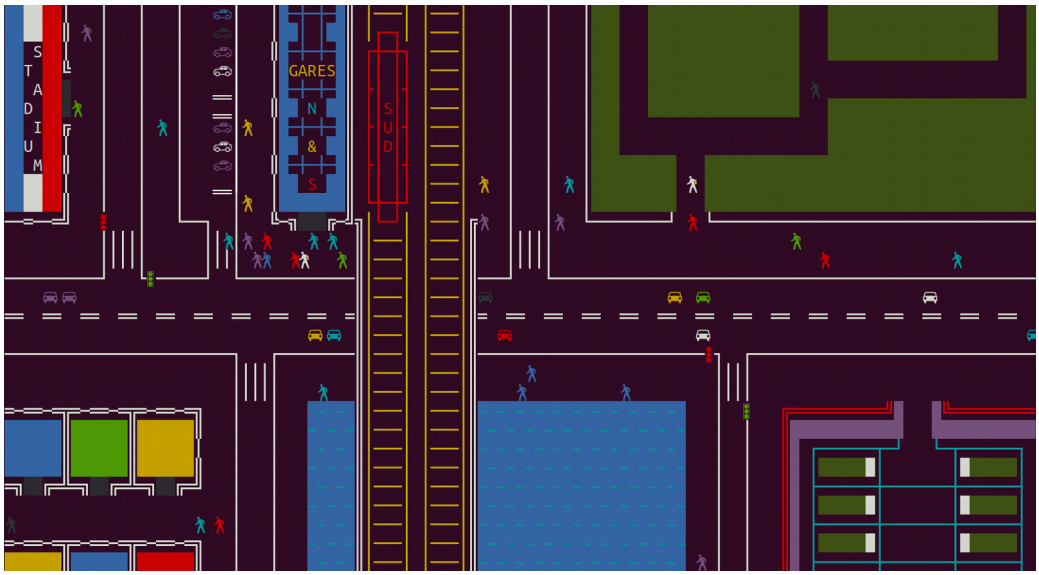
Projet en Programmation C

SIMULATEUR DE CIRCULATION URBAINE



Réalisé par : **Ignace ANDRIAMARO**

Thybault ALABARBE

Professeur : **M. FRANCOIS**

Année Scolaire : 2017 – 2018 ESIEA

Sommaire

I *INTRODUCTION :*

Dans ce projet, nous réalisons une simulation de la circulation urbaine. Dans cette circulation, il y a des voitures, des trains, des piétons et aussi des feux de circulations. Différents moyens sont utilisés afin de réaliser les intéractions entre ces éléments. Nous avons rendu disponibles deux modes : le mode fluide et le mode danger.

Valgrind nous a servi à déboguer tous les aspects liés à la mémoire.

Github a été utilisé afin de faciliter le développement à plusieurs sur différentes machines.

Le code source du projet est disponible à l'adresse suivante :

https://github.com/Soulthym/c-citysim

I- Réalisation des fichiers support

1) *Le plan de la simulation*

Afin de réaliser la simulation d'une circulation urbaine, nous avons besoin d'un plan où doit se dérouler cette simulation. Dans ce projet, le plan est inspiré d'un quartier près de la station RER de Cachan. Ce plan sera réalisé dans un fichier texte que le programme ouvrira au début de la simulation. Pour créer la Map, nous utilisons des caractères normaux mais aussi des caractères spéciaux. Ces caractères spéciaux vont nous causer quelques problèmes car ils n'ont pas la même taille en mémoire et lors de l'affichage, il faut donc leur allouer une taille variable, allant de 1 à 4 caractères.

De plus nous verrons plus loin que nous avons besoin de différents fichiers de propriétés, relatifs aux déplacements des piétons, des voitures, du Tram, aux couleurs ainsi qu'aux propriétés de la map.

Nous avons donc créé une structure map, qui stocke tous ces éléments pour chaque case, ainsi que son caractère spécial/ascii associé

.

2) Les véhicules et les piétons

Les véhicules, les trains et les piétons sont représentés par des caractères spéciaux. Pour les déplacer, il faut savoir dans quelles directions ils peuvent se déplacer, ici nous avons un choix entre les quatre directions : Nord, Est, Sud, Ouest. Cependant, les voitures ne peuvent pas aller sur les trottoirs tout comme les trains ne peuvent aller sur la route, il a donc fallu définir les endroits où chaque véhicule ou piéton peut aller. De ce fait, pour chaque « personnage », nous avons besoin d'un fichier texte pour stocker les directions possibles selon chaque position ; on a donc un fichier pour les voitures, un fichier pour le train et un fichier pour les piétons.

Dans ces fichiers texte, chaque direction est encodée en puissance de 2 afin de faciliter la manipulation, nous avons donc : 1 NORD, 2 EST, 4 SUD, 8 OUEST. Ces fichiers doivent bien se superposer avec le plan de la ville, afin de diminuer le temps nécessaire à leur création. Ainsi, le train par exemple ne peut se déplacer que dans la direction nord ou sud ; on a donc à la place des rails les caractère 1111 4444.

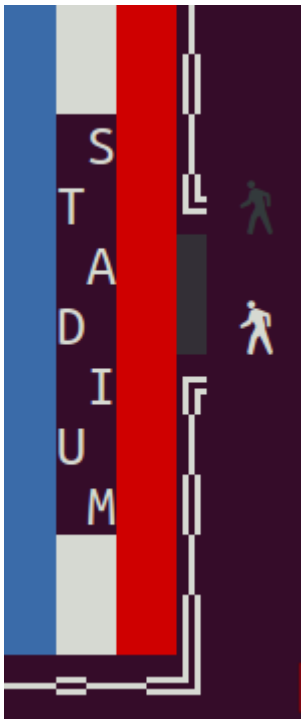
Les véhicules et les piétons ont leur propre structure. Un véhicule a comme caractéristique ses coordonnées dans le plan, sa direction, sa couleur son image et sa capacité à se garer ou pas.

De même, les piétons ont ces mêmes caractéristiques, moins la capacité à se garer.

```
typedef struct{
  int x;
  int y;
  char canPark;
  char direction;
  char *color;
  char *disp;
}car;

typedef struct{
  int x;
  int y;
  char direction;
  char *color;
  char *disp;
}walker;
```

3) Les fichiers couleur



La couleur du plan et de tous les caractères ascii sont par défauts celui du terminal de travail. Sous environnement GNU/Linux, tout sera de couleur blanc. Pour modifier cela, chaque couleur aura une lettre qui lui correspond, qui sera activée avant tout affichage.

L'intégration des couleurs de la ville se fait grâce à un fichier texte. Ce fichier contient les lettres dont chacune représente une couleur. Ces lettres sont disposées. La position des lettres doit donc correspondre au plan de la cité.

Afin de transformer les lettres correspondant à chaque couleur en leur couleur, nous avons besoin de caractères spéciaux. Ces caractères seront lu par le terminal comme étant des codes couleurs. La couleur ROUGE est par exemple associée à la chaîne de caractères « `\x1b[31m` ». Le nombre après le crochet indique la couleur souhaitée.

II- Réalisation du programme

1) Manipulation des fichiers

Afin de réaliser le projet, certain types de fichiers sont nécessaires : les fichiers .c qui contiennent les codes en C du programme, les fichiers .h qui contiennent les librairies utilisées pour pouvoir compiler ainsi que les définitions des structures , et les fichiers textes qui contiennent les informations sur la ville (comme le plan, les couleurs, les propriétés...). Pour pouvoir compiler tout le programme, on utilise make, dont les instructions sont stockées dans le makefile.

Un script bash contient les commandes de l'utilisation basique de git, utilisé pour le versionning .

Au début, avant d'afficher la ville, pour que la simulation se produise, il faut ouvrir tous les fichiers textes. La fonction LoadMap ouvre tous les fichiers dont nous avons besoins et créé la structure map.

Il nous suffira par la suite d'appeler les fonctions d'affichage de la map en fonction des besoins.

Nous avons cependant rencontré un problème lors de la suppression de caractères spéciaux sur le terminal : ils ne s'affichent pas tous sur un seul caractère, et souvent effacent les caractères suivant menant à ce genre de résultat...

Afin de régler ce problème, la fonction CleanMap, affiche les caractères « corrompus ».

2) ***Déroulement du programme***

L’utilisation du mode fluide requiert plusieurs étapes à effectuer dans l’ordre:l’initialisation des différents éléments avant la boucle principale, puis dans la boucle, l’effacement des éléments, la mise à jour de leurs interactions, leur affichage, la fonction CleanMap réglant les bugs graphiques, et une pause pour laisser le temps au bash d’afficher les modifications. Les différentes structures sont ensuite libérées.

Le mode danger en revanche, accélère le déroulement des actions, augmente drastiquement le nombre de piétons et de voitures, et enlève les feux, responsable du bon déroulement de la circulation.

3) ***Le déplacement des véhicules et des piétons***

Le simulateur de circulation urbaine simule le déplacement des véhicules et des piétons dans une ville. Comme dans la réalité, des voitures et piétons apparaissent et disparaissent de notre champs de vision. Pour générer cela, les voitures et piétons seront créés au hasard sur différents point d’apparition.

Les piétons peuvent entrer dans les bâtiments, cet effet est réalisé en faisant disparaître certains piétons devant l’entrée, et d’autres apparaissent aléatoirement devant le bâtiment.

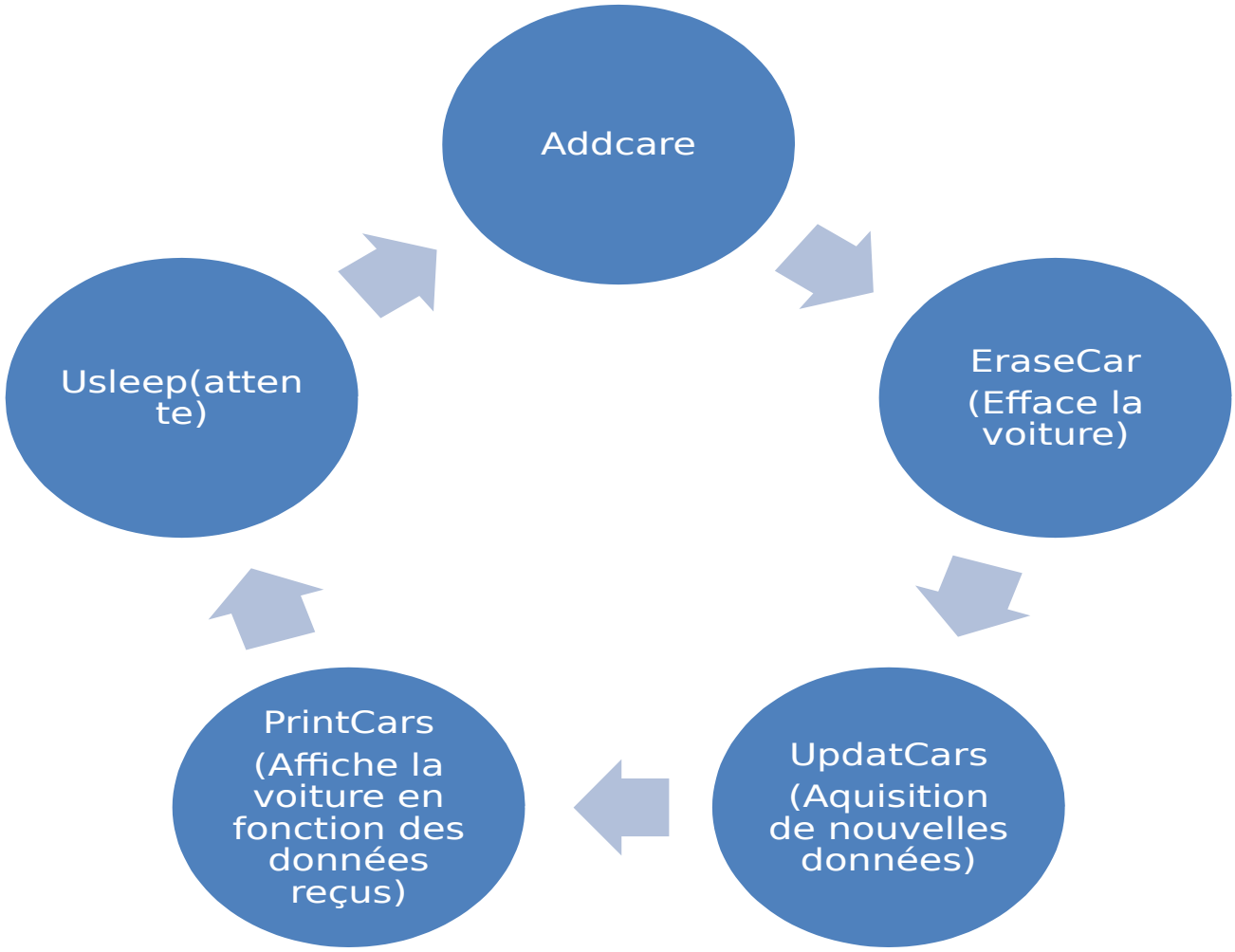
Dans la même idée les voitures peuvent se garer, faisant apparaître un piéton sur le trottoir. Si un piéton passe à proximité d’une voiture garée, il peut rentrer dedans et repartir dans la circulation.

Le tram lors de son arrivée en gare fera apparaître les piétons les piétons du bon côté de l’arrêt (devant la gare pour les passagers du train Sud).

Les feux permettent la régulation de la circulation des voitures et des piétons. Ils ont chacun un timer propre, 2 zones d’affichage(une pour les piétons, une autre pour les voitures), et des états leur dictant quelles caractéristiques de la map sont à changer pour adapter la circulation. Les états varient en fonction du timer.

a. Déplacement des voitures et piétons

Le déplacement des voitures se fait de la manière suivante : une direction est sélectionnée parmi les directions autorisées à une certaine position pour l'élément en question.



b. Déplacement des piétons

Les piétons sont plus difficile à gérer car ils peuvent se déplacer dans des paysages ou à côté des maisons. Nous devons donc gérer cette rencontre entre les piétons et les objets. Cependant la gestion des propriétés case par case permet, lorsque les fichiers sont bien configurés, une gestion automatique.

III- **Répartition des tâches**

Ignace :

- Création de fichiers de configuration

- Rédaction du rapport

- Création et design d'un menu

- Aide au développement du framework (aide sur le chargement et l'affichage de la map)

Thybault :

- Création de fichiers de configuration

- Rédaction du rapport

- Implémentation du menu

- Développement du framework (map, piétons, voitures, trams et leurs intéractions)

- Utilisation du framework pour la mise en place des 2 modes.