Compléments d'informatique

Projet 2 : simulateur d'écosystème

22 octobre 2025

L'objectif concret de ce projet est d'implémenter un simulateur simplifié d'un écosystème dans lequel évoluent différents animaux. L'objectif pédagogique est de vous faire pratiquer la programmation modulaire, le programme complet étant constitué de plusieurs modules séparés interagissant entre eux. Ce projet est à réaliser **seul ou à deux**. La date limite de remise est précisée sur eCampus et sur Gradescope.

1 Description du simulateur

Le principe du simulateur est de faire évoluer des animaux dans un environnement. On se focalisera sur deux types d'animaux, des lapins et des loups, représentant un système proie-prédateur classique. Les lapins, herbivores, se nourrissent d'herbe, alors que les loups, carnivores, se nourrissent des lapins. Chaque animal, s'il est bien nourri, peut se reproduire. Une fois le simulateur implémenté, vous pourrez reproduire l'évolution oscillatoire du nombre d'animaux dans chaque population caractéristique des systèmes proie-prédateur réels.

Représentation de l'environnement. L'environnement dans lequel les animaux évoluent sera représenté par une grille rectangulaire de taille $H \times W$, avec H la hauteur de la grille et W sa largeur. A chaque instant, une case de la grille peut être vide ou contenir soit de l'herbe, soit un (seul) animal.

Simulation. L'environnement évolue en temps discret. L'évolution de l'environnement sur un nombre T de pas de temps à partir d'un état initial est appelé une simulation. A chaque pas de temps, chaque animal peut effectuer une action selon des règles prédéfinies dépendant du type d'animal, l'amenant à se déplacer sur la grille, à manger et/ou à se reproduire. Chaque animal dispose d'un énergie propre qui est diminuée d'une certaine

valeur à chaque action et augmente lorsqu'il se nourrit. Si l'énergie de l'animal tombe à zéro, il meurt et est supprimé de l'environnement. Les animaux disposent également d'une priorité déterminant l'ordre dans lequel ils sont mis en action. Les animaux de valeur de priorité **plus faible** sont **les premiers à être activés**. Une fois que tous les animaux ont effectué leur action, de l'herbe peut pousser sur certaines cases vides, mettant fin au pas de temps en cours. Les règles d'évolution générales sont décrites ci-dessous, avant de détailler les règles spécifiques relatives aux lapins et aux loups.

Règles générales. Un pas de temps de simulation consiste à l'application des règles suivantes (dans cet ordre) :

- Pour chaque valeur de priorité p allant de 0 à la priorité maximale :
  - Pour chaque animal de priorité p présent sur la grille :
    1. Si son énergie est inférieure ou égale à zéro, l'animal meurt et laisse une case vide sur la grille.
    2. Sinon, l'animal détermine une action selon sa stratégie propre (voir plus bas). Une action est définie par un déplacement sur la grille et un booléen indiquant si l'animal peut se nourrir en se déplaçant sur la nouvelle case.
    3. L'action choisie par l'animal est réalisée par la simulation :
       (a) L'énergie de l'animal est diminuée d'un coût dépendant de l'animal.
       (b) L'animal est déplacé sur la nouvelle case, ce qui laisse la case précédente où se trouvait l'animal vide. Cette nouvelle case peut être vide ou contenir de l'herbe ou un animal. Ceci dépend de chaque animal. Par exemple, un lapin ne peut pas se déplacer sur une case où se trouve un autre animal, mais peut aller sur une case vide ou une case herbe, auquel cas il mangera l'herbe.
       (c) Si le déplacement permet à l'animal de se nourrir suite au mouvement, son énergie est augmentée d'une valeur dépendant de l'animal.
       (d) Dans le cas où l'animal se nourrit, il se reproduit dès que son énergie dépasse un certain seuil dépendant de l'animal. S'il se reproduit, sa descendance est placée sur la case qu'il vient de quitter et qui a été laissée vide, et il perd une certaine quantité d'énergie fixée.
- Pour chaque case vide de la grille, on y fait pousser de l'herbe avec une certaine probabilité p_h dépendante de la présence d'herbes dans les cases adjacentes.

Remarque : Il est possible de parcourir les animaux de la grille d'une même priorité dans n'importe quel ordre. Pour simplifier l'implémentation, on vous demande de les parcourir ligne par ligne (de la ligne d'indice 0 à la ligne d'indice $H-1$), et pour chaque ligne, de la colonne d'indice 0 à celle d'indice $W-1$. On fera attention cependant lors de ce parcours à ne mettre en action qu'une seule fois chaque animal et à ne pas mettre en action un animal nouvellement né.

Lapins. Les lapins se nourrissent d'herbes et essayent de fuir les loups. Il peuvent rester en place ou se déplacer d'une case vers la gauche, la droite, en haut ou en bas (pas de mouvement oblique). Soit (i, j), une des 5 positions accessibles par un lapin. On associera un score $S(i, j)$ à cette position selon la formule suivante :

$$\begin{cases} S(i,j) = -w_g d_g(i,j) + w_w d_w(i,j) & \text{Si } (i,j) \text{ est vide, contient de l'herbe ou le lapin,} \\ S(i,j) = \texttt{INT\_MIN}\ ^1 & \text{Si } (i,j) \text{ contient un autre animal (lapin ou loup),} \end{cases}$$

où $d_g(i, j)$ (resp. $d_w(i, j)$) est la distance de Manhattan [2] entre (i, j) et l'herbe (resp. le loup) la plus proche de (i, j). Notez que dans le cas où (i, j) contient de l'herbe, $d_g(i, j)$ vaudra 0. w_g et w_w sont deux poids positifs permettant de favoriser la recherche de nourriture ou la fuite face à un loup. Ce score tend donc à favoriser les cases proches d'herbes et éloignées de loups. Le lapin n'ayant pas une vision infinie, il ne peut pas voir au-delà d'une certaine distance d_{max}. Si aucune herbe (resp. aucun loup) ne se trouve à une distance d_{max} de (i, j), alors on fixera $d_g(i, j)$ (resp. $d_w(i, j)$) à $d_{max} + 1$. Le lapin se déplacera vers la case (i, j) de score maximal. Si il y en a plusieurs, alors la case devra être choisie aléatoirement parmi celles de score maximal. Dans le cas où la case (i, j) atteinte contient de l'herbe, le lapin se nourrira de cette herbe lors du déplacement. A noter que le lapin ne pourra évidemment pas sortir de la grille et il ne pourra pas non plus se déplacer vers une case contenant un autre animal, que ce soit un loup ou un lapin.

Loups. Les loups se nourrissent de lapins. Pour modéliser le fait qu'ils sont plus rapides que les lapins, ils peuvent eux se déplacer de zéro, une ou deux cases à chaque pas de temps, également sans déplacement oblique [3]. 13 cases leur sont donc potentiellement accessibles à partir de leur position initiale. Le score $S(i, j)$ associé à chacune des cases sera cette fois :

$$\begin{cases} S(i,j) = -d_r(i,j) & \text{Si } (i,j) \text{ est vide, contient de l'herbe, un lapin ou le loup,} \\ S(i,j) = \texttt{INT\_MIN} & \text{Si } (i,j) \text{ contient un autre loup,} \end{cases}$$

où $d_r(i, j)$ est la distance de Manhattan au lapin le plus proche de la position (i, j), avec une valeur minimale de 0 et une valeur maximale de $d_{max} + 1$ si aucun lapin n'est vu. Le loup se déplacera vers la case de score maximal parmi celle qui ne contiennent pas déjà un loup. Si la case de score maximum contient un lapin, le loup le mangera. Si cette case contient de l'herbe, elle sera écrasée par le déplacement du loup. Le loup aura une valeur de priorité plus faible que le lapin par défaut. Les remarques concernant le calcul de la distance et le choix de la case faites pour le lapin valent également pour le loup.

1. `INT_MIN` est le plus petit entier représentable dans un `int`, et est défini dans `limits.h`.
2. La distance de manhattan entre deux position (i, j) et (i', j') est définie par $|i - i'| + |j - j'|$.
3. Un loup peut atteindre une case oblique, mais en se déplaçant de deux cases. Par exemple, une vers le haut et une vers la gauche.

Herbes. Lorsque tous les animaux ont fait leur action, la simulation se charge de faire apparaître de l'herbe aléatoirement sur la grille. Sur chaque case vide (i, j), une nouvelle herbe a une probabilité p_h d'apparaître. Cette probabilité est calculée comme suit :

$$p_h = p_{h,init} + K \times p_{h,inc},$$

où $p_{h,init}$ est la probabilité initiale d'avoir de l'herbe, $p_{h,inc}$ l'incrément de probabilité pour chaque case adjacente contenant de l'herbe et K le nombre de cases adjacentes contenant de l'herbe (min 0, max 4).

2 Implémentation

L'implémentation du simulateur est basée sur plusieurs modules principaux (le premier fourni et les quatre autres devant être complétés par vos soins) :

- `animal.c/.h` implémentant un animal.
- `rabbit.c/.h` et `wolf.c/.h` implémentant une animal respectivement de type lapin et loup.
- `grid.c/.h` implémentant la grille.
- `simulation.c/.h` implémentant une simulation.

Nous vous fournissons également les fichiers suivants :

- `main.c` vous permettant de tester votre code (voir plus bas)
- `constants.c/constants.h` contenant la définition des différents constantes mentionnées plus haut régulant les comportements du lapin et du loup, ainsi que l'apparition de l'herbe.
- `position.h` et `action.h` deux fichiers contenant des définitions de types pour une position, un mouvement et une action (voir plus bas).

Ces différents fichiers et ce qu'ils doivent contenir sont décrits brièvement ci-dessous. Les différentes fonctions à implémenter ne sont pas toutes détaillées ici. Nous vous renvoyons aux définitions contenues dans les fichiers d'entête pour les détails.

2.1 Fichiers `animal.c` et `animal.h`

Ces fichiers qui vous sont fournis définissent le type opaque `Animal` qui sera utilisé pour représenter à la fois un lapin et un loup. La structure `Animal_t` contient les champs suivants :

- `energy` : l'énergie de l'animal
- `name` : le type de l'animal (une chaîne de caractère)
- `priority` : sa priorité
- `actionEnergy` : le coût en énergie d'une action
- `eatEnergy` : l'énergie gagné lorsque l'animal se nourrit
- `findAction` : un pointeur vers une fonction permettant à l'animal de déterminer l'action à effectuer

— `reproduce` : un pointeur vers une fonction permettant à l'animal de se reproduire

Les valeurs initiales de ces différents champs devront être fournies en arguments de la fonction `animalCreate` permettant de créer un nouvel animal. La fonction `findAction` a la signature suivante :

```
Action findAction(Animal *animal, Grid *, Position)
```

Étant donné l'animal en argument, la grille actuelle et sa position dans la grille, cette fonction doit renvoyer l'action effectuée par l'animal. La structure Action (définie dans `action.h`) contient un champ de type `Move` indiquant l'incrément des coordonnées actuelle l'amenant à sa nouvelle position et un booléen indiquant si l'animal se nourrit de l'animal ou l'herbe présente sur la nouvelle case atteinte.

La fonction `reproduce` a la signature suivante :

```
Animal *(*reproduce)(Animal *)
```

Elle doit renvoyer un nouvel animal du même type que l'animal en argument si ce dernier est en mesure de se reproduire (son énergie est plus grande ou égale à son seuil de reproduction), NULL si ce n'est pas le cas. Dans le cas d'une reproduction, l'énergie de l'animal en argument doit être diminuée du coût de la reproduction.

2.2 Fichiers `rabbit.c` et `wolf.c`

Ces fichiers ne doivent fournir chacun qu'une seule fonction permettant de créer l'animal correspondant (`rabbitCreate` et `wolfCreate`). Ces deux fonctions peuvent se réduire à un appel à `animalCreate` avec les bons arguments. Le fichier devra néanmoins contenir la définition des fonctions `findAction` et `reproduce` à passer en argument à `animalCreate` qui devront suivre les règles spécifiques mentionnées ci-dessus pour les deux types d'animaux. Les différents paramètres utilisés pour définir le comportement des animaux (énergie initiale, coût d'une action, seuil et coût de la reproduction, etc.) sont définies dans le fichier `constant.h`.

2.3 Fichiers `grid.c`, `grid.h` et `Position.h`

Ces deux fichiers définissent le type opaque `Grid` utilisé pour représenter une grille et les fonctions associées. La grille permet de stocker les animaux, de les retrouver, de les déplacer et de les tuer. La plupart de ces fonctions prennent comme argument une position dans la grille qui est représentée par le type `Position` défini de manière non opaque dans le fichier `Position.h`. Pour faciliter l'implémentation du calcul des scores pour chacun des animaux, vous devrez en particulier implémenter les deux fonctions suivantes dans ce fichier :

```
int gridFindClosestAnimal(Grid *grid, Position pos, int maxDistance, const char *name)
int gridFindClosestGrass(Grid *grid, Position pos, int maxDistance)
```

renvoyant la distance (de Manhattan) minimale entre la position `pos` et un animal de nom

`name` (respectivement de l'herbe), plafonnée à `maxDistance`. Si aucun animal (resp. aucune herbe) n'est trouvé, alors `maxDistance + 1` est retourné. Ces fonctions sont notamment utiles lors des calculs des scores.

2.4 Fichiers `simulation.c` et `simulation.h`

Ces deux fichiers définissent le type opaque `Simulation` utilisé pour gérer une simulation et les fonctions associées. Les fonctions à implémenter permettent de créer et libérer une simulation (`simulationCreate` et `simulationFree`), d'ajouter des animaux et de l'herbe dans la grille associée (`simulationAddGrass` et `simulationAddAnimal`), de compter le nombre de cellule d'un certain type au pas de temps courant (`simulationCountAnimals` et `simulationCountGrass`) et de faire avancet la simulation (`simulationStep`) d'un pas de temps.

2.5 Fichiers `main.c`

Un fichier `main.c` vous est fourni pour tester votre implémentation. Via le fichier `Makefile` fourni également, ce fichier `main.c` crée un exécutable `simulate` qui peut être utilisé pour effectuer une simulation. La commande suivante :

```
./simulate -s N -t T -w PW -r PR -g PG -f F
```

effectuera une simulation sur T pas de temps où :

- la grille a une taille de $\texttt{N} \times \texttt{N}$;
- la probabilité qu'une case soit initialement occupée par un loup sera `PW`
- la probabilité qu'une case soit initialement occupée par un lapin sera `PR`
- la probabilité qu'une case soit initialement occupée par de l'herbe sera `PG`
- la probabilité qu'une case soit initialement vide sera en conséquence $(1-\texttt{PW}-\texttt{PR}-\texttt{PG})$.

Cette commande créera un fichier nommé `F` contenant le gif animé généré. L'exécution de la commande `./simulate` sans argument est équivalent à l'appel suivant (qui ne génère pas de GIF) :

```
./simulate -s 100 -t 100 -w 0.003 -r 0.03 -g 0.1
```

Les fichiers `gifenc.c`, `gifenc.h`, `simulation_gif.c` et `simulation_gif.h` servent à générer le gif et ne doivent pas être modifiés.

3 Conseils d'implémentation

La difficulté principale du projet est de comprendre l'organisation du code en différents modules. La modularité fait que vous pouvez en principe implémenter les fichiers indépen-

damment les uns des autres.

Avant d'implémenter `rabbit.c` et `wolf.c`, lisez le code de `animal.c` et assurez vous que vous avez compris l'utilisation des pointeurs de fonction dans la structure. Pour implémenter ces deux fichiers, vous aurez besoin des constantes définies dans `constants.h` et des fonctions de `grid.h`. L'implémentation des règles de mouvement des animaux ne devrait pas vous poser de problème si vous utilisez les fonctions `gridFindClosestGrass` et `gridFindClosestAnimal`.

Pour implémenter `grid.c`, vous devrez choisir une manière d'implémenter la structure `Grid_t`. Les fonctions sans doute les plus compliquées du projet d'un point de vue algorithmique sont les fonctions `gridFindClosestGrass` et `gridFindClosestAnimal`. Pour implémenter efficacement ces dernières, il faut prendre autant que possible en compte l'argument `maxDistance` qui évitera de devoir parcourir l'entièreté de la grille.

L'implémentation de `simulation.c` demande également de choisir une structure associée qui devrait au minimum contenir un champ pour une grille. La fonction `simulationStep` est sans doute la plus longue mais ne devrait pas vous posez trop de problème. Comme indiqué plus haut, vous devrez trouver un mécanisme pour éviter de mettre en action deux fois un même animal et de ne pas non plus activer les nouveaux né.

4 Soumission

Vous devez soumettre (uniquement) les fichiers suivants sur Gradescope :

- `wolf.c`, `rabbit.c`, `grid.c`, et `simulation.c`.
- `rapport.txt`
- `figure.pdf`

Dans `rapport.txt` vous sont demandés les contributions de chacun au projet. Par la soumission de ce fichier vous attestez également avoir respecté le règlement en matière de plagiat du cours. Le fichier `figure.pdf` devra être une figure représentant l'évolution du nombre de lapins, de loups et d'herbe au cours de 1000 itérations obtenus en appelant `./simulate -t 5000`. [4]

Bon travail !

4. Pour récupérer les résultats affichés dans le terminal dans un fichier par exemple nommé `result.txt`, vous pouvez faire `./simulate -t 5000 > results.txt`.