

2012

Li260 - Rapport

COQUART KEVIN



Kevin Coquart

Li 260 – G1

11/06/2012

SOMMAIRE

Introduction	2
Les bases	2
Circuit / Voiture	2
Le MVC (Model View Controller).....	2
Les différentes stratégies.....	3
Les stratégies basé sur un radar	3
Le radar classique.....	3
Le radar dijkstra.....	4
La stratégie sélection.....	6
La stratégie arrivées.....	6
La stratégie point à point.....	7
La stratégie décorateur.....	7
Les modules	8
Les obstacles.....	8
Les zones.....	8
Conclusion sur la partie tactique	9
L'interface graphique	10
La fenetre d'accueil	10
Les Menus.....	11
1 – Fichier	11
2 – Editer	12
3 – Circuits.....	16
4 – Chargement	16
5 - Sauvegarde	16
6 – Application.....	17
7 – Observateur	17
8 – Vitesse	20
Conclusion sur la partie graphique	20
Les package	20
Résultats.....	21
Conclusion.....	22

INTRODUCTION

J'ai voulu suivre cette UE pour me perfectionner en JAVA, j'ai fait un peu de JAVA à mon premiers semestre (li230) et cela m'avait plus, de plus c'est un langage très prisé des professionnels.

La plupart du temps dans notre scolarité, nous étudions des « cas d'école » et réalisons des programmes sans but exacte. Réaliser un projet sur un cas concret et dans un domaine intéressant m'a tout de suite enthousiasmé.

L'intitulé de l'UE est « simulateur de course de voiture », le but était donc de faire faire à une voiture un tour de circuit.

Pour réussir cela nous devions créer les modèles (circuit et voiture), développer des algorithmes pour parcourir les différents circuits ainsi que développer une interface graphique pour rendre notre simulation plus interactif.

LES BASES

CIRCUIT / VOITURE

Nous avons commencé dans ce projet à créer les parties « physique » d'une course de voiture, c'est-à-dire le circuit ainsi que la voiture.

Le principe de création est le même pour les 2 :

- ─  Une interface
- ─  Une implémentation
- ─  Et une factory

Ce système découvert en cours est très utile, en plus il rend le code simple et propre.

LE MVC (MODEL VIEW CONTROLLER)

Pour avoir une représentation de notre simulation, il a fallu mettre en place un système d'écoute entre les différents objets.

Le modèle demandé était le système MVC, bien que compliqué à comprendre au départ, il permet de gérer des ensembles complexes, ce qui au fil du projet est devenu bien pratique.

Le fonctionnement est le suivant :

- 2 types de classes
 - o Les listener, il attend de recevoir un message de la part d'une classe émettrice (sender) pour se mettre à jour.
 - o Les sender, envoie à certains moments un ordre de mise à jour à la classe qu'il écoute.
- La simulation est une classe émettrice, elle émet après chaque coup (PlayOneShot).
- On ajoute à sa liste d'écouteur un contrôleur, ce dernier lors de la réception d'un ordre mettra à jour les observer (nous ferons le tour des observer plus loin).

Ce fonctionnement a permis dans les débuts de gérer un affichage statique et par la suite de passer à un affichage dynamique sans aucun soucis.

LES DIFFERENTES STRATEGIES

LES STRATEGIES BASE SUR UN RADAR

La 1^{er} approche que l'on a eu à était de projeter des rayons dans différentes direction, puis de choisir le rayon le plus intéressant pour envoyer la voiture dans cette direction.

Pour gérer tout cela il a fallu une stratégie radar, elle a besoin de connaitre un tableau d'angle¹, le tableau de commande² associé ainsi qu'un radar³.

Ensuite pour chaque angle, elle applique le radar qui va remplir un tableau de score. Le score le plus élevé correspond au faisceau le plus intéressant, on décide donc d'appliquer la commande associé.

LE RADAR CLASSIQUE

C'est notre tout premier radar, il se contente de projeter chaque rayons jusqu'à atteindre une case du circuit ou la voiture ne peut plus rouler. Ensuite il renvoie le nombre de pas qu'il a fait entre la voiture et cette case.

Le faisceau qui a était le plus loin renvoie donc le score le plus élevé et est celui choisi par la stratégie.

Le premier souci rencontré fut les demi-tours, si les angles étaient trop proches de Pi/2, la voiture avait tendance lors d'un virage à vouloir retourner en arrière. Pour corriger ceci il a fallu réduire les angles pour en quelque sorte mettre des œillères à la voiture pour qu'elle ne regarde que face à elle.

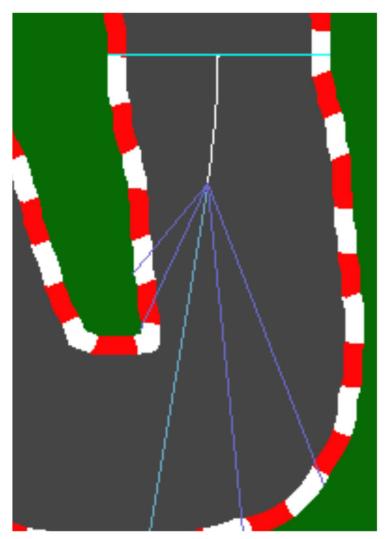
Une fois cela corrigé, tous les circuits en boucle ont pu être joué. Les circuits qui ne tourne pas en « rond » ne peuvent pas être joué avec ce radar, puisque plus la ligne d'arrivées est proche moins le radar renvoie va loin (il y a souvent un mur juste après la ligne), ce radar ne prenant pas en compte la ligne d'arrivées cela posé soucis. Pour passer outre ce problème, j'ai développé une stratégie arrivés qui sera vu plus loin.

¹ L'angle définit de combien on décale le rayon par rapport à la direction de la voiture.

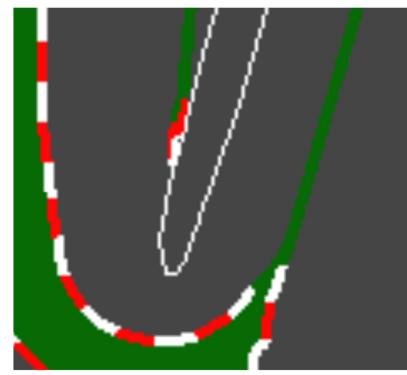
² Une commande se constitue de 2 nombré décimaux compris dans [-1 ; 1], il serve à quantifier l'accélération et la direction de la voiture.

³ Le radar est l'outil qui va renvoyer des scores à la stratégie pour qu'elle puisse choisir le faisceau le plus judicieux.

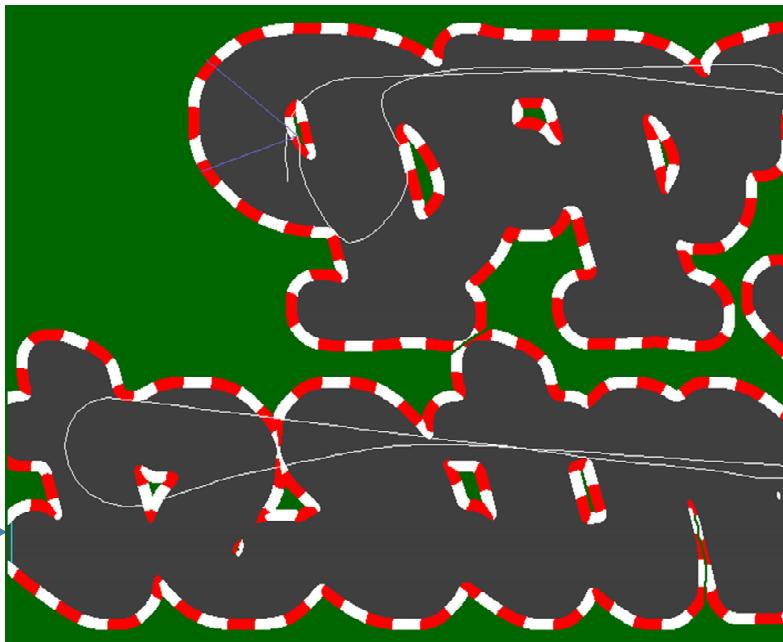
L'on remarque bien sur l'image que le faisceau qui va le plus loin est celui sélectionner (il est bleu clair au lieu



Un exemple de demi-tour effectué sur le circuit 3.



LE RADAR DIJKSTRA



Un exemple de ce qui se passe sur un circuit non cyclique.
L'arrivée donnant sur un mur, le faisceau le plus long n'est jamais celui qui passe par cette dernière.

Le radar classique montrant des limites (pas d'optimisation du parcours), on est passé sur un radar qui prend en compte la distance par rapport à la ligne d'arrivés.

Le principe de l'algorithme de dijkstra est d'initialisé un tableau d'entiers de la taille du circuit à l'infini puis chaque points de la ligne d'arrivées et placé dans un tas et leur case dans le tableau mise à 0.

Tant que le tas n'est pas vide on récupère le 1^{er} élément, on regarde si les cases aux alentours sont

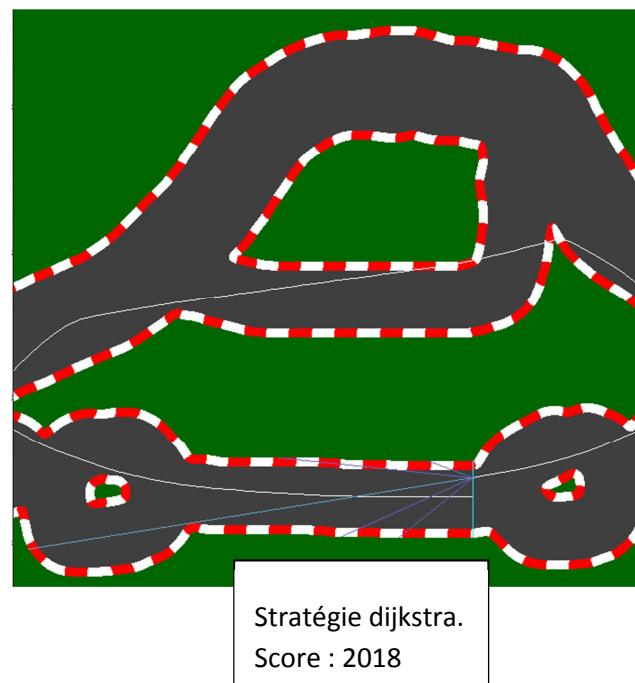
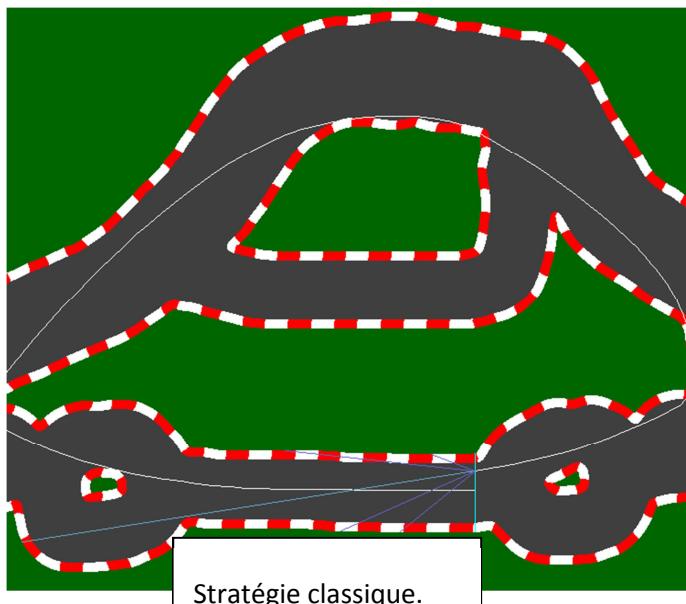
valide⁴, si elles le sont on calcule leur score (score de la case courant à qui on ajoute 10 pour une case directe ou 15 pour une case en diagonale (diagonale d'un carré : $\sqrt{côté^2 + côté^2}$, donc ici $\sqrt{2}$ ce qui multiplier par 10 se rapproche de 15)).

Une fois toutes les cases parcouru ont obtient un tableau d'entier.

Le premier souci rencontré a était de respecter le sens du circuit, pour cela il a fallu bloquer le passage de la ligne d'arrivées avec le produit scalaire (entre la direction d'arrivées et le sens dans lequel l'algorithme partait).

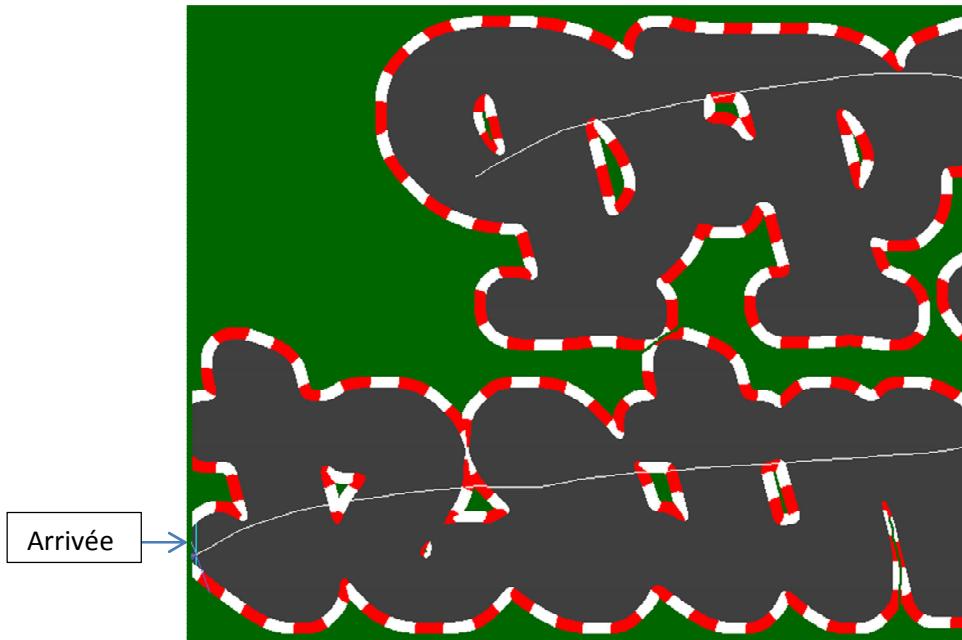
Le radar dijkstra projette ses faisceaux sur le tableau obtenu, la meilleure case vers laquelle se dirigée est la plus petite donc on renvoie le score en négatif pour s'adapter à la stratégie.

Avec ce nouveau radar la plupart des circuits ne posent plus de problèmes, certains circuit reste injouable ou bien le score obtenu est ridicule dut à des vitesses très faible pour passer dans des « tunnels ».



Les scores obtenus sont meilleurs avec le radar dijkstra qu'avec le radar classique.

⁴ Une case est valide si la voiture peut rouler dessus.



Avec l'algorithme de dijkstra, on n'a plus de soucis sur les circuits qui ne forment pas de boucle.

LA STRATEGIE SELECTION

La stratégie sélection regroupe plusieurs stratégies à qui ont associé un sélecteur⁵.

Elle parcourt la liste des sélecteurs dans l'ordre d'ajout, si le sélecteur renvoie vrai alors elle applique la stratégie qui lui correspond.

Par principe on ajoute une stratégie radar avec un sélecteur vrai en dernier choix pour éviter de bloquer la simulation.

Suite à cette stratégie, j'ai créé un sélecteur zone. Ce sélecteur prend un pourcentage et le tableau obtenu par dijkstra. Il fait le rapport du score de la case actuelle sur celui de la case de départ (score le plus grand sur la trajectoire) et si celui-ci est supérieur à celui donné en attribut il renvoie vrai.

Avec ces 2 ensembles on peut faire des mix de stratégies mais ceci est très compliqué à mettre en place et ne permet pas d'améliorer sensiblement les scores, c'est pourquoi je me limite la plupart du temps à ne pas mixer plus de 2 stratégies et n'utilise que très rarement un sélecteur zone.

LA STRATEGIE ARRIVEES

Le sélecteur de cette stratégie regarde si il y a un chemin possible entre la position de la voiture et les points de la ligne d'arrivés.

Pour raccourcir le temps d'exécution, on prend un point de la ligne d'arrivée sur 10.

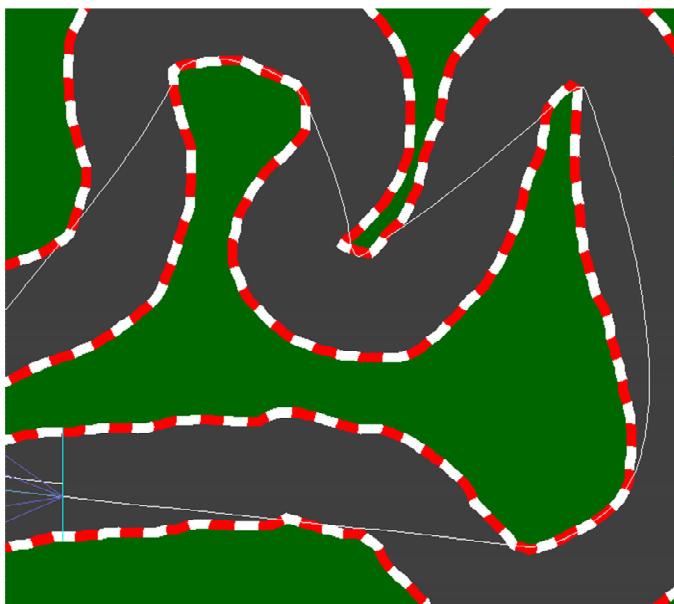
De plus, la stratégie devant refaire la même chose, j'ai choisi de lier les 2 pour éviter de doubler le parcours.

La stratégie effectue donc assez peu de calcul.

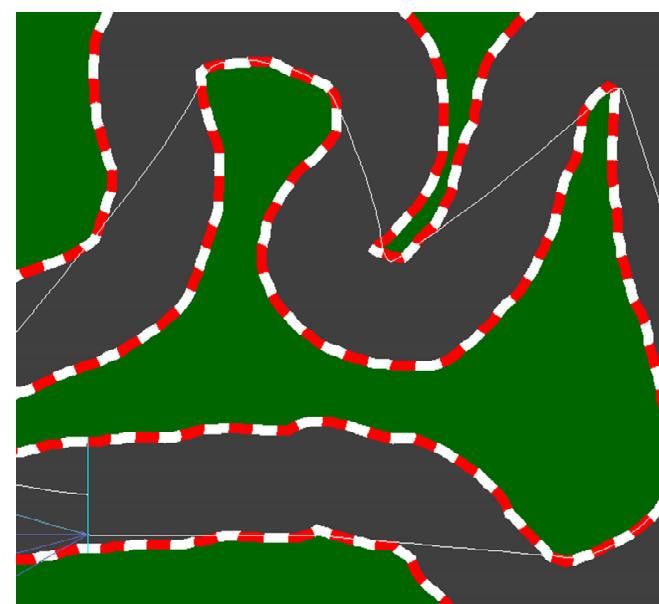
⁵ Le sélecteur analyse la situation et renvoie vrai si la stratégie peut être appliquée.

Cette stratégie ne peut être appliquée seul, d'où la stratégie précédente.

Les scores de dijkstra couplé à la stratégie arrivées améliore un petit peu les scores en prenant le chemin le plus court (pour le radar dijkstra, tous les points de la ligne d'arrivées sont à 0), le souci est dans l'adaptation des commandes à renvoyés en fonction de l'angle entre la voiture et le point de la ligne visé.



Stratégie dijkstra.
Pas de recherche de la meilleure



Stratégie dijkstra + arrivée.
Recherche de l'arrivée la plus

LA STRATEGIE POINT A POINT

Cette stratégie était suggérée dans l'un des tme puis pouvais avoir (et à eu) une utilité lors de l'examen sur machine.

Le sélecteur indique s'il y a un chemin possible entre le point visé et la position de la voiture.
La stratégie par la suite calcule l'angle entre la direction de la voiture et le vecteur (voiture, point visé).

En fonction de l'angle obtenu, une commande différente est renvoyée comme pour la stratégie arrivée, d'ailleurs le même problème se pose (les commandes moyennement adaptés).

LA STRATEGIE DECORATEUR

J'en ai pas parlé mais les stratégies précédentes peuvent cumuler plusieurs commandes de frein jusqu'à atteindre une vitesse nulle. Cela est embêtant.

Ma stratégie décorateur analyse la vitesse de la voiture et si cette vitesse est inférieure à un seuil décider dans le main alors l'accélération de la commande retournée est mise à 1.

C'est aussi ici que j'ai déplacé le bout de code permettant à la voiture de ne pas déraper puis lors de l'examen que j'ai traité le cas de la boue.

L'intérêt d'une stratégie décorateur est de traiter tous les cas à un seul endroit ainsi que de s'adapter directement à toute nouvelle stratégie.

LES MODULES

Ce que je nomme module sont des petites choses ajoutées sur le circuit qui donne un comportement différent à la simulation

LES OBSTACLES

L'on a vu précédemment que certains circuits restait injouable dut au parcours choisi par le radar dijkstra.

J'ai opté pour l'ajout d'obstacle pour bloquer ces chemins, ainsi le radar dijkstra choisi un autre chemin.

Pour ajouter ces derniers, j'ai créé une interface obstacle ce qui permet d'implémenter des obstacles de différentes forme bien que le traditionnel de forme rectangle soit le seul utile.

Du point de vue fonctionnel, les obstacles sont une surcouche c'est-à-dire qu'on ne modifie pas le circuit mais on ajoute un masque par-dessus.

Du point de vue code sa donne une nouvelle interface de circuit qui étend la précédente et connaît les méthodes pour gérer les obstacles. Niveau implémentation c'est de la décoration, on a un circuit de base et une liste d'obstacle. On parcourt la liste d'obstacle soit on est dans l'un d'eux soit on délègue.

LES ZONES

On a vu précédemment que définir des stratégies pour chaque portion du circuit était complexe. Lorsque j'ai créé les obstacles, je me suis demandé pourquoi ne pas faire des choses semblable mais qui donnerait un comportement différent à la voiture.

Les zones sont un petit peu comme des panneaux de limitation de vitesse dans la vie réelle, suivant le panneau on limite sa vitesse ou bien on va à fond.

J'ai pour le moment 3 types de zones :

- ─ la zone normal (zone par défaut, on ne fait rien de spécial)
- ─ la zone lente : la commande d'accélération est divisée par 2 pour permettre à la voiture de ralentir
- ─ la zone rapide : la commande d'accélération est mise à 1.

Ces zones possèdent en attribut leur type, pour les gérer, c'est le même principe que pour les obstacles une surcouche par-dessus le circuit.

Une zone qui aurait été bien utile aurait été la zone tunnel qui permettrait de passer dans les couloirs étroit, mais je n'ai pas eu le temps de me pencher dessus.

Grâce à la zone je gagne sur certains circuits quelque commande mais à vrai dire le résultat n'est pas flagrant.

CONCLUSION SUR LA PARTIE TACTIQUE

Mes scores sont pour la plupart raisonnables, j'ai même était en tête du classement à un moment de l'UE mais je n'ai ni pris le temps ni sut développer des stratégies plus complexes ou bien optimiser celle que j'avais déjà. Dans la suite du projet, je me suis surtout penché sur la création d'une interface graphique pratique et sobre qui me permette de faire énormément de chose sans avoir à bidouiller le main.

J'ai bien eu quelque idée d'optimisation comme la sélection des commandes automatiques suivant les angles renvoyés mais après maintes essaies qui furent tous des échecs (les scores obtenu était très loin de celui obtenu par la méthode traditionnel), j'ai décidé de stopper le développement et de me consacrer à d'autre chose.

L'INTERFACE GRAPHIQUE

J'ai passé une bonne partie de mon temps à essayer de faire une interface graphique modulable et qui puissent s'adapter à de nouveau environnement sans retouche.

Comme on a pu le voir sur les images précédentes, l'application est optimisé pour les grandes résolutions (comme sur les machines de la fac), c'est pour cela que je ne peux pas avoir l'affichage du circuit complet sur mon pc personnel.

LA FENETRE D'ACCUEIL



Comme je l'ai dit précédemment je voulais quelque-chose de sobre, c'est pour cela que j'ai choisi de travailler avec des barres de menu plutôt qu'avec des boutons.

Le seul bouton est celui qui donne accès à la notice, j'ai fait ce choix pour le mettre en évidence.

Pour créer cette fenêtre, j'ai utilisé :

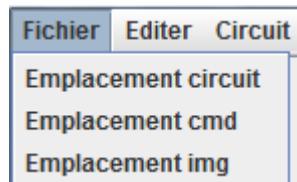
- Une JFrame (classe permettant d'ouvrir et de gerer une fenêtre).
- Une JMenuBar ajouté à la JFrame précédente par la méthode setJMenuBar.
- Un JPanel utilisant un BorderLayout, ceci définit un panneau que l'on va placer dans la fenêtre et où l'on pourra ajouter d'autre élément en les plaçant de manière structuré grâce

BorderLayout (ce Layout permet un placement gauche-droite / haut-bas / centre mais n'admet pas de combinaison comme bas-droite).

- ✚ Un JLabel pour l'image, placée au centre du panneau.
- ✚ Et enfin un JButton pour le bouton notice, placer en haut du panneau.

LES MENUS

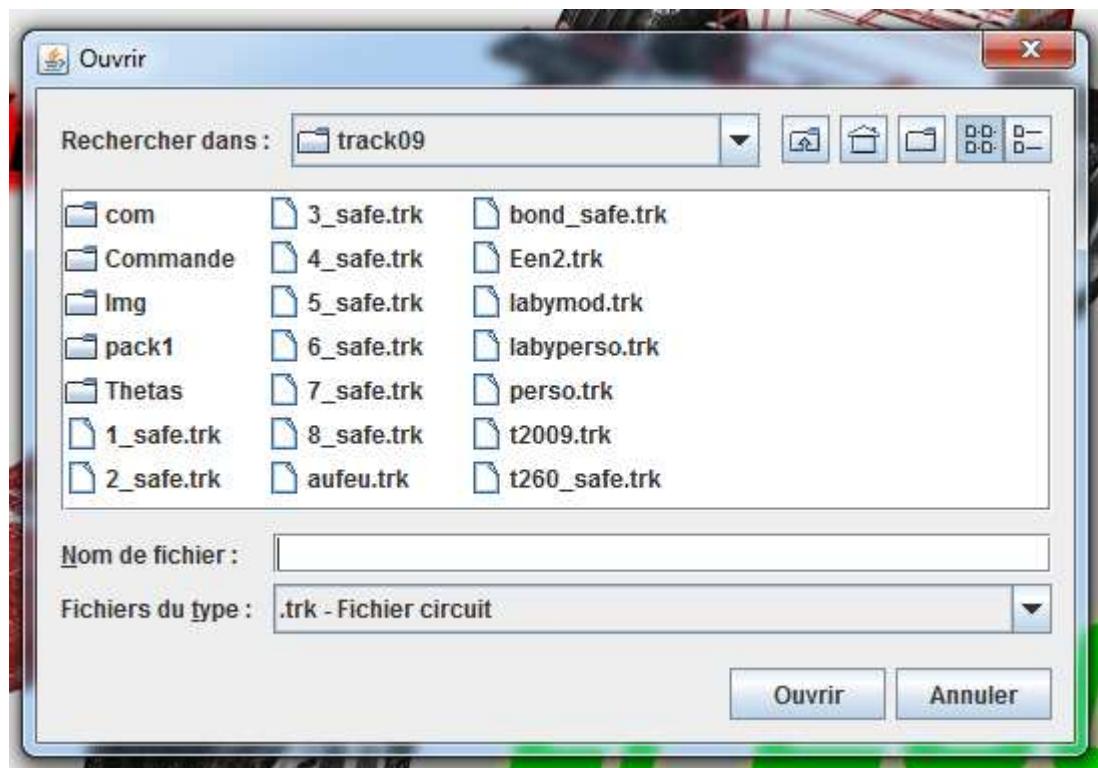
1 – FICHIER



Le JMenu fichier contient la partie pour gérer les variables globales de ce programme, cela se compose de 3 JMenuItem.

Les 3 actions qu'offre ces choix permettent de choisir l'emplacement de sauvegarde de la liste de commande, l'emplacement de sauvegarde de l'image et enfin de l'emplacement de chargement du circuit (et de lancer la création de l'instance en même temps).

Un clic sur l'un de ces 3 choix donne accès à une nouvelle fenêtre de ce type :



Cette « boîte de dialogue » est une JFileChooser couplé avec un FileFilter.

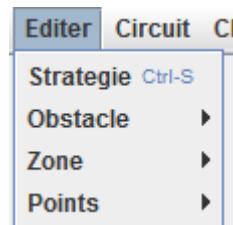
Cette objet tout fait de JAVA permet l'accès à une boîte de dialogue de sélection d'un fichier, le FileFilter tant qu'à lui permet de restreindre ce choix au type voulu (dans l'exemple à un .trk) ou de faire annuler la sélection (en retournant null), si le fichier choisi ne comporte pas la bonne extension.

L'on remarque que grâce à cette outil, l'on peut aller charger un circuit qui se trouve dans n'importe quel dossier voir périphérique relié à notre machine, ceci rentre parfaitement dans ma volonté de vouloir faire un logiciel qui s'adapte à de nouveau environnement comme un nouveau dossier de circuit.

Je prendrais comme exemple l'examen que l'on a eu, j'ai pu ouvrir les nouveaux circuits directement sans avoir besoin de retoucher un morceau du code.

Pour les commandes et l'image, ceci assigne le fichier choisi au variable globale définit dans le main.

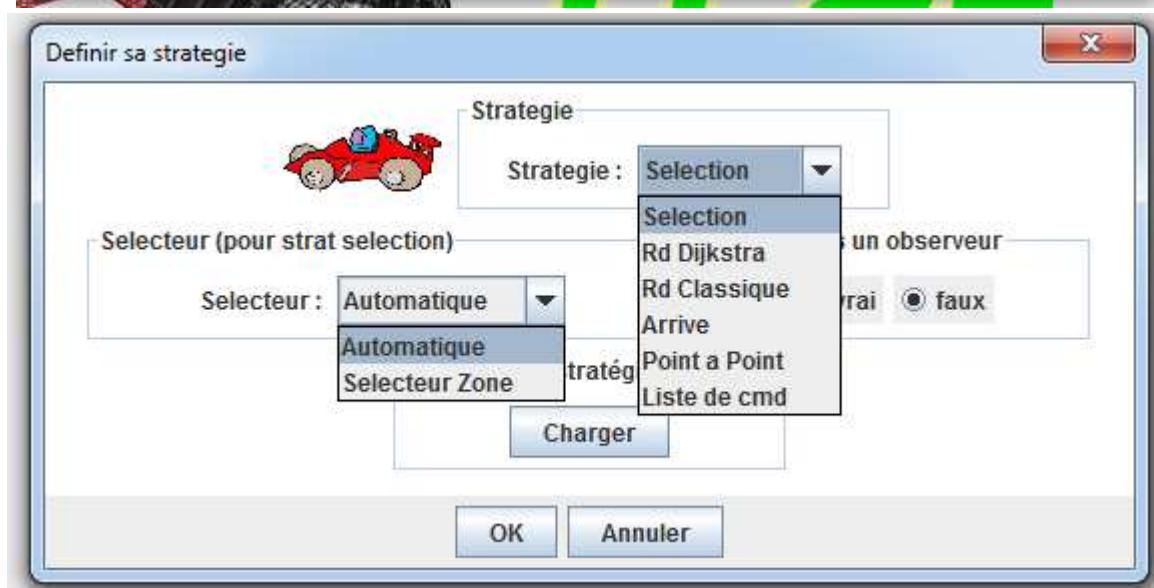
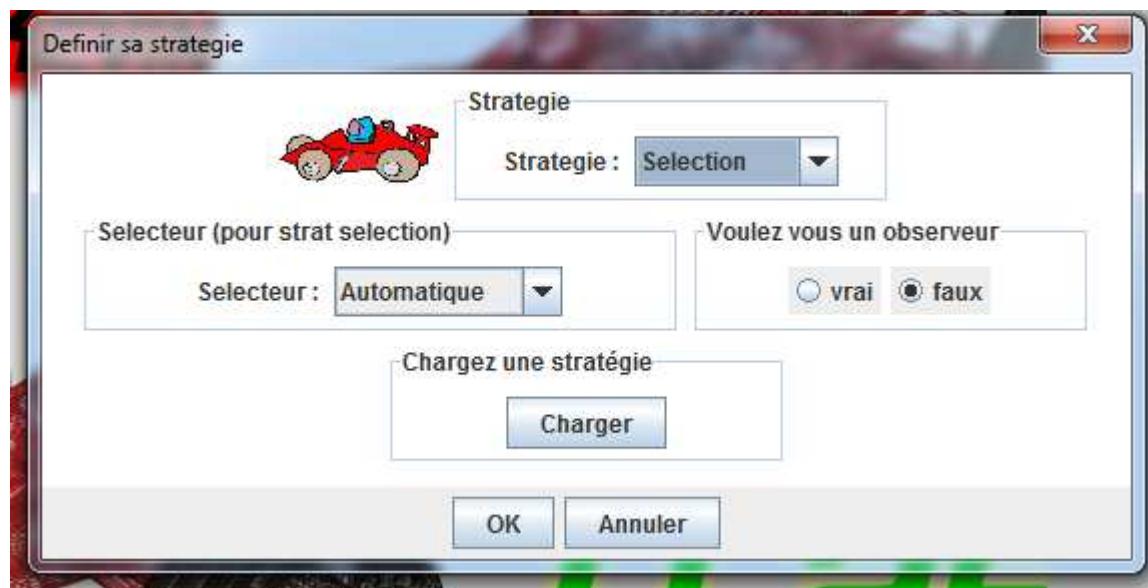
2 – EDITER



Avant d'accéder à ce menu, il faut tout d'abord lancer un circuit (comme on l'a vu précédemment en passant par une boîte de dialogue ou comme nous le verrons par la suite, les principaux sont répertoriés dans un menu et des raccourcis clavier leur sont attribués).

STRATEGIE

L'item stratégie ouvre une boîte de dialogue avec laquelle on définit la stratégie utilisé pour la simulation.



Dans l'onglet stratégie, on sélectionne la stratégie voulu :

Les choix Rd Dijkstra, Rd Classique, Arrive crée automatique la stratégie associé bien que je le rappelle la stratégie arrivé toute seul ne fonctionnera pas.

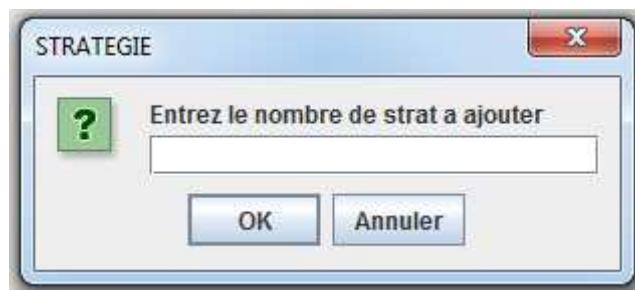
Pour la stratégie Point à Point, il faut avoir initialisé la liste de point avant (nous verrons sa dans le petit 4 de cette partie).

Lorsque l'on sélectionne « Liste de Cmd », une boîte de dialogue qui permet de choisir le fichier de liste de commande que l'on désire lire s'ouvre (même principe que précédemment).

Avec tous les choix précédents, le sélecteur se crée automatiquement.

Pour créer une stratégie combiné de plusieurs autres, c'est à dire une stratégie sélection il faut choisir le premiers choix, ce dernière ouvre une boîte de dialogue qui demande le nombre de stratégie à ajouter. Une fois un nombre rentré, on retombe sur la fenêtre précédente pour choisir la 1^{ère} stratégie à ajouter. L'ajout de stratégie sélection dans une autre stratégie sélection ne fonctionne pas en mode graphique.

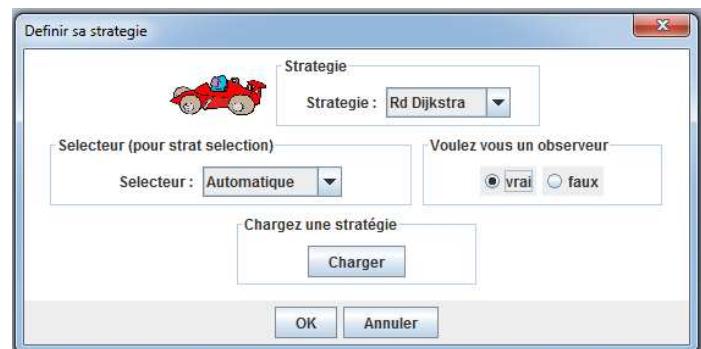
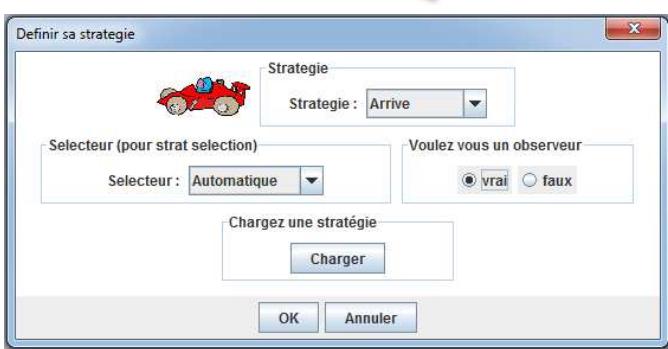
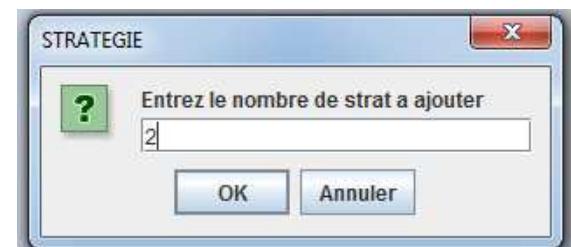
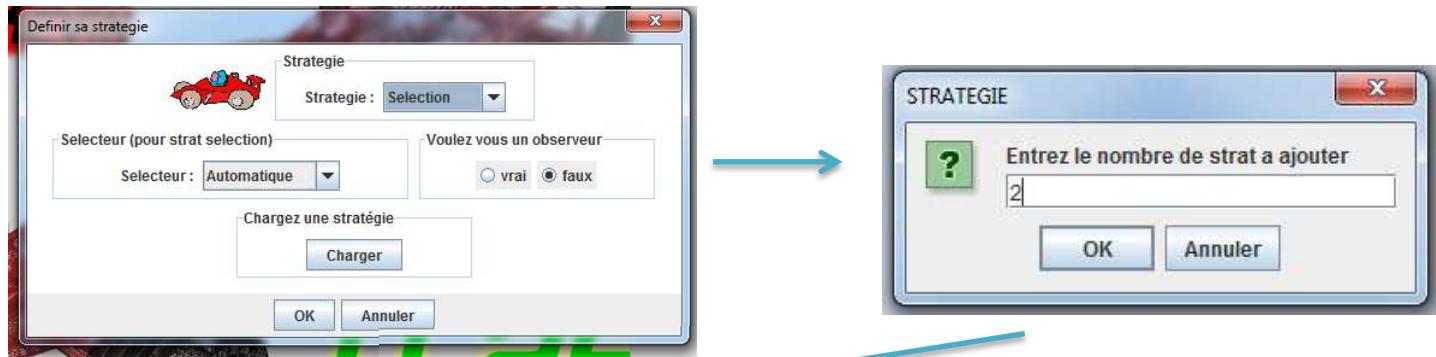
Si je ne rencontrais pas le problème précédent, choisir un sélecteur zone aurait une utilité, ce dernier correspond au sélecteur zone vu dans 1^{er} partie de ce rapport. Il ouvre une petite boîte de dialogue demande le % de la zone à couvrir.



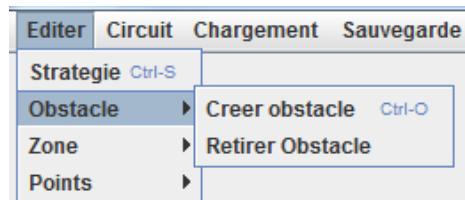
Pour le moment, la simulation fonctionne, mais on n'affiche pas la course, pour ajouter des observer pour nos stratégies, il faut choisir l'option vrai dans la fenêtre précédente. (Il faut par contre charger la base de l'affichage avant, nous verrons sa bientôt.)

Cette manière de choisir la stratégie n'est pas des plus pratiques, j'ai essayé un moment de voir si mettre sa sous forme de tableau était réalisable, mais je n'ai pas su faire des tableaux à plusieurs étages pour s'adapter à la stratégie sélection.

Voici un petit exemple pour choisir une stratégie dijkstra + arrivée avec observer :



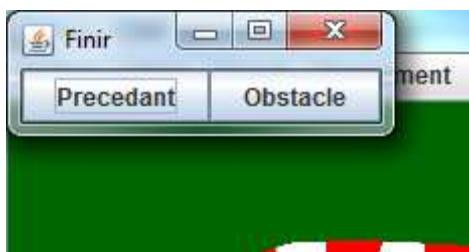
OBSTACLE



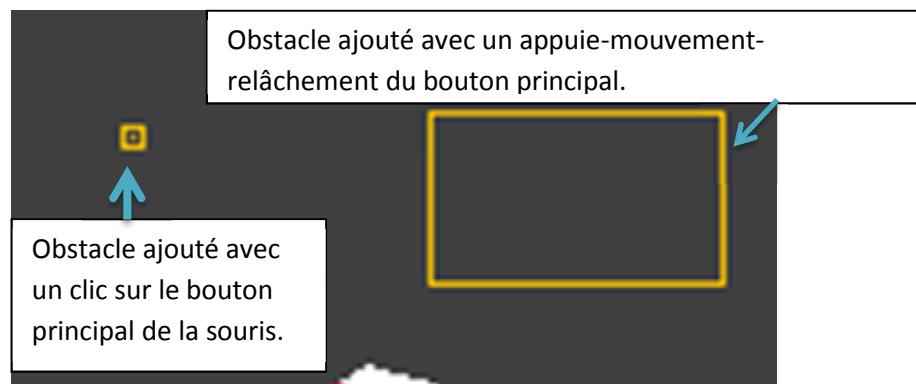
Pour passer certains circuits, on avait vu qu'il fallait mettre des obstacles.

L'action créer obstacle ajoute un MouseListener⁶ à la simulation, ce qui permet lorsque l'on clique sur le circuit d'ajouter un obstacle.

Ceci ouvre aussi une nouvelle fenêtre qui permet de gérer les obstacles.

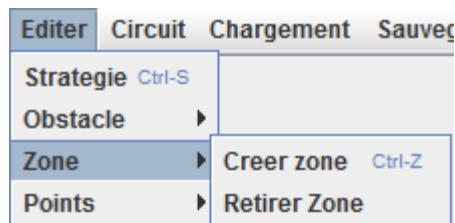


Le bouton « précédent » permet de retirer le dernier obstacle ajouté, le bouton obstacle permet de mettre fin à la création d'obstacle ce qui retire le MouseListener.



La commande « Retirer Obstacle » permet tant qu'à elle à retirer tous les obstacles.

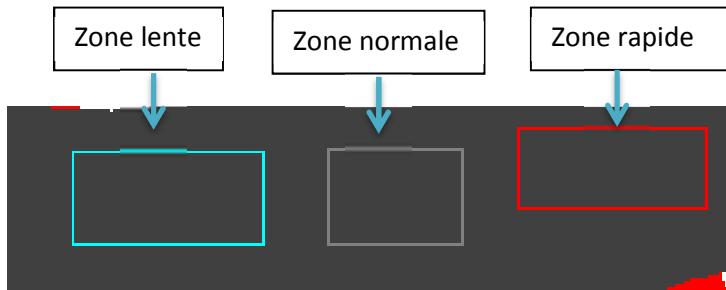
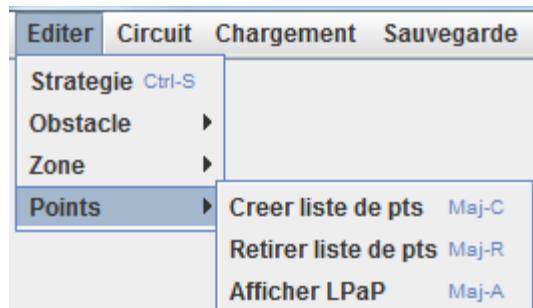
⁶ Un MouseListener est un écouteur de souris, il traite les actions effectuées par la souris.

ZONE

Même fonctionnement que les obstacles (ce qui est plutôt normal, la zone est une extension de l'obstacle), à un détail prêt lors de la création.

L'on retrouve le bouton précédent et celui nommé zone (correspondant au fonctionnement du bouton obstacle de tout à l'heure).

Ce qui diffère est la liste de choix pour définir la nature de la zone, ainsi que le bouton ok qui permet de valider ce choix.

**POINTS**

On retrouve le même principe avec « Crée » et « Retirer », « Afficher LPaP » affiche dans la console tous les points de la liste.

On retrouve les traditionnels boutons « Précédant » et « Points ».



3 – CIRCUITS

Circuit	Chargement
1	Ctrl-1
2	Ctrl-2
3	Ctrl-3
4	Ctrl-4
5	Ctrl-5
6	Ctrl-6
7	Ctrl-7
8	Ctrl-8
aufeu	Ctrl-9
bond_safe	Ctrl-0
Een2	
labymod	
labyperso	
perso	
t2009	
t260_safe	

L'onglet circuit permet d'accéder plus rapidement au circuit de base (à condition que les circuits soient dans un dossier nommé track09 dans le répertoire du projet).

De plus cela permet d'associer des raccourcis clavier à certains, cela est très utile pour les circuits de base lors de test répété.

4 – CHARGEMENT

Chargement	Sauvegarde
Charg Circuit	
Charg Stratégie	

On a vu qu'il était possible d'ajouter des obstacles et des zones à nos circuits, tout cela est bien joli mais temporaire, j'ai donc choisi de rendre cela serializable. La sauvegarde d'un circuit comprend aussi les obstacles et les zones mais pas dijkstra, on le recalcule (cela permet d'avoir un fichier pas trop gros). L'extension de sauvegarde est .trk.seria, j'ai choisi de quitter l'extension .trk pour ne pas me perdre entre les 2 types de fichiers circuits.

Le chargement de stratégie est similaire, cela consiste à récupérer une stratégie serialisé au format .strat.

5 – SAUVEGARDE

Sauvegarde	App
Sv Commande	
Sv paramètre	
Sv Circuit	
Sv Stratégie	

Une fois la simulation joué on peut enregistrer la liste des commandes dans le fichier que l'on veut (pour pas qu'elle soit écrasé par la simulation suivante) grâce à « Sv Commande ».

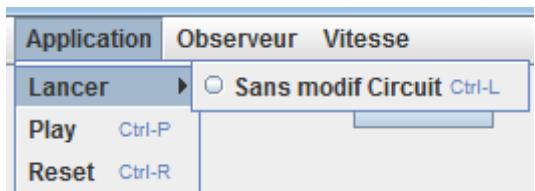
On a vu le chargement du circuit, « Sv Circuit » à l'inverse, l'enregistre. De même pour la stratégie.

« Sv paramètre » est un petit peu spécial, pour mes stratégie radar, j'ai besoin de rentrer les angles et les commandes associé manuellement dans le main, si je sauvegarde ces paramètres, lorsque je lance un circuit il regarde si il y a un fichier de paramètre existant (il sait lequel lui correspond en fonction du nom), si oui alors il remplace ceux du main par ceux de ce dernier sinon il joue la simulation avec les paramètres du main.

Suivant les circuits les paramètres changent, avec cette méthode je n'ai pas besoin de changer le

main à chaque changement de circuit puisque si je suis content du résultat obtenu je sauvegarde les paramètres.

6 – APPLICATION



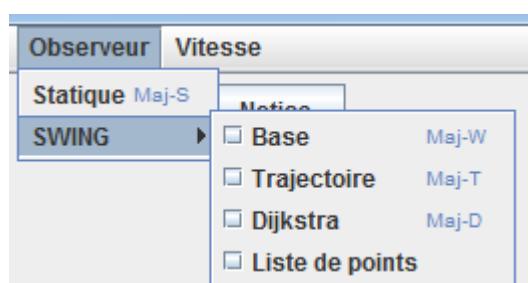
L'onglet lancer correspond à choisir un circuit. A l'origine il y avait 2 choix : choisir un circuit de base ou bien un algorithme modifier automatique le circuit pour boucher les « tunnels » mais cela ne fonctionnait pas tout le temps. Lancer un circuit « sans modif circuit » permet d'accéder à la boîte de dialogue de chargement d'un circuit comme Fichier->Emplacement Circuit.

La commande Play permet de lancer la simulation.

Le Reset remet la simulation et la stratégie à 0, par contre le circuit reste chargé.

Je préfère tout de même relancer le logiciel à chaque fois puisque de temps à autre le nombre de commande entre la simulation de base et la simulation lancé après un reset diffère.

7 – OBSERVEUR



C'est ici que l'on gère les différentes choses que l'on veut observer.

L'observer statique permet d'afficher la trajectoire de la voiture sur l'image qui est sauvegardé à la fin de la simulation.

L'affichage en temps réels passe par l'onglet « SWING ».

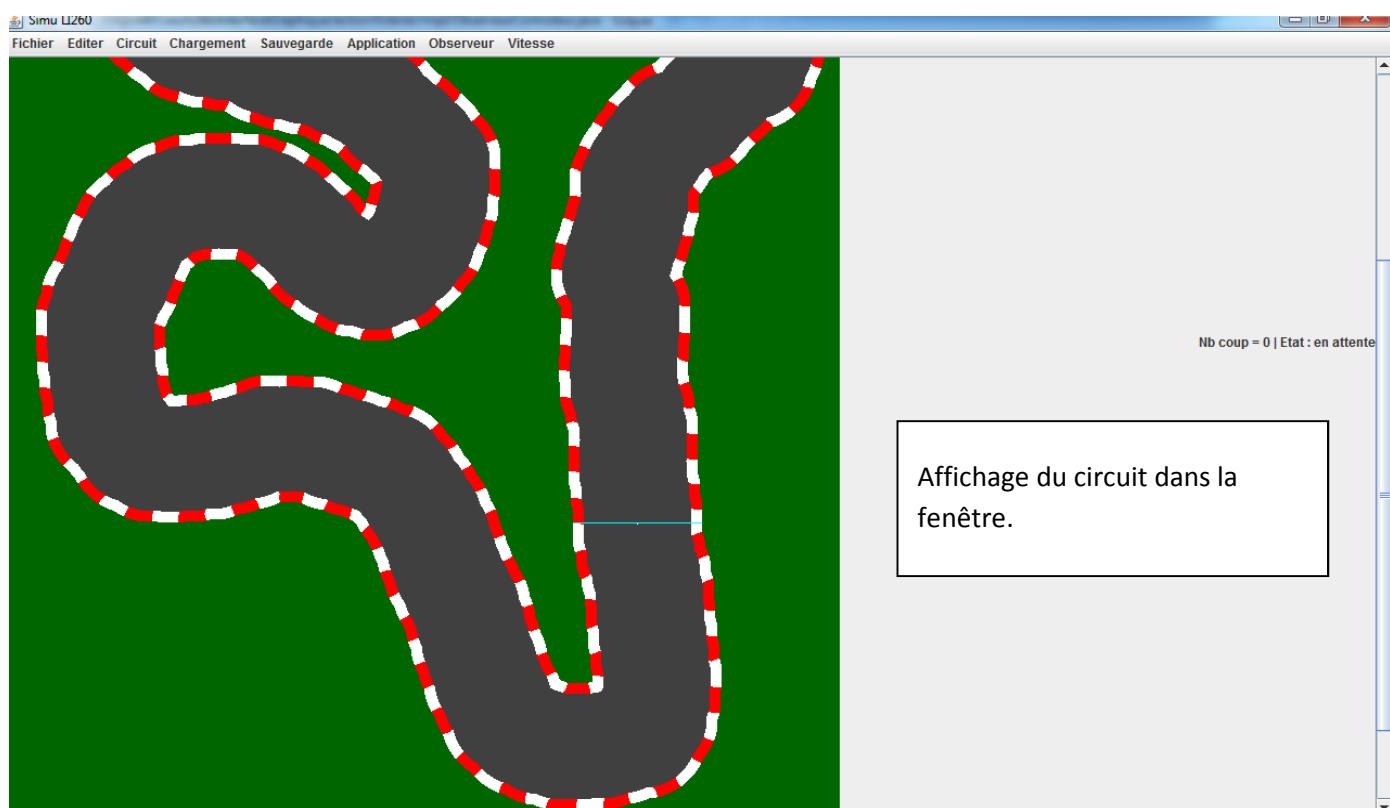
La Base correspond à l'affichage du circuit et permet d'ajouter d'autre composant graphique, on ne peut pas faire afficher la trajectoire sans avoir chargé au préalable cette partie. D'un côté faire afficher la trajectoire sans support n'a pas d'intérêt.

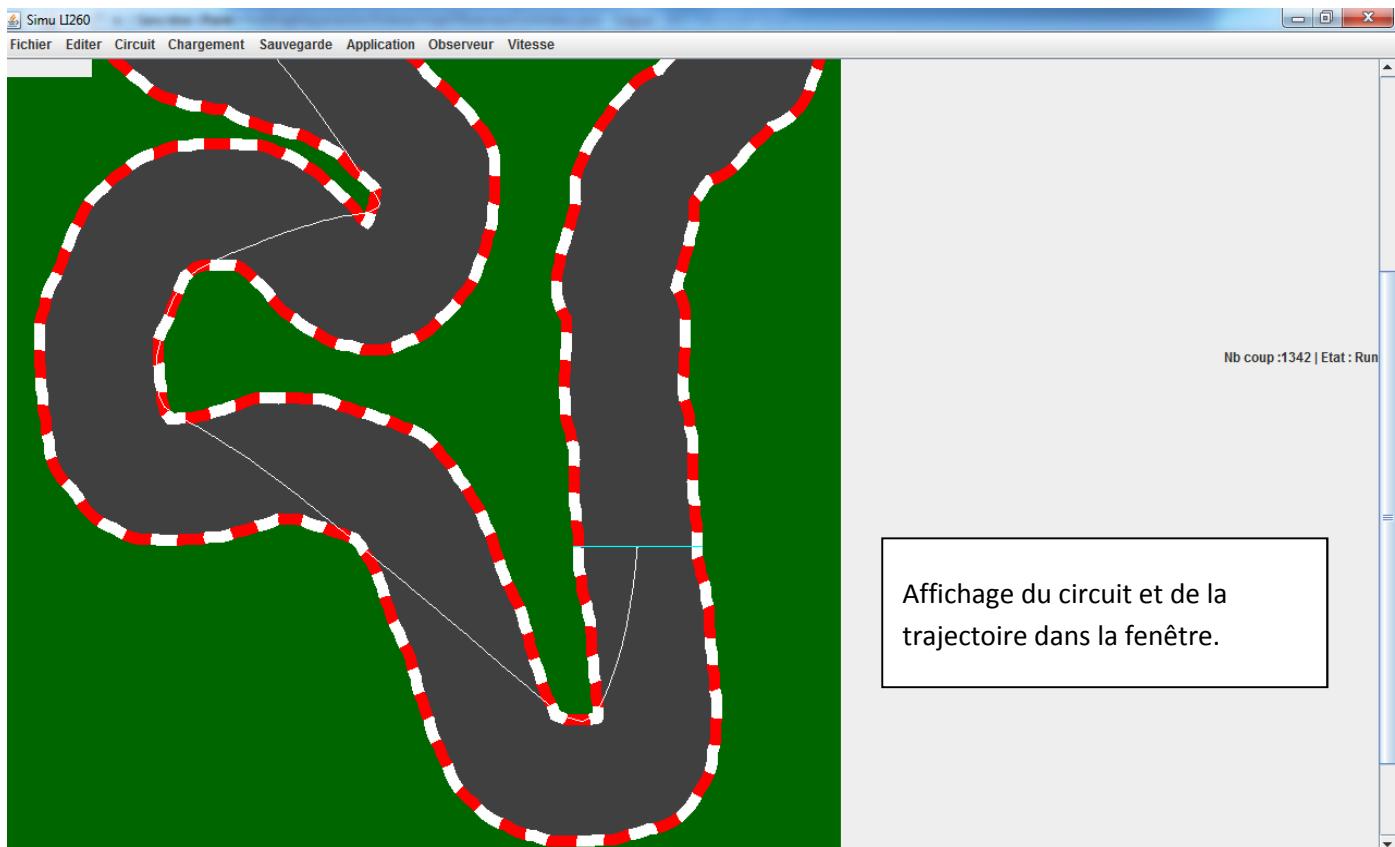
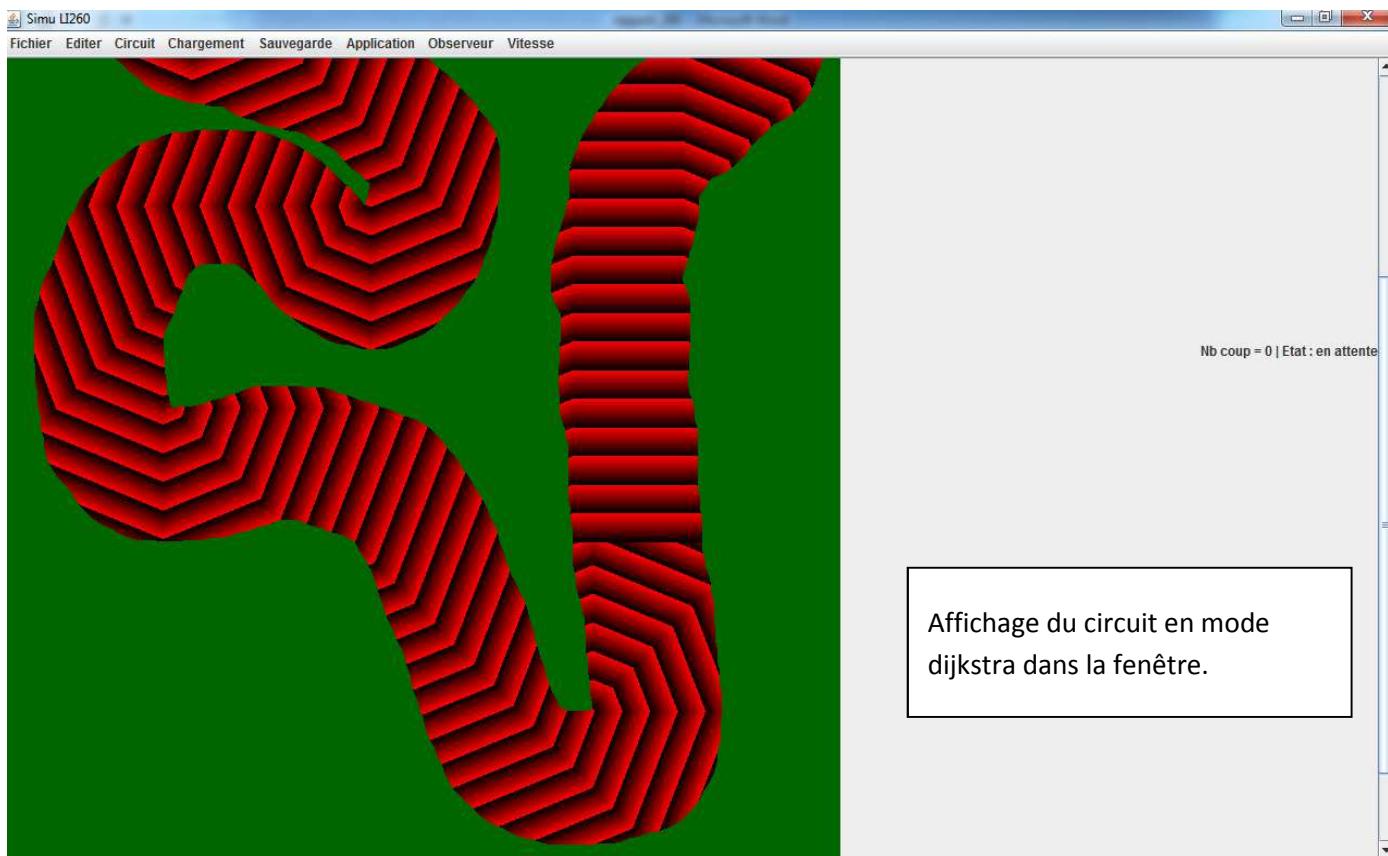
Comme son nom l'indique, « Trajectoire » fait afficher la trajectoire de la voiture. Si l'on branche cette partie, il n'y a pas besoin de charger l'affichage statique pour avoir la trajectoire sur l'image de fin, il y a un seul outil trajectoire pour visualiser la partie statique et la partie dynamique.

Un appuie sur « Dijkstra » entraîne l'affichage du circuit en fonction du calcul de dijkstra.

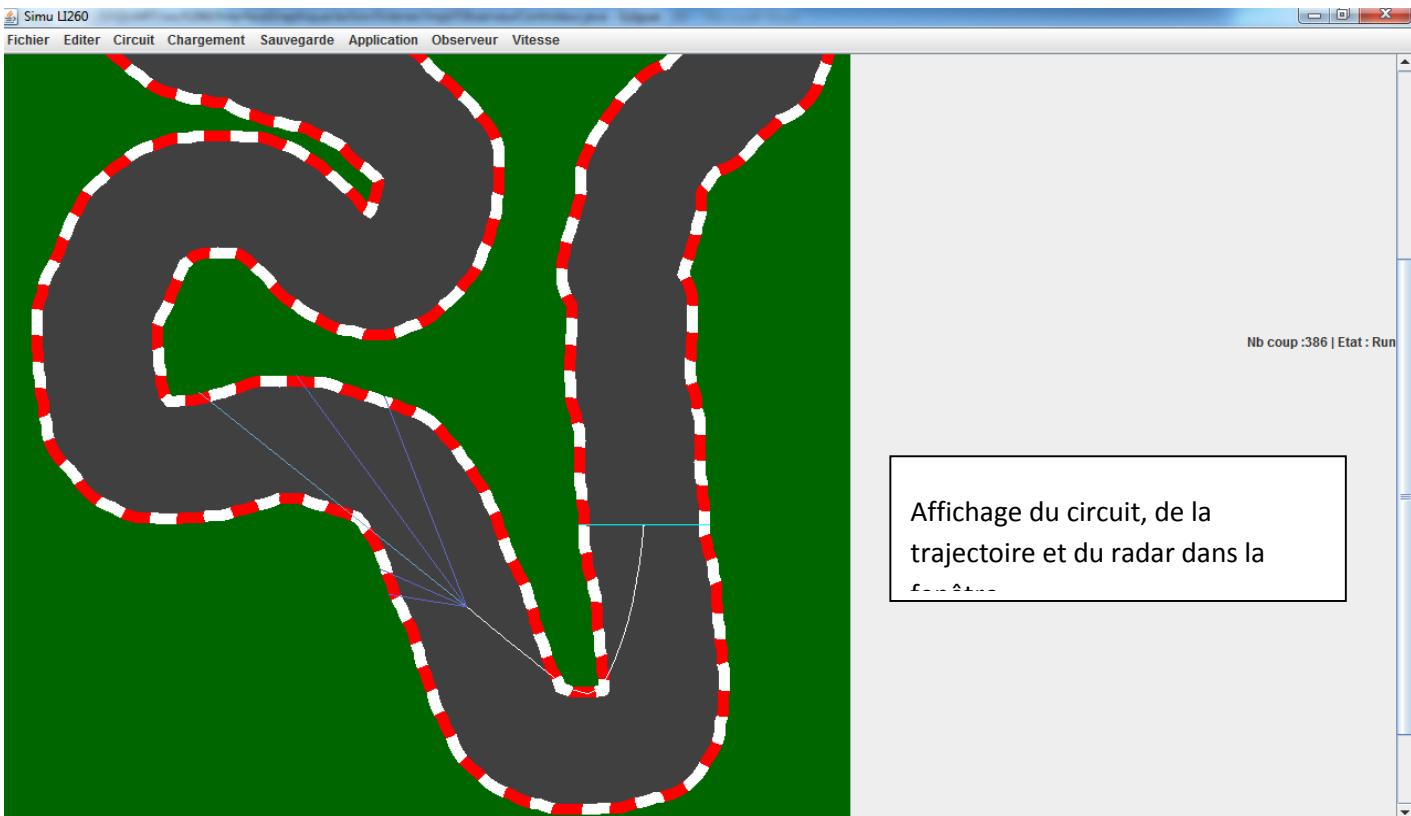
La dernière option n'a que peu d'utilité puisque lors de la création de point, l' s'ajoute automatiquement.

Il faut faire attention à l'ordre de sélection des observeurs, partir de celui qui occupe le plus d'espaces au plus ponctuel.

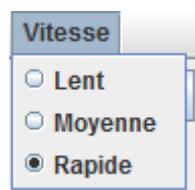




Pour afficher le rendu des stratégies, cela se passe lors du choix en cochant la case oui.



8 – VITESSE



Ce menu permet de régler la vitesse de la simulation, par défaut le réglage est sur rapide.

Changer la vitesse agit sur la variable qui gère le temps du Sleep autour du PlayOneShot.

CONCLUSION SUR LA PARTIE GRAPHIQUE

Je pense avoir réussi les objectifs que je m'étais imposé sur l'interface graphique, je la trouve assez complète tout en restant propre.

J'ai bien les outils graphiques correspondant à mes ajouts tactiques (obstacles, zones et points).

A l'exception des angles et commandes à modifier dans le main suivant les situations, j'ai réussi à mettre une barrière d'abstraction entre l'utilisateur et le code.

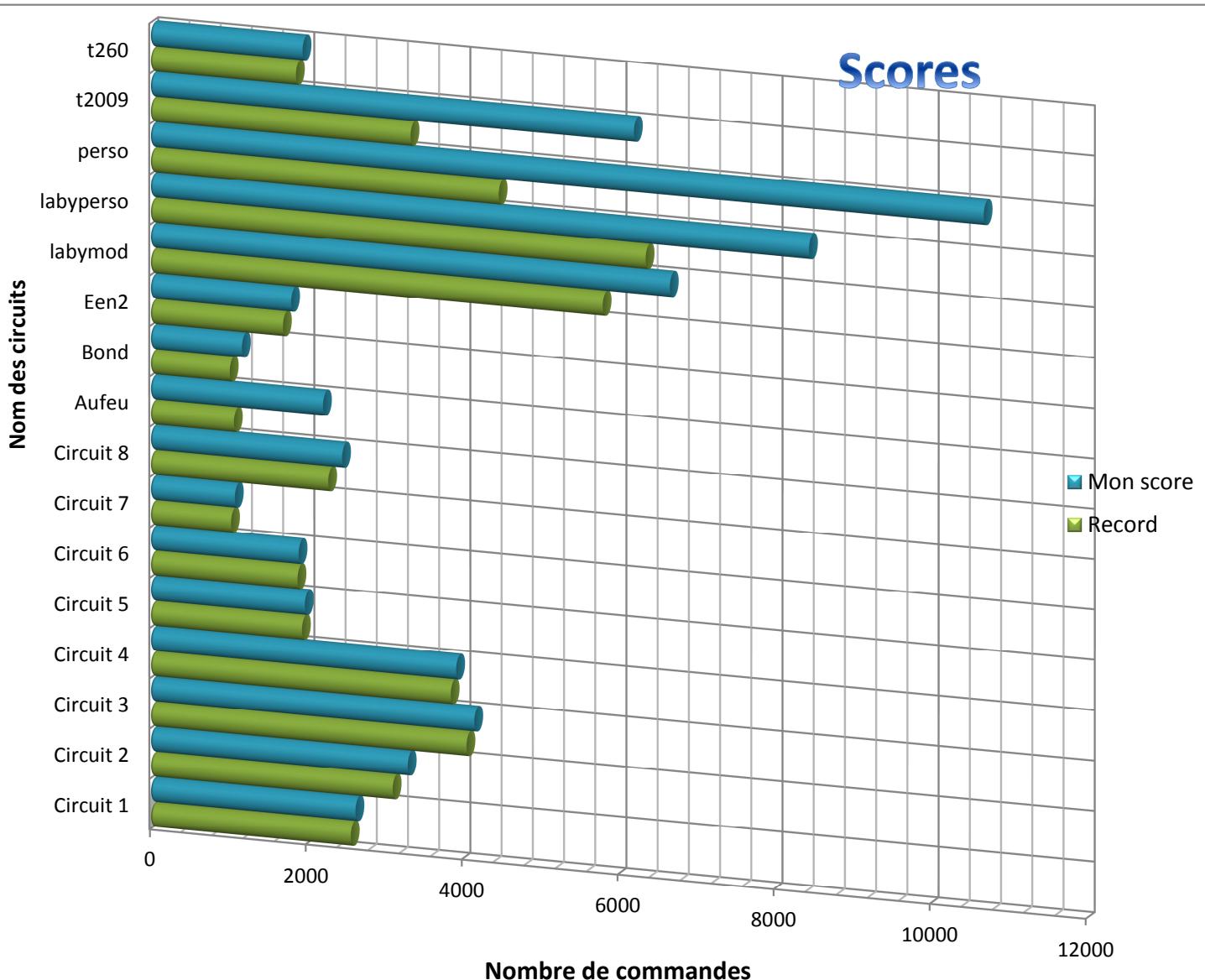
LES PACKAGE

Afin de naviguer de manière plus aisée dans un projet, le mieux est de le découper en différents dossier (package), j'ai choisi la hiérarchie suivante :

- ─ Interface Graphique, tout ce qui permet d'afficher et de gérer les commandes graphiques
 - Action-Listener
 - File, on y retrouve les filtres ainsi que les différentes méthodes de sauvegarde et chargement
 - JPanel, la fenêtre et son contenu
 - Mouse-Listener

- Mains
- MVC
 - Event
 - Listener
 - Sender
 - View-Observeur
- Plateau : les supports de la course
 - Circuit
 - Géométrie, gestion des classes touchant au vecteur
 - Obstacle, contient les obstacles et les zones
 - Voiture
- Simulation
- Tactique : contient « l'intelligence artificielle » du projet
 - Dijkstra
 - Radar
 - Sélecteur
 - Stratégie

RESULTATS



Le graphique précédent affiche mon score comparé au meilleurs score réalisé. L'on remarque que je suis assez proche du record sur la plupart des circuits par contre sur les circuits complexe j'ai certains scores ridicules (comme sur t2009, perso et labyperso).

Les fichiers de commandes fournis avec le projet (dans track09/com) ne correspondent pas toujours avec mes meilleurs scores obtenu (assez éloigné d'ailleurs sur certains). La raison principale est que je n'ai plus en tête l'endroit exact où placer mes obstacles ainsi que les commandes associées à chaque angle.

CONCLUSION

Cette UE m'a permis de découvrir les étapes d'accomplissements d'un projet qui prend de l'envergure et qui donne une approche d'une réalisation professionnel.

Travailler seul bien que n'étant pas vraiment le but recherché dans ce genre de travail permet en contrepartie de maîtriser toutes les notions du programme conçut et ceci m'a permis de progresser énormément en JAVA et dans la rigueur de programmation.

La principale amélioration à faire pour avoir un bon projet serait au niveau tactique.

Pour le moment la stratégie principale (dijkstra), regarde devant soi et repère le score le plus faible or ce score peut être sur un bord du circuit, tant que la voiture n'atteint pas une position qui donne accès au reste du circuit, elle fonce droit vers un mur.

Tourner au dernier moment conduit très souvent à un échec, il faudrait donc que la voiture puisse voir sa situation quelque peu plus loin (dans la vie réelle la stratégie en sport automobile et d'anticiper et de se préparer à l'approche d'un virage).

Grâce à ça on pourrait envisager de faire extérieur – intérieur – extérieur pour améliorer les chances de réussite et normalement aussi améliorer le score.