École Publique d'Ingénieurs en 3 ans

Challenge de programmation

Semestre 6 – 2021-2022

GRAND PRIX F1

le 24 mars 2022,

version 1.4

Éric Ziad-Forest

Karim-Eric.Ziad-forest@ensicaen.fr

Sébastien Fourey

Sebastien.Fourey@ensicaen.fr

ENSI CAEN

ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

www.ensicaen.fr

1Crédits

Le gestionnaire de course utilisé dans ce projet a été conçu et programmé par Julien Gosme, puis enrichi pas Gilles Lebrun et Sébastien Fourey.

Table des matières

Table des figures

Liste des tableaux

OBJECTIFS ET ORGANISATION

Introduction

Les objectifs et les modalités de ce challenge de programmation sont détaillées dans ce document. Votre travail donnera aussi lieu à une présentation orale devant un jury composé de deux enseignants, ainsi qu'à la rédaction d'un rapport décrivant la solution technique et algorithmique implémentée.

Votre enseignant référent pour ce projet est :

— Éric Ziad-Forest pour l'organisation générale, le rapport et la soutenance ;

Pour la partie compétition (Si c'est possible.)

Sébastien Fourey

Vous pouvez me contacter courriel, mais si vous pensez que la réponse à votre question est susceptible d'intéresser toutes les équipes, postez plutôt votre question sur l'un des deux forums dédiés sur la plateforme Moodle.

1 Objectifs

Les objectifs de ce challenge de programmation sont, entre autres :

— d'améliorer vos compétences en C, en algorithmique, en ODL, en gestion de projet ;

— de confronter vos talents de codeur avec les autres ;

— de participer à un projet de manière ludique !

2 Organisation

Le travail sera réalisé pour la plupart en binômes (62 étudiants). Pour tenir compte des contraintes dictées par l'organisation d'une compétition équitable d'une durée raisonnable (voir section 7), il a été choisi de répartir les étudiants de la manière suivante

— 28 binômes

— 2 trinômes

pour un total de 30 équipes (5 poules de 6 pilotes chacune).

Les binômes devront être saisis sur la plateforme Moodle, les étudiants qui souhaitent travailler en trinôme devront en faire la demande par courriel auprès de M. Ziad-Forest. Les 3 noms donnés dans 2 premiers courriels seront dès lors les 2 seuls trinômes !

3 Calendrier prévisionnel

Le calendrier prévisionnel est donné dans le tableau 1.

lundi 28 mars 2022	Lancement du challenge
Avant mercredi 30 mars 2022, 18 h 00	Remise du nom de l'équipe ainsi que ceux du binôme (sur Moodle)
Avant mardi 31 mai 2022, 18 h 00	Remise de votre pilote, rapport et diaporama au format PDF (Moodle)
vendredi 3 juin 2022	Matin : soutenance de votre travail devant un jury
	Après-midi : compétition

Tableau 1. Calendrier prévisionnel.

PRÉSENTATION DU CHALLENGE

L'objectif du challenge est de programmer un logiciel de pilotage d'une formule 1 (appelé dans la suite simplement « pilote ») évoluant sur un circuit virtuel. La course en elle-même est régulée par un logiciel qui vous est fourni, le gestionnaire de course (ou « GDC », figure 1). Votre pilote sera obligatoirement écrit en langage C compilable sous linux ubuntu 20.04 sans warning . Il communiquera avec le GDC via ses entrée et sortie standard, en mode texte (cf. section 4.2).

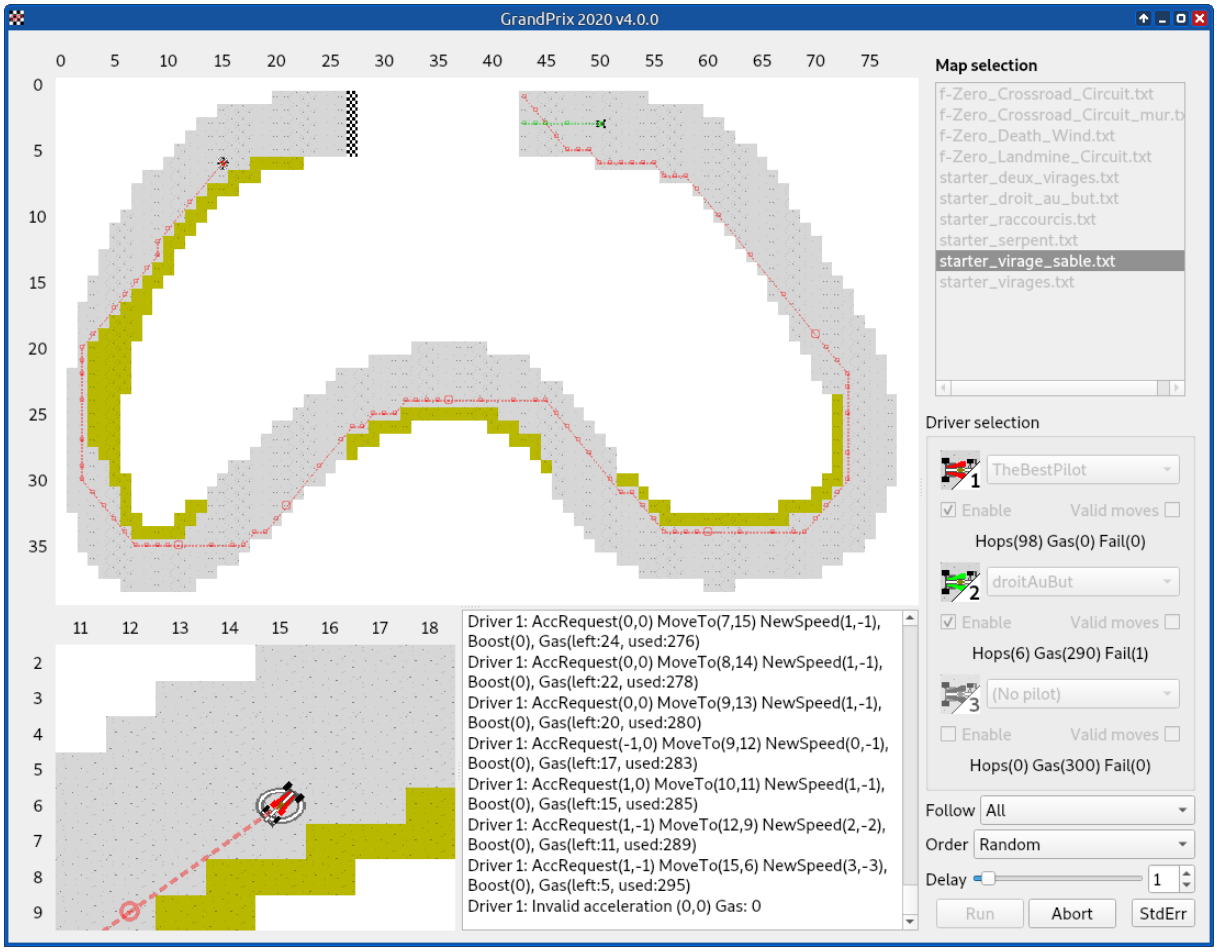


Figure *1. Fenêtre principale du gestionnaire de course.*

4 Les règles du jeu

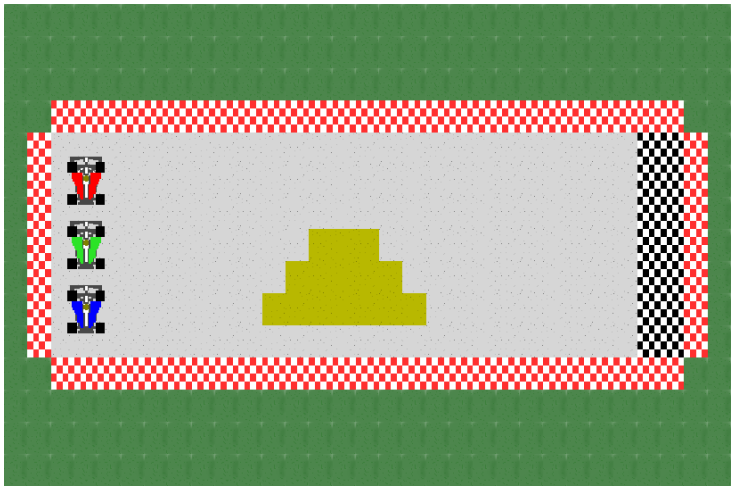
4.1 La piste

Une course se déroule sur une piste donnée dans une carte rectangulaire. La carte est une grille discrète de dimensions (dx, dy). Chaque case de la grille correspond à un des 4 types de terrain suivants :

Nom	Symbole	Description
Piste	#	Zone de piste normale, le comportement y est optimal pour une formule 1.
Sable	~	Ralentit fortement la formule 1.
Hors-piste	.	Terrain inaccessible à une formule 1.
Arrivée	=	L'objectif est d'atteindre ce type de terrain le plus rapidement possible !

Tableau 2. Différents types de cases dans une carte.

Il existe également 3 cases numérotées de 1 à 3 correspondant aux points de départ des 3 pilotes. Ainsi, la carte de la figure 2(a), telle qu'affichée par le GDC, est représentée par le texte de la figure 2(b).



(a)

```
31 15 35
..............................
..............................
..............................
..............................
..##########################==..
..#1########################==..
..##########################==..
..#2#########~~~############==..
..##########~~~~~##########==..
..#3#######~~~~~~~#########==..
..##########################==..
..............................
..............................
..............................
..............................
```

(b)

Figure *2. Carte « droit au but ».*

4.2 Protocole de communication avec le gestionnaire de course

Votre pilote (programmé en C) communique avec le gestionnaire de course par le biais des entrée et sortie standards. Le gestionnaire de course est responsable du démarrage de votre pilote.

1. Il envoie tout d'abord la composition du terrain (ligne par ligne), avec en première ligne sa largeur et sa hauteur puis le carburant disponible en début de course (cf. figure 2(b)).
2. Il fournit ensuite à tour de rôle à chaque pilote sa position (x, y) [1], ainsi que celles des 2 autres pilotes sous la forme :

   3 5\t3 7\t3 9\n (le joueur et les autres)
3. Après réception de cette ligne, chaque pilote doit alors donner son vecteur accélération sous la forme de deux chiffres séparés par un espace. Exemple : 0 1\n
4. Attention : Le pilote dispose d'*une seconde maximum* pour donner sa réponse ! En l'absence de réponse dans le temps imparti, le pilote « passe son tour » autant de fois que nécessaire. La voiture reste immobile tant qu'elle n'a pas donné de réponse, mais sa vitesse est conservée. Elle sera utilisée par le GDC pour appliquer l'accélération finalement demandée.

Remarque 4.1. *Les tours de passage sont aléatoires, mais équitables : l'ordre change à chaque tour. Par exemple, on pourra avoir le déroulement suivant : 1 3 2 ; 3 1 2 ; 2 1 3 ; 2 3 1 ; 3 2 1 ; etc.*

Ce déroulement est résumé dans le diagramme de séquence de la figure 3.

4.3 Les règles d'évolution

— Chaque pilote est caractérisé par une position (x, y) et un vecteur vitesse $\vec{v} = (v_x, v_y)$ qui est stocké par le GDC (et, très certainement aussi par le pilote lui-même).
— Initialement, chaque pilote a un vecteur vitesse nul.
— L'évolution du pilote se fait au travers du vecteur d'accélération $\vec{a} = (\mathrm{d}v_x, \mathrm{d}v_y)$, choisi par celui-ci, et *qui sera ajouté au vecteur vitesse* par le GDC, moyennant le respect des contraintes suivantes :
  — $\mathrm{d}v_x, \mathrm{d}v_y \in \{-1, 0, 1\}$ (soient 9 possibilités, voir figure 4(a)) ;
  — Un pilote peut utiliser 5 « *boosts* », à usage unique, permettant d'avoir une accélération deux fois plus importante, soit $(\mathrm{d}v_x, \mathrm{d}v_y) \in \{-2, -1, 0, 1, 2\}^2 \setminus \{-1, 0, 1\}^2$ (voir figure 4).
  — Le vecteur vitesse ne peut avoir une norme supérieure à 5 (dans le sable, cette limite vaut 1).
  — Si toutes les contraintes précédentes sont respectées, le nouveau vecteur vitesse $\vec{v}_{\text{new}} = \vec{v}_{\text{before}} + \vec{a}$ est ajouté à la position actuelle pour obtenir la nouvelle position du pilote.
  — La nouvelle position de la voiture doit être libre (pas de hors-piste, ni de voiture), ainsi que toutes les positions intermédiaires correspondant au déplacement selon $\vec{v}_{\text{new}}$ (des indications sur ce point sont données en section 4.5).

Si une des conditions limitatives n'est pas respectée (accélération ou vitesse invalide, hors-piste, collision), la voiture est immobilisée et sa vitesse est remise à zéro !

1. soit (colonne, ligne) relativement au coin haut gauche de coordonnées $(0, 0)$.

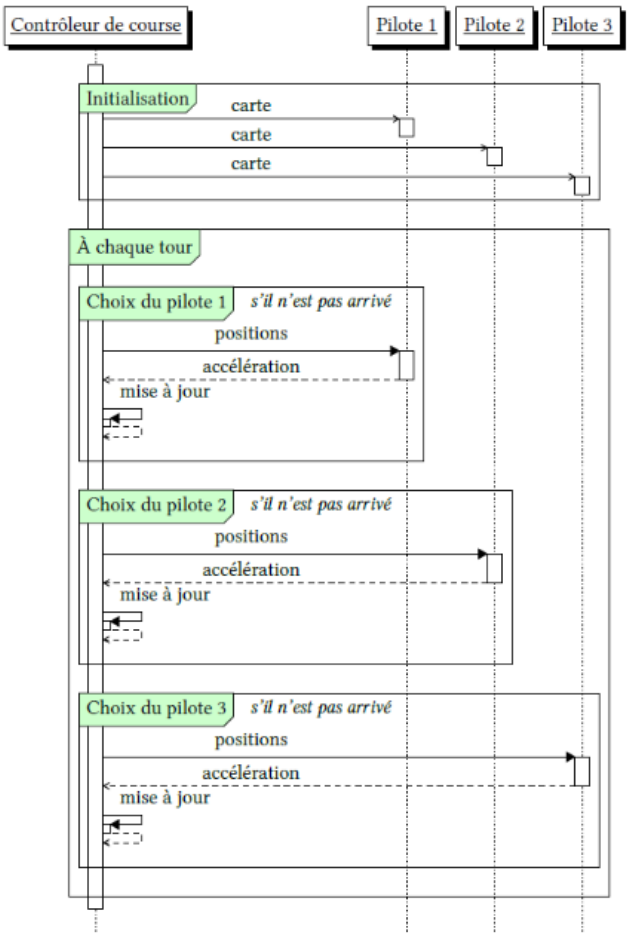
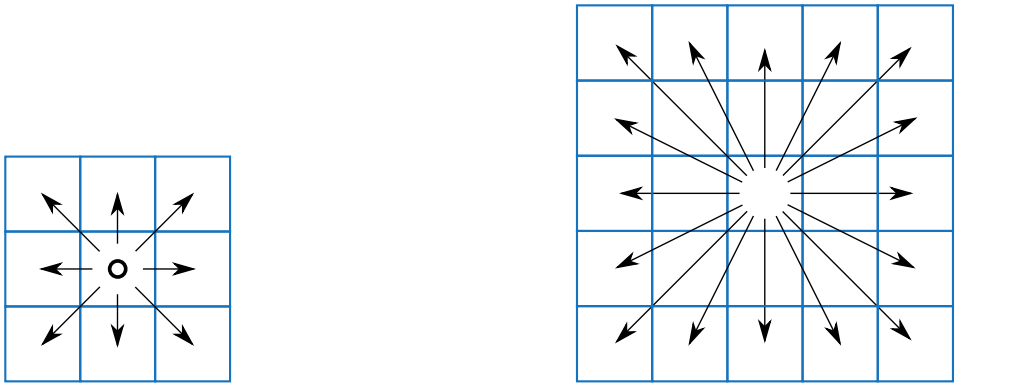


Figure *3. Diagramme de séquence de la gestion de course.*



$(dv_x, dv_y) \in \{-1, 0, 1\} \times \{-1, 0, 1\}$

(a)

$(dv_x, dv_y) \in \{-2, -1, 0, 1, 2\}^2 \setminus \{-1, 0, 1\}^2$

(b)

Figure *4. Vecteurs accélération possibles (a) en temps normal et (b) correspondant à l'utilisation d'un* boost.

4.4 Gestion du carburant

En début de course vous avez une réserve de carburant C qui vous est donnée (valeur 35 sur la première ligne de la carte, en figure 2(b)).

Les règles concernant la consommation sont les suivantes :

— Pour un pilote dont la vitesse courante est $\vec{v} = (v_x, v_y)$ et l'accélération demandée $\vec{a} = (\mathrm{d}v_x, \mathrm{d}v_y)$, la consommation de carburant Δc et calculée comme suit :

$$\Delta c = -\left((\mathrm{d}v_x)^2 + (\mathrm{d}v_y)^2 + \left\lfloor \sqrt{\frac{3(v_x^2 + v_y^2)}{2}} \right\rfloor \right) \tag{1}$$

où $\lfloor \cdot \rfloor$ désigne la partie entière.

— Une pénalisation de -1 unité de carburant est appliquée par tour passé dans le sable.

— Si la valeur de C est négative ou nulle, alors la voiture est arrêtée (vitesse nulle).

— Il incombe à votre pilote de calculer son carburant restant, et bien entendu de planifier au mieux sa consommation.

Remarque 4.2. *La consommation de carburant est calculée, et appliquée, indépendamment du fait que l'accélération demandée par le pilote est valide ou ne l'est pas.*

4.5 Interdiction de couper la route

La trajectoire résultant de l'application du vecteur vitesse à la position courante ne doit rencontrer aucun obstacle (hors-piste ou autre véhicule). Autrement dit, votre pilote ne se télé-porte pas de son point de départ à son point d'arrivée. Le déplacement illustré par la figure 5 est ainsi invalide et provoquera l'arrêt du pilote.

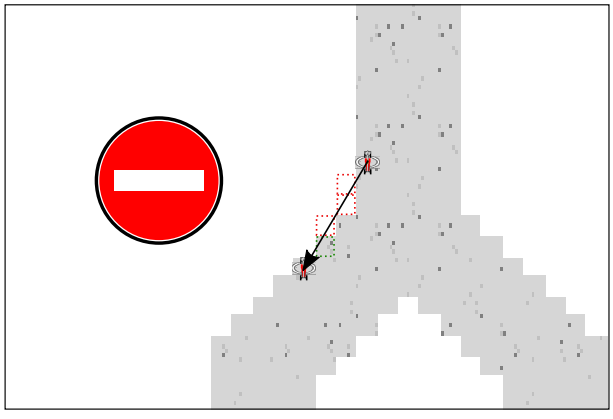


Figure *5. Mouvement interdit en raison de positions intermédiaires hors-piste (en pointillé rouge).*

Pour déterminer les positions intermédiaires correspondant au déplacement entre deux points de coordonnées entières, vous pouvez réutiliser un code qui vous est fourni dans le dossier `./follow_line`.

Ainsi, la fonction `traverse()` du fichier `main.c` vous montre comment utiliser la « bibliothèque » `follow_line` (voir annexe B).

`./follow_line/Makefile`	
`./follow_line/follow_line.h`	
`./follow_line/follow_line.c`	
`./follow_line/main.c`	Programme d'exemple (fonction `traverse()`).

Les plus curieux d'entre vous peuvent aussi aller voir comment le gestionnaire de course gère les collisions dans le deux fichiers suivants :

`./include/gpcontrol.h`
`./src/gpcontrol.cpp`

Remarque 4.3. *Les positions intermédiaires ne sont pas prises en compte pour déterminer si une voiture a* traversé *la ligne d'arrivée. Par conséquent, un pilote est considéré comme ayant atteint le but seulement si sa position finale après un mouvement élémentaire est sur une telle case.*

5 Fonctionnement du gestionnaire de course

5.1 Installation

Le code source du gestionnaire de course est disponible sur la page Moodle consacrée au challenge.

`GrandPrix2021_4.0.2.tar.gz`

Le GDC est écrit en C++ et utilise le cadriciel (ou *framework*) Qt version 5. Avant de compiler le programme, vous devez vous assurer de la présence sur votre machine de la version de développement de cette bibliothèque.

S'il est en théorie possible de compiler le GDC sous Windows, cela vous est déconseillé. En effet, la compétition se déroulera sous Linux et votre pilote devra donc se compiler et s'exécuter de manière fiable dans cet environnement. C'est pourquoi les instructions qui suivent ne concernent que le système Linux.

Pour installer Qt5 sous Linux (Debian, Ubuntu et dérivées), utilisez la commande suivante :

```
$ sudo apt install qt5-default
$ sudo apt install qtcreator    # IDE (facultatif)
```

Vous pouvez ensuite compiler le gestionnaire de course.

```
$ cd GrandPrix2021_4.0.2/
$ qmake     # (qmake-qt5 sous Fedora)
$ make      # Produit l'exécutable ./GrandPrix
```

5.2 Cartes

Les cartes utilisables par le GDC se trouvent dans le dossier `./tracks` de l'archive.

Au total, 10 cartes vous sont fournies :

`./tracks/f-Zero_Crossroad_Circuit_mur.txt`	
`./tracks/f-Zero_Crossroad_Circuit.txt`	
`./tracks/f-Zero_Death_Wind.txt`	
`./tracks/f-Zero_Landmine_Circuit.txt`	
`./tracks/starter_deux_virages.txt`	
`./tracks/starter_droit_au_but.txt`	Très facile
`./tracks/starter_raccourcis.txt`	
`./tracks/starter_serpent.txt`	
`./tracks/starter_virage_sable.txt`	Difficile
`./tracks/starter_virages.txt`	Normale

À titre d'exemple, la figure 6 représente la carte `starter_virage_sable.txt`.

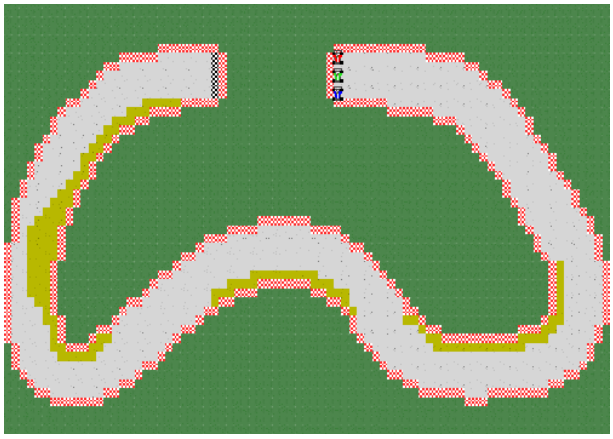


Figure *6. La carte « starter virage sable ».*

5.3 Pilotes

Les programmes des pilotes doivent se trouver dans le dossier `./drivers/` pour pouvoir être lancés par le GDC. La section 5.4 vous donne les instructions pour compiler un pilote minimal de test.

5.4 Autres fichiers fournis : le pilote « Droit au but »

Le code source d'un pilote naïf vous est fourni dans le dossier `test_pilot`.

```
./test_pilot/Makefile
./test_pilot/droitAuBut.c
```

Notez que, du fait de la présence à la racine d'un fichier Makefile (généré par le programme qmake, cf. section 5.1), le code du pilote fourni et son Makefile se trouvent dans un sous-dossier. Prenez la peine de consulter le Makefile. Pour compiler ce pilote, vous pouvez simplement lancer la commande :

```
$ make -C ./test_pilot install
```

Un extrait du pilote droitAuBut est donné en annexe A. On peut faire plusieurs remarques sur ce pilote :

— Il se contente d'accélérer constamment en fournissant au GDC un vecteur accélération $\vec{a} = (1, 0)$ (voir le fichier de log de ce pilote, figure 7 dans la section 6.2). Il finit donc par dépasser la vitesse limite et se voit donc affecter une vitesse nulle à l'issue du *round* 6 (position inchangée au *round* 7).

— Il ne mémorise ni la carte, ni sa position. Il est donc notamment incapable de calculer sa vitesse réelle ainsi que l'état de la piste, ce qui rend faux son calcul de consommation de carburant (d'où les valeurs négatives aberrantes à partir du *round* 7).

6 Comment commencer à travailler ?

6.1 Dossier de votre pilote

Commencez par copier le contenu du dossier `./test_pilot` dans un dossier `./LesFousDuVolants` si tel est le nom de votre équipe. Il vous suffira ensuite d'éditer et d'adapter le Makefile pour ce nom.

C'est aussi ce dossier (nettoyé de tous fichiers inutiles) que vous remettrez sous forme d'archive sur la plateforme Moodle en temps voulu.

Votre dossier de travail devant contenir tous les fichiers nécessaires à la compilation de votre pilote, vous pouvez aussi recopier les fichiers `follow_line.h` et `follow_line.cpp`.

6.2 Débogage

Cette section comporte des informations importantes pour vous aider à déboguer votre pilote.

6.2.1 Traces d'exécutions

Afin de déboguer votre programme, deux solutions s'offrent à vous :

Solution 1 Ouvrir dans votre programme un fichier texte en écriture et y écrire les informations pertinentes sur l'état et les actions de votre pilote.

Solution 2 Envoyer toutes les informations évoquées précédemment directement dans la *sortie d'erreur* (`stderr`) de votre programme. Le gestionnaire enregistre pour vous ces sorties dans un fichier

nommé `stderr_driver_N.txt` (où `N` est le numéro du pilote). De plus, vous pouvez consulter en temps-réel la sortie d'erreur des pilotes dans une fenêtre dédiée (figure 8) ouverte à l'aide du bouton « StdErr » situé en bas à droite de la fenêtre du GDC (figure 1).

Notez que ces solutions ne sont pas exclusives. Il est en effet envisageable d'écrire des informations dans plusieurs fichiers.

Le programme `droiAuBut` utilise la seconde méthode pour *logger* la carte reçue, les informations de position reçues du GDC ainsi que les accélérations demandées (voir figure 7, dans laquelle la carte n'apparaît pas par souci de concision). Ainsi, il est possible de constater par exemple que l'accélération demandée au *round* 6 a eu pour effet d'arrêter la voiture (position inchangée au round suivant). En effet, le pilote demande à chaque tour une accélération de $(1, 0)$ ce qui finit par augmenter sa vitesse en dehors des valeurs limites (section 4.3). Le GDC immobilise donc le véhicule.

```
=== >Map< ===
Size 32 x 16
Gas at start 35

(MAP DUMP HERE)

=== Race start ===
=== ROUND 1
    Positions: Me(3,5)  A(3,7), B(3,9)
    Action: 1 0   Gas remaining: 34
=== ROUND 2
    Positions: Me(4,5)  A(3,7), B(3,9)
    Action: 1 0   Gas remaining: 32
=== ROUND 3
    Positions: Me(6,5)  A(3,7), B(3,9)
    Action: 1 0   Gas remaining: 28
=== ROUND 4
    Positions: Me(9,5)  A(3,7), B(3,9)
    Action: 1 0   Gas remaining: 23
=== ROUND 5
    Positions: Me(13,5)  A(3,7), B(3,9)
    Action: 1 0   Gas remaining: 16
=== ROUND 6
    Positions: Me(18,5)  A(3,7), B(3,9)
    Action: 1 0   Gas remaining: 8
=== ROUND 7
    Positions: Me(18,5)  A(3,7), B(3,9)
    Action: 1 0   Gas remaining: -2
=== ROUND 8
    Positions: Me(19,5)  A(3,7), B(3,9)
    Action: 1 0   Gas remaining: -13
=== ROUND 9
    Positions: Me(21,5)  A(3,7), B(3,9)
    Action: 1 0   Gas remaining: -26
=== ROUND 10
    Positions: Me(24,5)  A(3,7), B(3,9)
    Action: 1 0   Gas remaining: -40
```

Figure *7. Log (partiel) du pilote droitAuBut.*

6.2.2 Gestion des « crashs »

En cas d'erreur de segmentation (*SEGFAULT*) et autres plaisirs liés à la mise au point, il vous est impossible de lancer le débogueur `gdb`. En effet, seul le GDC peut lancer votre programme. Une solution

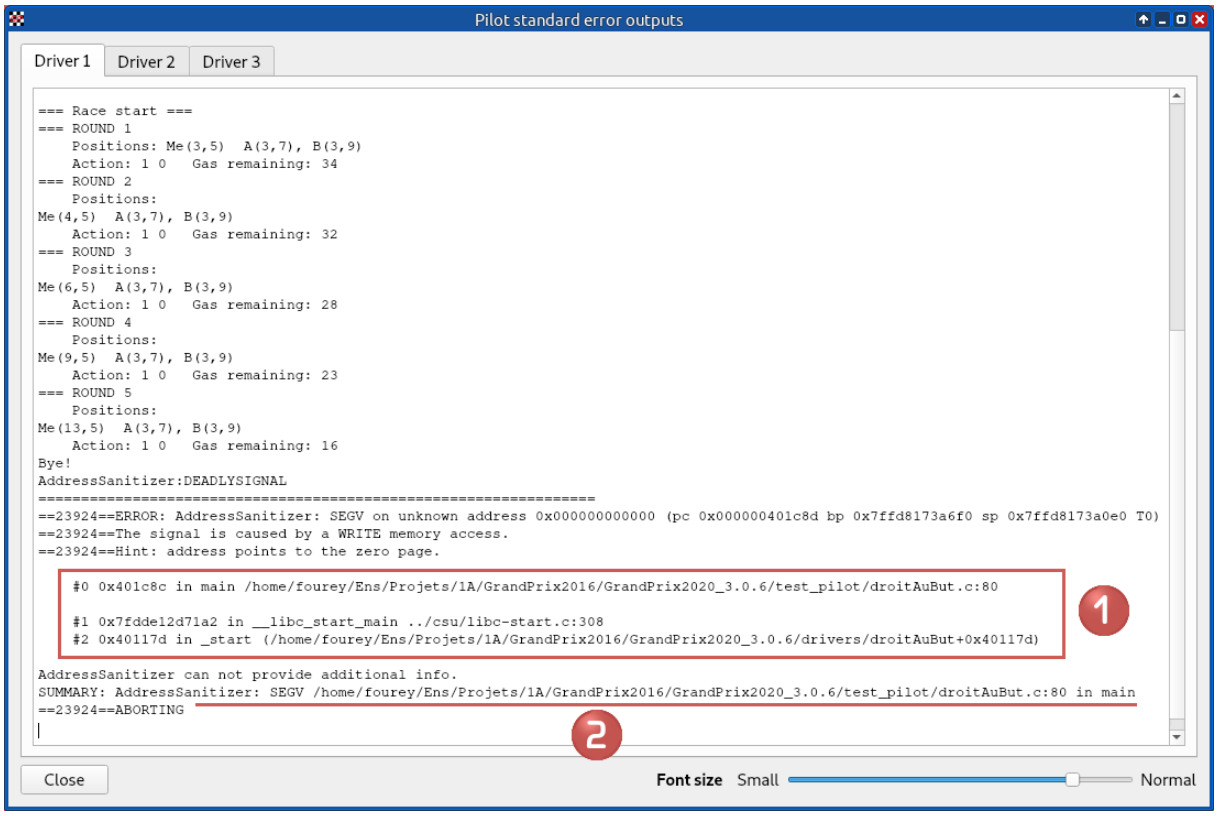


Figure *8. Fenêtre des sorties d'erreurs des pilotes.*

existe cependant pour vous aider à trouver l'origine de l'erreur.

Pour peu que vous ayez installé la bibliothèque `libasan` sur votre système, il est alors possible de lancer `gcc` avec l'option `-fsanitize=address`, ce qui ajoute aux fichiers objets et exécutables de quoi détecter automatiquement certaines erreurs et afficher des informations sur ces erreurs.

Le `Makefile` du pilote `droitAuBut` est prévu à cet effet. Commencez par installer le paquet `libasan` dont la version est donnée dans le tableau 3 en fonction de la version de votre compilateur :

```
$ gcc --version
gcc (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0
[...]
$ sudo apt install libasan4   # Puisque gcc est en version 7 !
```

Compilateur	Paquet à installer
gcc-4.8	libasan0
gcc-5	libasan2
gcc-6	libasan3
gcc-7	libasan4
gcc-8	libasan5

Tableau 3. Version de libasan associée à gcc.

Pour le test, modifiez le pilote `droitAuBut` en y insérant le code suivant pour provoquer une *Segmentation Fault* au 5e tour (juste après le `fflush` de la ligne 43 de l'annexe A).

Extrait de code: SegFault

```
if (round > 4) { /* Force a segfault for testing purpose */
  int * p = NULL;
  fprintf(stderr, "Good Bye!\n");
  fflush(stderr);
  *p = 0;
 }
```

Ensuite, compilez le pilote en activant l'option *SANITIZE* :

```
$ make -C ./test_pilot distclean
$ make SANITIZE=on -C ./test_pilot install
```

Enfin, lancez une course avec ce pilote sur le circuit `starter_droit_au_but`. Vous devriez voir apparaître dans la fenêtre de sortie d'erreur (figure 8) et dans le fichier `stderr_driver1.txt` les informations récoltées par `sanitize` suite à la *Segmentation Fault* du pilote numéro 1. Vous pouvez ainsi retrouver sur la figure 8, entre autres informations :

1. La pile d'appels de fonctions au moment où l'erreur a été détectée ;
2. Un résumé de l'erreur et sa localisation.

6.2.3 Avant de rendre votre pilote

Attention : Les sorties faites sur la sortie d'erreur ou dans un fichier peuvent ralentir votre pilote. De même, la compilation avec l'option `-fsanitize=address` diminue sensiblement les performances de l'exécutable. Avant de rendre votre pilote, pensez à désactiver l'enregistrement de *logs* et l'option `sanitize`. La compétition sera plus vivante si un maximum de pilotes a des temps de réponses raisonnables !

COMPÉTITION

La compétition se déroulera dans le grand amphi du bâtiment A. Les 4 meilleures équipes se verront attribuer des points de bonification sur leur note (voir section 13).

7 Formules possibles

La formule retenue est on ne peut plus classique pour une compétition : d'abord des rencontres en poules, puis une phase d'élimination directe. Cette section présente le choix qui a été effectué concernant la répartition en poules.

Les contraintes de la compétition pour la répartition en poules m'amène à établir les contraintes suivantes :

- Les 30 pilotes seront répartis de façon aléatoire en 6 poules de 5 pilotes en début de compétition.
- Une course oppose trois pilotes.
- Chaque pilote effectuera 3 courses dans sa poule
- Il y aura 5 courses par poule
- Dans une poule, chaque pilote rencontre soit 1 ou 2 fois chacun des autres pilotes de la poule.

  De plus, on se donne les contraintes suivantes :
  - tous les pilotes font le même nombre de courses en poule,
  - minimiser le nombre total de courses pour une durée raisonnable de compétition ;
  - maximiser le nombre de binômes et minimiser le nombre de trinômes.

  Avec 60 étudiants, les formules données dans le tableau 4 restent envisageables. Il a été choisi de privilégier le nombre total de courses, et la simplicité d'organisation de la phase suivante d'élimination directe la formule à 30 équipes a donc été retenue.

				Par poules						
Monômes	Binômes	Trinômes	Équipes	A	B	C	D	E	F	Courses
0	28	2	30	5	5	5	5	5	5	30

Tableau 4. Répartitions possibles en poules selon les contraintes énoncées.

8 Déroulement de la compétition

Toutes les courses de poules se dérouleront sur la même carte, choisie parmi les cartes présentes dans l'archive qui vous est fournie. La phase éliminatoire ainsi que la finale se dérouleront sur des cartes inconnues de vos pilotes.

8.1 Règlement de la course

Quelques règles seront appliquées durant la compétition.

— Quand un pilote a réussi à quitter la ligne de départ mais n'est pas parvenu à l'arrivée (panne de carburant, boucle, crash), son classement dans la course peut être calculé en fonction de sa distance à l'arrivée.

— Si un pilote abuse de manière flagrante de son temps de calcul en ralentissant énormément la course, tout en étant loin de la tête de course, il pourra être arrêté « artificiellement ».

— La compétition se déroule sur un PC aux performances voisines de celles des machines de salles de TP du bâtiment E. Sans plus de précision sur ces performances !

8.2 Courses en poules

Les équipes seront affectées dans les poules de façon aléatoire (devant vous) le jour de la compétition. Comme indiqué précédemment, chaque pilote effectue 3 courses durant cette phase. Les courses s'enchaînent en alternant les poules à chaque course. Le classement en poule est basé sur les points remportés par les pilotes à chaque course (tableau 5).

Position	Points
1er	4
2e	2
3e	0

Tableau 5. Points remportés à l'issue d'une course.

Remarque 8.1. *Un pilote qui ne quitte pas la ligne de départ ne remporte aucun point, même s'il est « virtuellement » second* ex-æquo*. Par exemple si un seul pilote est arrivé au but et les deux autres n'ont pas quitté la ligne de départ.*

8.3 Phase éliminatoire

Les 2 premières équipes de chaque poule vont directement en huitième de finales un tirage aléatoire de 2 pilotes parmi les 6 premiers ainsi que 2 pilotes parmi les 6 deuxièmes rencontreront chacune une des 4 équipes repêchées parmi les meilleurs 3ème, dans le cas d'ex æquo une course sera organisé. sinon pour les autres équipes de huitièmes de finale, le premier d'une poule y rencontre le deuxième d'une autre poule.
S'ensuivent dès lors les quart, demi-finales puis une finale.

8.4 Finale

La finale entre les deux meilleures équipes se joue en 3 courses au maximum (2 courses gagnantes). La petite finale, en une seule course, départage les pilotes qui seront classés 3^{e} et 4^{e}, déterminant ainsi le classement de l'édition 2021 du Grand Prix. . .

CONSIGNES

Vous trouvez dans cette partie des consignes que vous n'êtes pas censés ignorer. Prenez donc la peine de les lire attentivement.

9 Consignes à respecter

— Vous devrez déposer en temps voulu sur la plateforme Moodle :
  — votre rapport au format PDF ;
  — votre présentation au format PDF ;
  — une archive de votre pilote.
— Aucun retard ne sera toléré. Chaque partie non rendue à temps vous vaudra la note de 0 pour la partie correspondante.
— L'archive de votre pilote devra contenir tous les fichiers nécessaires (et seulement eux) à la compilation de l'exécutable sur cybele ou un PC du bâtiment E (architecture Linux de référence pour le challenge). Une vérification s'impose avant de soumettre votre archive.
— Le dossier de votre pilote devra comporter un fichier Makefile correct.
— Dans ce fichier Makefile, les drapeaux ci-dessous devront être activés, et tous les fichiers sources devront être compilés sans avertissement (*warning*).

```
-ansi -pedantic -Wall -Wextra
```

— Un code qui ne compilerait pas verrait le pilote être déclaré forfait, avec application d'une pénalité sur la note obtenue par l'équipe.

10 Rapport

Le rapport devra présenter votre travail et la solution algorithmique apportée par votre pilote.
Le document devra respecter la charte graphique de l'école (modèle fourni sur la plateforme Moodle : Communication, Modèles de documents). De plus, vous veillerez à tenir compte de tous les conseils qui vous auront été donnés, d'ici là si tout va bien, en cours et TD de l'enseignement « Communiquer dans un environnement professionnel ».
Quelques écueils à éviter :
— consacrer trop d'espace à la description du sujet ;

— inclure de nombreux extraits de code ;
— détailler et commenter des structures de données basiques ;
— inclure *N* images de tailles conséquentes des logos de tous les outils que vous avez utilisés (préférez une présentation critique et argumentée de vos choix d'outils) ;
— etc.

11 Soutenance

La présentation se déroulera de la manière suivante :
— installation rapide (pas d'ordinateur personnel, les PDF seront déjà sur le PC de la salle) ;
— temps de parole de 10 minutes à répartir entre les membres de l'équipe ;
— remarques et questions du jury pendant 5 minutes.
L'objectif est de convaincre le jury que votre pilote a été bien conçu. Là encore, des écueils sont à éviter. Vous trouverez sur Moodle un document de consignes à suivre pour préparer votre soutenance. Pensez qu'il s'agit bien d'un exercice de présentation.

12 Code source de votre pilote

Des consignes très précises concernant l'archive que vous devrez remettre sur Moodle vous seront données par courrier électronique en temps voulu. Le respect de ces consignes permettra une préparation de la compétition dans les meilleures conditions et dans un temps raisonnable (donc une date de rendu raisonnable elle aussi).
Attention, relisez la consigne donnée en section 6.2.3 avant de soumettre votre archive.

13 Évaluation

Votre note finale tiendra compte des points suivants :
— qualité de la présentation orale ;
— qualité du support de présentation ;
— qualité du rapport ;
— qualité du code source du pilote ;
— solution algorithmique (telle que présentée à l'écrit, à l'oral et soigneusement implémentée) ;

— application d'une bonification aux 4 meilleures équipes de la compétition selon le barème donné dans le tableau 6.

Classement	Bonification
1	+ 2 pts
2	+ 1,5 pts
3	+ 1 pt
4	+ 0,5 pt

Tableau 6. Barème de bonification.

14 Conclusion

Bon courage à tous. . .

et que les meilleurs gagnent !

Annexes

A Programme principal du pilote « droit au but »

Fichier: ./test_pilot/droitAuBut.c

```c
int main() {
  int row;
  int width, height;
  int gasLevel;
  int boosts = BOOSTS_AT_START;
  int round = 0;
  int accelerationX = 1, accelerationY = 0;
  int speedX = 0, speedY = 0;
  char action[100];
  char line_buffer[MAX_LINE_LENGTH];
  boosts = boosts;                          /* Prevent warning "unused variable" */
  fgets(line_buffer, MAX_LINE_LENGTH, stdin);      /* Read gas level at Start */
  sscanf(line_buffer, "%d %d %d", &width, &height, &gasLevel);
  fprintf(stderr, "=== >Map< ===\n");
  fprintf(stderr, "Size %d x %d\n", width, height);
  fprintf(stderr, "Gas at start %d \n\n", gasLevel);
  for (row = 0; row < height; ++row) {        /* Read map data, line per line */
    fgets(line_buffer, MAX_LINE_LENGTH, stdin);
    fputs(line_buffer, stderr);
  }
  fflush(stderr);
  fprintf(stderr, "\n=== Race start ===\n");
  while (!feof(stdin)) {
    int myX, myY, secondX, secondY, thirdX, thirdY;
    round++;
    fprintf(stderr, "=== ROUND %d\n", round);
    fflush(stderr);
    fgets(line_buffer, MAX_LINE_LENGTH, stdin);   /* Read positions of pilots */
    sscanf(line_buffer, "%d %d %d %d %d %d",
           &myX, &myY, &secondX, &secondY, &thirdX, &thirdY);
    fprintf(stderr, "    Positions: Me(%d,%d)  A(%d,%d), B(%d,%d)\n",
            myX, myY, secondX, secondY, thirdX, thirdY);
    fflush(stderr);
    /* Gas consumption cannot be accurate here. */
    gasLevel += gasConsumption(accelerationX, accelerationY, speedX, speedY, 0);
    speedX += accelerationX;
    speedY += accelerationY;
    /* Write the acceleration request to the race manager (stdout). */
    sprintf(action, "%d %d", accelerationX, accelerationY);
    fprintf(stdout, "%s", action);
    fflush(stdout);                             /* CAUTION : This is necessary  */
    fprintf(stderr, "    Action: %s   Gas remaining: %d\n", action, gasLevel);
    fflush(stderr);
  }
  return EXIT_SUCCESS;
}
```

B Parcours des positions intermédiaires d'un déplacement

Le code ci-dessous illustre la façon dont les fonctions `initLine` et `nextPoint` (fournies) peuvent être utilisées pour parcourir les positions intermédiaires d'une ligne droite. En adaptant ce code, vous serez en mesure de tester qu'un déplacement est valide (ni collision, ni sortie de piste).

Fichier: ./follow_line/main.c

```c
#include <stdio.h>
#include "follow_line.h"

void traverse(Pos2Dint start,  Pos2Dint stop)
{
  InfoLine vline;
  Pos2Dint p;
  initLine(start.x, start.y, stop.x, stop.y, &vline);
  while (nextPoint(&vline, &p, +1) > 0) {
    if (p.x == start.x && p.y == start.y) {
      /* We suppose that the start position is not worth visiting! */
      continue;
    }
    printf("Visiting point (%d,%d)\n", p.x, p.y);
  }
}

int main()
{
  Pos2Dint start = {0, 0};
  Pos2Dint stop = {10, 5};
  traverse(start, stop);
  return 0;
}
```

ENSI CAEN

ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

École Publique d'Ingénieurs en 3 ans

6 boulevard Maréchal Juin, CS 4505

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