

UNIVERSITÉ PARIS-DESCARTES

*Institut Français des Sciences et Technologies des
Transports et de l'Aménagement et des Réseaux*

RAPPORT DE STAGE

YATS : Yet Another Traffic Simulator

Auteur :
Julien GAGNEUX

Référent :
Dr. Julien SAUNIER

13 septembre 2013



Ces bricolages absurdes ne sont que des exercices,
mais comme on dit, pour arriver à l'intelligence artificielle,
il faut bien passer par l'imbécilité artificielle.

Hal Duncan,
Le Livre de Toutes les Heures ,
tome 1 - page 198

Remerciements.

Rien n'aurait été possible sans la bienveillance et le soutien constant de Mr Julien Saunier : mon encadrant. Aussi mes premiers remerciement lui sont destinés sans aucune réserve. Merci d'avoir compris certaines de mes difficultés et de ne pas m'en avoir tenu rigueur, de m'avoir critiqué et soutenu dans mes démarches de recherches, mais surtout merci pour m'avoir choisi pour ces 6 mois, je résignerais sans hésitation.

Je tiens aussi a remercier les doctorants Lancelot Six, Kevin Darty, Laurent Carafa ainsi que Abderraouf Zermane pour m'avoir aidé a construire mon modèle.

Je souhaite également remercier Sophie Lemonier, Kevin Darty, Céline Mateev et Laurent Carafa pour leurs distractions bienvenues, les controverses que nous avons partagé lors de pauses clope interminables m'ont fait traverser se séjour au LEPSIS sur un petit nuage. J'espère que nous sommes un peut plus que des collègues.

Merci aussi à ma Mère, Patricia, d'avoir supporter la relecture de ce rapport sans rien connaitre ou presque à l'informatique et de me dispenser de si bon conseil depuis si longtemps.

Enfin je voudrais remercier Oriane. Celle qui supporte mes étranges lubies, celle pour qui j'ai envi de me lever le matin et avec qui se coucher le soir est toujours un grand bonheur.

Table des matières

remerciement	2
Introduction	5
1 Contexte.	6
1.1 Présentation de l'ifsttar.	6
1.2 Redéfinition du sujet et problématique de recherche approfondi.	6
2 État de l'art et modèle théorique.	8
2.1 Agent intelligent.	8
2.2 L'encorporation de l'agent.	8
2.3 SMA	10
2.4 Simulation et Développement itératif.	10
2.5 Un rapport stigmergique avec le monde.	12
2.6 Environnement.	12
2.7 Affordance.	13
2.8 Mathématiques élémentaires	14
2.8.1 Algèbre linéaire	14
2.8.2 Trigonométrie et espace vectoriel	14
2.8.3 Homothétie et espace vectoriel	16
2.8.4 Accélération	16
3 Implementation.	18
3.1 Résumé schématique.	18
3.2 Choix et description des technologies utilisés.	19
3.2.1 Language.	19
3.2.2 IDE.	19
3.2.3 Bibliothèques.	19
3.3 Diagramme de classe.	20
3.3.1 Le point de départ - PointSim	21
3.3.2 Le réseau routier	21
3.3.3 InitialScreen et Time	23
3.3.4 Environnement et VueEnv.	23
3.3.5 Vehicle et Body.	24
3.4 Gestion du temps.	24
4 Perspectives algorithmiques.	25
4.1 Description de la prise de décision du virage du véhicule.	26
4.2 Description de la visibilité du véhicule.	27
4.3 Description du mouvement d'un véhicule, accélération ou freinage.	28

5	Fonctionnalités de confort.	30
5.1	Exemple de création d'un réseau routier.	30
5.2	Description du system de log du réseau routier.	31
5.3	Biometrie de l'application	33
5.4	Développement d'un benchmark pour les tests.	34
6	Perspectives et futurs développement.	41
6.1	Liste des fonctionnalités futurs	41
6.2	Comparaison des logs de véhicule avec un véhicule réel. . . .	43
7	Conclusion et Perspectives	44
	References	46
	Annexe 1 - Problème notable de développement.	47
7.1	Float == à un EPSILON près.	47
7.2	Rupture de cycle dans la déclaration des headers.	47
	Annexe 2 - logiciels utilisé pour construire ce document	49

Introduction.

La Simulation Informatique offre des perspectives de recherche ainsi que des possibilités de prédiction étonnantes. Par essence la simulation est au réel ce qu'est un traducteur à une œuvre littéraire : un traître.¹

Elle trahit la réalité dans le sens où celle-ci est "tordue" afin de convenir à un modèle lui même implémenté, donc lui aussi soumis à certaines lois comme le langage ou la plateforme sur lequel il est développé. Mais malgré ces entorses au réel, le développement d'un modèle est un moyen extrêmement efficace pour les chercheurs de comprendre plus en détail et plus en profondeur l'objet de leur recherche.

Ici l'objet de nos recherches sera : "*une modélisation agent des niveaux tactiques et opérationnels de la conduite, application au comportement latéral d'un véhicule*"². Ou, en raccourcissant : l'implémentation d'une nouvelle simulation de trafic : Yats.

Nous approfondirons tout d'abord le contexte de ce stage dans la partie 1, nous présenterons notamment l'IFSTTAR (l'institut dans lequel j'ai eu la chance d'effectuer ce stage) et nous aborderons les cause et conséquence de la re-formulation de sujet du stage ce qui nous permettra de clairement préciser la problématique.

A la suite de cela, de la partie 2 nous verrons l'état de l'art actuel sur les simulateurs, la simulation, les systèmes multi-agents, les équations utilisées ainsi que certains concepts et définitions clés ayant orienté la conception du modèle.

Dans la troisième partie nous aborderons l'aspect pratique c'est à dire l'implémentation concrète du modèle théorique. Nous y verrons certaines fonctionnalité, les technologies utilisées, les subtilités de développement et les solutions mises en place.

Certaines fonctionnalité ne peuvent être correctement décrit que dans une démarche pédagogique. Nous verrons donc dans la quatrième partie un manuel succinct d'utilisation, la présentation du benchmark et une biométrie légère de l'application.

Dans la cinquième et dernière partie de ce rapport nous nous intéresserons à ce qui pourrait être (ou qui sera) fait dans le futur ; que ce soit le perfectionnement de certains objets et concepts ou encore la comparaison entre les trajectoires des véhicules de YATS avec des trajectoires de conducteur réel.

Pour finir nous conclurons.

1. "Traduttore, traditore", littéralement : "traducteur, traître", est une expression italienne. Cette expression est une paronomase qui joue sur la ressemblance des deux mots. elle est souvent utilisée même dans d'autres langues, en raison de la concision de ce jeu de mots.

2. la fiche explicative du stage tel quelle à été diffusé se trouve en Annexe 1

1 Contexte.

1.1 Présentation de l'ifsttar.

J'ai eu la chance d'effectuer mon stage à l'IFSTTAR (*Institut Français des Sciences et Technologies des Transports et de l'Aménagement et des Réseaux*). Cet établissement public placé sous la tutelle conjointe du *Ministère de l'écologie, du Développement durable et de l'Energie*, et du *Ministère de l'Enseignement Supérieur et de la Recherche*.

Le but de l'Ifsttar est de conduire des travaux de recherche et d'innovation dans le domaine des transports, des infrastructures et des risques urbains.

Comprenant plus de 1200 agents³ l'institut est composé de plusieurs départements, composés eux même des plusieurs laboratoire. Pour ma part je me suis joint au département COSYS (*Composants et Systèmes*) dans le laboratoire LEPSIS (*Laboratoire exploitation, perception, simulateurs et simulations*) ayant pour directeur : M Didier AUBERT.

Le LEPSIS mène des recherches dans trois domaines complémentaires a la limite entre ingénierie et science humaines :

- Vision et Automatique Appliquées aux trajectoires,
- Systèmes Intelligents Appliqués aux Déplacement,
- Réalité Virtuelle appliquée aux simulateurs de conduite.

Je fut placé sous la tutelle de Mr Julien SAUNIER, Docteur en Informatique, maître de conférence à l'INSA-Rouen et chargé de recherche à l'IFSTTAR.

1.2 Redéfinition du sujet et problématique de recherche approfondi.

Pour mieux cerner la problématique de ce stage nous allons commencer par dresser un petit historique du LEPSIS.

Il y a une vingtaine d'année fut développé le simulateur de conduite et de trafic : ARCHISIM. L'initiative du projet fut lancé par M Stephane Espié au début des années 1990. Cela fait donc plus de 20 ans et durant ce temps les techniques de génie logiciel et les langages ont considérablement évolué.

Archisim fut développé en Modula-2⁴, langage dont l'apprentissage n'est pas très rependu. De plus Archisim fut développé majoritairement par des doctorants ou des stagiaires souhaitant en priorité affirmer (ou infirmer) leurs hypothèses. Ceci eu pour conséquence de ne pas avoir de réel suivi du

3. chiffres de 2011, source : [http ://www.ifsttar.fr/linstitut/colonne-1/lifsttar/quelques-chiffres-cles/](http://www.ifsttar.fr/linstitut/colonne-1/lifsttar/quelques-chiffres-cles/)

4. Langage de programmation créé en 1977 par Niklaus Wirth. Il est compilé, procédural, fortement typé et modulaire.

projet en terme de version ou de procédure de test. Actuellement il est donc difficile de rentrer dans le code d'Archisim ce qui est un frein a la recherche.

Il serait donc bienvenu d'avoir un logiciel plus léger, plus facile a appréhendé afin d'y tester certaines hypothèses sans pour autant avoir a apprendre un nouveau langage et rentrer dans une application complexe.

Avec le sujet de Mr saunier : Une modélisation agent des niveaux tactique et opérationnel de la conduite, en mettant l'accent sur le comportement latéral du véhicule, le sujet n'était pas uniquement de monter un modèle pour que le comportement latéral du véhicule soit plus cohérent avec la réalité mais également de produire un outil de prototypage.

En terme de recherche bibliographique la tache est importante. Chaque fonctionnalité de la simulation (par exemple l'accélération, le modèle multi-agent, etc..) a fait l'objet de nombreuses recherches et publication. Il m'aura fallu lire des dizaines d'articles et parler de nombreuses heures avec les chercheurs du LEPSIS avant de commencer à cerner le sujet.

En ce qui concerne l'implémentation, la technique pure, la aussi les choix du langage et des bibliothèques orienta fortement la manière de concevoir la plateforme et la mise en œuvre du le modèle.

Bien sur développer un logiciel implique forcément de développer certaines spécifications extra-fonctionnelles (ou fonctionnalités pour la plateforme et non pour le modèle), nécessaires mais chronophages, comme la gestion des logs ou le développement de réseau routier.

Finalement ce qui devait être une étude sur le la modélisation du comportement latéral du véhicule se transforme en un développement d'envergure d'une application pour développeur et chercheur nommé : "YATS"⁵ qui permettra par la suite de concevoir des modèles répondant a de nombreuse question, dont celle initialement prévu pour ce stage.

5. Acronyme de : "Yet Another Traffic Simulator"

2 État de l'art et modèle théorique.

Nous aborderons dans cette partie les concepts théoriques clés afin de bien comprendre les choix techniques de la partie 3. Nous justifierons nos choix par une bibliographie concise.

Le choix du type de simulation s'est immédiatement orienté vers un système multi-agent. Seulement concevoir un SMA repose sur un paradigme en développement n'ayant pas encore fait concrètement ses preuves dans le secteur privé ou dans l'industrie comme l'a fait le paradigme objet. Il est des lors facile de tomber dans des pièges que seule l'expérience permet d'éviter, d'où une étude approfondie de la littérature sur le sujet. Cependant nous resterons succincts dans nos définitions.

Débutons par la définition d'un Agent.

2.1 Agent intelligent.

Il est difficile de trouver dans la littérature scientifique une définition pour définir ce qu'est un agent intelligent et de faire l'unanimité parmi les chercheurs. Nous resterons donc classique en donnant une définition généraliste de Wooldridge et Jennings [6] :

"Un *agent* est un système informatique situé dans un *environnement*, étant capable d'*actions autonomes* dans le but de réaliser les objectifs pour le(s)quel(s) il a été conçu."

De cette définition découle quelques propriétés :

- Si un agent vit dans un environnement il est donc **situé**.
- Il prend ses propres décisions : il est donc **Autonome**.
- Il *vit* dans le but de réaliser ses objectifs personnels.

Cependant il manque une part importante de ce qui constitue l'agent : **la rationalité**. Prenons par exemple un agent stupide ayant pour but de rejoindre un point quelconque. Cet agent a le choix entre plusieurs chemins mais celui-ci prend la décision de tourner à droite à chaque intersection. Nul doute que cet agent risque de tourner en rond longtemps. Dès lors, et c'est en quelque sorte l'un des buts de l'intelligence artificielle : un agent se doit d'être rationnel pour en porter le nom. Bratman [2] et [1]

2.2 L'incorporation de l'agent.

Cette partie touche plusieurs matières scientifiques et est donc sujet à de grandes controverses. L'aborder reste cependant fondamentale pour le domaine de l'intelligence artificielle, mais aussi parce qu'elle a orienté la conception du modèle.

A l'origine des ses questionnement se trouve l'ouvrage des professeurs Pitti et Pfeiffer [8]. Ce livre propose de changer un peut notre vision de l'agent en tant qu'entité logique pur et dur. Ils avancent le fait que l'encorporation de l'agent (le fais que l'agent aie un corps à lui) physique (pour le cas des robots) ou virtuel (pour le cas des simulations) change les rapports qu'entretient l'agent avec le monde et lui même.

Ceci implique de très nombreuses réflexions, notamment celle de l'apprentissage. Nous (en tant qu'agent intelligent et rationnel) avons appris à utiliser notre corps. Notre cerveau (notre artefact de la décision) à lui aussi appris à force d'innombrable répétitions les conséquences physiques de nos actions sur le monde. Changez de corps et toutes ces perceptions et influes devront être adaptés et réappris.

Pourtant, et a l'insu de nos propres choix formulé rationnellement par notre cerveau, nôtre corps prends en permanence ses propres décision. Pitti & pfeifer avancent ici qu'en fin de compte si fiers de nos décisions, nous en oublions que beaucoup sont prises par notre corps, sans que l'on soit finalement consulté.

Un exemple : songez à vôtre pas. Vous marchez juste calmement dans la rue. Nul doute que vous avez pris la décision d'aller quelque part initialement, cependant pensez vous à chacun des éléments de décision que nécessite la marche ? Je ne pense pas mais soyons rigide et admettons que nous pouvons en effet segmenter la marche en une suite d'actes simple.

Lever la jambe → plier le genoux → poser le talon → transférer le poids sur l'autre pied → etc..

Néanmoins, avez vous conscience que vos muscle se détendent lors de la descente de votre pieds vers le sol en utilisant la gravité de la terre afin de faire moins d'effort ? Personne sur terre ne pense à cela à chaque pas effectué, et heureusement car notre tête exploserait littéralement sous la somme des calculs qu'il serait nécessaire de faire. C'est dans ce sens la que Pitti & pfeifer propose de repenser l'encorporation de l'agent.

Avec un corps c'est énormément de processus de bas niveau qui se doivent d'être automatiser, par l'apprentissage ou par quel-qu'autres moyens. l'obtention, ou la conceptions d'un corps propulse l'agent à un niveau de rationalité bien supérieur étant donné qu'il n'a qu'a se focaliser exclusivement sur les taches de réflexions et/où de décision.

La porté philosophique de cette hypothèse est tellement large que nous n'en débattons pas ici. Recentrons nous donc sur notre simulateur de trafic. Le but ici est de pouvoir changer de corps suivant les paramètres. Par changer de corps nous n'entendons pas changer de corps en cours de simulation évidemment, mais qu'a chacune des sources (nous y reviendrons) faisant naitre des véhicules, ceux-ci puissent avoir des caractéristique et des méthodes différentes.

Car au final c'est cela un corps : un ensemble d'attributs et de méthodes propres.

2.3 SMA

Un SMA (Systeme Multi Agent ou MAS : *Multi Agent System* en anglais) se distingue d'un système informatique classique par bien des aspects. selon Draa [4] un SMA se distingue d'un programme classique dans la mesure ou il n'y a pas de contrôle global du SMA.

Chaque agent est un fragment du système global et rempli sa tâche qui est une sous tâche de la tâche global.

Un SMA est donc composé d'entités artificielles intelligentes : les agents.

Ces agents évoluent dans un environnement dont ils font partie intégrant (nous reviendrons sur l'environnement par la suite).

Ces agents se doivent d'avoir des interactions entre eux dans le sens ou les agents seront mis en relation pour l'accomplissement de leur tâche, et Cela même dans le cas d'agents "autistes".

Un groupe d'agents (même petit) est par définition une société avec son organisation et ses règles.

Nous venons de voir ici une liste succincte des principales caractéristique d'un SMA. Pourtant ce que nous désirons construire est un logiciel de simulation qui lui aussi répond a des caractéristiques précises.

2.4 Simulation et Développement itératif.

Nous touchons ici a un sujet extrêmement sensible doté d'une littérature très abondante. Commençons par définir ce qu'est une simulation.

La première définition fut celle de Shanon (1976) [10] qui désigne la simulation comme :

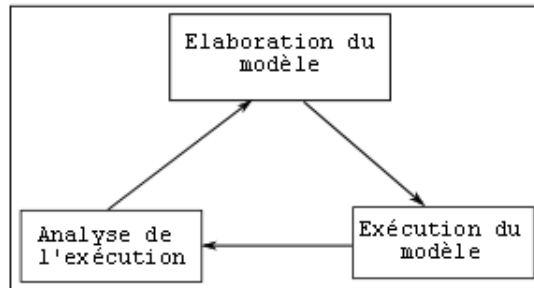
Un processus dont le point de départ est de concevoir un modèle a partir d'un system réel.

Ceci met en lumière 2 concepts clés : le système réel (que l'on cherche a reproduire artificiellement) et le modèle (l'implémentation du système réel).

Cela nous amène a la définition proposé par Fiswick (1994) [5] pour qui :

La simulation est un processus de conception itératif, cyclique et non-linéaire dont le point central est le modèle réalisé a partir d'un système.

Le schéma ci-dessous (aussi extrait de [5]) illustre parfaitement cette définition.



Nous remarquons donc que créer une simulation apparaît plus compliqué qu'il n'y paraît au premier abord.

Reformulons : nous souhaitons réaliser un logiciel d'un type particulier (une simulation) dont le développement engendre une technique de développement particulière (un développement itératif).

Le cycle itératif offre plusieurs avantages notamment la stabilité du projet et une meilleure gestion des risques. Cependant ce type de modèle de développement présente aussi des inconvénients comme sa lenteur (surtout à cause de la masse de test à effectuer à chaque itération). De plus comme nous sommes dans le cadre du développement d'une simulation le modèle en lui-même n'est pas totalement défini et il y a de grande chance que le cycle " n " doive être complètement réécrit car incompatible avec la l'élaboration du cycle " $n + X$ ".

Nous pouvons, à titre informatif lister l'ensemble des cycles traversés durant ces 6 mois de stage :

- Phase préliminaire : Reformulation du sujet - apprentissage des technologies - maquette.
- Cycle 1 : essais de véhicule respectant approximativement la physique 2D de base.
- Cycle 2 : essais de route et manipulation de la SDL.
- Cycle 3 : élaboration approfondie - Développement des fichiers source externes puis d'une technique de développement du réseau routier.
- Cycle 4 : développement d'un comportement réaliste des Véhicules/Agents. Accélération, ralentissement et virage.
- Cycle 6 : Approfondissement des fonctionnalités graphiques. déplacement, homothétie.
- Cycle 7 : Approfondissement des fonctionnalités des Sources

Nous reviendrons sur beaucoup des points listés ci-dessus mais ce qu'il est important de garder à l'esprit c'est que développer une simulation nécessite

la mise en place de procédure de développement spécifiques rigides si l'on ne veut pas se retrouver avec un système instable avec de erreurs impossible a résoudre a l'itération suivante.

Il est important de dire aussi que j'ai "naturellement" adopter un cycle itératif et que la découverte d'une technique de développement propre aux simulations est arrivé milieu fin de stage.

Continuons maintenant de décrire les concepts théoriques qui on orientés le développement de YATS.

2.5 Un rapport stigmergique avec le monde.

Tout d'abord définissons ce qu'est la stigmergie :

La stigmergie est une méthode de communication indirecte dans un environnement émergent auto-organisé, où les individus communiquent entre eux en modifiant leur environnement. [3] et [14].

Les fourmis sont un parfait exemple pour illustrer ce qu'est concrètement la stigmergie. Lorsque celles-ci se déplacent elles déposent des phéromones afin d'indiquer où elles sont passées. Les autres fourmis, lorsqu'elles voudront emprunter ce même chemin s'apercevront qu'une autre fourmi est déjà passé par là car elle a modifié son environnement et cette modification a été perçue. Les deux fourmis ont donc bien communiqué les unes avec les autres de manière indirecte en modifiant leur environnement.

Dans le cadre de notre simulation de trafic cette notion apparait comme fondamentale, les véhicules ne pouvant évidemment pas communiqué directement en utilisant des requêtes personnelles, ceux-ci devront modifier leur environnement.

Nous pouvons noter que lors de l'exercice de la conduite (dans la réalité et par des humains donc) c'est exactement ce qui se produit. Les conducteurs ne discutent pas ni pour obtenir des informations, ni pour prendre des décisions et cela même à l'arrêt. Lorsqu'un conducteur va tourner à gauche celui-ci modifie l'environnement en plaçant une balise visuelle clignotante lumineuse de couleur orangée sur le côté gauche de son véhicule. Les autres conducteurs perçoivent cette balise et en interprètent l'information : "-ce conducteur va tourner à gauche". les conducteurs se sont transmis une information à l'aide de l'environnement sans pour autant qu'il y ait eu de requête directe.

Dans notre simulation de trafic nous appellerons ces balises des **tags** [9].

2.6 Environnement.

L'environnement est l'un des éléments centraux d'une simulation basée sur un système multi-agents. Seulement comment le construire ? quel sont les propriétés qu'il doit posséder ou ne pas posséder ? quel sont les erreurs à éviter ?

A ma connaissance il n'existe pas de "manuel de développement de l'environnement idéal". Nous baserons donc nos réflexions sur trois documents de références de Weyns : 2005 [14] 2006 [12] et 2007 [13]

Nous pouvons déjà distinguer deux perspectives d'environnement :

- l'environnement pour un system d'agent situer ; c'est celui qui va nous intéresser
- l'environnement pour un system d'agent cognitif ; dans ce cas, si nous pouvons nous permettre une comparaison il représente un conteneur favorisant les actions de l'agent (ex : communication, taches externes, etc..).

Dans le cas de l'environnement pour des agents situés, celui-ci doit pouvoir permettre de faire des *actions situées*. Cela nous concerne tout particulièrement pour notre simulation de trafic ou lorsqu'un Agent tourne à gauche il le fait localement. L'Agent évolue donc dans un *monde externe* a part entière.

Nous avons parlé un peu avant de stigmergie dans lequel l'environnement joue le rôle de moyen de coordination et de communication. C'est donc un élément central du projet.

Nous venons de survoler ce qui devrait composer l'environnement. Nous prendrons comme référence La Définition que donne Weyns dans [12] (non traduit pour dénaturer le moins possible le sens) :

"The environment is a first-class abstraction that provides the surrounding conditions for agents to exist and that mediates both the interaction among agents and the access to resources."

Cette définition implique beaucoup de chose, notamment que l'environnement est essentiel pour l'existence de l'agent. Il structure sa vision du "monde", ses échanges et les actions qu'il permet ou non de faire. Il est en quelque sorte le garant de la cohésion "*sociale*" et "*physique*" du système.

Nous pouvons dégager de cela trois façon pour l'Agent d'interagir avec l'environnement :

- **Perception** : l'Agent perçoit localement son environnement.
- **Action** : l'Agent peut exercer des actions sur l'environnement.
- **communication** : l'Agent communique avec les autres agents par le biais de l'environnement.

L'environnement est donc un véritable support pour le système multi-agents.

2.7 Affordance.

Ce concept : l'affordance, a largement influencé la conception du modèle de notre simulateur. Ce concept fut développé initialement par le psychologue James J. Gibson [7] en 1977 et a été repris par la suite par de nombreux chercheurs dans de nombreuses matières, notamment en ergonomie

définition de l'affordance :

L'affordance est la capacité d'un système ou d'un produit à suggérer sa propre utilisation.⁶

Étant donné que dans notre modèle, qui orienté SMA, les agents vont utiliser des objets de leur environnement, ces objets se doivent d'avoir une utilisation "ergonomique". Nous reviendrons sur un exemple d'utilisation du concept d'affordance dans la partie pratique.

2.8 Mathématiques élémentaires

Yats peut se définir en quelques sorte comme un cas concret d'utilisation des concepts mathématiques de base. les références dans les code y sont fait en permanence que ce soit dans les procédures de décision, d'application des éléments physique ou encore de l'affichage.

Nous allons commencer au plus bas niveau de l'algèbre linéaire en passant par la trigonométrie et les espaces vectoriels. Nous verrons enfin un problème physique concret : l'accélération.

2.8.1 Algèbre linéaire

- L'équation d'une droite dans un plan se présente sous la forme :

$$y = ax + b$$

- Lorsque nous n'avons que deux points pour déterminer l'équation d'une droite :

$$\cdot PA(x_a, y_a)$$

$$\cdot PA(x_b, y_b)$$

$$a = \frac{y_b - y_a}{x_b - x_a}$$

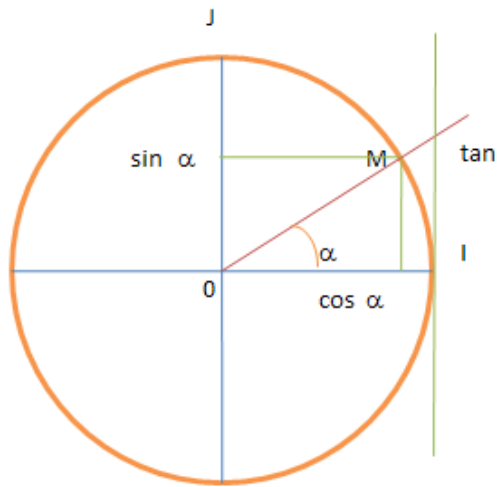
$$b = y_a - (a * x_a)$$

- Lorsque nous devons déterminer la distance cartésienne (à vol d'oiseau) entre deux points

$$\sqrt{(x_b - x_a)^2 + (y_b - y_a)^2}$$

2.8.2 Trigonométrie et espace vectoriel

- Lorsque nous n'avons qu'un point et un angle pour déterminer l'équation d'une droite nous allons devoir utiliser le cercle trigonométrique :



Il y a bien sur énormément de choses à dire sur ce schéma mais le plus important à retenir ici est que :

$$\vec{OM} = \cos(\alpha)\vec{OI} + \sin(\alpha)\vec{OJ}$$

nous allons nous servir de ceci pour projeter un point $P(x_1; y_1)$ à l'emplacement voulu selon l'angle α (en degré) sur une distance (\vec{OM})

$$x_2 = x_1 + (\vec{OM} * \cos(\alpha * \frac{\pi}{180}))$$

$$y_2 = y_1 + (\vec{OM} * \sin(\alpha * \frac{\pi}{180}))$$

- Déterminer l'angle β d'une droite par rapport à l'axe des abscisses :

$$\beta = -\arctan(pente) * \frac{180}{\pi}$$

- Pour déterminer l'angle entre deux droites :

Étant donné deux droites dont nous pouvons connaître l'angle par rapport à l'axe des abscisses :

$$diff(\beta_{d1}, \beta_{d2})$$

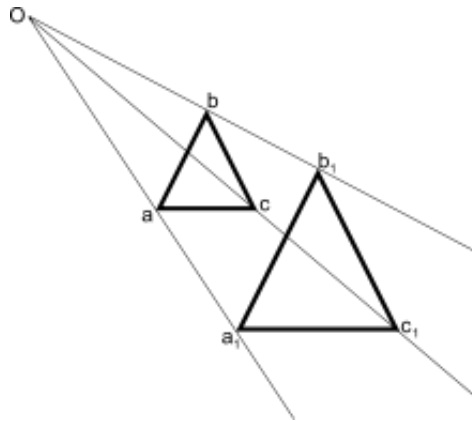
En pratique cela implique quelques subtilités algorithmiques qu'il n'est pas nécessaire de développer ici.

2.8.3 Homothétie et espace vectoriel

Zoomer ou Dé-zoomer revient concrètement à changer d'échelles.

En géométrie nous parlons d'homothétie dans un espace affine, c'est à dire que pour tout les points "P" de notre repère et selon le centre "O" et de rapport "r" (non nul) nous avons $\vec{OP} = r * \vec{OP}$

l'image ci-dessous exprime très simplement se concept.



Nous pouvons très simplement mettre en place cette formule par :

pour un point : $P(x_p, y_p)$

pour un point d'Origine (ici le curseur de la souris) : $O(x_o, y_o)$

pour un facteur : r

$$x_p = r * (x_p - o_x) + o_x$$

$$y_p = r * (x_y - o_y) + o_y$$

2.8.4 Accélération

Réussir à trouver une formule crédible pour l'accélération et la décélération (qui est aussi une accélération négative) fut un des plus gros challenge de ce projet. Certains passent une thèse à y réfléchir, ici nous n'avions pas ce luxe.

Plus encore, déterminer un moyen de faire avancer un véhicule de manière crédible n'étant pas une fonctionnalité principale du projet il fallut la réaliser rapidement. Mais après la lecture d'innombrable article peut stimulant j'ai été chercher de l'aide, j'avais avant tout besoin d'inspiration. Tout ceci fut débloqué grâce au léger coup de pouce de Mr Roland BREMOND, chercheur au LEPSIS, après avoir discuté d'une unité de mesure : le mettre par

seconde carré, ou ms^2 . C'était finalement si simple.

reprenons la problématique de cette partie : nous voulons que nos véhicules avance et freine de façon crédible.

concrètement, l'accélération correspond à la variation de la vitesse entre deux mesures de temps. nous pourrions aussi noter cela ainsi :

$$a = \frac{\delta_v}{\delta_t}$$

Où a est l'accélération, v est la vitesse et t est le temps (nous reviendrons sur le temps dans la partie 3 : "implémentation").

Ce qui nous intéresse tout particulièrement ce n'est pas d'avoir la formule ultime, mais bien de connaître la position de notre véhicule après un pas de temps. Et pour cela aussi il y a une formule : celle de la distance parcourue en fonction d'une durée :

$$d = d_0 + v_0 \delta_t + \frac{a \delta_t^2}{2}$$

d_0 représente le déplacement initial, v_0 la vitesse initiale, Δ_t la durée du trajet et a l'accélération

Que nous restait-il à faire ? Déterminer la valeur de a . La encore c'est Mr BREMOND Roland qui nous indiqua le chiffre d'or : 5.

Il est cependant intéressante de savoir que a dans la plupart des cas ou cette formule est utilisé, a vaut 9,81, soit la valeur de la gravité.

La formule que nous avons donc utilisé est très proche de celle de la distance parcouru par un solide dans l'espace pour un temps donné. Ce qui n'est au final pas surprenant.

L'implémentation de la formules est beaucoup plus facile a comprendre que la formule, même si ce n'est pas exactement sous cette forme qu'elle apparait dans le source de l'application.

```
1
2      // frequence d'echantillonnage en secondes
3      const double dt = 0.5;
4      const double accel = 9.81; // m/s carre
5      double speed = 0.0;        // m/s
6      double position = 0.0;     // m
7      double time;
8      for (time = 0.0; time < 2.5; time += dt){// end at 2.5s
9          position += (accel * dt * dt / 2.0) + speed * dt;
10         speed += accel * dt;}
```

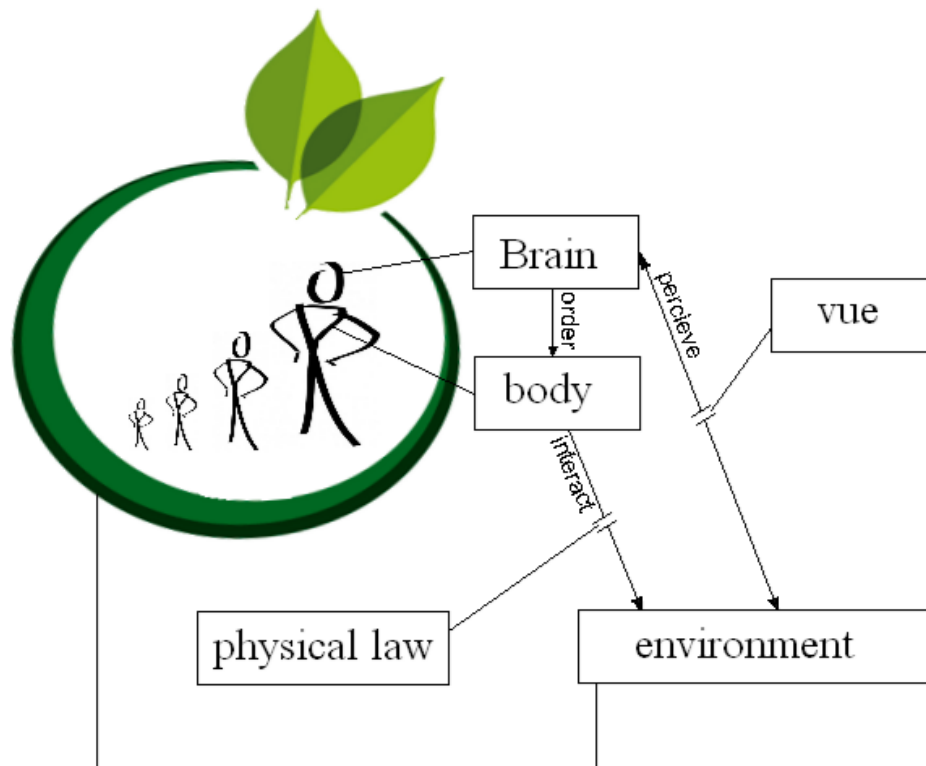
Avec ce segment de code nous allons basculer de chapitre et passer a l'implémentation du projet.

3 Implementation.

Avec ce chapitre nous allons nous confronter a des problématiques bien plus concrète Comme le choix des technologies ou l'architecture mis en place.

3.1 Résumé schématique.

Le Schéma ci-dessous nous fourni une vision simpliste mais réaliste de ce qui a voulu être construit.



Nous sommes dans le cadre d'un système multi-Agent. Chaque Agent est composé d'un *Brain* comme entité logique et d'un *Body* afin de pouvoir changer de véhicule dans le futur, par exemple utiliser des camion au lieu de voiture, changer la taille des voiture, etc.

L'agent interagi avec l'environnement dans lequel il doit respecter les lois physiques.

Afin d'augmenter le réalisme et comme il est clairement stipulé dans l'article de Weyns [14], il serait vue comme une erreur d'envoyer tout l'environnement en argument brute à l'agent. Nous avons donc créer une Vue qui correspond a un sous ensemble d'information provenant de l'environnement. L'agent perçoit sont environnement localement grâce a cette vue.

3.2 Choix et description des technologies utilisés.

Dans cette sous partie nous allons voir rapidement le langage que nous avons utilisé, dans quel environnement de développement et avec quelle bibliothèque.

3.2.1 Language.

L'un des but énoncé dans la partie 1.2 était de pouvoir appréhender plus facilement le code pour les chercheurs du LEPSIS. Pour cela il y a un moyen assez simple : utilisé un langage que tout le monde connaît. Nous avons donc choisi le C++.

Beaucoup de monde connaît le célèbre langage développer par Bjarne Stroustrup⁷, de plus il permet un développement d'une extrême finesse, des performances ainsi qu'une gestion de la mémoire optimale.

3.2.2 IDE.

Après quelques essais et difficulté nous avons opté pour Microsoft Visual C++ Express Édition car nous bénéficions d'une meilleur expertise de base sous cette IDE.

Il n'y a pas grand chose à dire de plus sur l'IDE, passons rapidement aux bibliothèques que nous avons utilisé.

3.2.3 Bibliothèques.

Évidement nous avons utilisé quelques une des classes de la STL.

Cependant il nous fallait représenter graphiquement nos véhicules et leur mouvements. Ceci offre plusieurs avantage, par exemple de pouvoir contrôler visuellement leur déplacement ou vérifier la fiabilité instinctive que nous avons du modèle.

Après quelques discussion nous avons opté pour la SDL.



SDL est l'acronyme de "Simple DirectMedia Layer". C'est une bibliothèque Cross-Platform permettant un accès bas niveau au son, clavier, souris, et dispositif d'affichage comme les écrans.

C'est une bibliothèque très performant ayant largement fait ses preuves, notamment grâce aux jeux vidéo⁸. Elle nous permettra de manipuler des

7. Est-il vraiment nécessaire de présenter Bjarne Stroustrup? Écrivain et professeur des Sciences Informatiques Danois. Inventeur notamment du langage C++.

8. pour ne citer qu'un seul exemple célèbre : mario fut développer en SDL.

Surface (ensemble de pixels rectangulaire à l'écran). Et c'est tout...

SDL est une bibliothèque très bas niveau ce qui implique qu'elle est développer en C. Son encapsulation dans des classe doit donc veiller au strict respect de l'ordre des appels des fonctions de base.

Certaines fonctionnalités que nous voulions développer n'était pas initialement possible avec la SDL. Nous avons donc deux choix. Soit redevelopper les fonctions qui nous intéressaient soit aller piocher dans les SDL_XXX. Ces bibliothèque fonctionne en annexe de la SDL par exemple SDL_ttf permet de manipuler du texte dans les images.

Nous avons besoin de deux éléments supplémentaire. Tout d'abord pouvoir faire des rotation de surface et des zoom. Nous avons donc rajouter au projet la bibliothèque SDL_GFX. Celle-ci inclue l'objet RotoZoom_Surface qui permet, comme son nom l'indique, de faire des rotations et des zooms.

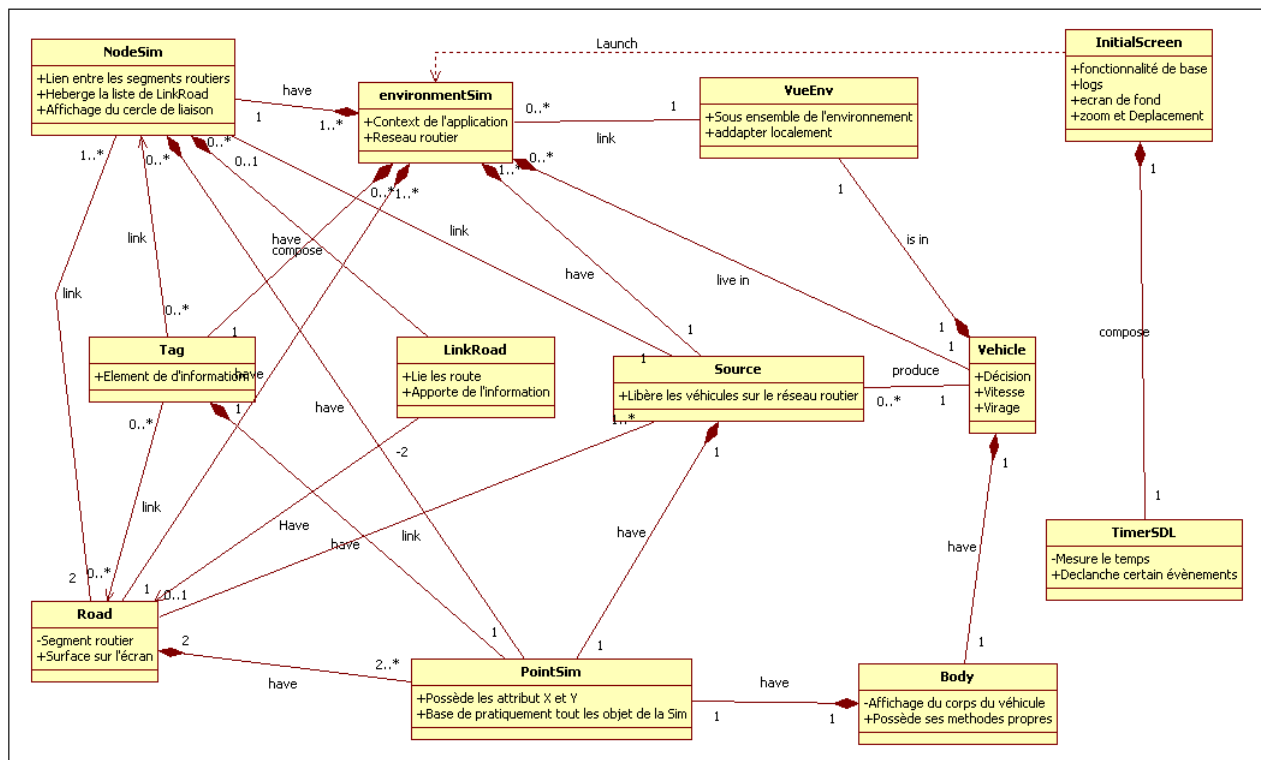
Nous avons aussi Besoin de faire des formes qui ne soit par rectangulaire : des ronds. SDL_Draw⁹ permet de faire toutes sortes de formes, cependant ce n'était que les rond qu'il nous fallait. Nous avons donc piocher uniquement un segment de code permettant de faire des surfaces rondes dans cette bibliothèque.

3.3 Diagramme de classe.

Le Schéma des classe si dessous nous offre une vision de l'architecture de l'application. Nous avons volontairement nommé celui-ci schéma de classe car cela n'est pas un diagramme de classe a proprement parler, il n'y a ni les attributs ni les méthodes. Ce schéma ne nous donne que le rôle global de chaque classe.

Nous avons préférer cette affichage plutôt qu'un diagramme de classe classique qui était devenu illisible, certains objets possédant plus d'une dizaine d'attribut et/ou de méthode.

9. développé par : Mario Palomo, José de la Huerga, Pepe Gonzalez et libre de droit




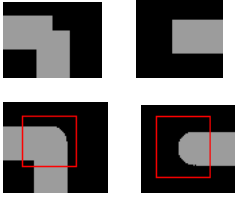
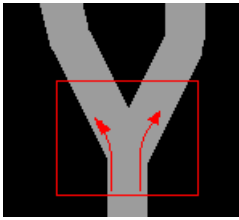
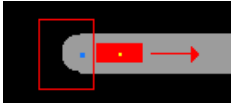
Nous allons maintenant passé brièvement en revue chaque objet. Cette démarche rébarbative nous permettra d'avoir une meilleur compréhension de l'organisation global du projet.

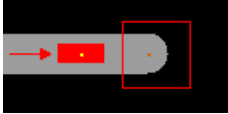
3.3.1 Le point de départ - PointSim

L'intégralité du projet repose sur cette classe très simple. Elle correspond a un point dans un repère en deux dimension. Elle possède donc deux float, un pour l'axe des abscisse, un autre pour l'axe des ordonnée.

3.3.2 Le réseau routier

Nous allons voir toutes les classe qui composent le réseau routier, nous en décrivons le rôle dans l'application puis nous l'associeront avec leur élément graphique.

Classe	Rôle	Image
Road	La Classe qui porte le nom "Road" est en faite un <i>segment de route</i> . Le réseau routier est donc composé d'un assemblage de segment de route. Cet objet possède un point en entrée et un point en sortie. Sur un segment de route un véhicule ne pourra aller que dans un seul et unique sens. Il possède d'autres attributs utiles pour d'autre objets comme l'angle par rapport a l'axe des abscisses ou la longueurs.	
NodeSim	Le Nœud est ce qui relie les segments de route entre eux. Il n'a donc qu'un seul point mais peut avoir plusieurs entrées ou sorties. Le Nœud héberge aussi une liste de RoadLink, que nous verrons juste après. Cet objet possède aussi une fonction qui permet de faire un cercle plein de couleur grise afin de combler l'espace entre les segments de route, comme monter sur l'image ci-contre.	
RoadLink	Le RoadLink, comme son nom l'indique, lie deux segment de route ensemble. Il fait partie du nœud mais est a l'usage du véhicule. Il contient des information essentiel a la décision comme l'angle entre deux segment de route, la vitesse maximal conseillé pour effectuer le virage en toute sécurité ou encore le gap a partir duquel il faudra entamer un virage.	
SourceSim	La Source est un objet encore en développement. C'est la source qui propulse les véhicules sur le réseau routier en fonction d'un certains nombre de (fréquence, vitesse, intervalle, etc..). Elle correspond au point d'entrée de nos agents dans l'environnement.	

Classe	Rôle	Image
Le puits	Le puit est un objet qui n'existe par réellement dans la simulation, seulement le concept est indispensable. lorsqu'un véhicule croise un puits il disparaît du réseau routier.	

Nous venons de voir les objets qui compose le réseau routier. Nous reviendrons plus tard dans la partie XXX sur la procédure pour créer un réseau routier.

3.3.3 InitialScreen et Time

La classe lancé par le Main est la classe InitialScreen. Celle-ci permet de faire de nombreuses choses couplé avec la classe TimerSDL développer initialement par PERRUCHON Romain pour le site developpez.com et modifier par mes soins pour qu'elle convienne au programme.

InitialScreen est la grande classe coordinatrice de ce projet. Elle permet la création du réseau routier depuis un ensemble de fichier, elle initie et optimise toute la partie graphique, elle permet d'agrandir ou de rétrécir l'image du réseau routier ainsi que de déplacer notre point de vu du réseau.

Elle régule aussi le nombre de FPS¹⁰ et par la même occasion régule toute la simulation à 40 cycles par secondes. Nous Reviendrons plus en détails sur ce sujet dans la partie "Gestion du temps".

Dernière fonctionnalité mais pas des moindres, elle ouvre une session de log et y consigne un certain nombre d'information importante.

Passons maintenant à la description de l'environnement.

3.3.4 Environnement et VueEnv.

La construction de cette classe fut sujet à d'interminable controverse. Si l'on suit les directives de Weyns [14] ainsi que celles de Kentenci [11], mettre l'environnement en tant que classe mère puis l'envoyer à chaque agent en tant que référence est une erreur de débutant.

Ces articles furent les bienvenus car c'est effectivement comme cela que j'aurais instinctivement construit le modèle du programme. Au lieu de cela, et avec l'aide de Julien Saunier, nous avons opté pour une solution plus orienté "MAS" c'est à dire avoir une vue de l'environnement.

L'environnement est un gigantesque conteneur qui rassemble et organise un grand nombre d'objet. Pour commencer tout les objets du réseau routier mais aussi les Tags, les véhicules et les vues.

10. Frames Par Seconde

La vue correspond à un sous ensemble de l'environnement et est propre à chaque Agent. Une vue est un ensemble de pointeur mis à jours à chaque cycle qui pointe sur les différents objets du réseau routier dont l'agent a besoin pour prendre ses décisions.

Ceci est à mon sens plus fidèle à la réalité dans le sens où nos perceptions ("nous" en tant qu'agent intelligent) ne contiennent absolument pas l'ensemble de notre environnement. Ce que nous percevons est majoritairement une ensemble d'image, de son et de sensation (la texture, la proprioception, etc..) localisé. Ces perceptions sont une Vue de notre environnement et elles sont strictement personnelles.

Voyons maintenant le l'acteur de notre simulation : le véhicule.

3.3.5 Vehicle et Body.

Cette classe (Vehicle) est de loin la plus complexe de ce projet et fut la plus difficile à développer de tout les projets auquel j'ai eu l'opportunité de participer.

Cette complexité est due à plusieurs choses. Tout d'abord le véhicule est l'Agent, c'est à dire qu'il doit être autonome. Ses décisions et l'application de ces décisions, les actions donc, doivent être répercute dans l'environnement à chaque cycle, et cela de manière crédible.

Très tôt dans le projet il a été fait comme choix d'associer le design de l'application (ce que l'on voit) au modèle de l'application (l'architecture). il en résulte un phénomène intéressant c'est qu'à chaque "faux pas" de l'agent, les répercussion sur le monde et sur ce que l'on en voit sont immédiates. Une mauvaise décision, un mauvais paramètre et ce sont tout les véhicules qui sortent de la route, l'erreur est saisissante ! Elle n'a donc pas sa place dans ce modèle.

La classe vehicle, qui correspond au brain de l'agent, est aussi composé de la classe Body qui représente le corps de l'agent.

Ce corps est représenté par un rectangle rouge en mouvement dans le réseau routier. Elle a aussi un point jaune en plein milieu. Ce point correspond à l'endroit exact du centre du véhicule et où se situe l'agent.



3.4 Gestion du temps.

Dans une simulation le temps est quelque chose primordial à prendre en compte. Il y aurait énormément de choses à dire alors nous allons nous concentrer sur les deux notions clés de la gestion du temps dans

Yats : le FrameRate et notion de cycle.

Pour ceux habitués à jouer aux jeux vidéos, cette notions coule de source. Traduit grossièrement le FrameRate correspond au "ratio d'image" (par seconde). Pour commencer à induire une réel impression de fluidité dans l'image il ne doit pas descendre en dessous de 24 fps (frame per second).

Nous avons fixé nôtre ratio à 40 fps, ce qui est bien au dessus du seuil d'impression de fluidité, mais qui reste en dessous du ratio exigé pour la plupart des jeux vidéo (fixé approximativement à 60 fps).

Nous fixons une limite au FrameRate car nous avons besoin qu'il soit constant afin de pouvoir définir un cycle.

Du fais que SDL est une librairie conçu en C, pour le C et que nous l'avons encapsuler dans du C++ afin d'être plus près du paradigme du SMA, celui-ci à besoin de mettre à jours l'image à chaque cycle de décision. bien évidemment les algorithmes de de distance et de vitesse sont calculés en fonction de ce ratio de cycle par seconde de sorte que nous puissions dire qu'une seconde de temps reel est égale à une seconde de temps simulé avec un pas de temps entre chaque cycle de 25 Millisecondes, ce qui nous donne 40 cylce par secondes.

Schématiquement il se passe a peut près cela :

Algorithm 1 Simplification d'un Cycle Yats

Require: Que ce soit le bon moment pour lancer un cycle (40 fois par seconde)

for each Vehicle dans l'Environnement **do**

 décider quoi faire.

 Effectuer la décision.

 Mettre à jours la position du véhicule a l'écran.

end for

Comme nous l'avons dis précédemment le modèle n'est pas séparer du Design. C'est pourquoi nous avons besoins de fixer le nombre de cycle par seconde.

Intéressons nous maintenant au fonctionnement interne de l'agent.

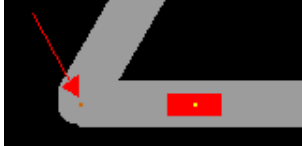
4 Perspectives algorithmiques.

Grâce aux articles scientifiques et à notre créativité nous avons réussi à produire des agents ayant un comportement réaliste. Nous allons dans cette partie détailler comment fonctionnent les algorithmes de notre modèle.

Par cette démarche nous allons aussi exposer un peut plus l'interaction qu'entretiennent les objets entre eux.

4.1 Description de la prise de décision du virage du véhicule.

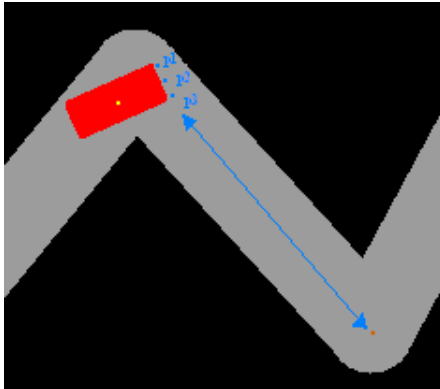
La première chose à savoir est qu'un véhicule pointe sa direction (nous appelons cela son point focal) sur un nœud. Lorsqu'il pointe dessus il fait apparaître un point de couleur marron dessus afin que l'observateur puisse anticiper le comportement du véhicule. L'image ci-dessous nous montre cela explicitement.



Afin de rompre avec la vision d'ARCHISIM où le véhicule est tracté sur des rails, nous avons décidé d'offrir un contrôle de l'agent sur la direction qu'il pouvait prendre.

Le véhicule possède donc la capacité de tourner à droite ou à gauche jusqu'à un angle par cycle de 0 à 3° avec un incrément de 0.60° par cycle. étant donné qu'il y a 40 cycles par seconde cela fait une capacité à tourner qui est ni trop grande ni trop petite.

Le plus intéressant étant de savoir quand est-ce que notre véhicule décide de tourner ? Nous allons imaginer notre explication.



A son état actuel (e_0) le véhicule se projette en trois états (e_1, e_2, e_3) sur 3 points correspondants ici à p_1 , p_2 et p_3 . Ces 3 points, ou ces 3 états correspondent à 3 actions que peut effectuer l'agent :

1. p_1 : tourner à gauche.
2. p_2 : ne pas tourner.
3. p_3 : tourner à droite.

Une fois ces trois points connus, l'agent calcule la distance qui le sépare de son point de mire en fonction des points p_1 , p_2 et p_3 . Le calcul de distance

qui est alors effectuer est la distance cartésienne comme nous avons vu dans la partie *algèbre linéaire*.

L'agent choisira alors l'action qui le séparera la moins possible de sont point de mire. Ici p3, l'agent tournera donc à droite en incrémentant son angle de braquage de 0.60° si celui ci n'est pas déjà à sont maximum de 3° .

4.2 Description de la visibilité du véhicule.

Il y eu une phase du projet ou le véhicule pouvait voir uniquement jusqu'au prochain nœud sur le réseau routier. Par voir nous entendons les objet auquel ont accès les véhicules depuis leur vue. les Les nœuds étant composé de RoadLink, liant les segment de route entre eux, mais dont le but principale est d'offrir aux véhicules un certain nombre d'information structurel sur le réseau routier de manière localisé.

Cette vision du modèle provoquait une erreur surprenante. Lorsque le segment de route suivant celui sur lequel était le véhicule était très court, le véhicule n'avait pas le temps d'adapter sont allure pour franchir le second nœud et donc loupait le troisième point focal (le troisième nœud a franchir). Le véhicule quittait donc le réseau routier pour orbiter à l'infini autour de ce troisième nœud loupé.

Il est bien dommage que nous ne puissions pas avoir de vidéo sur un document papier car cela aurait grandement facilité l'énonciation de ce problème.

Nous devons donc trouvé un moyen pour que le véhicule puisse voir un "certain nombre" de nœud a l'avance. Cela ne devait être ni trop, ni trop peut, ni trop long a calculer, ni trop lourd en mémoire.

Nous allons maintenant détailler ce qu'il se passe à chaque cycle.

Algorithm 2 Visibility

Require: destination initiale.

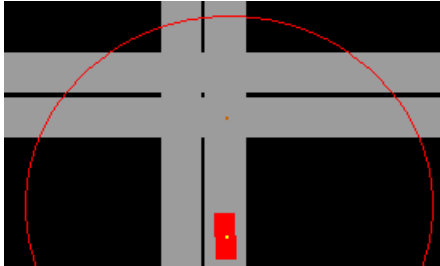
```

dist  $\leftarrow$  0
distStop  $\leftarrow$  calcul de la distance nécessaire pour que le véhicule arrive a
0 km/h
dist += Distance_Segment_Actuel - Distance_Deja_Effectué_Sur_Segment
while dist < distStop do
    Choisir un segment parmi ceux possibles
    ajouter ce segment dans la liste du chemin qui sera parcouru.
    dist += Longueur_du_Segment_Ajouter_Dans_La_Liste
end while
```

Bien sur cette algorithme est terriblement simplifier. la réalité de l'implémentation implique une grande quantité de paramètres a ajuster finement. Par exemple, d'une itération de l'algorithme de visibilité a l'autre pour le même véhicule

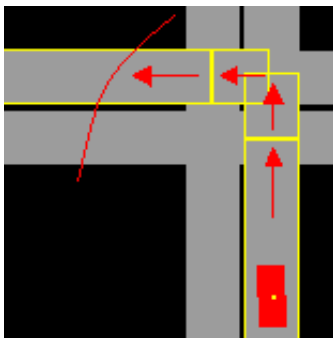
il n'y a quasiment pas de changement dans la liste car celle ci est persistante dans le temps. Nous ne faisons donc pas repartir l'attribut *dist* de 0.

L'image ci-dessous illustre assez bien ce qui se passe en interne.



Le cercle rouge représente la distance nécessaire au véhicule pour être à l'arrêt. Cette distance varie évidemment en fonction de la vitesse actuelle du véhicule.

Des indices sur cette image nous montre que le véhicule ne tournera pas immédiatement à gauche car le cercle serait plus petit, il n'ira pas non plus tout droit car le cercle serait beaucoup plus grand.



Il tournera donc effectivement à gauche et par la même occasion aura créer un chemin contenant 4 nœud. Sur ces 4 nœud il trouvera celui qui le fait ralentir le plus près possible de sa position actuel. sur l'image ci-dessus c'est le second nœud avec un angle entre les deux segment de route de 90° .

4.3 Description du mouvement d'un véhicule, accélération ou freinage.

Nous avons vu précédemment comment le véhicule tournait et comment il choisissait son chemin. Nous allons maintenant voir comment le véhicule se meut dans le réseau routier.

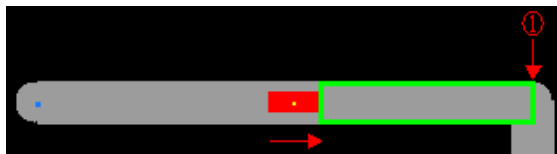
Cette fonctionnalité a en réalité été développée avant les deux autres, mais il est plus facile de la comprendre après la description des deux précédents algorithmes.

L'algorithme d'accélération (ou de freinage) utilise a plusieurs reprise la formule d'accélération décrite dans la partie "Mathématiques élémentaire".



Le véhicule sur l'image ci-dessus est en début d'accélération. mettons qu'il soit à peu près a 15 km/h. sa distance de freinage correspond au rectangle vert et au point annoter 1 le véhicule sera à 0 km/h.

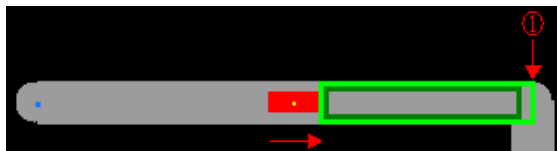
Le véhicule continue sur son segment de route et se trouve dans la situation suivante.



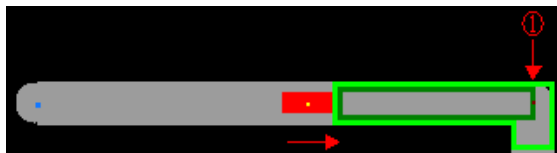
n'ayant pas rencontré de nœud entre l'image ci-dessus et l'image précédente, le véhicule a continué son accélération. Sa distance pour être a 0 km/h s'en trouve agrandi.

Le véhicule va maintenant pouvoir interagir avec un RoadLink et choisir son chemin. Ici il n'y a qu'un seul chemin, il tournera a droite dans la rue ayant un angle de 90° .

Ce RoadLink va transmettre au véhicule une vitesse conseillé. symbolisé ci-dessous par le rectangle vert foncé.



Le véhicule continuera d'accéléré jusqu'au ce que sa limite pour arriver au noeud soit atteinte.



une fois qu'elle sera atteinte il commencera a freiner.

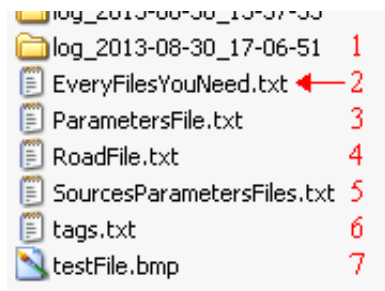
5 Fonctionnalités de confort.

5.1 Exemple de création d'un réseau routier.

La création du réseau routier fut l'une des première fonctionnalité développé sur Yats. Il est toujours très rébarbatif de coder une carte aussi cette démarche devait être simplifié. Nous avons opter pour l'organisation suivante.

Nous allons lister ici l'ensemble des fichier dont a besoin le programme pour fonctionner pour un réseau routier donné.

Tout d'abord le programme vous demande, via la console, le fichier qui liste tout ce dont il a besoin pour travailler.



Pour ce réseau routier la c'est le fichier "EveryFilesYouNeed.txt" dans l'ordre marqué sur l'image ci-dessus ce repertoire comprend :

1. le dossier de log que nous verrons dans la partie suivante.
2. le fichier donnant le chemin des autres fichiers dont a besoin l'application pour fonctionner.
3. le fichier de paramétrage de l'application
4. le fichier contenant le réseau routier
5. le fichier de parametrage des sources
6. le fichier pour les tags
7. le fichier pour l'optimisation de l'affichage de l'écran.

Attardons nous sur la composition du fichier ROADFile.txt en étudiant un exemple avec l'image ci-dessous.

```
Zone C
R080 ROAD (35;105) (35;85)
R055 ROAD (R054) (300;150)
R060 ROAD (R059) (316;180)
R061 ROAD (R060) (R027)
```

L'algorithme interprète ligne par ligne ce fichier. Une ligne est équivalent à un segment de route. Toute ligne qui ne respecte pas la mise en forme n'est pas prise en compte ce qu'il nous permet de mettre d'indispensables commentaires (par exemple : ici "Zone C" est un commentaire).

Le premier champs correspond au nom du segment de route. Celui-ci doit être unique dans le réseau routier. Le second correspond au type d'objet, ici "ROAD" pour un segment de route.

Les troisième et quatrième champs corresponde au points de départ (d'entrée) et d'arrivée (de sortie) du segment de route. ici il peut être sous deux formes. soit c'est une coordonnée $(x;y)$ séparé par ";", soit c'est le nom d'un autre segment routier présent dans le fichier. Si c'est le nom d'un autre segment de route il prendra en point d'entrée le point de sortie du segment de route sur lequel il point. Pour le point de sortie c'est l'inverse, il prendra le point d'entrée du segment sur lequel il pointe.

Dernier élément important : l'unité est le mètre. L'algorithme à travers lequel il passe se chargera de le convertir en pixel pour l'afficher à l'écran.

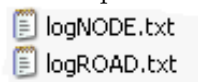
Ce système a grandement facilité le développement de réseau routier de test. Nous en verrons certains dans la partie Benchmark un peut plus loin, et par la même occasion les capacité de ce système.

5.2 Description du system de log du réseau routier.

Toute bonne application qui se respecte doit posséder un système de log afin de vérifier les évènements du système. Nous avons vu précédemment comment créer un réseau routier, nous allons maintenant voir les logs qu'il génère.

Lors du lancement de l'application un identifiant de session va être créé composé de : "logs"+"année"+"mois"+"jours"+"heure"+"minute"+"seconde". Cette identifiant est par essence unique. Les fichiers de logs seront donc stocker dans un dossier avec cette identifiant unique.

Comme nous l'avons vu dans la partie "Diagramme de classe" qu'il y avait un certains nombre d'objet autre que les segments de route qui composent le réseau routier. Ces objets et leurs informations correspondantes sont créé a partir des deux seuls points de chaque segment de route.



Le fichier "logROAD.txt" stocke les informations de chaque objet sous la forme suivante :


```

- ROAD : R003
    COORD (2000;285.714) // (642.857;285.714)
    ANGLE : 180
    LONGUEUR : 1357.14 ( pix )
    ENTRY NODE : Node3
    EXIT NODE : Node2

```

Pour chaque route nous avons :

- son nom
- ses coordonnées d'entrée et de sortie en pixel
- son angle en degré par rapport à l'axe des abscisses
- sa longueur en pixel
- son nœud d'entrée
- son nœud de sortie

toutes ces informations notés peuvent évidemment changer au cours de l'exécution de la simulation par exemple en changeant l'échelle de la carte ou en le déplaçant.

Le fichier "logNODE.txt" rentre plus en profondeur dans les objets composant le réseau routier.

```

Node : Node31( 2292.86 ; 900 )
  Entry : R027, R045,
  Exit : R035,
    LinkRoad : 0
      entry : R027
      exit : R035
      Angle : 71.5631
      gap : 15
      max speed : 0.817854
    LinkRoad : 1
      entry : R045
      exit : R035
      Angle : 18.4369
      gap : 15
      max speed : 3.01496

```

Ce fichier décrit en détail l'objet NodeSim, lui même composé d'une liste de RoadLink.

Il y a sa position (en pixel) dans le réseau routier, le nom de ses routes en entrée et en sortie, ainsi que les RoadLink qui le composent. Il y aura forcément au tant de RoadLink que $(nbr_{Entree} * nbr_{Sortie})$.

Les informations relative a chaque RoadLink sont aussi noté : le point d'entré, le point de sortie, l'angle entre les deux segments de route, la vitesse maximale conseillé et le gap.

5.3 Biometrie de l'application

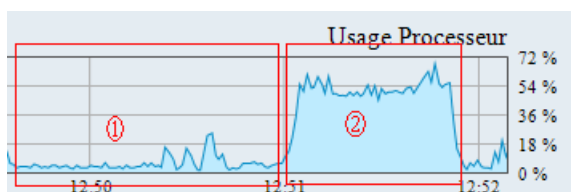
Comme tout programme en C++ qui se respecte, Yats a eu sont lot de fuite de mémoire à chaque cycle de développement. Nous avons donc mi en place un protocole de test couplé à la très bonne application System Explorer¹¹. Aucun développeur C++ ne peut ignorer ou travailler sans cette application qui permet de faire des diagnostique de la mémoire et des performance de notre ordinateur.

Yats Fu développer sur (ne sont présentes que les information pouvant influencer sur l'exécution de l'appplication) :

Processeur	Intel Core 2 Duo E8400 Cadencé à 3 GHz Nombre de coeurs : 2 physiques, 2 logiques
Carte Mère	Dell Inc. 0TP412
RAM	4 Go de mémoire totale de type DDR2 2 * Barrette Kingston de 2 Go
Carte Graphique	NVIDIA Quadro FX 1700 Fréquence du GPU : 460 MHz
Disque Dur	232.83 Go SATA II

Diagnostic emi par Ma-Config.com¹².

Revenons sur la biométrie de l'application, plusieurs paramètres peuvent être prix en compte afin de juger ce que fait l'application en terme de calcul brute.



Ce graphique a été réalisé avec le plus grand réseau routier construit pour Yats (approximativement 500m²).

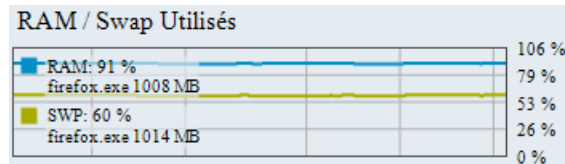
Nous remarquons qu'il y a une nette différence d'usage du processeur(uc) lors du lancement dans l'application. La zone 1 correspond à l'usage "normal" de l'ordinateur, avec tout ce que cela comprend de tache en fond. la

11. <http://fr.systemexplorer.net/>

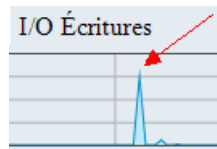
12. <http://www.ma-config.com/fr>

zone 2 correspond a une phase de lancement de Yats. Celle-ci consomme jusqu'à 72% de l'UC.

Cela nous indique deux chose. Déjà nous remarquons une certaine constance dans la quantité de ressource consommé, ensuite nous pouvons dire qu'il reste encore un peut de marge. Globalement 40% de l'uc reste disponible.



Le graphique ci-dessus indique la quantité de RAM utilisé. Elle reste constante durant l'exécution du programme ce qui nous indique une information capital : il n'y a pas de fuite de mémoire



Nous remarquons aussi un pique en ce qui concerne les écriture sur le disques. Cela correspond à l'inscription des logs lors du lancement de l'application.

5.4 Développement d'un benchmark pour les tests.

Comme nous l'avons vu dans la partie "Simulation et Développement itératif", tester notre modèle est une grande part du développement de notre application.

Ce Banc d'essais a pour but de faciliter les test de Yats. Il comprend un certains nombre de cartes pré-codé qui oriente l'essai vers l'un des aspect du développement.

- Vérification des Sources.

De nombreuses erreurs peuvent être commises sur les sources et notamment sur les angles de départ. Le bon coté est qu'elles sont identifiables rapidement et facilement si l'on a la méthodologie adéquat.

Rappel : la position des sources correspond aux petits points bleu sur l'image du réseau routier vide.

Deux réseau routier son déjà pré-coder pour cela.

A) Les erreurs d'angles Pures.

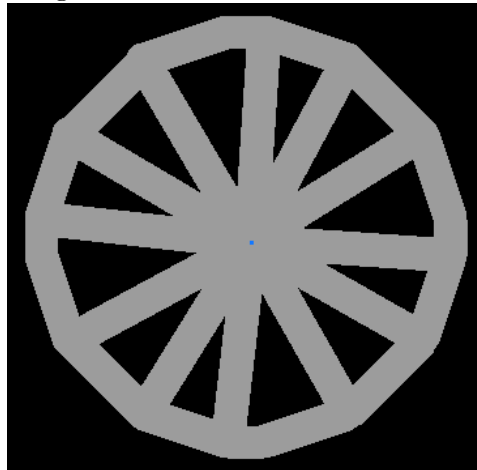
- ☐ "TestOnSourceSimClasse" : permet de visualiser si les véhicules sont correctement propulsés sur les points cardinaux (NORD,SUD,EST,OUEST). Ceux-ci correspondent aux angle critiques.
- ☐ Comportement type : les véhicules partent d'en haut a gauche et d'en bas a droite sur leurs deux routes correspondantes aux points cardinaux pures.
- ☐ image du réseau routier :



(échelle 1/2)

B) les erreurs d'angles mixtes.

- ☐ "HellSourceLoop" : permet de visualiser si les véhicules sont correctement propulser de large zone : (Nord/Est, Nord/Ouest, Sud/Est, Sud/Ouest), sans jamais propulser de véhicule sur les multiples de 90 °.
- ☐ Comportement type : partent simultanément dans tout les cotés à la fois en partant du centre. Ceci produit un effet esthétique assez intéressante.
- ☐ image du réseau routier :



(échelle 1/2)

- **ligne droite.**

- ☐ "SingleRoad" : permet de visualiser si les véhicules réussissent a atteindre correctement un puis depuis une source en 1 seul

segment de route. Ce cas est plus critique qu'il n'y paraît car les algorithmes du véhicule se base sur plusieurs segments de routes, le fait qu'il n'y en ai qu'un seul peut donc provoquer des erreurs.

- ☐ Comportement type : Le véhicule par de la source et disparaît une fois arrivé au puits sans provoquer d'erreur critique.
- ☐ image du réseau routier :

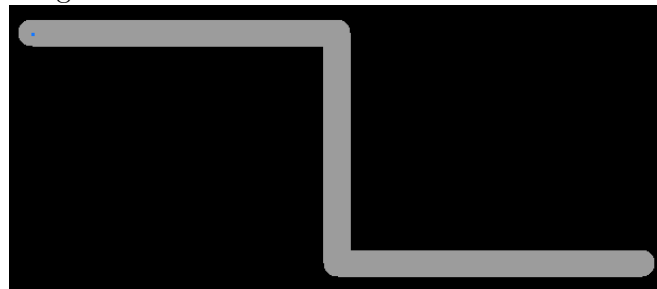


(échelle 4/10)

- Angles et Virages

A) Virage à Gauche/Droite Simple.

- ☐ "TwoTurnsingleroad" : Un réseau routier possédant une entrée, une sortie et seulement 2 virages simples, le premier à gauche, le second à droite, d'un angle de 90° .
- ☐ Comportement type : Le véhicule effectue correctement les virages successifs.
- ☐ image du réseau routier :



(échelle 4/10)

B) Angle Min, Angle Max.

- ☐ "WorkOnCorner" : Un réseau routier dont la succession de segment de route incrémente l'angle par rapport au segment de route précédent de 10° . Allant de 10° à 150° .
- ☐ Comportement type : Le véhicule parvient à tourner correctement quelque soit l'angle en gardant un comportement non choquant (sortie de route, virage en sur place, etc..).
- ☐ image du réseau routier :



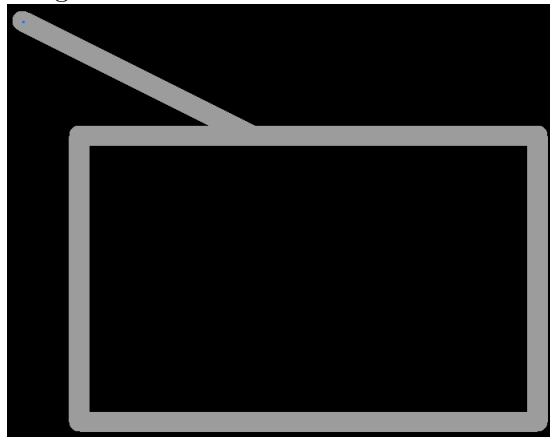
(échelle 3/10)

- Les boucles

Les boucle sont un espace de test clef a maitriser. Elle permettent de jauger d'une par les performance du système sur une grande plage de simulation avec les même véhicules, d'autres part a vérifier la cohérence des décision dudit véhicule au cours du temps.

A) Première boucle.

- ☐ "HellLoop" : Un réseau routier sans sortie avec 4 virages en angle droit à droite et une voie d'insertion.
- ☐ Comportement type : Le véhicule s'insère dans la boucle et prend successivement les virages.
- ☐ image du réseau routier :

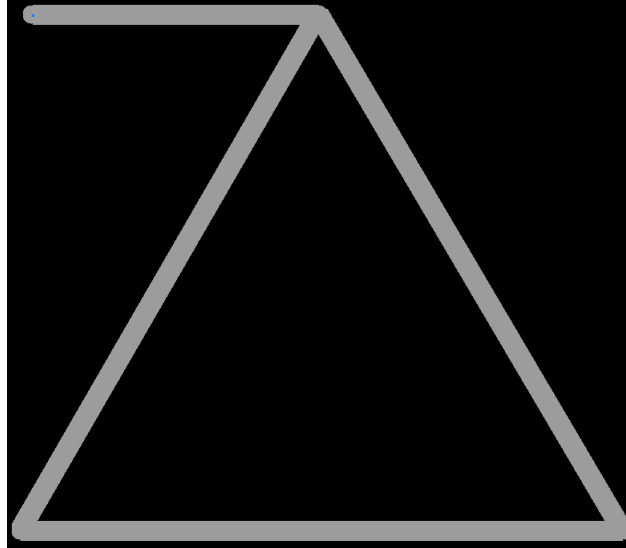


(échelle 3/10)

B) Seconde Boucle.

- ☐ "WorkOnCornerTriangle60" : Un réseau routier sans sortie avec 3 virages serré (120°) à droite et une voie d'insertion.
- ☐ Comportement type : Le véhicule s'insère dans la boucle et prend successivement les virages.

☐ image du réseau routier :



(échelle 3/10)

Les boucles seront un excellent moyen dans le futur de tester les algorithmes de suivi.

- **Les Carrefours.**

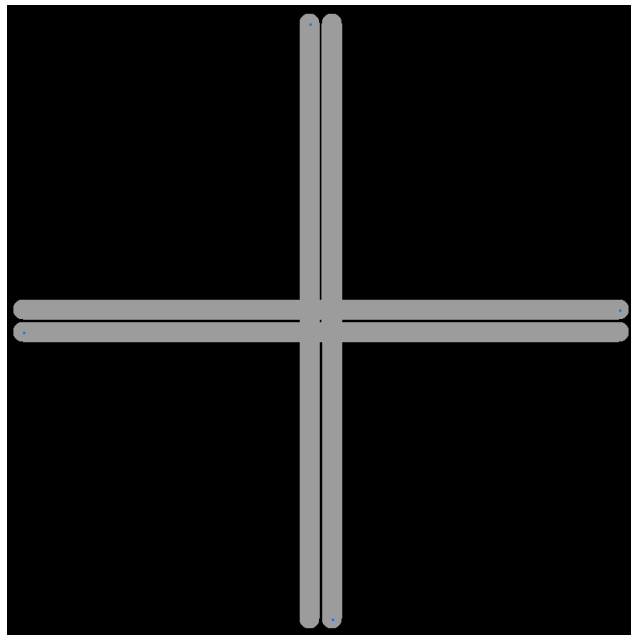
A) Les carrefours classiques.

☐ "classiqueCarrefour" : un réseau routier ayant 4 entrées et 4 sorties. Les routes se rejoignent au milieu en un carrefours classique.

Existe aussi en version pour un seul véhicule.

☐ Comportement type : Les véhicules doivent effectuer le franchissement du carrefour à la bonne vitesse sans louper leur prochaine destination.

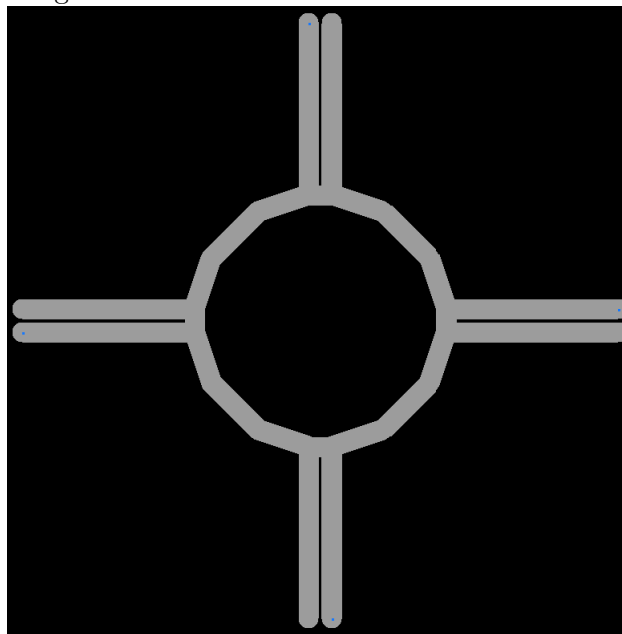
☐ image du réseau routier :



(échelle 3/10)

B) les ronds-points.

- ☐ "giratoireSingleRoad" : le réseau routier est un carrefour à sens giratoire possédant une seule route dans sa boucle, de 4 routes sources et 4 routes puits.
Existe aussi en version pour un seul véhicule.
- ☐ Comportement type : les véhicules franchissent le rond-point à la bonne vitesse sans louper leur prochaine destination.
- ☐ image du réseau routier :



(échelle 3/10)

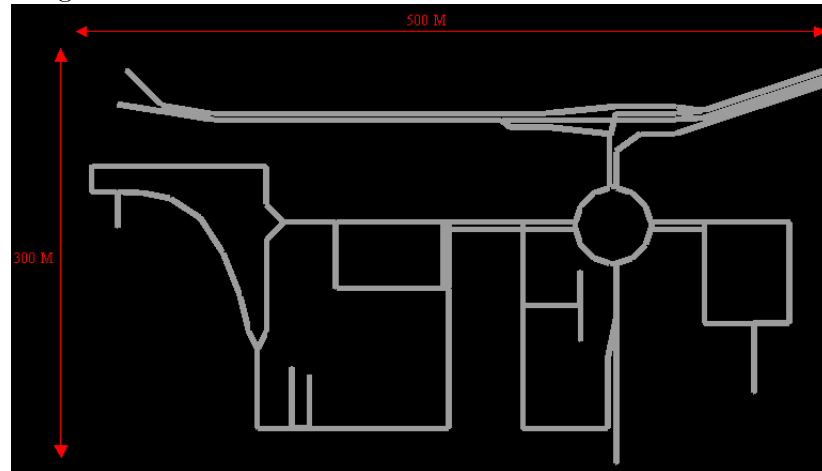
- **Grande carte.**

- ☐ "FirstBigMap" : un réseau routier de très grande taille afin de mieux prendre conscience des performance graphiques du simulateur.

Permet aussi de se rendre compte des possibilités de cartographie d'une zone avec une carte complexe.

- ☐ Comportement type : Les zooms et dé-zoom s'effectuent de façon fluide, ainsi que les déplacement de la cartes. Quelque soit la l'échelle du réseau routier les véhicules ne change pas de comportement.

- ☐ image du réseau routier :



un réseau routier de très grande taille afin de mieux prendre conscience des performance graphiques de la simulation

Bien évidemment le développement de nouvelles fonctionnalités entrainera obligatoirement le développement de nouveaux espace de test.

6 Perspectives et futurs développement.

Yats n'est en théorie pas destiné à rester dans cet état. De nombreuses fonctionnalités devraient être développées afin de rendre plus conforme à la réalité. Mr Saunier ainsi qu'un autre chercheur de l'ifsttar Sio XXX son sensé continuer son développement. Nous allons faire ici une petite liste afin de mieux visualiser le futur de Yats et par la même occasion de saisir dans quel perspective il a été construit.

6.1 Liste des fonctionnalités futurs

- **paramétrage fin des source.**

L'un des but de Yats pourrait être de vérifier si certaines routes, par exemple, ne deviennent pas de goulot d'étranglement, ou encore vérifier si un flux routier s'écoule bien. Pour cela il faut faire en sorte d'avoir des sources dont nous pouvons contrôler le début selon un intervalle précis. Nous devrions donc avoir un fichier "source.txt" qui récolterais tout ces paramètres.

- **Algorithme de suivi et détection des collisions.**

Celui-ci aurait été le prochain algorithme complexe implémenté dans Yats. Il aurait impliqué plusieurs chose :

- * la prise de conscience par l'agent de la présence d'autres véhicules, donc d'autres agents.
- * l'implémentation de la notion de véhicule contraignant. Si le véhicule qui précède l'agent roule moins vite, celui-ci sera contraint d'adapter son allure.
- * l'implémentation d'un système de collision pour que le développeur repère à quel moment les véhicules sont entrés en collision afin de faciliter l'investigation de la cause de cette collision.

- **Algorithme de croisement.**

Les véhicules doivent pouvoir franchir un croisement sans provoquer d'accident. Cela implique l'implémentation de la règle de la priorité, quelle soit à droite (par défaut) ou à gauche.

Cela aurait aussi été l'occasion d'implémenter le système de tags afin que le véhicule commence à prendre en compte la signalisation (les panneaux ainsi que le marquage au sol).

- **Algorithme de changement de voie.**

Comme nous l'avons vu précédemment les segments de routes sont reliés entre eux par des nœuds. Cette algorithme aurait été potentiellement problématique car les véhicule auraient changé de voie sans

pour autant passé par un nœud. Il aurait sûrement fallu rajouter au moins une variable à l'objet Road.

- **Algorithme de dépassement.**

Peut être l'un des algorithmes de plus haut niveau étant donné qu'il implique une décision stratégique. Pour dépasser un véhicule il faut : que le véhicule devant soit contraignant (qu'il aille moins vite que la vitesse actuelle du véhicule qui prend la décision de dépasser), qu'il y ait au moins deux voies et qu'il y ait un passage exploitable.

Cela fait beaucoup de paramètres à prendre en compte. Arrivé à ce niveau les algorithmes précédemment énoncés se doivent d'être irréprochables, mais plus encore ils se doivent d'avoir un environnement de test prévu spécialement afin de vérifier que les modifications du source ne provoquent pas de sévères régressions.

- **Élargissement des fonctionnalités temporels.**

Nous avons vu dans la partie "Gestion du temps" comment était géré le temps dans Yats. Pourtant il serait possible d'affiner notre conception de cette fonctionnalité.

* **Dilatation**

Que se passerait-il si le réseau routier est surchargé et que les calculs sont trop lourds pour permettre 40 cycles par seconde ? Nous aurions l'impression que les véhicules de notre simulation iraient anormalement lentement.

Nous pouvons considérer ce temps comme dilater. Par exemple pour un réseau routier surchargé au point de faire tomber le nombre de cycles à 20 par seconde nous aurons un ratio d' $\frac{1}{2}$. Il faudra donc 2 secondes réelles pour effectuer une seconde de temps simulé.

Nous avons donc un temps dilaté qui n'influe en rien du tout sur les décisions ou processus de la simulation étant donné que pour celle-ci une seconde équivaut à 40 cycles et cela quoi qu'il arrive.

* **Contraction**

C'est l'exact opposé de la dilatation.

La puissance de calcul de la machine permet d'accélérer le temps d'exécution de la simulation. Si la fréquence est par exemple de 120 cycles par seconde nous aurons donc 3 secondes de temps simulé par seconde réelle.

Les véhicules n'iront pas plus vite pour autant, seul leur parcours sera visionné plus rapidement.

Au niveau de l'interface ces deux fonctionnalités pourraient être accompagnées d'une petite jauge qui indique le niveau de dilatation ou de contraction.

6.2 Comparaison des logs de véhicule avec un véhicule réel.

L'un des buts du simulateur est de voir le modèle théorique validé par des données réelles. Il existe une base libre contenant ces données : La base NGSIM¹³.

NGSIM (Next Generation Simulation) est un programme de l'U.S. Department of Transportation. Il a pour but de favoriser le développement des algorithmes de comportement de conduite, collecter des données afin de valider les modèles émis et ainsi perfectionner les outils déjà développés.

La base NGSIM contient des données de trajectoire et de vitesse sur des lieux clés (intersection, virage, voie d'insertion, etc.). Ces données ont été récoltées via des caméras et se présentent sous la forme de méta-données.

L'analyse de ces trajectoires avec les logs des véhicules peut être un bon moyen de valider ou réfuter notre modèle. Nous pourrions aussi en extraire une échelle afin de savoir si nous sommes plus ou moins proches de la réalité.

D'autres méthodes peuvent aussi être envisagées. Par exemple effectuer un test de Turing. C'est à dire dans un réseau routier des véhicules réels et des traces de véhicules réels puis demander à un ensemble de personnes de les différencier. Si les cobayes n'y parviennent pas nous pourrions dire que nos véhicules simulés sont assez proches du modèle réel en ce qui concerne la perception que les humains en ont.

Cette liste n'est pas exhaustive et bien d'autres méthodes de test peuvent être envisagées.

13. <http://ops.fhwa.dot.gov/trafficanalysisitools/ngsim.htm>

7 Conclusion et Perspectives

Dans l'ordre, nous venons de voir l'état de l'art sur la construction de SMA et de Simulateur adapté à la problématique de la conduite et de la simulation de trafic. De nombreux challenges ont été relevés tant sur le plan théorique, ou la lecture d'article et discussion avec les chercheurs de l'IFSTTAR ont été primordial, que sur le plan pratique, le développement d'un simulateur n'étant pas le plus aisé des logiciels à implémenter.

Sur ce type de projet et sur ce type de stage il y a toujours un décalage qui s'effectue.

Concernant le projet, le passage du modèle expert (scientifique) à la modélisation formelle (algorithmique) provoque forcément un léger écart en terme de vision initiale. Cet écart s'agrandit lorsque l'on passe de la modélisation formelle à l'implémentation. Ici un effort particulier a été fait pour prévoir le tout en même temps : modèle expert, formel et implémentation.

Le sujet initial du stage a lui aussi évolué en fonction des discussions et préférences individuelles, Les miennes étant clairement orienté ingénierie et celle du M Saunier focalisés sur la recherche. L'interaction des deux aura donné Yats, un simulateur conçu avec un maximum des mes soins supervisé un maximum par la perspective d'utilisation de M Saunier.

Pour ma part je suis très satisfait du résultat même si, comme d'habitude, j'aurais voulu en faire plus et plus longtemps. Il semblerais que je ne soit pas le seul à être enthousiaste étant donné que le projet devrait être poursuivit par d'autres chercheurs de l'IFSTTAR en plus de mon encadrant.

C'est donc très satisfait de ces six derniers mois que je quitte l'IFSTTAR ou j'aurais appris dans une plus large mesure ce que signifie être chercheur et comprendre la rigueur intellectuelle que cela implique. Cela m'aura aussi permis de mieux comprendre ce que je souhaite concrètement faire après l'obtention de mon diplôme de fin d'étude, Je me sent en effet plus à l'aise dans la création et la technique que dans la recherche fondamentale.

Références

- [1] M. Bratman. *Intention, Plans, and Practical Reason*. Center for the Study of Language and Information - Lecture Notes Series. C S L I Publications/Center for the Study of Language & Information, 1987.
- [2] Michael E. Bratman, David J. Israel, and Martha E. Pollack. Plans and resource-bounded practical reasoning. Technical Report 425, AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, Sep 1988. Revised version.
- [3] Marco Dorigo, Eric Bonabeau, and Guy Theraulaz. Ant algorithms and stigmergy. *Future Gener. Comput. Syst.*, 16(9) :851–871, June 2000.
- [4] Chaib B. Draa, I. Jarras, and B. Moulin. Sytemes multi-agents : principes generaux et application. In Jean P. Briot and Yves Demazeau, editors, *Principes et architecture des systèmes multi-agents*, chapter 1. Hermès-Lavoisier, 2001.
- [5] Paul Fishwick. Computer simulation : Growth through extension. In *Society for Computer Simulation*, pages 3–20, 1994.
- [6] N. Jennings, N.R. Jennings, and M.J. Wooldridge. *Agent Technology : Foundations, Applications, and Markets*. Springer, 1998.
- [7] Donald A. Norman. Affordance, conventions, and design. *interactions*, 6(3) :38–43, May 1999.
- [8] Alexandre Pitti and Rolf Pfeifer. *La révolution de l'intelligence du corps*. Manuella Editions, November 2012.
- [9] Eric Platon, Nicolas Sabouret, and Shinichi Honiden. Tag interactions in multi-agent systems : Environment support. 2005.
- [10] Robert E. Shannon. Simulation modeling and methodology. *SIGSIM Simul. Dig.*, 8(3) :33–38, April 1977.
- [11] Jean Michel Auberlet Utku Gorkem Ketenci, Roland Bremond and Emmanuelle Grislin. Bounded active perception. *EUMAS*, 10, 2010.
- [12] Danny Weyns and Tom Holvoet. A reference architecture for situated multiagent systems. In Danny Weyns, H. Van Dyke Parunak, and Fabien Michel, editors, *E4MAS*, volume 4389 of *Lecture Notes in Computer Science*, pages 1–40. Springer, 2006.
- [13] Danny Weyns, Andrea Omicini, and James J. Odell. Environment as a first-class abstraction in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 14(1) :5–30, February 2007. Special Issue on Environments for Multi-agent Systems.

- [14] Danny Weyns, H. Van, H. Van Dyke Parunak, Fabien Michel, Tom Holvoet, and Jacques Ferber. Environment for for multiagent systems state-of-the-art and research challenges. pages 1–47. Springer, 2005.

Annexe 1 - Problème notable de développement.

Au cours du développement de Yats j'ai rencontré des erreurs "difficiles". celle-ci sont difficiles en raison du caractère vicieux du compilateur à ne pas nous aider à en diagnostiquer correctement sa provenance.

j'ai délibérément choisi des les inclure dans ce rapport et je suis persuadé que tout examinateur qui aura au moins une fois été confronter à ces erreurs attestera de la difficulté de les résoudre lorsque l'on a strictement aucune idée d'où elles peuvent venir. Ces cas sont typique et ne peuvent être éviter que par une grande pratique du langage, ce que je m'évertue à acquérir chaque jour.

7.1 Float == à un EPSILON près.

7.2 Rupture de cycle dans la déclaration des headers.

Lorsque le compilateur vous dit que vous n'avez pas déclaré le type de l'objet que vous manipulez (alors que vous l'avez effectivement fait) et qu'une foule d'erreur semblent sortir du fond des ages du C c'est que vous avez surement un cycle dans la déclaration de vos Headers.

Merci à Lancelot de m'avoir aider à comprendre en détail mécanismes du pré-processeur.

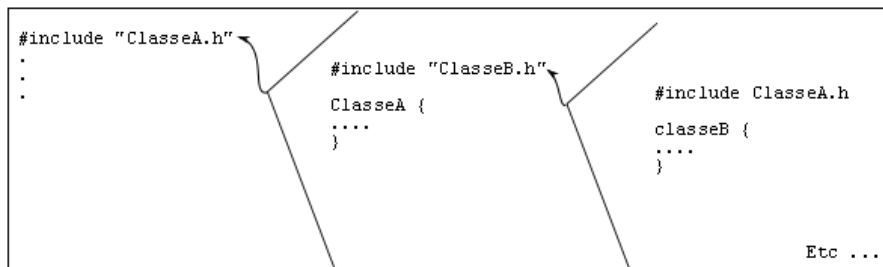
- *Comment reproduire cette erreur ?*

Deux classes suffisent pour reproduire cette erreur. Par exemple prenons une classe A contenant un objet de classe B ainsi qu'une classe B contenant un objet de classe A.

Lorsque le main va appeler notre classe A (qui contient un `#include "ClasseB.h"`) le pré-rocasseur va aussi appeler la classe B (qui contient elle même un `#include "ClasseA.h"`). Seulement la classe A ayant déjà été appeler va être appeler une seconde fois ce qui provoque une erreur de cycle dans la déclaration des headers.

- *Que se passe-t-il derrière ?*

Rien de mieux qu'un schéma pour illustrer cette situation.

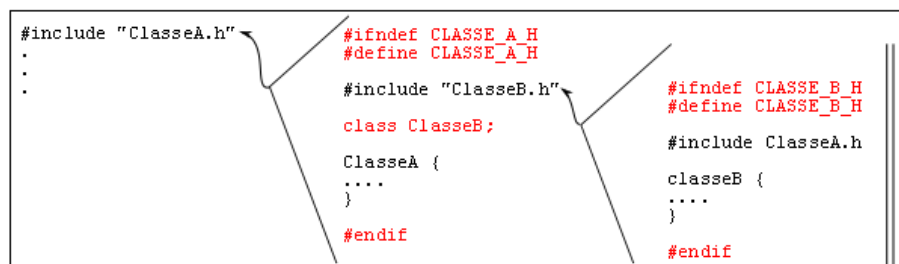


Le pré-processeur va vouloir charger les fichier ".h" en mémoire. Le problèmes lorsqu'il y a un cycle dans l'appel des headers c'est qu'il va appeler indéfiniment les fichiers de chaque classe. De plus nous nous retrouvons avec une inclusion multiple dans une même unité de compilation, ce qui est évidemment très mauvais.

Cette architecture directement hérité du C va nous forcé a utiliser les macros.

- *Comment fixer cette erreur ?*

Reprenons les schéma précédent en y ajoutant les bonne macros.



la macro `#ifndef` ("if not define") suivi d'un identificateur unique va nous permettre de créer une zone de déclaration protégée contre les inclusions multiples.

Évidement, immédiatement après nous devons créer l'identificateur en question auquel se référer. Nous avons donc la structure suivante :

```

1  #ifndef  IDENTIFICATEUR
2  #define  IDENTIFICATEUR
3
4  /* zone proteger contre les inclusions multiples */
5
6  #endif  /* guard */

```

- note sur la dépréciation ("deprecated")

Même si l'orthographe des macros présenté précédemment est très largement utilisé, nous sommes sensé écrire :

```

1  #if !defined(MA_VARIABLE)

```

Pour :

```

1  #ifndef MA_VARIABLE.

```

Annexe 2 - Logiciels utilisé pour construire ce document.

- L^AT_EX pour ce pdf avec T_EXMaker.
- InkScape pour les schemas. Paint pour la gomme.
- StarUml pour le schéma de classe.