Ecole Publique d'Ingénieurs en 3 ans

Enseignants encadrants : Mr Ziad-Forrest et Mr Fourney

Rapport

CHALLENGE DE PROGRAMMATION

le 10 avril 2017,

Nom d'équipe : La Picole Nationale

Clément JANTET,
clement.jantet@ecole.ensicaen.fr

Calliste RAVIX,
calliste.ravix@ecole.ensicaen.fr

www.ensicaen.fr

TABLE DES MATIERES

TABLE DES FIGURES

Figure 1 : logo de notre écurie

1. INTRODUCTION

Le challenge proposé consistait à programmer un **pilote de formule 1** évoluant sur un circuit virtuel. Le pilotage se fait en communiquant avec un gestionnaire de course qui fournit les informations sur la course et les autres pilotes. Le but est d'atteindre l'arrivée le plus rapidement possible tout en gérant la consommation de carburant.

Dans ce rapport, nous allons présenter notre approche algorithmique pour résoudre ce problème ainsi que les particularités de notre pilote. Nous allons également décrire les difficultés que nous avons rencontrées et les solutions que nous avons apportées. Enfin, nous allons discuter de ce que nous avons appris pendant ce projet.

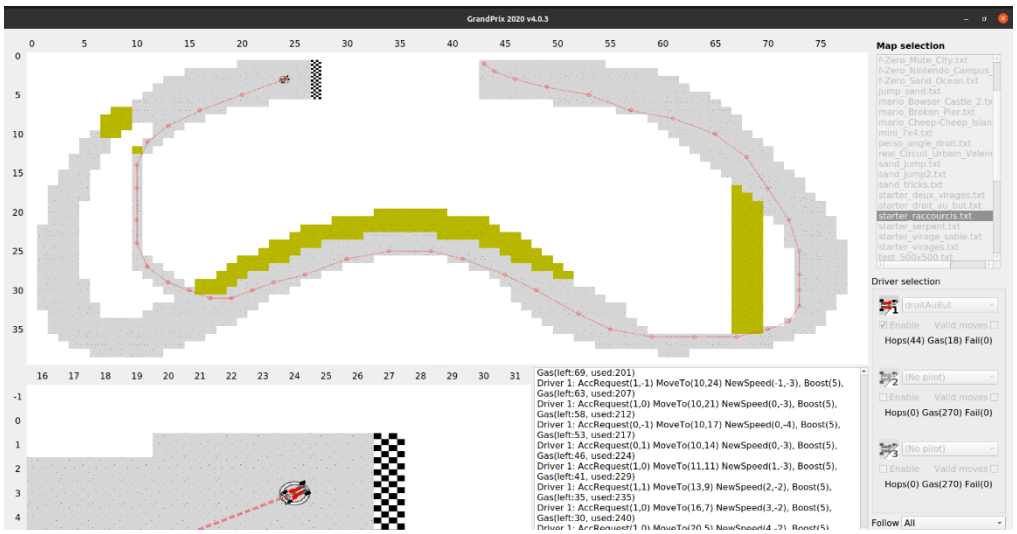

Figure 2 : notre pilote évoluant sur la carte starter_raccourcis

2. OUTILS

Pour ce projet, nous avons utilisé le **langage C** compilable sous **Linux Ubuntu 20.04 LTS**. Nous avons utilisé **Gitlab** ainsi que **VisualCode Live Share** pour la gestion de version et la collaboration. Nous avons travaillé en équipe de deux personnes, chacun étant responsable d'une partie du code. Clément s'est concentré sur **l'algorithme A*** et la fonction **findBestEnd** et Calliste s'est occupé de toutes les **fonctions annexes** (Même si en réalité, tout le monde a participé à toutes les fonctions).

3. OBJECTIFS

L'objectif principal était de programmer un pilote de formule 1 qui atteignait l'arrivée le plus rapidement possible tout en respectant les règles de la course, telles que la vitesse maximale, la consommation de carburant et les limites d'accélération. Pour atteindre cet objectif, nous avons établi les tâches suivantes :

- Analyse des règles de la course et des **entrées et sorties standard du GDC**.
- Conception d'un **algorithme de mouvement** pour le pilote en tenant compte des limites d'accélération et de vitesse.
- Implémentation de l'algorithme de mouvement en langage C.
- Test du pilote sur différents circuits pour **mesurer son temps d'arrivée** et sa **consommation de carburant**.
- Optimisation du code pour **améliorer les performances** du pilote.

4. CONCEPTION

4.1. IDÉE GÉNÉRALE

Nous avons commencé par **analyser les règles de la course** et les **entrées et sorties standard du GDC**. Nous avons compris que chaque pilote devait communiquer avec le GDC via ses entrées et sorties standard en mode texte. Le GDC envoyait la composition du terrain, la position des autres pilotes, et attendait que chaque pilote donne son vecteur d'accélération.

Nous nous sommes ensuite intéressés aux **déplacements du pilote**. La planification de la trajectoire consiste à déterminer la direction à prendre pour atteindre l'arrivée le plus

rapidement possible tout en évitant les collisions avec les autres voitures et les obstacles sur la piste.

Nous avons utilisé un algorithme de recherche de chemin classique appelé **A*** pour résoudre ce problème. L'algorithme A* fonctionne en évaluant les coûts de déplacement de chaque case de la grille et en choisissant la case qui a le coût total le plus faible. Nous avons utilisé une fonction **d'évaluation heuristique** pour estimer le coût restant pour atteindre l'arrivée. Cette fonction prend en compte une combinaison de la **distance euclidienne** et de la **distance de manhattan** entre la position actuelle et l'arrivée ainsi que les obstacles sur la piste.

Dans un premier temps, nous avons implémenté l'algorithme A* en le limitant à :

- Prendre uniquement **1 point d'arrivée**.
- Ne pas pouvoir aller dans le **sable**.
- Ne pas prendre en compte les **collisions**.
- Une **vitesse maximum de 1** pour vérifier que le trajet est optimal.
- Ne pas utiliser de **boosts**.

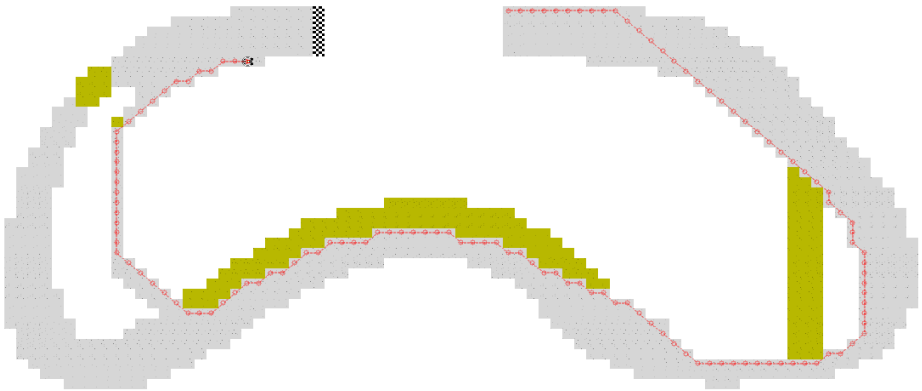

Figure 3 : Notre ancien pilote limité

Le but étant bien sûr **d'enlever ces limites** par la suite pour avoir le pilote le plus rapide.

Nous avons ensuite ajouté une fonction de **gestion du sable**. Pour gérer ce problème, nous avons mis un coût arbitraire au sable et à la route pour qu'il ne prenne le sable que lorsque c'est vraiment plus rapide.

Nous avons aussi créé une fonction de **détection de collision** pour éviter les collisions avec les autres voitures et les obstacles. Cette fonction vérifie si la trajectoire planifiée rencontre un obstacle et ajuste la trajectoire en conséquence.

Nous avons utilisé une fonction de calcul de **consommation de carburant** qui prend en compte la vitesse actuelle, l'accélération demandée et le type de terrain sur lequel la voiture se trouve. Nous avons également ajouté une fonction de planification de la consommation de carburant qui calcule la quantité de carburant nécessaire pour chaque tour de la course et ajuste l'accélération en conséquence.

4.2. DESCRIPTION DÉTAILLÉE DE L'ALGORITHME A*

Le principe de base de **l'algorithme A*** est de combiner les informations sur le coût du chemin déjà parcouru (g) et une estimation du coût restant pour atteindre la destination (h), pour évaluer les différents nœuds du graphe. L'algorithme calcule une valeur de coût total f pour chaque nœud en utilisant la formule **f = g + h**.

L'algorithme explore les nœuds du graphe en suivant le nœud **ayant la plus petite valeur f**, en prenant en compte les coûts du chemin déjà parcouru et l'estimation du coût restant. Ainsi, il cherche à atteindre la destination en suivant le **chemin qui minimise le coût total f**.

L'heuristique utilisée dans l'algorithme A* est une estimation du coût restant pour atteindre la destination. Elle doit être admissible, c'est-à-dire qu'elle doit toujours **sous-estimer le coût réel** pour atteindre la destination. Si l'heuristique est admissible, **l'algorithme A* est garanti de trouver le chemin le plus court**.

Plus concrètement, il faut implémenter deux listes : une **liste ouverte** et une **liste fermée**. Ce sont des structures de données utilisées pour gérer les nœuds visités lors de la recherche du chemin le plus court entre un nœud initial et un nœud cible dans un graphe.

La **liste ouverte** contient les nœuds qui ont été découverts, mais qui n'ont pas encore été explorés. Ces nœuds sont généralement ceux qui sont accessibles depuis le nœud actuel et qui n'ont pas encore été ajoutés à la liste fermée. La liste ouverte est habituellement implémentée sous la forme d'une file de priorité qui trie les nœuds en fonction de leur coût estimé pour atteindre le nœud cible. Le nœud ayant le coût le plus faible est placé en haut de la file de priorité, ce qui permet de l'explorer en premier.

La **liste fermée**, quant à elle, contient les nœuds qui ont déjà été explorés. Les nœuds présents dans cette liste sont ceux pour lesquels on a déjà calculé le coût réel pour les atteindre. Cela permet d'éviter d'explorer plusieurs fois les mêmes nœuds et de boucler indéfiniment. En général, la liste fermée est mise à jour chaque fois qu'un nœud est déplacé de la liste ouverte vers la liste fermée.

L'algorithme A* procède de la manière suivante : il extrait le nœud ayant le **coût le plus faible de la liste ouverte**, l'ajoute à la **liste fermée**, puis explore les voisins de ce nœud pour trouver le prochain nœud à explorer. Pour chaque voisin, on calcule le coût total du chemin depuis le nœud initial jusqu'à ce voisin en utilisant une fonction d'évaluation qui combine le coût réel déjà parcouru et une estimation du coût de l'heuristique pour atteindre le nœud cible. Si le voisin est déjà dans la liste ouverte ou fermée, on compare les coûts et on met à jour le nœud si le nouveau chemin est plus court. Si le voisin n'est ni dans la liste ouverte ni dans la liste fermée, on l'ajoute à la liste ouverte.

Ce processus se répète jusqu'à ce que le nœud cible soit atteint ou que la liste ouverte soit vide (ce qui signifie qu'il n'y a pas de chemin possible). Une fois que le nœud cible est atteint (l'arrivée), le chemin le plus court peut être reconstruit en remontant depuis le nœud cible jusqu'au nœud initial en utilisant les informations des parents stockées dans chaque nœud. (cf. figure 4)

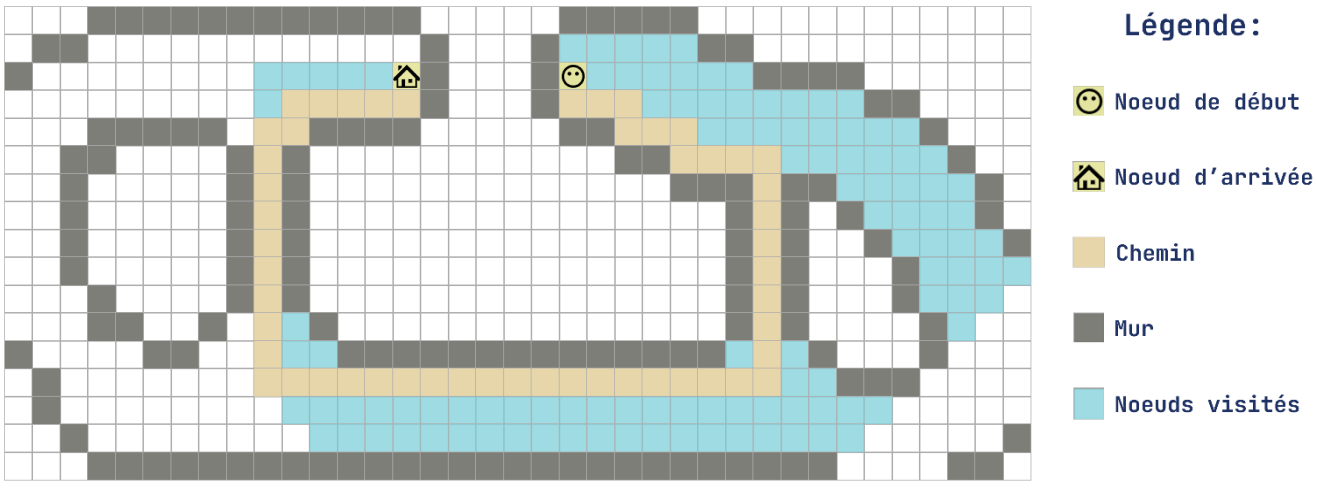

Figure 4 : algorithme a sur une simplification de starter_raccourcis avec l'outil de Jason Feng*

5. PARTICULARITÉS DE NOTRE PILOTE

Notre pilote a plusieurs particularités qui le distinguent des autres pilotes :

- **Utilisation de l'algorithme A*** pour la planification de la trajectoire, donc notre algorithme trouve le chemin le plus court en visitant le minimum de nœuds. L'idée est de calculer A* à chaque tour de jeu pour être à une **vitesse maximale**. On laisse également la possibilité d'utiliser des **boosts** soit pour réaccélerer, ou pour ralentir brusquement pour un virage serré. (cf. figure 5)

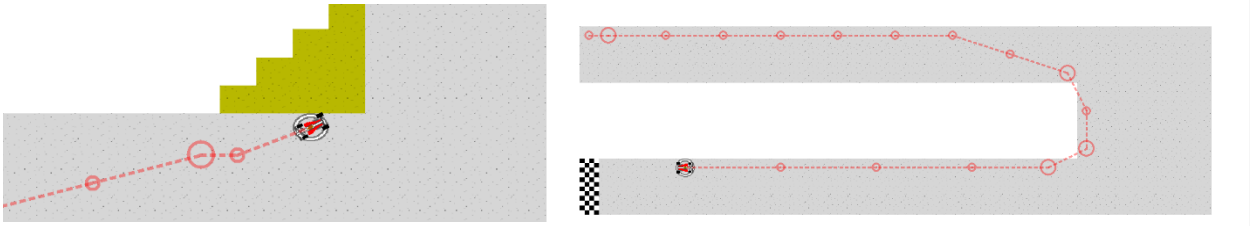

Figure 5 : Utilisation des boosts

- Utilisation d'une fonction de **détection de collision** pour éviter les collisions avec les autres voitures et les obstacles. On regarde si la case sur laquelle on souhaite se déplacer est disponible, sinon on recalcule le A*. On ne regarde pas plus loin pour éviter que notre pilote soit bloqué dans les couloirs de 1 de large. (cf. figure 6)

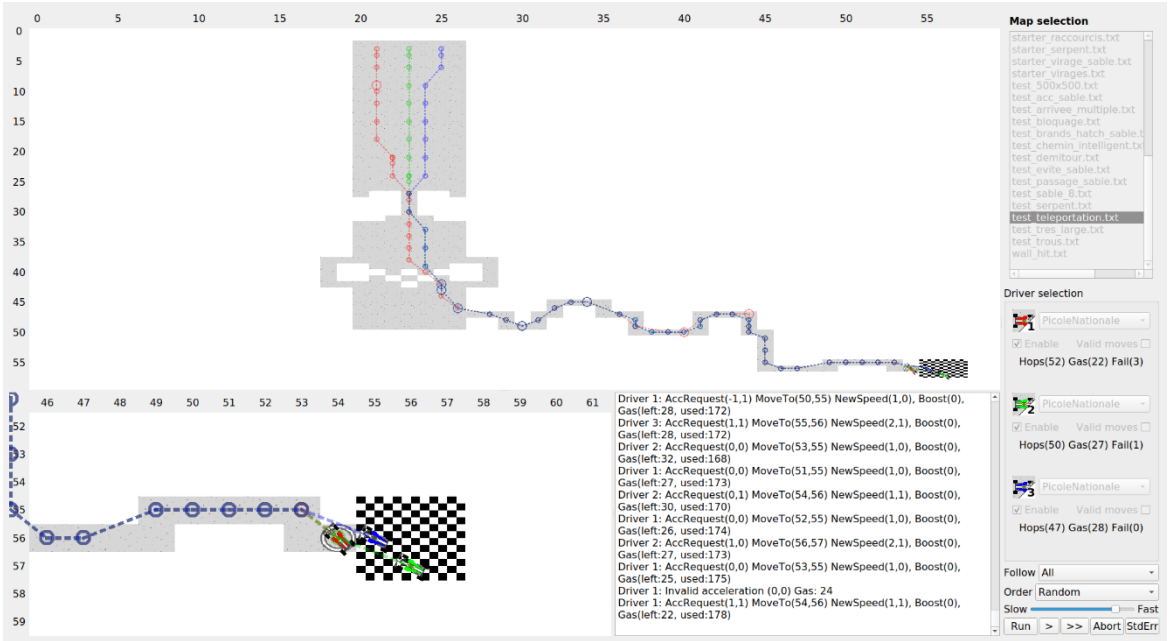

Figure 6 : Gestion des collisions

- Utilisation d'une fonction de **calcul de consommation de carburant** pour gérer la consommation de carburant (déjà fournie par les enseignants).
- Utilisation d'une fonction de **planification de la consommation de carburant** pour ajuster l'accélération en conséquence. On calcule le carburant nécessaire pour atteindre le nœud d'après dans la recherche de chemin. Finalement, on autorise les déplacements

6. DIFFICULTÉS RENCONTRÉES

6.1. LES DIFFICULTÉS

Au cours de ce projet, nous avons affronté de multiples obstacles et complications. L'un des défis majeurs que nous avons dû relever était de trouver un **équilibre adéquat entre la vitesse de notre pilote et sa consommation de carburant**. C'était un dilemme délicat à résoudre, car il était crucial de maintenir une vitesse élevée afin de terminer la course dans le temps le plus court possible, tout en veillant à ne pas épuiser le carburant de notre voiture avant d'atteindre la ligne d'arrivée.

Ensuite, l'optimisation de notre code pour assurer une performance adéquate dans le délai imparti a également été un obstacle non négligeable. Notre pilote était contraint à un **temps de réponse maximum d'une seconde**, ce qui signifiait que notre code devait être suffisamment efficace pour planifier la trajectoire et ajuster l'accélération de la voiture en moins d'une seconde. Nous avions une version de notre pilote qui parvenait à terminer tous les circuits, cependant, il mettait parfois plusieurs tours pour déterminer le chemin optimal sur certains d'entre eux.

Enfin, nous avons été confrontés à des problèmes liés à la **détection de collision**. Il était essentiel d'éviter les collisions avec les autres voitures et les obstacles sans compromettre la vitesse de notre véhicule afin de terminer la course dans les meilleurs délais.

6.2. SOLUTIONS APPORTÉES

Pour résoudre les difficultés que nous avons rencontrées, nous avons utilisé plusieurs stratégies :

Pour résoudre le dilemme entre vitesse et consommation de carburant, nous avons ajusté notre **fonction d'évaluation heuristique** pour prendre en compte ces deux paramètres. Nous avons utilisé des poids pour chaque critère dans la fonction d'évaluation, ce qui nous a permis de contrôler l'importance relative de chaque critère.

Pour améliorer l'efficacité de notre code et assurer son exécution dans le délai imparti, nous avons optimisé notre structure de données et éliminé les boucles inutiles. À l'origine, nous utilisions des structures de listes chaînées, mais nous avons fait la transition vers une **table de hachage** pour la **liste fermée** et **une file de priorité** pour la **liste ouverte**. Grâce à ces modifications, notre implémentation de l'algorithme A* a gagné en rapidité et en efficacité.

En ce qui concerne la **détection des collisions**, nous avons renforcé notre fonction de détection en intégrant des vérifications supplémentaires pour détecter les collisions avec les autres voitures et les obstacles. Nous avons modifié notre algorithme de pathfinding pour éviter les trajectoires qui menaient à un noeud occupé par une autre voiture. De plus, en cas de collision imprévue avec une voiture concurrente, nous avons mis en place un mécanisme de "fail" qui redémarre immédiatement notre voiture. Ainsi, même en cas de malentendu ou de comportement imprévu de la part des autres pilotes, notre voiture peut continuer la course sans perdre trop de temps.

7. RÉSULTATS ET OPTIMISATION

Les résultats ont montré que le pilote **atteignait l'arrivée** en un **temps raisonnable** et respectait **les limites de vitesse et de consommation de carburant**. Cependant, nous avons remarqué que le pilote pouvait parfois être trop prudent, ce qui le ralentissait. Nous avons donc optimisé le code pour améliorer les performances du pilote en utilisant des méthodes de recherche plus avancées pour déterminer le meilleur vecteur d'accélération.

8. PISTES D'AMÉLIORATIONS

Bien que notre pilote remplisse maintenant toutes les cases du cahier des charges, on remarque que les trajectoires ne sont pas tout le temps les plus optimales.

Pour régler ce soucis, nous avions pensé à **changer notre algorithme principal**. La première idée est d'améliorer notre A* avec, par exemple **l'algorithme JPS** (jump point search), qui serait plus optimisé. La seconde est de régresser en termes de complexité en changeant pour **Dijkstra**, mais **on explorerait tous les nœuds** donc on trouverait sûrement un meilleur chemin. (cf. figure 7)

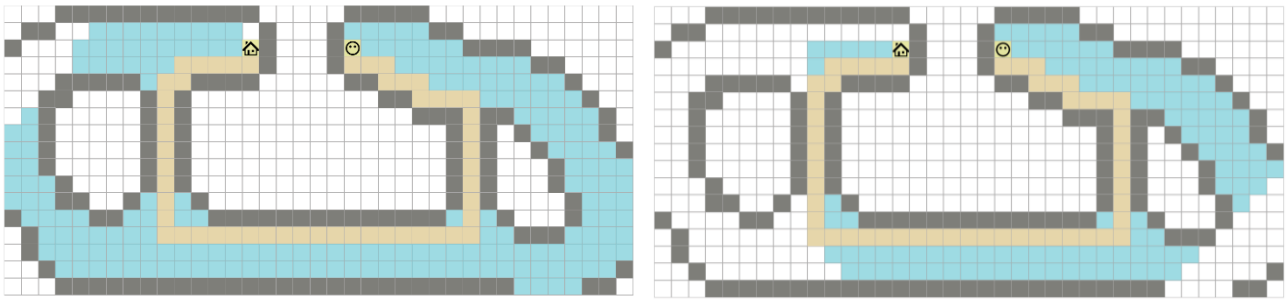

Figure 7 : Comparaison Dijkstra (à gauche) et A (à droite)*

On pourrait également jouer avec les **limites du gestionnaire de course** en faisant des 'fails' pour repartir directement dans les virages serrés. Cela serait plus optimisé que de ralentir avant le virage puis parcourir plusieurs nœuds dans ce dernier. Cependant, ce n'est pas vraiment l'esprit de la course, donc nous avons préféré ne pas nous concentrer sur cette amélioration.

9. CONCLUSION

En conclusion, ce projet nous a permis de mettre en pratique nos connaissances en algorithmique et en programmation. Nous avons réussi à résoudre le problème du pilotage d'une formule 1 sur un circuit virtuel en utilisant des algorithmes de recherche de chemin et en gérant la consommation de carburant. Nous avons aussi appris à optimiser notre code pour répondre aux contraintes de temps et à détecter les collisions avec les autres voitures et les obstacles.

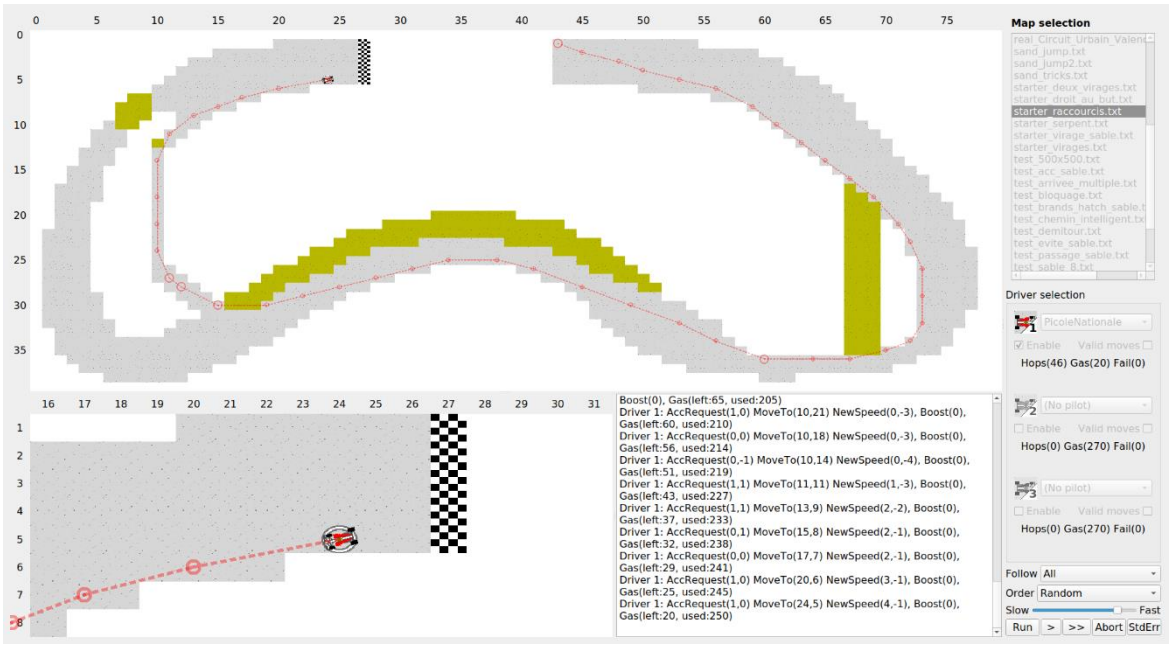

Figure 8 : Notre voiture finale évoluant sur le circuit starter_raccourcis

ENSI CAEN

ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Ecole Publique d'Ingénieurs en 3 ans

6 boulevard Maréchal Juin, CS 45053
14050 CAEN cedex 04

ministère éducation nationale enseignement supérieur recherche

Normandie Université

Cti Commission des titres d'ingénieur

cdefi Conférence des Directeurs des Écoles Françaises d'Ingénieurs

European Accreditation of Engineering Programmes EUR-ACE®

CONFÉRENCE DES GRANDES ÉCOLES