Projet C : Traffic Racer

Jessica FAVIN, Benjamin KRAFFT

Novembre 2016

Tuteur : Michael François

Table des matières

1 Introduction **3**

2 Mode manuel **3**
2.1 L'affichage 3
2.2 Les déplacements 4
2.3 La musique 4
2.3.1 Un simple son 4
2.3.2 Du son à la musique 5
2.3.3 Le script 5
2.3.4 D'une simple musique à une radio 5
2.3.5 De la musique à l'environnement 6

3 Mode automatique **7**
3.1 L'intelligence artificielle 7

4 Conclusion **8**

5 Annexe **9**
5.1 Librairies nécessaires 9
5.2 Bon fonctionnement du jeu 9
5.3 Easter Eggs 9

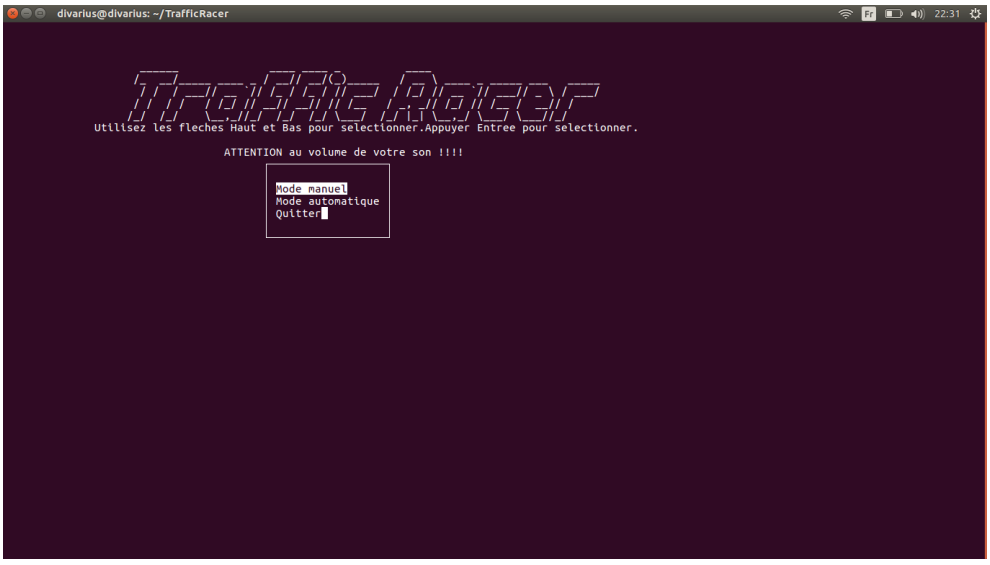
1 Introduction

Le but de ce projet était de coder un jeu de courses de voitures de type Traffic Racer en langage C.
Nous avons implémenté deux modes de jeu. Un mode manuel dans lequel l'utilisateur dirige lui-même la voiture et contrôle la vitesse de celle-ci afin de dépasser le plus de voitures possibles sans collisions. Et un mode automatique pour lequel nous avont implémenté une intelligence artificielle qui analyse la route et se déplace au mieux pour ne pas qu'il y ait de collision avec une autre voiture.
Le jeu est aussi dôté de musique, l'utilisateur peut choisir parmi plusieurs musiques de fond en jeu et à chaque action forte du jeu (départ, accélération, ...) un bruitage la suit pour l'appuyer.

2 Mode manuel

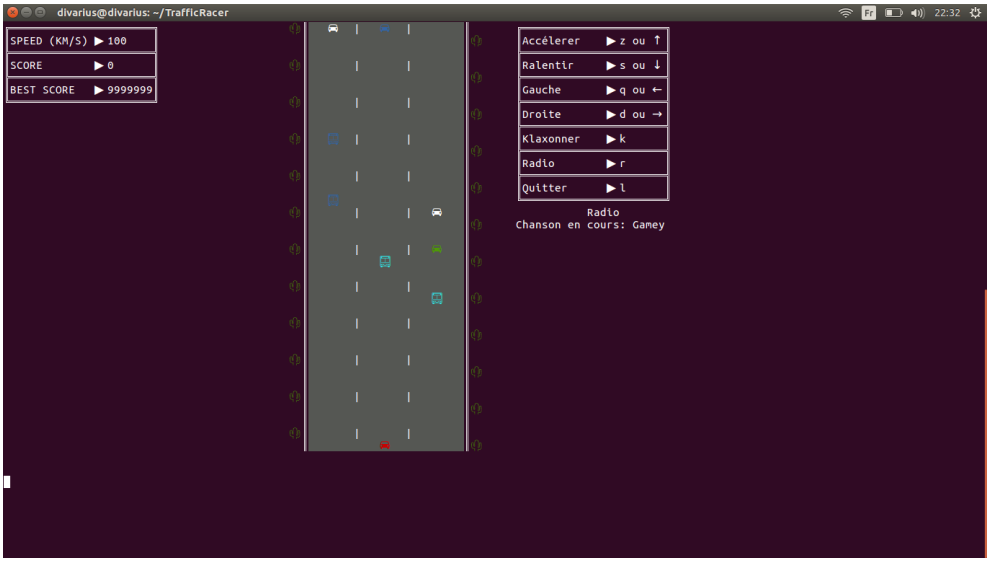
2.1 L'affichage

Nous avons opté pour una affichage en console pour un look rétro. Au lancement du jeu un menu, créé avec ncurses, se lance et affiche les différents choix à disposition de l'utilisateur.



Pour la gestion de l'affichage du jeu lui même nous dessinons au lancement de la partie toute la route, le panel de score/vitesse et le panel de contrôles. Puis nous mettons uniquement à jour les parties de l'écran où l'action se passe. Par exemple pour faire bouger une voiture nous récupérons ses anciennes coordonnées, nous l'"effaçons" en réécrivant des espaces par dessus puis nous déplaçons le curseur à la nouvelle position de la voiture ou nous la

redessinons.



2.2 Les déplacements

Les déplacements de la voiture du joueur se font à l'appuie de touches sur le clavier. Nous récupérons la touche appuyée et en fonction de si sa valeur correspond à celle d'un déplacement à droite ou à gauche nous modifions l'attribut posx de la voiture qui enregistre sa position sur l'axe horizontal puis nous lançons la fonction qui va déplacer la voiture sur l'écran du joueur.

Pour déplacer les voitures "adverses" on utilise une liste de voitures que l'on parcourt. A chaque voiture on compare sa vitesse à celle du joueur pour savoir dans quelle direction la déplacer, si elle a exactement la même vitesse elle reste en place, puis on met à jour l'attribut posy de la voiture qui enregistre sa position sur l'axe vertical et comme pour le joueur on met à jour l'affichage de la voiture.

2.3 La musique

2.3.1 Un simple son

Nous avons voulu rendre le jeu plus plaisant et dynamique, pour cela nous y avons ajouté des sons mais aussi de la musique. Tout d'abord notre objectif était de mettre de simple sons pour améliorer la qualité du jeu mais aussi le dynamiser. Pour cela nous devions avoir une façon d'interagir avec le son en fonction des événements du jeu, mais aussi des actions du joueur. Premier problème comment lancer un son ? Notre solution a étais d'utiliser

la librairie SOX (sous forme de lecteur audio). Pour lire un son, rien de plus simple, il suffisait d'effectuer la commande « play –q » suivis de nom du son voulu dans la fonction system () de notre code C. Viens ensuite notre second problème, si l'on effectue simplement ceci la commande sera effectuer dans notre propre terminal sous peine d'écraser notre affichage. Nous avons donc ajouté le symbole «&» à la fin de notre commande, ce qui nous permet de lancer le son en arrière-plan. A ce niveau nous pouvions donc lancer des sons simplement à n'importe quel moment. Mais nous nous nous sommes pas arrêté à cela.

2.3.2 Du son à la musique

Ayant réussi à ajouter des sons nous avons décidé d'ajouter de la musique de fond. Sur le même principe qu'un son il nous était très simple de lancer de la musique. Or cela nous a conduit à notre troisième problème, comment relancer la musique lorsqu'elle est finit, et ensuite comment l'arrêter si elle est relancée à chaque fois qu'elle se finit ? Nous avons solutionné le replay facilement en ajoutant à notre commande un «repeat 9999», elle est maintenant lancer est répété 9999 fois cependant il nous reste le problème d'arrêter la musique lorsque le joueur perd ou qu'il quitte le jeu. Ceci à étais l'un de nos plus gros problème pour l'ajout du son. Lorsque nous lancions un son il était lu par SOX en arrière-plan donc plus aucun moyen d'y accéder (car en dehors de notre programme). Après la recherche et l'expérimentation de plusieurs solutions, nous avons enfin trouvé la solution qui nous convenait le mieux. Cette solution consiste à écrire un script en bash, qui sera exécuté lors de son appel dans notre projet.

2.3.3 Le script

Ce script nous permet d'arrêter le processus SOX qui lit notre musique. Ce script est composé d'une suite d'instructions système :

```
var=$(ps -ef — grep "play -q $1" — grep -v grep — awk ' print $2 ') ;
kill -9 $var ;
```

La première ligne nous permet de récupérer le PID du processus SOX, et la deuxième nous permet de l'arrêter. Ce problème étant solutionné nous avons continué à ajouter du contenu sonore à notre jeu.

2.3.4 D'une simple musique à une radio

A ce moment nous pouvions lancer et arrêter n'importe quelle musique que nous voulions. Nous nous sommes dit pourquoi ne pas simuler une petite radio avec 4 fréquences ? Cela nous a étais très simple il suffisait d'associer

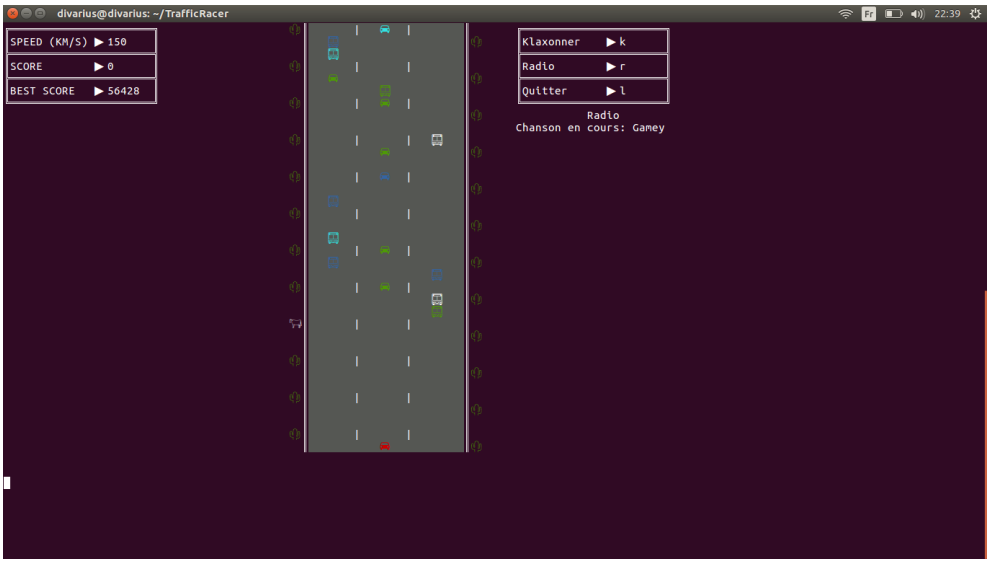
une musique a une fréquence par exemple Sound1 = fréquence 1. Pour simuler le changement de fréquence il nous a suffit d'arrêter l'ancienne musique et de lancer une autre en changeant de fréquence. A partir de ce moment nous avions une ≪radio≫ possédant quatre fréquences différentes avec chacune une musique répétée en boucle. L'appuie sur la touche "r" nous permet de changer de la fréquence de la radio.

2.3.5 De la musique à l'environnement

Nous nous sommes dis pourquoi ne pas faire de chaque fréquence un style différent et y associer un environnement différent. Nous avons donc pris 4 musiques totalement différentes et des thèmes différents pour chacune des fréquences. Nous avons donc adapté nos fonctions afin de pouvoir changer tout le visuel du jeu lorsque la fréquence est changée. Nous avons donc 4 fréquences radio avec chacune une musique et un thème différent. Nous avons aussi ajouté pour chaque thème un animal avec son cri régulièrement au cours du jeu. Nous affichons aussi le titre de la musique associé à la fréquence écouté. Un jeu avec un environnement sonore et visuel modifiable : A ce point nous avions donc tous les éléments cités précédemment plus quelque son généré à certaine action de l'utilisateur. Que l'on vous invite à découvrir en jeu.
Tous nos sons sont libres de droit, et ont été téléchargé sur les sites :
`http://www.universal-soundbank.com/` et `http://www.dl-sounds.com/`

3 Mode automatique

3.1 L'intelligence artificielle



Nous avons décidé d'ajouter un Intelligence artificielle au jeu qui vérifie les trajectoires possibles et les empruntent tant que possible. Le principe de l'IA est simple, elle doit pouvoir détecter la présence ou non de véhicule approchant et les éviter.

Première étape :
L'IA se rapproche au maximum de la voiture qui la précède. Lorsqu'elle en est juste derrière, elle ralentit pour atteindre sa vitesse.

Deuxième étape :
En fonction de la voie sur laquelle elle est, l'IA vérifie les trajectoires possibles. Tout d'abord elle vérifie si sa droite est libre et que la voiture arrivant sur sa droite n'est pas trop proche. Ensuite elle fait de même pour la voie de gauche. Si les deux voies sont libre, elle compte le nombre de voiture arrivant sur la voie de droite et sur celle de gauche et se dirige là où il y en à le moins et elle se déplace en conséquence. Si aucune des deux voie n'est libre elle attend qu'une des deux se libère. Si elle est sur une voie de bords elle fait de même que précédemment seulement pour la voie disponible bien sûr.

Troisième étape :
Lorsqu'elle a changé de voie elle vérifie que la voie est libre sur une assez longue distance pour accélérer jusqu'à sa vitesse max.

Quatrième étape :
Recommencer à chaque fois qu'une voiture arrive juste devant l'IA.

L'implémentation de cette IA à était la partie la plus compliquée car même si cela peut paraître simple nous avons dû avoir recours à plusieurs méthodes pour les calculs et la vérification de la route. Nous avons utilisé deux moyens pour les vérifications et les calculs, le premier étant la représentation matricielle de la route avec pour chaque case le véhicule si trouvant, si aucun véhicule n'est présent nous y avons mis un véhicule "fantôme" (élément neutre de notre matrice). Le second outil à était une liste de voiture que l'on parcourait pour chercher des informations ou pour en récupérer. Nous vous invitons à l'essayer en jeu. Le fonctionnement de la radio et de l'environnement est le même.

4 Conclusion

Nous avons conçu ce jeu pour qu'il soit le plus agréable possible pour le joueur et espérons que vous l'aimerez autant que nous avons aimé le coder.

5 Annexe

5.1 Librairies nécessaires

Ce projet a été codé en C il est donc nécessaire de pouvoir compiler un programme en C ainsi que de pouvoir éxécuter un Makefile.
Nous avons utilisé les librairies SDL2 (libsdl2-dev), ncurses (libncurses5-dev) et SOX (sox et libsox-fmt-mp3), soyez donc bien sûr que celle-ci soient installées pour le bon fonction du programme. Pour afficher les emoticônes soyez aussi sûr d'installer la librarie ttf-ancient-fonts.

5.2 Bon fonctionnement du jeu

En plus d'installer les différentes librairies, pour que le jeu puisse fonctionner correctement il faut impérativement que votre terminal soit en plein écran. Vous pouvez compiler le programme à l'aide du Makefile avec la commande "make" et vous pouvez éxécuter le jeu grâce à la commande "./exec".

5.3 Easter Eggs

Nous avons caché des sons activés à l'appuie de touches du clavier. A vous de les trouver ;)