UNIVERSITÉ
Clermont
Auvergne

Projet Simulation

Implémentation d'un système multi agents de prédateurs et de proies

Figure1 : Source personnelle

Réalisé par :

- Arnaud Rouchon
- Imene Ouahrani

Encadrant:

- David Hill

2017/2018

Remerciement

Nous tenons à exprimer nos sincères remerciements à tous les professeurs qui nous ont enseigné les modules « Simulation» ainsi que «Programmation et système » et qui par leurs compétences nous ont soutenu dans la poursuite de nos études. Nous voudrions également leur témoigner notre gratitude pour leur patience et leur soutien qui nous ont été précieux afin de mener notre travail à bon port.

Table des matières

Table des figures

❖ Introduction

Afin de réaliser notre projet en « Simulation », nous avons mis en œuvre un Système Multi Agents « SMA ». Ce projet a été implémenté en C.

Nous avons choisis de simuler l'environnement des animaux sauvages et les mécanismes naturels de régulation, les animaux prédateurs font disparaitre leurs proies, puis disparaitre eux-mêmes.

1. Conception du système multi-agents

1.1 Le milieu

Les agents évoluent dans un milieu, il peut être une campagne, une savane ou n'importe quel espace suffisamment vaste pour ne pas imposer la rencontre immédiate des prédateurs et des proies dans notre cas. On le représentera par un tableau où chaque case correspond à une aire où la rencontre éventuelle entre un prédateur et une proie serait inévitable et ne laisserait aucune chance de survie à la proie.

1.2 Les agents

Chaque agent a accès aux huit cases qui entourent la sienne (ainsi que la sienne) de manière équiprobable (sauf si un prédateur rencontre une proie, auquel cas il tentera sûrement une attaque). Nos agents étant des animaux, ils peuvent se reproduire mais aussi mourir de vieillesse. On ne peut pas avoir deux animaux sur une même case, si un prédateur rencontre une proie, il prend sa place pour la manger (comme le fait une pièce qui en prend une autre aux échecs).

a) Les prédateurs

Les prédateurs, comme dans la nature, se nourrissent de proies qu'ils parviendront à attraper. S'ils croisent une proie sur une case adjacente à la leur, ils la mangeront si leur attaque réussit, dans le cas contraire la proie s'enfuit. Contrairement aux proies, leur nourriture peut devenir un problème.

b) Les proies

Les proies sont généralement herbivores, elles trouvent donc facilement leur nourriture mais peuvent être chassées (et mangées) par les prédateurs. Leur durée de vie maximale est plus longue que celle des prédateurs.

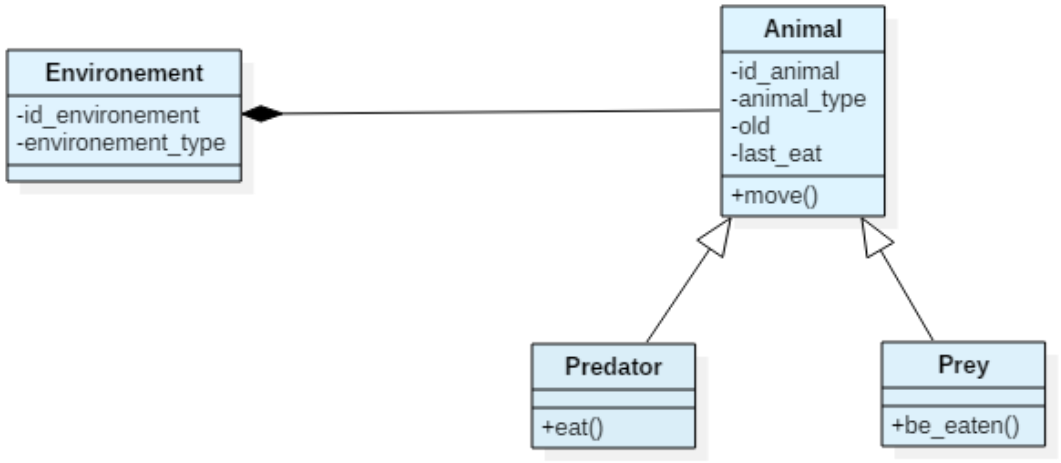

Figure 2.1 : « Diagramme de classes du SMA ».*Source personnelle*

2. Implémentation du SMA

2.1 Distinction des différents agents

a) Les types d'agents

Les agents sont stockés sous la forme d'une structure en langage C, on stocke donc le fait qu'ils soient des prédateurs ou des proies. L'autre type est « NONE » signifiant qu'il n'y a aucun agent dans la case concernée, en effet toutes les « cases » d'un tableau en C doivent contenir le même type d'entités, c'est à dire des animaux.

b) Les structures utilisées

L'espace est un tableau qui contient des structures appelées « animaux » qui sont soit des prédateurs soit des proies soit des prédateurs (soit une case vide, c'est à dire un animal ayant le type « none »). Leur type est un élément de l'énumération « type », ce qui permet d'être sûr de ne pas avoir de surprises sur des nouveaux types apparaissant par erreur. Un animal mort disparaît, sa case contiendra alors un animal de type « none ». Leur âge est également stocké en vue de programmer leur mort (oups, l'obsolescence programmée est interdite…).

```c
int i,j;

enum type
{
        PREY,PREDATOR,NONE
};

struct Animal{
        enum type animal_type;
        int old;
        int last_eat;
};
typedef struct Animal Animal;
```

Figure 2.1.1 : « énumération du type d'agent et structure animal »

c) évolution des agents

Les animaux ont un âge qui entre en compte sur leur durée de vie, sur l'impact lors de la rencontre avec un prédateur.

L'accès aux emplacements se fait de manière aléatoire aussi bien pour les proies que pour les prédateurs, sauf si un prédateur se retrouve voisin d'une proie (il ira la manger dans 90 % des cas).

Deux animaux de même espèce peuvent se reproduire à condition de rencontrer un animal avec lequel ils sont compatibles (50 % des cas, en effet on considère que la moitié des animaux sont des mâles, l'autre moitié étant des femelles, ça va de soi...).

2.2 Les règles d'implémentation

- Predator eats the prey when crossing with a probability = 90%;
- Predator moves twice faster than prey (1 case at a time for a prey), if a prey crosses a predator, the predator takes its place, (direction = random unless with a prey/predator);
- If an animal crosses another from his spice, another one is created (reproduction) with a probability = 50%;
- Predator can only survive 50 time units;
- Prey can only survive 60 time units;
- Predator can eating a prey, the prey eats as it needs;
- If a predator crosses two or more preys, only the oldest one is eaten, other go away;
- If a prey crosses two or more predators, only the youngest one can eat it.

```c
//create the grid

Animal** init_grid(int n){//on crée une grille de taille n*n
        Animal** retour = (Animal**) malloc(n*sizeof(Animal*));
        //printf("a");
        for (i=0; i<n; i++){
                retour[i] = (Animal*)malloc(n*sizeof(Animal));
                for (j=0; j<n; j++){
                        retour[i][j].animal_type = NONE;
                }

        }
        return retour;
}
void print_grid(Animal** grid, int n){
        for (i=0; i< n; i++){
                for (j=0; j< n; j++){
                        if (grid[i][j].animal_type == NONE){
                                printf(".");
                        }
                        if (grid[i][j].animal_type == PREDATOR){
                                printf("P");
                        }
                        if (grid[i][j].animal_type == PREY){
                                printf("X");
                        }
                        if (j==n-1){
                                printf("\n");
                        }
                }
        }
}
```

Figure 2.2.1 : « Fonction pour la création et l'affichage de la grille »

```c
//use it

int count_living_neighbours(int x, int y, int** grid, int grid_size){
        int living_neighbours = 0;
        x= x+grid_size;
        y = y+grid_size;
        if (grid[(x-1)%grid_size][(y-1)%grid_size]==1){
                living_neighbours = living_neighbours+1;
        }
        if (grid[(x-1)%grid_size][y%grid_size]==1){
                living_neighbours = living_neighbours+1;
        }
        if (grid[(x-1)%grid_size][(y+1)%grid_size]==1){
                living_neighbours = living_neighbours+1;
        }
        if (grid[x%grid_size][(y-1)%grid_size]==1){
                living_neighbours = living_neighbours+1;
        }
        if (grid[x%grid_size][(y+1)%grid_size]==1){
                living_neighbours = living_neighbours+1;
        }
        if (grid[(x+1)%grid_size][(y-1)%grid_size]==1){
                living_neighbours = living_neighbours+1;
        }
        if (grid[(x+1)%grid_size][(y+1)%grid_size]==1){
                living_neighbours = living_neighbours+1;
        }
        if (grid[(x+1)%grid_size][y%grid_size]==1){
                living_neighbours = living_neighbours+1;
        }
        return living_neighbours;
}
```

Figure 2.2.2 : « Fonction qui permet de compter le nombre de voisins »

```c
int count_eatable_preys(Animal** grid, int x, int y){
        int a;
        int b;
        for (a=x-1; a<x+2; a++){
                for (b=(y-1);b<(y+2); b++){
                        if (grid[x][y].animal_type == PREDATOR){
                                if (grid[a][b].animal_type == PREY){
                                        return 1;
                                }
                        }
                }
        }
        return 0;
}
```

Figure 2.2.3 : « Fonction qui compte le nombre proies mangées »