IA54
Systèmes multi-agents et résolution distribuée de problèmes

Université de Technologie de Belfort Montbéliard

Croisement de trains de véhicules

Étudiant :
Nathan Olff

Enseignant suiveur :
Franck Gechter

7 janvier 2016

Table des matières

1. Introduction

Le projet, développé dans le cadre d'un module sur les systèmes multi-agents à l'Université de Technologie de Belfort Montbéliard, consiste à créer une simulation de trafic de véhicules. Il ne s'agit pas de simuler n'importe quel trafic mais un cas bien particulier : les croisements de trains de véhicules autonomes.

On définit un train (aussi nommé *platoon*) comme étant une entité virtuelle, constituée de plusieurs véhicules (3 ou plus) qui se suivent en gardant une certaine distance entre chacun d'entre eux.

L'objectif du projet est de gérer une intersection de deux routes sur lesquelles deux trains vont se croiser et adapter leur vitesse afin d'éviter toute collision.

La solution implémentée mettra en pratique les concepts de systèmes multi-agents étudiés tout au long du semestre dans le cadre du module *Systèmes multi-agents et résolution distribuée de problèmes*.

Afin de se concentrer sur les comportements des agents et la résolution du problème (et non sur la présentation d'une interface graphique), un précédent projet développé par Florian Beck, sera utilisé comme base de code.

Une fois le travail sur la simulation terminée, une implémentation réelle sera effectuée à l'aide de robots Mindstorm EV3 dans le cadre d'un module de logiciel embarqué (non détaillé dans ce rapport).

2. Étude du système

2.1 Intersection de deux véhicules

Deux chemins différents sont présents sur les cartes utilisées pour la simulation. Ces deux chemins formant chacun un circuit fermé, se croisent en deux intersections.

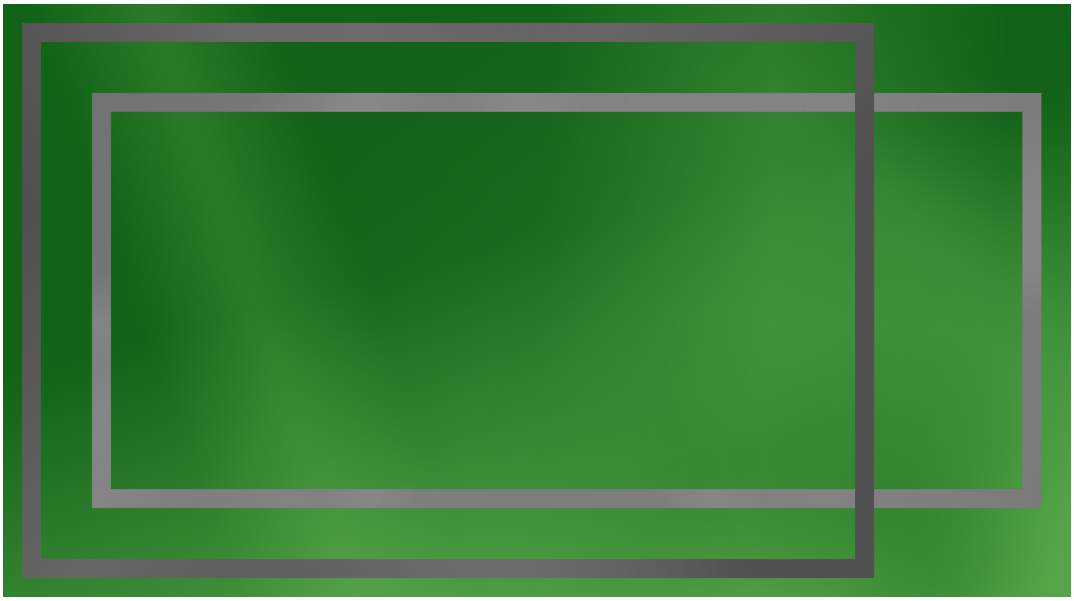

FIGURE 2.1 – Exemple de carte présentant les deux routes

Avec un véhicule sur chaque route, le croisement des deux véhicules revient à un problème de partage de ressource. Deux véhicules ne pouvant pas traverser la même intersection au même moment, il faut alors définir des règles de priorité.

Une solution typique pour ce type de problème est un carrefour de type feu rouge, laissant passer alternativement l'un puis l'autre véhicule.

2.2 Présentation du problème

C'est un problème légèrement différent qui est étudié ici. Sur chaque route, on ne trouve pas un, mais plusieurs véhicules, circulant sous forme de train. Ce train est un ensemble cohérent dont les éléments sont séparés par une même distance appelée inter-distance.

FIGURE 2.2 – Train formé de quatre véhicules

Le but du projet est de permettre à deux trains de véhicules de traverser ces intersections sans avoir à s'arrêter ou à définir un ordre de priorité.

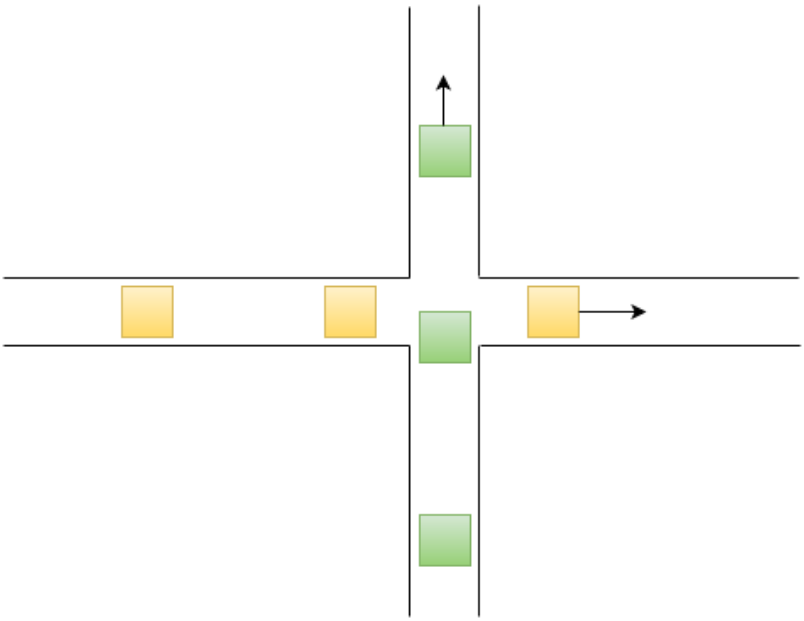

FIGURE 2.3 – Croisement de trains

Comme présenté sur la figure 2.3, les véhicule du train A traversent le croisement entre deux véhicules du train B. Ce fonctionnement permet de ne pas réserver le croisement pour tel ou tel train, et produit donc un trafic plus fluide qu'une solution type feu rouge.

Afin d'atteindre ce but, les trains doivent adapter leurs vitesses et leurs inter-distances à l'approche d'un croisement.

3. Implémentation

D'après la définition de Ferber (1999), un agent est une entité, physique ou virtuelle, capable d'agir dans un environnement et de communiquer avec d'autres agents. Il possède des ressources propres et ne dispose que d'une représentation partielle de l'environnement et du but global du système. Il possède des compétences et offre des services. Son comportement est lié à une/des fonction(s) de satisfaction / survie, qu'il cherche à optimiser.

Le problème est modélisé dans ce projet à l'aide d'un SMA (Système multi-agents).

Ce système possède les caractéristiques suivantes :

— Système fermé : Le nombre d'agents ainsi que leurs rôles sont défini au démarrage du système. En théorie, il devrait être possible de rendre le système ouvert et d'autoriser l'arrivée de nouveaux véhicules dans le système, ou le changement d'un véhicule d'un train à un autre, mais ces cas de figures ne font pas partie du cadre de ce projet.

— Système hétérogène : Comme expliqué ci-dessous, il existe deux types d'agents : véhicule et train.

— Système hiérarchique de composition : Les agents trains sont composés d'agents véhicules. On retrouve dans le système des interactions de type intra- et inter-sous-systèmes : entre les trains et leurs véhicules d'un côté, et entre les différents trains de l'autre (voir section 3.3.1).

3.1 MadKit

MadKit est une plate-forme multi-agents développé en Java. Elle contient une série de classes abstraites représentant différentes notions des systèmes multi-agents (agent, message...). Les agents peuvent créer des organisations et demander à la plate-forme l'attribution de rôles dans ces mêmes organisations (voir section 3.3.1)

Le projet originel de Florian Beck a été développé en utilisant MadKit. Ayant travaillé un peu avec cette plate-forme lors de travaux pratiques, j'ai décidé de continuer sur la même lancée.

3.2 Définition des agents

Afin de modéliser le problème de croisement de trains de véhicules, j'ai utilisé deux types d'agents : véhicule (appelé *Car*) et train, chacun représenté par un classe héritant de la classe MadKit Agent.

3.2.1 Agent véhicule

Chaque véhicule est représenté par un agent Car. Ses attributs sont répartis en plusieurs catégories :

- Identité : l'identifiant unique de l'agent, l'identifiant de son train ainsi que sa position à l'intérieur du train
- Position : la position absolue du véhicule sur la carte, mais également la liste des intersections dans lesquels le véhicule se trouve [1]
- Mouvement en cours : la vitesse actuelle ainsi que la distance entre lui et le véhicule précédant. Il existe d'autres attributs, utilisé pour le calcul de vitesse du premier véhicule du train. Ces derniers seront détaillés dans la partie concernant le comportement du dit *Leader* (voir section 3.4.1).
- Mouvement optimum : vitesse et distance du précédant à atteindre. Afin d'avoir une certaine fluidité de trafic, l'accélération et décélération des véhicules sont bornées. Cela peut donc entraîner une différence entre la vitesse courante et l'objectif de vitesse.

3.2.2 Agent Train

L'agent Train est un agent virtuel, qui pourrait en pratique être implémenté à l'intérieur d'un des véhicules du train (en plus de son propre agent Car) ou bien exécuté par un ordinateur à part.

Le train possède également plusieurs types d'attributs :

- Identité : numéro du train
- Position : liste de croisements dans lesquels au moins un véhicule du train se trouve. Une seconde liste répertorie les croisements dans lesquels le train ainsi qu'un autre train sont présents (gestion de conflit, voir section 3.4.2).
- Objectifs : vitesse moyenne du train et objectif d'inter-distance à maintenir

Du point de vue de la modélisation, on peut considérer les agents trains comme des holons.

1. Un véhicule est considéré à l'intérieur d'une intersection lorsqu'il est à moins d'une certaine distance de la dite intersection. Si deux intersections sont trop proches, il pourrait donc arriver qu'un véhicule soit dans deux intersections à la fois. Ce cas de figure n'ayant pas été étudié, je n'utiliserais pas ce type de cartes (voir section 4.3).

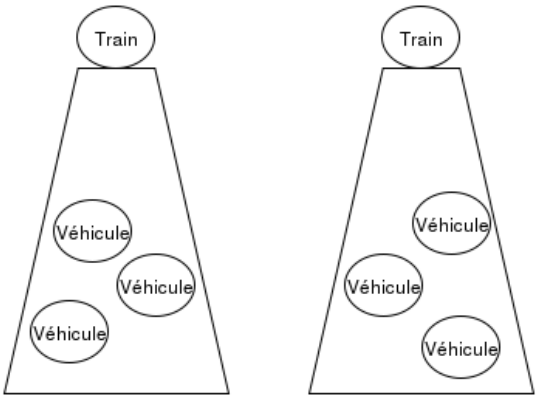

FIGURE 3.1 – Représentation des agents du système

3.2.3 Agent Environnement

Dans le projet de base de Florian Beck, l'environnement était représenté par un agent. J'ai diminué l'importance de cet agent au minimum. Alors qu'il était chargé originalement de détecter l'entrée et la sortie d'un train dans un croisement, il ne sert plus aujourd'hui qu'à récolter les informations sur l'état interne des agents pour afficher des statistiques à l'utilisateur. Il n'est donc plus nécessaire au bon fonctionnement du système.

3.3 Communication entre agents

3.3.1 Organisations

Afin de permettre (et de limiter) la communication entre agents, ces derniers sont répartis dans plusieurs organisations.

Une organisation principale, appelé Train, permet la communication entre l'Environnement et les deux Trains (ainsi qu'entre les trains eux-mêmes).

À l'initialisation, chaque agent Train créé sa propre organisation qu'il utilisera pour communiquer avec ses véhicules.

Les véhicules de deux trains différents ne sont donc pas en capacité de communiquer directement entre eux. Ils doivent impérativement passer par l'intermédiaire de leur train.

Bien qu'il soit théoriquement possible pour un véhicule de communiquer directement avec les autres véhicules de son propre train, j'ai décidé d'éviter de type de communication. Ainsi, les véhicules ne communiquent qu'avec leur train.

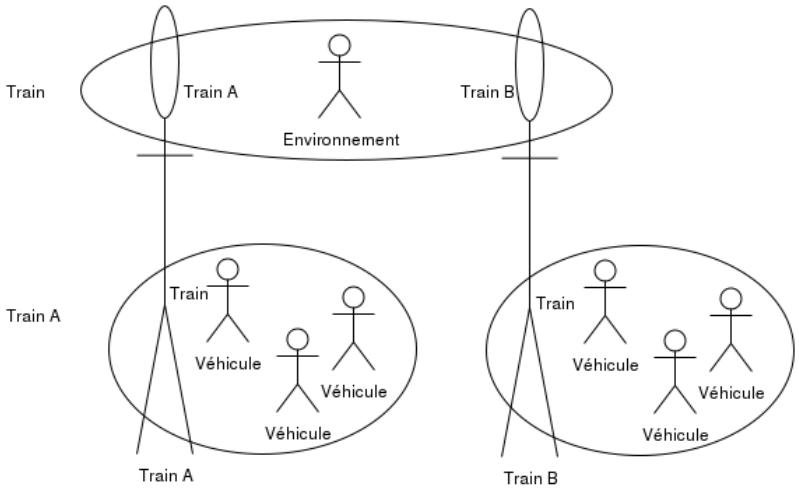

FIGURE 3.2 – Organisations des agents

3.3.2 Passage d'un train dans une intersection

L'entrée et la sortie d'un croisement mettent en jeu les différentes organisations. Voici un diagramme de séquence présentant les principes utilisés, ainsi qu'une description de chacun des niveaux de communications.

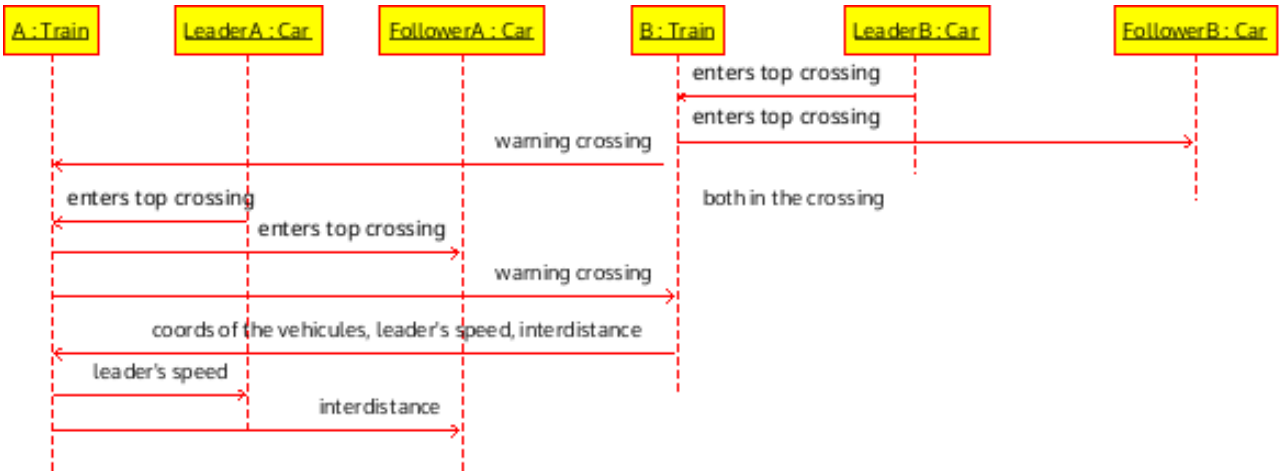

FIGURE 3.3 – Diagramme de séquence – Arrivé des deux trains dans un même croisement

Véhicule → Train : Remontée d'informations des véhicules vers leur train

Le premier véhicule (aussi appelé *Leader*) de chaque train a un rôle particulier à jouer. Il est le seul à appliquer strictement les ordres de vitesses données par son train (voir section 3.4.1). Il a aussi pour rôle de prévenir le train, via l'envoie d'un message, lorsqu'il arrive à proximité d'une intersection.

La queue du train (aussi appelé *Tail*) se charge, dès lors qu'elle a quitté une intersection, de prévenir son train de cette même sortie.

Train → Véhicules : Transmission de l'information vers les véhicules du train

Lorsque le train reçoit un message d'entrée ou de sortie de croisement, il transfère un message de même type à tous les éléments le composant. Ainsi, chaque voiture du train sait qu'il se trouve à proximité (ou non) d'un croisement et peut donc agir en conséquence.

Train → Trains : Prévenir les autres trains d'une entrée ou sortie de croisement

Lorsqu'un train rentre ou sort d'une intersection, il envoie un message d'avertissement aux autres trains [2] pour le(s) prévenir de sa position.

3.4 Comportements des agents

Les agents sont exécutés en parallèle, chacun dans une boucle contenant son exécution ainsi qu'une attente correspondante au pas de la simulation (fixé pour les essais à 35ms).

3.4.1 Agent véhicule

A chaque itération, l'agent véhicule effectue les actions suivantes :

1. Réception des messages.
2. Modification de l'état interne de l'agent en fonction des messages reçus.
3. Définition d'un objectif de vitesse suivant les messages précédent et le distance au précédant.
4. Exécution de l'ordre de vitesse dans la limite de l'accélération et décélération maximales.

Comme on peut le voir ci-dessus, les agents véhicules se contentent donc d'agir en fonction de l'observation qu'ils ont de l'environnement et des messages reçus par leur train. Il n'y a donc ni planification ni processus de réflexion. Ces agents sont bien des agents de type réactifs.

2. Seul les systèmes avec deux circuits, et donc deux trains au total ont été étudié ici (voir section 4.3 pour plus d'informations).

Tête du train

La tête du train, également appelé *Leader*, a pour rôle d'appliquer les ordres de vitesses envoyé par son train. La plupart du temps, il s'agit d'une vitesse constante à maintenir.

Il peut arriver que le train n'envoie pas une vitesse constante, mais une fonction polynomiale définissant la courbe de vitesse à suivre jusqu'à atteindre le croisement. Dans ce cas, le leader calcule à chaque itération la valeur de la fonction en fonction du temps écoulé depuis que le polynôme a été défini. (voir section 3.4.2)

Comportement des suiveurs

Les autres véhicules du train sont des suiveurs. Ils se contentent d'appliquer le dernier ordre d'inter-distance transmis par leur train.

Le respect de cet inter-distance est défini en utilisant un régulateur PID (Proportionnel, Intégral, Dérivé).

Le régulateur PID consiste en une fonction ayant pour but de diminuer une erreur par rapport à une valeur idéal. Dans notre cas, l'erreur est calculée en prenant la différence entre l'inter-distance idéale et la distance d'avec le véhicule précédant. Il prend en compte l'historique d'erreur (grâce à la partie intégrale de la fonction), une partie prédiction de l'erreur à venir (partie dérivée), ainsi qu'une partie proportionnelle à l'erreur courante.

Les paramètres (Ki, Kd, Kp) du PID ont été définit suite à plusieurs essais.

Queue du train

La queue du train (appelé *Tail*) est le dernier véhicule du train. Son comportement est celui d'un suiveur, à la différence près que lorsqu'il sort d'une intersection, il envoie un message à son train pour l'en notifier.

A l'origine, l'entrée et la sortie d'un croisement par un train était géré par l'agent environnement. Le fait d'avoir transféré cette responsabilité de l'environnement aux véhicules en tête et en queue du train permet de se passer entièrement de l'agent Environnement pour le bon fonctionnement du système.

3.4.2 Agent train

Le train étant un agent virtuel, il n'a pour but que de recevoir des messages, de définir un comportement global, et de transmettre ce comportement sous forme d'ordres à ses véhicules. Il fait le pont entre les agents véhicules qui le composent et l'autre train.

Lorsqu'un train A est dans une intersection et reçoit un avertissement indiquant l'entrée d'un autre train B dans cette même intersection, il lui répond en envoyant les coordonnées de chacun de ses véhicules.

Le train B étudie alors les positions et vitesses des véhicules du train A pour définir une stratégie de croisement. À partir de la position et de la vitesse de son propre leader, il calcule combien de temps il va mettre (à vitesse actuelle) pour atteindre le croisement. Il calcule également le temps que va mettre chacun des véhicules du train A pour atteindre le croisement.

Avec ces deux informations, le train B recherche entre quels deux véhicules du train A son leader devrait passer. Il calcule par la suite une vitesse moyenne optimale légèrement différente de la vitesse actuelle afin de permettre au leader d'atteindre le croisement au milieu des véhicules de A.

Algorithm 1 Comportement d'un train lorsqu'il reçoit les coordonnées des véhicules d'un autre train

```
tempsLeaderCroisement ← distance(leader, croisement)/vitesseLeader
Pour tout véhicule v du train opposé Faire
  tempsRestant[v] ← distance(v, croisement)/vitesseDuTrainOpposé
Fin Pour
minTemps ← 0
maxTemps ← infini
Pour tout valeur t de tempsRestant Faire
  Si t < tempsLeaderCroisement Et t > minTemps Alors
    minTemps ← t
  Sinon Si t >= tempsLeaderCroisement Et t < maxTemps Alors
    maxTemps ← t
  Fin Si
Fin Pour
Si minTemps = 0 Alors
  minTemps ← maxTemps - interdistanceDuTrainOpposé/vitesseDuTrainOpposé
Fin Si
Si maxTemps = infini Alors
  maxTemps ← minTemps + interdistanceDuTrainOpposé/vitesseDuTrainOpposé
Fin Si
minTemps ← minTemps + tailleVoiture/vitesseAutreTrain
maxTemps ← maxTemps - tailleVoiture/vitesseAutreTrain
tempsOptimal ← (minTemps + maxTemps)/2
vitesseOptimal ← distanceLeaderCroisement/tempsOptimal
Si |vitesseOptimal - vitesseDuTrainOpposé| < 5 Alors
  vitessePolynome ← vitesseOptimal
Sinon
  vitessePolynome ← calculePolynome(vitesseLeader, vitesseDuTrainOpposé, distanceLeader-
  Croisement, tempsOptimal)
Fin Si
envoieMessage(véhicules, vitessePolynome)
```

Première approche et problème rencontré

Dans un premier temps, j'ai essayé de transmettre cette nouvelle vitesse moyenne aux membres du train. Le leader atteignait alors bien le croisement au moment définit, mais les suiveurs étaient de plus en plus éloigné du comportement "idéal".

En réfléchissant un peu, on peut voir qu'il y avait effectivement un problème de conception dans cette solution.

En effet, pour que les véhicules des deux trains puissent se croiser correctement une fois que le leader ait atteint le croisement, il faut que les inter-distances et les vitesses des deux trains soient identiques.

Seconde approche : le polynôme de vitesse

A partir de ce constat, je suis parti à la recherche d'une solution alternative, respectant ce besoin de même vitesse au niveau du croisement.

Afin de trouver une solution viable, il faudrait trouver une fonction de vitesse en fonction du temps respectant les données ci-dessous :

— La vitesse moyenne calculée précédemment doit être respecté (calcule de valeur moyenne grâce à l'intégrale)
— La vitesse de départ de la fonction est connue.
— La vitesse d'arrivée, à t = durée optimal calculée précédemment, est égale à la vitesse de l'autre train.
— L'accélération et la décélération doit être les plus faibles possible. Il faut donc éviter toute fonction contenant des piques.

Polynôme de vitesse

Une solution réalisable relativement facilement consiste à représenter la vitesse sous la forme d'un polynôme de second dégrée.

La vitesse serait donc présenté sous la forme $at^2 + bt + c$ en fonction du temps. c est définie rapidement grâce à la vitesse initiale. Concernant les coefficients a et b, ils sont définie grâce à la résolution d'un système de deux équations à deux inconnus défini ci-dessous.

A $t = 0$, on a : $at^2 + bt + c = c = vitesseLeader$

Au croisement, on a $t = t_f$ = durée optimal pour atteindre le croisement

$at_f^2 + bt_f + vitesseLeader = vitesseDuTrainOppose$ (1)

En intégrant le polynôme de départ, on obtient :

$a\frac{t_f^3}{3} + b\frac{t_f^2}{2} + vitesseLeader \times t_f = distanceLeaderCroisement$ (2)

La résolution du système d'équation de (1) et (2) permet d'obtenir les coefficients a et b du polynôme de vitesse.

La gestion du croisement est donc effectuée par une sorte de planification (à court terme) de la vitesse à adopter. On peut donc considérer, dans une certaine limité, l'agent Train comme un

agent cognitif.

3.5 Résultats

Une fois la solution du polynôme implémentée, une série de tests a été effectué pour vérifier l'efficacité du la méthode sur le long terme. En termes de paramétrage de la solution, il semble qu'une vitesse d'environ 80 pixels par secondes et une inter-distance égale à 6 fois la taille d'un véhicule donne de bons résultats. Si l'on augmente de trop l'inter-distance, on se retrouve avec le problème évoqué en 4.3. Une augmentation de la vitesse des robots nécessiterait une augmentation de l'inter-distance.

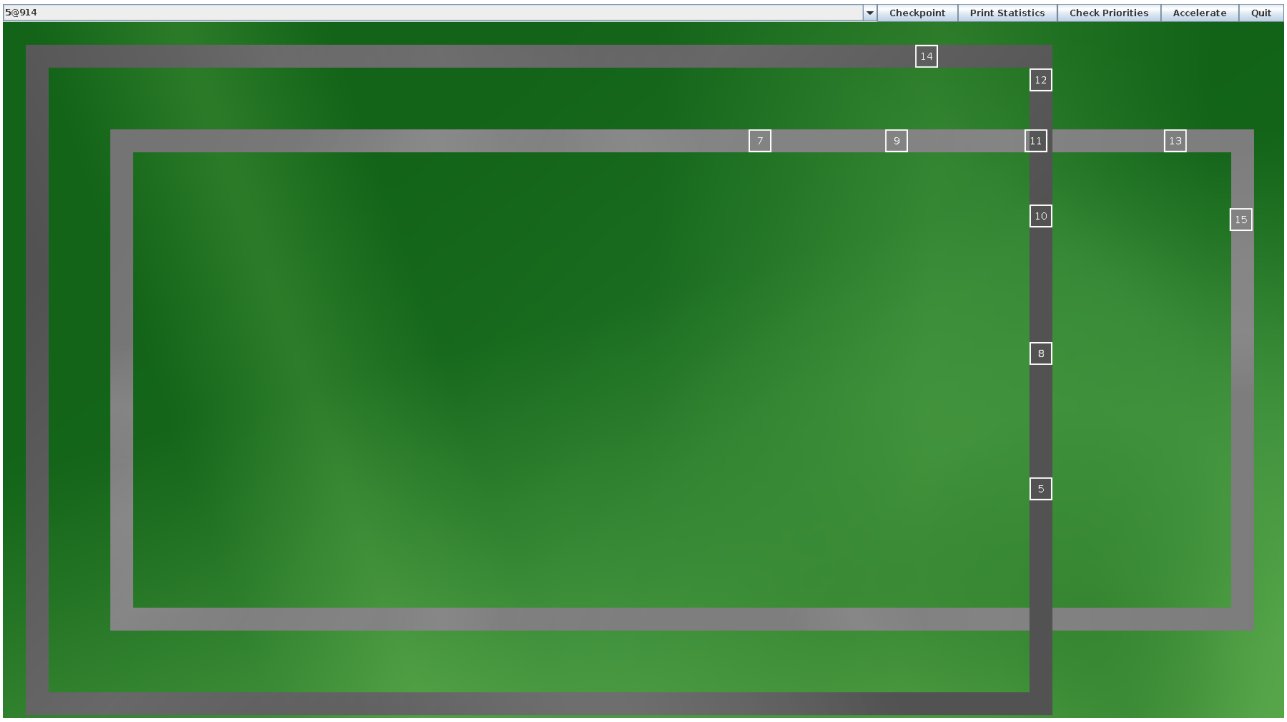

FIGURE 3.4 – Exemple de croisement

L'utilisation de la librairie JFreeChart par Florian Beck dans son projet permet d'obtenir, encore maintenant, des statistiques sur l'inter-distance et les vitesses effectives de chaque véhicule. On voit bien sur la figure 3.5 l'application du polynôme de vitesse au Leader du train 1, ainsi que l'application, après un temps de propagation, de polynômes plus "aplatis" sur les véhicules suiveurs.

Cette différence s'explique par le fait que les suiveurs n'appliquent leur vitesse qu'en réaction à ce qu'il se passe devant eux. Ainsi, au moment où le Leader atteint le sommet de sa courbe de vitesse et commence à ralentir, le deuxième véhicule perçoit une différence d'inter-distance plus faible et commence donc lui aussi à ralentir.

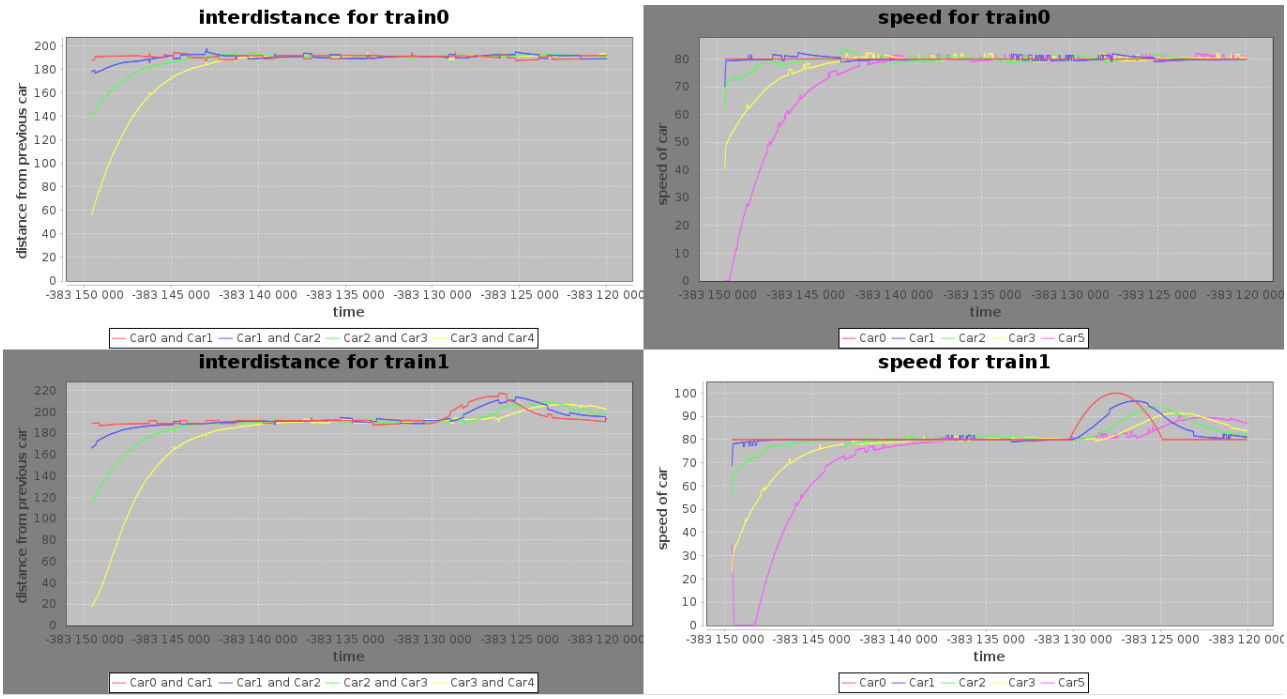

FIGURE 3.5 – Statistiques liés au croisement ci-dessus

4. Améliorations possibles

4.1 Évolution des vitesses de trains

Dans l'état actuel des choses, une vitesse principale est définie pour l'ensemble des trains. Lorsque deux trains se rencontrent au niveau d'une intersection, le deuxième train entrant calcul une vitesse moyenne (ou une fonction polynomiale de vitesse, voir section 3.4.2) et l'envoie à ses véhicules pour exécution. Une fois sortie de l'intersection, le train renvoie une consigne de vitesse avec la vitesse de base. Ainsi, les deux trains circulent la plupart du temps à la même vitesse, définie en dur dans le programme.

De nombreux essais ont été effectués pour tenter d'avoir deux sortes de vitesses : une vitesse générale et une vitesse, plus faible, spécifique aux intersections. Le principal problème rencontré lors de ces tests est le passage d'une vitesse à l'autre qui ne se fait pas immédiatement pour tous les véhicules. En effet, de par la nature des comportements de leader / suiveurs, un ordre de vitesse met un certain temps jusqu'à ce que le dernier véhicule atteigne la nouvelle vitesse. Cela avait donc pour effet de fausser les estimations effectuées par l'autre train pour la résolution des conflits de croisement, et donc rendait le système complètement inefficace.

Bien qu'il aurait été intéressant d'étudier la question, j'ai dû renoncé à me pencher sur ce problème par manque de temps.

4.2 Définition de l'inter-distance

L'inter-distance utilisée dans ce projet a été défini par une série de tests manuels. Une amélioration possible pour ce projet serait d'avoir une méthode pour définir l'inter-distance d'un train en fonction de sa vitesse et de la capacité pour un autre train de le croiser sans encombre.

4.3 Plus de deux trains

Pour l'instant, seulement des cartes contenant deux circuits, chacun avec un train en déplacement, ont été essayé. Le programme actuel ne permet en effet pas de gérer un troisième chemin

contenant un troisième train qui viendrait se croiser avec les deux autres.

De plus, des tests effectués à vitesse élevée ont permis de déceler un problème majeur qui viendrait directement impacter l'efficacité d'un système à plus de deux trains : Il se peut qu'un train ayant calculé un plan de traversé d'une intersection soit à son tour utilisé pour estimer la vitesse que doit prendre l'autre train. (voir exemple ci-dessous)

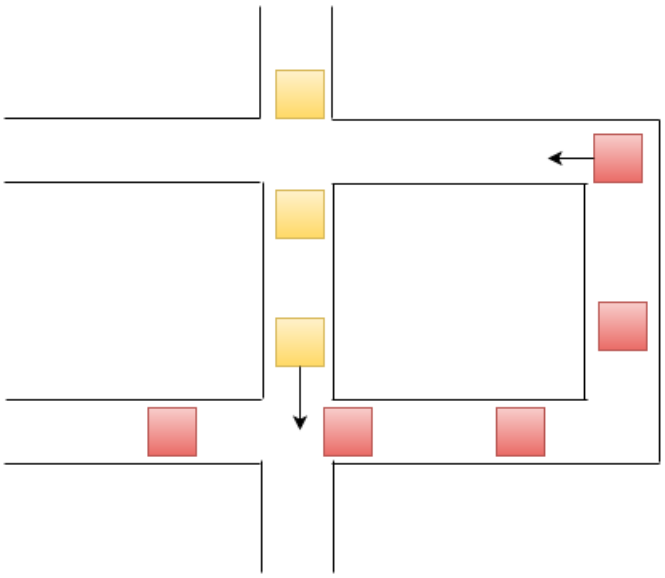

FIGURE 4.1 – Problème de croisement proches

Si on multiplie le nombre de circuits se croisant, on rapproche également les croisements. Il y a fort à parier que la situation précédente se produirait alors de nombreuses fois, à la manière de l'exemple ci-dessous.

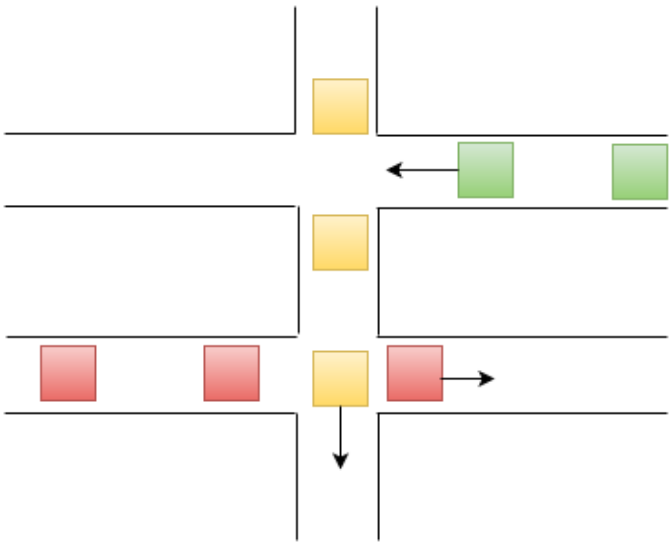

FIGURE 4.2 – Problème de croisement proches, appliqué à 3 trains

4.4 Plusieurs trains par voie

La gestion de multiples trains sur un même circuit est un problème tout aussi intéressant. On pourrait imaginer avoir deux circuits fermé, comme actuellement, mais où chaque circuit contiendrait non pas un, mais deux trains. Cela générerait un peu plus de complexité dans la gestion des intersections et le fonctionnement du Leader.

Il pourrait même être envisageable de n'avoir qu'un seul circuit, sous forme de huit, sur lequel circulerait tous les trains du système.

4.5 Rendre le système dynamique

Une des idées de base évoquée dans la définition du sujet est le fait de laisser le choix aux véhicules de changer de train lors de l'exécution du système. On pourrait en effet imaginer qu'un véhicule ayant comme objectif un point A décide de passer du train 0 au train 1 pour l'atteindre plus rapidement.

5. Conclusion

Le croisement de trains de véhicules est un sujet très intéressant, notamment lorsque l'on réfléchit aux applications dans le domaine des véhicules autonomes et aux questions de gestion de trafic que ces derniers vont générer.

Une quantité non négligeable de travail a été nécessaire à la compréhension et la correction du projet original. Des problèmes liés aux coordonnées stockées en pixels, ou encore aux arrondis effectués dans les déplacements ont été autant de problèmes à corriger qui ne faisaient pas réellement partie du but du projet.

La gestion du croisement a demandé également un certain travail de réflexion suite aux premières expérimentations (voir section 3.4.2).

Bien que le travail de conception et de simulation soit bel et bien fini, il reste encore du travail à effectuer : trouver une stratégie de modification de l'inter-distance en fonction de la vitesse, gérer plus de deux trains sur la carte...

Ayant décidé de coupler ce projet avec un autre module de logiciel embarqué, il me reste encore une lourde tâche à faire : l'implémentation du système sur de véritables robots.