TP langage C # 2
FSM (*Finite State Machine*
La fourmi de Langton)

Antoine Tauvel

Octobre 2021

1 Introduction

La notion de machines à état finis (Finite State Machine) est un concept de programmation permettant de décrire le comportement de n'importe quel système dépendant du temps, du moment qu'en effet le nombre d'état pris soit finis.

Le système que nous allons étudier est la fourmi de Langton, du nom de son inventeur. Les règles qui régissent la fourmi sont très simples : la fourmi est caractérisée par une position (en X et en Y, variant entre 0 et DIMX-1 et entre 0 et DIMY-1) et une direction. 0 correspond au Nord, 1 à l'Est, 2 au Sud et 3 à l'Ouest.

Chaque case du plateau de jeu peut être blanche ou noire, et peut changer d'état.

A chaque tour :

— Si la fourmi est sur une case blanche, elle tourne à droite de 90°.

— Si la fourmi est sur une case noire, elle tourne à gauche de 90°.

— La case où est la fourmi change d'état.

— La fourmi avance d'une case dans sa nouvelle direction.

Ces règles simples amène un comportement amusant. Au bout d'un certain nombre d'itération (plusieurs milliers), de l'ordre émerge du chaos et la fourmi commence à dessiner une "autoroute" sur le plateau de jeu, comme on le voit sur l'image ci-dessous extraite de Wikipedia.

FIGURE 1 – (c) Wikipedia - 10 000 itérations de la fourmis (le point rouge) de Langton sur un plateau blanc.

2 Préparation

Pour la préparation de ce TP, merci de regarder la vidéo suivante, pour comprendre ce que l'on souhaite faire : https://www.youtube.com/watch?v=qZRYGxF6D3w.

Pour la préparation de ce TP, merci de relire le cours sur les listes chaînées.

Pour la préparation de ce TP, dans les parties suivantes, il vous faut répondre par avance aux questions faisant apparaître un petit cerveau comme ceci :

— Ceci est une question de préparation.

3 Préliminaire

- Créez un répertoire de travail pour notre projet.
- Codez une fonction `ETAT * createFirstState(int direction, int x, int y);` dans un fichier nommé "langton.c". Pour le moment cette fonction ne fait rien.
- Créez un fichier "langton.h" contenant le prototype de createFirstState.
- Tester votre programme dans un main de prototype `int main (void)` ;.
- Compilez, modifiez les éventuelles erreurs de compilation et algorithmique.

4 Première partie : initialisation de la fourmi

On souhaite pourvoir rejouer les différentes étapes. On va donc stocker l'état du tableau et de la fourmi dans une liste chaînée.

- Pourquoi stocker l'information dans une liste chaînée plutôt que dans un tableau ?
- En quoi une liste doublement chaînée (previous et next) serait plus intéressante ? Compte tenu du temps imparti, on fera une liste simplement chainée.
- Complétez le fichier d'entêtes ci-dessous.
- Comment déclarer une structure de type ETAT avec une allocation statique ?
- Comment déclarer une structure de type ETAT avec une allocation dynamique ?

Voici un extrait du fichier de déclaration "langton.h" :

```c
#include <unistd.h>     // utile plus tard : fonction usleep();
#include <stdlib.h>
#include <stdio.h>

#define DIMX 5
#define DIMY 5

typedef struct etat{
    int tableau[DIMX][DIMY];
    int fourmiDirection;        // 0 : Nord, 1 : Est, 2: Sud, 3 : Ouest
                                // A vous de compléter
    }ETAT;

typedef ETAT * ptETAT;

ptETAT createFirstState(int x, int y, int direction);
void displayState(ETAT * e);
void createNextState(ETAT * tete);
```

Pour mémoire, l'allocation dynamique de mémoire s'écrit comme ci-dessous. Vous trouverez également quelques exemple d'accès aux variables. Pour mémoire, `(* structPtr).toto` est équivalent à `structPtr->toto`.

```c
ETAT * newState = calloc(1,sizeof(ETAT));

newState->fourmiDirection = 2;        // Ceci est un exemple, pas la réalité.
newState->fourmiX = 3;                // Etc.
```

— Complétez le fichier langton.h avec votre préparation.
— Ecrivez la fonction `ETAT * createNewState(int direction, int x, int y);` qui initialise un nouvel état avec un tableau d'état à 0, et les variables d'état de la fourmi passées en paramètres.
— Testez à l'aide de ddd (n'oubliez pas qu'il faut compiler avec l'option -g). Vous devriez obtenir quelque chose ressemblant à la capture d'écran suivante.

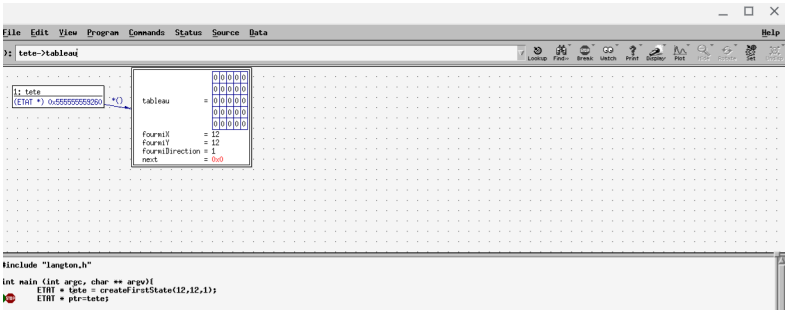

FIGURE 2 – Vue de ddd après l'initialisation d'une structure.

5 Ce qui est difficile : void createNextState(ETAT * tete) ;

Attention, cette partie est longue et la plus technique. Validez bien chaque sous partie avant de passer à la suivante sous peine de code impossible à corriger.

L'état suivant dépend de l'état précédent, comme dans toute machine à état. La première étape consiste donc à créer un nouvel état, et à chainer cet état dans la chaîne existante.

Un pointeur, par exemple `ETAT * old` pointe sur le dernier état tandis que `ETAT * new` pointe sur un nouvel état.

Dans la fonction `void createNextState(ETAT * tete)` :

- Définissez les pointeurs old (pour le moment une recopie de tete), et new (par appel à la fonction `createNewState(0,0,0)`).
- Faite parcourir la liste chainée à old jusqu'à obtenir le dernier élément de la chaîne (il s'agit d'une boucle tant que le pointeur * next n'est pas à la valeur NULL).
- Chaîner le nouvel élément.
- Visualisez à l'aide de ddd.

Notre prochaine étape consiste à traiter le tableau de jeu. Pour mémoire, `new->tableau` a presque le même état que `old->tableau` sauf sur la case de coordonnée `old->fourmiX`, `old->fourmiY`, qui change d'état. Pour changer d'état, l'opération mathématique est (où x représente l'état de la case) :

$$x = 1 - x$$

.

Pour recopier les autres cases, du tableau d'origine, on balaye les cases grace à une double boucle for.

Dans la fonction `void createNextState(ETAT * tete)` :

- Ecrivez une double boucle for qui recopie l'état de `old->tableau` vers `new->tableau`.
- Changez l'état de la case où se trouve la fourmi.
- Visualisez à l'aide de ddd.

Il faut maintenant changer la direction de la fourmi. Selon la valeur de la case `old->tableau[old->fourmiX][old->fourmiY]`, je vais soit additionner 1 à `old->fourmiDirection` soit retrancher 1. Bien entendu, parvenu à 4 je reviens à 0, et parvenu à -1 je reviens à 4.

Dernière étape : la fourmi avance d'une case dans son ancienne direction. C'est l'occaion ou jamais d'utiliser un switch / case en fonction de `old->fourmiDirection` pour générer `new->fourmiX` et `new->fourmiY`.

Pour le moment, on ne se préocupe pas de ce qui arrive si la fourmi "heurte les murs"...

Dans la fonction `void createNextState(ETAT * tete)` :

- A la suite du code précédent, générez les bonnes valeurs pour `fourmiDirection`, `fourmiX`, `fourmiY`.
- Visualisez à l'aide de ddd. Vous devriez obtenir quelque chose ressemblant à la capture d'écran ci-dessous.

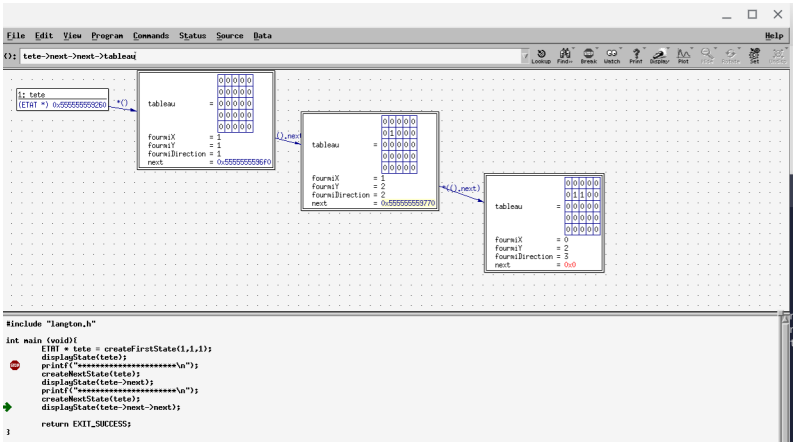

FIGURE 3 – Vue de ddd après trois appels à createNextState.

6 Un peu de décor autour...

On veut maintenant obtenir de "belles" animations en mode ASCII. Tout d'abord, il faut être capable d'afficher le terrain de jeu et la fourmi. La fourmi sera représenté par 4 caractères selon sa direction : ^, >, < ou |.

Dans la fonction `void displayState(ETAT * e)` :

- A l'aide d'une double boucle for :
- Si la case est celle de la fourmi, afficher l'icone de la fourmi en fonction de sa direction (switch ou tableau de constante selon vos goûts).
- Sinon affichez selon la case du tableau soit un espace, soit un X.
- Testez en appelant 3 fois la fonction d'affichage et de création d'un nouvel état. Vous devriez obtenir quelque chose ressemblant à la capture ci dessous (il y a une erreur d'affichage dans cette capture sur l'orientation de la fourmi).

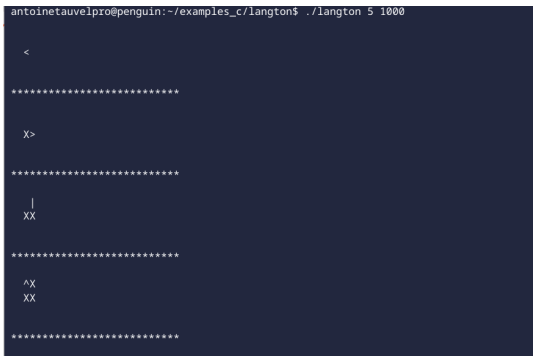

FIGURE 4 – En mode texte.

7 Cherries on cakes

Il nous restent maintenant à ajouter une animation en mode texte. Pour cela, vous avez besoin d'effacer l'écran, et d'attendre.

Effacer l'écran va se faire avec une ligne merveilleuse : `printf("\e[1;1H\e[2J");`. Il s'agit d'une expression réglière qu'il n'est pas besoin de comprendre ici.

Attendre est plus simple, il s'agit d'un appel à la fonction usleep, auxquel vous donnez le temps en microsecondes. Attention, c'est un appel système linux (ne fonctionnant pas sur tous les OS).

— Modifiez le programme de manière à lui passer deux paramètres, le temps d'attente entre deux "frames" et le nombre d'itération.
— Compilez et testez.

Vous avez fini avant la fin ? Pas de panique ! Vous pouvez maintenant ajouter les options suivantes :

— Rebonds de la fourmi sur les bords du tableau
— Ajout d'un paramètre stipulant un affichage uniquement à partir d'une certaine trame.
— Management de la mémoire : si la liste chaînée devient "trop grande", on libère les N premières cases de la mémoire.

8 Conclusion

Beaucoup de concept majeurs ont été abordés dans ce TP. Tout d'abord, sur les FSM : on a séparé l'évolution de l'état de la machine de ses sorties. L'état interne (direction, X, Y) est à priori invisible. Ces concepts seront approfondis au prochain semestre.

Ensuite, une revue de l'allocation dynamique de mémoire. Pensez à désallouer !

Enfin, les listes chaînées. C'est un outil très puissant de modélisation. Elles servent ici à historiser les états, mais auraient aussi pu modéliser le terrain de jeu.