The Circle Of Life

Simulation concurrente d’un écosystème proies–prédateurs

Rapport de projet – Programmation Parallèle et Concurrente (PPC)

Titre du projet
Simulation concurrente d’un écosystème proies–prédateurs avec python

Étudiants
LEVAL Enzo & MARTIN Ugo

Année universitaire
2025–2025

Encadrants
Razmig Kéchichian, Hugo Reymond & Areski Himeur

1. Introduction

L'objectif de ce projet est de concevoir une simulation d'écosystème simple mettant en interaction plusieurs entités concurrentes : un environnement, des proies, des prédateurs et une interface graphique.
Chaque entité évolue de manière autonome tout en interagissant avec les autres à travers des mécanismes de communication et de synchronisation.

Le projet vise à mettre en œuvre des concepts fondamentaux de la programmation parallèle et concurrente, tels que la gestion de processus, la synchronisation d'accès à des ressources partagées et la communication inter-processus tout en reproduisant un modèle biologique.

Ce projet a été conçu dans une volonté de rapprocher les concepts de la programmation parallèle et concurrente d'un fonctionnement observé dans la nature. L'écosystème simulé repose sur un environnement central, capable de faire émerger des entités vivantes (proies et prédateurs), puis de les laisser évoluer de manière autonome. Une fois créées, ces entités ne sont plus pilotées directement par l'environnement : elles cherchent à se nourrir, interagissent avec leur milieu et disparaissent de manière indépendante, en fonction de leur propre état interne et des ressources disponibles.

L'environnement joue ainsi un rôle analogue à celui de la nature : il maintient un état global cohérent (ressources, conditions climatiques, population), tandis que chaque entité vivante agit localement en consultant cet état pour prendre ses décisions. Le travail réalisé consiste donc à établir une analogie aussi élégante que possible entre les mécanismes de la programmation parallèle et concurrente et ceux de la "dame nature", en mettant en évidence l'autonomie des entités, la concurrence pour les ressources et le caractère non déterministe de l'évolution du système.

2. Choix techniques

2.1 Modèle de concurrence

Nous avions à notre disposition plusieurs approches pour modéliser le fonctionnement de l'écosystème. ex : une gestion centralisée des comportements, une autonomie complète des entités une fois créées …
En considérant l'objectif d'une modélisation proche d'un environnement réel, notre choix s'est porté sur l'implémentation d'un environnement capable de créer des entités indépendantes et soucieuses de leur existence.Notre environnement joue alors un rôle de créateur et de régulateur global.

L'environnement est chargé de maintenir l'état global du système, notamment les ressources disponibles et les conditions générales, alors que chaque proie ou prédateur, une fois créé, agit de façon autonome en interrogeant cet environnement pour obtenir les informations nécessaires à sa survie.

Naturellement, les entités vivantes ont été conçues afin de refléter :
- une prise de décision locale et non centralisée
- une concurrence naturelle pour l'accès aux ressources
- une évolution non déterministe du système.

L'utilisateur quant à lui peut essayer d'interférer la simulation à la lumière d'une puissance divine en ajoutant de nouvelles entités ou en altérant le climat.

2.2 Représentation des entités

Chaque proie est modélisée comme un processus autonome, disposant de son propre état interne, notamment un niveau d'énergie qui diminue au cours du temps. Cette autonomie permet à chaque proie de prendre des décisions indépendantes en fonction de sa situation et des ressources disponibles dans l'environnement pour survivre, manger de l'herbe dans son cas.

De même les prédateurs sont également représentés par des processus autonomes à la seule différence qu'il possède la capacité de terminer (tuer) un processus proie pour se nourrir.

L'environnement, quant à lui, est centralisé et héberge l'ensemble de l'état global de l'écosystème. Il assure la cohérence des ressources partagées et fournit une interface contrôlée permettant aux entités de consulter ou de modifier cet état. L'ensemble de ses mécanismes est contrôlé par des threads permettant de garder une mémoire commune.

2.3 Mécanismes de communication

L'un des points clés de ce projet réside dans le fait que l'environnement met à disposition une mémoire partagée représentant l'état global de l'écosystème à l'ensemble des entités qu'il crée. Afin d'assurer la cohérence des données et d'éviter les race conditions, l'accès à cette mémoire est strictement contrôlé par un BaseManager qui fournit un accès sécurisé et structuré.

De plus, nous avons mis en place un serveur TCP afin d'offrir un canal de communication distinct entre l'environnement et ses entités pour des informations ponctuelles et asynchrones via leur PID.

L'interface graphique est alimentée par des messages queues, permettant de transmettre régulièrement des données mises à jour depuis l'environnement. Permettant la visualisation de la nature en temps réel.

Évidemment l'usage de signaux est automatique afin de terminer les processus de manière propre.

3. Architecture générale et Protocole d'échanges

3.1 Vue d'ensemble

L'architecture globale repose sur un environnement central et plusieurs entités autonomes. L'environnement joue le rôle d'orchestrateur et conserve l'état global, tandis que les proies et les prédateurs interagissent uniquement via des interfaces contrôlées.

Un schéma représentant les processus et les flux de communication permet de visualiser cette architecture.

3.2 Architecture des processus

L'environnement constitue le point d'entrée du système et est lancé en premier afin d'assurer l'initialisation correcte de l'écosystème. Il met en place la mémoire partagée qui représente l'état global du monde à l'aide du Basemanager et se rend détectable pour toutes ses futures créations à l'aide du serveur TCP.

Une fois la mémoire partagée prête et le serveur TCP prêt, l'environnement démarre ses threads internes chargés de simuler les phénomènes naturels, tels que le passage du temps, la croissance de l'herbe. L'utilisation de threads permet à ces mécanismes de s'exécuter de manière concurrente tout en partageant le même état global.

Ce n'est qu'après cette phase d'initialisation que l'environnement lance les processus correspondant aux proies et aux prédateurs grâce à subprocess. Chaque entité animale fonctionne de cette manière : elle s'enregistre à son démarrage auprès de l'environnement grâce à une socket TCP et communique son identité (son PID), reste en éveil jusqu'au moment où la fin se fait ressentir, puis se termine explicitement lorsque ses conditions de survie ne sont plus satisfaites.

Cette organisation hiérarchique garantit un démarrage maîtrisé du système, une séparation claire des responsabilités et une évolution cohérente de l'écosystème concurrent.

3.3 Architecture mémoire

La gestion de la mémoire au sein du système repose sur une séparation entre la mémoire locale des processus et la mémoire partagée de l'écosystème. Cette distinction permet de clarifier les responsabilités de chaque composant et de limiter les accès concurrents aux seules données réellement nécessaires.

La mémoire locale est propre à chaque entité vivante et contient des informations telles que le niveau d'énergie ou l'état interne de la proie ou du prédateur. Ces données n'ont pas vocation à être partagées, car elles ne concernent que le comportement individuel de l'entité. Par souci de simplicité et de lisibilité, cette mémoire a été laissée au niveau du processus lui-même. Ce choix facilite également une évolution future du modèle, notamment

l'introduction de comportements aléatoires ou de caractéristiques individuelles propres à chaque entité.

À l'inverse, la mémoire partagée regroupe les informations globales de l'écosystème, telles que la quantité d'herbe disponible, l'état de la sécheresse ou encore les listes des proies vivantes et mangeables. Ces données doivent être accessibles à l'ensemble des entités, car elles conditionnent leurs décisions et leurs interactions.

3.4. Protocoles d'échange

La communication entre une proie et l'environnement s'effectue principalement via la mémoire partagée exposée par l'environnement. Dès son démarrage, une proie s'enregistre auprès de l'environnement en ajoutant son identifiant de processus (PID) à une structure partagée. Permettant aux prédateurs de connaître par le biais de cette mémoire leur futurs repas.

Au cours de son cycle de vie, la proie de la même manière que le prédateur met à jour son état en fonction de son niveau d'énergie. Lorsqu'elle devient affaiblie, elle signale à l'environnement qu'elle est mangeable. Ainsi débute la recherche de consommation d'herbe qui lui permet de ne pas mourir d'épuisement. Enfin, lorsqu'une proie meurt, que ce soit par manque d'énergie ou par prédation, elle se désenregistre proprement de la mémoire partagée.

Les prédateurs interagissent indirectement avec les proies à travers l'environnement. Lorsqu'un prédateur a besoin de se nourrir, celui-ci va chercher quel est le PID d'une des proies mangeables, puis en fera son repas en la terminant avec un signal.

Ces protocoles intègrent également la gestion des erreurs. Un prédateur peut être confronté à des situations telles qu'une proie déjà terminée, un PID invalide ou une liste de proies vides. Ces cas sont explicitement pris en compte afin d'assurer la robustesse du système et d'éviter les blocages ou comportements indéterminés. L'ensemble de ses demandes sont traitées soit par des opérations atomiques protégées par des sémaphores mutexs soit par des canaux de communication sûrs qui garantissent une cohérence globale de l'environnement à l'aide des semaphore mutexs.

4. Algorithmes importants (pseudo-code) (cf Annexe B)

5. Implantation et tests

5.1 Implantation

Le développement du projet a été volontairement mené de manière progressive et incrémentale. Chaque étape a été validée avant de passer à la suivante, afin de garantir une compréhension complète des mécanismes mis en œuvre et de limiter l'introduction d'erreurs difficiles à comprendre dans un contexte concurrent avec autant de processus.

La première phase a consisté à mettre en place l'environnement central, responsable de l'hébergement de l'état global de l'écosystème. Cette étape fondatrice a permis de définir les structures de données partagées et les interfaces d'accès, sur lesquelles l'ensemble du système repose. Une fois l'environnement créé, la gestion du temps et de la croissance de l'herbe a été introduite. Cette phase a permis de simuler une évolution autonome du monde et de valider l'utilisation des threads et non pas des processus. Avant l'ajout d'autres entités, il était nécessaire de créer le serveur TCP dans un autre thread de sorte à pouvoir établir la connexion entre lui et les autres entités.

Une fois ces modalités remplies, l'ajout des processus proies a ensuite été réalisé de manière isolée. De sorte à ce qu'il fonctionne seul, puis nous l'avons légitimement connecté aux informations de son environnement. Cette étape a été essentielle pour tester la communication inter-processus et la cohérence de la mémoire partagée. Les prédateurs n'ont été intégrés qu'après la validation complète du comportement des proies et nous l'avons réalisé de la même manière d'abord isolé puis connecté.

Enfin, une phase de synchronisation et de corrections a permis d'identifier et de résoudre les problèmes liés aux accès concurrents, aux conditions de course et à la gestion du cycle de vie des processus. Ainsi nous sommes naturellement passé à l'interface graphique qui se faisait ultérieurement dans le terminal à l'aide des très nombreux débogage qui nous permettaient de nous retrouver entre ses processus.

5.2 Tests

La phase de tests a constitué une étape essentielle du projet, compte tenu du caractère concurrent et intrinsèquement non déterministe du système développé. Des tests fonctionnels simples ont d'abord été menés afin de vérifier le bon fonctionnement de chaque composant pris isolément. Par la suite, des tests de concurrence ont été réalisés en faisant intervenir simultanément plusieurs proies et plusieurs prédateurs, ce qui a permis d'observer les interactions entre processus et de détecter d'éventuelles des races conditions. L'utilisation omniprésente de commandes de débogage et de traces d'exécution a joué un rôle déterminant pour comprendre l'ordre réel des événements, analyser les comportements inattendus.

6. Exécution et utilisation

L'exécution du programme s'effectue à partir du script principal de l'environnement, lequel joue un rôle central en initialisant l'état global de l'écosystème, en lançant les mécanismes internes de simulation et en créant les différentes entités nécessaires à son fonctionnement. Une fois démarré, le système évolue de manière autonome : l'herbe croît progressivement en fonction des règles définies, les proies voient leur énergie diminuer au fil du temps et cherchent à se nourrir afin de survivre, tandis que les prédateurs interviennent lorsque les proies deviennent affaiblies, matérialisant ainsi le phénomène de prédation. Cette évolution est rendue observable en temps réel grâce à une interface graphique, qui permet de visualiser sous forme de graphiques l'évolution de la quantité d'herbe ainsi que le nombre de proies et de prédateurs en fonction du temps. L'interface offre également la possibilité d'interagir dynamiquement avec l'écosystème, notamment en ajoutant des proies, des

prédateurs ou en déclenchant un épisode de sécheresse. L'ensemble de ces interactions produit une évolution dynamique et non figée de l'écosystème. Il convient toutefois de souligner certaines limites inhérentes au système : le comportement observé est non déterministe, dépend fortement du timing d'exécution des processus et repose sur un modèle volontairement simplifié. Ces limites sont assumées dans le cadre du projet, car elles permettent de mettre en évidence les problématiques réelles de la programmation parallèle et concurrente tout en conservant une architecture lisible et maîtrisée

7. Problèmes rencontrés et analyse critique

Le développement de ce projet a mis en évidence plusieurs difficultés techniques et organisationnelles, inhérentes à la conception d'un système concurrent. L'une des premières problématiques rencontrées a été une confusion initiale entre l'utilisation de threads et de processus, conduisant à une duplication involontaire de l'environnement et à des incohérences dans l'état global du système. Cette erreur conceptuelle a notamment entraîné des comportements inattendus, tels que des modifications d'état non partagées ou des mécanismes de synchronisation inefficaces. Par ailleurs, des désenregistrements prématurés de proies ont été observés, empêchant leur prise en compte correcte par les prédateurs. D'autres difficultés sont apparues autour des accès concurrents non protégés à la mémoire partagée, générant des races conditions difficiles à observer. De plus se sont ajoutées des contraintes organisationnelles, notamment la synchronisation du travail entre nous, la gestion des branches, des *merge* et des commits via Github, ainsi que la nécessité de se mettre d'accord sur une architecture commune et des méthodes partagées. Enfin, la rédaction du rapport a également posé des contraintes pratiques, l'impossibilité d'utiliser certains outils comme Microsoft Word et de se rabattre sur Google Docs.

D'un point de vue critique, plusieurs améliorations pourraient être envisagées pour enrichir le modèle. Tel que l'introduction de comportements plus complexes pour les entités vivantes comme la reproduction et le paramétrage complet de la simulation depuis l'interface graphique. Ces pistes d'amélioration montrent que le projet, bien que volontairement limité dans sa complexité, constitue une base solide pour des développements futurs. La réalisation de ce projet était particulièrement intéressante car elle nous a montré une nouvelle manière de programmer.

8. Conclusion

En conclusion, ce projet nous a permis de pouvoir mettre en place l'usage de différentes méthodes de programmation parallèle concurrente. Cela a été un projet enrichissant et challengeant bien qu' il ait été plus compliqué que prévu. Quelques jours de plus auraient permis d'implémenter certaines options importantes comme la reproduction.

Annexe A – Usage de l'IA

L'IA a été utilisée comme outil d'aide tout au long du projet. Elle a principalement servi à clarifier certains aspects du cours et des TDs, en proposant des explications adaptées à notre niveau et formulées en français. Elle nous a également aidé à comprendre certaines erreurs rencontrées, à formaliser les algorithmes sous forme de pseudo-codes présentés en annexe et à utiliser la librairie Matplotlib qui réalise notre interface graphique.

L'IA n'a pas été utilisée pour écrire le code du projet ni pour prendre des décisions d'architecture. Elle a uniquement joué un rôle pédagogique et d'accompagnement, comparable à un support de cours, tandis que la conception et l'implémentation finale relèvent d'un travail personnel.

Annexe B – pseudo-codes

1. Environnement

```
INITIALISER l'environnement et la mémoire partagée
DÉMARRER les processus Display, Proies et Prédateurs

TANT QUE la simulation est active FAIRE
    SI pas de sécheresse ALORS
        FAIRE CROÎTRE l'herbe
    SINON
        METTRE À JOUR la durée de sécheresse
    FIN SI

    ENVOYER l'état des populations au Display
    ATTENDRE 1 unité de temps
FIN TANT QUE
```

2. Proie

```
INITIALISER l'énergie

TANT QUE la proie est vivante FAIRE
    DIMINUER l'énergie
    SI faim ALORS
        CONSOMMER de l'herbe si disponible
    FIN SI
    SI énergie ≤ 0 ALORS
        TERMINER le processus
    FIN SI
    ATTENDRE 1 unité de temps
FIN TANT QUE
```

3. Prédateur

INITIALISER l'énergie

TANT QUE le prédateur est vivant FAIRE
    DIMINUER l'énergie
    SI faim ALORS
        CHASSER une proie disponible
    FIN SI
    SI énergie ≤ 0 ALORS
        TERMINER le processus
    FIN SI
    ATTENDRE 1 unité de temps
FIN TANT QUE

4. Display

INITIALISER l'interface graphique

TANT QUE la fenêtre est ouverte FAIRE
    RÉCUPÉRER les données de l'environnement
    METTRE À JOUR les courbes
    RAFRAÎCHIR l'affichage
FIN TANT QUE