Jeu de la Vie

Projet Algorithmique-Programmation
STI 3ème année

P. Clemente, C. Toinard

20 novembre 2014

1 Objectif

Le but de ce projet est de coder une version très améliorée du jeu de la vie (allez lire le principe du jeu de la vie sur internet). Cette version très différente de la version originale va permettre de faire vivre un écosystème marin dans lequel Bob, Alice et les autres vont essayer de construire chacun un pont pour atteindre l'autre bord de la carte.

2 Simulation de l'ecosystème

L'ecosystème est une grille $N \times N$ ou chaque case est occupée par un des animaux/végétaux suivants :
– animaux : orques, baleines, requins, bar, thon, corail ...
– végétaux : plancton
Ces animaux et végétaux peuvent interagir selon un certain nombre de règles précises. Leur survie et reproduction dépendent de leur capacité à se nourrir. Chaque tour représente un état de la simulation. Au début, $tour\_courant = 0$.

2.1 Classe d'animal

Soit X une classe d'animal :
– Variable *dernier_repas(X)* : c'est le numéro du tour où l'animal X a pris son dernier repas.

- Variable *satiete(X)* : c'est l'autonomie de l'animal que lui a procuré son dernier repas.
- Variable *derniere_reproduction(X)* : c'est le numéro du tour ou l'animal X s'est reproduit la dernière fois.
- Constante *duree_survie(X)* : c'est le nombre de tours pendant lequel l'animal survit sans manger.
- Constante *taille(X)* : c'est la quantité de nourriture apportée à un prédateur.
- Constante *taille_du_bide(X)* : c'est la quantité maximale que l'animal peut stocker.
- Constante *saut_max(X)* : c'est le nombre case maximum pour un déplacement de l'animal.
- Constante *metabolisme(X)* : c'est l'énergie dépensée à chaque tour par X pour vivre.
- Constante *gestation(X)* : c'est le facteur d'énergie à dépenser pour se reproduire.
- Constante *frequence_reproduction(X)* : c'est le nombre de tour entre deux reproductions successives.

2.2 Règles de vie

Règle Survie(X) : X peut-il encore survivre ?
Si *satiete(X)= 0* ET *tour_courant − dernier_repas(X)> duree_survie(X)* Alors l'animal meurt.

Règle du Tour(X) :
A chaque tour, *satiete(X)<− max(satiete(X)− metabolisme(X), 0)*.

Règle Prédation(X,Y) : X peut-il manger Y ?
Si X a dans son voisinnage immédiat (8 cases) un élément Y et *satiete(X)+ taille(Y)< taille_du_bide(X)*, Alors il se déplace sur la case de Y et *dernier_repas(X)<− tour_courant* et *satiete(X)<− satiete(X)+ taille(Y)*. L'animal Y meurt et disparait.

Règle *Déplacement(direction, X, nb_case)* : X se déplace dans une direction.
L'animal X veut se déplacer de *nb_case*, dans une des 8 directions (donnée par la variable direction). Chaque case que l'animal parcourt lui fait perdre un point de satiete(X). Le déplacement se fait pendant *nb_case* tant qu'il n'y a pas d'obstacles.
Les règles suivantes doivent être vérifiées avant le déplacement :
- *satiete(X)−nb_case >= 0*
- *nb_case<saut_max(X)*

Le déplacement effectué conduit à :
- modifier les coordonnées de X
- *satiete(X)<− satiete(X)− nb_case*'

où *nb_case*' est le nombre de case réellement parcourues.

Règle Reproduction(X) :
La reproduction d'un individu X est possible si les conditions suivantes sont réunies :
- satiete(X) >= gestation(X) * metabolisme(X)
- une des 8 cases adjacentes est libre
- *derniere_reproduction(X)+ frequence_reproduction(X)< tour_courant*

Si les conditions sont réunies, alors :
- *satiete(X)<− satiete(X)− gestation(X)∗ metabolisme(X)*

– le nouvel individu occupe une des 8 cases adjacentes
– *derniere_reproduction(X)<− tour_courant*

2.3 Tableau de prédation

X mange Y	plancton	corail	bar	thon	pollution	pyranha	requin	orque	baleine	pêcheur	pont
plancton											
corail	×										
bar	×										×
thon	×										
pollution		×									
pyranha			×	×						×	
requin			×							×	
orque							×			×	
baleine	×										×
pêcheur			×	×		×	×	×	×		
pont											

Le tableau précédent donne les dépendances de prédation entre les différentes espèces. Cela permet de définir quant entre X et Y la règle Prédation(X,Y) est possible.

2.4 Caractéristiques particulières

Plancton
– Survit presque indéfiniment (même si non nourri)
– Ne se déplace pas

Pollution
– Survit presque indéfiniment (même si non nourri)
– Ne se reproduit pas

3 Actions des pêcheurs

Les pêcheurs évoluent sur un pont qui est en construction en partant de la terre ferme (bord de la carte). Ils peuvent pêcher des animaux utilisés comme matière pour la construction du pont. Les ponts peuvent être détruis petit à petit par l'écosystème, par exemple par les bars les baleines.

Les actions suivantes prennent un tour pour être effectuées et on ne peut faire deux actions à la fois.

Le pêcheur possède un sac de matière constructible dans la variable *sac*.

3.1 Action pêche

Chaque pêcheur peut pêcher de la façon suivante :
– A la canne à pêche : pêcher 1 unité de X à distance *taille_canne_a_peche* (constante). *sac <− sac + taille(X)*.
– Au filet : pêcher $k < 9$ unités de X à distance *distance_peche_filet* (constante). $sac < -sac + \sum_{i=1..k} taille(X_i)$.

3.2 Action déplacement

A chaque tour, le pêcheur peut se déplacer d'une case sur le pont.

3.3 Action construction

A chaque tour, le pêcheur peut construire des morceaux de pont sur les cases contiguës au pont existant à l'aide des matières contenues dans son sac. Lorsqu'il construit une nouvelle case, $sac--$. Il ne peut construire qu'une case par tour.

3.4 Respawn

Le pêcheur peut tomber à l'eau si le pont de sa case est consommé par un animal.
- Une première version simple considérera que le pêcheur retourne directement à son point de départ : dans ce cas on considère qu'il respawn. (il ne peut être "mangé" par personne).
- Une version plus compliquée à mettre en œuvre consiste à gérer le fait que le pêcheur doit nager et retourner sur la plus proche zone pédestre tout en gérant le fait qu'il existe des prédateurs du pêcheur dans l'écosystème (il peut être mangé !). Quand le pêcheur nage, il se déplace deux fois moins vite que sur la terre (deux tours sont nécessaires pour se déplacer d'une case). Si le pêcheur est consommé, il respawn au point de départ (son oncle prend la relève).

3.5 Lancer le poisson

Dans le cas du jeu à plusieurs pêcheurs concurrents (partant de points opposés et souhaitant traverser dans des directions opposées), il est possible de "lancer" le poisson pêché près du pont adverse dans l'espoir qu'il le mange.

La distance de jet est limitée à $3 * taille\_canne\_a\_peche$.

4 Déroulement d'un tour

Un tour de jeu se déroulera schématiquement de la façon suivante :

1. Evolution de l'écosystème
   - Parcourir les cases de l'écosystème
   - La case est soit vide, soit contient un individu X
     - Appliquer la règle de Survie
     - Appliquer la règle de Reproduction
     - Appliquer la règle de Predation
     - Appliquer la règle de Déplacement
     - Appliquer la règle de Tour
2. Afficher la représentation de l'écosystème à l'écran
3. Proposer au joueurs d'effectuer une action
4. Réaliser l'action choisie
   - Action déplacement
   - Action pécher

- Action construire pont
- Action lancer le poisson

5. Afficher la représentation de l'écosystème à l'écran

Evidemment, cet algorithme est schématique et donne le principe général de votre programme. Il deviendra de plus en plus complexe au fur et à mesure que vous allez ajouter des nouvelles fonctionnalités.

5 Phases du projet

5.1 Etape 1 : simulation de l'écosystème

La première étape du projet consiste à réaliser la simulation de l'ecosystème, sans prendre en compte le pécheur et et le pont. L'écosystème doit pouvoir évoluer au tour par tour en respectant les règles de vie du système.

5.2 Etape 2 : paramétrage de l'écosystème

Pour que l'écosystème survive, il faudra le paramétrer convenablement (à l'aide des constances des espèces). Il faudra sans doute adapter les constantes pour obtenir un système qui soit "stable" c'est à dire qui survive pendant un nombre de tours raisonnable.

Il sera intéressant de présenter le paramétrage obtenu et en quoi il est pertinent.

5.3 Etape 2bis : analyse de l'évolution de l'écosystème

Les courbes d'évolution des populations peuvent permettre de montrer comment le système évolue. A titre d'exemple, vous trouverez en Annexe une interface du projet Wator qui montre une courbe avec 2 populations. Pour votre projet, vous tracerez bien entendu les courbes pour toutes vos populations pour pouvoir observer le comportement global du système.

5.4 Etape 3 : le pêcheur et le pont

A ce stade, votre écosystème fonctionne. Vous pourrez alors ajouter la gestion du pont (il faut que l'écosystème n'écrase pas le pont, sauf certains prédateurs qui mangent le pont).

Puis il sera alors possible de gérer un pêcheur qui se déplace sur ce pont. Vous pourrez alors implémenter les différentes actions du pêcheur une à une.

5.5 Etape 4 : analyse de l'évolution de l'écosystème incluant le pêcheur

Il y a de fortes chances pour que la présence du pêcheur modifie l'équilibre de l'écosystème. Du coup le paramétrage analysé en question 3bis doit être recalibré. Proposez ce nouveau calibrage et analysez le (courbes) comme en question 2bis.

5.6 Etape 5 : multi-pêcheur

Enfin, lorsqu'un premier pêcheur sera géré, il sera envisageable de gérer plusieurs pêcheur à la fois qui jouent en même temps sur la même machine.

6 Modalités générales

Le projet sera réalisé en binômes. Les groupes seront définis sur le serveur enseignement dans le cours associé au projet.

L'évaluation du projet sera constituée de :

- Un compte rendu PDF comprenant :
  - L'analyse du problème, vos choix d'implémentation
  - Les algorithmes et explications
  - Les résultats expérimentaux (courbes, matrices de paramétrage, etc...)
  - Les choix que vous avez été conduit à faire et les libertés que vous avez pris par rapport au sujet (toutes les initiatives sont les bienvenues si elles sont pertinentes et justifiées).
- Un dossier de programmation comprenant :
  - Vos programmes C dans un .tar.gz facilement compilable (une aide pour la compilation est bienvenue (fichier texte))

7 Initialisation de l'écosystème

Il faut remplir la grille de l'écosystème avec des individus. Nous donnons l'algorithme naïf pour initialiser cette grille.

7.1 algorithme naïf

L'idée la plus simple est de décider arbitrairement d'un taux d'occupation pour une espèce donnée, par exemple, la pollution occupe 15% des cases. Ensuite, on tire pour chaque individu "pollution" une case au hasard sur la grille. On obtient donc une répartition équiprobable de tous les individus sur la grille.

Vous déterminerez expérimentalement un bon taux d'occupation pour chaque espèce.

7.2 algorithme évolué

Vous réfléchirez à une méthode algorithmique pour initialiser la grille qui se rapprochera d'un modèle plus réaliste (banc de poissons, nuage de pollution, etc...).

8 Annexes

8.1 Références

- Jeu de la vie `http://fr.wikipedia.org/wiki/Jeu_de_la_vie`
- Wator `http://www.cip.physik.uni-muenchen.de/~wwieser/sim/wator/`
- Bar `http://fr.wikipedia.org/wiki/Bar_(poisson)`
- Thon `http://fr.wikipedia.org/wiki/Thon`

8.2 Exemple d'interface pour un écosystème à 2 populations

Vous trouverez en figure 1 une interface graphique représentant une simulation pour le projet Wator ou des requins mangent des poissons. Vous remarquerez la courbe d'évolution

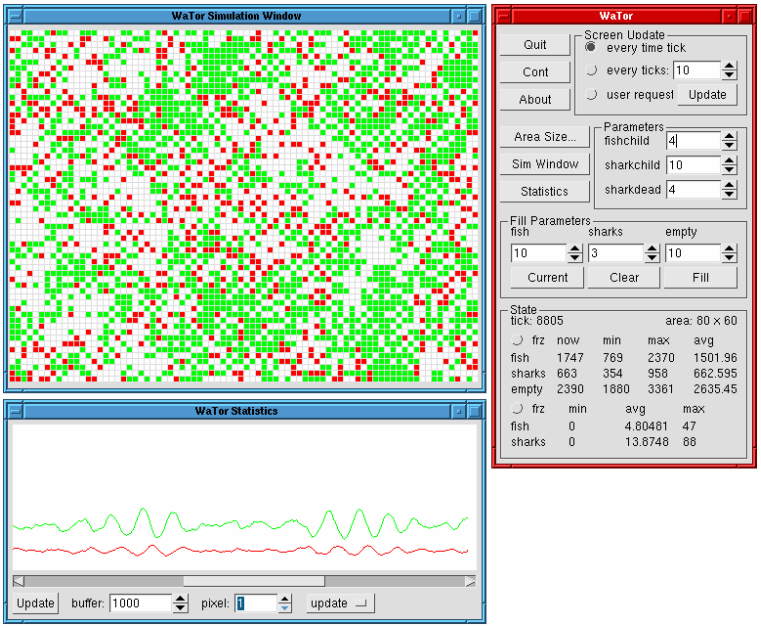

FIG. 1 – Exemple d'interface pour la simulation Wator

des populations qui montre qu'une population dépend de l'autre. A droite, on trouve les différents paramètres qui permettent de régler le modèle. Bien entendu, votre version sera en mode texte mais les plus courageux peuvent se lancer dans des interfaces graphiques.

8.3 Idées pour l'interface graphique

La première version de votre programme n'aura pas d'interface graphique. Le but n'est pas d'avoir une interface graphique développée mais plutôt de soigner vos algorithmes d'évolution de l'écosystème et le contrôle du pêcheur. La première idée est donc d'afficher en mode texte l'écosystème et de le réafficher, à la suite en faisant défiler la console.

Une version améliorée de l'affichage en mode texte peut consister à afficher la grille et à la mettre à jour en temps réel (sans faire défiler la console).

Une version ultra améliorée peut consister à utiliser des librairies de C graphique (par exemple la SDL) pour afficher la grille.

8.4 Exemple de définition des constantes

Voici deux exemples d'affectation arbitraires dont la pertinence n'est pas garantie pour la survie de l'écosystème.

X = bar

- Constante *duree_survie(X)*=4
- Constante *taille(X)*=3
- Constante *taille_du_bide(X)*=2
- Constante *saut_max(X)*=1

- Constante *metabolisme(X)*=0.1
- Constante *gestation(X)*=4
- Constante *frequence_reproduction(X)*=3

X = pollution

- Constante *duree_survie(X)*=infini
- Constante *taille(X)*=0
- Constante *taille_du_bide(X)*=taille(plancton)
- Constante *saut_max(X)*=1
- Constante *metabolisme(X)*=0
- Constante *gestation(X)*=0
- Constante *frequence_reproduction(X)*=0 (jamais)

X = plancton

- Constante *duree_survie(X)*=infini
- Constante *taille(X)*=1
- Constante *taille_du_bide(X)*=1
- Constante *saut_max(X)*=0
- Constante *metabolisme(X)*=0
- Constante *gestation(X)*=0
- Constante *frequence_reproduction(X)*=1