Projet de Synthèse d'Image et d'Algorithmique avancée en C

Année 2007

—IMAC première année—

Gestion d'une fourmilière

L'objectif sera, dans un premier temps, de segmenter une image en entrée représentant un graphe de fourmilière. La gestion au tour par tour du développement de la colonie pourra ensuite être implantée. Enfin, des fonctions de gestion du temps continu sont proposées pour aller plus loin.

Introduction

Une fourmilière est une structure enfouie constituée de tunnels et de salles aux attributions spécifiques : la salle de la reine, les réserves de nourriture, les fermes d'élevage, la sortie vers l'extérieur et les salles de transit. À l'état initial, la reine, immobile, règne. Elle met au monde des ouvrières qui la nourrissent et se nourrissent elles-même. Pour cela elles doivent ramasser la nourriture à l'extérieur ou traire les pucerons de la ferme. Une fois collectée, la nourriture est conservée dans des réserves en prévision de l'hiver car comme chacun le sait, la fourmi est prévoyante.

1 Format du fichier en entrée

En entrée, les paramètres sont transmis au programme via un fichier en mode texte. Son format est le suivant :

```
@FOURMIMAC 1
#premiere ligne de commentaire
#deuxieme ligne de commentaire
image=fichier.ppm
ciel=0 0 255
sol=128 128 128
reine=255 0 0
reserve=0 255 0
ferme=64 64 64
transit=255 255 0
sortie=0 0 0
saison=hiver
```

1.1 Version

La première ligne est donc `@FOURMIMAC`, suivi d'un espace, puis de la version du format de fichier, un entier :

```
@FOURMIMAC 1
```

1.2 Commentaires

Les lignes qui suivent et commençant par # sont des commentaires et doivent être ignorées :

```
#premiere ligne de commentaire
#deuxieme ligne de commentaire
#...
```

1.3 Image

Le nom du fichier image est ensuite indiqué :

```
image=fichier.ppm
```

1.4 Couleurs

La liste des couleurs affectées à chaque salle dans l'image est donnée sous la forme de triplets RGB (un entier entre 0 et 255 par composante).

Par exemple :

```
reine=255 0 0
```

signifie que la salle de la reine est rouge.

La couleur blanche est réservée : on ne peut pas l'utiliser pour définir une salle.

1.5 Saison

Le champs `saison` spécifie la saison au moment du début du jeu et peut prendre une des valeurs :
- `printemps`,
- `ete`,
- `automne`,
- ou `hiver`.

1.6 Validation du fichier d'entrée

Pour s'assurer que le fichier d'entrée est bien constitué, on vérifiera simplement que la première ligne est valide. Cette condition satisfaite, on supposera que :
- les champs sont donnés dans le bon ordre,
- les espaces et les sauts de lignes sont tels que spécifiés ici,
- la liste des couleurs est bien formée,
- l'image est un fichier qui décrit une fourmilière valide selon les critères définis dans la suite.

2 Construction du graphe

L'image représente la topologie du graphe. Elle est constituée de salles en couleur et de tunnels en blanc. Les centres des salles où les fourmis peuvent se rendre sont identifiés par un pixel blanc. Il y a deux salles spéciales dans lesquelles les fourmis ne peuvent pas aller :

1. le sol
2. et le ciel.

Dans la suite, ne sont désignés par *salles* que les zones où les fourmis peuvent se rendre, excluant ainsi le sol et le ciel. La figure **??** illustre le contenu d'une image d'entrée.

Votre mission consiste à segmenter cette image afin de reconstruire le graphe de la fourmilière. Dans ce graphe, les noeuds sont des salles et les arcs des tunnels.

3 Pondération des arcs

La pondération d'un arc se fait par l'évaluation de la longueur du chemin que parcourt une fourmi pour se rendre d'une salle à l'autre. Pour cela, on suppose que la fourmi se situe au centre de la salle de départ et se rend au centre de la salle d'arrivée. Son trajet se divise alors en trois étapes :

– du centre de la salle de départ à l'entrée du tunnel, elle se déplace en ligne droite,
– dans le tunnel, elle marche sur la paroi du sol ou du plafond,
– de la sortie du tunnel au centre de la salle d'arrivée, elle se déplace en ligne droite.

Ainsi, il y a donc deux poids possibles pour un arc selon que la fourmi emprunte le chemin du sol ou le chemin du plafond. Le poids de l'arc est la longueur normalisée du chemin le plus court. La figure 1 illustre les deux chemins envisageables pour une fourmi. Le poids des arcs est normalisé de sorte que :

$$\sum_{i=1}^{n} w_i = 1 \tag{1}$$

où n est le nombre total d'arcs dans le graphe et w_i le poids de l'arc i.

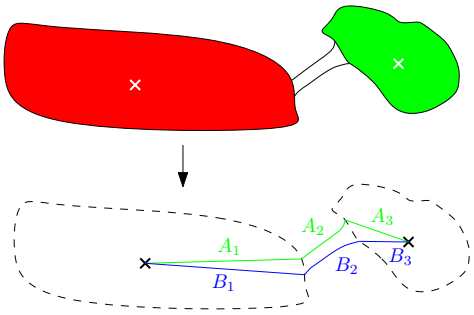

FIGURE 1: Pour se déplacer de la réserve à la maison de la reine, la fourmi emprunte le chemin le plus court entre $A_1 + A_2 + A_3$ et $B_1 + B_2 + B_3$. Attention, les étapes A_2 et B_2 ne sont pas effectuées en ligne droite mais le long des parois du tunnel.

4 Visualisation de la fourmilière et du graphe

Pour visualiser cette première partie, on écrira une application *openGL* et *glut*. Il peut s'agir de l'affichage de l'image d'origine ou d'une interprétation personnelle, ce qui est encouragé, mais alors les les propriétés fondamentales de la donnée image, à savoir qu'une "zone" est dans une salle ou un tunnel bien défini doivent être conservées. On pourra aussi visualiser le graphe et les poids des arcs en surimpression de la fourmilière.

5 Format de fichier version 2

Quand vous commencez à implanter cette nouvelle partie, le numéro de version du format de fichier passe de 1 à 2.

5.1 Nouveaux paramètres

Voici un exemple d'un fichier de version 2. Les points de suspension sont à remplacer par les paramètres de la version 1.

```
@FOURMIMAC 2
....
....
vitesse_fourmi=1e-2
duree_saison=10
duree_production_printemps=20.5
coeff_duree_production_ete=2.1
coeff_duree_production_automne=3.5
coeff_duree_production_hiver=45
coeff_duree_production_ferme=3.
production_ferme=1e3
production_exterieur=1e5
consommation_fourmi=1e-2
coeff_consommation_reine=5
duree_vie_fourmi=1024
resistance_fourmi=10.
coeff_alpha_nourriture=1.
coeff_alpha_reine=1.
coeff_alpha_conseil=1.
```

Le début du fichier reste donc inchangé mais les paramètres listés dans la figure 2 ont été ajoutés. Tous ces nouveaux paramètres sont des réels.

5.2 Unité de l'image

On suppose que l'image qui a été passée en paramètre représente une surface en m^{-2}. Plus précisément, un pixel de l'image est un carré de $4 * 10^{-6} \cdot m^{-2}$.

6 Vie et mort d'une fourmi "voyageuse"

6.1 Naissance

Seule la reine donne naissance à de nouvelles fourmis. Comme la reine ne se déplace pas, toutes les fourmis naissent dans la salle de la reine, au centre. On estime qu'à un instant t la probabilité que la reine ponde une fourmi est donné par la fonction $\mathscr{P}(t)$:

$$\mathscr{P}(t) = \frac{t - t_d}{t - t_d + \alpha} \tag{2}$$

où t_d est le temps écoulé depuis la dernière ponte et α est le "coefficient de fourmilière". La valeur de α dépend de la gestion de la fourmilière et est détaillé dans la section 6.6. La figure 3 présente un aperçu de

Nom	Paramètre	Unité S.I.	Tronqué
$\mathscr{V}_{fourmi}$	`vitesse_fourmi`	$m \cdot s^{-1}$	0
$\mathscr{D}_{saison}$	`duree_saison`	s	1
$\mathscr{D}_{printemps}$	`duree_production_printemps`	s	1
$\mathfrak{C}_{ete}$	`coeff_duree_production_ete`	sans unité	0
$\mathfrak{C}_{automne}$	`coeff_duree_production_automne`	sans unité	0
$\mathfrak{C}_{hiver}$	`coeff_duree_production_hiver`	sans unité	0
$\mathfrak{C}_{ferme}$	`coeff_duree_production_ferme`	sans unité	0
$\mathscr{P}_{ferme}$	`production_ferme`	J	0
$\mathscr{P}_{ext}$	`production_exterieur`	J	0
$\mathscr{C}_{fourmi}$	`consommation_fourmi`	$J \cdot s^{-1}$	0
$\mathfrak{C}_{reine}$	`coeff_consommation_reine`	sans unité	0
$\mathscr{D}_{vie}$	`duree_vie_fourmi`	s	1
$\mathscr{R}_{fourmi}$	`resistance_fourmi`	s	0
$\mathfrak{C}_n$	`coeff_alpha_nourriture`	sans unité	0
$\mathfrak{C}_r$	`coeff_alpha_reine`	sans unité	0
$\mathfrak{C}_c$	`coeff_alpha_conseil`	sans unité	0

FIGURE 2: Paramètres supplémentaires du format de fichier 2. La signification de la colonne “tronqué” est expliquée dans la section 7.2.

$\mathscr{P}(t)$ pour différentes valeurs de α. L'unité de $\mathscr{P}(t)$ est s^{-1}.

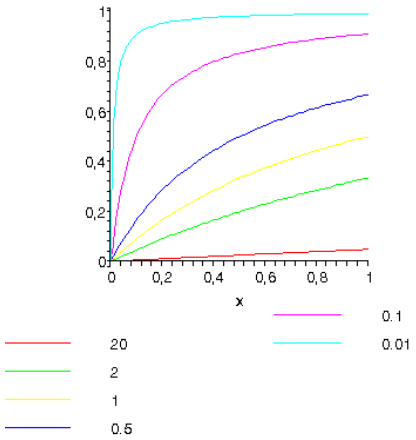

FIGURE 3
Allure de $\mathscr{P}(t)$ pour différentes valeurs de α en prenant $t_d = 0$.

6.2 Une vie de fourmi

La fourmi exécute la tâche qui lui a été affecte :
- ramasser de la nourriture à l'extérieur,
- traire les pucerons de la ferme,
- s'occuper de la reine dans la salle de la reine,

– ou attendre un ordre, qui est la tâche par défaut.

Une tâche consiste en plusieurs actions élémentaires effectuées de façon cyclique. Les tâches et actions d'une fourmi sont résumées dans la figure 4 et sont décrites dans les sous-sections 6.2.2 à 6.2.5.

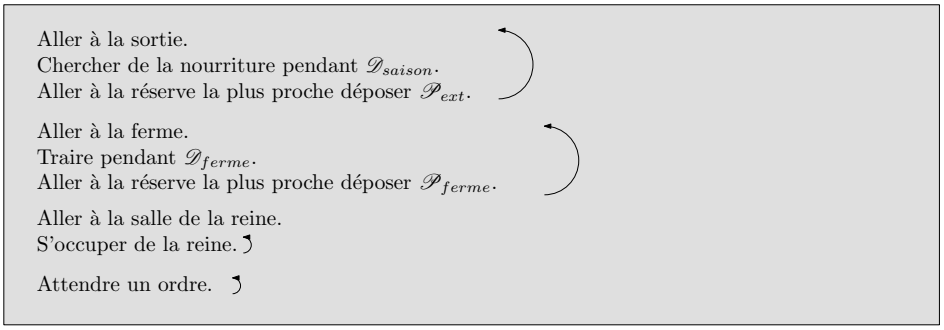

FIGURE 4
Tâches et actions d'une fourmi.

6.2.1 Déplacement

Certaines tâches impliquent un déplacement de la fourmi d'une salle à une autre, *a priori* non reliée directement par un tunnel. La procédure de déplacement est alors la suivante :

– identifier les salles intermédiaires,

– parcourir, d'une salle adjacente à une autre, la trajectoire calculée [(1)] dans la section 3.

Cette modélisation du déplacement des fourmis implique qu'une salle est représentée par son centre et nous l'accepterons. D'autre part, la vitesse d'une fourmi est donnée en $m \cdot s^{-1}$ et l'image en m^-2. Le déplacement doit respecter ces paramètres.

6.2.2 Collecte de la nourriture à l'extérieur

Pour cette tâche, la fourmi exécute un cycle de collecte à l'extérieur et de dépôt dans la réserve. L'extérieur n'est pas matérialisé et on suppose que quand la fourmi a atteint la salle de sortie, elle est à l'extérieur. La collecte prend un certain temps, qui dépend de l'abondance de nourriture, et la fourmi sera donc occupée pendant une durée $\mathscr{D}_{printemps}$, $\mathscr{D}_{ete}$, $\mathscr{D}_{automne}$ ou $\mathscr{D}_{hiver}$ selon la saison a laquelle commence la collecte. On à :

$$\mathscr{D}_{ete} = \mathscr{D}_{printemps}\mathfrak{C}_{ete} \tag{3}$$

$$\mathscr{D}_{automne} = \mathscr{D}_{printemps}\mathfrak{C}_{automne} \tag{4}$$

$$\mathscr{D}_{hiver} = \mathscr{D}_{printemps}\mathfrak{C}_{hiver} \tag{5}$$

où $\mathfrak{C}_{ete}$, $\mathfrak{C}_{automne}$ et $\mathfrak{C}_{hiver}$ sont les coefficients qui modélisent la durée nécessaire à la collecte de la quantité de nourriture $\mathscr{P}_{ext}$ comme un multiple du temps qu'il faut pour collecter cette quantité au printemps. Au terme de cette durée, la fourmi rapporte donc la quantité $\mathscr{P}_{ext}$ (en Joules) de nourriture à la réserve.

6.2.3 Traite des pucerons

Pour les fourmis, une autre façon de produire de la nourriture est de traire des pucerons dans la ferme. La production de la ferme ne dépend pas de la saison car celle-ci est à l'abri dans la fourmilière. Par contre, le

(1). Il pourra alors être judicieux, une fois calculée, de conserver cette trajectoire.

nombre de fourmis pouvant être affectées à cette tâche est limité par le nombre de pixels dans la salle de la ferme. La production est telle que, au bout d'une durée

$$\mathscr{D}_{ferme} = \mathscr{D}_{printemps}\mathfrak{C}_{ferme} \tag{6}$$

la fourmi rapporte à la réserve la quantité de nourriture $\mathscr{P}_{ferme}$.

6.2.4 S'occuper de la reine

S'occuper de la reine modélise toutes les activités liées au confort de la reine, à l'entretien de la fourmilière, à la nourriture des larves et au temps passé à fourbir des plans machiavéliques pour conquérir le monde. Lorsqu'une fourmi s'occupe de la reine, elle reste simplement dans la salle de la reine.

6.2.5 Attendre

Une fourmi qui attend ne fait rien tant que l'on ne lui donne pas une tâche à effectuer.

6.3 Réserve de nourriture

6.3.1 Augmentation de la réserve

Quand une fourmi a fini de produire de la nourriture, elle la dépose dans la réserve la plus proche. Le compteur global de nourriture de la fourmilière $\mathscr{N}_{reserve}$ augmente de $\mathscr{P}_{ferme}$ ou $\mathscr{P}_{ext}$ selon que la nourriture provient de la ferme ou de l'extérieur.

6.3.2 Diminution de la réserve

La nourriture est consommée par toute la colonie. La consommation d'une fourmi est de $\mathscr{N}_{fourmi}$ par unité de temps, la consommation de la reine est un multiple de la consommation d'une fourmi donné par $\mathscr{N}_{fourmi}\mathfrak{C}_{reine}$. La réserve de nourriture diminue donc de

$$\mathscr{N}_{fourmi}(\mathscr{F}-1) + \mathscr{N}_{fourmi}\mathfrak{C}_{reine} \tag{7}$$

par seconde. $\mathscr{F}$ est la population totale de la fourmilière, reine comprise. $\mathscr{N}_{reserve}$ ne peut pas diminuer en dessous de zéro.

6.4 Mort d'une fourmi

6.4.1 Vieillesse

La durée de vie d'une fourmi, excepté la reine, est fixée à $\mathscr{D}_{vie}$ secondes. Quand une fourmi atteint cet âge avancé elle est retirée du jeu.

6.4.2 Famine

Tant que $\mathscr{N}_{reserve}$ est égal à zéro, la colonie est en famine. Chaque fourmi, hormis la reine, a, quel que soit son âge, la probabilité $\frac{1}{\mathscr{R}_{fourmi}}$ par seconde de disparaître.

6.5 Conseil de la fourmilière

À chaque fois qu'elle le souhaite, pour motiver ses troupes, la reine peut convoquer le *Conseil de la Fourmilière*. Toutes les fourmis suspendent alors leur activité et se rendent dans la salle de la reine.

6.6 Calcul du coefficient de la fourmilière α

La gestion de la fourmilière consiste à optimiser $\mathscr{P}(t)$ pour que la reine ponde souvent. On va modéliser α de sorte que la reine ponde plus quand :

- il y a beaucoup de nourriture en réserve relativement à la consommation,
- il y a de beaucoup de fourmis qui s'occupent d'elle,
- un *Conseil de la Fourmilière* a eu lieu récemment.

Pour cela, on utilisera la formule de l'équation 8 :

$$\alpha = \frac{1}{\mathfrak{C}_n \frac{\mathscr{N}_{reserve}}{\mathscr{N}_{fourmi}(\mathscr{F}-1)+\mathscr{N}_{fourmi}\mathfrak{C}_{reine}} + \mathfrak{C}_r \mathscr{F}_r - \mathfrak{C}_c t_c} \tag{8}$$

Où :

- α est le coefficient de la fourmilière,
- $\mathfrak{C}_n$ est le coefficient de nourriture,
- $\mathscr{N}_{reserve}$ est la quantité de nourriture en réserve,
- $\mathscr{F}$ est la population totale de la fourmilière,
- $\mathscr{N}_{fourmi}$ est la consommation d'une fourmi,
- $\mathfrak{C}_{reine}$ est le coefficient de consommation de la reine,
- $\mathfrak{C}_r$ est le coefficient de confort de la reine,
- $\mathscr{F}_r$ est le nombre de fourmis qui s'occupent de la reine,
- $\mathfrak{C}_c$ est le coefficient du *Conseil de la Fourmilière*,
- t_c est le temps écoulé depuis le dernier *Conseil de la Fourmilière*.

Avec α, $\mathfrak{C}_n$, $\mathfrak{C}_r$ et $\mathfrak{C}_c$ supérieur à 0.

7 Le temps

7.1 Évolution des saisons

Toutes les $\mathscr{D}_{saison}$ secondes, la saison change, modifiant le temps nécessaire pour récolter la quantité $\mathscr{P}_{ext}$ de nourriture à l'extérieur (et éventuellement, modifiant l'aspect de la fourmilière).

7.2 Gestion informatique

7.2.1 Tour par tour

Le jeu se déroule au tour par tour. Lorsque le joueur appuie sur la touche d'espacement, le temps augmente d'une seconde. Les paramètres liées au temps dans le tableau 2 sont tronquées à l'entier.

7.2.2 En continu...

Dans un deuxième temps, si vous le souhaitez, vous pouvez expérimenter le temps continu à l'aide des fonctions décrites ici. Pour gérer la dynamique du temps en continu vous utiliserez *glutIdleFunc* et la fonction [(2)] `temps_ecoule()` qui permet de récupérer le temps en secondes écoulé depuis le début de la simulation.

```
#include <stdlib.h>
```

(2). La fonction , ou temps_ecoule() est donnée à titre informatif. De la même façon qu'il n'est pas nécessaire de connaître les arcanes de la fission nucléaire pour utiliser un sèche-cheveux et le fonctionnement interne de la fonction printf() pour afficher un caractère, il n'est pas nécessaire de la comprendre, ni de la taper, elle est disponible sur la page du projet.

```
#include <sys/time.h>

double temps_ecoule()
{
  static int init=1;
  static struct timeval temps_initial;
  if(init)
    {
      gettimeofday(&temps_initial,0);
      init = 0;
      return 0.;
    }
  struct timeval temps_fourmi;
  gettimeofday(&temps_fourmi,0);
  double mili_secondes = (temps_fourmi.tv_sec - temps_initial.tv_sec) * 1000;
  if (temps_fourmi.tv_usec > temps_initial.tv_usec)
    mili_secondes += (temps_fourmi.tv_usec - temps_initial.tv_usec) / 1000.;
  else
    mili_secondes -= (temps_initial.tv_usec - temps_fourmi.tv_usec) / 1000.;
  return mili_secondes / 1000;
}
```

7.2.3 Exemple

On souhaite savoir si une fourmi a atteint l'âge de son retrait du jeu. Supposons que $\mathscr{D}_{vie}$ (c.f. section 6.4.1) soit :

```
double duree_de_vie=20.;
```

Si la date de naissance de la fourmi est :

```
double naissance=2412.00;
```

Alors, à chaque passage dans la callback définie par *glutIdleFunc*, on vérifie :

```
if( temps_ecoule() - naissance >= duree_de_vie)
/* la fourmi doit etre retiree du jeu */
```

7.2.4 Gestion de la ponte

La fonction $\mathscr{P}(t)$ (c.f. section 2), est une probabilité définie "par unité de temps".

C'est un problème !

Pour l'illustrer, imaginons que $\mathscr{P}(t)$ soit une constante $\mathscr{P}(t) = 1$. On s'imagine alors générer une fourmi par seconde. De la même façon, si $\mathscr{P}(t) = 0.5$ on s'attend à générer une fourmi toute les deux secondes.

Revenons au cas $\mathscr{P}(t) = 1$. Si on teste $\mathscr{P}(t)$ plusieurs fois dans la même seconde, on va générer autant de fois une fourmi que l'on aura fait de test puisqu'à chaque fois, $\mathscr{P}(t)$ nous dit que la probabilité est 1. Deux solutions s'offrent à nous pour palier à ce problème :

– s'assurer que l'on "teste" $\mathscr{P}(t)$ exactement une fois par seconde,

– évaluer et “tester” l'intégrale $\int_{t_0}^{t} \mathscr{P}(t)dt$ ou t_0 est le temps du dernier “test”.
C'est la deuxième solution qui est mise en oeuvre dans la fonction `pond`.

```
#include <stdlib.h>
#include <sys/time.h>
#include <time.h>

int pond(double probabilite, double temps)
{
  static int init = 1;
  static double temps_precedent;
  static double precedente_probabilite;
  if(init)
    {
      srand(time(NULL));
      temps_precedent=0.;
      precedente_probabilite=0.;
      temps_precedent = 0;
      init = 0;
    }
  double delta_t =  temps - temps_precedent;
  double aire_rectangle = delta_t * precedente_probabilite ;
  double aire_triangle = delta_t * ( probabilite - precedente_probabilite ) / 2. ;
  double integrale = aire_rectangle + aire_triangle  ;
  temps_precedent = temps;
  double r = ((double)rand() / ((double)RAND_MAX + 1));
  if( r >= integrale )
    {
      precedente_probabilite = probabilite;
      return 0;
    }
  precedente_probabilite = 0.;
  return 1;
}
```

7.2.5 Exemple

La fonction `pond(..)` s'utilise très simplement [3]. À un certain temps `t=temps_ecoule()`, on veut savoir si la reine pond, la fonction $\mathscr{P}(t)$ nous donne `probabilite` et l'appel `pond(probabilite, t)` renvoie 1 si la reine doit pondre, 0 sinon.

À propos de l'interface

Pour l'interface, une fourmi pourrait être modélisée par un carré de couleur, on la sélectionne en cliquant dessus ou a proximité et on lui affecte une tâche en cliquant sur la salle concernée. On pourra aussi utiliser la molette de la souris, la fourmi-molette en l'occurrence. Si le modèle de la fourmi s'étend sur plusieurs pixels pour des raisons esthétiques, son centre de gravité et sa position restent ponctuels. On pourra utiliser la console pour entrer des commandes et afficher des informations.

(3). Même remarque que pour la fonction temps_ecoule() mais scanf().
avec une génératrice hydraulique un grille-pain et la fonction