Ecole Publique d'Ingénieurs en 3 ans

Rapport

CHALLENGE DE PROGRAMMATION

LY Johann
TOUJANI Mohamed

Tuteur : **FOUREY Sebastien**
Tuteur : **Karim-Eric Ziad-Forest**

www.ensicaen.fr

TABLE DES MATIERES

1. Introduction

1.1. Contexte et motivations du projet

Ce rapport présente la conception et l'implémentation du comportement d'une voiture autonome sur un circuit 2D. Ce projet s'inscrit dans le cadre de notre cursus informatique de première année au sein de l'ENSICAEN mettant en application nos connaissances en algorithmique et gestion de projet. L'objectif principal est de développer un comportement autonome pour une voiture évoluant sur un circuit en deux dimensions.

1.2. Position du problème

Le problème est de concevoir et d'implémenter un comportement autonome pour une voiture sur un circuit en deux dimensions. Ce projet nous amène à faire preuve d'une capacité à gérer les contraintes de circuit, de vitesses, d'accélération, de consommation de carburant tout en intégrant un algorithme de plus court chemin. Le déroulement de la course se fait en interaction continue avec le gestionnaire de course fourni par l'école.

1.3. Objectifs du projet

Les objectifs principaux du projet sont les suivants :

1. Comprendre le fonctionnement du gestionnaire de course
2. Concevoir une stratégie globale du comportement de la voiture
3. Définir et implémenter les structures de données nécessaires pour représenter le circuit ainsi que notre pilote.
4. Déterminer le chemin optimal en implémentant un algorithme de plus court chemin
5. Mettre au point la lecture et l'écriture des informations entre le pilote et le gestionnaire de course (GDC)

2. Organisation du projet

2.1. Répartition des tâches et des responsabilités

Dans le cadre de ce projet réalisé en binôme, la répartition des rôles a été repartie de manière égale et flexible, favorisant ainsi la bonne gestion du projet.

2.2. Outils utilisés

Dans le cadre de ce projet, nous avons utilisé différents outils afin de garantir la qualité et la collaboration au sein de notre binôme. Ces outils sont les suivants :

Outils de communication :

- GitHub

Outils techniques :

- Langage de programmation C
- Vim
- Visual Studio Code
- Gestionnaire de course

3. Fonctionnement du GDC

3.1. Présentation du gestionnaire de course (GDC)

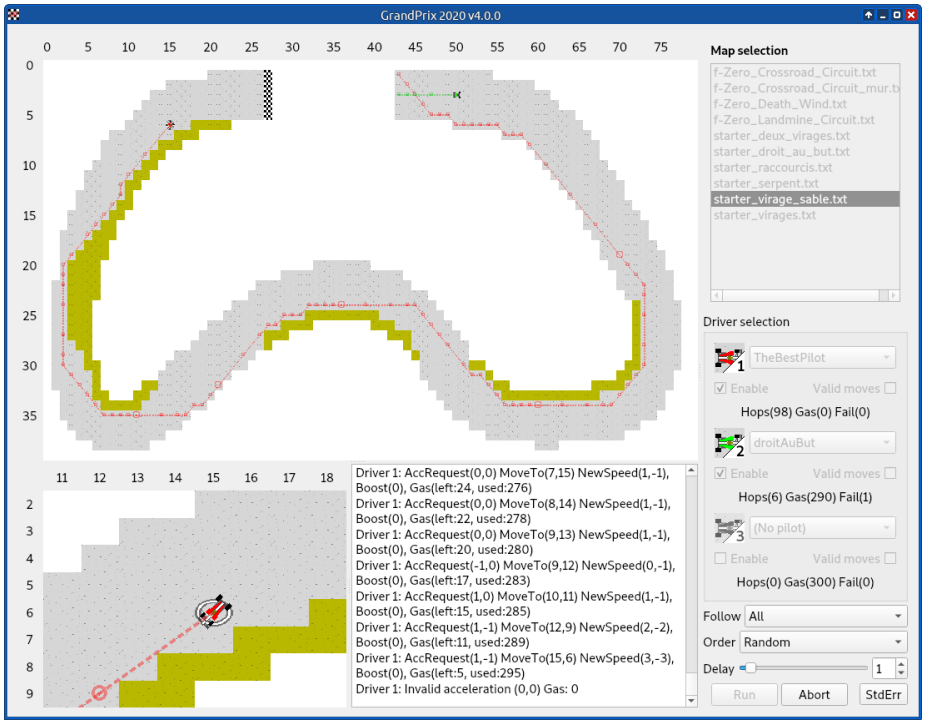

Figure 1 GDC

3.2. Descriptions des circuits

Chaque circuit est représenté par des caractère ASCII ainsi que de 3 entiers représentant dans l'ordre la largeur et la hauteur du circuit ainsi que la quantité de carburant disponible au départ de la course.

Tableau 1 Type de sol

Nom	Symbole	Description
Piste	#	Zone de piste normale
Sable	~	Ralentit fortement la voiture
Hors-piste	.	Terrain inaccessible
Arrivée	=	Atteindre ce type de sol

Figure 2 Exemple de carte

3.3. Règles du GDC

Tout au long de la course, notre voiture interagit avec le GDC via ses entrées et sorties standard, en mode texte.

Le pilote fourni au GDC un vecteur accélération *(accX, accY)* ∈ *{-1, 0, 1}*

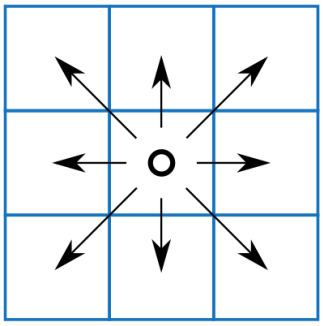

Figure 3 Possibilités Accélérations

Afin d'élaborer la solution du problème, nous avons besoin de structures de données préliminaires.

4. Structures préliminaires

4.1. Listes

Une liste est une structure de données définie comme un pointeur sur la première cellule de la liste.

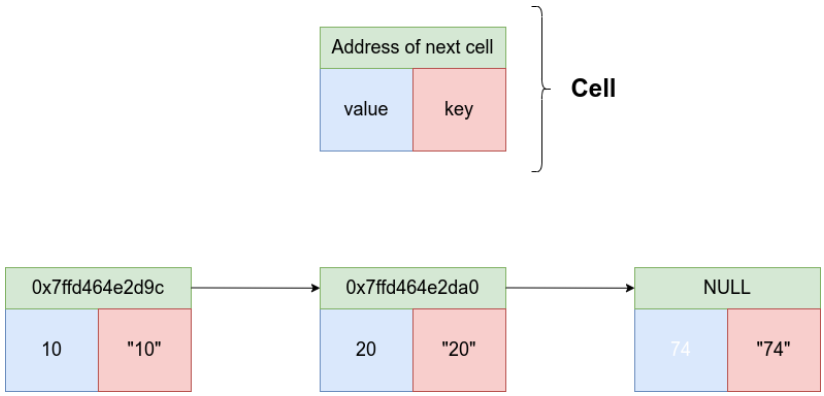

Figure 4 Representation D'une Liste

4.2. Files de priorité

Une file de priorité (ou tas) est une structure de données qui permet d'ordonner et de gérer un ensemble d'élément où chaque élément lui est associé une clé (ou priorité), facilitant ici l'accès à l'élément de plus basse priorité.

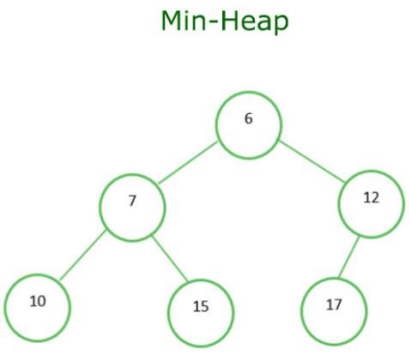

Figure 5 Heap

L'ensemble du projet est réalisé en langage de programmation C.

5. Développement de la solution

5.1. Structure de l'arborescence du projet

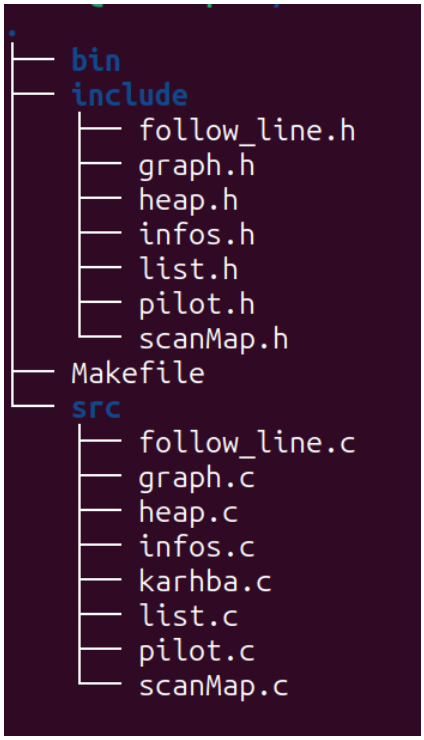

Figure 6 Arborescence

5.2. Modélisation du circuit

Lors du départ de la course, le GDC nous envoie via l'entrée standard les informations sur le circuit de la course.

On va donc lire chaque caractère et les stockés en mémoire.

On représentera le circuit en lui-même par une matrice de *pixel_t*

Figure 7 Définition structure pixel_t

5.3. Création du graphe

Afin de modéliser l'ensemble de la course, on créer une structure de données contenant de nombreuses informations concernant les informations de la course en temps réel.

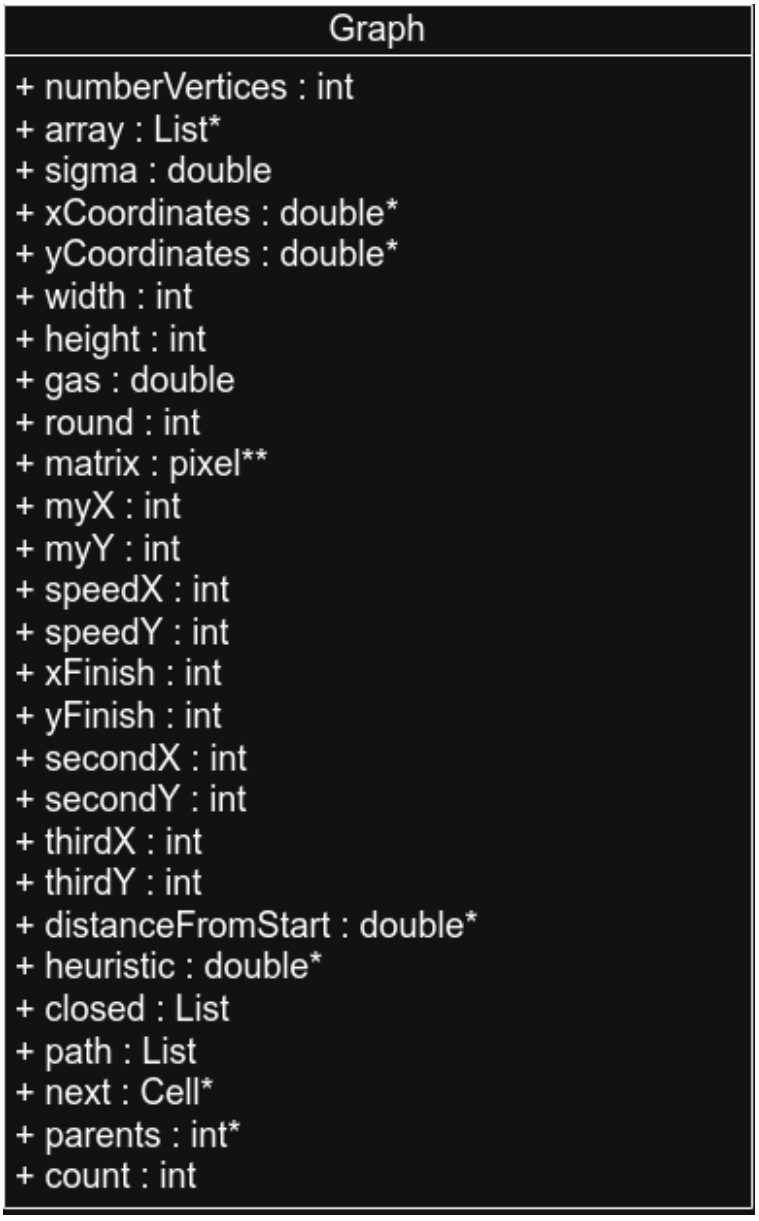

Figure 8 Définition structure Graph

La course, simulée par le graphe est représentée par de nombreuses données comme la position de toutes les voitures, le niveau de carburant, la vitesse de notre pilote et la ligne d'arrivée.

Chaque point accessible du circuit est représenté par un nœud du graphe dont les coordonnées seront stockées dans le graphe. Les connexions entre les points sont représentées par des arêtes. Une arête existe entre deux nœuds si la distance euclidienne entre eux est inférieur à un certain seuil. On pourra jouer sur ce seuil afin d'autoriser différents types de mouvement. En parcourant la matrice représentant le circuit, on créer tous les chemins existants parmi l'ensemble des nœuds du graphe. La représentation du graphe se fait ici par tableau de listes d'adjacences.

5.4. Algorithme du plus court chemin

L'algorithme A* est utilisé pour trouver le chemin le plus court entre deux points, ici entre le sommet de départ et le sommet d'arrivé, en tenant compte du coût réel pour atteindre un point et d'une estimation du coût restant pour atteindre le sommet d'arrivé. On appelle cette estimation l'heuristique. Dans le cadre de notre projet, on a choisi comme heuristique la distance de Tchebychev.

Définition [modifier | modifier le code]

Entre deux points *A* et *B*, de coordonnées respectives $(A_0, \ldots, A_n)$ et $(B_0, \ldots, B_n)$, la distance de Tchebychev est définie par :

$$d(A,B) = \max_{i \in [[0,n]]} (|A_i - B_i|).$$

Autrement dit : c'est la distance associée à la norme « infini ».

Figure 9 Source Wikipedia

L'algorithme repose sur l'utilisation d'une file de priorité (*open*) basée sur le coût réel et l'heuristique. Au départ de A* on ajoute le sommet de départ dans le tas. Tant que *open* n'est pas vide, on défile *open* et on ajoute la valeur défilée dans *closed* puis pour tous les sommets successeurs s de la valeur défilée n'appartenant pas à *closed*, on met leur clé à jour si nécessaire, de façon à minimiser leur clé. C'est-à-dire si on a trouvé un plus court chemin pour aller du sommet de départ à s en passant par le sommet défilé, on mets à jour sa clé. On retiendra en mémoire le parent de chaque nœud. On sort de l'algorithme une fois que le sommet d'arrivé soit en position de tête de *open*. Si la boucle "Tant que" finie, il y a une erreur sur l'implémentation de l'algorithme. Pour trouver le plus court chemin, on part du sommet de départ et on regarde sommet père ainsi de suite jusqu'au sommet de départ.

Ainsi, après avoir scanné le circuit et après que le GDC nous ait fourni les positions de départ, on calcule le plus court chemin en distance afin d'arriver à la ligne d'arrivée.

```
A*(G=(V,E), w, s, s') :
    open = file de priorité ordonnée par f[t]=d[t]+h[t]
           contenant s au départ
    closed = ensemble vide
    Tant que open n'est pas vide
        t = Défiler(open)
        ajouter t à closed
        si t = s' alors retourner d[s']
        Pour tout arc (t,u) de G avec u pas dans closed :
            si d[u] n'existe pas ou d[u]>d[t]+w(t,u):
                d[u] = d[t]+w(t,u)
                Calculer h[u] si nécessaire
                ajouter u à open s'il n'y est pas
    Retourner une erreur
```

*Figure 10 Algorithme de A**

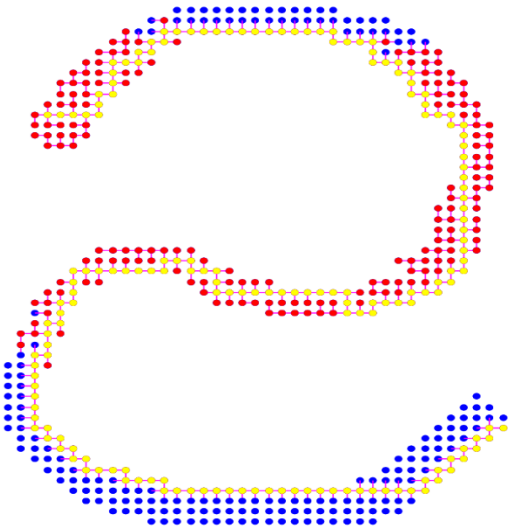

Figure 11 Test de A sur la map SERPENT*

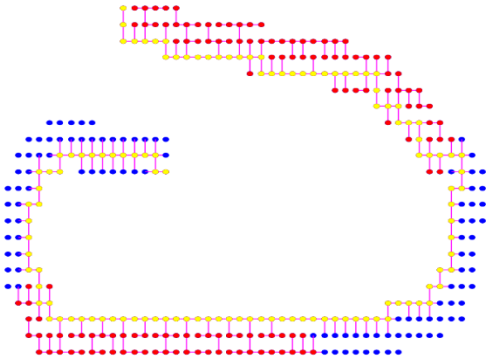

*Figure 12 Autre test de A**

5.5. Gestion des vitesses et accélérations

La gestion des vitesses et des accélérations est indispensable pour le comportement autonome de la voiture. La vitesse de la voiture est définie par le vecteur *(vX,vY)* avec *vX* les déplacements horizontaux et *vY* les déplacements verticaux. A chaque tour, on met à jour la vitesse de notre voiture en envoyant une requête d'accélération *(dvX, dvY)* au GDC. La norme de la vitesse résultante ne doit pas dépasser 5. Cette requête d'accélération est calculée afin de suivre le chemin optimal déterminée par l'algorithme A*, en prenant en considération les obstacles et virages.

Malgré nos efforts pour gérer efficacement les vitesses, lors de la gestion de celle-ci, nous avons fait face à des difficultés afin de maintenir un comportement précis de la voiture. En conséquence, la voiture ne dépasse pas 1 de vitesse en valeur absolue que cela soit pour *vX* ou *vY*. Cela limite grandement la performance de la voiture.

En résumé, la gestion des vitesses et des accélérations est cruciale afin de produire un comportement réaliste concernant la voiture. Cependant, des optimisations sont nécessaires et indispensables pour augmenter sa vitesse tout en maintenant son comportement stable et précis.

5.6. Gestion des collisions

Durant la course, on est amenés à donner un vecteur accélération au GDC amenant la voiture sur une case occupée par une autre voiture. Pour gérer ses potentielles collisions, nous devons donc mettre à jour le chemin optimal de tel sorte que la case occupée ne soit pas dans le chemin optimal. Pour cela, nous augmentons l'heuristique de cette case, rendant ainsi ce point moins attractif pour A*. Ainsi, lorsque A* est réappelée depuis la position courante, cette case est exclue du chemin optimal. Ainsi, nous évitons les collisions en adaptant dynamiquement le trajet de la voiture.

6. Conclusion et Perspectives

Dans le cadre de ce projet, nous avons élaboré et implémenté le comportement d'une voiture autonome sur un circuit deux dimensions en utilisant A*, un algorithme de plus court chemin. Les résultats montrent que notre implémentation permet à notre voiture de se diriger vers la ligne d'arrivée en suivant le trajet optimal tout en évitant les obstacles et les collisions avec d'autres voitures.

Néanmoins, plusieurs aspects de notre implémentation nécessitent une amélioration. Le principal problème est la gestion de la vitesse. En effet, la gestion inefficace de la vitesse diminue grandement les performances de notre voiture, amenant parfois celle-ci à s'épuiser de son carburant.

Ce problème met donc en évidence la nécessité de repenser notre gestion de la vitesse pour optimiser la gestion de carburant et donc optimiser la capacité de la voiture à finir le circuit à chaque fois.

Malgré ces obstacles, nous avons pu renforcer nos connaissances en programmation C, en algorithmique et nos méthodes de travail. Nous sommes confiants que le projet est sur la bonne voie. Les leçons tirées de ce projet nous serviront pour aborder d'autres projets à l'avenir.

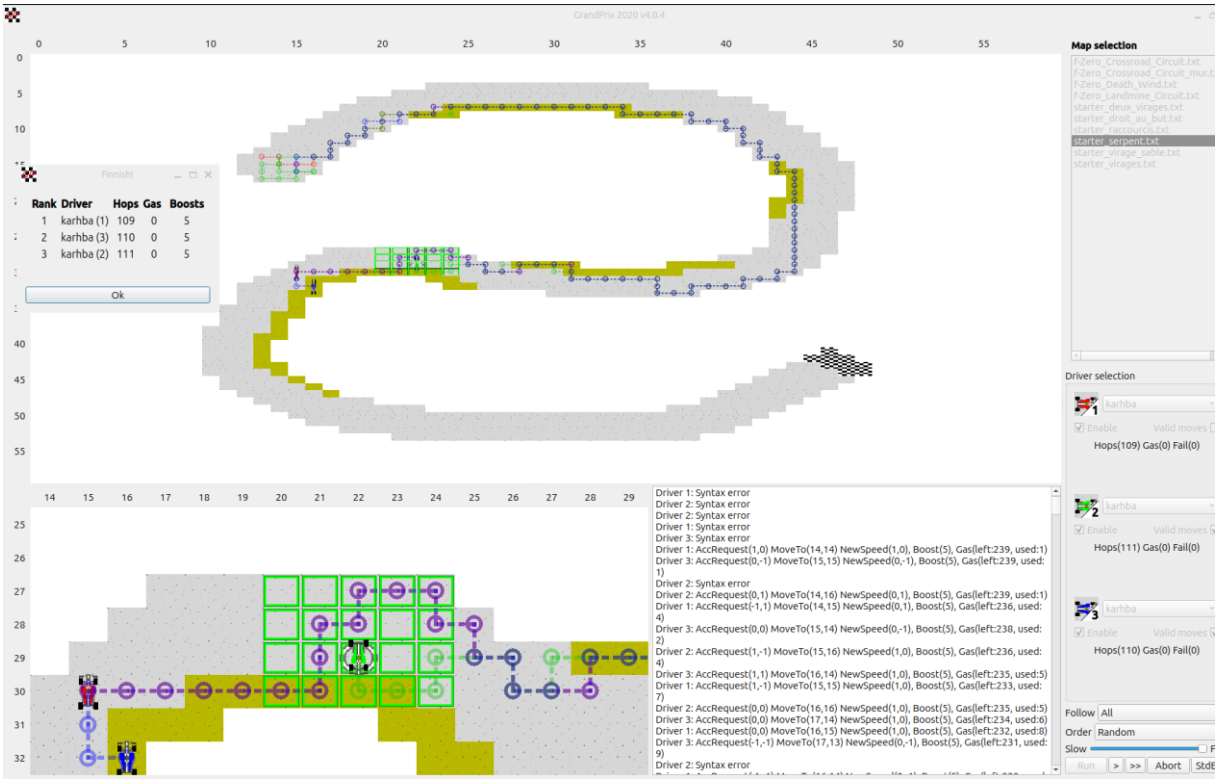

Figure 13 Simulation Finale

TABLE DES FIGURES

TABLE DES TABLEAUX

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