ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Rapport du Projet Grand Prix F1

KHALI Omar
omar.khali@ecole.ensicaen.fr

OUAZZANI CHAHDI Hamza
hamza.ouazzani-chahdi@ecole.ensicaen.fr

GOUBRAIM Ayoub
ayoub.goubraim@ecole.ensicaen.fr

FEST
RACING TEAM

Le 27 mai 2024

Enseignants encadrants

Mr Ziad Forest et Mr Fourney

Contents

1 Introduction

1.1 Contexte du projet

Dans ce challenge, nous avons programmé un pilote virtuel de Formule 1. Notre code devait naviguer de manière autonome à travers différents circuits en compétition avec d'autres pilotes codés par nos adversaires. Le pilotage se fait par communication avec un gestionnaire de course virtuel qui fournit toutes les informations concernant la piste et les positions des autres pilotes en temps réel.

Ce rapport décrit notre approche algorithmique pour résoudre ce problème. Nous explorerons également les obstacles auxquels nous avons été confrontés et discuterons des solutions que nous avons mises en œuvre.

1.2 Objectifs

Les principaux objectifs fixés pour ce défi sont multiples, visant à optimiser les performances du pilote qui doit terminer le circuit dans les plus brefs délais :

- **Atteindre l'arrivée le plus rapidement possible** : Le principal objectif est de faciliter le pilote à passer moins de temps sur un circuit afin de combiner l'efficacité en termes de vitesse avec la stratégie de course.
- **Respect des règles de la course** : Le pilote doit respecter les limitations de vitesse maximale, les règles de consommation de carburant et les restrictions d'accélération imposées par le règlement des courses.

Pour atteindre ces objectifs, nous avons établi les tâches suivantes :

- **Conception d'un algorithme de déplacement optimal** : Développer un algorithme qui garantit le parcours en le plus court chemin possible, prenant en compte les divers éléments du circuit et les règles de la course.
- **Tests sur différents circuits** : Valider les performances du pilote en le testant sur divers circuits pour mesurer son temps d'arrivée et gérer sa consommation de carburant.
- **Optimisation du code** : Améliorer les performances du pilote en optimisant le code source pour accroître l'efficacité de l'algorithme, réduire les temps de calcul et par conséquent améliorer les temps de course.

1.3 Outils

Pour mettre en œuvre ce projet, nous disposons du **langage C** compilé sur un système **Ubuntu Linux**. Nous avons utilisé **Visual Studio Code** comme éditeur de texte, qui fournit un environnement de développement intégré. Pour déboguer le code, nous avons utilisé **stderr** qui permet de tracer efficacement les erreurs et les avertissements, facilitant ainsi la résolution des problèmes ainsi que l'optimisation du code. Ces outils ensemble ont formé une suite qui soutenait notre processus de développement depuis l'écriture du code jusqu'à la résolution des bugs.

1.4 Arborescence du projet

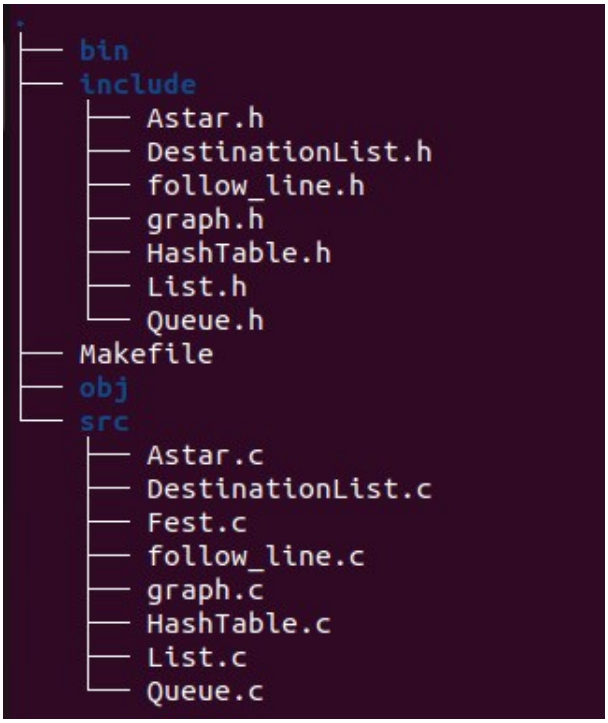

Figure 1: Répertoire du projet

2 Organisation du travail collectif

2.1 Répartition des tâches

Nous avons organisé la répartition du travail selon les compétences de chacun. Nous vous présentons ici le mode de travail selon les membres de l'équipe et l'échéancier. Trois cellules ont été mises en place, chacune ayant un rôle précis mais restant en étroite communication afin de ne pas perdre de temps.

2.2 Mode de travail

Afin que l'information soit la mieux partagée, des réunions hebdomadaires entre les différents membres du groupe étaient organisées. La répartition du travail était effectuée à ce moment-là afin d'établir les tâches à court et long terme essentielles au développement de notre projet.

2.3 Échéancier

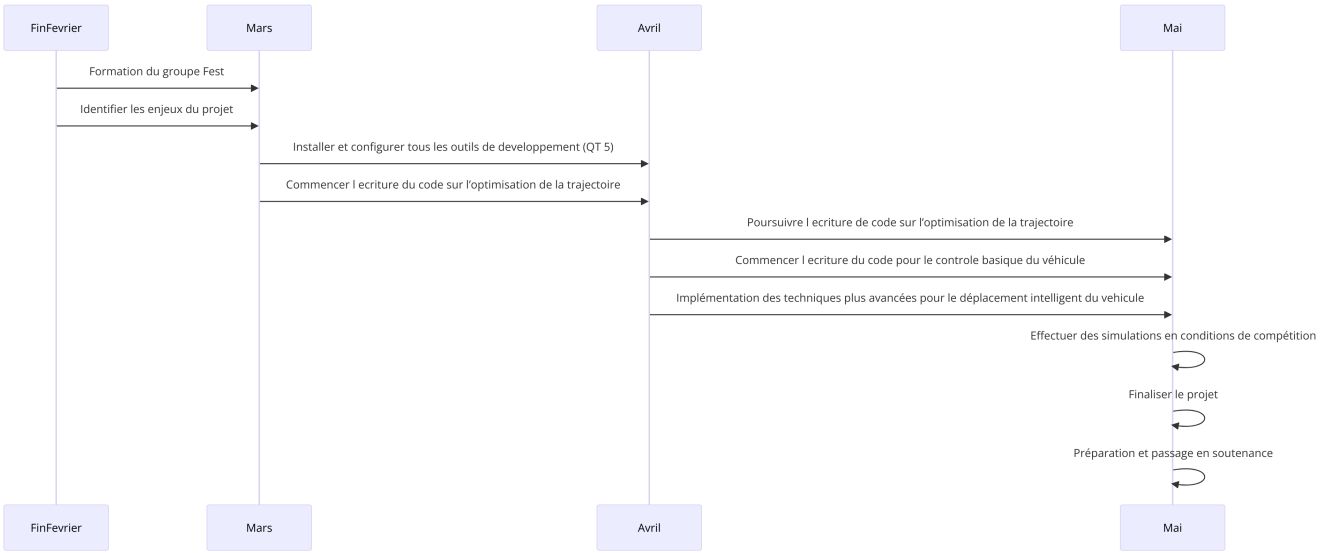

Figure 2: Diagramme de séquence pour l'échéancier.

3 Approche pour la résolution de ce problème

3.1 Pourquoi choisir l'algorithme A* ?

Différents algorithmes de recherche de chemin existent, chacun présentant ses propres bénéfices et inconvénients. **Dijkstra** et **A*** sont les algorithmes de recherche les plus populaires. L'algorithme de Dijkstra est efficace pour trouver le chemin le plus court dans un graphe pondéré, mais il ne prend pas en compte les heuristiques, ce qui peut rendre sa recherche plus lente dans certains cas. Dijkstra explore **tous les chemins possibles** à partir du nœud de départ pour garantir le chemin optimal, mais cela peut être très coûteux en termes de temps de calcul.

A* combine les avantages de Dijkstra avec une fonction **heuristique** guidant la recherche de manière plus intelligente. A* peut souvent trouver le chemin le plus court plus rapidement que Dijkstra en explorant **uniquement les chemins les plus prometteurs** grâce à une estimation de la distance restante jusqu'à l'objectif. C'est la raison pour laquelle nous avons opté pour A* dans le cadre de ce projet car il garantit un équilibre optimal entre performance et précision, élément essentiel dans un environnement de course en temps réel.

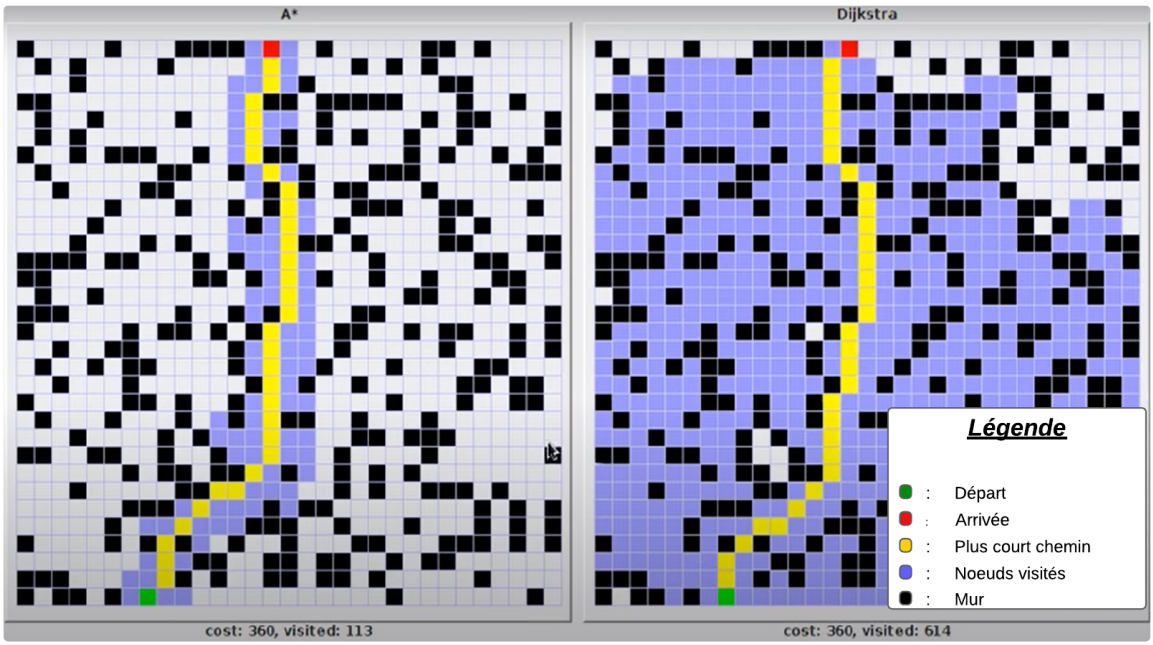

Figure 3: Comparaison entre A* et Dijkstra

3.2 Fonctionnement de l'algorithme A*

L'algorithme A* est un algorithme heuristique qui permet de trouver **très rapidement** un plus court chemin entre deux points avec d'éventuels obstacles. Outre sa vitesse, cet algorithme est réputé pour **garantir** une solution en sortie.

Le principe de l'algorithme A* se repose sur deux informations : l'heuristique (**h**) qui est une estimation du coût du chemin restant pour atteindre le nœud cible et le coût

du chemin entre le nœud actuel et le nouveau nœud (**g**). Ainsi, l'algorithme évalue les différents nœuds du graphe en se basant sur le coût total $f = g + h$ pour ensuite choisir le nœud ayant le plus petit coût total f. Ceci va permettre de trouver le chemin pour atteindre la destination avec **un coût total f minimal**.

L'heuristique doit impérativement être admissible. En effet, l'admissibilité garantit que l'algorithme A* trouve toujours le chemin optimal. Lorsqu'une heuristique est admissible, A* explore les nœuds de manière à trouver le chemin avec le coût total le plus bas sans manquer le chemin optimal en raison d'une surestimation du coût.

Pour gérer les nœuds à explorer, A* utilise deux structures de données principales : la **liste ouverte** et **la liste fermée**. La liste ouverte contient les nœuds à explorer triés selon leur coût total estimé f. L'algorithme commence en ajoutant le nœud de départ à la liste ouverte. À chaque itération, il sélectionne et retire le nœud avec le plus petit coût f de la liste ouverte pour l'explorer. Les voisins du nœud sélectionné, appelés successeurs, sont ensuite évalués. Pour chaque successeur, les coûts g, h et f sont calculés. Si un successeur n'est pas déjà dans la liste ouverte, il y est ajouté. Si un successeur est déjà présent avec un coût g plus élevé, la liste ouverte est mise à jour avec le nouveau coût plus bas.

La liste fermée contient les nœuds déjà explorés. Lorsqu'un nœud est retiré de la liste ouverte pour être exploré, il est ajouté à la liste fermée. Avant d'explorer un nœud successeur, l'algorithme vérifie s'il est déjà dans la liste fermée. Si c'est le cas, il n'est pas réexaminé, ce qui permet d'éviter les boucles infinies et de réduire les calculs redondants, rendant l'algorithme plus efficace.

L'algorithme se poursuit ainsi en explorant les nœuds avec le coût total le plus bas jusqu'à ce que le nœud objectif soit trouvé. Une fois l'objectif atteint, le chemin optimal est reconstruit en partant du nœud objectif et en remontant à travers les parents de chaque nœud jusqu'au nœud de départ. L'utilisation combinée des listes ouverte et fermée permet à l'algorithme A* de trouver le chemin optimal de manière efficace, garantissant ainsi une solution optimale sans manquer le chemin le plus court en raison d'une surestimation des coûts.

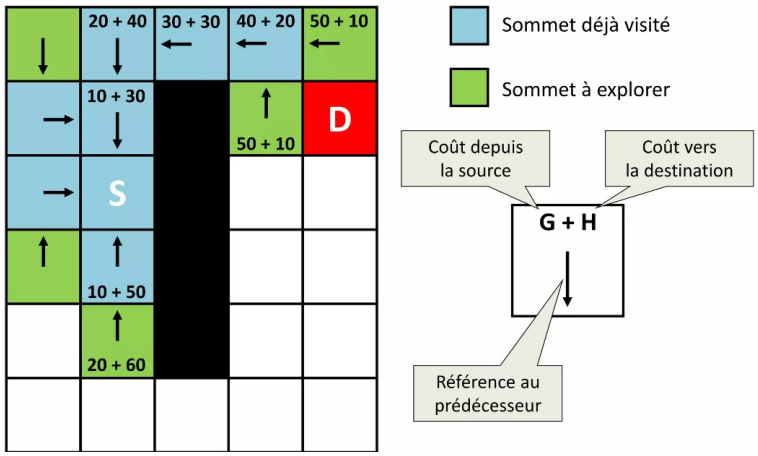

Figure 4: Le fonctionnement de A*

4 Difficultés rencontrées

4.1 Allocation initiale des nœuds

L'une des premières difficultés rencontrées au début du projet a été liée à la gestion de la mémoire pour stocker les informations des nœuds. Initialement, nous avions décidé **d'allouer tous les nœuds au départ** et de stocker leurs informations dans un tableau bidimensionnel. Cette approche semblait logique pour faciliter l'accès rapide aux informations des nœuds pendant l'exécution de l'algorithme.

Cependant, nous avons rapidement rencontré un problème majeur : sur des cartes de grande taille, cette méthode entraînait un crash du pilote dès le début. Le crash était dû à **une consommation excessive de mémoire**, car l'allocation d'un tableau de deux dimensions pour stocker toutes les informations des nœuds nécessitait une quantité de mémoire très élevée. Cela dépassait les limites de mémoire disponibles, surtout pour les grandes cartes avec un grand nombre de nœuds.

4.2 Gestion du temps de calcul

Pour respecter la contrainte d'une seconde pour envoyer les commandes d'accélération, nous avons initialement implémenté la liste ouverte (open list) de l'algorithme A* en utilisant une liste chaînée. Cette structure de données permet une insertion rapide, mais elle s'est avérée très lente lorsqu'il s'agissait de trouver et de défiler le nœud avec le plus petit coût. La recherche du nœud optimal dans une liste chaînée non triée nécessite de parcourir toute la liste, ce qui ralentissait considérablement l'algorithme, surtout sur de grandes cartes avec de nombreux nœuds à explorer.

De plus, pour la liste fermée (closed list), nous utilisions également une liste chaînée. Cela impliquait qu'à chaque fois que nous devions vérifier si un nœud existait déjà dans la liste, nous devions parcourir toute la liste. Cette opération répétée ajoutait un surcoût important au temps de calcul global, rendant l'algorithme encore moins performant.

4.3 Problèmes liés aux collisions entre pilotes

Un autre défi majeur rencontré au cours du développement était la gestion des collisions entre les pilotes. Lors de la simulation de la course, les pilotes peuvent se retrouver sur des trajectoires convergentes, ce qui entraîne des collisions. Il fallait identifier correctement et rapidement les moments où deux voitures occupaient la même position ou passaient par le même point intermédiaire.

Un autre aspect délicat était de gérer les situations où un pilote pouvait se "téléporter" d'une case à l'autre en passant par un mur, ce qui violait les règles de déplacement et nécessitait une détection et une gestion appropriées pour maintenir l'intégrité de la simulation.

4.4 Le choix de la meilleure heuristique

Pendant le développement, l'un des défis rencontrés a été de sélectionner la bonne heuristique pour l'algorithme A*. L'algorithme est fortement influencé par l'efficacité de

l'heuristique utilisée pour estimer les coûts restants jusqu'à l'objectif. Des chemins sous-optimaux peuvent être provoqués par une heuristique mal adaptée, ce qui peut considérablement augmenter le temps de calcul. Il était nécessaire de trouver une heuristique précise et performante, capable de guider efficacement l'algorithme à travers le circuit tout en respectant les contraintes temporelles imposées par la compétition.

4.5 Problème du carburant sur les terrains difficiles

Un autre défi était la gestion du carburant sur les terrains difficiles. Sur certaines cartes, il était impossible de terminer la course en passant par la route normale en raison de sections de sable obligatoires. Le sable ralentit fortement la voiture et consomme plus de carburant, ce qui rendait la planification du trajet extrêmement complexe. Les pilotes devaient trouver un équilibre entre prendre des raccourcis à travers le sable et rester sur la piste normale pour économiser du carburant, ce qui ajoutait une couche supplémentaire de difficulté à l'optimisation des trajectoires.

5 Solutions proposées

5.1 Allocation sélective des nœuds

Pour résoudre le problème de la gestion excessive de la mémoire, nous avons adopté une approche plus dynamique et efficace. Plutôt que d'allouer tous les nœuds dès le départ, nous avons décidé de n'allouer la mémoire pour un nœud que lorsqu'il répond aux conditions nécessaires pour être ajouté à la liste ouverte de l'algorithme A*.
Concrètement, un nœud n'est créé et stocké que s'il est évalué comme un successeur potentiel du nœud actuellement exploré et s'il a un coût total f compétitif par rapport aux autres nœuds. Cette méthode permet de réduire considérablement l'utilisation de la mémoire, car seuls les nœuds réellement pertinents pour la recherche du chemin optimal sont alloués et stockés. De plus, cette approche garantit que la mémoire est utilisée de manière efficace en se concentrant uniquement sur les nœuds nécessaires à l'exploration en cours, évitant ainsi les surcharges de mémoire qui provoquaient des crashs lors de l'utilisation de grandes cartes.

5.2 Utilisation d'une file de priorité et table de hachage

La décision d'utiliser une file de priorité plutôt qu'une liste chaînée pour implémenter la liste ouverte de l'algorithme A* s'est avérée être une solution efficace et judicieuse. En adoptant cette approche, nous avons pu améliorer considérablement les performances de l'algorithme en réduisant le temps nécessaire pour trouver et défiler le nœud avec le plus petit coût. La file de priorité, basée sur une structure de données comme une file (queue), permet des opérations d'insertion, d'accès et de suppression rapides, ce qui a permis à notre algorithme de respecter la contrainte de temps tout en conservant son efficacité, même sur des cartes de grande taille avec un grand nombre de nœuds à explorer. En fin de compte, cette solution nous a permis de surmonter les difficultés initiales liées à la gestion de la mémoire et à l'optimisation des performances, garantissant ainsi le bon fonctionnement de notre système dans des conditions variées.

En parallèle, nous avons utilisé une table de hachage pour la liste fermée (closed list), ce qui s'est révélé extrêmement bénéfique. Cette structure nous a permis d'accéder rapidement à un élément pour vérifier s'il appartient déjà à la liste fermée. Grâce à la table de hachage, les opérations de recherche et de vérification sont devenues beaucoup plus rapides, contribuant à la réduction du temps de calcul global de l'algorithme. L'utilisation de la table de hachage a ainsi permis d'optimiser encore davantage la gestion des nœuds explorés, assurant une performance élevée et une efficacité accrue de notre algorithme A*.

5.3 Gestion des collisions

Pour gérer efficacement les collisions dans le cadre de la planification de trajectoire pour notre véhicule, nous avons mis en place une solution qui repose sur deux fonctions principales : `collisionDetection` et `reachableNode`.

La fonction `collisionDetection` est chargée de vérifier la présence d'obstacles ou de positions occupées le long d'une trajectoire donnée entre deux points. En utilisant une approche de balayage de ligne, nous parcourons chaque point le long de la trajectoire pour détecter tout obstacle ou toute position occupée. Si un tel obstacle est rencontré ou si une position secondaire est occupée, la fonction renvoie un signal indiquant une collision potentielle.

D'autre part, la fonction `reachableNode` nous permet de déterminer si un nouveau nœud est atteignable à partir de la position actuelle du véhicule. Elle vérifie si le nouveau nœud se trouve à l'intérieur des limites de la carte, s'il y a un obstacle à sa position, et si le nouveau nœud correspond à la position actuelle du véhicule. Si ces conditions sont remplies, la fonction indique que le nœud est atteignable et qu'aucune collision n'est à craindre.

En combinant ces deux fonctions dans notre algorithme de planification de trajectoire, nous nous assurons que seuls les nœuds atteignables et sans collisions sont pris en compte pour la recherche du chemin optimal. Cette approche garantit que le chemin calculé est sûr et viable pour notre véhicule, en évitant les trajectoires obstruées par des obstacles ou des positions occupées.

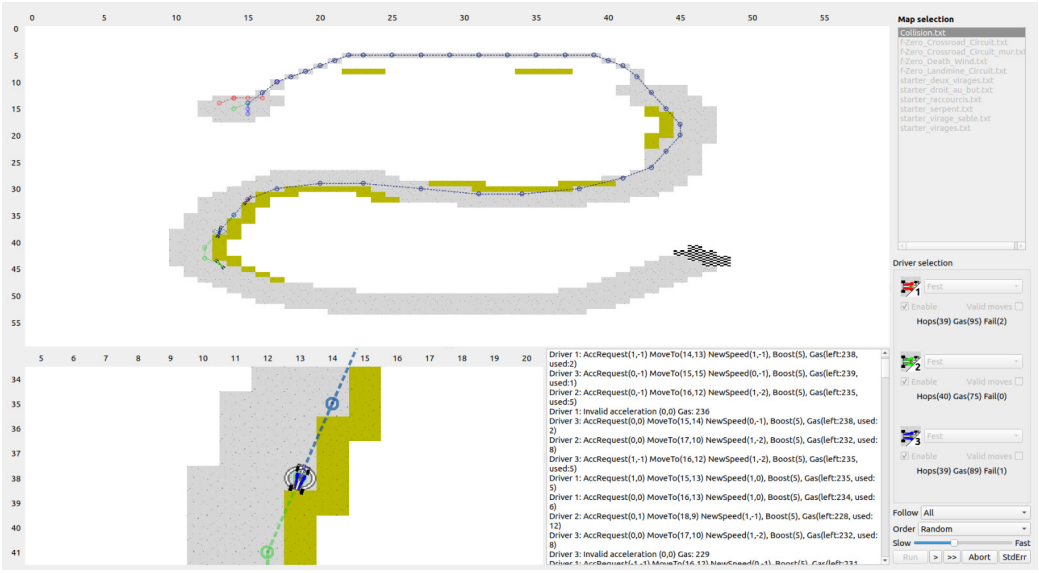

Figure 5: Notre pilote dans une map où la gestion des collisions est impérative

5.4 Heuristique de Chebyshev

L'heuristique de Chebyshev est une méthode utilisée dans l'algorithme A* pour estimer les coûts restants jusqu'à l'objectif. Elle prend en compte les mouvements horizontaux, verticaux et diagonaux sur une grille.

Concrètement, cette heuristique évalue la distance entre deux points en mesurant la plus grande des différences absolues entre les coordonnées horizontales et verticales. Autrement dit, elle considère le nombre minimum de mouvements nécessaires pour atteindre l'objectif en autorisant les déplacements diagonaux, et en prenant en compte que ces déplacements sont équivalents en coût aux déplacements horizontaux et verticaux.

L'utilisation de l'heuristique de Chebyshev permet à l'algorithme A* d'estimer de manière efficace les coûts restants pour atteindre l'objectif en tenant compte des mouvements possibles dans toutes les directions sur la grille. Cette approche optimise la gestion des virages, en fournissant des trajectoires plus naturelles et fluides. En évitant de surestimer les parcours diagonaux, l'heuristique garantit que tous les types de mouvements sont traités de manière équivalente, ce qui conduit à des chemins plus courts et mieux optimisés pour atteindre la destination.

5.5 Gestion du carburant en évitant le sable

Pour optimiser la gestion du carburant sur des terrains difficiles comportant des sections de sable, nous avons choisi d'introduire une pondération supplémentaire dans notre algorithme de planification de trajectoire. Cette pondération vise à encourager l'évitement des zones de sable, connues pour ralentir considérablement le véhicule et entraîner une consommation de carburant accrue. Notre approche consiste à attribuer un coût plus élevé aux déplacements sur le sable par rapport à une route normale, ce qui incite l'algorithme à privilégier les chemins présentant le moins de sections sablonneuses possibles. Ainsi, en introduisant cette pondération, nous visons à optimiser la consommation de carburant et à garantir une planification de trajectoire efficace pour atteindre la ligne d'arrivée dans les délais impartis.

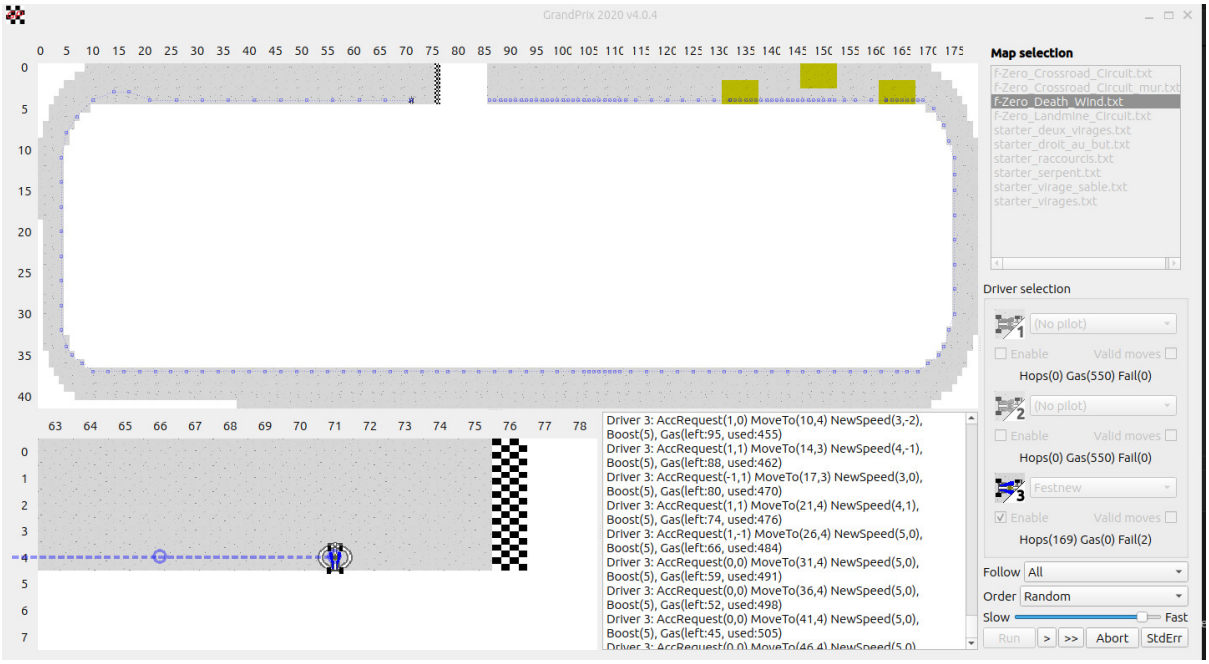

Figure 6: Pilote sans gestion du carburant

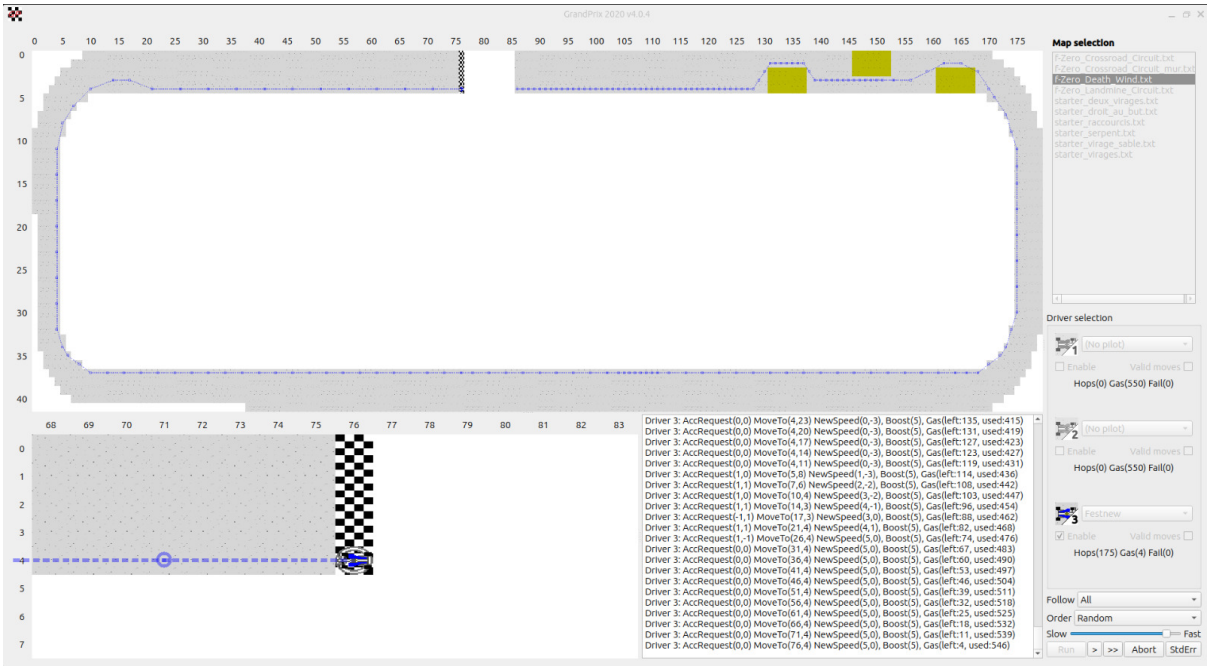

Figure 7: Pilote avec gestion du carburant

6 Ce qui nous différencie des autres

Ce qui rend notre implémentation de l'algorithme A* unique par rapport aux autres, c'est notre approche innovante de gestion des accélérations possibles et de la création minimale de nœuds. Contrairement aux approches classiques qui explorent de nombreux nœuds potentiels, nous parcourons **uniquement les accélérations** possibles et testons les variables obtenues par ces accélérations, telles que la vitesse et la position. Cette méthode permet de réduire considérablement le nombre de nœuds à examiner, ce qui

améliore l'efficacité de l'algorithme.

De plus, nous recalculons l'A* à chaque tour pour adapter dynamiquement la trajectoire du pilote en cherchant à maintenir une vitesse maximale. Si, à cette **vitesse maximale**, aucune route n'est possible, nous **décrémentons progressivement la vitesse jusqu'à trouver une route viable**. Cette double stratégie de minimisation des nœuds et de recalcul dynamique assure non seulement une performance optimale en termes de temps de calcul, mais aussi une navigation plus fluide et rapide sur le circuit, permettant ainsi au pilote de s'adapter aux conditions changeantes de la course tout en optimisant sa vitesse et sa trajectoire.

7 Conclusion

Le projet **"Grand-Prix F1"** a été bien plus qu'une simple expérience ; il a été une véritable aventure qui a profondément enrichi notre parcours. Nous avons eu l'opportunité unique de développer et d'appliquer des compétences avancées en programmation et en algorithmique dans un contexte passionnant et stimulant. Chaque étape de ce projet nous a confrontés à des défis exigeants, mais notre engagement sans faille et notre esprit d'équipe nous ont permis de les surmonter avec succès. Notre pilote s'est distingué par ses performances exceptionnelles lors des tests, illustrant parfaitement le niveau d'excellence que nous visons. Forts de cette expérience, nous sommes plus que confiants quant à ses capacités à briller lors des compétitions à venir. En résumé, le projet **"Grand-Prix F1"** a été bien plus qu'un simple exercice technique ; il a été une véritable aventure humaine, éducative et inspirante.

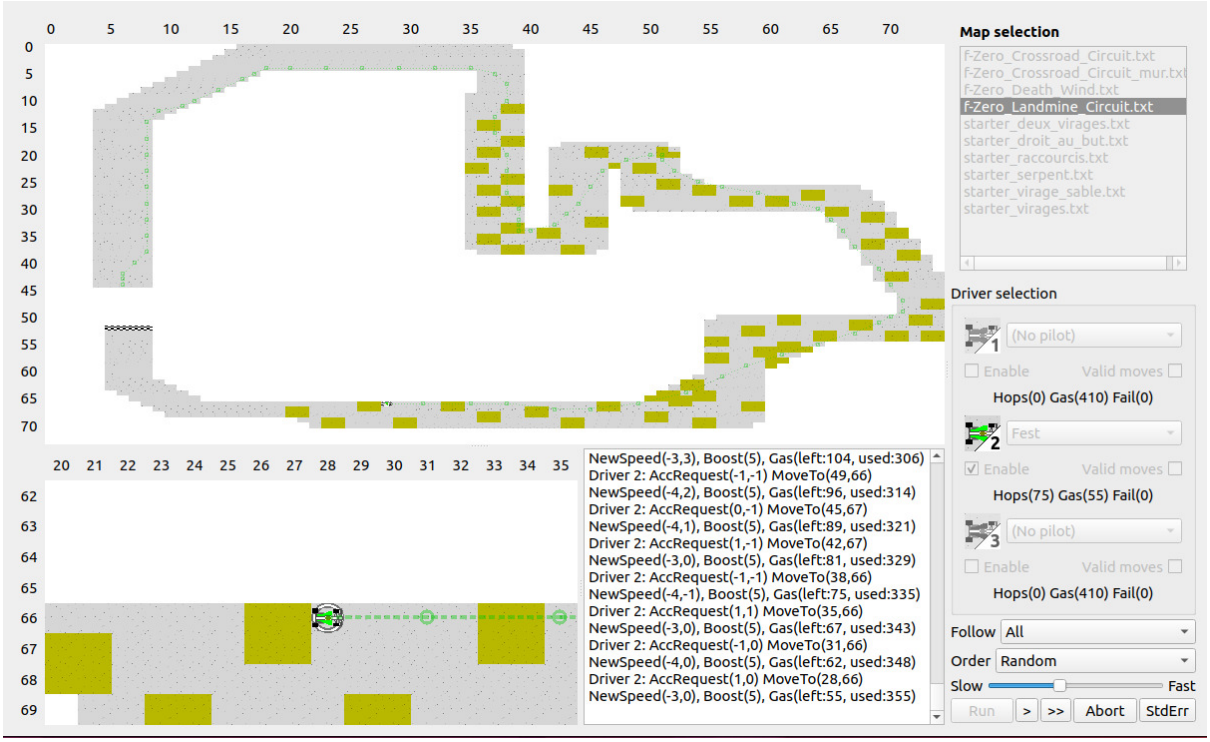

Figure 8: Notre pilote final