Algorithmie & Programmation
3e année ESIEA
Année 2017 - 2018

Projet C : Rapport

Professeur: Michael François

Etudiants: Armand Sylvain & Tim Colin

Sommaire

1) Introduction
   a. Objectifs
   b. Préambule & README

2) Outils & Librairies utilisés

3) Fonctionnement
   a. Les différentes structures
   b. Leur mécanique
   c. Randomisation
   d. Le menu
   e. Mode Danger

4) Problèmes rencontrés et solutions envisagées

5) Bonus

6) Ce qui aurait pu être ajouté

7) Sources & Documentation

1) Introduction

Le rapport suivant concerne la réalisation du Projet - C réalisé par Armand Sylvain & Tim Colin, tous deux en 3A (Groupe 31) à l'ESIEA au Campus de Paris.

a. Objectifs

L'objectif du PROJET-C est de réaliser en langage C un simulateur dynamique de circulation urbaine dans le terminal afin de s'approprier les bases du langage C, en deux mois.

b. Préambule & README

Avant de lancer le programme, assurez-vous d'avoir téléchargé préalablement la police d'écriture utilisée dans le Projet - C en entrant la ligne de commande suivante, afin de ne pas avoir d'erreur d'affichage.

sudo apt-get install ttf-ancient-fonts

Le Projet a aussi requis l'usage de librairies, en voici la liste (visibles dans le librairies.h) :

- stdlib.h
- stdio.h
- signal.h
- termios.h
- unistd.h
- fcntl.h
- time.h

Le Projet - C a été réalisé principalement sur des machines ayant pour OS Ubuntu (ou une de ses distributions : Lubuntu), il n'est donc nullement assuré qu'il fonctionne sur un autre type d'OS.

De plus, il est conseillé de dézoomer 2 à 3 fois avant d'exécuter le programme, afin d'avoir un affichage propre. Le nombre de dézooms dépend de la résolution utilisée durant le lancement du programme.

2) Outils & librairies

Commençons par citer les différents outils qui nous ont permis de mener à bien le Projet - C.

Tout d'abord, l'éditeur de texte «Sublime text» nous a permis de gagner en rapidité d'écriture et de lecture: l'autocomplétion et le coloriage automatique de ce dernier nous ont grandement facilité la tâche.

Aussi, le site Github.com nous a facilité l'échange de code, afin de garder un suivi de l'avancement de chacun des membres du binôme, facilitant ainsi la coordination du projet.

Un autre outil qui nous a été d'une grande utilité fut gdb : « The GNU Project Debugger ».

Plus technique, il s'agit de d'un debugger (débogueur en français).
En l'installant et en rajoutant quelques lignes de plus dans le makefile, ce dernier nous a permis de déboguer notre programme de nombreuses fois, notamment en pointant du doigt les erreurs commises lorsqu'un « segmentation fault » (erreur de segmentation) survenait.
Le guide succin d'utilisation à l'adresse suivante :

http://perso.ens-lyon.fr/daniel.hirschkoff/C_Caml/docs/doc_gdb.pdf

nous a été d'une grande utilité et nous en remercions son auteur.

De plus, une police d'écriture spéciale a été utilisée afin d'améliorer le rendu. Nous remercions Piliapp d'avoir mis à disposition sa police d'écriture à l'adresse suivante :

https://fr.piliapp.com/twitter-symbols/

Aussi, nous avons fait l'usage de matrices (char **) et des listes chaînées afin de stocker les différentes données nécessaires au bon fonctionnement du programme.

3) Fonctionnement

a. Les différentes structures

Des structures ont étés réalisées afin de stocker en mémoire les différentes entités: Trafficlights, Bateaux, Véhicules, Tramways, Piétons, Lapins.

Chacune des structures possède une liste chaînée et d'un ensemble de fonctions qui lui sont associées (Spawners pour «instancier» une structure, Eater pour en désallouer l'espace mémoire) afin de faciliter sa gestion en mémoire.

Voici un exemple typique de structure et de sa liste chaînée qui lui est associée :

```
32      typedef struct Vehicule
33  {
34      Direction Direction;
35      int posX;
36      int posY;
37      int Compteur;
38      char custom;
39      char CaseDecision;
40      Carburant Carburant;
41      int virage;
42
43  } Vehicule;
44
45
46  typedef struct VehiculeList
47  {
48      Vehicule* Vehicule;
49      struct VehiculeList *next;
50
51  } VehiculeList;
52
```

Structure & Liste chaînée des Véhicules
(Sublime Text 3)

dec.txt :

Il s'agit d'un fichier texte calqué sur le map.txt. Il contient les caractères codant les trajectoires des différentes entités circulant sur la map. Il contient donc les routes des Vehicules, les lignes des Tramways, les courants des Boats, les passages des Pietons, et le champ des Lapins. Dec étant le diminutif de « décision », ce fichier texte nous a permis d'organiser le trafic fluvial et urbain.

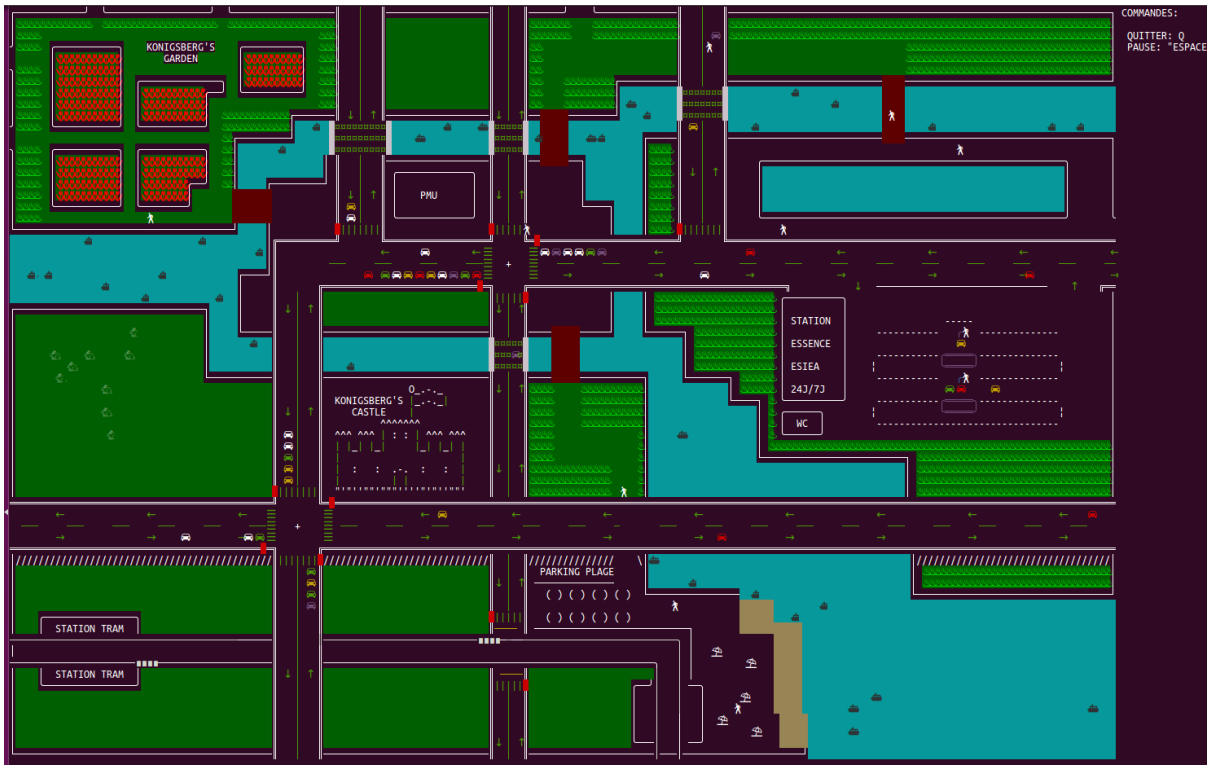
map.txt :

Il s'agit d'un fichier texte contenant en dur la map. Il n'est utilisé qu'une seule fois lors du chargement du programme, afin de l'afficher et de le stocker dans une matrice (char**) nommée MatriceMap. Le plan a été très largement inspiré du plan de la ville de Königsberg (aujourd'hui Kaliningrad), ville connue pour ses ponts qui donnèrent naissance à la théorie des graphes.

Un plan de cette ville est disponible à l'adresse suivante :

https://fr.wikipedia.org/wiki/K%C3%B6nigsberg#/media/File:K%C3%B6nigsberg_Stadtplan_1905.svg

La map en mode « Safe » :



Aperçu de la carte dans un Terminal (bash)

char** MatriceDecision:

Nous avons fait l'usage d'une matrice (char ** MatriceDecision) afin de faciliter la mécanique des entités suivantes : véhicules, piétons et bateaux.

Les coefficients de la matrice codent les différentes actions des entités. En effet, la position d'une entité est représentée de manière cartésienne par un couple (x,y) où x est l'abscisse et y l'ordonnée. Ainsi, en choisissant au préalable une bijection par entité (à chaque caractère correspond une action pour une entité donnée), la MatriceDecision permet aux entités de «décider» de leur trajectoire. Cela se fait de manière pseudo-aléatoire pour certains caractères, notamment lorsque l'entité a plusieurs possibilités de trajectoires et qu'un choix s'impose alors.
Par exemple, si le coefficient (x,y) de la MatriceDecision est un 'd', et qu'un véhicule se trouve à cette position, il ira à droite à l'itération suivant. Si le coefficient (x,y) vaut 'x', la voiture ira soit en haut, soit à droite, selon la parité d'un nombre tiré pseudo-aléatoirement.

char** MatriceMap:

Afin d'éviter d'appeler notre fonction AffichageMap() à chaque itération (cela avait pour résultat d'afficher notre map entièrement et provoquait donc un clignotement au niveau des entités non statiques sur la map), nous avons décidé de stocker les charactères contenus dans le fichier «map.txt» dans une matrice: char** MatriceMap. Ainsi, dès qu'une entité actualise sa position, elle récupère dans la MatriceMap le caractère qu'elle doit printf à son ancienne positition, et printf son caractère sur sa position suivante. La MatriceMap nous a donc permis d'obtenir un affichage plus soigné.

b. Leur mécanique

Mécanique des Tramways:

La mécanique du Tramway est assez limitée. En effet, les deux Tramways circulent sur des lignes, leur mouvement est donc complètement déterministe, et déterminé par les lignes de Tramway. Au début, nous avions opté pour une structure pour le moins naïve, mais facilitant grandement la mécanique du Tramway: une structure par wagon. Le tramway suivait donc les rails à la manière d'un véhicule mais un problème d'affichage persistait lors de l'arrêt du Tramway à la station. Nous avons donc choisi de ne faire qu'une structure par Tramway, et de gérer son affichage en fonction de ses coordonnées. La grande difficulté fut de gérer l'affichage lors des virages des Tramways, cette difficulté fut dissoute par une fonction supplémentaire gérant l'affichage dans le cas d'un virage.

Mécanique des Véhicules:

La mécanique des véhicules est aussi déterminée par la MatriceDecision, mais certains caractères de celles-ci peuvent donner lieu à plusieurs issues (cf. plus haut avec le caractère ‘x’). De plus, les véhicules peuvent s’arrêter s’ils se trouvent à un embranchement où le feu est passé au rouge ou encore lorsque qu’un véhicule se trouve devant.
Enfin, les véhicules s’ils en ont le besoin, peuvent s’arrêter à la station essence afin de faire le plein de leur véhicule. On remarque que les vilains automobilistes laissent derrièrent eux des déchets à l’entrée de la pompe à essence.

Mécanique des Piétons:

La mécanique des piétons est similaire à celle des voitures, sauf qu’ils marchent sur les trottoirs. Ils peuvent cependant traverser la route si la couleur du TrafficLight le leur permet.

Mécanique des Bateaux :

Les bateaux suivent un courant défini par des caractères contenus dans la MatriceDecision. La plupart sont des caractères donnant lieu à plusieurs trajectoires possibles. Le mouvement des bateaux est donc dans un sens aléatoire.
Ils s’arrêtent si et seulement si un bateau se trouve devant eux, ainsi, la gestion du trafic fluvial s’auto-régule et ce, même pour un nombre de bateau ridiculement grand.
De plus, les bateaux passent sous des ponts. En utilisant habilement la structure du bateau et la MatriceDecision, il nous a été possible de faire passer les bateaux sous la route, ou sous un pont.

Mécanique des hélicoptères:

Les hélicoptères sont des entités qui apparaissent rarement au bord de la carte, à chaque frame, il y une chance sur 300 qu’un hélicoptère débute sa course sur l’un des bords de la carte. En moyenne, ils traversent la carte de manière diagonale. En effet, selon le bord de la carte sur lequel ils apparaissent, ceux-ci ont le droit à deux degrés de liberté seulement. Par exemple, si un hélicoptère apparaît sur le côté droit de la carte, il aura pour sûr le déplacement vers la gauche qui lui sera autorisé, et un déplacement soit vers le bas, soit vers le haut.

L’hélicoptère effectuera donc une marche aléatoire qui en moyenne est diagonale.
Une fois qu’ils atteignent un bord de la map, on désalloue leur mémoire à l’aide d’un Eater et ils disparaissent.

Mécanique des Lapins :

La mécanique des Lapins a été pensée afin de se rapprocher au maximum de la mécanique d'un automate cellulaire, à la manière du «Game of Life» de John Conway. Ces derniers évoluent dans un champ délimité à gauche de la carte, leur mouvement est actualisé plus rarement que celui des autres entités afin de donner une impression de lenteur.

Nous avions offert aux Lapins la mécanique suivante, très similaire à celle d'un automate cellulaire. Mais un automate cellulaire, (contrairement à nos Lapins qui interagissent dans un champ fini) n'a à priori pas de limite de taille, et donc pas de problèmes de mémoire.

Le nombre de nos Lapins grandissait parfois de manière exponentielle et causait des segmentation fault, qui étaient inévitables.

Nous avons donc finalement opté pour laisser les Lapins se mouvoir de manière aléatoire dans le champ, sans qu'il n'y ait possibilité d'accouplements, à notre plus grand regret.

Le code associant une structure d'automate cellulaire aux Lapins reste néanmoins disponible en commentaire dans le fichier lapins.c.

c. Randomisation

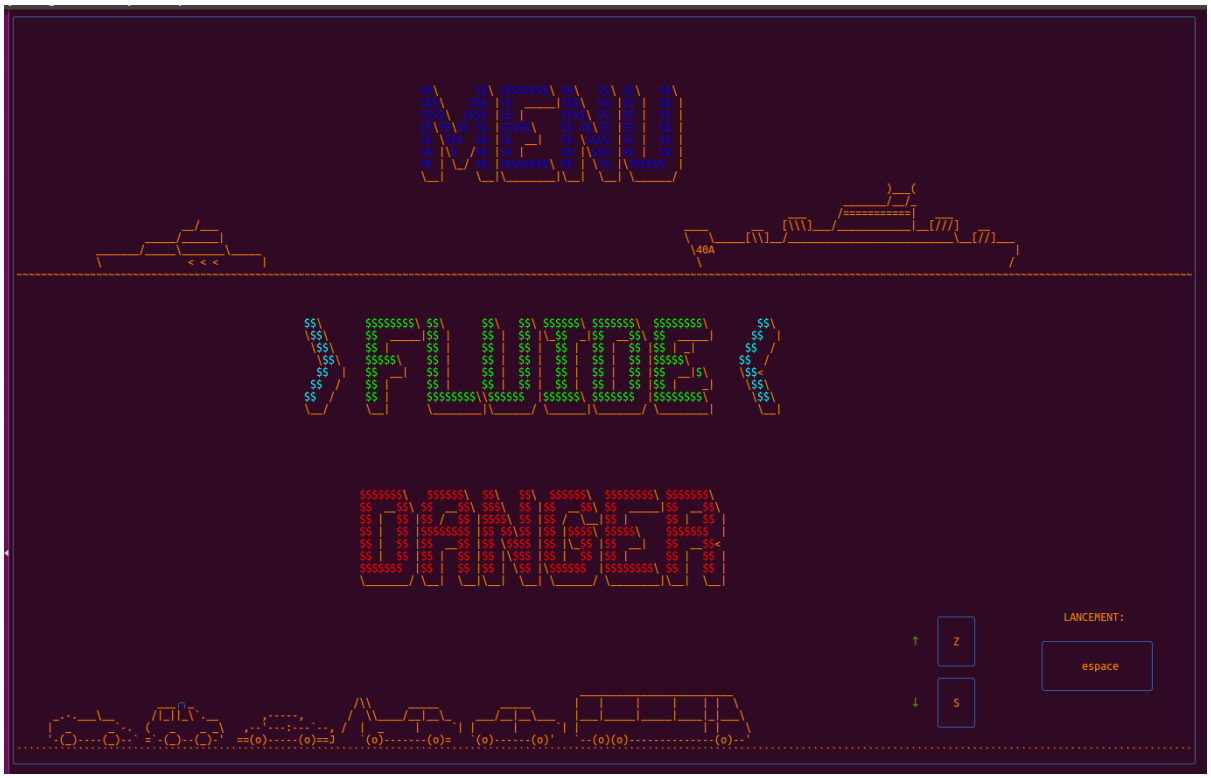
L'un des objectifs du Projet-C était de le rendre aléatoire, de faire en sorte qu'une exécution du programme ne donne jamais deux fois le même résultat. Nous avons donc rendu les mouvements des différentes entités pseudo aléatoire à l'aide de la fonction rand().

De plus, nous avons aussi rendu aléatoire l'apparition des entités sur la carte (et ce, dès le lancement). Ainsi, il est impossible de prédire à l'avance où apparaitra la prochaine entité sur la carte, à moins d'avoir de la chance.

L'hélicoptère reste néanmoins l'entité la plus aléatoire, tant par son point d'apparition, qui peut-être n'importe où sur les bords de la map, que par sa trajectoire en ligne brisée.

d. Le menu

Le menu apparaît au lancement du programme. Il est constitué de deux fichiers textes et de quelques lignes de code afin de permettre à l'utilisateur de choisir entre le mode Safe et le Mode Danger.



Aperçu du menu dans un Terminal (bash)

e. Mode Danger

Le mode Danger comporte quelques différences par rapport au mode "Safe".
Tout d'abord, les entités apparaissent plus fréquemment sur la carte, créant ainsi un effet d'encombrement.

Aussi, un accident peut avoir lieu entre un Tramway et une voiture. En effet, nous avons réalisé une animation d'un automobiliste sortant sur de sa voiture sur la ligne de Tramway, avant de se faire percuter par le Tramway, laissant derrière lui une gerbe de flammes.

4) Problèmes rencontrés & Solutions envisagées

Comme dit plus haut, l'un des premiers problèmes rencontrés fut les «segmentation fault».

En effet, nous (les deux membres du binôme) venons de prépa, et n'avons codé jusqu'alors qu'en Python ou CamL, deux langages où des mots comme «allocation mémoire» ou encore «pointeurs» ne sont pas au menu. Ainsi, les « segmentation fault » survenant de temps à autre, gdb nous a été d'une aide cruciale pour repérer la cause de ceux-ci.

Un autre problème, d'affichage cette fois, fut en partie résolu par notre très cher et adoré professeur Mr.François qui nous conseilla de rajouter «\n» lors du printf d'un caractère afin de ne pas causer de désagréments sur la ligne sur laquelle le caractère se trouve.
Notre affichage s'en est retrouvé épuré et soigné.
Cependant cela n'a pas empêché les caractères spéciaux de se comporter comme deux caractères, écrasant le caractère suivant et générant des problèmes au niveau du caractère précédent. C'est pourquoi nous avons modifié la condition d'arrêt des voitures entre la verticale et l'horizontale.

Un autre soucis fut lors de l'usage de la fonction fgets. Celle-ci devait récupérer ligne par ligne le contenu d'un fichier texte. A défaut de ne pas savoir s'en servir proprement, nous avons préféré utiliser la fonction «getline», qui joue le même rôle.

Aussi, les Tramways furent source d'ennui. En effet, nous étions partis sur une méthode consistant à associer une structure par wagon de Tramway. Tout marchait sauf l'affichage qui dégénérait lors de l'arrêt du Tramway à la station. Nous avons donc dû repartir de zéro et opter pour une structure par Tramway, la gestion de l'affichage a été refaite en conséquence, dans la douleur et l'acharnement. Finalement, les Tramways fonctionnent.

5) Bonus

De nombreuses entités sont des bonus et n'apparaissaient pas dans le cahier de charges du Projet. Parmi celles-ci, on peut citer :

- Les bateaux
- Les hélicoptères
- Les lapins

Aussi, afin de gérer l'interaction entre la ligne de Tramway et la route (Oui, il y a un croisement entre la route et la ligne de Tramway, pour épicer la circulation), nous avons réalisé une barrière qui s'active afin de barrer le passage des voitures, lors du passage d'un Tramway.

6) Ce qui aurait pu être ajouté

Nous remarquons qu'une redondance apparait dans notre code. Chaque entité possède un ensemble de fonctions qui lui sont propres, mais qui permettent de faire la même chose que d'autres ensembles de fonctions réservés à d'autres entités. En effet, nous avons une fonction une fonction « Spawner » par entité, une fonction Eater par entité, etc …

Nous aurions pu faire des fonctions prenant en argument n'importe quelle entité, afin de factoriser le code, et ainsi avoir moins de fonctions.

Cela aurait pu se factoriser à l'aide du « polymorphisme » (vocabulaire propre au CamL, mais l'idée est la même ici) offert par le type void *.

En effet, si une fonction possède un argument « void * » dans son prototype, elle accepte tout type de pointeur, sans sourciller. Nous aurions donc pu n'avoir qu'une seule fonction Spawner, qu'une seule fonction Eater, etc … Une grosse factorisation du code en aurait découlé.

Aussi, nous aurions aimé ajouter du son en 8 bits pour améliorer l'ambiance du programme, mais il nous a été impossible de faire fonctionner sox.

7) Sources & Documentation

Ici se trouvent les différents liens sur Internet nous ayant permis de d'avancer dans le projet de manière autonome.

https://www.cs.bu.edu/teaching/c/linked-list/delete/	Pour les Listes chaînées
https://en.wikipedia.org/wiki/ANSI_escape_code	Pour les couleurs
https://openclassrooms.com/	Pour beaucoup de choses
https://fr.wikipedia.org	Pour beaucoup de choses