

**Rapport de projet :**

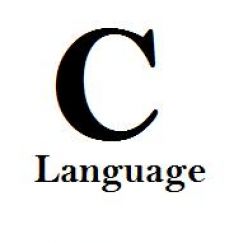
**Simulation d’une course de NASCAR**

**UV LO41**

**Département Génie informatique**

**Automne 2012**

**G**





Clément GHNASSIA

Valdes FOADEY

PLAN DU RAPPORT :

PLAN DU RAPPORT :

INTRODUCTION 3

1. PHASE ANALYSE 4
   1. Contexte du projet 4
   2. Approche 5
   3. Réseau de pétri 6
2. PHASE DE CONCEPTION 10
   1. Gestion des voitures 10
   2. Gestion du circuit 13
   3. Gestion des directeurs de course 14
   4. Gestions des stands 14
   5. Génération des accidents 15
   6. Gestion de la course 15
   7. Gestion des résultats 15
   8. Gestion des commandes utilisateurs et de l’affichage 16
3. AMELIORATIONS POSSSIBLES 17
4. UTILISATION 18
   1. PRÉREQUIS 18
   2. SIMULATION 18

CONCLUSION 19

INTRODUCTION

Comme les années précédentes, les étudiants ayant suivi l’Unité de Valeur Logiciel 41 (LO41) sont appelés à réaliser un projet qui leur est proposé par le responsable de la matière, pour leurs permettre de mettre en pratique les connaissances reçues au cours du semestre. C’est dans cette optique qu’il nous a été proposé de sujet suivant « **simuler le comportement de véhicules lors d’une course de Nascar** ». Il s’agit d’un sujet qui doit être développé dans un domaine et un contexte qui nous est plus ou moins connu et qui exige l’utilisation des notions et concepts vus en cours.

Afin de vous présenter de façon plus explicite le projet que nous avons réalisé et la démarche entreprise pour mener à terme ce projet, notre rapport s’articulera comme suit :

Dans un premier temps, nous vous présenterons la phase d’analyse du fonctionnement globale de l’application en mettant des points sur les éléments principaux : voiture, circuit, segment, etc…ainsi que des interactions entre eux.

Ensuite nous présenterons la phase de conception de l’application proprement dite sans oublier de vous faire part des difficultés rencontrées et des améliorations possibles du code

Enfin dans une dernière partie, nous vous présenterons brièvement les possibilités de l’utilisateur de jouer sur différents paramètres et comment accéder aux informations durant la course.

1. PHASE D’ANALYSE :

Cette partie consiste à rappeler le sujet qui nous a été proposé et les modalités à respecter lors du développement de notre application.

* 1. Contexte du projet :

Le projet consiste à développer une application de simulation d’une course de Nascar. La course doit se dérouler en deux phases : une phase de qualification et la course proprement dite. Le but est donc de simuler le comportement de 42 véhicules lors d’une course.

La piste devra être constituée de deux anneaux concentriques permettant les véhicules de se dépasser et d’un segment de piste facilitant l’entrée aux stands (puits). Chaque équipe est constituée de 2 véhicules.

Déroulement de la course :

1. Qualification

Les véhicules doivent réaliser le meilleur temps lors de cette séance afin d’obtenir une place de choix sur la grille de départ. Durant cette séance, seule une voiture par équipe est autorisée à utiliser la piste soit 21 véhicules à la fois. Chaque voiture a la possibilité de réaliser 3 tours. Seul le meilleur des tours est comptabilisé.

1. Course

Les voitures sont tout d'abord placées dans la ligne des puits, derrière la voiture officielle ("pace car") dans l'ordre établi lors des qualifications. La "pole position" se trouve à l'intérieur de la piste, c'est à dire à gauche. Le deuxième se situe à côté du premier, à l'extérieur. Les suivants sont placés selon le même principe. Après les cérémonies d'avant départ, le traditionnel "Messieurs, démarrez vos moteurs !" est envoyé par l'annonceur, le drapeau jaune est agité, la voiture officielle démarre et le groupe s'ébranle en conservant leurs positions. Contrairement à la Formule 1, où les voitures démarrent de la grille de départ à la lumière verte, le départ d'une course d'une série NASCAR s'effectue lancé, après trois tours de chauffe à faible vitesse sous drapeau jaune. Après ces tours, la voiture officielle sort, et le drapeau vert est donné à la sortie du dernier virage avant la ligne d'arrivée... La course est officiellement lancée, dans le rugissement instantané de 42 monstres de 750 chevaux. Durant la course lors d'un incident, la décision de sortir le drapeau jaune est prise par le(s) directeur(s) de course; en règle générale, si l'incident est mineur et que la voiture incriminée peut repartir sans danger pour la sécurité, le vert reste présent. Par contre, s'il y a des débris ou des liquides sur la piste, ou qu'une voiture reste immobilisée, le drapeau jaune est immédiatement sorti et la voiture officielle prend le contrôle de la course à la ligne de départ. Par contre, il est important de noter que la course reste active jusqu'au passage par la ligne d'arrivée... Chaque conducteur ayant échappé à l'incident peut encore essayer de gagner quelques places jusqu'à ce point. Dès la ligne d'arrivée franchie, les conducteurs doivent rester en formation sur une file derrière la voiture officielle jusqu'au redémarrage, qui se fera par la sortie du drapeau vert à l'entrée de la ligne droite précédant la ligne de départ. Les pilotes profitent généralement d'un drapeau jaune pour retourner aux puits et y faire plein d'essence et/ou le changement de pneus. Dans notre simulation les véhicules changeront par deux fois leurs pneus durant la course en partant du principe qu’un train de pneus ne peut couvrir que 70% de la distance du challenge, le plein d’essence ne couvrant que 60 % du parcours.

Cette compétition est régie par des directeurs de course par l’utilisation de drapeaux selon des règles précises :

* Jaune: C'est le drapeau d'incident de course, impliquant un ralentissement. La course passe sous le contrôle de la voiture officielle qui gouverne la vitesse maximum.
* Rouge: C'est l'arrêt complet de la course, qui peut être dû à un grave accident, aux mauvaises conditions atmosphériques, etc... Cet arrêt peut être temporaire ou définitif.
* Bleu à diagonale jaune ou orange: Il est destiné aux conducteurs ayant plus d'un tour de retard pour leur signaler la présence d'une des voitures de tête et leur demander de céder le passage.
* Noir: Il est associé à un tableau indicateur placé sous la plateforme du porte-drapeau, et qui indique le numéro de la voiture à qui le drapeau noir est destiné. Les causes d'un drapeau noir sont nombreuses. Il peut signaler au conducteur que sa voiture perd un de ses liquides, fume, ou roule trop lentement. Il peut aussi indiquer une pénalité, suite à un manquement aux règles de course. Dans tous les cas, le conducteur doit se rendre immédiatement à son puits, et éventuellement y purger sa pénalité.
* Noir avec croix blanche: c'est le dernier drapeau qu'un conducteur désire voir, car il signifie au conducteur pointé qu'il a refusé d'obtempérer à un drapeau noir et que ses tours cessent d'être comptabilisés jusqu'à ce qu'il se rende au puits.
* Blanc: Il n'apparait qu'une seule fois au cours d'une course, et indique le début du dernier tour de la voiture de tête.

Qu’avons-nous donc compris de ce qui nous est demandé ?

* 1. Approche

Il nous est donc demandé de faire appel aux notions et concepts vus en cours, travaux dirigés et travaux pratiques pour modéliser et simuler des voitures, un circuit sur lequel se déroulera la course, des stands pour s’occuper des voitures, des drapeaux pour signaler des évènements, des directeurs de course pour lever ses drapeaux, des condition de course (météorologique, accidents, ...). Pour le faire, nous avons eu à trouver des analogies avec les concepts et outils informatiques et mathématiques dont nous disposons. Ainsi, nous avons :

* Voiture :

Les voitures seront représentées comme des threads. Chaque voiture aura :

* une vitesse pouvant évoluer entre une valeur min et une valeur max de façon aléatoire, mais de façon continue en respectant son accélération;
* une accélération pouvant également évoluer d’une valeur min a une valeur max ;
* une quantité d’essence et un niveau de pneu limites en dessous desquels elle demande à entrer dans son stand;
* une possibilité de freiner grâce à une capacité de freinage;
* une position sur la piste (le circuit) ;
* un état  qui peut être:
* critique : une voiture est hors service ;
* mauvais : la voiture a été accidentée, mais peu rentrer (à faible allure) au stand
* moyen : accident qui n’est pas fatale pour la voiture – celle ci doit toute de même rentrer au stand ;
* normale : la voiture est dans un état normal et dispose de toutes ses capacités ;
* le circuit :

Le circuit sera considéré comme étant un tableau en trois dimension accessibles par les différents threads. Il s’agira d’un tableau de structure, chacune des cases permettant la synchronisation des voitures en fonction de leur position. Le circuit sera protégé puisqu’il s’agit bien évidemment d’une zone critique.

* Les directeurs :

Des threads qui agiront sur une portion de circuit et détecteront les accidents et pannes des voitures présents sur leur segment, ou tout autre évènement qui nécessite d’informer les voiture ou de limiter leur vitesse.

* Les drapeaux :

Les drapeaux seront implémentés en signaux utilisés par les directeurs de course en fonction de chaque évènement.

* Les stands :

Eux aussi seront implémentés en threads. Ils communiqueront avec les voitures à l’aide de file de message. Ils prendront des décisions concernant l’enter ou non d’une voiture dans le stand en fonction de la priorité que celle-ci présente dans son message.

* Les accidents :

Des accidents pourront intervenir lors de la course mais chaque accident devra être générer aléatoirement. En s’assurant que les accidents les moins graves puissent arriver moins fréquemment que les accidents légers.

* 1. Réseau de Petri
     1. Enter dans les stands :

Une voiture ne demande à entrer dans son stand que si elle est soit en dessous du niveau mini d’essence autorisé, ou en dessous du niveau minimum de pneu autorisé, ou encore si elle a eu un accident. Sa demande sera acceptée ou non selon le message que le stand lui retournera. En sortant du stand elle reprend son état normal. Ceci est modélisé par le réseau de Petri ci-dessous :

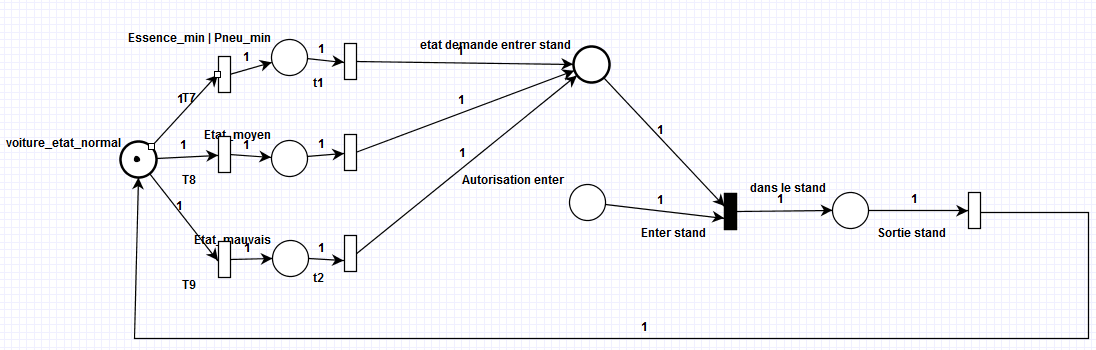
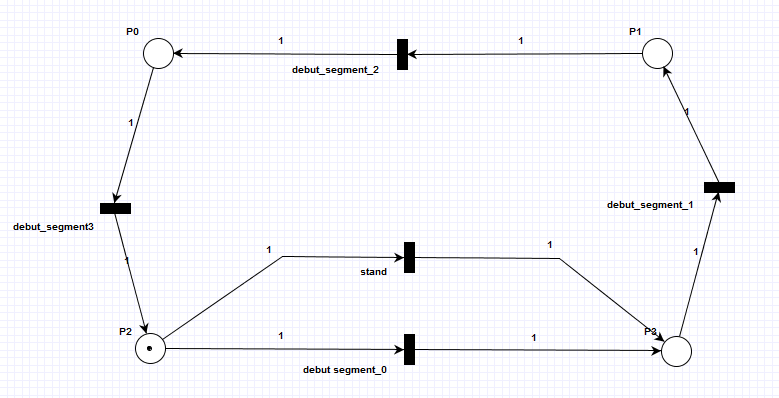


Figure1 : Entrer au stand

* + 1. Evolution sur le circuit :

Sur le circuit, les voitures évolueront de case en case avec la possibilité de se déplacer d’une ligne à l’autre. La ligne la plus prioritaire étant celle la plus à l’intérieur, puisque en étant sur ce dernier, la distance à parcourir est plus petite, avec la possibilité d’aller au stand quand le besoin se présente mais seulement à un endroit bien précis.



|  |  |  |
| --- | --- | --- |
|  | n’ |  |
| n-1 | n | n+1 |
|  | n’’ |  |

|  |  |  |
| --- | --- | --- |
| 0 | n’ |  |
| 1 | n | n+1 |
| 2 | n’’ |  |

|  |  |  |
| --- | --- | --- |
|  | n’ |  |
| n-1 | n | n+1 |
|  | n’’ |  |

Segment 3

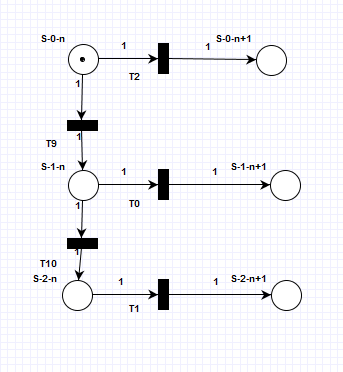
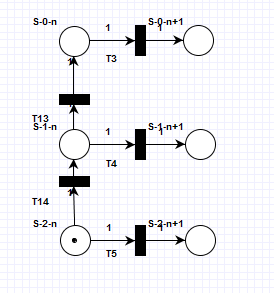
Priorité 1

Priorité 2

Priorité 3

Figure2 : déplacement sur le circuit

* Si une voiture se trouve sur le segment 0, elle cherchera en premier a avancer. Si ce n’est pas possible, elle cherchera à dépasser pas la droite. Si cette solution n’est pas possible, elle attend qu’une case se libère.
* Si elle se trouve sur la ligne 1 elle cherchera d’abord à aller sur la ligne 0. Si ce n’est pas possible, elle cherchera a avancer sur la courante. Si cette possibilité ne lui est pas offerte, elle cherche donc a faire un dépassement par la droite. . Si cette solution n’est pas possible, elle attend qu’une case se libère.
* Enfin, si elle se trouve sur la ligne 2, alors elle cherchera a aller sur la ligne 1. Si cela est impossible elle cherchera a avancer sur la ligne courante. Si cette solution n’est pas possible, elle attend qu’une case se libère.
* L’algorithme est généralisé à un nombre indéfini de lignes, puisque ce dernier est modifiable en modifiant un paramètre dans le fichier de configuration

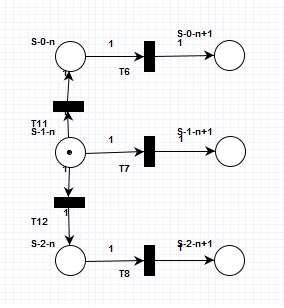
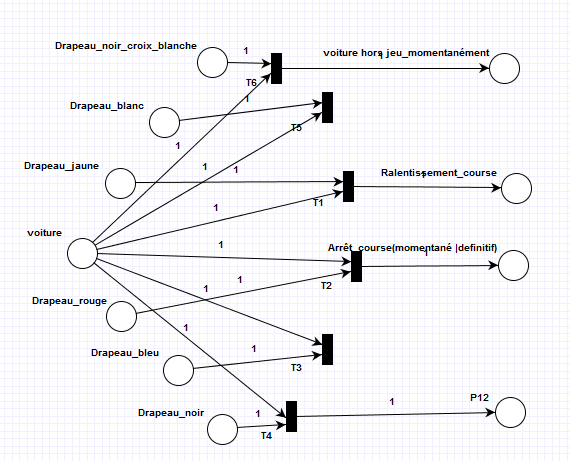


Figure3 : détails des déplacements

* + 1. Gestion des signaux :

Dans une course de NASCAR, les directeurs de course disposent d’un certain nombre de drapeaux qui sont représentés sur le diagramme ci-dessous. Toutefois, seuls les drapeaux les plus importants ont été considérés dans la simulation : le drapeau vert, le drapeau jaune, et le drapeau rouge.



NB : le logiciel utilisé pour réaliser les réseaux de pétri est PIPIv4.2.1

Nous allons maintenant vous présenter la phase de conception ou nous détaillons une partie du code source de l’application ainsi que les outils utilisés

1. PHASE DE CONCEPTION

Cette partie vise à présenter les raisons de nos choix, notre code,

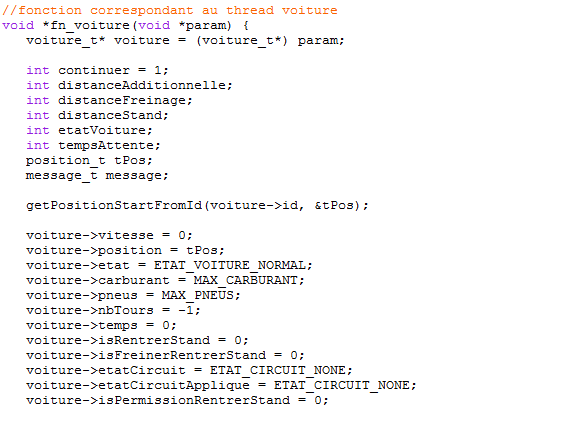
Organisation du programme :

Le programme a été développé sous le système d’exploitation OpenIndiana en langage C et est facilement portable vers d’autres versions de linux en faisant quelques modifications (au niveau des signaux). Il est constitué des fichiers suivants :

* config.h : contient toutes les constantes et nécessaires pour la course (vitesse moyenne, accélération, nombre de cases, etc...). Elles sont pour la plupart modifiables et n’entraveront pas le bon fonctionnement du programme.
* types.h : contient l’ensemble des définitions de structures nécessaire au bon fonctionnement du logiciel.
* utils.h/utils.c : contient l’ensemble des fonctions utilisées par l’un ou l’autre thread et qui ne lui est pas propre (fonction pour générer des nombres aléatoires par exemple).
* voiture.h/voiture.c : contient l’ensemble des variables statiques et des fonctions en rapport avec les voitures
* circuit.h/circuit.c : contient l’ensemble des variables statiques et des fonctions en rapport au circuit
* stand.h/stand.c : contient l’ensemble des variables statiques et des fonctions en rapport avec les stands
* directeur.h / directeur.c : contient l’ensemble des variables statiques et des fonctions en rapport avec les directeurs de course
* accident.h/accident.c : contient l’ensemble des fonctions qui gèrent les accidents
* affichage.h / affichage.c : contient l’ensemble des fonctions qui gèrent l’affichage et l’interaction inter-utilisateurs
* message.h/ message.c : contient l’ensemble des fonctions qui gèrent la file de message (initialisation, suppression, envoi d’un message et réception d’un message)
* nascar.h/nascar.c : contient la fonction principale du programme (le main)
* Makefile : permet d’avoir une compilation automatisée
* Ipc-nettoyage : permet de supprimer les IPC restant après la fin du programme. Normalement, le nettoyage est géré par le programme, même si l’utilisateur l’arrête à l’aide du CTRL+C
* course : c’est le binaire à exécuter après la compilation
  1. Gestion des voitures :

Pour représenter les voitures, nous avons choisi d’utiliser les threads. En effet l’avantage des threads par rapport au processus est qu’on a plus besoin de s’assurer que chaque processus fils ne copie pas entièrement le programme de son père. C’est-à-dire qu’il ne répète pas les forks suivants réalisés par son père. Avec les threads chaque voiture a une fonction qu’il réalise et qui est personnalisé en fonction du paramètre « int id » (identifiant), unique pour chaque voiture et qui est passé en paramètre dans la fonction pthread\_create().

La voiture fait une boucle while dans sa fonction fn\_voiture(void param) jusqu’à ce qu’elle termine sa course, ou qu’elle ait un accident fatale. Mais avant cela, elle est initialisée avec les paramètres suivants :



Continuer : permet de dire si la voiture doit continuer la course après un passage par la boucle while.

DistanceStand : distance entre la voiture et le stand

IsrentrerStand : indique si la voiture est rentrée au stand ou pas.

etatCircuitApplique : indique l’état du circuit après un accident….. Nous avons donc plusieurs variables dont la plupart grâce à leur nom indiquent leurs rôles.

Au cours de cette boucle while, elle effectue plusieurs opérations par des fonctions. Le contenu de notre fichier voiture.h renseigne sur la majorité des actions effectuées par une voiture lors d’un passage dans une boucle while :

#ifndef VOITURE\_H  
#define VOITURE\_H  
  
#define ETAT\_VOITURE\_NORMAL 0  
#define ETAT\_VOITURE\_MOYEN 1  
#define ETAT\_VOITURE\_MAUVAIS 2  
#define ETAT\_VOITURE\_CRITIQUE 3  
  
static int nbEquipes;  
static int nbVoituresEquipe;  
  
static voiture\_t \*voitures;  
static int \*departs;  
  
static pthread\_mutex\_t verrouInitialisationVoitures = PTHREAD\_MUTEX\_INITIALIZER;  
  
void signalVoiture(int);  
  
void initVoitures(int,int);  
  
void getVoitureAt(int,voiture\_t\*);  
  
int getNbEquipes();  
  
int getNbVoituresEquipe();  
  
int getIdVoiture(int,int);  
  
int getEquipeFromId(int);  
  
void getPositionStartFromId(int, position\_t\*);  
  
void afficherVoiture(voiture\_t\*);  
  
void afficherVoitures();   
  
int calculerVitesse(int, int, int, int);  
  
double calculerUsementPneus(int, int, int);  
  
double calculerConsommationCarburant(int, int, int, int);  
  
int calculerTemps(int, int, int);  
  
int calculerDistanceFreinage(int);  
  
void deplacerVoiture(position\_t\*, int, position\_t\*);  
  
int getNbVoituresAjacentes(position\_t\*);  
  
void removeVoitures();  
  
int calculerTempsReparation(int);  
  
#endif

SignalVoiture() pour la gestion des signaux reçus par la voiture du directeur de course.

Initvoiture() permet d’initialiser les voitures au début, lors de la création des voitures et des équipe.

getPositionFromStartFromId() pour déterminer la position que doit prendre la voiture au départ de la course (les voitures sont alignés les uns à la suite des autres en fonction de l’ordre de création).

AfficherVoiture() : permet à tout moment d’afficher des informations sur les voiture grace a un signal envoyer recu.

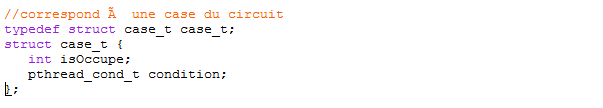
CalculerVitesse() : juste pour calculer la nouvelle vitesse de la voiture. Si elle est déjà max il faudra la faire varier légèrement entre valeurs définies entre lesquelles doit varier la valeur de la vitesse max d’une voiture :

deplacerVoiture() : tant que la voiture n’a pas encore fini sa course, elle doit se déplacer. Cette fonction a été écrite conformément à principe décrit dans les réseaux de pétrie sur les déplacements d’une voiture sur le circuit vu plus haut. Lors du déplacement dans une case la voiture s’assure que celle-ci est libre grâce à un flag qui indique l’état de la case. Si oui elle verrouille la case grâce à une mutex pour ainsi éviter que deux voitures se retrouvent dans une même case à la fois. En quittant la case, il déverrouille l’accès à la case ce qui permet de réveiller une voiture qui serait en attente qu’il bouge.

getNbVoitureAdjacente() : elle permet de savoir le nombre de voiture adjacentes et est utile pour la réalisation aléatoire d’un accident par exemple entre ces voitures.

* 1. Gestion du circuit :

Le circuit est un tableau en trois dimensions de structure. La première dimension permet de diviser les tableaux en segments. Les deux autres dimensions permettent de diviser chaque segment en lignes et en en cases. Les éléments compris dans chaque « case » du tableau sont du type suivant



Is Occupe pour savoir si une voiture se trouve dans la case et pthread\_cond\_t pour le verrouillage ou le déverrouillage.

L’accès au circuit nécessite l’accès par une mutex, celle là même qui est associée aux conditions présentées dans le tableau.

Le déplacement, comme dit plus haut, est fait de case en case. Lors du déplacement dans une case la voiture s’assure que celle-ci est libre grâce à un flag qui indique l’état de la case. Sinon elle se met en attente sur la condition stockée dans la case. En quittant la case, le véhicule signale qu’il libère la case grâce à cette même case et qui permet de réveiller une voiture qui serait éventuellement e.

Dans circuit.c se trouvent les fonctions nécessaires permettant aux voitures de changer de case.

* 1. Gestion des directeurs de course

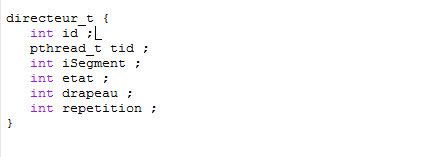
Rôle :

Les directeurs ont la responsabilité de surveiller l’état du circuit (donné par l’état des voitures présentes sur le circuit) et de donner des consignes aux voitures à l’aide de drapeaux.

Il y’a un directeur par segment. Chaque directeur s’occupe de surveiller l’état des voitures présentes sur son segment. Lorsqu’il détecte une voiture présente sur son segment avec un état qui nécessite de ralentir la course et donc de lever le drapeau jaune (état de la voiture mauvais) ou un état qui nécessite le d’arrêter la course et donc de lever le drapeau rouge (état de la voiture critique), il avertit les autres directeurs de course, puis le signale à l’aide des drapeaux à toutes les voitures présentes sur son segment. De même, tous les autres directeurs de course avertiront les voitures présentes sur le segment dont ils ont la responsabilité.

Implémentation :

Les attributs du directeur de course sont les suivants :



id : l’id du directeur

tid : l’id du thread du directeur

iSegment : le segment dont le directeur a la responsabilité

etat : etat du segment

drapeau : dernier drapeau levé

repetitions : nombre de fois que le dernier drapeau a été levé

Pour avertir les voitures, le directeur utilise des signaux.

Pour avertir les autres directeurs, il met à jour une variable accessible uniquement aux directeurs de course.

L’attribut répétitions à toute son importance. Du fait qu’il y’a plusieurs directeurs de course et que ceux-ci ne signalent pas exactement au même moment (chaque directeur à son propre thread), il faut lever plusieurs fois le drapeau (envoyer plusieurs signaux) pour être sûr que toutes les voitures ont bien reçu l’information. Normalement, lever deux fois le drapeau devrait nous assurer que toutes les voitures sont informées.

* 1. Gestions des stands

Rôle :

Il y’a un stand par équipe. Un stand reçoit les demandes où certaines information des voitures de son équipe et peut y répondre favorablement. Les différentes demandes et informations sont :

* la voiture demande à rentrer
* la voiture informe qu’elle est sortie du circuit pour rejoindre le stand
* la voiture informe qu’elle est sortie du stand pour rejoindre la course

Un stand répond aux voitures :

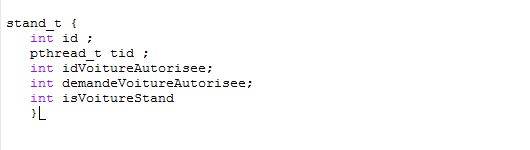
* favorablement : la voiture est autorisée à rentrer
* défavorablement : la voiture n’est pas autorisée. Cette dernière va continuer à effectuer des demandes jusqu’à être acceptée

Les stands utilisent une notion de priorité, qui dépend de l’état de la voiture. Pour expliquer ce concept, prenons un exemple :

* Une voiture dans un état normal veut accéder au stand. Selon toute vraisemblance, celle-ci veut faire le plein et changer ses pneus.
* Le stand, qui n’a pas encore acceptée de demande d’une autre voiture et qui est donc libre accepte
* Une autre voiture subit un accident, est dans un état grave et roule à faible allure. Elle demande un accès au stand. Si la première n’est pas encore sortie du circuit pour atteindre le stand, l’autorisation de venir au stand lui est annulée, et on autorise la voiture accidentée. En revanche si celle-ci est déjà « en route » pour le stand, elle ne sera pas déroutée et c’est la voiture qui attendra.

Implémentation :

Les attributs du stand de course sont les suivants :



id : l’id du stand

pthread\_tid : l’id du thread du stand ;

idVoitureAutorisee : l’id de la voiture autorisée à rentrer au stand :

demandeVoitureAutorisee : la demande de la voiture autorisée ;

isVoitureStand: si une voiture est « déjà sortie » du circuit pour rejoindre le stand.

Concrètement, l’attribut demandeVoitureAutorisee permet de connaître l’état de la voiture pour permettre au stand de gérer les priorités

* 1. Gestion des accidents

Les accidents arrivent de façon aléatoire. Toutefois la probabilité pour une voiture d’avoir accident est fonction de sa vitesse, mais aussi du nombre de voitures adjacentes. De plus, plus l’accident est grave, plus la probabilité d’avoir cet accident est faible.

Plusieurs types d’accidents ont été implémentés :

* ETAT\_MOYEN : la voiture subit un léger accident. Elle réduit sa vitesse et doit retourner au stand, mais cela n’a pas d’impact sur le circuit.
* ETAT\_MAUVAIS : la voiture subit un accident. Elle circule à vitesse très réduite et doit retourner au stand. Les directeurs de course instaurent le drapeau jaune jusqu’à que la voiture ait rejoint le stand.
* ETAT\_CRITIQUE : la voiture subit un grave accident : Elle est immobilisée et ne pourra plus repartir. Les directeurs de course instaurent le drapeau rouge jusqu’à que la voiture ait été évacuée.

Enfin, il est possible pour l’utilisateur de générer un accident sur une voiture aléatoire dans le menu (appuyer sur CTRL+Z)

* 1. Gestion de la course

L’implémentation de la course est très simple. La partie du programme correspondante est présente dans nascar.c. Elle permet d’instancier tous les éléments nécessaires dans le bon ordre et de gérer le contact avec l’utilisateur.

* 1. Gestion des résultats

Une fois qu’une voiture a fini ou a abandonnée, son classement est enregistrée. Cela permettra ainsi à la fin de la course d’afficher le résultat général. Ce rôle est joué par les fichiers resultat.c et resulta.h.

Implémentation :

Cela correspond simplement à un tableau dans lequel est enregistré pour chaque voiture :



id : l’id de la voiture

rank : le classement de la voiture

time: le temps que la voiture a mis pour finir la course

* 1. Gestion des commandes utilisateurs et de l’affichage

Lors d’un CTRL+Z, on accède au menu permettant non seulement de visualiser les informations relatives aux voitures, au circuit, au directeur et aux stands, mais aussi permettant une interaction avec l’utilisateur. Ainsi, puisque la génération d’accidents est aléatoire, celui-ci peut provoquer un accident qui permettra ensuite de visualiser le comportement des différents acteurs et éléments.

1. LES AMELIORATIONS POSSIBLES

Nous avons essayé de faire une course la plus proche possible de la réalité, en prenant en compte un nombre beaucoup de paramètres. Toutefois certains paramètres ont été délibérément ignorés du fait de leur difficulté d’implémentation ou d’un apport très limité à la simulation.

Parmi les paramètres qui ont été ignorés figurent les conditions extérieures à la course. Les conditions relatives à la météo par exemple, elles n’ont pas été intégrées dans la simulation. Ce choix a été fait parce que nous avons pensé que cela n’aurait pas apporté beaucoup au programme.

Tous les drapeaux n’ont pas été implémentés du fait de leur faible pertinence dans le contexte de la simulation.

1. UTILISATION
   1. PREREQUIS

Remarque importante : du fait du temps limité lors de la présentation, le nombre de tour de chauffe a été définis à 0 et le nombre de tours à 1. Pour changer cela, il suffit de modifier les constantes NB\_TOURS et NB\_TOURS\_CHAUFFE présents dans le fichier config.h et de recompiler à l’aide du MakeFile :

$ make

Il a été remarqué qu’avec 42 voitures, le programme consomme beaucoup de ressources. Cela n’affecte pas le bon déroulement de la simulation, mais ralentit l’ensemble. Encore une fois, pour les besoins de la présentation, le nombre de voitures est de 20 (et 2 par équipes). Pour changer cela, il suffit de modifier les constantes NB\_EQUIPES et NB\_VOITURES\_EQUIPE présent dans le fichier config.h en conséquence et de recompiler à l’aide du MakeFile :

$ make

La plupart des constantes présents dans le fichier son modifiables à souhaits. Par exemple, on pourra réduire ou augmenter le nombre de lignes, de cases, de segments, modifier les vitesses et les accélérations et les freinages, etc... Il pourrait être intéressant de faire varier les différents paramètres et d’observer les conséquences sur la simulation.

* 1. COURSE

Pour lancer la course, il suffit juste d’exécuter le binaire. Ensuite le menu permet de visualiser et d’interagir sur la course vous est accessible via la commande CTRL+Z. Pour arrêter la course, il suffit de taper la commande CTRL+C. Le programme s’arrêtera alors proprement.

A la fin de la course, les résultats sont affichés dans l’ordre d’arrivées des voitures.

CONCLUSION

La réalisation de ce rapport nous a permis d’acquérir une certaine expérience dans le domaine de la programmation système sous linux. Cela nous a permis d’être confronté a un cas réelle de simulation. Ce projet nous a également permis d’utiliser les notions aborder en classe, de mieux les comprendre, de les adapter à la résolution d’un problème précis ou d’une tâche et de réaliser simultanément nos révision pour les finaux. Le projet était très intéressant, un peu compliquer car il demandait assez de réflexion et nécessitait un peu de recherche mais nous avons réussi a nous en sortir. Cependant le projet présente des points qu’on pourrait développer plus loin. Par exemple mise en place d’une interface graphique ou encore la gestion des circuits (plusieurs circuits au choix de l’utilisateur de l’application), la possibilité pour l’utilisateur de choisir les paramètres de voiture qu’il veut etc…. en raison du temps qui nous était imparti nous n’avons pas pu ajouter ces accessoires a notre application mais celle-ci fonctionne correctement.