Avrile Floro (n° étudiant : 22000086)

Programmation Orientée Objet

Le 9 juin 2024

Table des matières

Programmation Orientée Objet

Projet - Écosystème

Avrile Floro
Étudiante n°22000086

1 Projet final - Écosytème

1.1 Présentation du projet Écosystème

Dans le cadre du cours de Programmation Orientée Objet, nous avons souhaité développer un projet visant à simuler un écosystème dans lequel coexistent diverses espèces animales et végétales. Au sein de cet écosystème, les interactions entre organismes sont représentées à travers ce que nous appelons le cycle de la vie.

1.1.1 Le but du projet

Ce projet a pour objectif de créer une simulation dynamique d'un écosystème qui reproduit les comportements observés dans des écosystèmes réels, tels que la reproduction, la prédation et le vieillissement.

Pour mener à bien ce projet, il est crucial de modéliser les interactions entre les différentes espèces. Nous avons introduit divers types d'animaux (mammifères, oiseaux, reptiles), chacun suivant un régime alimentaire spécifique (charognard, carnivore, herbivore, légumivore, ovivore). Plusieurs types de végétaux sont également représentés au sein de l'écosystème (arbres, herbes, légumes). Les espèces animales et végétales interagissent au sein de l'écosystème, notamment pour se nourrir.

L'un des principaux défis du projet réside dans la simulation du cycle de la vie. Cela implique que les animaux puissent se reproduire avec d'autres individus du sexe opposé et de la même espèce. En outre, notre simulation prend en compte le vieillissement des êtres vivants qui, avec le temps, peuvent mourir de vieillesse ou même empoisonnés.

Enfin, créer un écosystème dynamique offre une perspective enrichissante sur le fonctionnement des écosystèmes naturels. Au fil de nombreux essais avec notre simulation, nous avons réalisé qu'il était extrêmement complexe de prédire les évolutions de notre écosystème, ce qui souligne le délicat équilibre nécessaire au maintien des écosystèmes naturels.

1.2 Contexte

Bien que notre projet ne fasse pas appel à une interface graphique, nous avons utilisé divers outils pour modéliser un écosystème dynamique.

Le langage de programmation choisi est Java, conformément aux exigences du cours de Programmation Orientée Objet.

Concernant l'environnement de développement intégré (IDE), le cours recommandait initialement Eclipse. Cependant, trouvant son interface peu intuitive, nous avons opté pour IntelliJ IDEA que nous pouvons télécharger gratuitement en tant qu'étudiante. Nous avons été satisfaite de notre choix, notamment en raison de l'expérience utilisateur supérieure. Nous avons même commencé à utiliser PyCharm, également édité par JetBrains, pour d'autres projets. Nous avons pu constater que l'utilisation d'un IDE nous a fait gagner du temps.

Le développement de notre écosystème s'est effectué sur un système d'exploitation Mac OS, ce qui correspond à notre configuration habituelle. Nous disposions d'une machine virtuelle Linux Ubuntu mais elle n'a pas été nécessaire pour ce projet.

Afin de représenter notre écosystème, nous avons utilisé plusieurs structures, telles que des classes, des classes abstraites, des interfaces et des énumérations. Cela nous a permis de simuler de manière réaliste les intéractions entre les différentes espèces comprises au sein de notre écosystème. Par ailleurs, les concepts étudiés dans le cadre du cours, tels que l'héritage, le polymorphisme ou la réflexion, ont permis d'améliorer notre compréhension

et notre capacité à modéliser un écosystème complexe.

1.3 Organisation et fonctionnement

Bien que nous ayons préparé un diagramme de classe UML, celui-ci n'est pas directement inclus dans ce document en raison de ses dimensions. Il a néanmoins été ajouté en annexe pour consultation.

1.3.1 Présentation générale de l'écosystème

La classe Écosystème

Au coeur de notre simulation se trouve la classe `Ecosystème`, qui est directement liée à la classe `Organisme` par une relation de composition. Cette relation indique que l'écosystème contient et gère plusieurs organismes.

La classe Organisme

La classe `Organisme` sert de base pour toutes les entités vivantes dans l'écosystème. Elle inclut des attributs communs tels que `id`, `ecosystème` (pour l'écosystème auquel l'organisme appartient), ainsi que des méthodes pour obtenir et définir ces attributs. Cela permet une gestion unifiée des différentes entités vivantes au sein de l'écosystème.

La classe Végétal et ses dérivés

La classe `Végétal` est la superclasse pour les entités végétales. Trois sous-classes abstraites en dérivent : `Légume`, `Herbe`, et `Arbre`, permettant de catégoriser les plantes selon les besoins alimentaires spécifiques de certains animaux. Par exemple, les classes `Rhubarbe`, `Asperge`, et `Laitue` héritent de la classe abstraite `Légume` ; `Menthe` provient de `Herbe` ; et `Bouleau` de `Arbre`.

La classe Animal et ses dérivés

Similairement, la classe `Animal` sert de superclasse pour toutes les entités animales. Elle se subdivise en trois autres classes abstraites : `Mammifère`, `Oiseau`, et `Reptile`, respectant une structure parallèle à celle des végétaux. Cette classification reflète les groupements naturels dans un écosystème réel. Des exemples incluent la `Tortue` qui hérite de `Reptile` ; les `Ours`, `Biche`, et `Loup` de `Mammifère` ; et le `Corbeau` de `Oiseau`.

Nous avons opté pour l'utilisation d'interfaces pour définir le régime alimentaire et le mode de reproduction des animaux, offrant ainsi une flexibilité accrue. Les régimes alimentaires sont représentés par des classes concrètes telles que `Herbivore`, `Carnivore`, `Charognard`, `Legumivore` et `Ovivore`. La classe abstraite `Animal` contient une liste de régimes alimentaires. Chaque sous-classe concrète d'`Animal` initialise sa liste de régimes de façon appropriée dans son constructeur. Par exemple, la `Tortue` adopte un régime `Légumivore`, tandis que l'`Ours` est `Omnivore`, combinant les régimes `Carnivore`, `Herbivore` et `Ovivore`.

En ce qui concerne la reproduction, chaque animal a un seul mode de reproduction défini par un attribut de la classe abstraite `Animal`. Le mode de reproduction est représenté par une interface `Reproduction`, avec des implémentations concrètes comme `Vivipare` et `Ovipare`. Les `Tortues` et les `Corbeaux` sont des exemples d'`Ovipares`, tandis que les `Biches`, les `Loups` et les `Ours` sont classés comme `Vivipares`.

La classe Oeuf

Les instances de la classe `Oeuf` sont générées suite la ponte des femelles ovipares lors de la reproduction. Cet objet conserve alors les caractéristiques des deux parents (de la mère dès la ponte et du père après la fécondation), permettant ainsi de tracer les attributs transmis aux descendants.

1.3.2 Respect des consignes

Nous allons reprendre pas à pas les consignes du projet et expliciter la façon dont nous les avons suivies.

> Dans cet exercice, nous souhaitons modéliser un écosystème dans lequel nous trouverons des végétaux ainsi que des animaux.

Notre simulation d'écosystème incorpore une variété d'organismes : des végétaux, des animaux et des oeufs. Les végétaux sont issus de la classe abstraite `Vegetal`, tandis que les animaux proviennent de la classe abstraite `Animal`. Les oeufs sont représentés par la classe concrète `Oeuf`.

> Pour les animaux, nous trouverons des herbivores, des carnivores, des omnivores et des charognards.

Pour modéliser les régimes alimentaires des animaux, nous avons employé une interface et plusieurs classes concrètes spécifiques :

L'interface `Regime` définit la méthode `peutManger(Organisme)` qui est implémentée par toutes les classes de régimes alimentaires.
La classe concrète `Herbivore` implémente l'interface `Regime` et spécifie les animaux se nourrissant de végétaux.
La classe concrète `Legumivore`, qui étend `Herbivore`, spécifie les animaux se nourrissant exclusivement de légumes.
La classe concrète `Carnivore` implémente l'interface `Regime` et spécifie les animaux se nourrissant d'animaux vivants.
La classe concrète `Charognard`, qui étend `Carnivore`, spécifie les charognards se nourrissant uniquement d'animaux déjà morts.
La classe concrète `Ovivore`, qui implémente l'interface `Regime`, spécifie les animaux se nourrissant d'oeufs.

Les animaux à régime omnivore, comme l'`Ours`, combinent plusieurs régimes en incluant des instances des classes `Herbivore`, `Carnivore` et `Ovivore` dans leur liste de régimes alimentaires, reflétant ainsi leur alimentation variée.

> Les herbivores sont des animaux pouvant manger des végétaux de n'importe quel type ou exclusivement des légumes.

Les animaux herbivores de notre écosystème peuvent se nourrir de tous les types de végétaux disponibles. Toutefois, les légumivores (une variante du régime alimentaire herbivore) se limitent à la consommation exclusive de légumes.

Par exemple, les biches suivent un régime alimentaire herbivore et peuvent donc consommer une variété de végétaux, tandis que les tortues, dont le régime alimentaire est légumivore, se nourrissent uniquement de légumes.

Les herbivores peuvent manger des végétaux de n'importe quel type
Où : Classe Herbivore, méthode peutManger
Comment : La méthode vérifie si l'organisme est un végétal (Vegetal), vivant, comestible, et appartenant à un

écosystème.

Les herbivores peuvent manger exclusivement des légumes
Où : Classe Legumivore, méthode peutManger
Comment : La méthode vérifie si l'organisme est un légume (Legume), vivant, comestible, et appartenant à un écosystème.

Exemple avec une Biche herbivore
Où : Classe Biche, méthode initialiseRegime
Comment : La méthode initialise le régime alimentaire de la biche en ajoutant Herbivore à un ensemble de régimes alimentaires
Où : Classe Biche, constructeur
Comment : Le constructeur appelle initialiseRegime pour définir le régime alimentaire de la biche.

Exemple avec une Tortue légumivore
Où : Classe Tortue, méthode initialiseRegime
Comment : La méthode initialise le régime alimentaire de la tortue en ajoutant Legumivore à un ensemble de régimes alimentaires.
Où : Classe Tortue, constructeur
Comment : Le constructeur appelle initialiseRegime pour définir le régime alimentaire de la tortue.

> Les carnivores sont des animaux pouvant manger des herbivores ainsi que d'autres carnivores.

Les animaux carnivores, comme le Loup, peuvent consommer tout animal vivant qui suit un régime alimentaire herbivore ou carnivore (ainsi que leurs variantes, comme les régimes alimentaires charognard et légumivore) et qui se trouve dans l'écosystème.

Les carnivores peuvent manger des herbivores
Où : Classe Carnivore, méthode peutManger
Comment : La méthode vérifie si l'organisme est un animal (Animal). Ensuite, elle vérifie si l'animal est vivant, suit le régime herbivore (Herbivore.class), et appartient à un écosystème. La vérification du régime se fait via la méthode suitRegime (qui se trouve dans la classe Animal) qui compare la classe du régime avec Herbivore.class.

Les carnivores peuvent manger d'autres carnivores
Où : Classe Carnivore, méthode peutManger
Comment : La méthode vérifie de manière similaire à celle des herbivores, si l'animal suit le régime carnivore (Carnivore.class).

Exemple avec un Loup carnivore
Où : Classe Loup, méthode initialiseRegime
Comment : La méthode initialise le régime alimentaire du loup en ajoutant un Carnivore à un ensemble de régimes alimentaires.
Où : Classe Loup, constructeur
Comment : Le constructeur appelle initialiseRegime pour définir le régime alimentaire du loup.

> Les omnivores eux peuvent manger des carnivores, des herbivores des plantes ainsi que des oeufs.

Dans notre écosystème, les omnivores adoptent un régime alimentaire diversifié. L'ours, par exemple, est un omnivore qui combine les régimes alimentaires herbivore, carnivore, et ovivore. Il est capable de consommer d'autres animaux vivants, tous les types de végétaux disponibles, ainsi que des oeufs.

Les omnivores peuvent manger des végétaux
Où : Classe Ours, méthode initialiseRegime, ajout de Herbivore

Comment : La méthode ajoute le régime herbivore (Herbivore) à l'ensemble des régimes alimentaires de l'ours.

Les omnivores peuvent manger des oeufs
Où : Classe Ours, méthode initialiseRegime, ajout de Ovivore
Comment : La méthode ajoute le régime ovivore (Ovivore) à l'ensemble des régimes alimentaires de l'ours.

Les omnivores peuvent manger des herbivores
Où : Classe Carnivore, méthode peutManger
Comment : La méthode vérifie si l'organisme est un animal (Animal), vivant, et suit le régime herbivore (Herbivore.class).

Les omnivores peuvent manger des carnivores
Où : Classe Carnivore, méthode peutManger
Comment : La méthode vérifie si l'organisme est un animal (Animal), vivant, et suit le régime carnivore (Carnivore.class).

Exemple avec un Ours omnivore
Où : Classe Ours, méthode initialiseRegime
Comment : La méthode initialise le régime alimentaire de l'ours en ajoutant Herbivore, Ovivore, et Carnivore à l'ensemble des régimes alimentaires.
Où : Classe Ours, constructeur
Comment : Le constructeur appelle initialiseRegime pour définir le régime alimentaire de l'ours.

> Les charognards sont des animaux mangeant des animaux morts.

Les charognards se nourrissent exclusivement d'animaux déjà morts.

Les charognards peuvent manger des animaux morts
Où : Classe Charognard, méthode peutManger
Comment : La méthode vérifie si l'organisme est un animal (Animal). Ensuite, elle vérifie si l'animal est mort et appartient à un écosystème. La vérification de l'état de l'animal se fait via la méthode isVivant qui doit renvoyer false.

Exemple avec un Corbeau charognard
Où : Classe Corbeau, méthode initialiseRegime
Comment : La méthode initialise le régime alimentaire du corbeau en ajoutant un Charognard à un ensemble de régimes alimentaires.
Où : Classe Corbeau, constructeur
Comment : Le constructeur appelle initialiseRegime pour définir le régime alimentaire du corbeau.

> De plus, chacune des espèces présentent dans cet écosystème pourra se reproduire.

Dans notre écosystème, toutes les espèces, tant animales que végétales, sont capables de reproduction.

Chaque espèce peut se reproduire

Où : Classe abstraite Animal, attribut reproduction
Comment : Le mode de reproduction est un attribut de type Reproduction, qui est une interface avec des implémentations concrètes comme Vivipare et Ovipare.

Où : Classe Vivipare, méthode seReproduire
Comment : La méthode seReproduire permet à un animal de type vivipare de se reproduire si les conditions (âge, écosystème, sexe, etc.) sont remplies.

Où : Classe Ovipare, méthode seReproduire
Comment : La méthode seReproduire permet à un animal de type ovipare de pondre des œufs et de fertiliser des oeufs existants si les conditions sont remplies.

Exemple d'attribution d'un mode de reproduction
Où : Classe Loup, constructeur
Comment : Le constructeur de la classe Loup attribue un mode de reproduction Vivipare en appelant this.setReproduction(new Vivipare(this)).

Où : Classe Tortue, constructeur
Comment : Le constructeur de la classe Tortue attribue un mode de reproduction Ovipare en appelant this.setReproduction(new Ovipare(this)).

> Les végétaux posséderont un nombre de graines indiquant le nombre de reproductions possibles.

Dans la réalité, les végétaux se reproduisent soit de manière autonome soit par pollinisation. Nous avons modélisé cette capacité en déterminant que le désir de reproduction chez les végétaux se manifeste environ une fois sur sept, soit 15% du temps. Les végétaux n'ont pas besoin de partenaire pour se reproduire.

Les caractéristiques des végétaux, comme le nombre de graines ou l'espérance de vie, sont générées aléatoirement selon des fourchettes prédéfinies pour chaque espèce, sans héritage direct des caractéristiques du parent.

Les végétaux possèdent un nombre de graines indiquant le nombre de reproductions possibles
Où : Classe Vegetal, attribut nbGraines
Comment : L'attribut nbGraines indique le nombre de graines disponibles pour la reproduction.

Où : Classe Vegetal, méthode seReproduire
Comment : La méthode seReproduire vérifie les conditions de reproduction (et notamment s'il reste au moins une graine) et crée une nouvelle instance de la même espèce si les conditions sont remplies, diminuant le nombre de graines (nbGraines) à chaque reproduction réussie.
Lignes : 164, 174
Exemple avec le Bouleau
Où : Classe Bouleau, constructeur
Comment : Le constructeur initialise le nombre de graines (nbGraines) de manière aléatoire.

> Du côté des animaux ceux-ci nécessiteront un partenaire du sexe opposé pour se reproduire et seront, soit vivipares, soit ovipares et pondront des oeufs.

Nos animaux se classent en deux catégories selon leur mode de reproduction : `ovipares`, qui pondent des oeufs, et `vivipares`, où l'oeuf se développe dans l'utérus maternel. Les Tortues et les Corbeaux sont des `ovipares` ; les Biches, les Ours et les Loups sont `vivipares`.

Nous avons modélisé ces modes de reproduction en utilisant une interface `Reproduction` dont héritent les classes concrètes `Vivipare` et `Ovipare`. Chaque animal possède un attribut `reproduction` de type `Reproduction`, défini par chaque sous-classe concrète (`Biche`, `Ours`, `Tortue`, etc.) dans son constructeur.

Pour les `vivipares`, la reproduction se déclenche lorsque la femelle, dans 25% des cas, cherche un mâle de la même espèce qui souhaite également se reproduire (ce qui se produit aussi dans 25% des cas). Si une rencontre réussie a lieu, ils produiront un descendant héritant aléatoirement des caractéristiques héréditaires de l'un ou l'autre parent.

Pour les `ovipares`, tout d'abord, dans 25% des cas, la femelle ovipare pond un oeuf ou plusieurs oeufs. Ensuite, dans 25% des cas, l'ovipare mâle cherche à féconder un ou des oeufs de son espèce.

Les femelles `vivipares` possèdent un attribut `nombre de petits`, qui indique le nombre d'individus auxquels elles peuvent donner naissance lors d'une reproduction, tandis que cet attribut est `null` chez les mâles.

Pour les `ovipares`, les femelles possèdent aussi un attribut `nombre de petits` indiquant le nombre d'oeufs qu'elles peuvent pondre. Chez les mâles, cet attribut indique le nombre d'oeufs qu'ils peuvent féconder par cycle reproductif.

Les animaux nécessitent un partenaire du sexe opposé pour se reproduire et seront soit vivipares, soit ovipares, et pondront des oeufs

Où : Classe Animal, méthode bebeDeReproduction
Comment : La méthode bebeDeReproduction (utilisée par les vivipares) crée un nouveau-né en utilisant les caractéristiques des deux parents de la même espèce. Elle vérifie que les parents sont de sexes opposés en utilisant l'attribut sexe.

Où : Classe Vivipare, méthode seReproduire
Comment : La méthode seReproduire recherche un partenaire mâle pour une femelle de la même espèce. Elle vérifie que le partenaire potentiel est vivant, en âge de se reproduire, du sexe opposé, et souhaite se reproduire. Si les conditions sont remplies, elle crée les petits en utilisant la méthode bebeDeReproduction.

Où : Classe Ovipare, méthode seReproduire
Comment : La méthode seReproduire permet aux femelles de pondre des oeufs et aux mâles de fertiliser les oeufs non fertilisés. Les femelles pondent un nombre d'oeufs basé sur leurs caractéristiques. Les mâles cherchent et fertilisent les oeufs de la même espèce et du même écosystème.

Détail sur la nécessité d'un partenaire du sexe opposé
Où : Classe Vivipare, méthode seReproduire
Comment : Pour la reproduction, la méthode vérifie que l'animal est une femelle. Elle parcourt ensuite les organismes de l'écosystème pour trouver un partenaire mâle de la même espèce.

Où : Classe Ovipare, méthode seReproduire
Comment : Les femelles pondent des oeufs sans besoin immédiat de mâle. Les mâles fertilisent les oeufs non fertilisés de la même espèce et du même écosystème.

> Ajoutez à chacun de vos animaux un attribut vitesse, une espérance de vie (attribut décroissant) et un attribut faim fixé par défaut à 10, qui pourra décroître jusqu'à 0.

Nous avons introduit des attributs tels que `vitesse`, `espérance de vie` (qui diminue avec le temps), et `faim`, fixée par défaut à 10 et décroissante.

Ces attributs sont attribués aléatoirement selon l'espèce lors de l'instanciation initiale des animaux dans notre écosystème, c'est-à-dire lorsqu'ils ne sont pas issus de la reproduction.

Si un animal provient d'une reproduction au sein de l'écosystème, il héritera aléatoirement des attributs `taille`, `poids`, `empoisonné`, et `vitesse` de l'un de ses parents. Pour les ovipares, l'attribut `nombre de petits` est également hérité aléatoirement de l'un des parents. Chez les vivipares, seules les femelles hériteront de cet attribut de leur mère, cet attribut étant défini à `null` chez les mâles. De plus, les animaux issus de la reproduction conservent une référence à leurs parents via leurs identifiants, tandis que pour ceux instanciés directement dans l'écosystème, les identifiants des parents sont à `null`.

L'attribut `espérance de vie` est aussi attribué de manière aléatoire en fonction de l'espèce de l'animal, que l'animal soit issu de la reproduction ou instancié directement.

Ajoutez à chacun de vos animaux un attribut vitesse, une espérance de vie (attribut décroissant) et un attribut faim fixé par défaut à 10, qui pourra décroître jusqu'à 0
Où : Classe Animal, attributs vitesse, esperanceVie, et faim

Comment :
vitesse : Attribut défini pour indiquer la vitesse de l'animal.
esperanceVie : Attribut décroissant qui représente l'espérance de vie de l'animal.
faim : Attribut initialisé à 10 qui pourra décroître jusqu'à 0.

Où : Classe Animal, méthode vieillir (254-263)
Comment : La méthode vieillir gère la diminution de l'attribut faim. Chaque jour, si l'animal n'a pas mangé, sa faim décroît de 1. Si l'attribut faim atteint 0, l'animal meurt de faim.

Exemple d'attribution selon les espèces d'animaux
Où : Classe Biche, constructeur
Comment :
vitesse : Attribuée aléatoirement entre 10 et 17.
esperanceVie : Attribuée aléatoirement entre 6 et 12.
faim : Initialisée par défaut à 10 dans la classe Animal.

Où : Classe Ours, constructeur
Comment :
vitesse : Attribuée aléatoirement entre 10 et 12.
esperanceVie : Attribuée aléatoirement entre 7 et 9.
faim : Initialisée par défaut à 10 dans la classe Animal.

Attributs en cas de reproduction dans l'écosystème
Où : Classe Animal, méthode bebeDeReproduction
Comment : Lors de la reproduction, l'attribut vitesse du nouveau-né est déterminé aléatoirement en fonction des caractéristiques de la mère ou du père.

Méthode eclore
Où : Classe Oeuf, méthode eclore
Comment : Lors de l'éclosion des oeufs, l'attribut vitesse du nouveau-né est également déterminé aléatoirement en fonction des caractéristiques de la mère ou du père.

> Codez les interfaces "ovipares" et "vivipares" dont hériteront certaines espèces d'animaux.

Nous avons implémenté une interface `Reproduction` et des classes concrètes `Vivipare` et `Ovipare` pour modéliser la reproduction de nos animaux et éviter au maximum la duplication de code.
Chaque animal a un attribut `reproduction` de type `Reproduction`. Son attribution est effectuée automatiquement dans le constructeur de chaque classe concrète d'animal. Les Tortues et les Corbeaux sont classifiés comme `ovipares`, tandis que les Biches, les Loups, et les Ours sont considérés comme `vivipares`.

Interface Reproduction
L'interface Reproduction doit être implémentée par les classes Vivipare et Ovipare.

Où : Classe concrète Vivipare, implémentation de Reproduction
Comment : La classe Vivipare implémente l'interface Reproduction et définit la méthode seReproduire pour gérer la reproduction des animaux vivipares. La reproduction nécessite un partenaire du sexe opposé.

Où : Classe concrète Ovipare, implémentation de Reproduction
Comment : La classe Ovipare implémente l'interface Reproduction et définit la méthode seReproduire pour gérer la reproduction des animaux ovipares. Les femelles pondent des œufs et les mâles les fertilisent.

Exemple d'attribution des modes de reproduction
Où : Classe Loup, constructeur
Comment : Le constructeur de la classe Loup attribue un mode de reproduction Vivipare en appelant this.setReproduction(new Vivipare(this)).

Où : Classe Corbeau, constructeur

Comment : Le constructeur de la classe Corbeau attribue un mode de reproduction Ovipare en appelant this.setReproduction(new Ovipare(this)).

> Ajoutez l'objet "oeuf". Cet objet aura des caractéristiques par défaut tant qu'il n'a pas été fertilisé par un mâle de son espèce, et disparaîtra de son environnement pour laisser place à un animal une fois éclos. À noter que celui-ci peut être consommé par les animaux omnivores.

Chez les `ovipares`, dans 25% des cas, la femelle pond des oeufs. Le nombre d'oeufs dépend de l'attribut `nombre de petits`. Lorsqu'un oeuf est pondu, il hérite des attributs de sa mère : `espèce`, `idMere`, `tailleMere`, `poidsMere`, `mereEmpoisonnee`, `mereVitesse`, et `mereNbPetits`. Garder trace de ces informations est essentielle car si la mère disparaissait de l'écosystème avant l'éclosion de l'oeuf, ses attributs ne seraient plus accessibles.

Les attributs du père, tels que `idPere`, `taillePere`, `poidsPere`, `pereEmpoisonnee`, `pereVitesse`, et `pereNbPetits` sont initialisés à `null` dans le cas où l'oeuf n'est pas fécondé. Ils sont mis à jour avec les informations du père lors de la fécondation, si elle a lieu.

Lors de l'éclosion de l'oeuf, la réflexion est utilisée pour instancier un nouvel animal de l'espèce appropriée, héritant aléatoirement des caractéristiques de l'un ou l'autre parent. L'oeuf est ensuite retiré de l'écosystème, et le nouvel animal est ajouté.

Pour gérer la population d'oeufs, une méthode élimine ceux qui pourrissent après cinq jours sans éclosion. De plus, les oeufs doivent être fécondés avant trois jours pour éclore.

Par ailleurs, l'Ours utilise la classe `Ovivore`, via son régime alimentaire, pour se nourrir d'oeufs.

Caractéristiques par défaut de l'oeuf
Où : Classe Oeuf, attributs age, fertilise, tailleMere, poidsMere, mereVitesse
Comment : age est initialisé à 0. fertilise est initialisé à false. Les caractéristiques héritées de la mère (taille, poids, vitesse, etc.) sont définies lors de la création de l'oeuf (lignes 31-36).

Ponte de l'oeuf
Où : Classe Ovipare, méthode seReproduire
Comment :
Conditions nécessaires : L'animal doit être une femelle ovipare en âge de se reproduire.
Effets de la ponte : La femelle pond un certain nombre d'oeufs (nbOeufsSouhaites). Les oeufs sont créés avec les caractéristiques de la mère (taille, poids, vitesse, etc.) dans le constructeur de la classe Oeuf.

Fertilisation de l'oeuf
Où : Classe Oeuf, méthode estFertilise
Comment :
Conditions nécessaires : Un mâle de la même espèce doit fertiliser l'oeuf. L'oeuf et le mâle doivent appartenir au même écosystème.
Effets de la fertilisation : L'oeuf devient fertilisé (fertilise = true). Les caractéristiques du père sont enregistrées (taillePere, poidsPere, pereVitesse, etc.) (lignes 210-217).

Éclosion de l'oeuf
Où : Classe Oeuf, méthode eclore
Comment :
Conditions nécessaires : L'oeuf doit atteindre un certain âge (3 jours). L'oeuf doit être fertilisé. (263-265)
Effets de l'éclosion : Un nouvel animal de l'espèce correspondante naît avec des caractéristiques héritées de ses parents (aléatoirement du père ou de la mère) (270-300). L'oeuf disparaît de l'écosystème (312-315).

Consommation de l'oeuf par les omnivores
Où : Classe Oeuf, méthode estMange et Classe Ovivore, méthode peutManger
Comment :
Conditions nécessaires : L'oeuf doit appartenir à un écosystème. L'animal omnivore doit avoir le régime alimentaire Ovivore.

Effets de la consommation : L'oeuf est retiré de l'écosystème. L'oeuf disparaît, simulant sa consommation par l'animal omnivore.

> Ajoutez un animal pondant des oeufs.

Les Tortues et les Corbeaux sont les deux animaux de notre écosystème qui pondent des oeufs.

Exemple avec une Tortue ovipare
Où : Classe Tortue, constructeur
Comment : Le constructeur de la classe Tortue attribue le mode de reproduction Ovipare.

Exemple avec un Corbeau ovipare
Où : Classe Corbeau, constructeur
Comment : Le constructeur de la classe Corbeau attribue le mode de reproduction Ovipare.

Gestion de la ponte des oeufs
Où : Classe Ovipare, méthode seReproduire
Comment : La méthode seReproduire gère la ponte des oeufs pour les femelles ovipares et la fertilisation des oeufs pour les mâles ovipares.

> Implémentez votre environnement. Celui-ci possédera une liste d'espèces (végétales + animales) et pourra ajouter, retirer ou afficher les espèces de cette liste.

Notre écosystème contient une liste des espèces présentes, y compris les espèces végétales, les espèces animales, et les oeufs. Nous avons la capacité d'afficher toutes les espèces présentes dans l'écosystème. De plus, nous avons développé des méthodes permettant d'ajouter ou de supprimer une espèce de l'écosystème. Lorsqu'une espèce est supprimée, tous les individus appartenant à cette espèce sont également retirés de l'écosystème.

Nous avons également mis en place des listes par espèce, qui permettent de lister tous les membres d'une même espèce présents dans l'écosystème. Cette fonctionnalité s'est révélée très utile pour suivre l'évolution de notre écosystème.

Ajouter une espèce
Où : Classe Ecosysteme, méthode ajouterEspece
Comment : Ajoute une espèce à l'ensemble des espèces de l'écosystème.

Retirer une espèce
Où : Classe Ecosysteme, méthode supprimerEspece
Comment : Supprime une espèce de l'ensemble des espèces de l'écosystème. Parcourt les listes d'animaux, de végétaux et d'oeufs pour retirer tous les organismes de cette espèce.

Afficher les espèces
Où : Classe Ecosysteme, méthode afficherEspece
Comment : Affiche toutes les espèces présentes dans l'écosystème.

> Codez le cycle de la vie : définissez aléatoirement un nombre d'animaux et de végétaux (empoisonnés ou non), ainsi qu'un nombre de jours à réaliser. À chaque tour de boucle (jour), les végétaux et animaux pourront ou non se reproduire aléatoirement. De plus, à chaque tour, l'espérance de vie des animaux ainsi que leur faim décroît de 1. Si une espérance de vie atteint 0, alors l'animal mourra, de même pour sa faim.

Dans la classe `Main`, nous illustrons le cycle de la vie. Un écosystème nommé `foretTemperee` est créé, peuplé

initialement avec entre (aléatoirement) 5 à 8 instances de chaque espèce, parmi lesquelles figurent Asperge, Biche, Bouleau, Corbeau, Laitue, Loup, Menthe, Ours, Rhubarbe et Tortue. Les attributs de ces entités sont définis aléatoirement selon des fourchettes établies pour chaque espèce car l'instanciation est directe, et non pas issue de la reproduction.

Pour les instances directement instanciées, nous avons choisi que les animaux naissent non empoisonnés. Toutefois, un animal issu d'une reproduction peut hériter de l'attribut `empoisonne` de l'un de ses parents. Concernant les végétaux, seule la rhubarbe est empoisonnée par défaut, capable d'empoisonner les herbivores qui la consomment. Ces derniers peuvent alors transmettre leur empoisonnement à leur progéniture ainsi qu'aux carnivores qui les consomment.

À chaque cycle de la simulation, la possibilité de reproduction des végétaux est déterminée aléatoirement, avec une probabilité de 15% si les conditions nécessaires (comme la disponibilité de graines) sont remplies.

Chez les animaux suivant un mode de reproduction vivipare, la reproduction est initiée par une femelle désirant se reproduire, ce qui se produit dans 25% des cas. Elle cherche alors un mâle de la même espèce ayant également l'intention de se reproduire, une occurrence qui a également 25% de chance de se produire.

Chez les animaux ayant un mode de reproduction oviapare, une femelle pond un oeuf une fois sur quatre. Ensuite, dans 25% des cas, les mâles ovipares cherchent à féconder des oeufs de leur espèce.

Lors de chaque itération de la boucle, l'espérance de vie et la faim des animaux ainsi que l'espérance de vie des végétaux diminuent, progressant ainsi vers leur éventuelle mort si ces valeurs atteignent zéro.

Définir un nombre aléatoire d'animaux et de végétaux
Où : Dans la méthode main de la classe Main
Comment : Utiliser un objet Random pour générer un nombre aléatoire d'instances de chaque espèce.
Lignes : 13-64

Définir un nombre de jours à réaliser
Où : Dans la méthode main de la classe Main
Comment : Utiliser un objet Random pour déterminer un nombre aléatoire de jours pour la simulation.
Lignes : 66-67

Simulation du passage du temps
Où : Dans la méthode passageTemps de la classe Ecosysteme
Comment : Pour chaque jour de la simulation, appeler la méthode vivreUnJour pour chaque animal, végétal et oeuf.

Décroissance de l'espérance de vie et de la faim des animaux
Où : Dans la méthode vieillir de la classe Animal
Comment : À chaque jour, diminuer l'espérance de vie et la faim de chaque animal de 1. Si l'espérance de vie ou la faim atteint 0, l'animal meurt.

Reproduction aléatoire des végétaux et des animaux
Où : Dans les méthodes seReproduire de la classe Animal et Vegetal
Comment : Les organismes peuvent se reproduire aléatoirement chaque jour.

> Une fois la faim arrivée à 5 ou moins, l'animal aura besoin de se nourrir. Pour cela, il essayera de manger aléatoirement un objet faisant partie de son régime alimentaire. Si cet objet est plus lent que lui, alors il sera consommé et la faim remontera à 10. Sinon, l'animal ratera son dîner. Bien évidemment, pour les végétaux, aucun problème : les herbivores pourront les consommer à tous les coups.

Lorsque la faim d'un animal atteint 5 ou moins, il cherche à se nourrir en parcourant les organismes disponibles dans son écosystème selon son régime alimentaire. Les options de nourriture sont mélangées de manière aléatoire, ainsi que les régimes alimentaires de l'animal. Pour chaque régime, l'animal essaie de trouver une proie.

Si la proie est un autre animal vivant, la réussite de l'attaque dépend de la vitesse relative des deux animaux ; si l'animal est plus rapide que sa proie, il mange et sa faim remonte à 10. Les herbivores réussissent toujours à consommer des végétaux, et les charognards se nourrissent des animaux morts, dont la vitesse est considérée comme nulle. De même, les ovivores peuvent réussissent systématiquement à manger les oeufs.

Vérification de la faim
Où : Dans la méthode manger de la classe Animal
Comment : Vérifie si la faim est à 5 ou moins pour déclencher le besoin de se nourrir.
Ligne 318

Sélection d'une proie aléatoire
Où : Dans la méthode trouverProie de la classe Animal
Comment : Mélange la liste des organismes de l'écosystème et choisit une proie aléatoire en fonction de son régime alimentaire.

Vérification de la vitesse de la proie pour les animaux
Où : Dans la méthode trouverProie de la classe Animal
Comment : Si la proie est un animal, vérifie si l'animal chasseur est plus rapide que la proie avant de la désigner.
Lignes : 364-368

Consommation de la proie et mise à jour de la faim
Où : Dans la méthode manger de la classe Animal
Comment : Si la proie est consommée, remet la faim de l'animal à 10.
Lignes : 335-350

Consommation automatique des végétaux
Où : Dans la méthode trouverProie de la classe Animal
Comment : Les herbivores peuvent consommer les végétaux automatiquement sans vérifier la vitesse.
Lignes : 370-375

> Au bout de 3 jours, les animaux empoisonnés mourront et les oeufs écloreont pour laisser place à l'animal.

Si un animal consomme un végétal ou un autre animal empoisonné, son attribut `empoisonne` est activé et `poisonDepuis` s'initialise à 0. Les descendants peuvent également hériter de l'empoisonnement si l'un des parents était affecté. Avec chaque cycle journalier, ce compteur augmente et, atteignant 3, l'animal meurt.

Les oeufs, suivis par leur attribut `age`, éclosent après trois jours s'ils sont fécondés. Les oeufs non éclos de cinq jours pourrissent et sont retirés de l'écosystème.

Mort des animaux empoisonnés après 3 jours
Où : Dans la méthode vieillir de la classe Animal
Comment : Vérifie si l'animal est empoisonné et si le nombre de jours depuis l'empoisonnement est égal à 3. Si c'est le cas, l'animal meurt.
Lignes : 265-269

Éclosion des oeufs après 3 jours
Où : Dans la méthode eclore de la classe Oeuf (259 et suivantes)
Comment : Vérifie si l'âge de l'oeuf est égal à 3 jours et si l'oeuf est fertilisé. Si ces conditions sont remplies, l'oeuf éclot et un nouvel animal est créé avec des caractéristiques héritées aléatoirement des parents.

> Chaque élément mort devra être mangé une nouvelle fois avant de disparaître de l'environnement.

Les animaux qui meurent restent dans l'écosystème car ils peuvent être consommés par des Charognards. En revanche, les végétaux qui meurent sont retirés immédiatement de l'écosystème car aucun animal présent dans

notre écosystème ne consomme de végétaux morts.

Gestion de la mort des animaux
Où : Méthode mourrir dans la classe Animal
Comment : Lorsque l'animal meurt, il ne disparaît pas immédiatement de l'écosystème. Son état est mis à jour pour refléter qu'il est mort (ligne 288), mais il reste dans l'écosystème jusqu'à ce qu'il soit mangé par un autre animal.

Gestion du fait d'être mangé par un animal
Où : Méthode estMange dans la classe Animal
Comment : Lorsque l'animal mort ou vivant est mangé, il disparaît de l'écosystème. Cette méthode gère l'élimination de l'animal de l'écosystème après avoir été mangé.
Lignes : 216-230

> Faites en sorte que vos tours de boucle ne soient pas trop rapides afin de pouvoir visualiser les changements de l'environnement.

Pour permettre une observation claire, la simulation dans `Main` est configurée pour se dérouler sur une durée aléatoire de 15 à 20 jours, avec un intervalle de pause de 10 secondes entre chaque cycle journalier. Cette pause aide à suivre visuellement les évolutions et interactions au sein de l'écosystème. Ces configurations sont facilement modifiables dans `Main`.

Gestion des tours de boucle pour visualiser les changements
Où : Dans la classe Main, méthode main.
Comment : À chaque tour de boucle (jour), une pause est introduite à l'aide de la méthode sleep. Cela permet de ralentir l'exécution et de rendre les changements de l'écosystème plus visibles.
Lignes : 75-76

1.4 Tests réalisés

Tous les tests réalisés sont de type normal. Étant donné que notre écosystème est très majoritairement autonome, l'objectif principal des tests est de vérifier que le système peut s'exécuter sans erreurs.

1. `testAspergeSeReproduit` : Teste la reproduction d'une asperge.
2. `testAspergeMeurt` : Vérifie la mort naturelle d'une asperge due à la vieillesse.
3. `testAspergeVitJour` : Simule un jour dans la vie d'une asperge.
4. `testBicheMange` : Teste une biche qui mange.
5. `testBicheSeReproduit` : Vérifie la reproduction d'une biche.
6. `testBicheVitJour` : Simule un jour dans la vie d'une biche.
7. `testBicheMeurt` : Teste la mort naturelle d'une biche due à la vieillesse.
8. `testBouleauSeReproduit` : Vérifie la reproduction d'un bouleau.
9. `testBouleauMeurt` : Vérifie la mort d'un bouleau due à la vieillesse.
10. `testBouleauVitJour` : Simule un jour dans la vie d'un bouleau.
11. `testCorbeauMange` : Un corbeau mange.
12. `testCorbeauEmpoisonneMeurt` : Un corbeau meurt empoisonné après avoir mangé un animal empoisonné.
13. `testCorbeauPondOeuf` : Un corbeau pond un oeuf.

14. `testCorbeauFertiliseOeuf` : Un corbeau fertilise un oeuf.
15. `testOeufCorbeauEclot` : Un oeuf de corbeau éclot.
16. `testCorbeauVitUnJour` : Simule un jour dans la vie d'un corbeau.
17. `testAfficherEspeceEco` : Affiche les espèces de l'écosystème.
18. `testAfficherCompteEco` : Affiche le nombre d'individus de chaque espèce dans l'écosystème.
19. `supprimerEspeceEco` : Supprime une espèce de l'écosystème.
20. `testLaitueSeReproduit` : Teste la reproduction d'une laitue.
21. `testLaitueMeurt` : Vérifie la mort naturelle d'une laitue.
22. `testLaitueVitUnJour` : Simule un jour dans la vie d'une laitue.
23. `testLoupMange` : Un loup mange.
24. `testLoupSeReproduit` : Vérifie la reproduction d'un loup.
25. `testLoupMeurt` : Un loup meurt de faim.
26. `testLoupVitUnJour` : Simule un jour dans la vie d'un loup.
27. `testMentheSeReproduit` : Teste la reproduction d'une menthe.
28. `tesMentheMeurt` : Vérifie la mort naturelle d'une menthe.
29. `testMentheVitUnJour` : Simule un jour dans la vie d'une menthe.
30. `testOeufEstMange` : Un oeuf est mangé par un animal ayant un régime alimentaire ovivore.
31. `testOeufPourri` : Un oeuf pourrit.
32. `testOeufVitJour` : Un oeuf vit un jour.
33. `testOursMangeVegetal` : Un ours mange (illustration : végétal).
34. `testOursMangeAnimal` : Un ours mange (illustration : animal).
35. `testOursMangeOeuf` : Un ours mange (illustration : oeuf).
36. `testOursSeReproduit` : Vérifie la reproduction d'un ours.
37. `testOursVitJour` : Simule un jour dans la vie d'un ours.
38. `testOursMeurt` : Un ours meurt de vieillesse.
39. `testRhubarbeSeReproduit` : Teste la reproduction d'une rhubarbe.
40. `testRhubarbeMeurt` : Vérifie la mort naturelle d'une rhubarbe.
41. `testRhubarbeVitJour` : Simule un jour dans la vie d'une rhubarbe.
42. `testTortueMange` : Une tortue mange.
43. `testTortuePondOeuf` : Une tortue pond un oeuf.
44. `testTortueFertiliseOeuf` : Une tortue fertilise un oeuf.
45. `testOeufTortueEclot` : Un oeuf de tortue éclot.
46. `testTortueVitUnJour` : Simule un jour dans la vie d'une tortue.

1.5 Problèmes rencontrés et décisions de conception prises

Durant le développement de notre écosystème, nous avons été confrontée à plusieurs difficultés qui ont nécessité des décisions de conception spécifiques.

1.5.1 Problème rencontré : Représentation inexacte de l'héritage au sein du diagramme UML

Nous avons élaboré un diagramme UML pour représenter l'organisation de notre projet. Cependant, en raison de la complexité du projet, il s'est avéré difficile de représenter intégralement le système sur un format tenant en une page. Pour surmonter cette contrainte, nous avons utilisé la version locale de PlantUML, qui offre la possibilité de télécharger des images de haute qualité, permettant de zoomer. Bien que les fonctionnalités de `PlantUML` nous aient généralement satisfaite, nous avons constaté des limites dans la représentation de l'héritage par plusieurs classes d'une même classe parente. En effet, cette situation devrait idéalement être représentée par une flèche unique se divisant à un angle de 90°, ce qui n'est pas le cas actuellement.

1.5.2 Problème résolu : Modification de l'environnement pendant son parcours

Un des principaux problèmes rencontrés a été la modification de l'environnement pendant qu'il était parcouru par des processus. Cela entraînait des erreurs. Pour résoudre ce problème, nous avons opté pour l'utilisation systématique de copies des listes d'entités. Cette approche assure que toute modification sur la liste originale n'affecte pas la liste en cours de parcours.

1.5.3 Problème résolu : Disparition des géniteurs d'un oeuf

Un autre problème rencontré était la gestion des attributs des géniteurs pour un oeuf, surtout dans les cas où les géniteurs disparaissaient avant l'éclosion de l'oeuf. Pour assurer la transmission correcte des caractéristiques génétiques sans dépendre de la présence continue des parents dans l'environnement, nous avons choisi de stocker les attributs transmissibles directement dans l'oeuf au moment de sa création et de sa fécondation. Cette solution permet à l'oeuf de conserver toutes les informations nécessaires pour le développement de la nouvelle entité, indépendamment des modifications subies par les parents postérieurement à la ponte ou à la fécondation.

1.5.4 Décision de conception : Attributs des végétaux

Les végétaux dans notre écosystème possèdent plusieurs attributs clés, tels que `empoisonne`, `vivant`, `comestible`, `nbGraines`, et `dureeVie`.

Dans notre simulation, tous les végétaux sont considérés comme vivants à leur introduction. Les végétaux morts sont retirés de l'écosystème, car actuellement, il n'existe pas d'espèces capables de consommer les végétaux déjà morts. Ainsi, les végétaux dont la durée de vie atteint zéro ou ceux qui sont consommés disparaissent de l'écosystème.

Concernant les attributs `empoisonne` et `comestible`, ceux-ci varient selon les espèces. Par exemple, nous avons dans notre écosystème une espèce de végétaux empoisonnés, représentée par la rhubarbe. La rhubarbe est comestible mais sa consommation est toxique, ce qui signifie qu'elle peut être consommée mais elle entraîne un empoisonnement. D'un autre côté, nous avons le bouleau, un arbre non comestible mais non toxique, indiquant qu'il n'est pas destiné à la consommation par les animaux, sans pour autant être nocif. À l'exception de ces cas spécifiques, nos autres végétaux comme la menthe, la laitue, et les asperges, sont comestibles et non empoisonnés.

Enfin, nous avons opté pour une approche où les végétaux n'héritent pas directement des caractéristiques de leur parent. Nous avons estimé que les attributs définis par les fourchettes de l'espèce sont plus déterminants que les traits individuels du parent pour les végétaux.

1.6 Pistes d'amélioration

Parmi les pistes d'amélioration de notre projet, nous pouvons envisager les suivantes :

Ajout de méthodes spécifiques selon les types d'animaux ou de végétaux : Par exemple, les animaux de type `Oiseau` pourraient hériter d'une méthode `voler()`, tandis que les végétaux de type `Arbre` pourraient bénéficier d'une méthode `perdreFeuille()`. L'architecture de l'écosystème a été conçue pour permettre l'intégration aisée de ces fonctionnalités supplémentaires.

Intégration d'une nouvelle classe d'animaux, les Insectes : Ces insectes pourraient jouer un rôle écologique en consommant les végétaux morts, ce qui nous permettrait de maintenir ces végétaux dans l'écosystème après leur mort, jusqu'à ce qu'ils soient consommés par les insectes. Cela ajouterait une couche de complexité et de réalisme à notre simulation.

Utilisation de threads pour la gestion de l'environnement : Comme suggéré dans le bonus du projet, l'emploi de threads pourrait améliorer la manipulation de l'environnement.

1.7 Conclusion

Le projet Écosystème s'est révélé être stimulant intellectuellement, en particulier dans l'élaboration des algorithmes nécessaires pour assurer le fonctionnement adéquat des fonctionnalités demandées. La partie sur la reproduction a été particulièrement exigeante, et il a été très satisfaisant de voir le projet fonctionner correctement après de nombreux efforts. Nous avons également apprécié observer l'évolution significative entre l'écosystème initial développé lors du TP et notre écosystème final, qui est devenu bien plus autonome et dynamique. Ce progrès nous a donné le sentiment de pousser le concept bien plus loin dans le cadre du projet.

Nous avons également apprécié pouvoir travailler dans un premier temps sur le TP et améliorer ensuite les aspects de notre travail qui méritaient d'être changés dans le cadre du projet final. Cela nous a donné l'occasion de directement corriger et améliorer notre travail, ce qui est assez courant dans le cadre de l'enseignement traditionnel présentiel, mais très rare à l'IED où nous recevons très rarement un feedback permettant une application directe.

Concernant la Programmation Orientée Objet (POO), nous avons commencé ce cours sans aucune connaissance préalable en Java ni en POO. Nous avons trouvé l'approche orientée objet particulièrement intéressante et ludique. Java, en tant que langage, nous a paru accessible et a facilité une prise en main rapide. Nous aimerions continuer à explorer la POO et souhaitons particulièrement apprendre à intégrer des interfaces graphiques dans nos futurs projets, pour enrichir notre expérience de développement et celle des utilisateurs finaux.

1.8 Sources

Sources principales : le cours, les TD et les TP

Autres :
- https://plantuml.com/
- https://www.geeksforgeeks.org/java/