



ÉCOLE POLYTECHNIQUE DE L'UNIVERSITÉ DE TOURS
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. (33)2-47-36-14-14
Fax (33)2-47-36-14-22
www.polytech.univ-tours.fr

Rapport Final

Fourmi de Langton

Auteur(s)

Alizée Buatois
[alizee.buatois@etu.univ-tours.fr]
Valentin Montmirail
[valentin.montmirail@gmail.com]

Encadrant(s)

Nicolas Monmarché
[nicolas.monmarche@univ-tours.fr]

Table des matières

Introduction

Le but de ce projet était de réaliser un automate cellulaire en 3D qui était à la base créé pour être utilisé en 2D.

Cet automate s'appelle La Fourmi de Langton, et il permet de mettre en évidence un comportement émergent. Nous voulions à travers ce projet, voir si ce comportement émergent se produisait aussi en 3D.

Un grand merci à Heiko Hamann (heiko.hamann@uni-paderborn.de), qui nous a donné des pistes afin de simplifier notre code et grâce à ses conseils, ce projet a pu être mené à bien.

Un merci aussi à Nicolas Monmarché (nicolas.monmarche@univ-tours.fr), qui grâce à ses questions/conseils, a réussi à éviter que l'on se perde dans des détails d'implémentation au lieu de se focaliser sur l'algorithme principal du déplacement de cet automate.

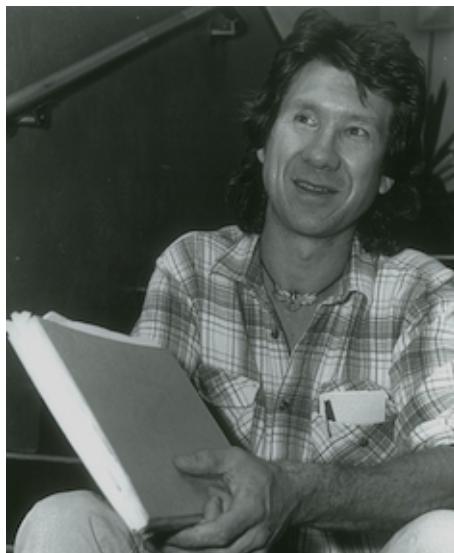
Un grand merci finalement, à la K'Fet du Département Informatique qui grâce à ses cafés et ses thés, nous a permis de débugger ce projet et de le rendre opérationnel.

CHAPITRE 1

Qu'est-ce que la Fourmi de Langton ?

1.1 Qu'est-ce que la Fourmi de Langton à l'Origine

La Fourmi de Langton est à l'origine un automate cellulaire bidimensionnel programmable. Inventée par l'américain Christopher Langton en 1986,



elle comporte un jeu de règles très simples (Left et Right) dans un repère composé de cases de deux couleurs différentes : noir et blanc.

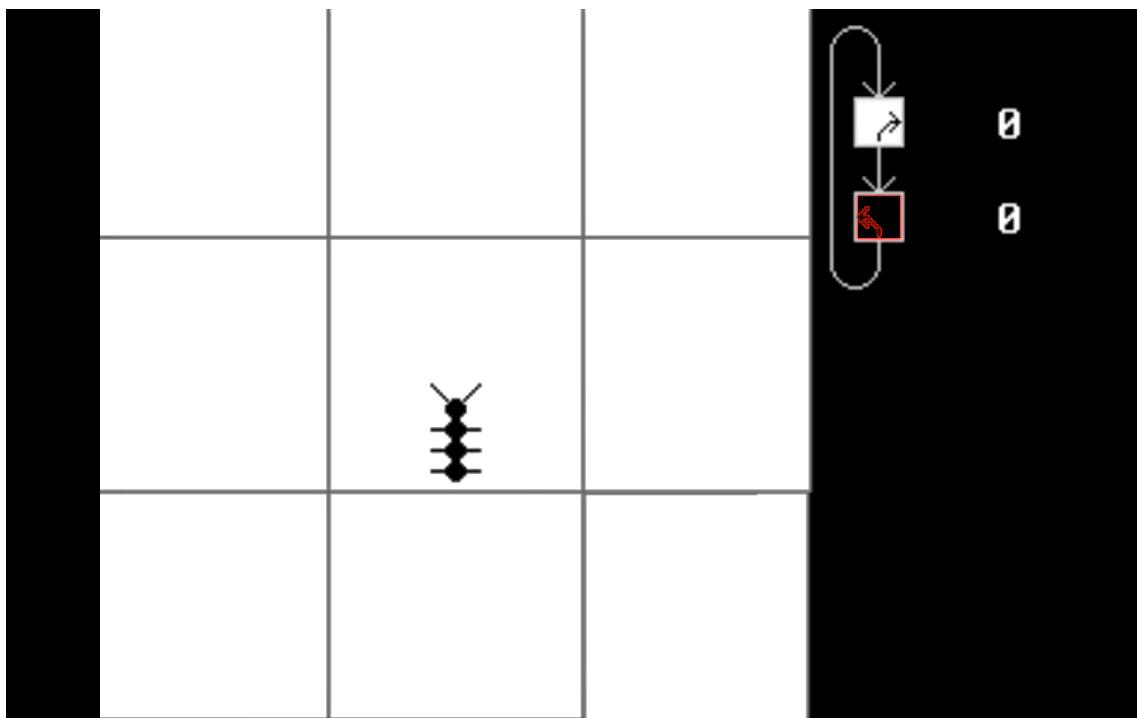
Les règles sont les suivantes : RL

Une case est définie arbitrairement comme case initiale.

-Si la fourmi est sur une case blanche, elle tourne de 90 degrés vers la droite (R), change la couleur de la case en noir, et avance d'une case.

-Si la fourmi est sur une case noire, elle tourne de 90 degrés vers la gauche (L), change la couleur de la case en blanc, et avance d'une case.

La fourmi subit donc les règles de déplacement associées aux cases qui sont colorées.

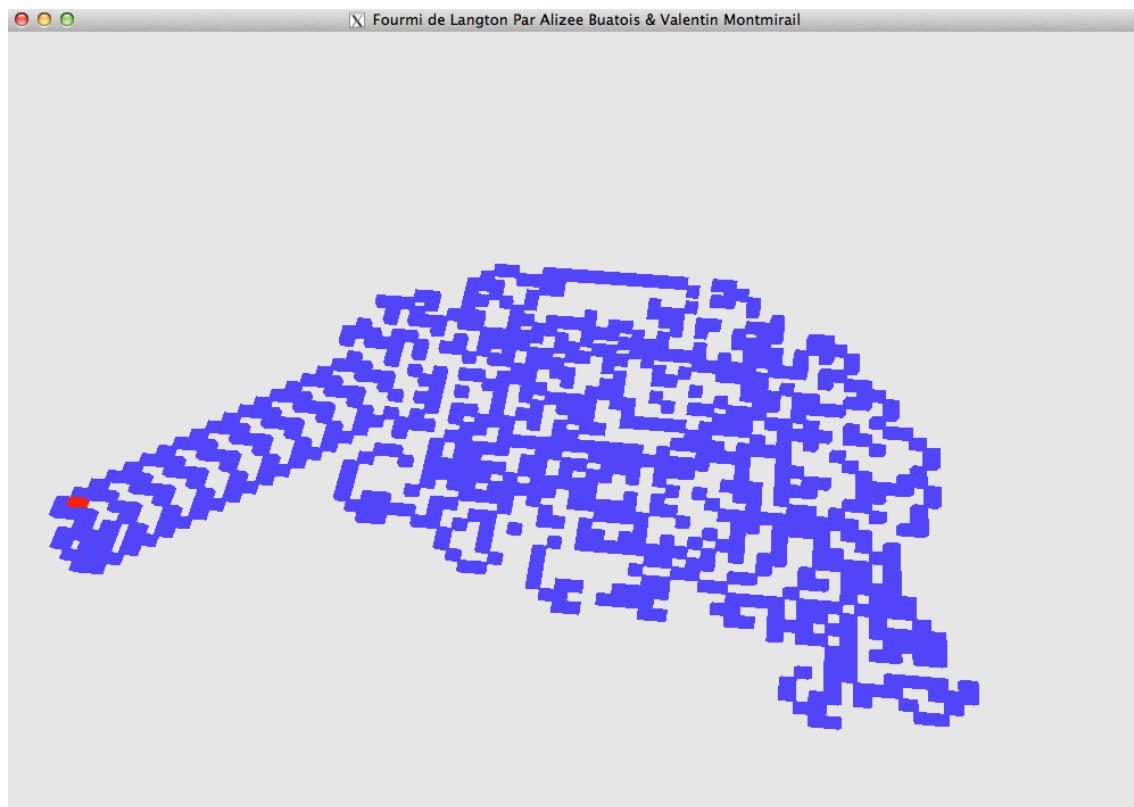


En déroulant ces règles de déplacement, après un certain de nombre de déplacements chaotiques de la fourmi, un comportement en "autoroute" est apparent.

En effet, pour un jeu de règles prédéfini, la fourmi aura toujours le même comportement dans un monde coloré donné (par défaut le monde est rempli de cases blanches), et quelquesoient le monde initial, cette "autoroute" apparaitra toujours.

Cet automate permet donc de mettre en évidence un comportement émergent.

L'apparition de cette autoroute est à l'heure actuelle toujours au stade de conjecture, aucune démonstration mathématique n'ayant expliqué l'origine de son apparition, mais aucun contre exemple ne prouve le contraire non plus.



1.2 Comment nous avons amélioré la Fourmi pour qu'elle fonctionne en 3D

Pour définir les règles de la Fourmi de Langton en 3D, nous avons bien évidemment du prendre en compte les règles déjà définies en 2D, ce qui implique certaines contraintes. En effet, les règles ainsi définies nous donnent une première condition essentielle :

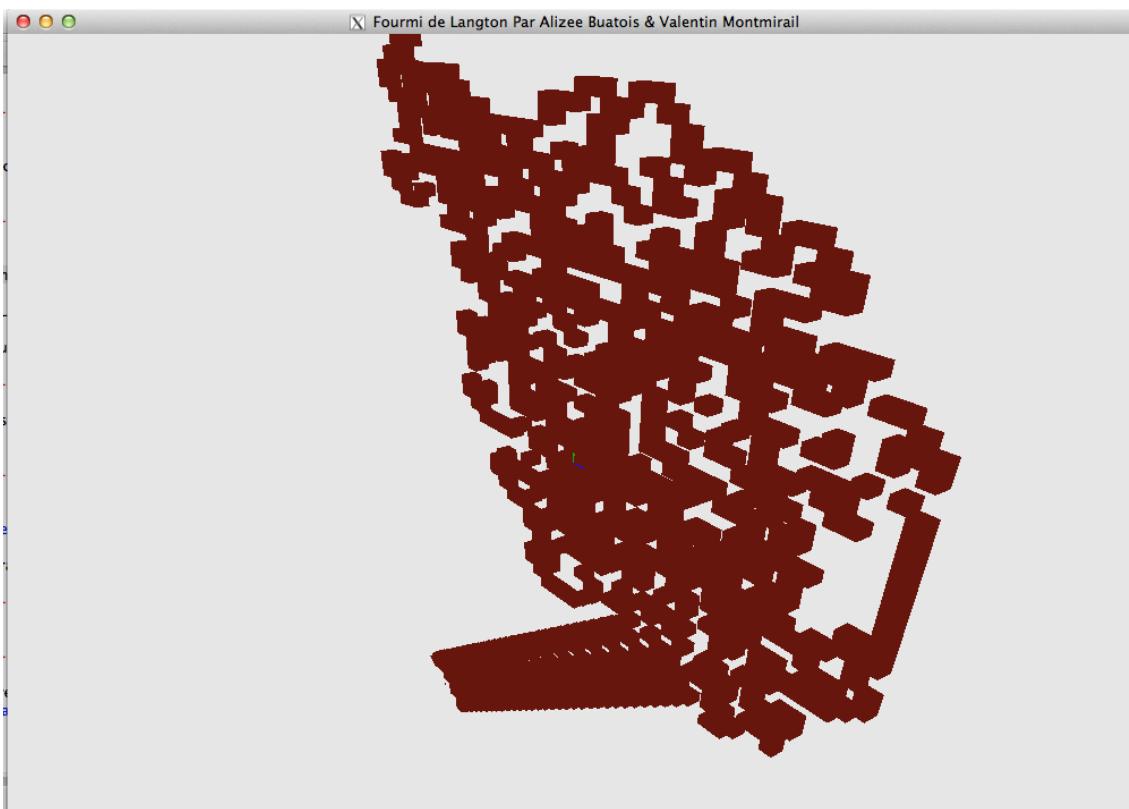
-répéter 4 fois la même règle (la fourmi subit 4 fois de suite une case contenant la même règle) renvoie la fourmi sur sa case de départ, ce qui est normal car elle aura effectué 4 fois de suite une rotation de 90 degrés dans le même sens (=rotation de 360 degrés : vecteur nul).

Ensuite, nous avons du créer deux nouvelles règles (Up et Down) pour créer cette 3D. Ces deux nouvelles règles U et D seront donc dans un plan orthogonal au plan contenant les règles R et L.

En fait, exécuter les règles U et D doit donner le même schéma de déplacement de la fourmi que celui donné dans le plan de départ avec les règles R et L.

Les règles U et D seront donc dans un plan orthogonal au plan contenant les règles R et L.

Les règles U et D correspondent donc aux règles R et L dans un autre plan.



Voici le déplacement de la fourmi avec les règles U et D, que l'on peut comparer avec l'image précédente montrant le déplacement de la fourmi avec les règles L et R : les déplacements forment un dessin identique mais dans des plans différents.

1.3 Explications des Directions

Pour la Fourmi de Langton classique, qui se déplaçait dans le Plan, seules 4 directions étaient nécessaires :

Devant

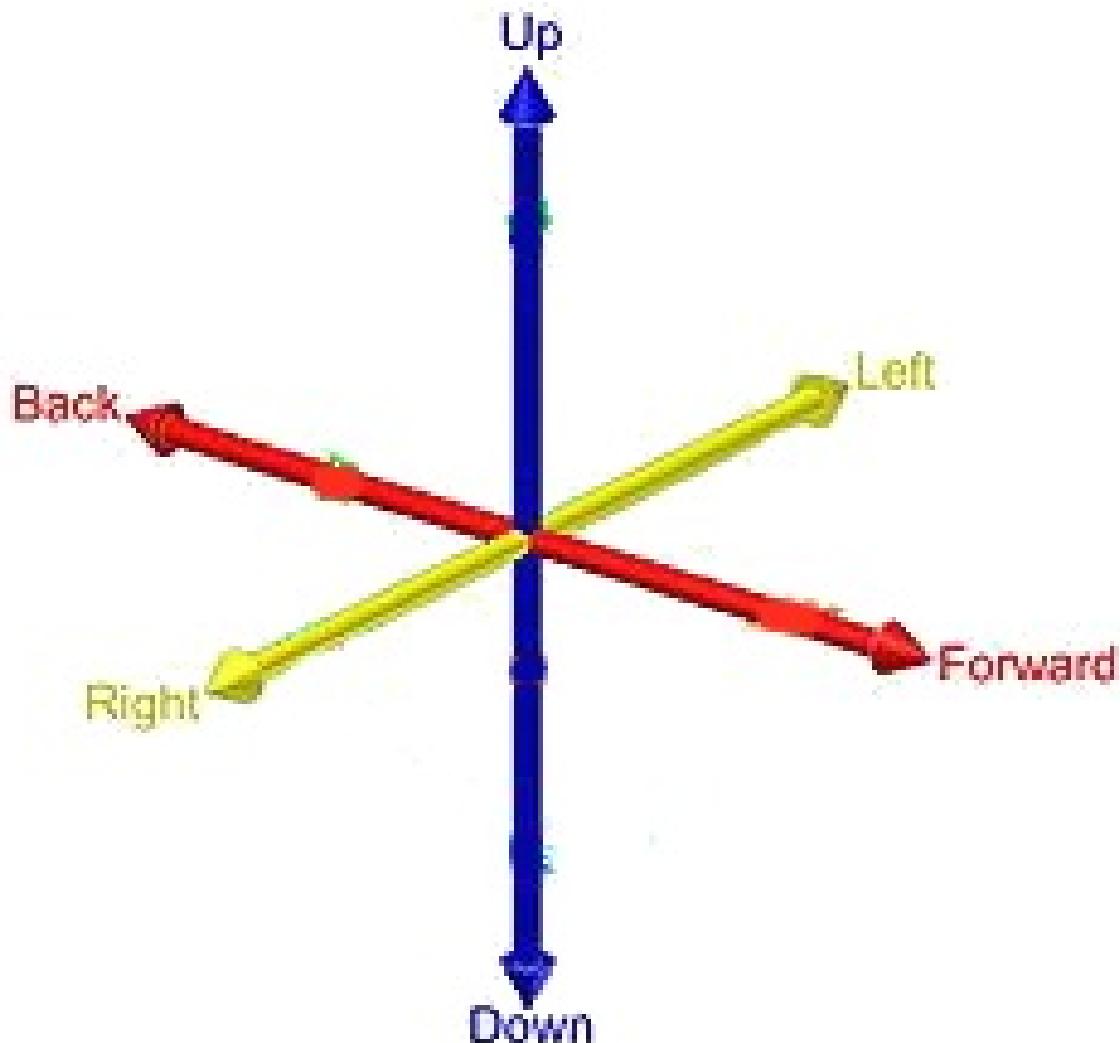
Derrière

Gauche

Droite

Seulement comme nous l'avons vu dans la partie précédentes, nous avons fait passer la Fourmi de Langton en 3D, et pour ce faire, nous avons eu besoin de 2 directions différentes, ces deux directions sont : Haut et Bas et ce sont ces dernières qui nous permettent de passer du plan vers l'espace.

Comme une image vaut 1000 mots, voici l'ensemble des directions possibles à chaque instant pour la Fourmi de Langton.



Nous verrons dans la partie suivante comment chaque règle fait changer les directions. Mais pour le moment, nous savons que la Fourmi, à tout instant t, a 6 choix possibles :

Haut

Bas

Gauche

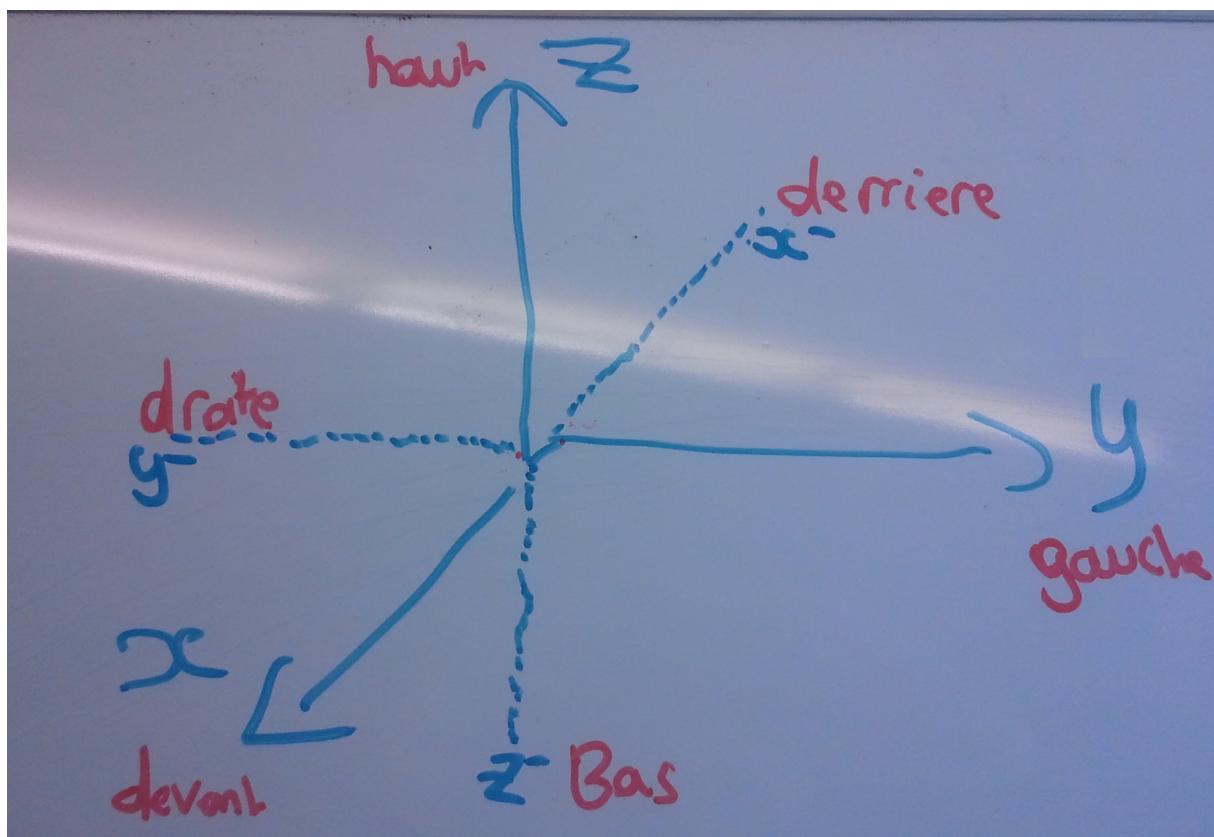
Droite

Devant

Derrière

Mais techniquement, ce ne sont que les coordonnées de la Fourmi qui évoluent, l'histoire de directions (puis de plan que nous verrons plus tard) nous sert simplement à décider quelles coordonnées vont évoluer et quelle sera la direction suivante.

Pour cela, il faut donc définir à quel axe du repère est parallèle à ce que nous appelons depuis le début le "Devant".



On voit donc que le Devant est en fait une direction parallèle à l'axe des X+. Ce qui signifie qu'en fonction de la direction de la Fourmi

- si la direction est GAUCHE , les y de la Fourmi vont être décrémentés.
- si la direction est DEVANT , les x de la Fourmi vont être incrémentés.
- si la direction est DROITE , les y de la Fourmi vont être incrémentés.
- si la direction est DERRIERE , les x de la Fourmi vont être décrémentés.
- si la direction est HAUT , les z de la Fourmi vont être incrémentés.
- si la direction est BAS , les z de la Fourmi vont être décrémentés.

Nous allons voir maintenant comme chaque règle change la direction de la Fourmi.

1.4 Explications des Règles

Nous avons vu précédemment qu'en fonction de la direction courante de la Fourmi, ses coordonnées changeaient. Par contre, pour éviter les boucles infinies, il faut bien évidemment que la direction de la Fourmi change aussi.

Dans la pratique, la direction change en fonction de la direction précédente, mais aussi du repère courant.

Mais pour simplifier ce rapport, nous tiendrons compte dans cette partie simplement du Repère 0, qui est le repère (X+, Y+, Z+).

Pour trouver les changements de directions dans les autres plans, il suffit simplement de faire une comparaison par rapport au plan 0.

1.4.1 Règle L(**eft**)

Si la règle de la case est L alors :

Si la direction courante est DEVANT alors

La direction devient : GAUCHE

Si la direction courante est DERRIERE alors

La direction devient : DROITE

Si la direction courante est GAUCHE alors

La direction devient : DERRIERE

Si la direction courante est DROITE alors

La direction devient : DEVANT

Si la direction courante est HAUT alors

La direction devient : GAUCHE

Si la direction courante est BAS alors

La direction devient : GAUCHE

1.4.2 Règle R(ight)

Si vous avez suivi ce rapport, la Règle Right est parfaitement symétrique par rapport à la Règle Left.

Si la règle de la case est R alors :

Si la direction courante est DEVANT alors

La direction devient : DROITE

Si la direction courante est DERRIERE alors

La direction devient : GAUCHE

Si la direction courante est GAUCHE alors

La direction devient : DEVANT

Si la direction courante est DROITE alors

La direction devient : DERRIERE

Si la direction courante est HAUT alors

La direction devient : DROITE

Si la direction courante est BAS alors

La direction devient : DROITE

1.4.3 Règle U(p)

Les Règles UP et DOWN, en plus de changer les directions, change aussi le repère, mais nous allons traiter ce cas dans une autre partie, ici nous tiendrons simplement compte du changement de directions que font subir ces règles.

Si la règle de la case est U alors :

La direction devient HAUT, qu'importe la direction précédente, dans le repère de base.

1.4.4 Règle D(own)

Les Règles UP et DOWN, en plus de changer les directions, change aussi le repère, mais nous allons traiter ce cas dans une autre partie, ici nous tiendrons simplement compte du changement de directions que font subir ces règles.

Si la règle de la case est D alors :

La direction devient BAS, qu'importe la direction précédente, dans le repère de base.

1.5 Pourquoi les règles U et D font changer de plan

Notre fourmi se trouve donc dans un premier temps dans le plan (Oxy), les pieds en direction du BAS, le haut de la tête côté HAUT, les directions DEVANT et DERRIERE (respectivement X+ et X-) GAUCHE et DROITE (Y+ et Y-).

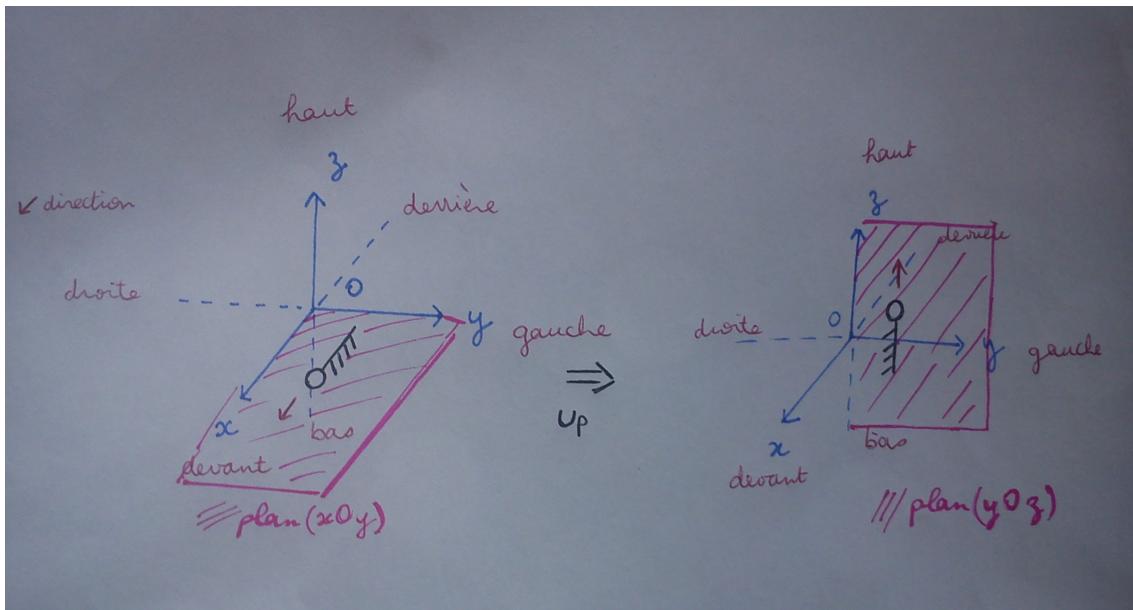
IL faut considérer que ces directions sont fixes dans ce repère 0 que nous notons (X+, Y+, Z+).

En effet, si la fourmi n'exécute que les règles R et L, elle gardera les pattes en direction du BAS (Z-), on restera donc dans le plan (Oxy), et par conséquent dans le repère 0.

Or, dès lors que la fourmi exécutera U ou D, la fourmi aura les pattes en direction de DERRIERE (X-), DROITE (y-), DEVANT (x+) ou GAUCHE (y+) selon la direction que la fourmi avait avant de faire U ou D.

Par exemple si la fourmi avait une direction de DEVANT avant de faire U, ses pattes seront en direction de DEVANT (son ancien BAS deviendra DEVANT) (cf figure), si elle avait une direction de GAUCHE avant de faire U, ses pattes seront en direction de GAUCHE (son ancien BAS deviendra GAUCHE), etc.

C'est pourquoi la fourmi sera alors dans un autre plan ((Oyz) ou (Oxz)) selon la direction qu'avait la fourmi).



1.6 Les Changements de Repère

Pour éviter de se perdre et de devoir traiter tous les cas (quoi devient quoi) afin que les déplacements restent concordants avec la droite, la gauche le devant et le derrière propres à la fourmi, nous avons décidé de faire des changements de repères afin que les directions choisies restent les mêmes. Voici les différents repères utilisés, définis à partir du repère initial 0 :

$$0 = (X+, Y+, Z+)$$

$$1 = (X+, Y-, Z-)$$

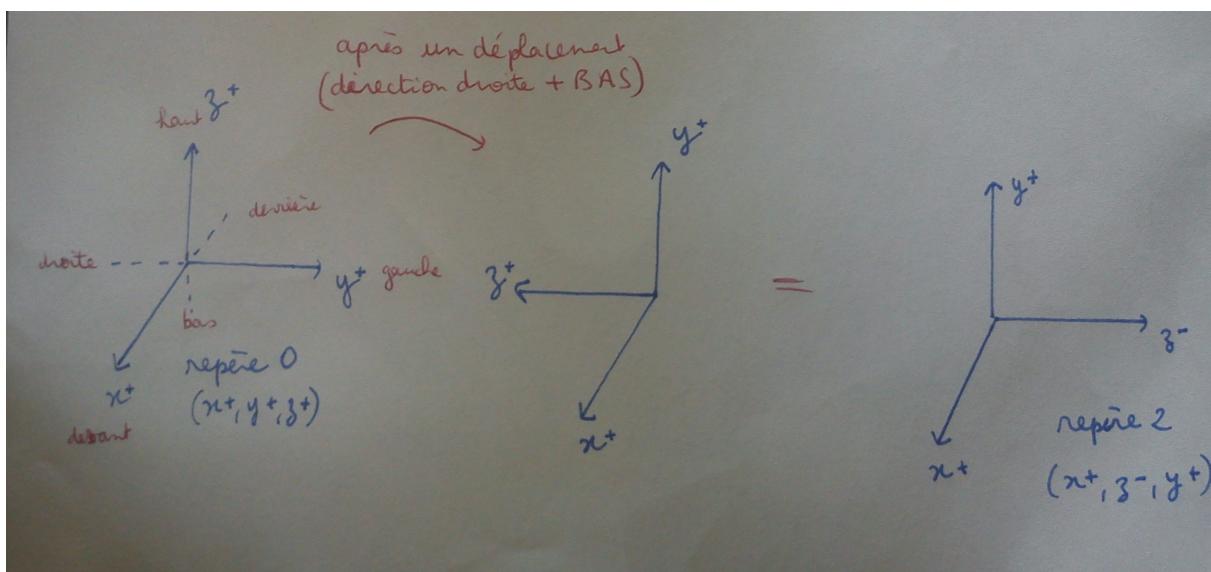
$$2 = (X+, Z-, Y+)$$

$$3 = (X+, Z+, Y-)$$

$$4 = (Z-, Y+, X+)$$

$$5 = (Z+, Y+, X-)$$

Ces repères sont au nombre de 6 car lorsque la fourmi avance d'une case et subit une des quatre règles (R, L, U, D), elle a 6 directions possibles qui peuvent lui être attribuées. Exemple de passage d'un plan à un autre :



Ces repères ont été un peu fastidieux à choisir, car il faut une bonne vision 3D. Cette partie nous a pris plus de temps que prévu, car elle était en fait au cœur des déplacements de notre fourmi.

Nous avons fait en sorte que le repère suive la fourmi, afin que l'axe des Z reste toujours le haut pour elle, et qu'ainsi nos règles de déplacements concordent avec celles définies dans un plan (Fourmi de Langton en 2D).

Nous avons donc ensuite du réfléchir à ce que devenaient nos directions. En effet la fourmi de Langton en 2D a la particularité de changer de direction lorsqu'elle effectue une rotation de 90 degrés après un déplacement.

Mais ici en 3D, en changeant de repère nous avons du dérouler tous les cas pour savoir quels déplacements doit faire la fourmi, et quelle direction lui attribuer.

Voici donc quelques exemples :

Dans le repère 0, La fourmi en direction de DEVANT et marchant sur une case qui lui dit de faire Up fait Z++ et sa nouvelle direction devient alors HAUT, et elle change de plan.

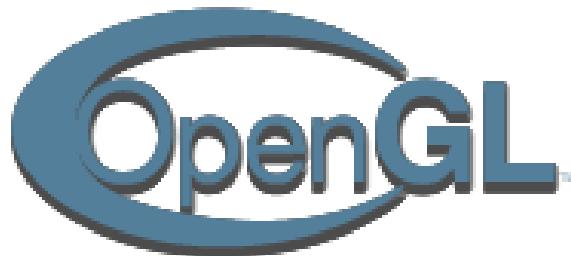
Dans le repère 3, La fourmi en direction de DROITE et marchant sur une case qui lui dit de faire Down fait Z-, et sa nouvelle direction devient DERRIERE. Etc...

Après avoir défini tous ces déplacements un par un (les déplacements en X et en Y étant toujours les mêmes quelque soient le repère, mais les directions variant à chaque étape), nous avons pu constater que la Fourmi de Langton fonctionnait en 3D dans notre monde !

CHAPITRE 2

Comment nous avons réalisé la Fourmi de Langton

2.1 Pourquoi avoir utilisé GLUT/OpenGL



Pour plus de simplicité au niveau du code, nous avons choisi d'utiliser la librairie OpenGL et la bibliothèque GLUT, car elle était appropriée à ce type de programme.

L'OpenGL est faite pour la conception d'applications générant des images 3D et 2D, facile d'utilisation.

GLUT est une interface entre le programme et l'humanoïde qui se trouve derrière l'écran, la souris, ou le joystick.

Sa programmation est de type événementielle et chaque agitation de l'humain est considéré comme un événement qui sera géré par des fonctions personnalisées.

Pour l'utiliser, il suffit d'inclure « glut.h » au programme et compiler le projet avec la librairie « glut32.lib ».

La création d'un programme avec GLUT ne prend que quelques lignes de code. Par ailleurs, elle respecte les conventions et la syntaxe d'OpenGL ; les noms de fonction commencent par le préfixe « glut » et les macros par « GLUT ».

Comme GLUT propose des routines pour le dessin de quelques objets simples à trois dimensions (sphère, cube...), cela était approprié pour notre projet : le dessin de cubes pour le déplacement de la fourmi.

2.2 Algorithme de Déplacement de la Fourmi

2.2.1 Version en Pseudo Code

Expliquons maintenant, grâce à l'algorithme que nous avons utiliser, comment se déplace concrètement notre Fourmi.

Algorithm 1 Algorithme de Déplacement de la Fourmi**Require:** f une Fourmi**Ensure:** \emptyset , La Fourmi a la bonne direction et elle est dans le bon plan.

CaseDuMonde : entier ;

```
1: if ( $f->direction = GAUCHE$ ) then
2:      $f->y <- f->y - 1$  ;
3: else if ( $f->direction = DEVANT$ ) then
4:      $f->x <- f->x + 1$  ;
5: else if ( $f->direction = DROITE$ ) then
6:      $f->y <- f->y + 1$  ;
7: else if ( $f->direction = DERRIERE$ ) then
8:      $f->x <- f->x - 1$  ;
9: else if ( $f->direction = HAUT$ ) then
10:     $f->z <- f->z + 1$  ;
11: else if ( $f->direction = BAS$ ) then
12:     $f->z <- f->z - 1$  ;
13: end if
```

CaseDuMonde $<- \text{regles}[\text{monde}[f->x][f->y][f->z]]$;

```
14: if ( $CaseDuMonde = 0$ ) then
15:     /* La case demande de faire 90 degrés à Gauche */
16:     Gauche( $f$ ) ;
17: else if ( $CaseDuMonde = 1$ ) then
18:     /* La case demande de faire 90 degrés à Droite */
19:     Droite( $f$ ) ;
20: else if ( $CaseDuMonde = 2$ ) then
21:     /* La case demande de faire 90 degrés à Haut */
22:     Haut( $f$ ) ;
23: else if ( $CaseDuMonde = 3$ ) then
24:     /* La case demande de faire 90 degrés à Bas */
25:     Bas( $f$ ) ;
26: end if
```

```
27: monde[f->x][f->y][f->z]  $<- \text{monde}[f->x][f->y][f->z] + 1$  ;
28: monde[f->x][f->y][f->z]  $<- \text{monde}[f->x][f->y][f->z] \bmod \text{nbRegles}$  ;
```

2.2.2 Version en C

```
void subir_la_regle_associee_a_la_case(struct Fourmi* f) {  
  
    switch(f->direction) {  
  
        case GAUCHE: f->y--;  
        break;  
        case DEVANT: f->x++;  
        break;  
        case DROITE: f->y++;  
        break;  
        case DERRIERE: f->x--;  
        break;  
        case HAUT: f->z++;  
        break;  
        case BAS: f->z--;  
        break;  
    }  
  
    if (regles[monde[f->x][f->y][f->z]] == 0) {  
  
        fprintf(fichierLog, "Gauche\n");  
        Gauche(f);  
    }  
    else if (regles[monde[f->x][f->y][f->z]] == 1) {  
  
        fprintf(fichierLog, "Droite\n");  
        Droite(f);  
    }  
    else if (regles[monde[f->x][f->y][f->z]] == 2) {  
  
        fprintf(fichierLog, "Haut\n");  
        Haut(f);  
    }  
    else if (regles[monde[f->x][f->y][f->z]] == 3) {  
  
        fprintf(fichierLog, "Bas\n");  
        Bas(f);  
    }  
  
    monde[f->x][f->y][f->z]++;  
    monde[f->x][f->y][f->z]%=nbRegles;  
  
    glutPostRedisplay();  
}
```

2.2.3 Explication du Code

Maintenant que vous avez vu le code sous sa forme abstraite en Pseudo-Code et sous sa forme concrète avec les contraintes du langage C et OpenGL.

Voici maintenant les explications des variables et du code en général :

Nous avons utilisé dans le code de déplacement : 2 variables très importantes.

'Regles', qui est un tableau stockant l'ensemble des règles que l'utilisateur a demandé à la Fourmi de subir. Ainsi si l'utilisateur envoie comme règles : RLUD Alors :

Regles[0] = 1 // R

Regles[1] = 0 // L

Regles[2] = 2 // U

Regles[3] = 3 // D

Et la deuxième variable très importante pour comprendre le code est la suivante :

Monde[TAILLE_MONDE][TAILLE_MONDE][TAILLE_MONDE] : entier

qui est simplement le monde sous sa forme cubique, monde dans lequel la Fourmi va évoluer. la ligne un peu barbare :

`regles[monde[f>x][f>y][f>z]] <- regles[monde[f>x][f>y][f>z]]+1 ;` s'explique donc très facilement :

si la règle de la case en (x,y,z) était la règle numéro i, cette case contient après déplacement de la fourmi, la règle numéro i+1. On y applique aussi un modulo(nbRegles) pour faire en sorte que les règles forment un cycle et que la dernière +1 arrive bien sur la première règle.

2.2.4 Rotation à 90 degrés vers la Gauche

Nous allons maintenant vous commenter le code d'une des Rotations, afin de vous montrer comment est réalisée la rotation à 90 degrés.

La structure est la même pour toutes les directions et toutes les règles :

Que ce soit pour la règle U,D, L ou R la structure de test est la même :

On teste la direction courante de la Fourmi.

Une fois que l'on a détecté sa direction, on va tester dans quel repère on se trouve,

Puis en fonction du repère, nous allons mettre à jour la direction de la Fourmi.

La seule différence se trouve dans les règles U et D, où en plus de mettre à jour la direction, le repère dans lequel on se trouve est lui aussi, mis à jour. (voir partie précédente).

Algorithm 2 Gauche : Rotation à 90 degrés

Require: f une Fourmi

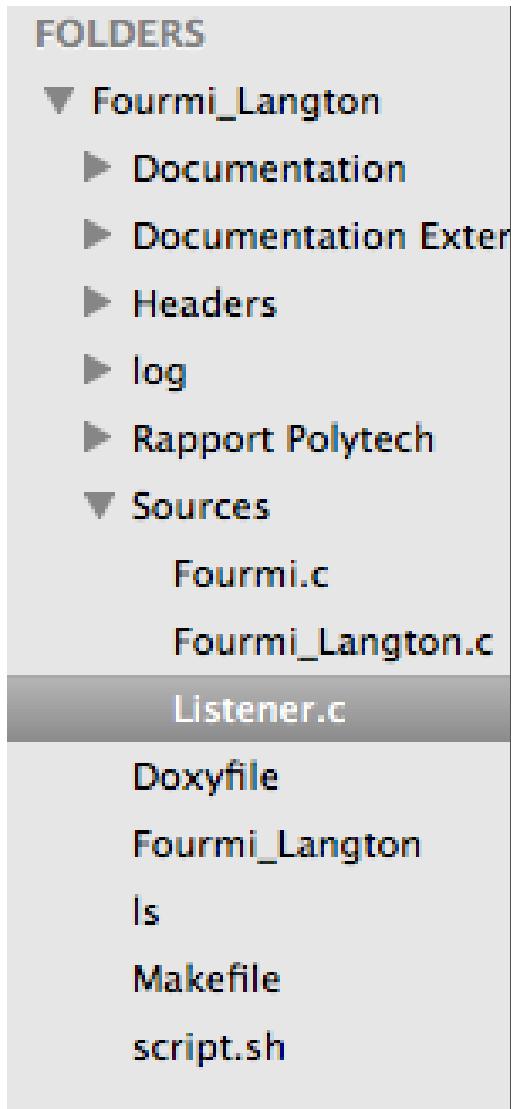
Ensure: \emptyset , La Fourmi a réalisé une rotation de 90 degrés vers la Gauche

```
if ( $f->direction = DEVANT$ ) then
2:    if ( $plan = 0$ ) then
        f->direction= GAUCHE ;
4:    else if ( $plan = 1$ ) then
        f->direction= DROITE ;
6:    else if ( $plan = 2$ ) then
        f->direction= GAUCHE ;
8:    else if ( $plan = 3$ ) then
        f->direction= GAUCHE ;
10:   else if ( $plan = 4$ ) then
        f->direction= HAUT ;
12:   else if ( $plan = 5$ ) then
        f->direction= BAS ;
14:   end if
else if ( $f->direction = GAUCHE$ ) then
16:    if ( $plan = 0$ ) then
        f->direction= DERRIERE ;
18:    else if ( $plan = 1$ ) then
        f->direction= DEVANT ;
20:    else if ( $plan = 2$ ) then
        f->direction= BAS ;
22:    else if ( $plan = 3$ ) then
        f->direction= HAUT ;
24:    else if ( $plan = 4$ ) then
        f->direction= GAUCHE ;
26:    else if ( $plan = 5$ ) then
        f->direction= GAUCHE ;
28:    end if
else if ( $f->direction = HAUT$ ) then
30:    if ( $plan = 0$ ) then
        f->direction= GAUCHE ;
32:    else if ( $plan = 1$ ) then
        f->direction= GAUCHE ;
34:    else if ( $plan = 2$ ) then
        f->direction= GAUCHE ;
36:    else if ( $plan = 3$ ) then
        f->direction= DROITE ;
38:    else if ( $plan = 4$ ) then
        f->direction= DERRIERE ;
40:    else if ( $plan = 5$ ) then
        f->direction= DEVANT ;
42:    end if
else if ( $f->direction = DERRIERE$ ) then
44:    ...
else if ( $f->direction = DROITE$ ) then
46:    ...
```

2.3 Comment sont structurés les fichiers

Nous avons choisi de structurer les fichiers de la manière la plus simple possible :

Tout ce qui sert à la gestion de la Caméra et du Clavier se situe dans le fichier Listener.c



Toujours aussi simplement, ce qui sert à l'affichage du monde et à la gestion de la Fourmi se situe dans le fichier Fourmi.c

Ce qui sert à l'initialisation, des branchements que nécessite Glut/OpenGL ainsi que le Main du programme se situe dans Fourmi_Langton.c

Pour ce qui est des variables comme le Monde, la taille du monde, le tableau de règles, le nombre de règles, le nombre de Fourmis que l'on voudrait exécuter dans le programme, tout se situe dans le fichier Constantes.h

```
25 #define TAILLE_MONDE 200
26 unsigned char monde[TAILLE_MONDE][TAILLE_MONDE][TAILLE_MONDE];
27
28 Couleur* couleurs;
29 char* regles;
30
31 /*
32 * Les Propriétés de la Fenêtre Graphique
33 */
34
35 #define WINDOW_WIDTH    1000 /* la longueur */
36 #define WINDOW_HEIGHT   1000 /* la hauteur */
37
38 #define WINDOW_X        130  /* centrée à ( 130 , 0 ) */
39 #define WINDOW_Y        0    /* centrée à ( 130 , 0 ) */
40
41 #define WINDOW_TITLE    "Fourmi de Langton Par Alizee Buatois & Valentin Montmirail"
42
43 /*
44 * Fin des Propriétés de la Fenêtre Graphique
45 */
46 #define GAUCHE    0
47 #define DEVANT    1 /* les directions de la Fourmi. */
48 #define DROITE    2
49 #define DERRIERE  3
50 #define HAUT      4
51 #define BAS       5
52
53 int plan=0;
54
55 FILE* fichierLog;
56 char* nomFichierLog;
57
58 int vitesse = 1; /* détermine un coefficient de vitesse.
59 * par défaut, on fait 1 mouvement puis on l'affiche. */
60
```

Et enfin pour ce qui est des structures que l'on a défini pour ce projet, elles se trouvent toutes dans le fichier : Structures.h

```
1 #ifndef STRUCTURES_H_INCLUDED
2 #define STRUCTURES_H_INCLUDED
3
4
5 /**
6     Une couleur est représenté par 3 octets :
7         1 qui encode le Rouge.
8         1 qui encode le Vert.
9         1 qui encode le Bleu.
10 */
11 typedef struct str_Couleur {
12
13     char R;
14     char G;
15     char B;
16
17 } Couleur;
18
19
20 /**
21     Une Fourmi est simplement une Case spéciale
22     qui n'a pas de règle, mais qui a une direction.
23 */
24 typedef struct Fourmi {
25
26     int x,y,z;
27     Couleur* couleur;
28     int direction;
29
30 } Fourmi;
31
32 #endif
33
```

2.4 Gestion de la Caméra et du Clavier

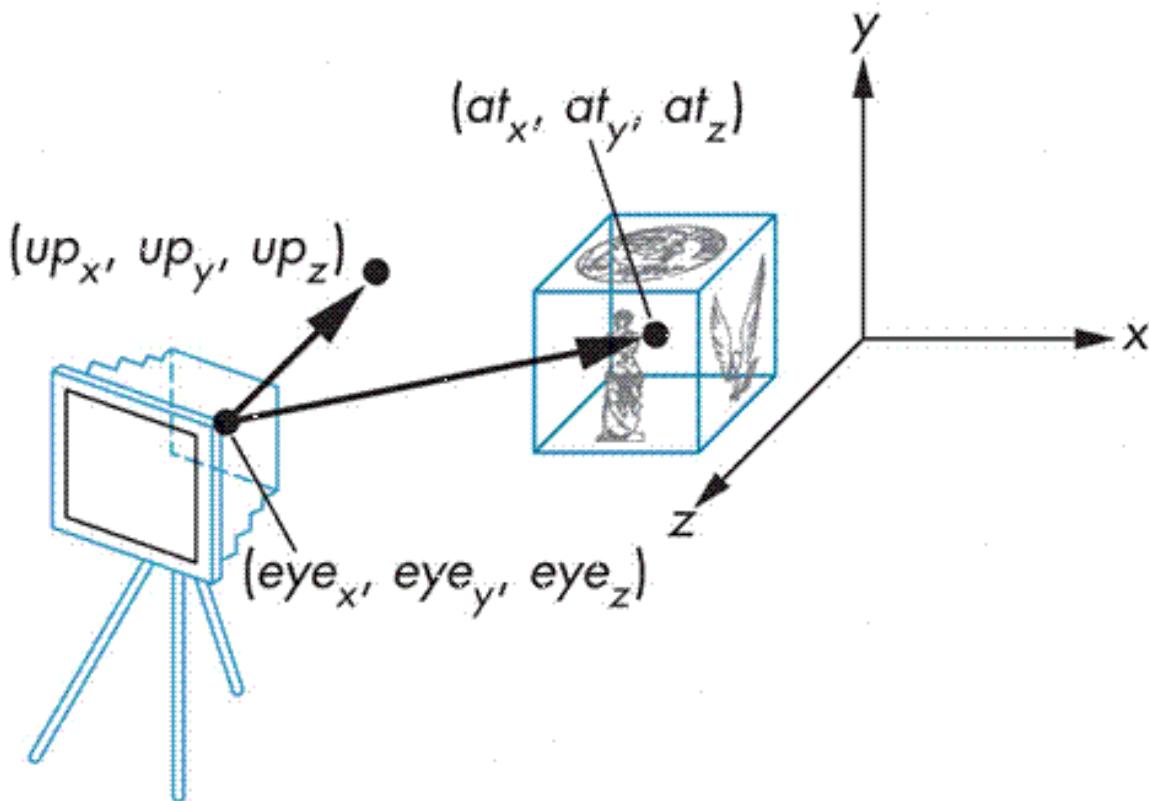
2.4.1 La Caméra

La gestion de la Caméra se fait en OpenGL/Glut grâce à la fonction : glutLookAt

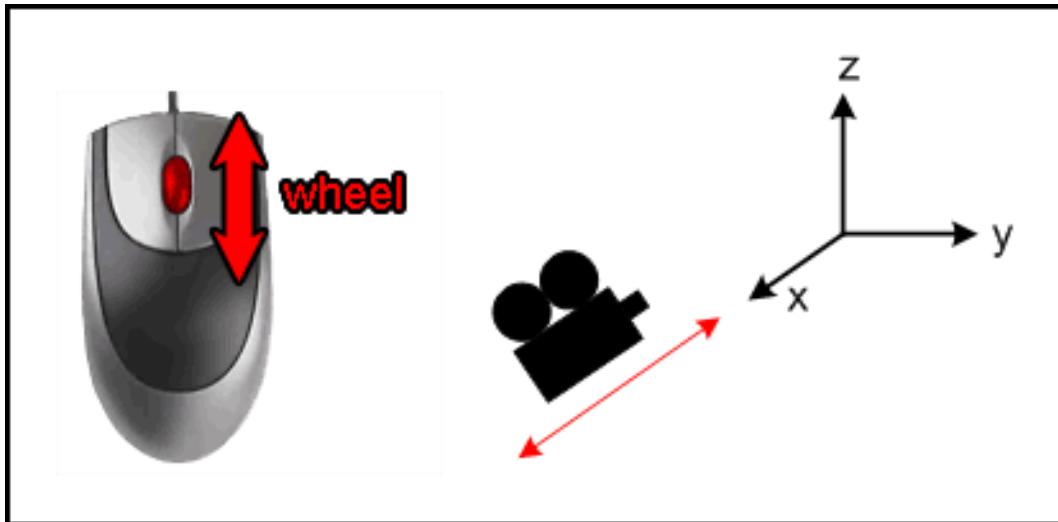
Cette fonction s'appelle ainsi :

```
gluLookAt(eye.x, eye.y, eye.z, at.x, at.y, at.z, up.x, up.y, up.z);
```

Mais voici un schéma qui illustrera bien plus facilement cette gestion que ces noms de variables.



Pour prendre du recul ou au contraire nous rapprocher de l'objet/scène que nous souhaitons visualiser, nous allons tout simplement utiliser la molette. Un coup de molette en avant pour zoomer, un coup de molette en arrière pour dézoomer, rien de plus intuitif :

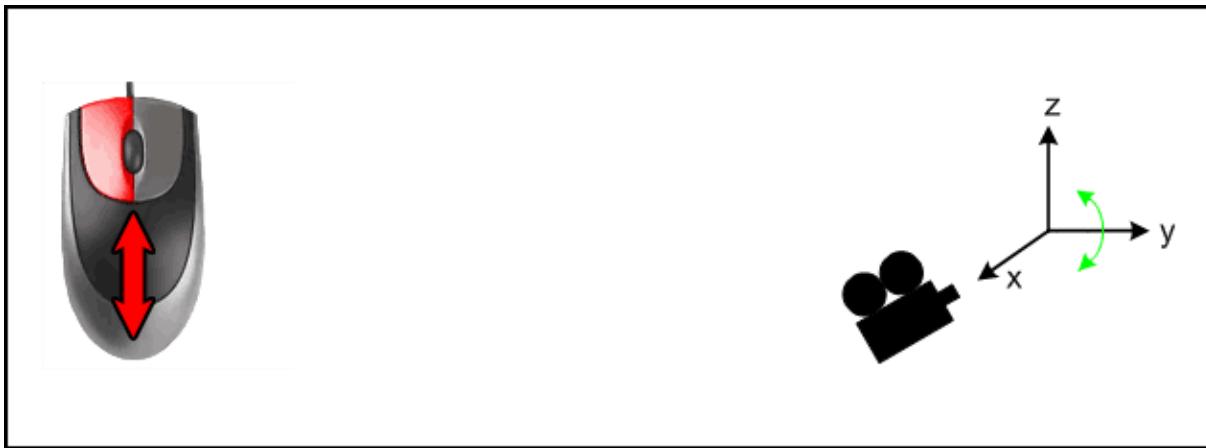


En maintenant le bouton gauche de la souris enfoncé, les mouvements de la souris feront tourner la scène :

- un mouvement horizontal de la souris donne une rotation horizontale de la scène
- un mouvement vertical de la souris donne une rotation verticale de la scène.

Ces mouvements sont illustrés par les schémas ci-dessous :





Il faut aussi faire les branchements des "Ecoutes" afin de savoir quand la souris bouge, et où est sa nouvelle position. Ou encore pour savoir quel bouton de la souris a été activé. Pour ce faire, Glut nous offre 2 fonctions :

glutMotionFunc (DeplacementSouris) ;

glutMouseFunc (MouseListener) ;

Il ne reste donc qu'à mettre le code dans

DeplacementSouris, qui va gérer par exemple que l'on est entrain de déplacer horizontalement la souris.

MouseListener, qui va gérer par exemple, que la molette de la souris est activée vers le Haut ou vers le Bas.

2.4.2 Le Clavier

En plus de la Caméra qui a été gérée, une écoute est réalisée sur le Clavier afin de traiter le cas où une touche du clavier a été frappée.

Comme pour la Caméra, Glut nous fournit des fonctions qui prennent en paramètre des pointeurs de fonctions afin de pouvoir redéfinir un traitement en cas de touche de clavier frappée.

Cette fonction s'appelle : glutKeyboardFunc(KeyBoardListener) ;

Et comme l'un des paramètres de cette fonction va forcément être :

un octet non signé que l'on peut appeler touche

il ne reste qu'à faire des structures Si conditionnelles du type :

Si touche = 'q' alors

QuitterLeProgramme

FinSi

2.5 Gestion des Fichiers de Logs

L'ensemble des déplacements réalisés par la Fourmi de Langton est stocké dans un log/fichier.log

le fichier portera le nom des règles qui ont été utilisées pour déplacer l'automate cellulaire. Après une exploitation, il permet de déterminer à partir de quelle itération la Fourmi de Langton exécute sa célèbre "Autoroute".

Il serait tout à fait possible d'exploiter ces fichiers de logs afin de déterminer quel jeu de règles est le plus ressemblant d'un autre jeu de règles, afin de les classer.

Voici par exemple les 20 premières itérations de la Fourmi de Langton classique, celles dont les règles sont RL

Droite, Droite, Droite, Gauche, Droite, Droite, Droite, Gauche, Droite, Droite, Droite, Droite, Droite, Droite, Gauche, Gauche, Gauche, Droite, Gauche,

Ces fichiers de logs nous ont été très utiles dans nos phases de déboggage puisqu'il était possible de faire un pas à pas et voir à partir de où, l'automate ne faisait pas le mouvement qui était attendu.

Documentation	►	LDLDR.log
Documentation Externe	►	LL.log
Doxyfile	►	LLL.log
Fourmi_Langton	►	LLLL.log
Headers	►	RL.log
log	►	RLLL.log
ls		RLLLLDU.log
Makefile		RLUD.log
Rapport Polytech	►	
script.sh		
Sources	►	

2.6 Comment utiliser le programme

La méthode la plus simple est tout d'abord de taper :

./Fourmi_Langton -help

qui vous affichera l'aide en mode console. Puis ensuite 2 modes s'offrent à vous :

Soit le mode aléatoire, qui va créer une séquence de règles puis va exécuter la Fourmi de Langton

Pour lancer ce mode aléatoire, il faut exécuter comme suit :

./Fourmi_Langton -alea N

avec N la taille de la chaîne des règles que vous voulez.

Soit le mode déterminé, où vous indiquez la séquence de règles que vous voulez pour votre Fourmi de Langton

Pour lancer ce mode déterminé, il faut exécuter comme suit :

./Fourmi_Langton -deter {L,R,U,D}*
par exemple pour exécuter les règles RLUD. Il faut exécuter :

./Fourmi_Langton -deter RLUD

```
MacBook-Pro-De-Valente:Fourmi_Langton mystelven$ ./Fourmi_Langton -help
Réalisation de l'automate Cellulaire : La Fourmi de Langton
Par Alizée Buatois et Valentin Montmirail, 2 élèves en 3e année à Polytech Tours.

Sujet :
A l'origine, il s'agit d'un automate cellulaire bidimensionnel, comportant un jeu de règles très simples :

- Si la fourmi est sur une case noire, elle tourne de 90° vers la droite,
change la couleur de la case en blanc et avance d'une case.

- Si la fourmi est sur une case blanche, elle tourne de 90° vers la gauche,
change la couleur de la case en noir et avance d'une case.

Aussi bizarre que cela puisse paraître, après un certain temps de comportements chaotiques,
un comportement en "autoroute" se met à émerger.
Ce projet vise à implémenter cet automate et étudier la complexification de ce système (passage en 3D, etc.)
Ce projet peut être utilisé avec les arguments suivants :

Avec -alea n , où n , entier, varie entre 2 et 255.
Avec -deter n , avec n symboles parmi l'alphabet {R,L,U,D}

Vous pouvez taper -help pour afficher l'aide, mais pour voir ceci, vous le savez déjà.
Par défaut, le programme sera exécuté avec les règles RLUD

MacBook-Pro-De-Valente:Fourmi_Langton mystelven$
```

Une fois le programme lancé, certaines touches du clavier permettent d'exécuter des actions spécifiques.

la touche R permet de regénérer aléatoirement les couleurs au niveau de l'affichage sans détruire le parcours qu'a déjà réalisé la Fourmi.

la touche c permet de nettoyer le monde sans replacer la Fourmi à l'origine du repère.

la touche + permet d'accélérer la Fourmi dans ses déplacements.

la touche - permet de ralentir la Fourmi dans ses déplacements.

la touche v permet de remettre la Fourmi à sa vitesse initiale.

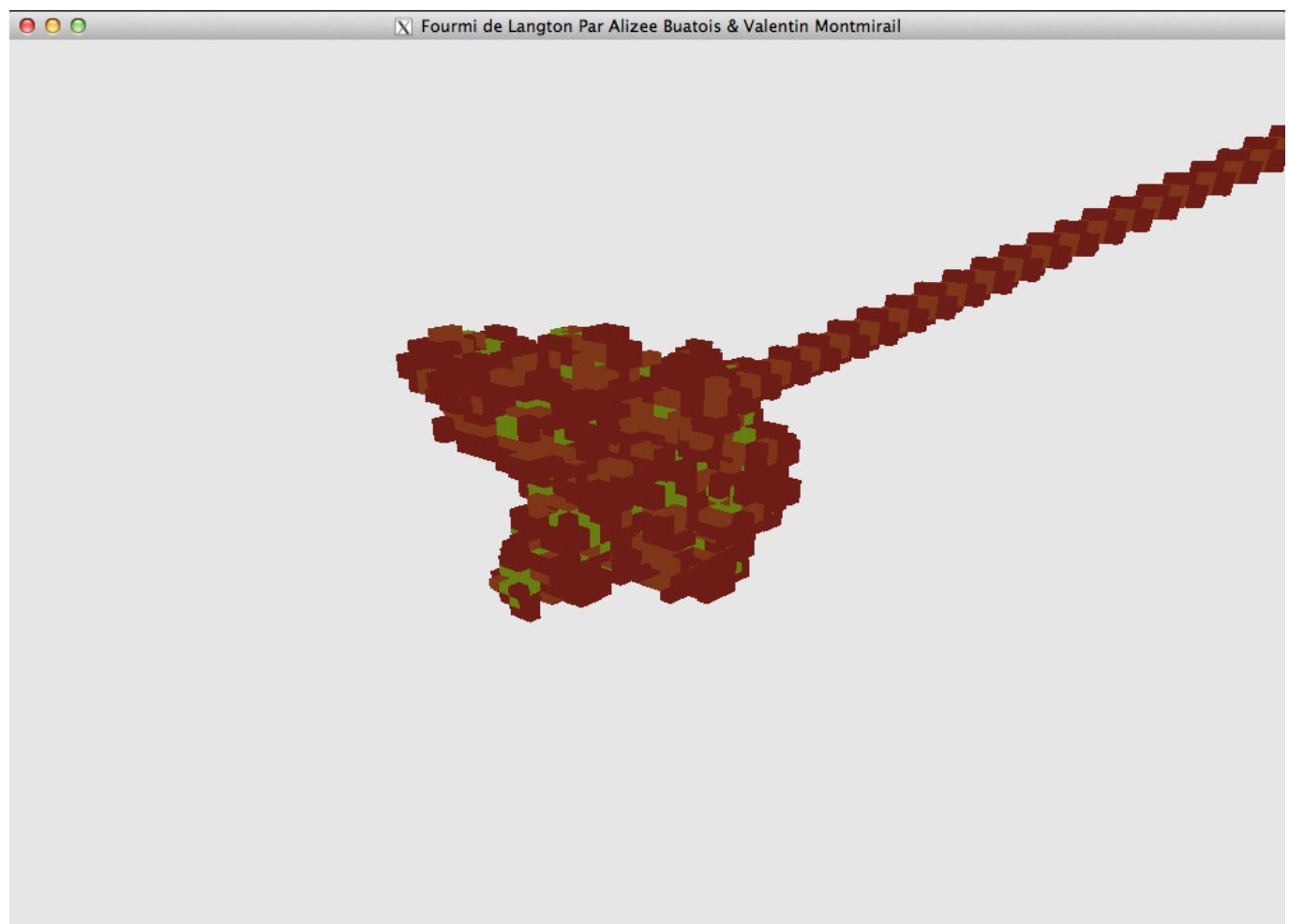
la touche p permet d'exécuter la Fourmi de Langton en pas à pas.

la touche ESPACE permet de mettre en pause/relancer l'automate cellulaire.

la touche q permet de quitter proprement le programme.

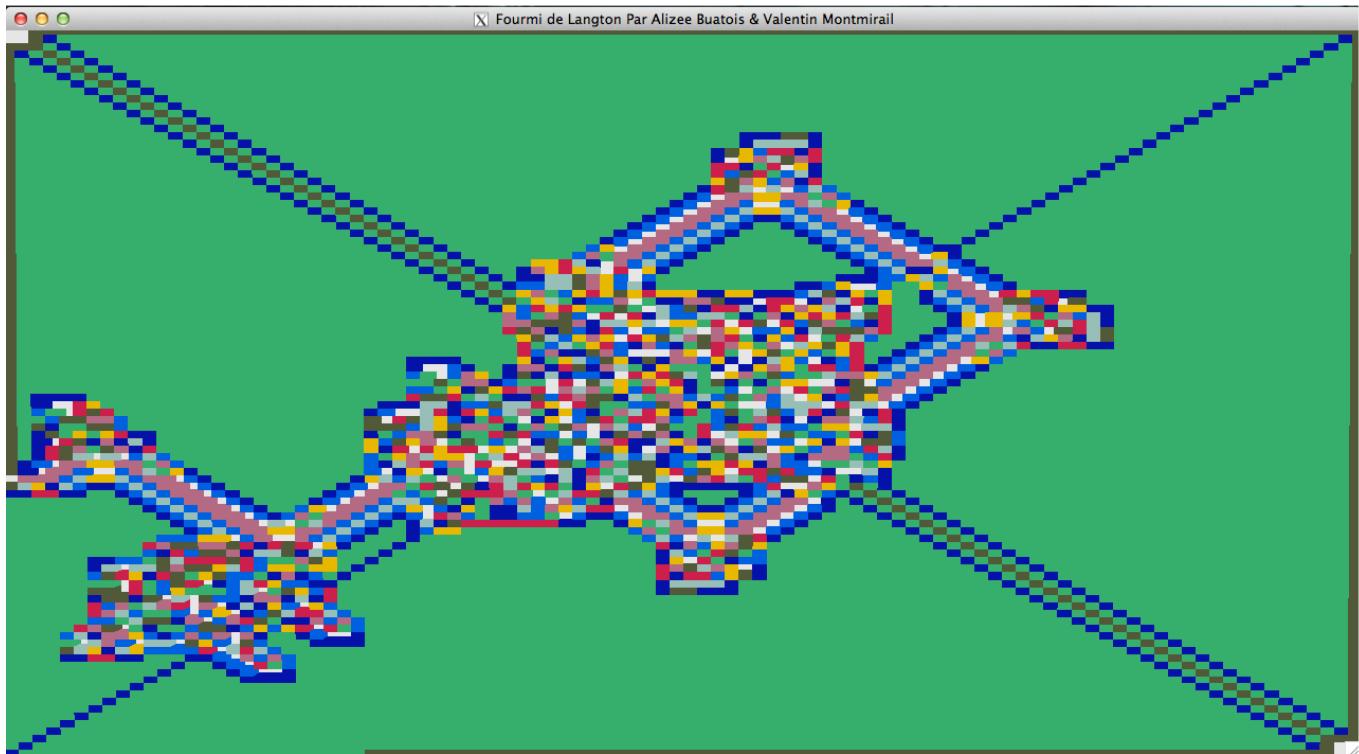
2.7 Exemples de résultats visuels intéressants

2.7.1 LRDU



On voit sur cette automate, le phénomène d'autoroute se produire en 3D !

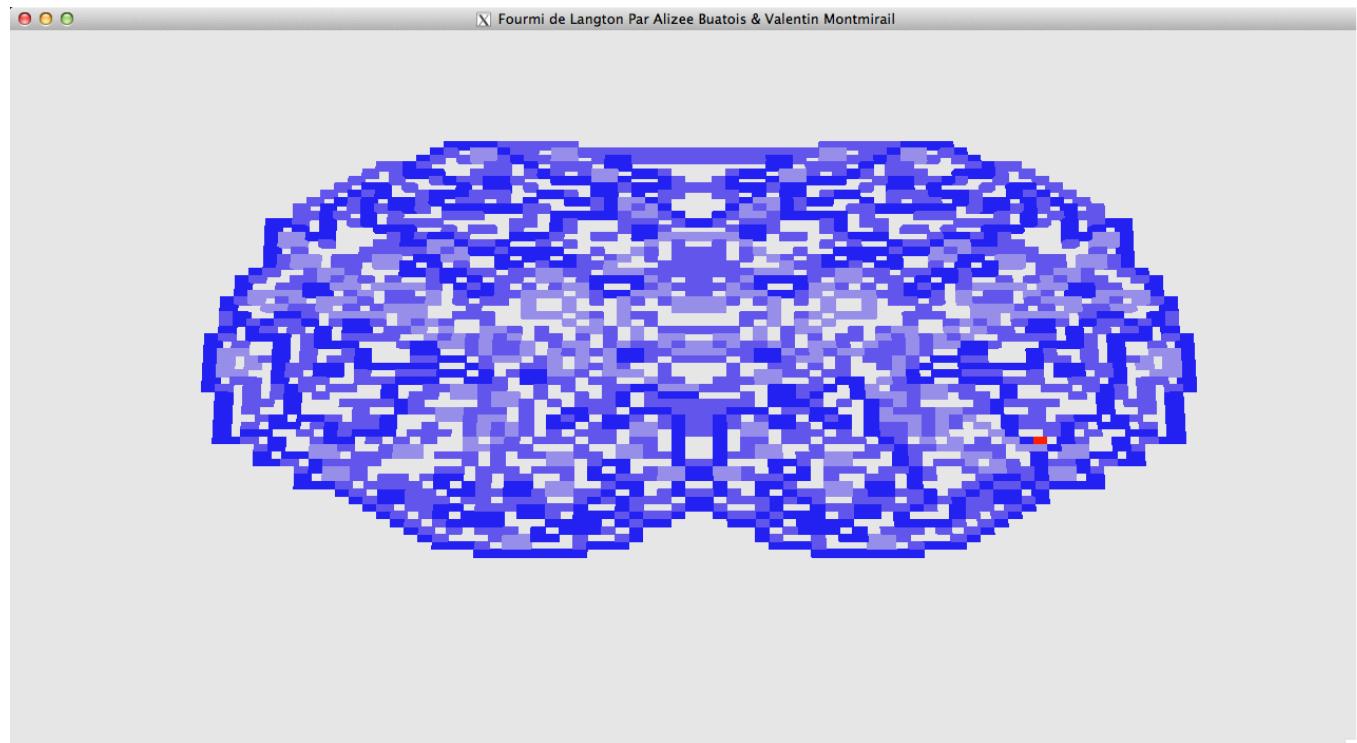
2.7.2 LRRRRRLLR



Ce résultat est lui aussi très intéressant, on voit 2 phénomènes sur cette image

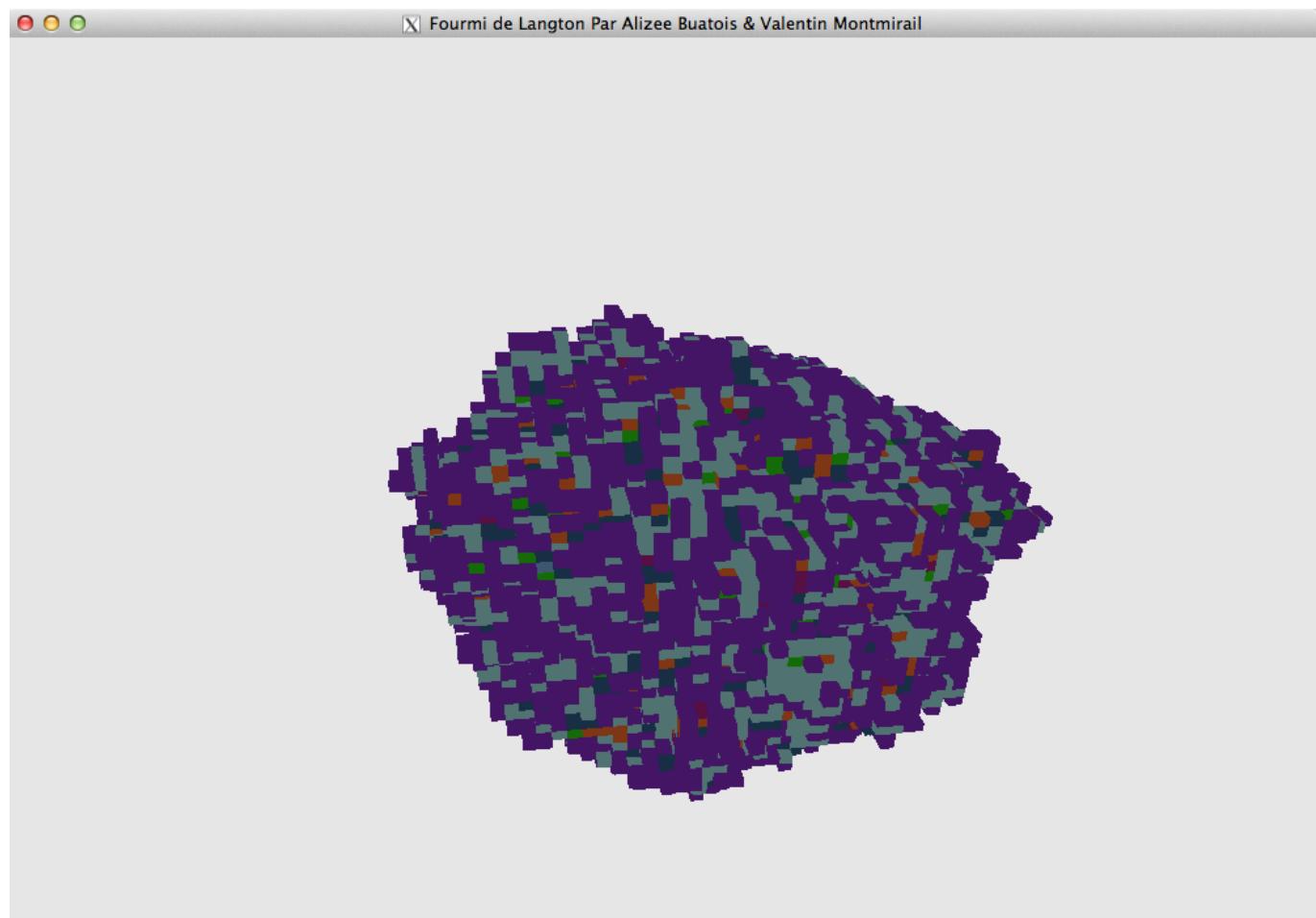
Le premier : la Fourmi dessine un carré tout autour d'elle , qu'elle propage à l'infini (ici en vert) Puis à l'intérieur de ce carré, elle va réaliser son phénomène d'autoroute qui va taper dans les 4 coins et au fur et à mesure le carré autour (en vert) s'agrandit.

2.7.3 RRLL



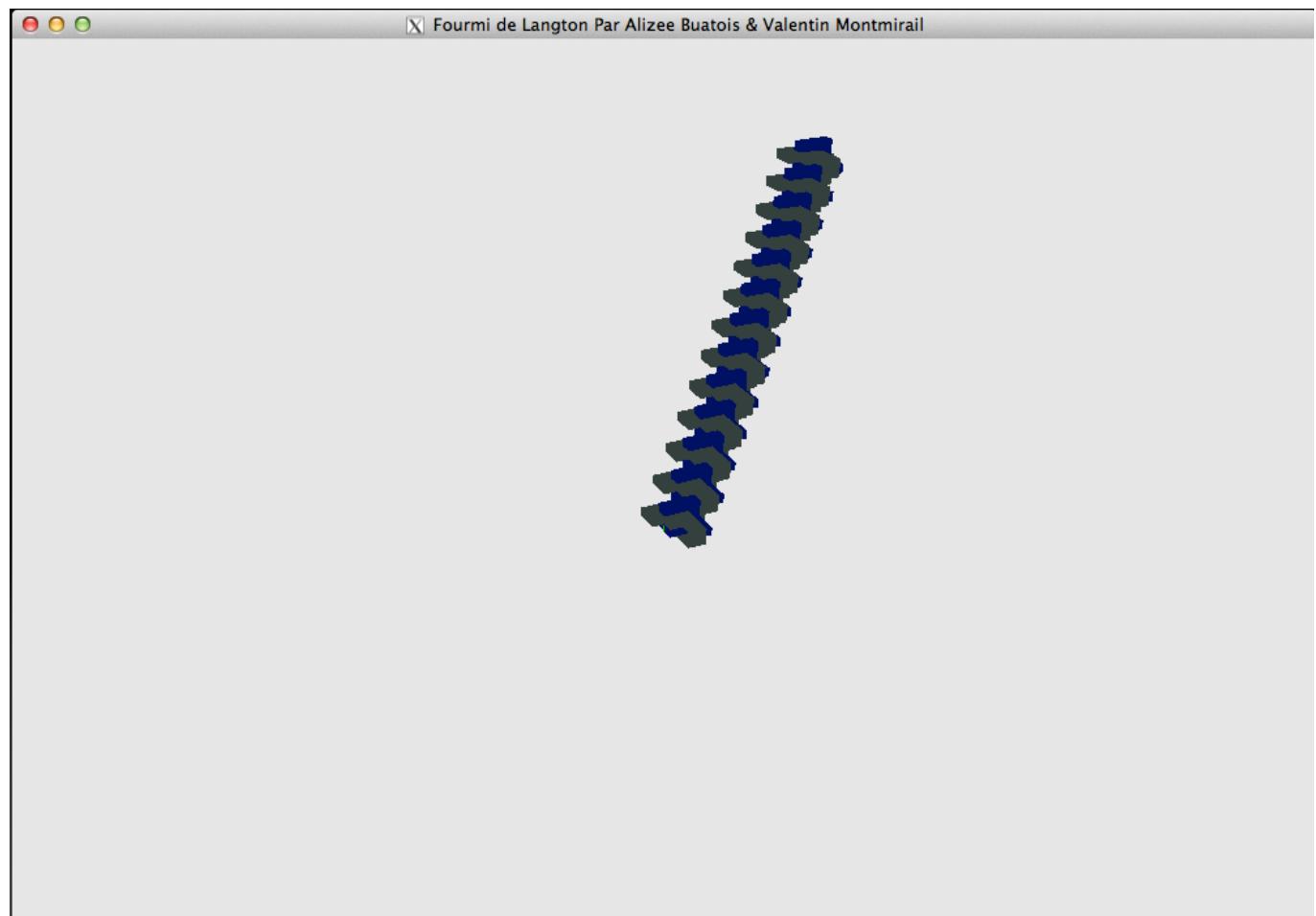
Ici, sur cette image, on ne voit pas encore le phénomène d'autoroute car nous n'avons pas laissé tourner l'automate cellulaire assez longtemps, en effet quand les règles sont symétriques, le résultat visuel l'est aussi, et il est beaucoup plus long d'obtenir un phénomène émergent comme l'Autoroute de Langton dans une structure stable comme celle-ci.

2.7.4 RRLLUUDD



Tout comme RRLL, la Fourmi en RRLLUUDD forme aussi une structure extrêmement stable et symétrique, ce qui fait que nous devons attendre énormément d'itérations avant de voir apparaître la fameuse autoroute...

2.7.5 RLD



La Fourmi de Langton avec les règles RLD est extrêmement intéressantes car elle permet de visualiser directement le phénomène d'autoroute sans aucune phase de déplacement chaotique.

CHAPITRE 3

Améliorations Possibles

3.1 Ajouter des classes d'équivalence entre les Règles

Afin de mieux comprendre pourquoi ce phénomène d'autoroute apparaît, et pourquoi dans certains cas l'autoroute arrive rapidement et dans d'autres au bout de beaucoup plus de temps, nous avons conclu qu'il fallait essayer de classer les séquences de règles choisies.

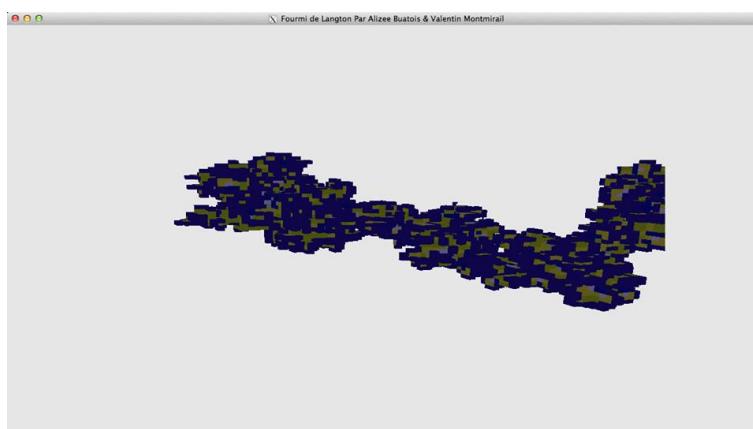
En effet, nous avons remarqué que les fourmis ayant créé une autoroute étaient classables selon la longueur de règles qui lui sont associées. Il s'avère qu'après plusieurs recherches, il existe plusieurs classes d'équivalences de taille de période de l'autoroute, non forcément définies par le nombre de règles. Cette classification en classes d'équivalence prend en compte le temps et l'espace nécessaire avant le début de l'autoroute dans les mouvements de la fourmi, et fait aussi intervenir des classes de ressources. Etant assez complexe, et n'aboutissant pas à de grandes conclusions, nous avons choisi de ne parler que des classes d'équivalence.

Voici ce que nous avons trouvé :

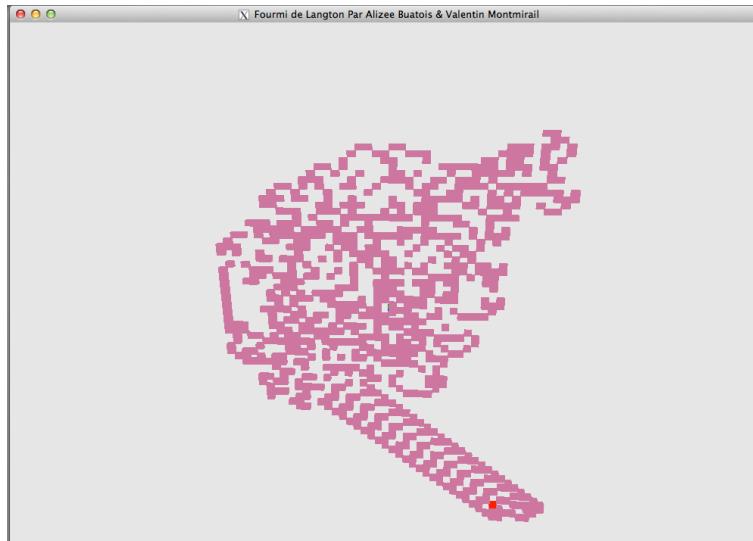
Classe n : contient toutes les fourmis formant des autoroutes ayant moins de quatre à la puissance n plus un étapes par période de l'autoroute, et ne pouvant pas être incluses dans une autre classe.

Dans notre cas, voici 2 exemples :

-screenshot avec un plus simple



Dans ce premier exemple avec les règles RLRUUUL, on remarque une autoroute complexe, car selon la classification, tout mouvement périodique "infini" est considéré comme une autoroute. Sa période est donc longue, et cette fourmi appartient donc à une classe d'équivalence grande (classe 7 avec une période de 25436 étapes).



Dans ce deuxième exemple, déjà vu, l'autoroute a une période beaucoup plus courte. (classe 2).

L'hypothèse émise par certains chercheurs serait que la fréquence de changement de plans diminuerait avec un nombre important d'étapes différentes et non symétriques.

En comparaison, les fourmis en 2 dimensions étant dans la classe 1 (RRL, RRRL) ou dans la classe 2 (RL) avec une courte combinaison de règles sont facilement trouvables.

3.2 Rajouter de la génétique pour influencer le résultat

Il serait aussi possible par exemple, pour améliorer ce projet, de rajouter une partie "génétique" à cet automate.

Imaginons par exemple que le résultat visuel avec les règles RLUD nous plaise mais que ce résultat commence à moins nous plaire passé 10 000 itérations. Alors il faudrait trouver un système,

soit en pré-générant un monde qui ne serait pas "tout blanc"

soit en trouvant un moyen de complexifier les règles de déplacement

Et faire en sorte qu'avec ce système, on puisse générer un résultat visuel qui plaise plus visuellement que le résultat initial.

Le point pour démarrer sur cette idée serait selon nous :

que des règles qui s'annulent forme des résultats visuels symétriques des règles du type RRLL , UUDD , .. RRLLUUDD ... vont forcément donner un résultat symétrique, si la personne qui utilise le programme n'aime pas ce genre de résultat, il est donc inutile de générer ce type de règles.

Il doit peut être exister un moyen de trouver des "formes" de résultats en fonction des règles, et qu'en croisant 2 formes différentes de résultats, on puisse faire un résultat visuel hybride entre les 2.

3.3 Donner le choix dans les couleurs

Dans notre programme, nous avons choisi de générer des couleurs aléatoirement, pour plus de simplicité, la couleur de la fourmi n'ayant pas d'importance particulière.

Cependant, attribuer une couleur différente à chaque règle était essentiel afin d'avoir un rendu visuel assez bon pour distinguer les déplacements de la fourmi.

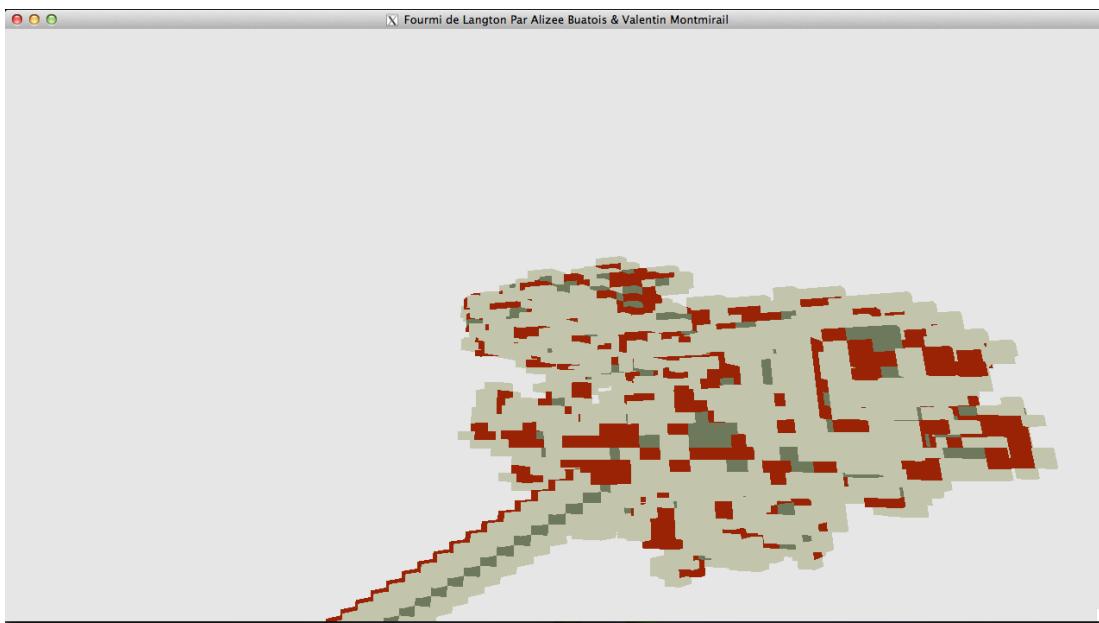
Mais la principale faiblesse de notre choix est donc l'apparition de combinaisons de couleurs non assez significatives. En effet, il nous arrive assez souvent de changer de couleurs parce que le rendu visuel n'est pas assez bon (trop de couleurs foncées côte à côte, trop de couleurs claires, tec.).

L'idée pour éviter cela, serait la possibilité de faire une interface utilisateur pour choisir les couleurs.

Cette interface serait faite en fonction du nombre de règles, s'adaptant aux choix de l'utilisateur. Par exemple, si certaines couleurs et/ou combinaisons ne lui plaisent pas, il pourra l'indiquer au programme, et ces dernières ne seront donc plus générées.

Au bout de plusieurs choix de couleurs de l'utilisateur, pour un même nombre de règles choisies, le programme ne devrait donc générer plus que des combinaisons "ressemblant" à ce que l'utilisateur aura choisi.

Par exemple, nous souhaiterions éviter ce genre de combinaisons qui ne rendent pas très visibles les déplacements :



Conclusion

Ce projet nous a permis de voir un exemple de comportement émergent.

En effet, nous avons défini des règles extrêmement simples qui est de tourner :

soit 90 degrés vers la Gauche

soit 90 degrés vers la Droite

soit 90 degrés vers le Haut

soit 90 degrés vers le Bas

et pourtant, la Fourmi se met à aller tout droit avec son phénomène d'autoroute, comme si elle était attirée par quelque chose vers l'infini.

l'écart entre la simplicité de sa définition et la complexité que peuvent atteindre certains résultats visuels est vraiment remarquable.

Le fait que des automates cellulaires puissent simuler n'importe quelle machine de Turing amène aussi des problèmes indécidables d'une autre nature.

Par exemple, pour la Fourmi de Langton, les problèmes suivants sont indécidables :

* savoir si, étant donné un motif m présent initialement au centre, un motif va apparaître au centre au bout d'un certain temps. (l'autoroute)

* savoir si un motif m peut apparaître arbitrairement tard dans l'évolution ou, au contraire, s'il existe un temps au-delà duquel m n'apparaît plus quelle que soit la configuration de départ.

D'une manière plus concrète, ce projet nous a aussi fait manipuler OpenGL et Glut ainsi que le langage C de manière intensif.

On a ainsi appris à rendre un projet résistant aux bugs à travers des méthodes de développement afin de pouvoir rendre un livrable qui a un certain gage de qualité.

ANNEXE A

Bibliographie

Voici une petite liste d'URL intéressantes au sujet de ce projet :

- http://fr.wikipedia.org/wiki/Fourmi_de_Langton
- www.complex-systems.com/pdf/14-3-4.pdf
- <http://scienctonnante.wordpress.com/2011/03/21/la-fourmi-de-langton/>
- http://en.wikipedia.org/wiki/Langton%27s_ant
- <http://en.wikipedia.org/wiki/Emergence>
- http://en.wikipedia.org/wiki/Turing_machine
- <http://code.google.com/p/fourmi/>

Fourmi de Langton

Rapport Final

Résumé : On nomme fourmi de Langton un automate cellulaire bidimensionnel comportant un jeu de règles très simples. Elle constitue l'un des systèmes les plus simples permettant de mettre en évidence un exemple de comportement émergent. Nous avons donc passé cet automate cellulaire en 3D afin de voir si le phénomène de comportement émergent se produit aussi.

Mots clé : Langton, Fourmi, Automate Cellulaire, 3D, OpenGL, Comportement Emergent

Abstract : Langton's ant is a two-dimensional Turing machine with a very simple set of rules but complicated emergent behavior. We have upgrade this cellular automaton in 3D to see if the emergent order happens or not in 3D.

Keywords : Langton, Ant, Cellular automaton, 3D, OpenGL, Emergent Order

Auteur(s)

Alizée Buatois

[alizee.buatois@etu.univ-tours.fr]

Valentin Montmirail

[valentin.montmirail@gmail.com]

Encadrant(s)

Nicolas Monmarché

[nicolas.monmarche@univ-tours.fr]

Ce document a été formaté selon le format EPUStagPeiP.cls (N. Monmarché)

École Polytechnique de l'Université de Tours
64 Avenue Jean Portalis, 37200 Tours, France
<http://www.polytech.univ-tours.fr>