Animauphine

Projet Programmation C, L2 MIDO 2019/2020

1 Dates importantes

- Deadline pour la composition des binômes sur le Wiki de MyCourse : 4 décembre 2019
- Rendu du projet : à définir.
- Soutenances : à définir. Probablement la semaine du 3 février.

2 Versions

Ce sujet est susceptible d'être modifié.

- 29 novembre 2019 : mise en ligne du sujet.

3 En quelques mots

Le but du projet est de modéliser la sélection naturelle des espèces selon leur adaptabilité à l'environnement. On considérera un monde où un ou plusieurs individus initiaux se déplacent selon un code génétique, et dont les descendants se reproduisent, évoluent et s'adaptent à leur environnement au bout de plusieurs (millions de) générations. Sur le long terme, cela permet de déterminer quelle population s'est le mieux adaptée au milieu.

On supposera le temps comme étant "tour par tour". Le monde est considéré comme une grille à 2 dimensions de cellules. Les animaux se déplacent d'une cellule à une cellule adjacente à chaque tour, au prix d'une unité d'énergie. Quand un animal n'a plus d'énergie, il meurt et disparaît de la simulation. Des morceaux de nourriture sont placés à chaque tour de jeu. Quand un animal atteint une case contenant de la nourriture, il la mange et son énergie augmente. Si son énergie dépasse un certain seuil, il se sépare en 2 animaux récupérant chacun la moitié de son énergie.

Chaque animal possède un "chromosome" qui est ici un vecteur de 8 entiers positifs, appelés gènes. Ce chromosome n'est pas modifié pendant toute la durée de vie de l'animal. À chaque tour, un gène parmi les 8 est sélectionné aléatoirement (il est *activé*). Le choix aléatoire est pondéré par l'entier correspondant au gène, le(s) gène(s) ayant la plus grande valeur est dit *dominant*. L'activation du gène $k, k = 0, \ldots, 7$ fait faire une rotation d'angle $45k$ degrés par rapport à la position actuelle de l'animal. Après sa rotation, l'animal avance d'une cellule vers l'avant. Par conséquent, un animal dont le gène dominant est le gène 0 aura tendance à avancer en ligne droite, si son gène dominant est le gène 2, il aura tendance

à faire des cercles serrés. Ainsi, le chromosome de l'animal influe sur ses mouvements et donc sur sa chance de trouver de la nourriture et de survivre.

Un animal hérite du chromosome de son parent modulo des petites mutations. Ceci va permettre de voir les mutations favorables ou non à la survie (spéciation). Une petite partie du monde sera considérée comme "fertile", avec plus de nourriture, elle sera appelée la "Beauce" (au contraire du "désert" où la nourriture est plus rare).

Le but de la simulation est de mettre en évidence deux grandes classes d'espèces. Une dite "du désert" où ses gènes lui permettent de faire de longues distances pour trouver plus facilement de la nourriture éparpillée, et une autre espèce dite "Beauceronne", avec des gènes lui permettant de rester dans la Beauce (cercles concentriques). Il faudra certainement plusieurs millions d'itérations pour obtenir un tel résultat.

4 Description plus détaillée

Le *monde* est un rectangle dont les dimensions de hauteur (h) et largeur (l) sont spécifiées à la création. Il se comporte comme un tore : si un animal quitte le monde par le bas, il apparaît par le haut. Idem pour les côtés. Le monde est une grille discrétisée en $h \times l$ cellules. On note la cellule située à la ligne i et la colonne j par (i, j). La case en haut à gauche est la case $(0, 0)$, celle en bas à droite est $(h - 1, l - 1)$. Chaque cellule comporte 8 voisines, numérotées de 0 à 7 (y compris pour les cellules du bord du monde) dans le sens contraire des aiguilles d'une montre (donc le sens trigonométrique), où la cellule 0 est celle qui se trouve directement sous elle (au sud), la 2 celle qui se trouve directement à sa droite (à l'est) etc.

La *Beauce* est un sous-rectangle du monde dont les dimensions et positions sont spécifiées à la création (une dimension de 0×0 indique qu'il n'y a pas de Beauce). Par défaut, on peut considérer que la Beauce est toujours au centre du monde. Un projet plus évolué peut considérer que la Beauce peut être positionnée n'importe où. Certes, la structure en forme de tore du monde implique que la position de la Beauce est immatérielle, donc toutes ses positions sont équivalentes. Cependant, cette position a une implication pour les calculs de distance des animaux à la Beauce qui seront nécessaires (explications plus loin).

À chaque pas de temps, un morceau de *nourriture* est déposé sur une cellule choisie aléatoirement. Si la Beauce existe, il y a de plus un autre morceau de nourriture qui est déposé sur une de ses cases choisie aléatoirement. Un morceau de nourriture contient de l'énergie (la valeur est spécifiée à la création). Une cellule peut contenir plus d'un morceau de nourriture à un moment donné.

Les *animaux* se déplacent de cellules en cellules. Plusieurs animaux peuvent être sur une même cellule à un moment donné. À chaque pas de temps, chaque animal se déplace sur une cellule de son voisinage selon un mécanisme expliqué plus loin. Un déplacement coûte 1 unité d'énergie à l'animal. Si un animal n'a plus d'énergie (0), il meurt et est retiré du monde. S'il arrive sur une cellule contenant un ou plusieurs morceau(x) de nourriture, il consomme un morceau et gagne de l'énergie.

Chaque animal possède un *chromosome* $\langle g_0, g_1, \ldots, g_7 \rangle$, où chaque gène est un entier $\geqslant 1$ (une version $\geqslant 0$ est envisageable). Un chromosome représente un animal, il ne change pas au cours de sa vie.

Un animal se *déplace*. Il fait face à une des 8 cellules qui l'entourent. À chaque pas de

temps, un des 8 gènes de l'animal est sélectionné aléatoirement et est *activé*. L'activation de ce gène fait tourner l'animal de $45k$ degrés selon le sens contraire des aiguilles d'une montre par rapport à son orientation courante, où k est le gène activé. Par conséquent, si l'animal était face à la direction $d, d \in \{0, \ldots, 7\}$, l'activation du gène $g_k, k \in \{0, \ldots, 7\}$ le fera faire face à la direction $(d + k) \bmod 8$. Après son orientation, l'animal se déplace d'une case en avant.

La *probabilité de sélection d'un gène* est proportionnelle à la valeur du gène. Par conséquent, un gène avec une haute valeur est plus probablement activé qu'un autre. Plus exactement, le gène k est activé avec probabilité $g_k / \sum_{i=0}^{7} g_i$. Par exemple, dans le chromosome $\langle 1, 1, 2, 1, 10, 10, 5, 5 \rangle$, le gène g_4 est activé avec probabilité $10/35$. À vous de trouver un moyen de simuler cet aléatoire pondéré.

Quand un animal dépasse (supérieur ou égal) un certain seuil d'énergie (défini à la création), il se *reproduit*, en se séparant en deux de manière asexuée. Les deux parties sont des clones ayant chacun la moitié (troncation) de l'énergie du parent. Cependant, pendant la reproduction, un des gènes (choisi aléatoirement et de manière uniforme) de l'animal nouvellement créé peut muter. La mutation potentielle d'un gène du chromosome consiste à lui ajouter un nombre aléatoire de l'ensemble $\{-1, 0, 1\}$ (probabilité uniforme). Notons qu'un gène ne pourra toutefois pas descendre en dessous de 1 (ou 0 si cette version est envisagée).

La présence de la nourriture ou non dans chaque case sera gérée à l'aide d'un *tableau*. Les animaux seront eux stockés dans une *liste chaînée* (à chaque pas de temps, on parcourt la liste et on effectue le déplacement et les éventuelles actions pour chaque animal). De manière optionnelle, on désire également connaître les descendants de chaque *famille d'animaux* – les animaux présents lors de la création du monde représentent chacun une famille. Pour cela, chaque animal fera également partie d'une *liste chaînée de sa famille*. Il devra être possible de lister les familles (description plus loin).

Pour gérer l'*aléatoire*, vous devez faire appel à la fonction `rand()`. Pour avoir des nombres aléatoires différents à chaque lancement de la simulation, vous devez faire appel une seule fois et au début de votre programme à la fonction `srand`[1]. Voir également les lignes de codes fournies sur MyCourse. Pour comparer vos programmes et tester en ayant le même comportement, on demande une version de la simulation où cette ligne initiale n'est *pas* appelée. Ainsi, tous les choix aléatoires devraient être les mêmes et donc votre simulation devrait sortir la même chose à partir d'une même configuration initiale.

La simulation doit se dérouler de la manière suivante : une fois le monde chargé, on effectue autant d'itérations de temps que demandé. Pour chaque itération de temps, on ajoute aléatoirement de la nourriture sur une case du monde, on ajoute aléatoirement de la nourriture sur une case de la Beauce si elle existe, et on parcourt la liste des animaux. Pour chaque animal, on l'oriente selon le gène activé, on le déplace, on le nourrit s'il y a de la nourriture sur la case, on le tue s'il n'a plus d'énergie, on le fait se reproduire s'il a trop d'énergie, l'animal nouvellement créé subit une éventuelle mutation d'un de ses gènes et il est ajouté en tête de la liste des animaux (pour ne pas le retraiter ce tour ci). Une fois toutes les itérations finies, on effectue des exports d'images comme décrit plus loin, on trie la liste des animaux comme décrit plus loin et on exporte le fichier comme décrit plus loin.

Les projets les plus aboutis et ayant déjà implémenté tout ce qui est décrit dans le sujet

1. http://manpagesfr.free.fr/man/man3/rand.3.html

pourront réfléchir à des améliorations (par exemple des animaux avec une vitesse plus importante au prix d'une dépense d'énergie plus grande, des animaux avec des bras plus longs pouvant ainsi attrapper de la nourriture plus loin au prix d'une dépense d'énergie plus importante etc). Avant de s'y engager, discutez-en avec un des chargés de TD (par email ou de vive voix).

Format de fichier

Votre simulation doit lire les paramètres du monde dans un fichier d'extension nommée *phine*. La forme générale d'un fichier phine est la suivante, les commentaires sont symbolisés par un dièse (la suite de la ligne est alors ignorée jusqu'au bout) :

```
#monde de test avec petite Beauce
Monde 200  200   #hauteur largeur
Beauce  95 95 10 10   #position du coin superieur gauche de la Beauce puis sa
    hauteur largueur
Energie Nourriture 80   #gain d'energie si de la nourriture est mangee
Seuil Reproduction 300   #seuil d'energie a partir duquel un animal se duplique

#animaux, un par ligne
(0 0) 0 200 | 5 5 5 5 5 5 5 5 | #position (x,y) initiale de l'animal, direction
    initiale de l'animal (entre 0 et 7), energie initiale de l'animal puis
    chromosome de l'animal
```

Le programme doit indiquer à l'utilisateur que le format du fichier est illisible ou incorrect si c'est le cas.

Entrées Sorties

Le programme (appelé `animauphine` dans l'exemple) se lancera de la manière suivante : `./animauphine a n monde.phine sortie.phine [c]`

où `a` vaut 1 s'il y a un appel à la fonction `srand` comme indiqué plus haut, ou 0 sinon, `n` $\geqslant 0$ est le nombre d'itérations effectuées par la simulation (un entier), `monde.phine` est le nom du fichier phine décrivant la configuration initiale du monde, `sortie.phine` est le nom du fichier phine donnant la situation du monde une fois la simulation terminée, et `c` $\geqslant 0$ est un paramètre optionnel (un utilisateur peut lancer le programme sans indiquer ce nombre, les crochets symbolisent le fait qu'il est optionnel et ne doivent pas être tapés) décrit plus loin. Vous pouvez utiliser la fonction `sscanf` pour vous aider. Si les arguments du programme ne correspondent pas à ce qui est attendu, le programme doit afficher ce qu'il attend et quitter. Pour vous aider, nous fournissons sur MyCourse quelques lignes de code récupérant ces arguments.

Comme il faut plusieurs centaines de milliers, voire des millions, d'itérations pour rendre la simulation intéressante, on attend que la simulation indique le nombre d'itérations effectuées toutes les 100 000 étapes.

Optionnellement, vous pouvez gérer le nombre d'itérations selon une notation à suffixe K, M, G. Ainsi, `n` peut valoir 10K et correspondre à 10000 itérations, 15M pour 15 millions ou encore 2G pour 2 milliards.

Pour la sortie, on attend que les animaux vivant toujours dans le monde soient *triés selon leur distance à la Beauce* (afin de pouvoir analyser les résultats de la simulation), du plus près au plus loin. La fonction de distance devant être utilisée entre un animal situé en cellule (a_i, a_j) et le centre de la Beauce situé en (b_i, b_j) est le maximum entre le nombre de cases sur l'axe des abscisses et le nombre de cases sur celui des ordonnées, c'est-à-dire si la Beauce est centrée : $\max(|a_i - b_i|, |a_j - b_j|)$. Attention au cas où la Beauce n'est pas centrée dans le monde (qui est un tore).

Optionnellement, on veut également connaître les animaux par famille dans ce fichier de sortie. Pour cela, après la liste des animaux, on aura 2 lignes vides, puis les animaux de la première famille, puis une ligne vide, puis les animaux de la seconde famille, puis une ligne vide, etc. (chaque animal apparaît donc au total 2 fois dans le fichier, une fois dans la première liste (triée par distance), puis une fois dans la liste de sa famille). Comme on désire que les fichiers de sorties puissent également être utilisés en entrée de la simulation, on fera en sorte que la lecture de l'entrée s'arrête après la lecture de la première liste des animaux.

La génération d'images est réservée aux projets les plus aboutis. Le paramètre `c` est un entier correspondant au nombre de *fichiers images générés* par la simulation une fois les `n` itérations effectuées. Chaque génération d'image est entrecoupée d'une itération du temps. Par exemple, si le programme est lancé de la manière suivante : `./animauphine 0 1000000 monde.phine sortie.phine 20`, la simulation effectuera 1 million d'itérations (sans `srand`), puis générera un fichier image, effectuera une itération de temps, générera un fichier image, effectuera une itération de temps etc. (donc 20 images et 1000019 itérations au final). Les images seront nommées sortie0001.ppm, sortie0002.ppm, ..., sortie0020.ppm. Puis le fichier sortie.phine sera créé (on remarque que si `n=0` et `c=1`, le programme génère l'image correspondant au monde initial).

Les images doivent être générées selon le format PPM ASCII https://fr.wikipedia.org/wiki/Portable_pixmap (nombre magique P3). Ce format permet de décrire une image en indiquant le niveau de Rouge, Vert, Bleu de chaque pixel en format ASCII. La plupart des visionneurs d'images lisent ce format. Sous Linux, `display` le fait par exemple[2]. Vous pouvez générer une sortie d'animation de vos images avec la commande `animate -delay 50 sortie*.ppm`.

Votre image doit représenter le monde (animaux, nourriture, position de la Beauce). Le plus simple est de représenter chaque case par un pixel. Optionnellement, vous pouvez améliorer cela en représentant chaque case par un carré de 4 ou 9 pixels (i.e. un carré de côté 2 ou 3 plutôt que 1). Chaque famille d'animaux doit avoir *une couleur différente* sur l'image. Si plusieurs animaux de familles différentes sont sur une même case, la couleur affichée est celle de l'animal avec le plus d'énergie (pour cela, vous ne devez pas avoir à parcourir la liste des animaux – vous devez être plus malins).

5 Conditions de rendu

Le projet est à effectuer en **binôme**, *i.e.* par 2 (**DEUX**) personnes. En cas de nombre impair d'élèves, **un seul** groupe sera autorisé à effectuer le projet en **trinôme** (notation plus

2. d'autres possibilités multiplateformes : https://nomacs.org/, https://www.digikam.org/...

sévère). Si au final il y a plus d'un trinôme ou plus de zéro monôme, les projets correspondants auront une note de 0. Pour former les groupes, utiliser le wiki mis à disposition sur l'espace MyCourse.

Le projet est à rendre avant la date et l'heure indiquées plus haut sur l'espace MyCourse dédié. **Chaque heure de retard sera pénalisée d'un point sur la note finale** (une heure entamée étant due). Une fois votre fichier envoyé, vous devez avoir un écran de confirmation avec votre fichier et la date de l'envoi. Le format de rendu est une archive au format ZIP contenant :

- Le code-source de votre projet (éventuellement organisé en sous-dossiers).
- Un répertoire *docs* contenant :
  - Une documentation pour l'utilisateur *user.pdf* décrivant à un utilisateur quelconque comment se servir de votre projet.
  - Une documentation pour le développeur *dev.pdf*, devant justifier les choix effectués, les avantages et inconvénients de vos choix, expliquer les algorithmes et leur complexité, indiquer quelles ont été les difficultés rencontrées au cours du projet ainsi que la répartition du travail entre les membres du binôme. Un programmeur averti devra être capable de faire évoluer facilement votre code grâce à sa lecture. En aucun cas on ne doit y trouver un copier/coller de votre code source. Ce rapport doit faire le point sur les fonctionnalités apportées, celles qui n'ont pas été faites (et expliquer pourquoi). Il ne doit pas paraphraser le code, mais doit rendre explicite ce que ne montre pas le code. Il doit montrer que le code produit a fait l'objet d'un travail réfléchi et minutieux (comment un bug a été résolu, comment la redondance dans le code a été évitée, comment telle difficulté technique a été contournée, quels ont étés les choix, les pistes examinées ou abandonnées...). Ce rapport est le témoin de vos qualités scientifiques mais aussi littéraires (style, grammaire, orthographe, présentation).
  - Un court rapport d'expériences appelé *exp.pdf* où vous analyserez le résultats de vos simulations (selon le nombre d'itérations, la configuration initiale, la taille du monde/de la Beauce, les seuils d'énergie, etc., quels types d'espèces survit ? Pourquoi ?).
- Un ou plusieurs fichiers phine de tests.
- Optionnellement un makefile.

L'archive aura pour nom Nom1Nom2.zip, où Nom1 et Nom2 sont les noms des membres du binôme par ordre alphabétique. L'extraction de l'archive devra créer un dossier Nom1Nom2 contenant les éléments précisés ci-dessus.

Il va sans dire que les différents points suivants doivent être pris en compte :

- Projet compilant sans erreur ni warning avec l'option -Wall de gcc et fonctionnant sur les machines de l'université (on vous conseille de compiler vos projets avec l'option `-O3` qui permet **une exécution plus rapide** de vos programmes).
- On préfère un projet fonctionnant parfaitement avec peu de fonctionnalités qu'un projet qui a tenté de répondre à tout mais mal.
- Vérification des données entrées par l'utilisateur et gestion des erreurs.

- Uniformité de la langue utilisée dans le code (anglais conseillé) et des conventions de nommage et de code.
- La documentation ne doit pas être un copié-collé du code source du projet.
- Les sources doivent être commentées, dans une unique langue, de manière pertinente (pas de commentaire "fait un test" avant un if.).
- Les noms des variables et fonctions doivent être choisis judicieusement.
- Le code doit être propre et correctement indenté.
- Bonne gestion de la libération de la mémoire (utilisez valgrind pour vérifier que le nombre de `free` est égal au nombre de `malloc` : il faut d'abord compiler avec l'option `-g`, puis lancer valgrind avec en argument le nom du programme. Par exemple :

  ```
  gcc -g main.c
  valgrind ./a.out
  ```

  Valgrind peut également vous indiquer la ligne où un segmentation fault a eu lieu, pratique !
- Le projet doit évidemment être propre à chaque binôme. **Un détecteur automatique de plagiat sera utilisé**. Si du texte ou une petite portion de code a été empruntée (sur internet, chez un autre binôme), il faudra l'indiquer dans le rapport, ce qui n'empêchera pas l'application éventuelle d'une pénalité. Tout manque de sincérité sera lourdement sanctionné (conseil de discipline) – c'est déjà arrivé.

La documentation (rapports, commentaires...) compte pour un quart de la note finale.

5.1 Soutenance

Une soutenance d'une dizaine de minutes aura lieu pour chaque binôme, 2 binômes par 2 binômes dans l'ordre (2 jurys), à la date et à l'heure indiquées plus haut. Elle doit être préparée et menée par le binôme (*i.e.* fonctionnant parfaitement du premier coup, avoir préparé des jeux de tests intéressants, etc.). Nous aurons téléchargé vos projets (tels qu'envoyés sur MyCourse) sur les machines unix (donc vous n'avez pas besoin d'amener votre ordinateur et vous n'aurez pas besoin de vous loguer). Vous devrez en revanche compiler votre code.

Pendant la soutenance, ne perdez pas de temps à nous expliquer le sujet : nous le connaissons puisque nous l'avons écrit. Essayez de montrer ce qui fonctionne et de nous convaincre que vous avez fait du bon travail.