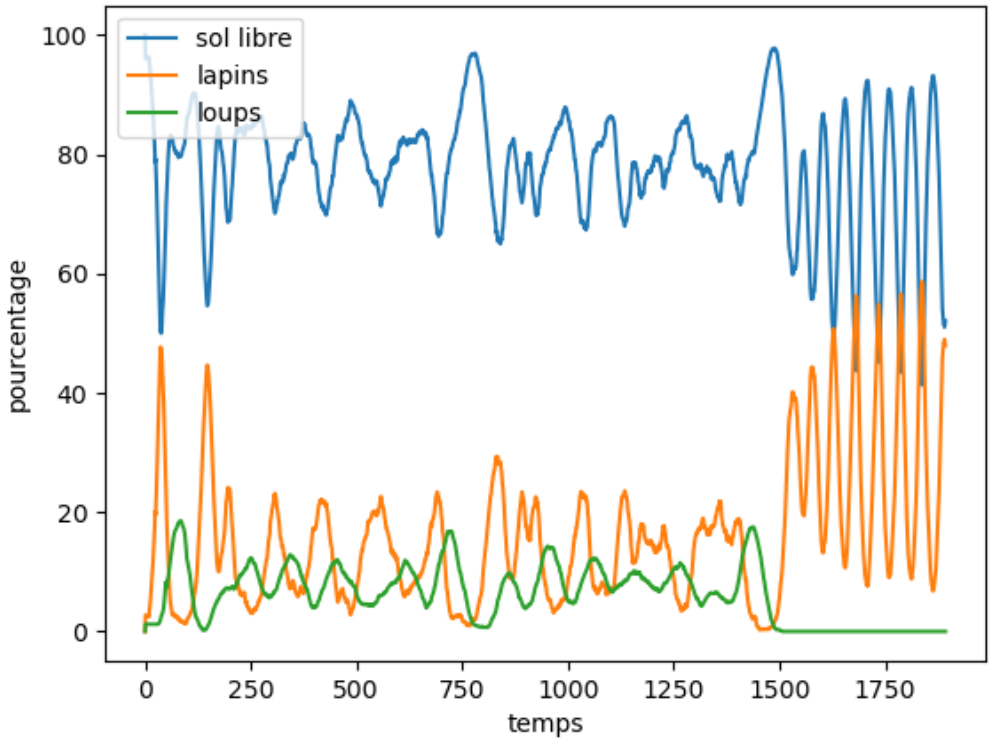
Mini-projet : Un modèle d'écosystème proie-prédateur

I. Description

On observe dans la nature que les populations des proies et des prédateurs ne se stabilisent pas autour d'un certain point d'équilibre, mais évoluent au cours du temps de manière cyclique : trop de prédateurs finissent par tuer trop de proies, ce qui cause le déclin des prédateurs par manque de nourriture. Ce déclin permet à la population de proies de croître à nouveau, causant à son tour l'augmentation du nombre de prédateurs, et ainsi de suite.
Simuler un tel écosystème peut se faire sur une grille de cases similaire à celle du Jeu de la Vie, mais avec des cellules et des règles plus complexes. On considère ici des lapins (proies) et des loups (prédateurs). Les règles suivantes sont issues du simulateur en ligne `http://www.shodor.org/interactivate/activities/RabbitsAndWolves/` :

- Chaque cellule contient de l'herbe et au plus un animal (lapin ou loup).
- L'herbe pousse d'une unité de nourriture (UN) par unité de temps (UT), jusqu'à une valeur maximale de 100 UN.
- Au départ, chaque cellule a 20 UN d'herbe et les lapins et les loups sont répartis aléatoirement (leur quantité est laissée au choix et peut être changée).
- Chaque animal peut se déplacer d'une cellule dans l'une des quatre directions à chaque UT.
- Les lapins se nourrissent d'herbe et les loups se nourrissent de lapins.
- Un loup mange un lapin (pour une valeur de 10 UN) s'il est dans son voisinage.
- Le métabolisme de chaque animal consomme 2 UN par UT pour les loups et 3 UN/UT pour les lapins.
- Un animal meurt s'il est trop vieux (25 UT pour les lapins, 50 UT pour les loups) ou si son métabolisme devient négatif.
- Ce métabolisme ne peut pas dépasser une certaine valeur (45 UN pour les lapins, 200 UN pour les loups).
- Un animal se reproduit avec une probabilité de 50% s'il est suffisamment âgé (10 UT) et suffisamment nourri (40 UN pour un lapin, 120 UN pour loup).
- Un lapin ne peut pas se reproduire si un loup est dans son voisinage.

Le code initial fourni est en deux fichiers Python. Le fichier `Carte.py` gère la classe `Carte` qui hérite de la classe `Canvas` de `tkinter`. Elle permet l'affichage mais aussi la gestion des données du programme. La variable `carte` qui en est l'instance dans le fichier `ProiesPredateurs.py` est l'entrée pour toutes les manipulations et obtentions de valeurs, les paramètres y étant inclus.
Le fichier `ProiesPredateurs.py` est le fichier principal, celui qui sera à compléter et modifier.
Le but est de gérer les cycles en définissant toutes les fonctions qui s'y exécutent, et également de produire un graphique montrant l'évolution des proportions de chaque espèce.
Un exemple d'un tel graphique est montré ci-après :



II. Instructions

- Lire dans un premier temps le code des fichiers `ProiesPredateurs.py` et `Carte.py`, en particulier les commentaires, pour comprendre la structure et s'informer des différente fonctions et classes (avec leurs attributs et méthodes respectifs) définies, ainsi que leur utilisation.
  Une fonction `exemples()` fournit en particulier certaines illustrations des possibilités (mais pas forcément toutes).
- Compléter le code du fichier `ProiesPredateurs.py` aux endroits où se trouve l'instruction `pass` afin de faire fonctionner totalement le programme.
  Le fichier `Carte.py` n'est *a priori* pas à modifier ; en tout cas ce n'est pas nécessaire pour effectuer le travail demandé.
- Ajouter tout autre élément supplémentaire selon l'envie et les possibilités.
  Par exemple :
  * Permettre de configurer davantage de paramètres ;
  * Ajouter la possibilité d'afficher des statistiques : nombre total de lapins et de loup depuis le début, nombre de vivants de chaque espèce, nombre de morts, classifiées par type (de faim, de vieillesse, mangé), etc.
  * Modifier, voire permettre de modifier, des règles ;
  * Permettre d'ajouter un loup ou un lapin dans le terrain par un clic.
    Pour cela, il faut gérer le clic avec un `bind` (faire des recherches sur le `bind` en `tkinter`), et définir une fonction qui, aux coordonnées du curseur lors du clic, associe celles de la case sur laquelle il se trouve.
  * Ajouter d'autres animaux avec leurs propres règles ;
  * Améliorer éventuellement la structure, qui n'est pas parfaite ;
  * Toute autre idée d'ajout ou d'amélioration est bienvenue.

III. Indications supplémentaires

- Il est recommandé de concevoir le programme de façon progressive et de tester régulièrement.
- Dans les fonctions `vieillir_animaux()`, `nourrir_animaux()` et `reproduire_animaux()`, gérer une liste des morts ou des nouveaux nés lors du parcourt des animaux.
  On ne peut pas modifier le contenu des animaux (qui est un dictionnaire) pendant son parcourt.
  C'est donc une fois le parcourt effectué que l'on supprime ou ajoute de nouveaux éléments.
  Il faudra bien entendu veiller à ne jamais placer deux animaux sur une même case (par déplacement ou par naissance). Des assertions dans le code du fichier `Carte.py` sont là pour s'en assurer.
  Pour la fonction `deplacer_animaux()`, on travaille sur une copie, donc on peut déplacer immédiatement s'il y a la place.
- Pour le graphique, il faudra effectuer des recherches sur `matplotlib` pour savoir comment le tracer.
  Effacer le graphique à chaque mise à jour et le retracer est possible.
  On pourra étudier la possibilité de simplement ajouter des données au graphique pour le mettre à jour.