Ecole Publique d'Ingénieurs en 3 ans

Rapport

CHALLENGE DE PROGRAMMATION

Le 27 Mai 2024

Florestan Trillot - Benjamin Boccara - Matthieu Castetz

www.ensicaen.fr

TABLE DES MATIERES

1. INTRODUCTION 4

1.1. Présentation du projet 4

2. APPROCHE ALGORITHMIQUE 6

2.1. Présentation du concept 6

2.2. Solution algorithmique 9

3. DEVELOPPEMENT DU PROJET 11

3.1. Outil de développement 11

3.2. Répartition des tâches 11

4. DIFFICULTES ET SOLUTIONS 12

4.1. Difficultés 12

4.2. Solutions 12

5. CONCLUSION 13

TABLE DES FIGURES

Figure 1 : Interface du gestionnaire de course 4
Figure 2 : Champ de vision du pilote 6
Figure 3 : Représentation des cases scannées 6
Figure 4 : Ensemble des cases pouvant être atteintes 7
Figure 5 : Détermination de la meilleure case du prochain déplacement 8
Figure 6 : Suppression d'un chemin à l'aide d'une prédiction 8
Figure 7 : Structure Move 9
Figure 8 : Structure Case 9

1. INTRODUCTION

1.1. Présentation du projet

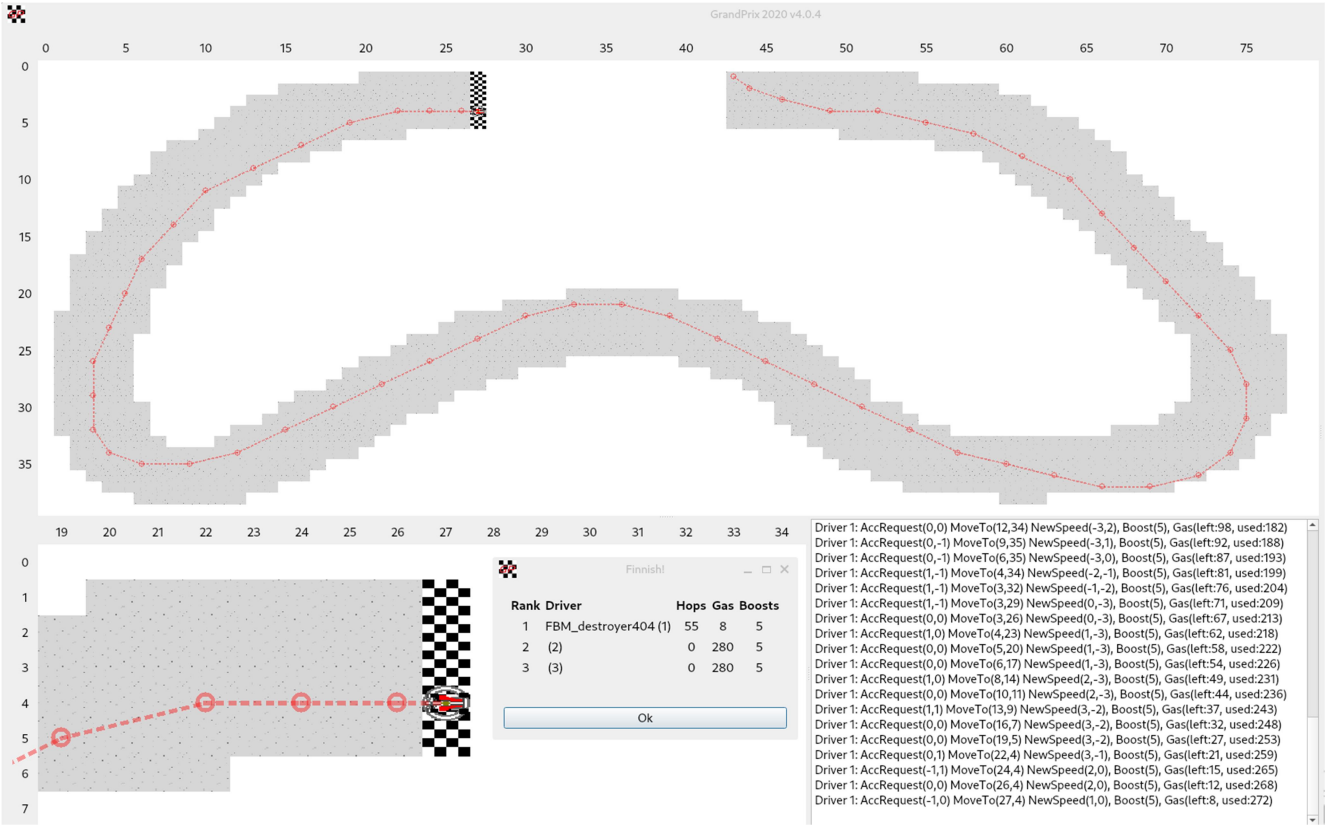

Figure 1 : Interface du gestionnaire de course

A l'occasion d'une course virtuelle organisée par l'école, en tant que participant il nous a été demandé de développer un pilote capable d'interagir avec un gestionnaire de course. Au premier tour, le gestionnaire communique la position initiale de chaque pilote. À chaque tour, le pilote doit ensuite indiquer la vitesse à laquelle il souhaite avancer tout en obéissant à certaines règles :

- Il ne peut pas occuper une position déjà prise par un autre pilote,
- Il n'est pas possible de couper les virages en empruntant une case hors du circuit, si le déplacement prévu par le pilote passe par une telle case, son déplacement est annulé
- Une vitesse maximale est définie par le gestionnaire et celle-ci ne doit pas être dépassée sous peine d'arrêter complètement la voiture,
- Une quantité de carburant est déterminée au début de chaque course en fonction de la carte choisie, chaque action telle qu'avancer et freiner en consomme proportionnellement à la vitesse, lorsque le carburant est épuisé, le pilote s'arrête et ne peut plus finir la course
- Un nombre limité de "boosts" est mis à disposition pour chaque course, l'utilisation de ce boost permet de doubler l'accélération pour un tour

Les pilotes sont classés par ordre d'arrivée ou par leur avancée dans la course s'ils ne la terminent pas.

Ils trouvent également sur leur route différentes surfaces telles que la route, qui n'affecte pas leur vitesse, du sable qui la limite fortement et la ligne d'arrivée à atteindre absolument.

Nous verrons par ailleurs qu'il faudra parfois se résoudre à entrer dans le sable quitte à réduire sa vitesse pour parvenir plus rapidement à l'arrivée et espérer remporter la course.

2. APPROCHE ALGORITHMIQUE

2.1. Présentation du concept

Nous avons opté pour une approche intuitive du pilotage consistant à modéliser l'ensemble de la piste se trouvant devant le pilote comme s'il s'agissait de son champ de vision.

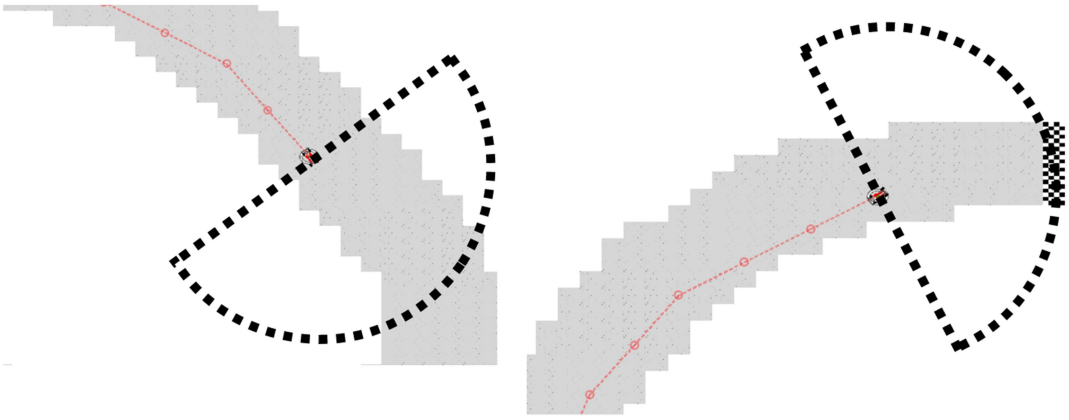

Figure 2 : Champ de vision du pilote

Le pilote va ainsi regarder à une certaine distance devant lui pour chercher les trajectoires qui s'offrent à lui. Parmi celles-ci, lorsqu'il détecte la ligne d'arrivée, en partie ou en totalité, il fait en sorte de l'atteindre en priorité.

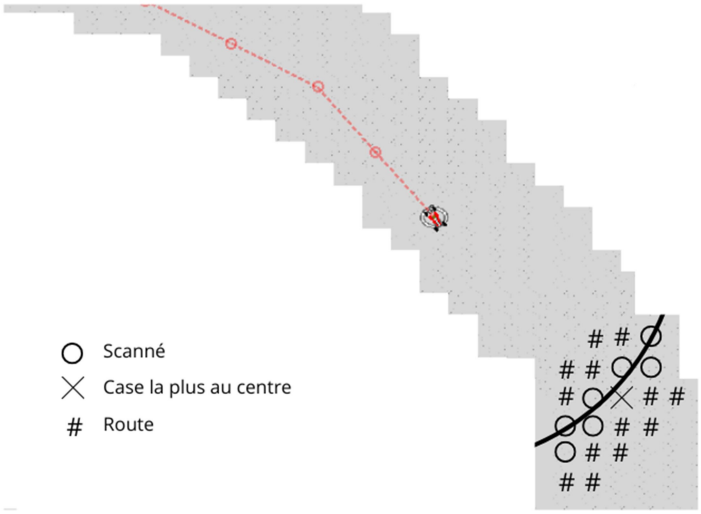

Figure 3 : Représentation des cases scannées

Si aucune partie de la ligne d'arrivée n'est détectée, le champ de vision est réduit à un arc de cercle d'un certain rayon coupant la piste.

Les cases concernées sont marquées comme accessibles, et la case la plus au centre devient alors l'objectif "indirect" du pilote. Elle est marquée d'une croix sur le schéma. Une trajectoire vers le centre de la piste permet de maximiser la vitesse, y compris lors de la prise de virages.

Toutes les cases accessibles ont maintenant été trouvées et un champ de possibles s'ouvre devant la voiture, c'est à ce moment que sa vitesse et son accélération potentielle entrent en jeu.

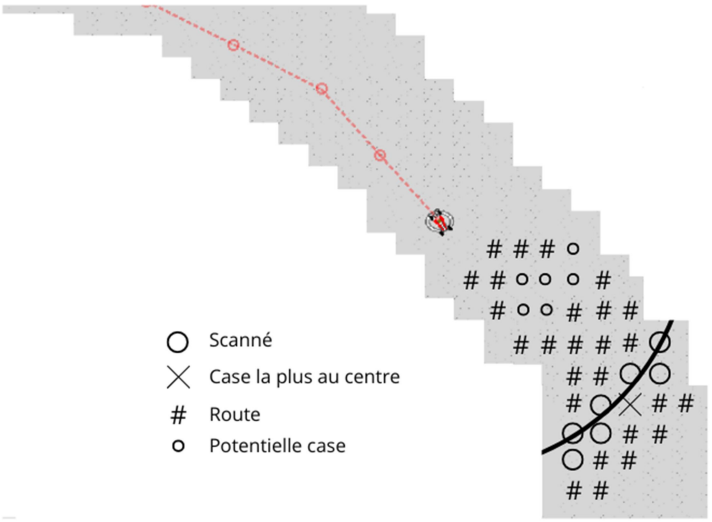

Figure 4 : Ensemble des cases pouvant être atteintes

Parmi les cases accessibles, le pilote calcule celles qu'il va pouvoir atteindre à l'aide de sa vitesse actuelle et celles qu'il atteindrait en accélérant.

Dans le cas où accélérer conduirait à un crash, l'accélération est réduite ce qui permet entre autres de mieux négocier les virages.

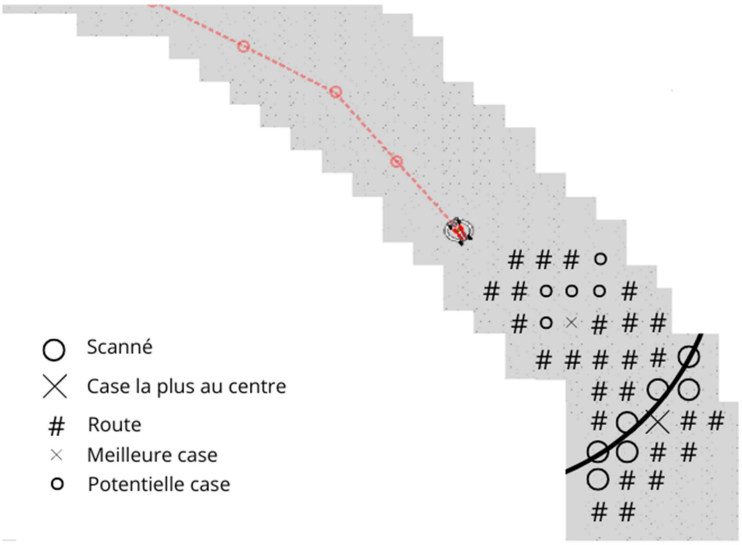

Figure 5 : Détermination de la meilleure case du prochain déplacement

Parmi toutes les cases qu'il peut atteindre avec son accélération, il va finalement choisir celle qui est le plus proche de l'objectif. L'objectif étant au plus proche du milieu de la piste, notre pilote choisira souvent une trajectoire proche du centre.

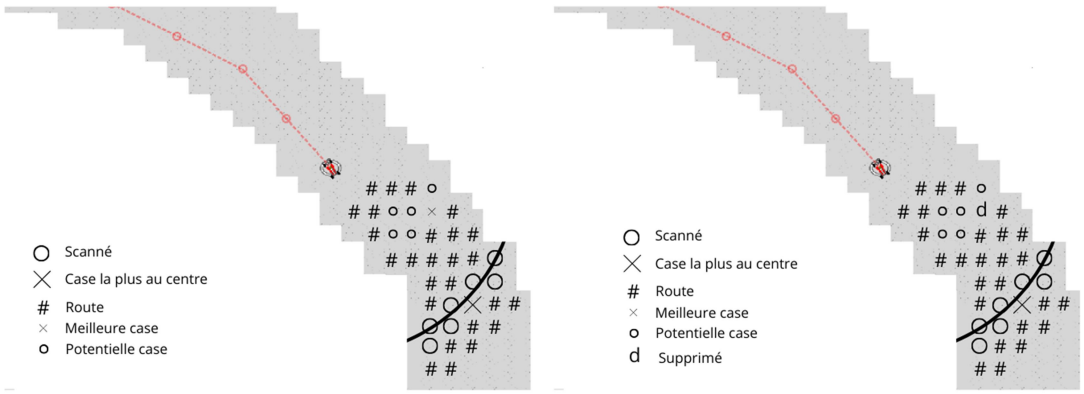

Figure 6 : Suppression d'un chemin à l'aide d'une prédiction

Cependant, ce choix optimal à un moment t a peu de chances d'être optimal sur l'ensemble de la course. C'est pourquoi le coup suivant est calculé pour vérifier si le premier ne met pas le pilote dans une position délicate, le forçant par exemple à rentrer dans un mur à cause d'une trop forte accélération.

Si le coup simulé place la voiture dans le mur, la position précédente est supprimée des cases accessibles.

2.2. Solution algorithmique

Afin d'implémenter notre pilote, nous avons utilisé deux structures de données.

La structure Move permet de stocker un mouvement de la voiture à un instant t, avec les composantes en x et en y d'une position, et les composantes en x et en y de l'accélération à effectuer pour atteindre cette position.

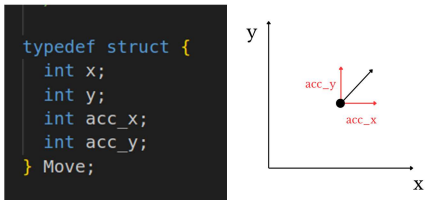

Figure 7 : Structure Move

La structure Case permet de stocker la valeur d'un pixel de la map, pour réagir en conséquence. Les valeurs prises par ce pixel peuvent évoluer selon les actions de la voiture, pour lui permettre de se frayer son chemin jusqu'à l'arrivée.

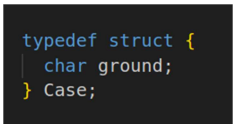

Figure 8 : Structure Case

Pour commencer, nous voulions que la fonction radar scanne uniquement les cases accessibles devant notre pilote. Nous avons donc réalisé les deux fonctions suivantes : isAhead() et isAccessible().

1) La fonction ***isAhead*** :

La fonction ***isAhead*** évalue si un point donné se trouve devant la voiture ou non. Elle prend en entrée les coordonnées de la voiture, son vecteur vitesse, et les coordonnées du point à vérifier. Pour cela, elle calcule l'angle entre le vecteur vitesse de la voiture et le vecteur qui relie la voiture au point en question. Si la valeur absolue de cet angle est inférieure à 90°, le point est considéré comme étant devant la voiture, et la fonction retourne true

2) La fonction ***isAccessible*** :

La fonction ***isAccessible*** se décompose en plusieurs parties :

1. Elle prend en entrée les coordonnées de la position à vérifier, les coordonnées de notre pilote ainsi que les coordonnées des deux autres pilotes.

2. La fonction commence par vérifier si une ligne peut être tracée entre la position actuelle et la position à vérifier en utilisant la fonction verifyLine. Cette dernière utilise

l'algorithme de Bresenham pour convertir une ligne continue en une ligne discrète, ce qui permet de vérifier que toutes les cases traversées sont soit des routes, du sable ou une ligne d'arrivée.

3. Si cette ligne est vérifiée, elle compare les coordonnées de la position à vérifier avec celles des deux autres pilotes afin de s'assurer que la case n'est pas déjà occupée.

4. Si les conditions précédentes sont remplies, la fonction retourne 1, ce qui signifie que la case est accessible.

3) NextMove:

La fonction NextMove parcourt le tableau des coordonnées accessibles et compare le chemin vers ces coordonnées au déplacement actuellement considéré comme nous rapprochant le plus de notre objectif, et en le mettant à jour si ce nouveau déplacement est plus avantageux.

Dans le cas où la distance est la même, on cherche la trajectoire consommant le moins d'essence possible.

Enfin, dans le cas où la distance est nulle, on renvoie directement le déplacement associé.

3. DEVELOPPEMENT DU PROJET

3.1. Outil de développement

L'IDE Visual Code Studio

Distribution Ubuntu

3.2. Répartition des tâches

Nous avons utilisé la plateforme collaborative GitHub pour développer notre code. Ainsi, chacun de nous pouvait accéder facilement aux dernières avancées.

Cela nous a permis de nous entraider facilement si l'un de nous avait un problème, ou était bloqué sur une fonctionnalité.

Florestan	Benjamin	Matthieu
Implémentation isAccessible et isAhead	Implémentation gasConsumption et calculateSpeed	Stockage de la map
Implémentation de la fonction radar	Implémentation de la fonction radar	Implémentation de la fonction radar
Debuggage des prises de décision du pilote grâce à la fonction printMap afin d'optimiser les paramètres	Mise en place de simulation pour éviter les crashs	Stockages des dernières positions de la voiture pour éviter les retours en arrière

4. DIFFICULTES ET SOLUTIONS

4.1. Difficultés

La modélisation d'une trajectoire linéaire dans un environnement pixélisé est loin d'être facile. Or, afin de vérifier que les cases devant le pilote sont accessibles, il faut s'assurer que le chemin menant à elles est valable. Cette vérification est assurée par la fonction verifyLine qui étudie la nature de toutes les cases entre deux positions données.

La gestion du carburant a été une étape particulièrement difficile et contraignante pour nous. En effet, on s'est souvent retrouvé dans la situation où les trajectoires prises par le pilote étaient "parfaites", mais il ne terminait jamais la course, faute de carburant.

De plus, cette limitation du carburant à compliqué l'utilisation des boosts consommant une grande quantité de carburant.

4.2. Solutions

Une solution pour implémenter verifyLine a été d'utiliser l'algorithme de Bresenham permettant d'approximer une ligne entre deux points en utilisant les pixels adjacents choisis grâce à une marge d'erreur.

Une solution pour terminer la course avec du carburant a été de modifier la trajectoire "parfaite" prise par notre pilote, pour une trajectoire moins efficace, mais moins gourmande en carburant. Cela montre que l'efficacité est relative.

Malgré l'avantage certain qu'auraient pu nous apporter les boosts, nous avons fait le choix de ne pas les utiliser, les pilotes ayant parfois du mal à terminer les circuits sans les boosts.

5. CONCLUSION

Ce projet nous a appris à être créatif dans notre approche de problèmes algorithmiques, en utilisant les compétences que nous avons acquises et les moyens mis à disposition pour répondre à des problématiques inédites. De plus, nous avons appris à être rigoureux notamment lors de l'utilisation de pointeurs et de structures de données.

Venant tous trois de classes préparatoires, il s'agissait là de notre deuxième expérience de développement collectif après le traitement d'image du premier semestre. Cela a été l'occasion d'apprendre à collaborer et de mieux comprendre ce que signifiait que de travailler en équipe, composer avec les contraintes de chacun, se répartir efficacement les tâches, se mettre régulièrement à jour sur l'avancée des autres parties, etc.

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