Projet algorithmique et programmation — Proies et prédateurs

Paul Gaborit

IFIE1 – Octobre 2023 – IMT Mines Albi

Table des matières

1 Consignes

Ce travail s'effectue en binôme. Le rendu se fera sur Campus sous la forme d'une archive (`.zip`, `.rar`, `.7z`, `.tgz`...) **portant les deux noms du binôme** (un seul dépôt par binôme).

Contenu attendu dans l'archive :

1. un document (en **PDF**) contenant :
   a. les noms et prénoms des membres du binôme,
   b. une analyse des problèmes auxquels vous avez été confrontés,
   c. les algorithmes mis en œuvre,
   d. un descriptif de ce que vous avez programmé expliquant les choix que vous avez faits,
2. votre script `pp.py` et, éventuellement, d'autres scripts et modules (tous les fichiers `.py` que vous aurez écrits vous-même),

Les éléments qui **ne doivent pas être** dans cette archive sont : votre environnement virtuel (le répertoire `venv-pp`) et d'éventuels répertoires `__pycache__`.

2 Objectif

L'objectif de ce mini-projet est de simuler l'évolution d'une population de prédateurs et d'une population de proies (la première se nourrissant de la seconde).

Dans la suite de ce projet, nous prendrons comme exemple des renards et des lapins mais nous aurions pu choisir d'autres couples : requins et dorades, boas et petits vertébrés, buses et mulots, vampires et humains, etc.

3 La simulation

Dans notre simulation, nous considérons un espace fermé (représentant par exemple une île) dans lequel évolueront nos populations. Cet espace rectangulaire est découpé (ou *discrétisé*) par un quadrillage délimitant de petites zones carrées (les zones élémentaires).

Une zone élémentaire est soit vide, soit occupée uniquement par des lapins, soit occupée uniquement par des renards. On considère les lapins ou les renards qui occupent une zone comme une entité unique que nous appellerons par la suite *un lapin* ou *un renard* (mais représentant en fait un petit groupe de lapins ou de renards).

Le temps est divisé en période. Pour notre exemple, une période dure une semaine.

Voici les règles qui régissent le comportement de nos populations dans notre espace :

— Un renard comme un lapin peut mourir par d'autres causes que celles considérées dans notre

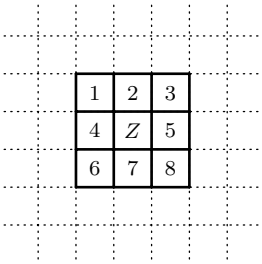

Figure 1 – *Les 8 zones adjacentes à la zone Z.*

simulation (ne serait-ce que de vieillesse). Donc au début de chaque période, un renard ou un lapin a une certaine probabilité de mourir.

— À chaque période, un lapin d'une zone (ayant consommé les ressources de cette zone) doit trouver une zone adjacente vide pour s'y déplacer. Comme le lapin se reproduit, il a une certaine probabilité de laisser sur place un rejeton qui deviendra actif à la période suivante. Si aucune zone adjacente est vide, le lapin de la zone ne peut se déplacer et meurt.

— À chaque période, un renard d'une zone chasse : il choisit une zone adjacente contenant un lapin, s'y déplace et dévore le lapin présent. Si la chasse est infructueuse (aucune zone adjacente ne contient de lapin), le renard se déplace vers une zone adjacente vide sans manger (si un renard ne mange pas pendant un certain nombre de périodes, il meurt). Tout comme le lapin, le renard a une certaine probabilité de laisser sur place un rejeton qui deviendra actif à la période suivante. Si la chasse est infructueuse et si aucune zone adjacente est vide (donc si toutes les zones adjacentes contiennent un renard), le renard ne peut pas se déplacer et meurt.

En pratique, à chaque période, on considère chacun des renards et des lapins présents dans l'espace au début de la période (dans un ordre arbitraire et un par un) et on lui applique toutes les règles ci-dessus. Conséquences, dans une même période :

— un lapin peut se déplacer puis se faire dévorer ;

— un lapin peut se faire dévorer avant même d'avoir pu se déplacer ;

— un rejeton apparu après le déplacement d'un lapin peut se faire dévorer avant même de devenir actif ;

3.1 Les zones adjacentes

Notre espace est découpé en zones carrées. La figure 1 représente les 8 zones adjacentes d'une zone. Évidemment, lorsqu'on considère un zone au bord de l'espace de simulation, certaines des 8 zones adjacentes n'existent pas.

3.2 Probabilités et autre paramètres généraux

Un lapin sauvage a une espérance de vie de 8 ans (sans chasse ni prédation). Ce qui donne une probabilité de survie par semaine de 0,9976 ($\approx 1 - \frac{1}{52\times 8}$) [1].

Un couple de lapins sauvages a environ 4 portées de 5 petits par an. La population est donc multipliée par 11 chaque année. Ce qui donne une probabilité de reproduction de 0,04719 chaque semaine ($0{,}04719 \approx 11^{(1/52)} - 1$).

Un renard sauvage a une espérance de vie de 6 ans. Ce qui donne une probabilité de survie par semaine de 0,99679 ($\approx 1 - \frac{1}{6\times 52}$).

Un couple de renards sauvages a environ 5 petits par an. La population est donc multipliée par 3,5 chaque année. Ce qui donne une probabilité de reproduction de 0,02438 chaque semaine ($0{,}02438 \approx 3.5^{(1/52)} - 1$).

Un renard peut survivre sans manger au maximum pendant 8 semaines.

Dans un premier temps, nous considérerons un espace constitué de 100×100 zones élémentaires. Ces zones seront peuplées de 3500 lapins et 130 renards (venant juste de manger) dont les positions initiales seront choisies au hasard.

1. Si $E = \sum_{k=1}^{\infty} p^k$ alors $p = 1 - \frac{1}{E}$

Table 1 – *Synthèse des probabilités et autres paramètres généraux*

Espace	
Nombre de zones	100×100
Lapins	
Population initiale	3500
Proba. de survie	0,9976
Proba. de reproduction	0,04719
Renards	
Population initiale	130
Proba. de survie	0,99679
Proba. de reproduction	0,02438
Durée maximum de jeûne	8 semaines

Note : les probabilités sont données à la semaine

4 Éléments techniques

4.1 Le hasard, le module `random` et le module `time`

De nombreux choix (la position initiale des renards et des lapins, la zone adjacente choisie pour se déplacer si plusieurs sont possibles, la mort ou la naissance, etc.) sont effectués par tirage de nombres aléatoires. Ces nombres seront produits via le module `random`. Voici quelques fonctions qui pourraient vous servir :

- `random.seed(seed)` – permet de choisir la graine définissant la suite pseudo-aléatoire utilisée (à n'appeler qu'une seule fois en début de script). Le paramètre `seed` peut être un nombre ou une chaîne de caractères. Pour utiliser une graine différente à chaque exécution, vous pouvez prendre le résultat de la fonction `time.time_ms()` du module `time`.
- `random.random()` – retourne un nombre flottant aléatoire dans l'intervalle $[0, 1[$.
- `random.choice(list)` – retourne un élément choisi aléatoirement parmi les éléments de `list`.

4.2 Représentation de l'espace à l'écran via le module `grid_display`

Pour représenter notre espace à l'écran, vous devez utiliser la classe `GridDisplay` du module `grid_display` :

```
from grid_display import GridDisplay
```

📎 le module Python `grid_display.py` est une pièce-jointe de ce fichier PDF.

4.2.1 L'environnement virtuel et `pygame`

L'utilisation du module `grid_display` nécessite de créer un environnement virtuel dans lequel est installé le package externe `pygame` :

```
%% python3 -m venv venv-pp
%% source ./venv-pp/bin/activate
(venv-pp) %% pip install pygame
```

```
C:> python -m venv venv-pp
C:> .\venv-pp\Scripts\activate.bat
(venv-pp) C:> pip install pygame
```

4.2.2 Création et affichage de la grille

Pour créer une fenêtre d'affichage d'une grille, il suffit de créer un objet de la classe `GridDisplay`.

Les paramètres de création d'un objet `GridDisplay` sont :

- `size` : paramètre obligatoire, sous la forme d'un tuple de deux entiers indiquant la taille de la grille à afficher (largeur et hauteur). Ex : `(10, 30)`.
- `nb_pixels_by_box` (par défaut `4`) : la taille en pixels de chaque case.
- `period_duration` (par défaut `100`) : la durée en millisecondes de chaque période. Permet de ralentir la vitesse de simulation (si `0`, il n'y pas d'attente).
- `grid_color` (par défaut `(220,220,220)`, un gris clair) : la couleur des bordures des cases.
- `bg_color` (par défaut `(255,255,255)`, du blanc) : la couleur de fond de la fenêtre.
- `text_color` (par défaut `(0,0,0)`, du noir) : la couleur du texte (pour afficher le numéro de période).

4.2.3 Remplissage des cases

La méthode `draw_box(pos, color)` permet de remplir la case dont les coordonnées sont `pos` (un couple) avec la couleur `color` (un triplet de valeurs RVB).

L'appel suivant remplit en magenta la case sur la 4e colonne et la 5e ligne :

```
gd.draw_box((3, 4), (255, 0, 255))
```

Note 1 : la case `(0,0)` est en haut à gauche de la grille.

Note 2 : en fait, le remplissage effectif n'aura réellement lieu qu'à l'appel de la méthode `next_period()`.

4.2.4 Période suivante et fin du script

La méthode `next_period()` :

- affiche l'état de la grille (ce n'est qu'à ce moment-là que sont réellement affichées toutes les cases colorées par les appels à `draw_box()` ainsi que le numéro de période ;
- retourne `False` si le script doit s'arrêter ;
- attend la durée fixée pour une période ;
- prépare un nouvelle grille vide (prête pour la période suivante).

Cette méthode accepte l'argument optionnel `text` afin d'afficher un texte supplémentaire (inséré après l'affichage du numéro de période).

Les événements indiquant la fin du script sont la fermeture de la fenêtre, un clic souris dans la fenêtre ou bien l'appui sur `Ctrl-Q` ou `Ctrl-C`.

4.3 Conversion de chaînes de caractères en valeurs utiles via le module `convert`

Pour convertir les arguments de la ligne de commande en valeurs utiles, nous vous fournissons le module `convert.py`.

Utilisation sans argument :

```
(venv-pp) %% python3 exemple.py
```

```
(venv-pp) C:> python exemple.py
```

Résultat obtenu :

```
Erreur -- nombre d'arguments incorrect
Usage: ./exemple.py seed grid_size nb_balls nb_black_holes p_dup p_change_direction duration

                seed: (un mot) graine pour le générateur aléatoire
           grid_size: (sous la forme 19x30) taille de la grille
            nb_balls: nombre de balles
      nb_black_holes: nombre de trous noirs
               p_dup: probabilité de duplication d'une balle
  p_change_direction: probabilité de changement de direction d'une balle
            duration: durée d'affichage (en ms) d'une période

Simule des balles qui rebondissent dans un espace discrétisé.
```

Utilisation avec arguments :

```
(venv-pp) %% python3 exemple.py 0 100x50 50 3 0.001 0.01 100
```

```
(venv-pp) C:> python exemple.py 0 100x50 50 3 0.001 0.01 100
```

Figure 2 – *Utilisation du script* `exemple.py` *sans ou avec arguments*

📎 le module Python `convert.py` est une pièce-jointe de ce fichier PDF.

Ce module fournit des fonctions permettant de convertir une chaîne de caractère en un type spécifique de valeur. Si la conversion réussit, la valeur décodée est retournée. En cas d'échec, c'est `None` qui est retourné.

- `str_to_positive_int(str)` retourne un entier positif (donc sans signe). Exemples de chaînes acceptées : `0`, `1000000`
- `str_to_float(str)` retourne une valeur réelle (un *flottant*). Exemples de chaînes acceptées : `1`, `+2.3e12`, `-.4`
- `str_to_definition(str)` retourne une définition : un couple (*largeur*, *hauteur*). Exemples de chaînes acceptées : `1x1`, `1920x1080`
- `str_to_color(str)` retourne une couleur sous la forme d'un triplet d'entiers RGB (dans l'intervalle $[0, 255]$). Exemples de chaînes acceptées : `0,0,0`, `255,255,255`, `123,0,250`

4.4 Exemple d'utilisation des modules `grid_display` et `convert`

Le script `exemple.py` sert de démonstration d'utilisation des modules `grid_display` et `convert`.

📎 le fichier Python `exemple.py` est une pièce-jointe de ce fichier PDF.

La figure 2 montre des exemples d'appel à ce script.

Vous pouvez recopier tout ou partie du code `exemple.py` pour l'adapter à vos besoins. Vous devez vous inspirer des bonnes pratiques et astuces mises en œuvre dans ce script... dont :

- Une fonction principale `main()` qui gère toute la logique du script.
- Une fonction `decode_arguments()` qui analyse les arguments reçus via la ligne de commande et qui fait appel à la fonction `usage()` en cas d'erreur.
- Une fonction `usage()` qui rappelle à l'utilisateur du script comment appeler ce script depuis la ligne de commande.
- La boucle générale qui, à chaque période, effectue les actions de déplacements puis redessine tous les éléments nécessaires.
- L'espace de simulation et ses éléments codés dans les classes `Space`, `Ball` et `BlackHole`.
- Aucune utilisation de variables globales.
- Tous les noms de variables, de fonctions et de méthodes en `snake_case`, les noms de classes en `CamelCase` et les noms des constantes en `UPPERCASE` (et tous ces noms en anglais, éventuellement abrégé).
- Un dictionnaire associant une position (la clé) à un objet afin de gérer efficacement les objets présents dans l'espace de simulation.

5 Ce qu'il faut produire

Votre script Python s'appellera `pp.py` (`pp` pour *proies-prédateurs* ou *prey-predators*).

Il recevra via la ligne de commande les 5 paramètres principaux de la simulation. Dans l'ordre : la graine, la taille de l'espace, le nombre initial de renards, le nombre initial de lapins, la durée d'affichage (en millisecondes) d'une période.

Ne modifiez pas le fichier `grid_display.py` et n'utilisez pas d'autres modules que ceux proposés dans l'énoncé !

Il vaut mieux écrire de nombreuses méthodes ou fonctions courtes que peu de fonctions et méthodes très longues (application du précepte « *diviser pour régner* »).

Mises à part d'éventuelles constantes (dont, par convention, la valeur n'est jamais modifiée), vous ne devez pas utiliser de variables globales.

Si vous reprenez tout ou partie du code du script `exemple.py`, n'oubliez pas d'adapter tous les messages et commentaires !

6 Pour aller plus loin...

Pour ceux qui souhaitent aller plus loin, vous pourrez proposer un ou plusieurs autres scripts (dérivés de `pp.py`) en ajoutant une ou plusieurs fonctionnalités au script initial.

Voici des idées de fonctionnalités supplémentaires :

1. prendre en compte des arguments supplémentaires sur la ligne de commandes (les probabilités, les couleurs...).
2. fixer pour chaque espèce un âge maximum.
3. fixer pour chaque espèce un nombre minimum de périodes séparant deux reproductions.
4. élargir la distance à laquelle les renards détectent les lapins (pour ne pas choisir une zone adjacente libre totalement au hasard si aucun lapin n'est présent dans une zone adjacente).
5. augmenter la vitesse de déplacement des renards.
6. faire en sorte qu'une zone venant d'être occupée par un lapin devienne temporairement inoccupable par un autre lapin (le temps que les ressources se reconstituent).
7. collecter le nombre de lapins et de renards présents dans l'espace de simulation à chaque génération pour stocker toutes ces valeurs dans un fichier afin de pouvoir tracer une courbe d'évolution des populations en fonction du temps ou, mieux, dans l'espace des phases.
8. ajouter un test d'arrêt pour que la simulation s'arrête automatiquement.

7 Évaluation

Pour chacune des fonctions ou méthodes que vous aurez codées vous-même, vous devrez fournir l'algorithme associé.

Pour les schémas bulle, les objets reçus en paramètre et modifiés durant l'exécution des instructions de l'algorithme devront apparaître dans la partie gauche et dans la partie basse du schéma bulle.

Ne dupliquez pas votre code Python dans votre rapport (il sera dans votre archive).

Voici des éléments d'appréciation qui nous serviront pour évaluer votre travail.

Concernant le rapport :

- la correction de vos schémas bulle,
- le respect des notations algorithmiques (telles que définies dans les supports de cours et de TP),
- la logique et la qualité de votre analyse et de vos explications,
- la logique et la clarté de vos algorithmes,
- le respect des spécifications de cet énoncé,

Concernant le code Python :

- le respect des spécifications de cet énoncé,
- le respect des bonnes pratiques de Python,
- la qualité des commentaires ou des chaînes de documentation,
- la conformité du code Python avec les algorithmes,
- la pertinence du nommage des variables, des fonctions, des méthodes, des classes...

Nous prendrons aussi en compte les variantes et autres améliorations que vous proposerez (si vous en proposez) ainsi que les analyses ou explications liées aux expérimentations réalisées.