_DROIT)
 *   temps_alim    : Temps l'alimentation minimal du bobinage de l'aiguillage.
 */
void diriger_aiguillage(int no_aiguillage, int direction, int temps_alim);

/*
 * Attend l'activation du contact donne.
 *   no_contact : No du contact dont on attend l'activation.
```

```
75  */
76 void attendre_contact(int no_contact);
77
78 /*
79  * Arrete une locomotive (met sa vitesse a VITESSE_NULLE).
80  *   no_loco : No de la loco a arreter.
81  */
82 void arreter_loco(int no_loco);
83
84 /*
85  * Change la vitesse d'une loco par palier.
86  *   no_loco        : No de la loco a stopper.
87  *   vitesse_future : Vitesse apres changement.
88  * Remarque : Dans le simulateur cette procedure agit comme la fonction
89  *            "mettre_vitesse_loco". Son comportement depend de l'option
90  *            "Inertie" dans le menu ad hoc.
91  */
92 void mettre_vitesse_progressive(int no_loco, int vitesse_future);
93
94 /*
95  * Permettre d'allumer ou d'eteindre les phares de la locomotive.
96  *   no_loco : No de la loco a controler.
97  *   etat    : Nouvel etat des phares. (ETEINT ou ALLUME)
98  * Remarque : Dans le simulateur cette fonction n'a aucun effet.
99  *            Les phares sont toujours allumes, et indiquent le sens de la loco.
100  */
101 void mettre_fonction_loco(int no_loco, char etat);
102
103 /*
104  * Inverse le sens d'une locomotive, en conservant ou retrouvant sa vitesse originale.
105  *   no_loco : No de la loco a inverser.
106  * Remarque : Dans le simulateur, le comportement depend de l'option "Inertie" dans le menu ad hoc.
107  */
108 void inverser_sens_loco(int no_loco);
109
110 /*
111  * Change la vitesse d'une loco.
112  *   no_loco : No de la loco a controler.
113  *   vitesse : Nouvelle vitesse.
114  * Remarque : Dans le simulateur, le comportement depend de l'option "Inertie" dans le menu ad hoc.
115  */
116 void mettre_vitesse_loco(int no_loco, int vitesse);
117
118 /*
119  * Indique au simulateur de demander une loco a l'utilisateur. L'utilisateur entre le
120  * numero et la vitesse de la loco. Celle-ci est ensuite placee entre les contacts
121  * "contact_a" et "contact_b".
122  *   contact_a   : Contact vers lequel la loco va se diriger.
123  *   contact_b   : Contact a l'arriere de la loco.
124  *   numero_loco : Numero de loco choisi par l'utilisateur.
125  *   vitesse     : Vitesse choisie par l'utilisateur.
126  * Remarque : cette methode n'est pas utilisee dans le simulateur.
127  *            Veuillez utiliser assigner_loco(...);
128  */
129 void demander_loco(int contact_a, int contact_b, int *no_loco, int *vitesse);
130
131 /*
132  * Indique au simulateur d'assigner une loco.
133  * Le numero et la vitesse de la loco sont definies, et elle est ensuite placee
134  * entre les contacts "contact_a" et "contact_b".
135  *   contact_a   : Contact vers lequel la loco va se diriger.
136  *   contact_b   : Contact a l'arriere de la loco.
137  *   numero_loco : Numero de loco.
138  *   vitesse     : Vitesse de la loco.
139  */
140 void assigner_loco(int contact_a, int contact_b, int no_loco, int vitesse);
141
142
143 /*
144  * Selectionne la maquette a utiliser.
145  * Cette fonction termine l'application si la maquette n'est pas trouvee.
146  * La maquette est cherchee dans le repertoire contenant les maquettes.
147  *   maquette : Nom de la maquette.
148  */
149 void selection_maquette(const char *maquette);
150
151 /*
152  * Affiche un message dans la console principale
153  *   message : chaine de caractere qui sera affichee dans la console.
154  */
155 void afficher_message(const char* message);
156
157 /*
158  * Affiche un message dans la console d'une loco
159  *   numLoco : numero de la locomotive
160  *   message : chaine de caractere qui sera affichee dans la console.
161  */
162 void afficher_message_loco(int numLoco,const char* message);
```

```
163
164 /*
165  * Fonction bloquante permettant de recevoir la prochaine commande
166  * entree par l'utilisateur.
167  *   return : la commande entree par l'utilisateur
168  */
169 const char* getCommand();
170
171 /*
172  * Copie le rÃ©sultat de la commande saisie par l'utilisateur dans le
173  * tableau passÃ© en paramÃštre
174  *   commande : tableau de caractÃšres
175  *   taille : taille du tableau
176  */
177 void getCommandInArray(char *commande, int taille);
178
179 #ifdef __cplusplus
180 }
181 #endif
182
183 #endif
```

Listing 2.1 – ctrain_handler.h

Le *listing* 2.2 montre un fichier source minimal pour l'utilisation de la librairie. On remarque l'inclusion `#include "ctrain_handler.h"` qui fournit l'accès au fonctions. L'appel à `init_maquette()` et `mettre_maquette_hors_service()` sont à effectuer obligatoirement en début et fin de programme.

```
1  #include <pthread.h>
2  #include "ctrain_handler.h"
3  #include <errno.h>
4
5
6  // structure qui definit une locomotive
7  typedef struct
8  {
9    int no;
10   int vitesse;
11 } Locomotive;
12
13
14 // Declaration des deux locomotives
15 Locomotive loco1;
16
17
18 void emergency_stop()
19 {
20   printf("\nSTOP!");
21
22   // on arrete les locomotives.
23   arreter_loco(loco1.no);
24 }
25
26
27 // Contacts a parcourir
28 #define NB_CTS 7
29 int parcours[] = {6, 11, 10, 13, 14, 19, 3};
30
31
32 void cmain()
33 {
34   int ct;
35
36   loco1.no =1;
37   loco1.vitesse = 12;
38
39   selection_maquette("MAQUET_B");
40   init_maquette();
41
42   // Demande au simulateur de placer une loco entre les contacts 6 et 11
43   // Recupere le numero et la vitesse saisis par l'utilisateur.
44   assigner_loco(  parcours[1],
45                   parcours[0],
46                   loco1.no,
47                   loco1.vitesse);
48
49   // Dirige les aiguillages sur le parcours
50   diriger_aiguillage(7,TOUT_DROIT,0);
51   diriger_aiguillage(8,DEVIE,0);
52   diriger_aiguillage(5,TOUT_DROIT,0);
53   diriger_aiguillage(9,DEVIE,0);
54   diriger_aiguillage(10,TOUT_DROIT,0);
55   diriger_aiguillage(14,TOUT_DROIT,0);
56   diriger_aiguillage(13,DEVIE,0);
57   diriger_aiguillage(1,TOUT_DROIT,0);
58
```

```
59
60    // Demarre la loco
61    mettre_vitesse_progressive(loco1.no, loco1.vitesse);
62
63    // Attend que la loco passe sur les differents contacts de son parcours.
64    for (ct = 1; ct < NB_CTS; ct++) {
65      attendre_contact(parcours[ct]);
66      printf("Loco %d de vitesse %d a atteint le contact %d.\n", loco1.no, loco1.vitesse, ct);
67    }
68
69    // Stoppe la loco
70    arreter_loco(loco1.no);
71
72    // Fin de la simulation (a effectuer une seule fois en fin de programme, sans effet
73    // sur le simulateur, mais necessaire sur les maquettes reelles).
74    mettre_maquette_hors_service();
75  }
```

Listing 2.2 – Fichier source minimal

Utilisation 3

3.1 Généralité

Pour programmer le comportement des locomotives dans QTrainSim, il faut utiliserer le langage C. Il est neamoins 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`. En aucun cas il n'est nécessaire, ni même souhaitable de modifier les autres fichiers du projet. En cas de modification, il faut savoir que le programme pourrait ne pas fonctionner sur la maquette physique. Toutes les fonctions nécessaires se trouvent dans l'API représetée par le fichier `ctrain_handler.h`. Ces fonctions peuvent être appelées directement et sans préfixe depuis `cmain.c`. Le simulateur se lance directement à partir de QtCreator. Afin d'indiquer au simulateur la maquette et les locomotives à utiliser, les fonction `selection_maquette()` et `assigner_loco()` doivent être appelées à l'intérieur de la fonction `cmain()` du fichier `cmain.c`.

3.2 Interface graphique

La figure 3.1 représente l'interface graphique dans son integralité.

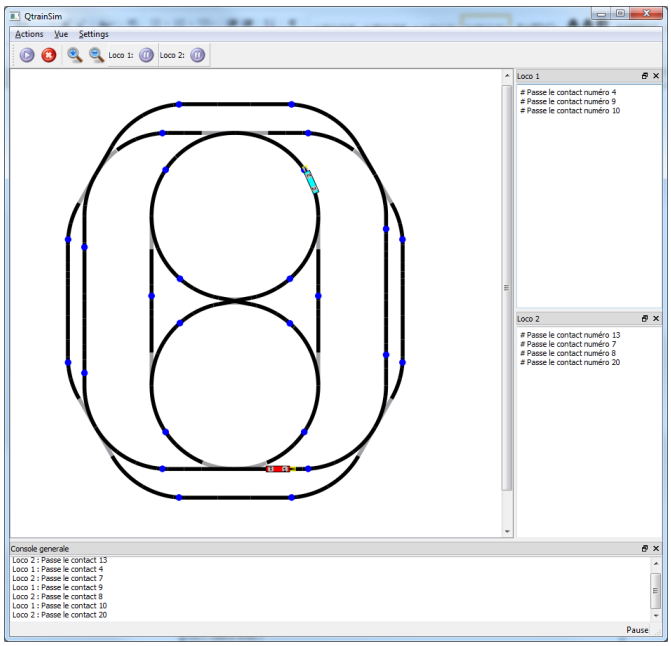


FIGURE 3.1 – Interface graphique du simulateur

3.2.1 Barre de menu

3.2.1.1 File

– **File→Print** permet d'imprimer la vue de la maquette en cours.
– **File→Exit** permet de quitter l'application.

3.2.1.2 Actions

Le menu **Actions** de la barre de menu permet d'interagir directement sur la vue contenant la maquette virtuelle du simulateur. Voici les différentes actions définies :
- **Actions→Resume/Pause** permet de démarrer ou mettre en pause le système en fonction de l'état de celui-ci. Cette action n'est applicable qu'en simulation, qui doit notamment être lancée via ce biais. Sur la maquette l'expérience débute immédiatement au lancement du programme.
- **Actions→Emergency stop** fait appelle à la fonction *emergency_stop()* située à l'intérieur du fichier `cmain.c`. C'est la responsabilité de l'utilisateur de placer le code spécifique à cet appel.
- **Actions→Pause loco n/Restart loco n** permettent de mettre en pause ou de redémarrer une locomotive spécifique représentée par le numéro n en fonction de son état actuel.

3.2.1.3 Views

Le menu **Views** de la barre de menu permet d'intéragir directement sur la vue contenant la maquette virtuelle du simulateur et les vues relatives aux logs des locomotives. Voici les différentes actions définies pour ces vues :
- **Views→Zoom in** permet de réaliser un zoom progressif au sein de la vue contenant la maquette virtuelle du simulateur.
- **Views→Zoom out** permet de réaliser un zoom regressif au sein de la vue contenant la maquette virtuelle du simulateur.
- **Views→Zoom fit** permet de dimensionner la maquette virtuelle du simulateur au sein de sa vue en fonction de la taille de l'application.
- **Views→Rotate +** permet de réaliser une légère rotation de la maquette virtuelle du simulateur sur la droite.
- **Views→Rotate -** permet de réaliser une légère rotation de la maquette virtuelle du simulateur sur la gauche.
- **Views→View Loco logs** permet d'afficher ou non les logs des locomotives dans leur vue respective.
- **Views→View contact number** permet d'afficher à proximité de chaque contact son numéro.
- **Views→View aiguillage number** permet d'afficher à proximité de chaque aiguillage son numéro.

3.2.1.4 Settings

Le menu **Settings** de la barre de menu permet d'interagir sur différents paramètres de l'application. Actuellement, il est existe uniquement l'action **Settings→Inertia** qui permet d'ajouter ou non de l'inertie lors de l'arrêt des locomotives. En pratique, activer l'inertie permet une simulation plus fidèle aux conditions réelles. Les locos changeront alors leurs vitesses de manière progressives. Cela est notamment important dans la gestion des distances de freinage.

3.2.2 Barre d'outils

La barre d'outils offre différents raccourcis pour les actions **Resume/Pause**, **Emergency stop** , **Zoom in** , **Zoom out** et **Pause loco n/Restart loco n**.

3.2.3 Maquette virtuelle

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. La où 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. Il est possible de modifier l'orientation d'une voie variable en cliquant directement sur celle-ci.

3.2.4 Console générale

La console générale offre un journal des informations de toutes les locomotives dans l'ordre chronologique.
Il est possible d'y afficher des messages en appelant la fonction `afficher_message()`.

3.2.5 Console locomotive

Chaque locomotive est également dotée de sa propre console sur la droite de la fenêtre. Cette console offre un journal des informations de la locomotive concernée dans l'ordre chronologique.
Il est possible d'y afficher des messages en appelant la fonction `afficher_message_loco()`, en passant notamment le numéro de la locomotive en paramètre.

3.2.6 Interactions

Il est possible d'agir sur les aiguillages en cliquant dessus. Les aiguillages changent alors de sens, autant en simulation que sur les maquettes réelles.
Lors de l'appel à la fonction `attendre_contact()`, le contact devient vert et le reste jusqu'à ce qu'une locomotive passe dessus.
Les locomotives peuvent être articifiellement placées en pause en cliquant sur le bouton correspondant. Attention, cette fonctionnalité n'est valable qu'en simulation.

3.2.6.1 Entrée utilisateur

Un champ texte est présent en haut de l'interface utilisateur. Il permet d'entrer du texte qui peut ensuite être récupéré par le contrôle des locomotives, via la fonction `getCommand()` ou la fonction `getCommandInArray()`. Attention, ces 2 fonctions sont bloquantes.

4 Plan des maquettes

La figure 4.1 indique la composition des différentes maquettes utilisables sur le simulateur.

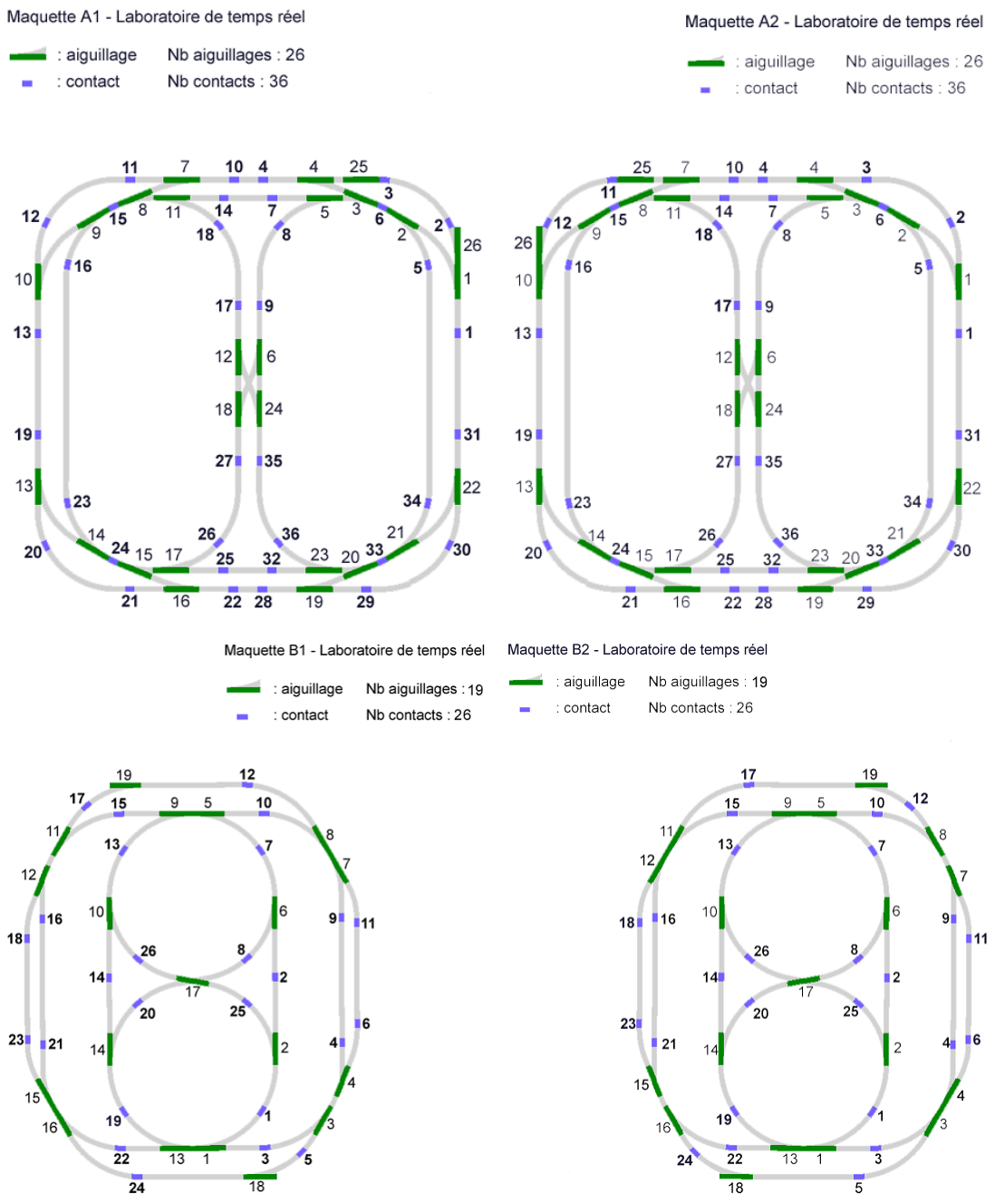


FIGURE 4.1 – Plan des maquettes