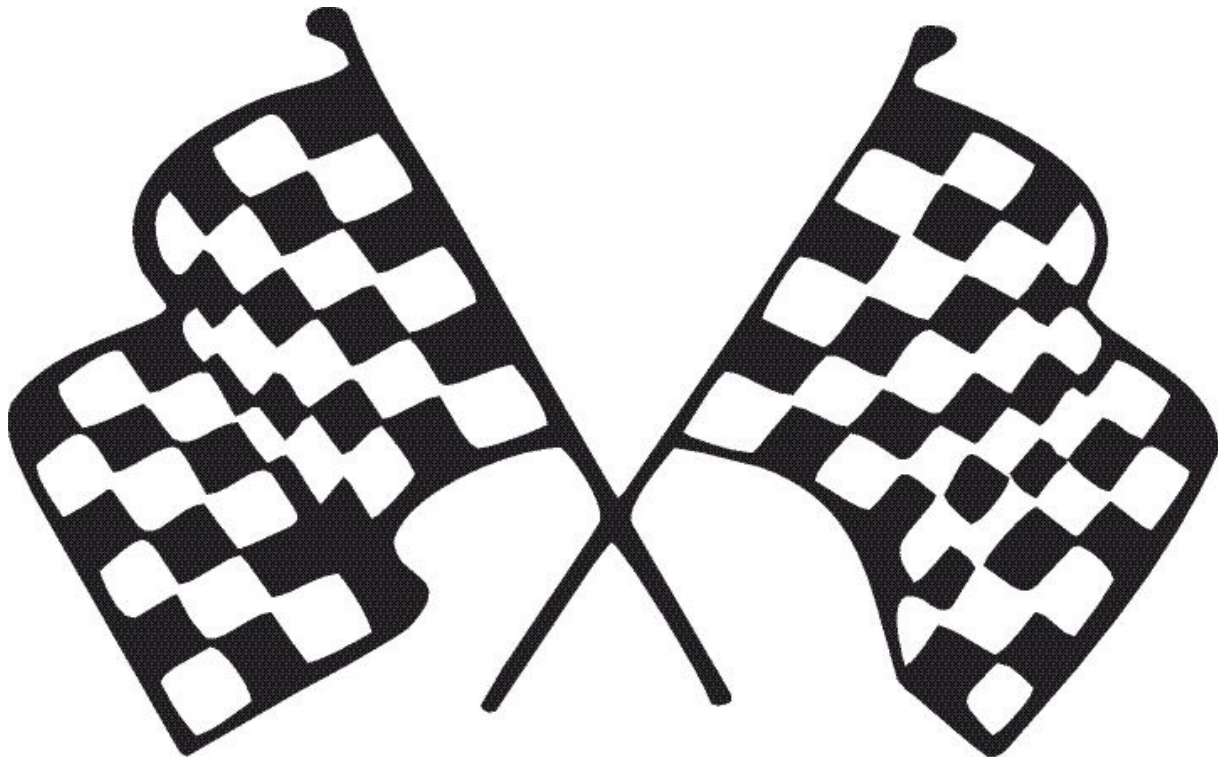




Li260 - Projet en Ocaml

Mars – Avril 2007

Sommaire :

<u>I - Présentation générale du projet</u>	Page 3
<u>II - Au niveau de la voiture</u>	Page 3
1°) <u>Les problèmes rencontrés</u>	Page 3
2°) <u>Nos améliorations</u>	Page 4
<u>III - Contrôle de la voiture avec un radar</u>	Page 4
1°) <u>Principe du radar</u>	Page 4
2°) <u>Mise en œuvre du radar</u>	Page 4
3°) <u>Problèmes rencontrés</u>	Page 8
<u>IV - Apprentissage par renforcement</u>	Page 9
1°) <u>Principe de l'apprentissage</u>	Page 9
2°) <u>Mise en œuvre de l'algorithme (état d'avancement du projet)</u>	Page 9
<u>V – Nos idées pour la suite</u>	Page 9
1°) <u>Ce que l'on compte faire</u>	Page 9
2°) <u>Ce que l'on pourrait faire ...</u>	Page 10

I - Présentation générale du projet

Le projet de l'UE LI260 propose de réaliser un simulateur autonome de voiture à travers une interface graphique. Il propose aussi de rendre compte des différents algorithmes d'apprentissages.

Afin d'aller au but de ce projet, nous avons développé un programme qui se subdivise en plusieurs modules :

- Un module qui permet la gestion de la voiture
- Un module qui gère l'affichage graphique à l'écran
- Un module pour le radar
- Un module outils avec des fonctions utiles
- enfin un module qui s'occupe de la simulation

Pour réunir et compiler ensemble ces modules nous utilisons un makefile.

II - Au niveau de la voiture

1°) Les problèmes rencontrés

D'une certaine façon, la mise en place de la voiture ainsi que de son contrôle au clavier ne fut guère aisée.

D'une part, les caractéristiques fournies dans les sujets de TME n'autorisaient pas un comportement physique réel de la voiture. Celle-ci avançait lentement ou tournait trop vite. On a pu corriger cela en affectant à la voiture les caractéristiques suivantes :

```
val car : voiture = { vitesse = 0.;  
  direction = { x = 1.; y = 0.; };  
  position = { x = 100.; y = 50.; };  
  angle = 0.;  
  pivot = 0.;  
  caract = { v_max = 50.;  
    braquage = 0.2;  
    f_acc = 1.;  
    f_frein = 8.;  
    turn = 0.05;  
    f_1 = 1.;  
    f_2 = 1.;  
    f_retour_volant = 0.05;  
    m = 1500. } }
```

D'autre part, pour le déplacement du véhicule au clavier, nous avons essayé plusieurs solutions jusqu'à obtenir une version satisfaisante.

Une première version de cet algorithme a consisté en une lecture dite bloquante des événements. C'est à dire en utilisant seulement `Key_pressed` dans la fonction `wait_next_event` qui récupère les événements. Cette première version ne convenait pas car elle ralentissait trop l'exécution et rendait lourde l'interaction avec l'utilisateur.

Une seconde version de cet algorithme a consisté à une lecture dite non bloquante des événements. Pour cela nous avons utilisé l'événement `Poll` dans la fonction `wait_next_event` et

la fonction `read_key()` qui nous a permis de récupérer la touche frappée par l'utilisateur. Cette seconde version était satisfaisante car permettait une grande interaction avec l'utilisateur.

2°) Nos améliorations

Nous avons eu le temps d'améliorer un peu le comportement de la voiture et de l'inertie dans celle-ci.

En effet lorsque l'utilisateur lâche une touche qu'il vient de presser, la voiture ne se stoppe pas brutalement, mais ralenti jusqu'à atteindre une vitesse nulle. Explicitement dans le code nous détectons l'appuie d'aucune touche et nous calculons quand même la prochaine position de la voiture (avec l'action Rien comme paramètre) et nous l'affichons. Ainsi la voiture subit toujours les mêmes forces mais n'accélère pas, donc la vitesse de la voiture diminue progressivement, ce qui crée son inertie.

III - Contrôle de la voiture avec un radar

1°) Principe du radar

Afin de rendre autonome notre voiture sur un circuit, nous avons mis en place un système de radar qui doit normalement conduire notre véhicule de la ligne de départ à la ligne d'arrivée du circuit dans lequel elle se trouvait.

Le principe du radar est donc le suivant : à partir de la position courante la voiture, on calcule la meilleure distance (pas forcément la plus grande distance, mais la plus adaptée à la situation) mais la et le meilleur angle pour lesquelles la voiture ne sort pas de la route et on s'y rend. De cette manière la voiture traverse le circuit jusqu'à la ligne d'arrivée en ayant obtenue après chaque appel au radar une distance maximum de parcours du terrain pour notre position.

2°) Mise en œuvre du radar

Après plusieurs essais non concluants, on arrive enfin à l'algorithme suivant pour le radar :

Tant que la voiture est sur une case noir

On choisit la meilleure action pour la voiture

On fait avancer la voiture avec l'action que l'on a choisit

On affiche la position de la voiture à l'écran

Fin tant que

Pour choisir la meilleur action à effectuer :

On calcule la distance séparant la position de voiture au bord de la piste

Si cette distance est grande **alors**

On choisit AccGauche ou AccDroite selon la grandeur des distances à gauche et à droite.

Sinon

On effectue un balayage d'angle : pour chaque angle, on calcule la distance la distance séparant la voiture du bord de la piste. A la fin du balayage, on choisit le meilleur angle (c'est-à-dire dont la distance correspondante est la plus grande).

Si on est proche d'un mur **alors**

On freine puis on tourne selon la meilleure direction.

Sinon

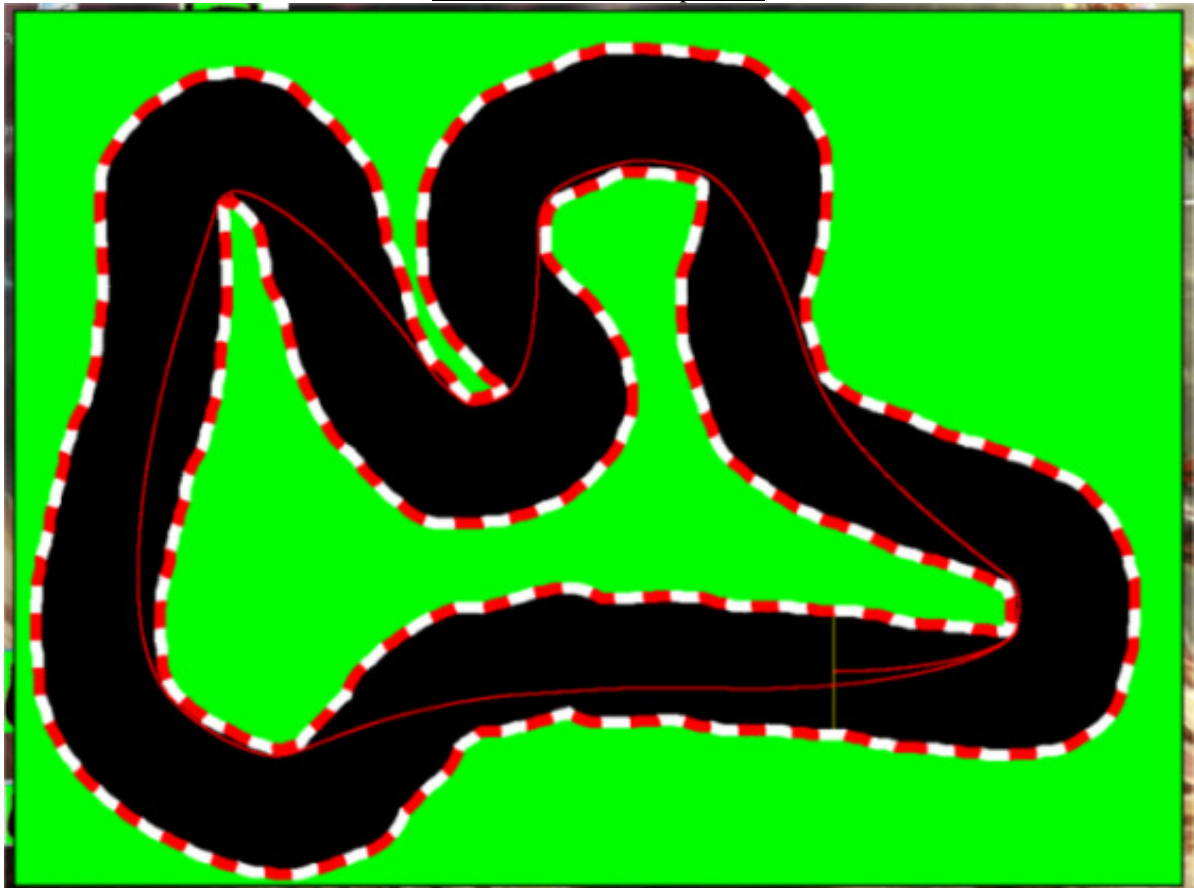
On tourne selon la meilleure direction sans accélérer.

Fin si

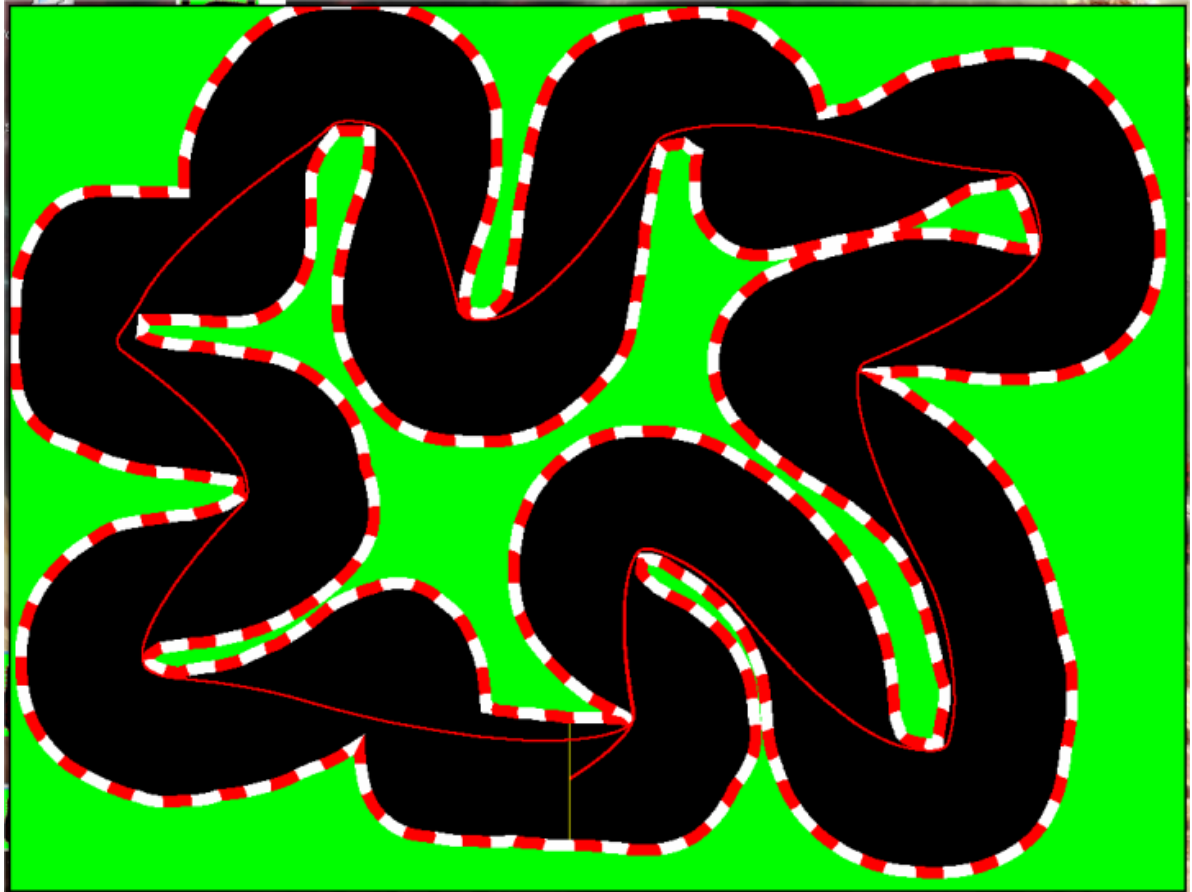
Fin si

Sa mise en œuvre sur les différents circuits donne les résultats suivants :

Circuit n°1 : 1 Simple.trk



Circuit n°2 : Hairpins.trk



Circuit n°3 : Sepang.trk



Circuit n° 4 : EatYouAlive.trk



Malgré ces quelques résultats concluants, on s'aperçoit que notre radar est vitre déboussolé face à des circuits plus complexes tels que le circuit n°6 - IcfpContest.trk :



En effet, la voiture circule sur le circuit sans jamais trouver la ligne d'arrivée qui est un peu retirée du circuit. Ceci est normal puisque quand la voiture est proche de la ligne d'arrivée, la radar effectue un balayage d'angle et trouve que la distance la plus grande n'est pas celle menant à la ligne d'arrivée mais décrite sur l'image.

Cette méthode présentée ici montre alors ses limites. Il nous faut donc mettre en place un nouveau moyen de rendre notre voiture autonome.

3°) Problèmes rencontrés

Durant l'implémentation de notre radar, nous nous sommes confrontés à de nombreux problèmes.

D'une part, lors des premiers tests du radar, la voiture ne bougeait pas, elle restait sur sa position de départ. Après plusieurs tests on a remarqué que pour le programme la voiture se trouvait sur du vert, alors qu'à l'écran elle se trouvait sur la route. On a alors que la fonction qui dessine la matrice terrain à l'écran faisait une sorte de transposition de la matrice. Pour accéder à la case (i,j) de notre matrice originale il fallait donc faire $\text{terrain}.(H-i).(j)$ où H est la hauteur de l'image représentant le terrain (moins -1 car on démarre à 0 pour la matrice).

D'autre part, une fois ce problème réglé, notre radar ne fonctionnait pas correctement dans le sens où il sortait souvent de la piste. Pour régler ce nouveau problème, on a du changer la fonction qui choisit l'action à effectuer à chaque appel du radar.

IV - Apprentissage par renforcement

1°) Principe de l'apprentissage

Le principe de l'apprentissage par renforcement est le suivant : un agent parcourt le circuit avec différentes actions et dès qu'il trouve un bon chemin il le note en valorisant par un score assez élevé (la case ayant le score maximum est la ligne d'arrivée du circuit appelé aussi trésor) les cases par lesquelles il est passé. Au contraire, il valorise par un faible score les obstacles et les chemins qui ne mènent nulle part. Ainsi au fur et à mesure des itérations, l'agent parcourt de façons multiples le circuit et a donc plus de chance de trouver un bon chemin vers l'arrivée.

2°) Mise en œuvre de l'algorithme (état d'avancement du projet)

A l'heure où nous écrivons ce texte, nous ne sommes pas encore parvenu à faire fonctionner correctement l'apprentissage par renforcement.

En effet, nous réfléchissons et expérimentons encore les manières d'implémenter en Ocaml le principe décrits dans la sous partie précédente.

Ceci dit, nous comptons rattraper ce petit retard pendant les vacances de Pâques qui approchent.

V – Nos idées pour la suite

1°) Ce que l'on compte faire

Dans cette partie, nous allons détailler quelques idées que nous comptons mettre en place par la suite pour notre projet :

Dans l'immédiat, nous comptons mettre en place l'algorithme de Dijkstra. Cet algorithme va nous être précieux, car il nous indique le chemin le plus court pour arriver à la ligne d'arrivée à partir de la position de la voiture.

Ensuite, nous tenterons d'améliorer les différents algorithmes d'apprentissages (renforcement, Dijkstra ...) afin de les optimiser et de les rendre plus performants.

Nous allons aussi coupler les différents algorithmes (radar / apprentissage par renforcement / Dijkstra) de telle sorte qu'on puisse avoir sur n'importe quel circuit le trajet le plus rapide. Pour cela nous créerons par exemple une fonction qui interrogera les trois algorithmes pour la position courante de la voiture et nous choisirons la meilleure action à effectuer selon le retour des algorithmes. Cette meilleure action sera choisie par exemple en donnant plus de poids à l'algorithme de Dijkstra qui semble être le plus performant mais en tenant compte tout de même de l'apprentissage par renforcement qui peut se révéler très utile et du radar mais dans une moindre mesure car il est moins performant sur les circuits complexes.

On voudrait aussi intégrer un dérapage dans ce modèle de voiture. On peut utiliser la variable pivot inutilisée dans laquelle on stocke le seuil de vitesse avant de déraper.

En effet, on pourrait tester lors de la mise à jours des caractéristiques de la voiture (dans la fonction calcNextPos), si la vitesse actuelle est supérieur ou égal au pivot et qu'on choisit de tourner (à gauche ou à droite) alors on modifie le comportement de la voiture dans la façon de la faire tourner, c'est-à-dire qu'on la fait dérapier.

Nous tenterons aussi de remplacer le point de la voiture qui s'affiche à l'écran par une image 2D de voiture afin de rendre plus réaliste notre simulation. Pour cela nous créerons une matrice de couleurs aux dimensions de l'image dans laquelle on chargera l'image représentant la voiture. Ainsi lorsque qu'on voudra tourner (à gauche où à droite) il faudra en faire une sorte de rotation de cette matrice afin de voir sur l'écran la voiture qui tourne.

2°) Ce que l'on pourrait faire ...

Dans cette partie, nous détaillerons quelques idées que l'on pourrait mettre en place, mais que malheureusement nous ne mettrons pas en œuvre par manque de temps ou par complexité :

On pourrait transformer notre simulateur 2D en un simulateur 3D à l'aide d'OpenGL. A première vue, cela ne semble pas compliqué, il suffirait de modifier la partie graphique de notre projet en rajoutant à chaque vecteur 2D une dimension supplémentaire qui serait la hauteur. Ceci dit, plusieurs problèmes se posent : quel hauteur mettre pour chaque type d'objet, d'obstacle ? Comment gérer les collisions avec le bord des pistes ? Etc. Répondre à ces questions en implémentant un modèle en 3D prendrait trop de temps et nous préférons nous concentrer sur les algorithmes d'apprentissages.