

# Rapport du projet

UE :2I013

XIANG Yangfan










# Introduction

UE de projet 2I013 est l'unique UE qui nous permet de réaliser un projet complet. Il nous apporte une approche plus concrète en informatique. Concentré sur la pratique et un peu de théorie, l'UE nous offre une bonne entraînement et une amélioration considérable au niveau de la programmation. C'est donc naturellement que j'ai choisi cette UE pour une mise en pratique et un approfondissement de ma capacité de programmation suite au UE d'initiation à la programmation Objet que j'ai suivi au premier semestre.

## I. Structure et organisation générale du code

Mon code est organisé et répartie par leur fonctionnalité dans le produit final. Cette manière la serai plus simple pour la recherche et une meilleur compréhension puisque la première renseignement la notation des classes et des packages. Il est en conséquent très important de donner des noms pertinent pour la clarté et la prise en main.

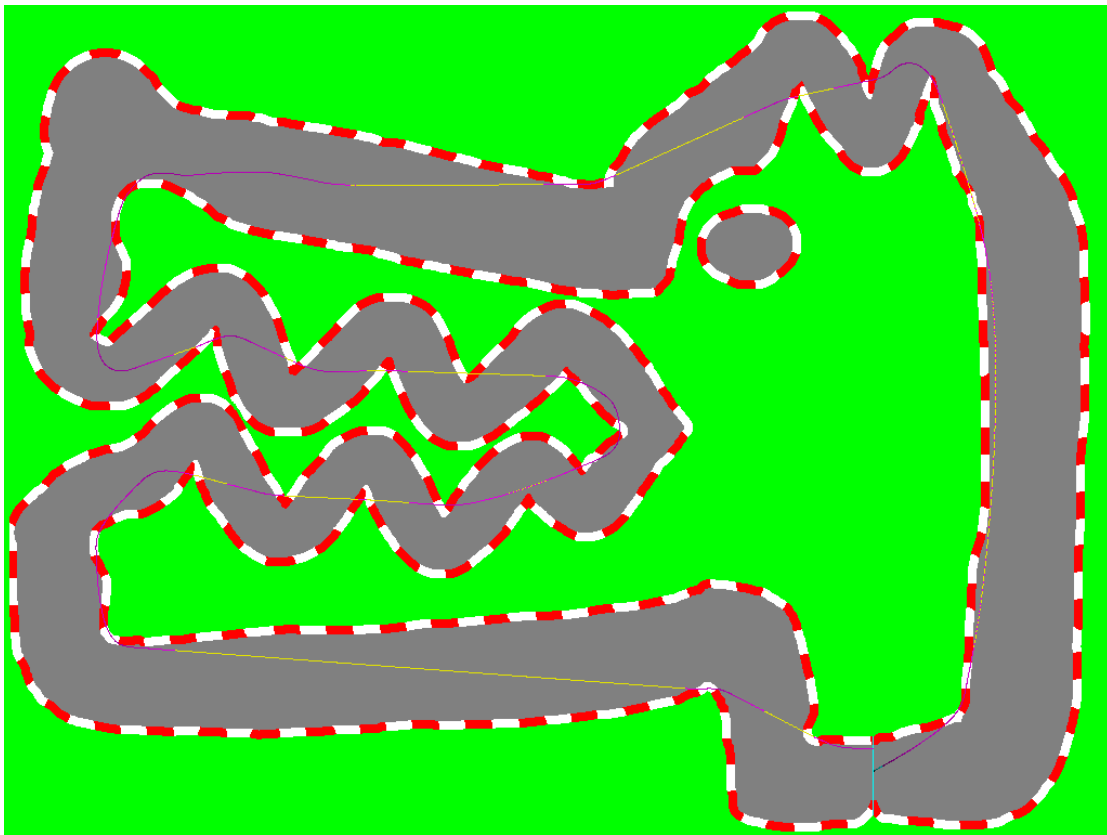
Voici donc mes packages :

-  voiture : model physique de la voiture
-  circuit : le model physique du circuit et du terrain avec les fonctions élémentaires
-  radar : le model et les différents radars développer
-  strategy : ensemble de stratégie développer
-  algorithmeGenetique: ensemble des classes implémentent l'algorithme génétique
-  algorithmeOutil : les classes static sur la manipulation des fichiers et les matrices ainsi que les algorithmes servant pour les autres classes.
-  controleur : les classes assurant la communication entre les classes dans l'interface graphique et aussi les effets des buttons.
-  view : implémentation de l'interface graphique et les observer des différents objets
-  mains : tous les produits finals développer

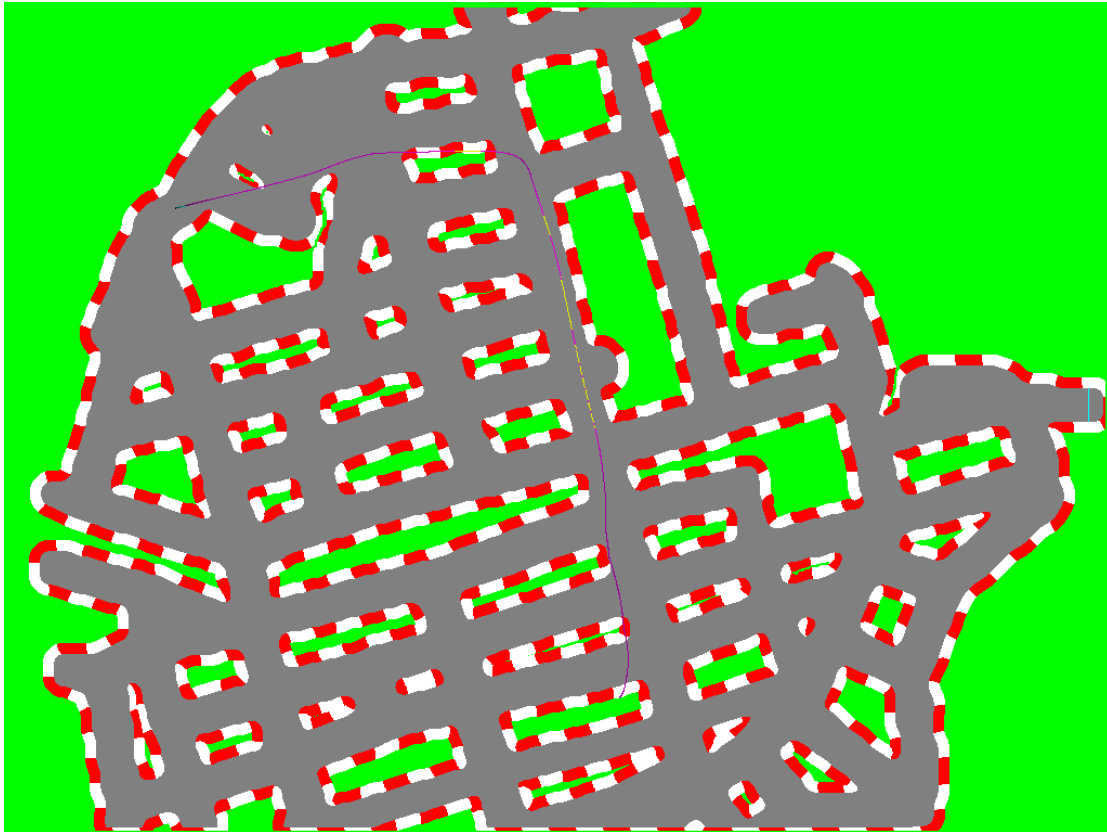
## II. Développement des Radars

### 1) Radar Classique

La logique du radar classique fait qu'il marche que sur les circuits unidirectionnels simples comme 1\_safe ou 2\_safe. On ne peut donc aller trop loin avec ceci sur les circuits plus complexes ni d'obtenir un score satisfaisant. Néanmoins il y a quelque difficulté lors de l'implémentation. Lorsque la voiture atteinte près de la ligne d'arriver, le radar classique identifie et considère la ligne d'arriver comme un mur. A ce moment là, la voiture se dirige vers un coin de la ligne d'arriver. Par chance il arrive a l'extrémité de la ligne ou se crash sur l'herbe. Pour résoudre, il faut donc ajouter une teste pour tester si le radar détecte la ligne d'arriver, de considérer la ligne d'arriver comme une distance qui se situe à l'infinie de la voiture et donc foncer vers la ligne. Pour améliorer un peu plus et de gagner une dizaine de coup, il faut détecter la direction où la distance entre la voiture et la ligne soit minimale.



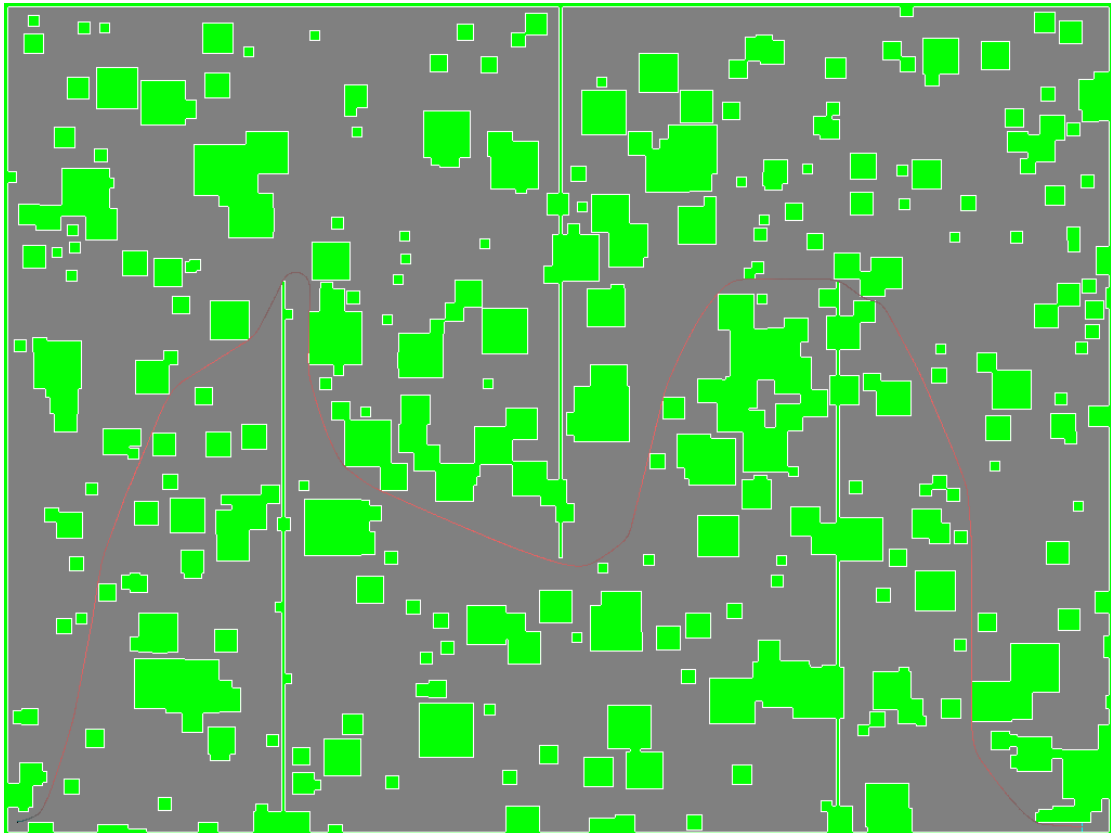
4\_safe avec Radar Classique



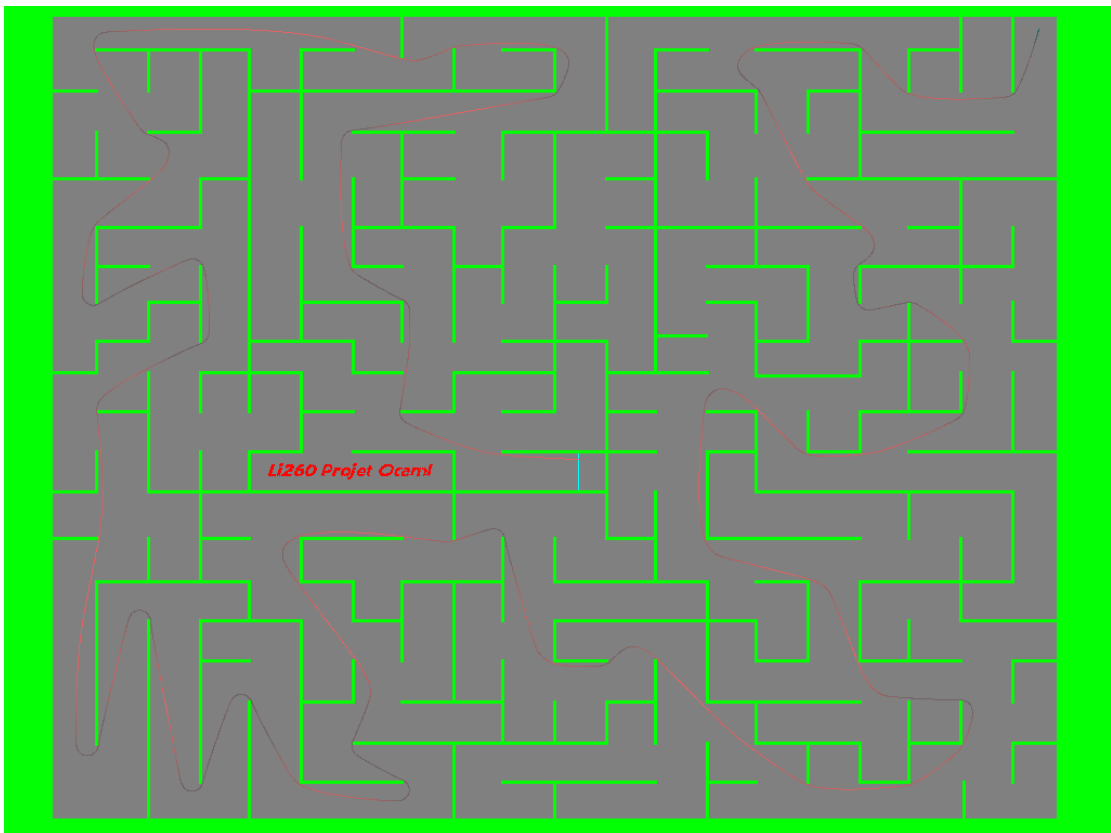
7\_safe avec Radar Classique

## 2) Radar Dijkstra

L'algorithme de dijkstra permet de connaître la distance tous les points de l'univers par rapport à un point. Dans notre cas, c'est la distance de tous les points du circuit par rapport à la ligne d'arrivée. L'implémentation n'étant pas très difficile mais il faut faire attention à utiliser un tas pour stocker les points de calcul à suivre. Il faut aussi à dérouler l'algorithme dans le sens de la ligne d'arrivée au départ. Ce qu'on fait concrètement est lors d'initialisation de l'algorithme de dijkstra, d'ajouter les points autour de la ligne d'arrivée ou sont dans le sens contraire de la sens d'arrivée. Ensuite le radar envoie des faisceaux et vérifie la distance des cases par rapport à la ligne d'arrivée pour déterminer quelle direction à prendre. Le problème de la ligne d'arrivée se pose aussi dans ce cas mais se résoud de la même façon.



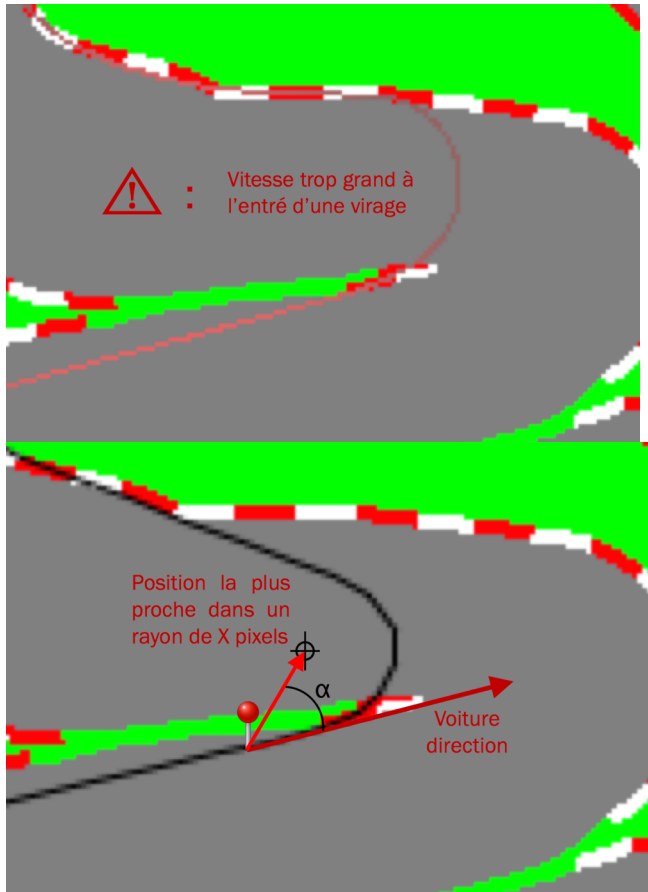
8\_safe avec Radar Dijkstra



labyperso avec Radar Dijkstra

### 3) Recherche personnel : Radar Prévoyance

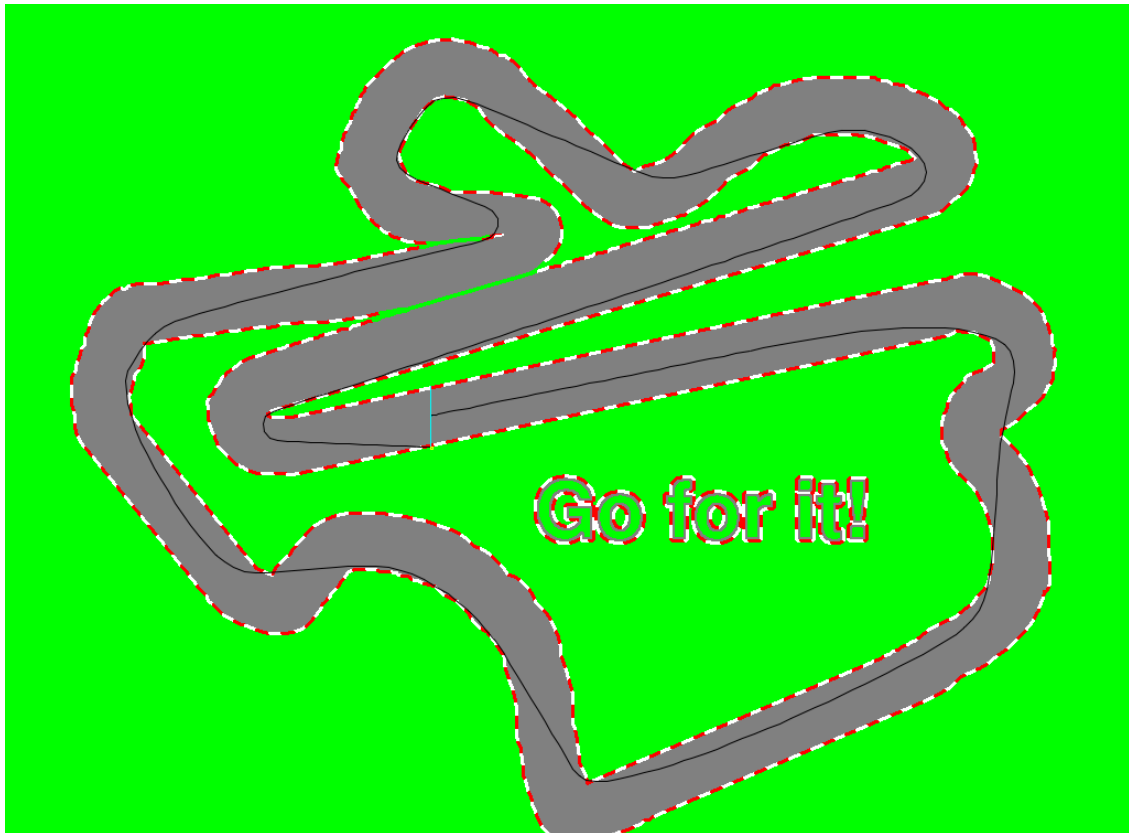
La détermination de la direction à prendre donc l'angle de tourner n'est pas le point de difficulté mais la gestion des accélérations nous pose des problèmes.



On voit ici que la vitesse d'entre dans le virage est trop grand, dans certain circuit la voiture lors de la virage prend directement l'herbe et se crash.

J'ai donc pensé à développer une projection en avant et déterminer X pixels après le point le plus proche de la ligne. Ensuite de déterminer l'angle entre la position de la voiture et le point calculer et la direction de la direction de la voiture. On peut donc maintenant savoir l'angle de virage et déterminer en fonction de cet angle la commande d'accélération à adopter.

Ainsi on peut minimiser le courbure de la trajectoire mais en gardant une maximum de vitesse tout en passant le virage. Cela permet donc de gagner plusieurs dizaines de coup et de passer les virages impassable au par avan



3\_safe avec Dijkstra et Prevoyance



t260\_safe avec Dijkstra et Prevoyance

### III. Stratégies

#### 1) Modèle des stratégies

Si les radars ont pour fonction de donner des informations sur la course, alors les stratégies ont pour rôle d'interpréter et d'analyser les données reçues pour au final retourner une commande la plus optimale possible.

J'ai donc mis en place 3 stratégies principales qui exploitent au mieux les données :

-stratégie à 5 paramètres : qui ne fait qu'une analyse minimum sur l'état de la voiture et les informations du radar, elle est donc pas très performant

```
if( Math.abs(thetas[radar.getBestIndex()]/voiture.getMaxTurn() > 1){  
    if(thetas[radar.getBestIndex()] < 0){  
        turn = -voiture.getMaxTurn();  
    }  
    else {  
        turn = voiture.getMaxTurn();  
    }  
}  
else{  
    turn = thetas[radar.getBestIndex()]/voiture.getMaxTurn() * voiture.getMaxTurn();  
}  
  
if(distPix[distPix.length/2] < dist_secu){  
    acc = acc_secu;  
}  
if(distPix[radar.getBestIndex()] < 1.5*dist_secu){  
    acc = acc_secu;  
}  
if(fact_angle_secu * voiture.getMaxTurn() < Math.abs(thetas[radar.getBestIndex()]/voiture.getBraquage())){  
    acc = acc_virage;  
}  
if(voiture.getVitesse() < vit_lim){  
    acc = 1;  
}  
  
return new Commande(acc, turn);
```



-stratégie à 8 paramètres : on peut dire qu'elle exploite au maximum et donc les courses se sont jouées avec les scores plutôt satisfaisant mais il reste de la marge.

```
radar.scores();
double turn = radar.thetas()[radar.getBestIndex()]/voiture.getBraquage();
double turnAbs = Math.min(Math.abs(turn), voiture.getMaxTurn());

double acc = 1; // accélération par défaut
if(RadarClassique.distInPixels(0, voiture, circuit)<param[0]){
    acc = param[1]; //acc secu
}
else if(RadarClassique.distInPixels(radar.thetas()[radar.getBestIndex()], voiture, circuit) < param[0]*0.5){
    acc = param[1]; //acc secu
}
else if(param[2]*turnAbs < Math.abs(turn)){ //fac angle secu
    acc = param[3]; //acc virage
}
else if(param[4]*turnAbs < Math.abs(turn)){ //fact angle secu
    acc = param[5]; //acc virage
}

if(voiture.getVitesse() < param[6]){ //vit lim max
    acc = 0.5;
}
if(voiture.getVitesse() < param[7]){ //vit lim max
    acc = 1;
}

return new Commande(acc, turnAbs * Math.signum(turn));
```

-stratégie à 9 paramètres celui de la prévoyance : le 9eme paramètre est celui de l'angle alpha de la virage situant devant la voiture, grâce à cela nous pouvons contrôler au mieux l'accélération de la voiture

```
prevoyance.setPrevdist(param[8].intValue()); //la distance a voire

radar.scores();
double turn = radar.thetas()[radar.getBestIndex()]/voiture.getBraquage();
double turnAbs = Math.min(Math.abs(turn), voiture.getMaxTurn());

double acc = 1; // accélération par défaut
if(RadarClassique.distInPixels(0, voiture, circuit)<param[0]){
    acc = param[1]; //acc secu
}
if(RadarClassique.distInPixels(radar.thetas()[radar.getBestIndex()], voiture, circuit) < param[0]*0.5){
    acc = param[1]; //acc secu
}
if(param[2]*turnAbs < Math.abs(turn)){ //fac angle secu
    acc = param[3]; //acc virage
}
if(param[4]*turnAbs < Math.abs(turn)){ //fact angle secu
    acc = param[5]; //acc virage
}

if(prevoyance.getAngle() > param[7]){ //angle de virage devant la voiture
    acc = -1;
}

if(voiture.getVitesse() < param[6]){ //vit lim max
    acc = 1;
}

return new Commande(acc, turnAbs * Math.signum(turn));
```

## 2) Optimisation des stratégies : algorithme génétique

Dans notre cas, nous sommes amenés à donner les paramètres nécessaires pour évaluer les données et déterminer la commande de la voiture. Il n'est pas imaginable de tester les paramètres manuellement car il y a une infinité. C'est pourquoi entre en jeu l'algorithme génétique qui fait le croisement entre les meilleurs paramètres et de génération en génération on s'améliore et se rapprochent des paramètres idéals.

J'ai remarqué tout de suite après quelque essai de cet algorithme que si nous voulons repartir avec les paramètres existants, il faut aller directement dans le code et modifier les paramètres par défauts. C'est une opération très lourde et peu efficace. Une solution que j'ai apportée est à chaque début de l'algorithme, on cherche dans un fichier qui contient les paramètres pour chaque circuit et des lires. Si un circuit n'a pas de paramètres déjà existants alors on appliquera les paramètres par défaut. La terminaison de l'algorithme s'accompagne d'une écriture des paramètres optimales dans un fichier identifié aussi par les circuits.

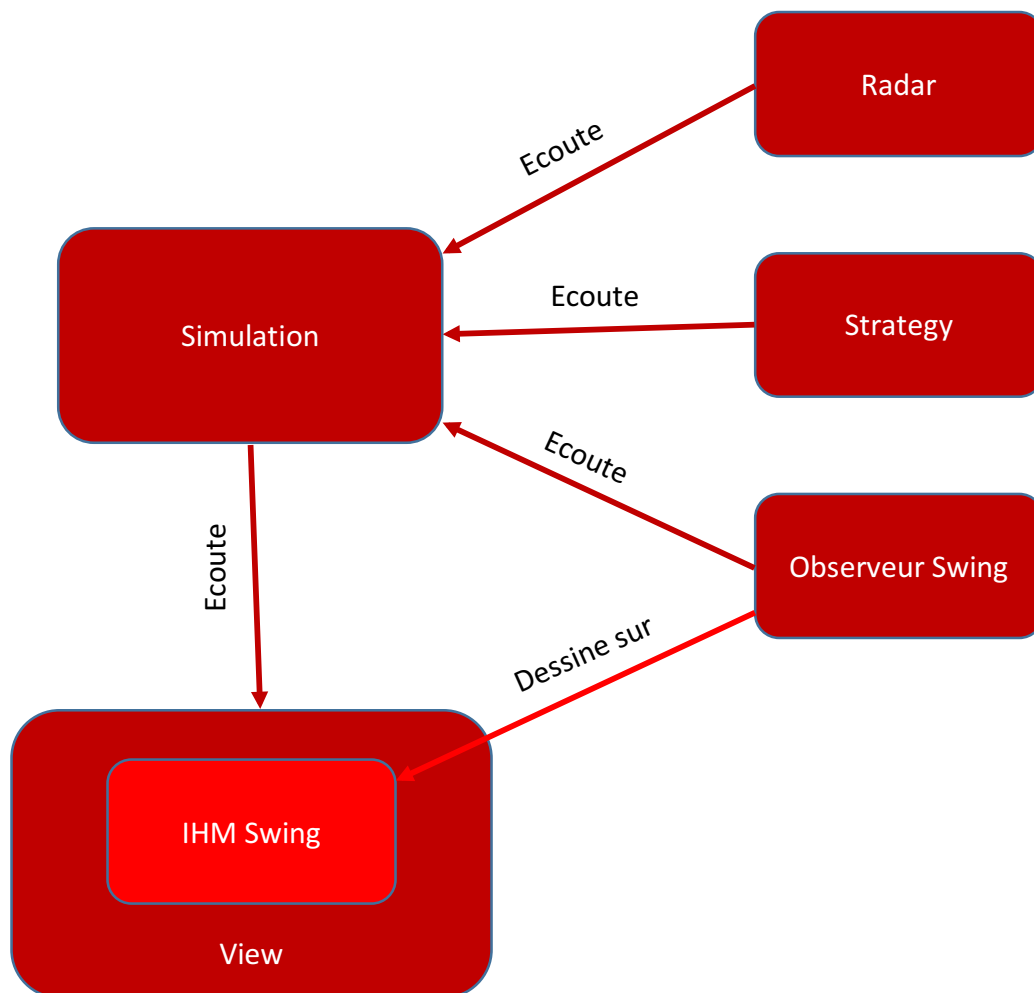
```
GenomeGenerator<Double> genomeGn =  
    new GenomeGeneratorImplPrevoyance(FileTools.loadParam(savePlace+nomCircuit+saveMode));
```

```
Double[] param;  
if(paramInput != null){  
    param = paramInput;  
}  
else {  
    param = defaultVal;  
}
```

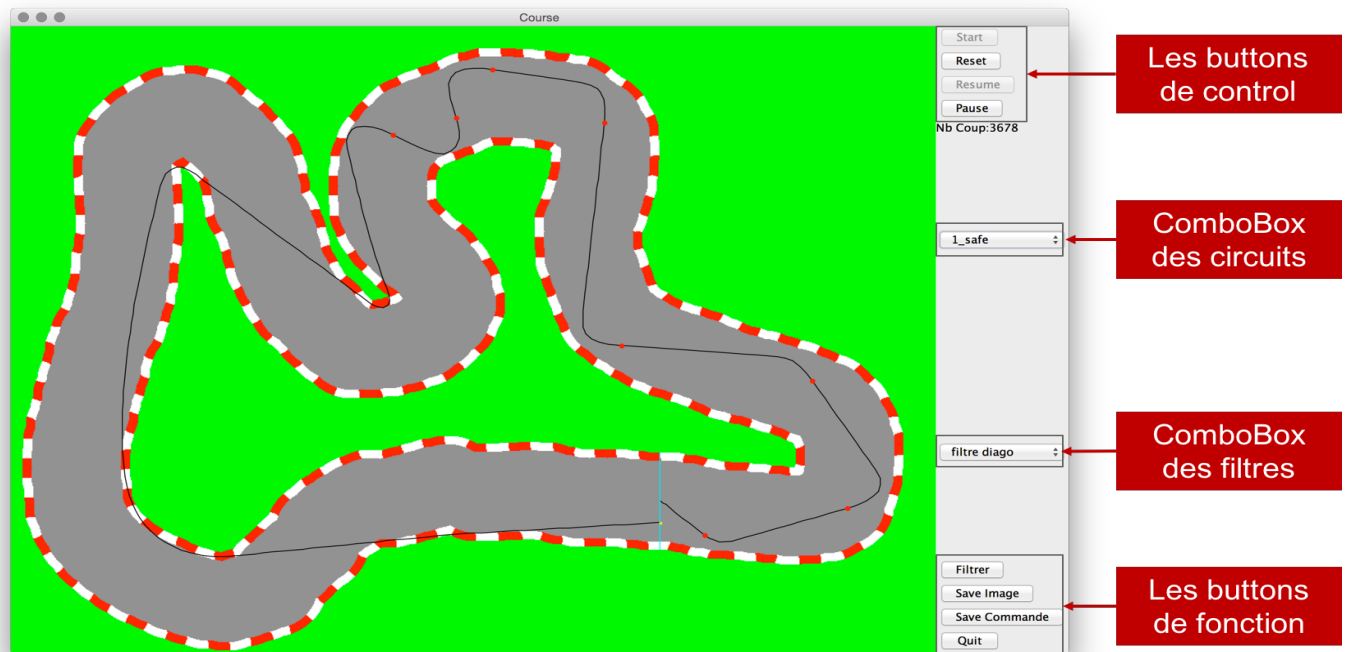
```
FileTools.saveParam(genomeOptimal, savePlace+nomCircuit+saveMode);
```

### III. L'interface graphique

La difficulté de l'interface graphique est la communication entre les composants. Il faut a fois que les objets soient suffisamment modulaire pour que dans le main on doit simplement les assembler, de faire communiquer les objets indépendants les un des autres n'est pas une tache facile. J'ai utilisé pas mal de *listener* et de *sender* de messages/signaux pour qu'une action dans un objet a un impacte sur un autre. Voici un simple schéma de description des liaisons entre les objets que j'ai développés :



Le resultat obtenue ressemble à ça:



Mon interface graphique est composé de deux parties, une partie d'affichage du course et une partie composé des options mise en place pour avoir quelque controle sur la course. Dès le debut je rencontre la difficulter de faire une pause de la course et ensuite de reprendre. Le première approche est d'agire directement dans la simulation mais une fois arreter et repris, la simulation ne peut plus à nouveau s'arreter car on est toujours dans la boucle de simulation. Après avoir fait quelque recherche sur l'internet, la solution est de mettre a chaque fois la simulation sur un nouvel file d'execution et d'arreter cette file pour fait pause et de relancer la simulation sur un nouvel file pour reprendre.

Un comboBox qui permet de passé un circuit à un autre facilite l'echange entre l'homme et la machine. Il faut lors de changement du circuit faire la mise à jour du circuit pour tout les classes dépendant de circuit, c'est une mise à jour en casclade pour assurer le bon fonctionnement du programme.

On peut encore choisir des filtres et de cliquer sur Filtrer pour appliqué sur la matrice representant le circuit pour par exemple dans le circuit bond\_safe eliminer le raccourcis entre le cravate ou de licher les bornes comme dans le circuit 2\_safe pour les points sortants qui risque de faire écrasé la voiture.

A partir d'une interface graphique on peut enplus realiser une strategie point à point qu'on peut cliquer directement sur le circuit pour forcer la voiture à passé sur ces points. Cette fonction est mise en oeuvre par une strategie composite qui reuni le strategie dijkstra/prevoyance et la strategie point à point.

## Conclusion

Cette UE de projet donne l'occasion de savoir qu'est ce que la programmation objet avec ses qualités. Une travaille plus libre et l'intependance face à des problèmes nous apprenant comment chercher et où chercher pour au final proposé une solution adapté. C'est un vrai experience de rencontré les problèmes courants dans le développement de logiciel et de résoudre à notre tour. L'UE de projet est l'ue incontournable pour l'acquisition des competances informatique.

Important!!!!

- Tous les circuits doivent se trouver dans le fichier CarRace l'emplacement courant du projet pour l'eclipse.
- pour les mains d'algorithme génétique on choisi le circuit à évaluer avec *nomCircuit* en static, pas besoin de mettre l'extention juste le nom du circuit suffit
- pour un circuit, l'algorithme génétique lit les paramètres en fichier, en absence de fichier correspondant il appliquera les paramètres par default.
- pour tous les autres mainSWING il suffit de l'exécuter
- en changeant de circuit, le chargement des paramètres pour le circuit choisi s'effectue automatiquement.
- j'ai changé l'initialisation de l'algorithme génétique pour pouvoir garder les paramètres lu.
- main image et main SWING(radar classique) sont deux classes servant au depart comme des classes de teste de validité des fonctions. On peut maintenant l'ignorer si on veut.
- il peut que pour le composite et point a point, la voiture tourne en rond autour d un point, c'est à cause de la vitesse minimum de la voiture
- pour il faut aller dans fitness pour changer le radar à optimiser par default cest en dijkstra