Simulateur d'Évolution
Rapport Final - Groupe EF-4

Romain Peyremorte, Pierre Rossi, Maël Berrier, Killian Brisset, Simon Demarty

25 mai 2022

Table des matières

Table des figures

1 Explication du sujet

Notre groupe s'est intéressé à la création d'un simulateur d'évolution générique. Cette application à pour vocation d'être aussi générique que possible en laissant le choix à l'utilisateur de tous les paramêtres.

Il s'agit de choisir des espèces dont les individus évolueront dans un environnement. Notre objectif est de fournir un outil sérieux d'étude et de prévision de l'évolution d'espèces au fil du temps. En outre, ce projet peut aussi servir d'illustration du principe même d'évolution.

2 Principales fonctionnalités

Notre application propose de nombreuses fonctionnalités que nous avons nous même codées.

En effet, elle propse le choix de l'environnement dans lequel les espèces évoluent. Les individus que l'on place dans l'environnement seront aussi personnalisable : aussi bien l'espèce à laquelle ils appartiennent, mais aussi leur nombre. On peut évidemment créer de nouvelles espèces (dont tous les paramêtres sont modifiables).

Ensuite, une fois que les paramêtres précédemment cités sont validés, on peut observer l'environnement et les individus avec leurs caractèristiques. Ainsi, alors que la simulation se déroule, on peut observer les individus intéragir entre eux et avec leur environnement.

3 Architecture de l'application

Nous allons ici décrire l'architecture de notre application. Celle-ci se décompose en plusieurs partie que nous avons d'abord représentées sous la forme d'un diagramme de classe. Ensuite, nous décrirons les paquetages créé pour implanter les différentes fonctionnalités.

3.1 Diagramme de classe

Vous trouverez en Annexe le diagramme UML de notre application sous forme simplifiée pour voir les relations entre les classes (Figure 3), puis en détail avec les diagrammes UML de chaque classe.

3.2 Pacquetages

Notre application se découpe donc comme ci-dessus. Aussi, nous avons naturellement été amenés à organiser notre application en paquetages. Les paquetages sont les suivants :

- AFFICHAGE : c'est la partie dans laquelle on gère l'afficage de tout ce qui est implanté dans les paquetages ci-dessous. C'est aussi avec ce paquetage que l'on lance notre fenêtre d'application.
- ESPECE : on y créé le type espèce et le type individu. Les individus ont un attribut espèce.
- RESSOURCES : On y définit les ressources mangeables par certaines espèces. Ces ressources sont disposées aléatoirement sur l'environnement.
- ENVIRONNEMENT : ce paquetage permet la définition de l'environnement dans lequel évoluent les espèces. Il y a plusieurs caractèristiques d'environnement : des points d'eau, des montagnes...
- SIMULATION : On y gère la gestion de la simulation. Chaque itération y est calculée.
- UTILE : Ce paquetage gère les objects communs à tous les paquetages. Par exemple, nous avions besoin d'un objet position (pour l'environnement, les individus, les ressources...). C'est donc ici que nous les avons définis.
- TEST : c'est le paquetage dans lesquel nous plaçons nos fichier de tests utilisant JUnit. Cela nous a été très utile par la suite.

4 Réalisation du projet

Après avoir choisi notre sujet, il nous a fallu le mettre en place. Aussi nous somme passés par différentes étapes telles que : l'utilisation des méthodes agiles, la répartition des tâches. Nous avons aussi été ammenés à faire des choix de conception, nous menant à des problèmes que nous avons dû résoudre. Nous avons aussi dû revoir nos ambitions pour ajouter des fonctionnalités, et en enlever d'autres.

4.1 Mise en oeuvre des méthodes agiles

Lors de la réalisation de ce projet, nous avons été initiés aux méthodes agiles. Nous allons d'abord décrire l'utilisation que nous en avons faites, puis nous décrirons en quoi celles-ci nous ont été utiles.

Pour commencer, nous avons décomposé notre projet de la vision aux Story. C'est-à-dire que nous avons identifié les besoins de l'utilisateur (il s'agit de la vision) : c'est notre simulateur. Puis, nous avons partagé cette vision en différentes User Stories (Initialisation des paramètres, lancement de la simulation...). Ensuite, nous avons encore divisé notre apréhension de notre application en Stories. Il s'agit là de la création d'espèces, de l'affichage de la simulation...). Enfin, nous en sommes venu aux Epics, c'est-à-dire aux tâches les plus élémentaires : choix du nombre d'invidus, affichages de menu...

L'utilisation des ces notions comme le Backlog Produit, nous a permi d'identifier les points importants du point de vue du client (c'est la valeur métier) et les partie que nous jugions facile ou pas. C'est que l'on nomme l'effort. Cela nous a permi de nous mobiliser principalement sur les points qui avaient une forte valeur métier et un faible effort nécessaire dans un premier temps.

Voici donc le graphe de notre projet (de la vision jusqu'aux Epics) :

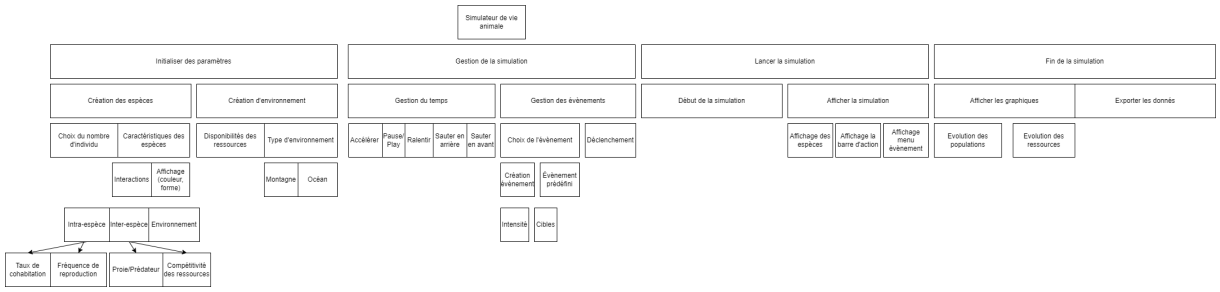

FIGURE 1 – De la vision aux Epics

Aussi, nous avons ensuite organisé notre travail autour d'itérations, lors desquelles nous finissions des petites tâches (les Epics), et nous les validions. De fait, nous avons pu avancer sereinement dans notre projet tout en étant efficace.

Pour conclure ce paragraphe sur ces nouvelles méthodes que nous apprenions à mettre en place, leur utilisation nous a permis une efficacité sans faille. En outre, c'est une méthode de travail qui est généralisable et donc, sa mise en place était enrichissante.

4.2 Choix de conception

Après avoir découper notre application en Stories, cela nous a conféré d'une vision plus claire de sa conception. En effet, au vue des fonctionnalités à implanter, il nous a paru évident de définir les objets individu, espèce, ressource et environnement.

4.2.1 Individu

Concernant, la partie individu, nous avons pris le parti de ne pas faire des reproductions entre plus que 2 individus (même si la fonction qui l'implémente est suffisament générique pour permettre des

reproductions entre trois, quatre, où même n individus).

En outre, dans la partie individu, nous gérons aussi la mise à jour de l'individu considéré avant de mettre à jour l'ensemble de la simulation.

4.2.2 Espèce

Pour la partie Espèce, nous avons décidé de créer une interface "Espece" servant de base à toutes les espèces créées. Ainsi, chaque nouvelle classe d'espèce implémente cette interface.

Par ailleurs, nous avons choisi de créer des classes espèces prédéfinies (aigle, loup, mouton, ...) permettant de tester la simulation dans un premier temps et surtout permettant à l'utilisateur d'avoir des exemples simples pour une première utilisation. Nous avons aussi ajouté une classe "Personnalisé" implémentant "Espece" permettant cette fois-ci à l'utilisateur de pouvoir créer une espèce de son choix en choisissant tous les paramètres.

Nous avons aussi pris le parti de mettre le nombre de bébé minimum, après reproduction, à zéro. Cela évite un surchargement d'individu à certains endroits lors de la simulation.

Enfin, nous avons décidé d'attribuer une couleur à chaque espèce afin de rendre plus claire la simulation pour l'utilisateur.

4.2.3 Ressource

Pour la partie ressource, nous avons décidé de créer une classe Ressource qui servira de base d'héritage pour d'autres ressource plus spécifiques. Nous avons choisi de ne pas faire d'interface ni de classe abstraite afin d'éviter des répétitions de codes ainsi que la possibilité de créer des ressources personnalisables. Ainsi, les ressources particulières telles que Arbre héritent de la classe Ressource avec certains attributs définis, et peuvent modifier les fonctions propres à Ressource.

Il a été aussi décidé de créer deux constructeurs pour chaque ressource : un constructeur dont on indique la position de la ressource et un constructeur où la position de la ressource est aléatoire (dans un interval mis en argument). Nous avons décidé de développer le second constructeur pour que la position de la ressource soit aléatoire lorsqu'on l'a fait apparaitre dans un environnement.

Nous avons pris le parti que lorsqu'une ressource est mangée, elle renvoie un booléan. Cependant, dans des développement possibles, il sera sûrement intéressant de retourner directement la valeur nutritive, surtout dans le cas des classes Arbre et Arbuste où ces derniers ont eux-mêmes des ressources (Pomme et Baie) qui sont mangées avant eux.

4.2.4 Environnement

La création de la classe Environnement ainsi que les relations de cette dernière avec les classes d'environnements pré-définies est justifiée de la même façon que pour Ressource : la volonté d'avoir une classe permettant d'avoir un objet personnalisé, ainsi que de limiter les lignes de code.

Pour différencier les environnements, une représentation linéaire a été choisie autour du taux d'humidité :

Cette représentation est utile pour :

- La différenciation des environnements autrement que par leur nom
- L'apparition de ressource propres à certains environnement ou pas du tout présent dans certains environnements (pas d'algue dans le désert)
- Une variable précise et simple pour la personnalisation d'un environnement.

Cette solution n'est pas optimale pour autant car elle limite les environnements possibles (des oasis par exemple).

Cette variable d'humidité est aussi utilisée pour les points d'eau : plus un environnements est humide, plus il y aura de point d'eau.

Comme cité plus tôt, un environnement est caractérisé aussi par la présence de point d'eau. En effet, une matrice de la taille de l'environnement mémorise en chaque position l'altitude ce qui permet d'avoir des altitudes négatives pour des points d'eau, et des altitudes strictement positives pour des montagnes. L'attribution d'altitudes négatives pour les points d'eau limite malheuresement la présence de crevasse ou de grottes. Nous avons décidé que la génération des points d'eau et des montagnes est fait à l'aide d'une répartition gaussienne autour d'un point.

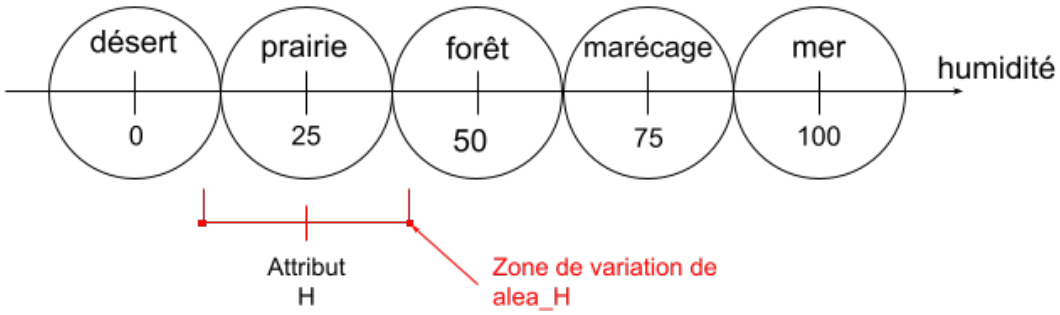

FIGURE 2 – Répartition linéaire des environnements en fonction du taux d'humidité

4.2.5 Simulation

Pour la partie simulation, les choix réalisés concernent la modélisation du comportement des individus. En effet, initialement, nous modélisions les déplacements des individus de manière basique : aléatoire. Ainsi, un loup par exemple, ne pouvait manger un mouton que si il tombait par hasard dessus. Cette représentation étant très peu fidèle à la réalité, nous avons par la suite décidé de l'améliorer : les individus se déplacent dorénavant suivant leurs besoins, ou si ils n'en n'ont aucun, de manière aléatoire. Si un loup a faim et qu'il voit un mouton, il va donc s'en approcher pour le manger. Le deuxième besoin modéliser est le besoin de reproduction. Nous avons choisi que la faim ait une priorité plus importante que la reproduction.

4.2.6 Affichage

Pour la partie affichage, les choix réalisés concernent :

- Le choix des espèces et de l'environnement et la façon de les faire apparaître en faisant apparaître un menu "choix" ou le peut indiquer le nombre et l'espèce à ajouter et où l'on peut choisir l'environnement.
- La possibilité de créer des espèces supplémentaires dans le menu "choix" en cliquait sur le bouton "Ajouter une espèce". Lors de cette action une fenêtre pop-up s'ouvre qui permet de choix les caractéristiques de nos nouvelles espèces.
- La création de la simulation et l'ajout des individus dans la simulation en fonction du nombre sent en cliquant sur le bouton "Lancer" dans le menu "choix". Une fois le bouton appuyé on change de menu pour le menu "évolution" où l'on va pouvoir voir notre simulation.
- La gestion du temps une fois la simulation démarrée. En effet dans le menu "évolution" nous

avons 5 buttons à notre disposition qui permet d'accélérer ou de ralentir la vitesse de la simulation.

- L'affichage des paramètres des espèces dans le menu "évolution" sur le côté droit le nombre et les caractéristiques d'une espèce.

4.3 Répartition des tâches

Afin d'établir un plan d'action efficace, les méthodes agiles nous ont fourni une bonne base de travail. En effet, après la division du travail en tâches élémentaires, nous avons planifier notre travail selon les itérations. En outre, nous avons pris le parti de répartir le travail selon la méthode suivante : chacun s'occuperait d'un paquetage différents (cf. les paquetages précédemments mentionnés).

Aussi, nous avons décomposé notre équipe comme suit :

- Pierre : Espece
- Romain : Environnement et Ressource
- Maël : Simulation
- Simon : Individu
- Killian : Mise en relation des codes précédents & affichage

Cela nous a permi une plus grande efficacité de codage puisque chacun connaissait bien son code, et, d'un autre côté, cela nous a également offert la possibilité de travailler sur plusieurs User Stories.

4.4 Problèmes rencontrés

Un des problèmes que nous avons rencontré lors de la réalisation de notre application a été d'optimiser les calculs de chaque itération. Pour cela nous avons essayé de syntéthiser certaines opérations, en faisant attention de ne pas répéter deux fois le même calcul (par exemple, dans les boucles *for* on élimine les individus d'un ensemble à tester afin de ne pas tester deux fois si un individu se reproduit avec un autre).

4.5 Améliorations futures

Malgré tout ce que nous avons implanté, nous avons été portés par ce projet que nous souhaiterions continuer dans le futur. Aussi avons nous pensé à différents axes d'amélioration qui sont les suivants.

Pour commencer, nous avons donné des valeurs pour chacunes des espèces de manières complétement aléatoire. Cela donne des comportement qui bien que cocasse, ne reflète pas trop ceux que nous aimerions représenter. Cela ne demande pas beaucoup de temps, et relève de l'amélioration mineure, mais cela ajouterai un plus non négligeable.

Notre application ayant pour ambition d'être aussi générique que possible, il manque la possibilité pour l'utilisateur de créer ses propres environnements. Cela est faisable car il suffit de mettre a profit les constructeurs déjà existants. Cependant, ce ne sera pas facilement implémenté car il faudrait modifier la gestion de l'affichage, et cela change l'architecture de notre code.

Une autre amélioration de l'environnement possible est de changer la vision qui différencier les environnements. En effet, les environnements sont différencier de manière linéaire autour de la variable d'humidité, mais cela ne permet pas l'existence, de certains environnement comme une oasis ou une plage. Une amélioration éventuelle serait de changer cette différenciation mais cela demanderait de changer une grosse partie de l'architecture du code.

Pour les ressources, on peut remarquer le manque d'exploitation de la Catégorie fruit. Une amélioration possible serait de mieux exploiter la possibilité d'avoir des plantes produisant des fruits (partiellement implémentée) ou bien de définir de nouvelles ressources comme des carottes par exemple.

Avec la limite précédente, vient la gestion de l'historique de la simulation pour pouvoir "retourner dans le passé". Il s'agirait de stocker les informations de la simulation à des irtervalles de temps régulières (par exemple sauvegarder les 20 dernières itérations) dans des fichiers textes annexes. Cette gestion de l'historique permettrait aussi, via un deuxième fichier où seraient stockées des informations élémentaires, de proposer un bilan de la simulation à la fin de celle-ci avec notamment l'évolution de la population de chaque espèce au cours des itérations, les ressources consommées ...

Aussi, notre simulation ne modélise que de manière très grossière la réalité. Des améliorations concernant cet aspect pourrait être réalisées, tels que l'implémentation du mécanisme de soif (avec le besoi de boire), influant sur les déplacements, ou le mécanisme de fuite (un mouton ne se laisserait pas simplement manger par un loup, le voyant approcher). D'autres interactions intra et interespèces étant aussi possible, autre que simplement la reproduction et la chasse, il serait aussi possible de les implémenter pour augmenter le réalisme de notre application.

Il est aussi notable que notre simulation met du temps à calculer une itération dès lors que le nombre d'individu dépasse les quelques centaines. Nous souhaiterions donc optimiser notre manière de faire les calculs afin d'accélérer notre simulation, pour la rendre fonctionnelle à grande échelle.

Finalement, nous voudrions améliorer la présentation de notre application afin de la rendre plus attrayante graphiquement parlant, bien qu'elle soit déjà parfaite à notre goût (c'est un peu notre bébé).

5 Conclusion

Pour conclure, ce projet a été très enrichissant sur plusieurs plans.

En effet, nous avons été confrontés au travail d'équipe, menant à des différents de compréhension et interprétations du sujet, et donc à des erreurs dans le code. Heureusement, avec ce même travail d'équipe, et grâce à notre communication qui nous a permis de détecter ces erreurs et de nous coordonner, nous avons réussi à produire la version finale et fonctionnelle de notre application.

En outre, cela nous a conféré un aperçu de ce à quoi peut ressembler le travail en entreprise : aussi bien du point de vue de l'organisation du projet (méthodes agiles, choix de conceptions, problèmes rencontrés, solutions proposées...) que de la réalisation de celui-ci.

Aussi, nous avons pris plaisir à réaliser ce projet en équipe.

6 Annexe

6.1 Diagramme UML général

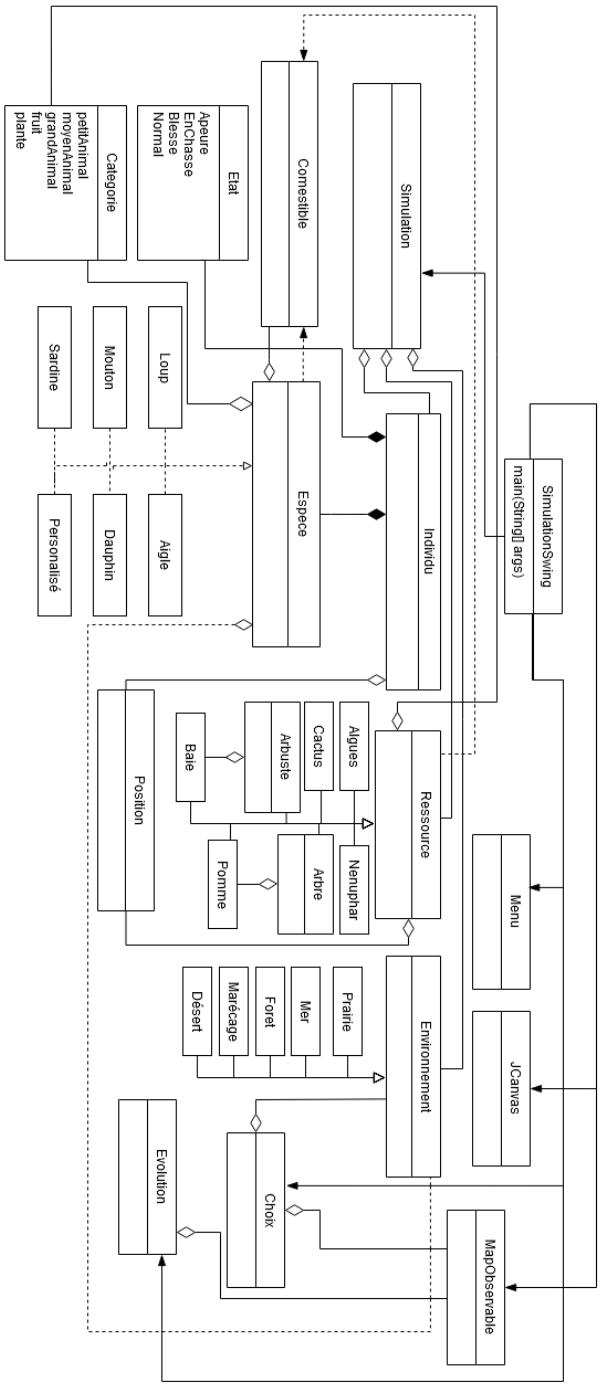

FIGURE 3 – Diagramme UML des relations entre classe

6.2 Package Environnement

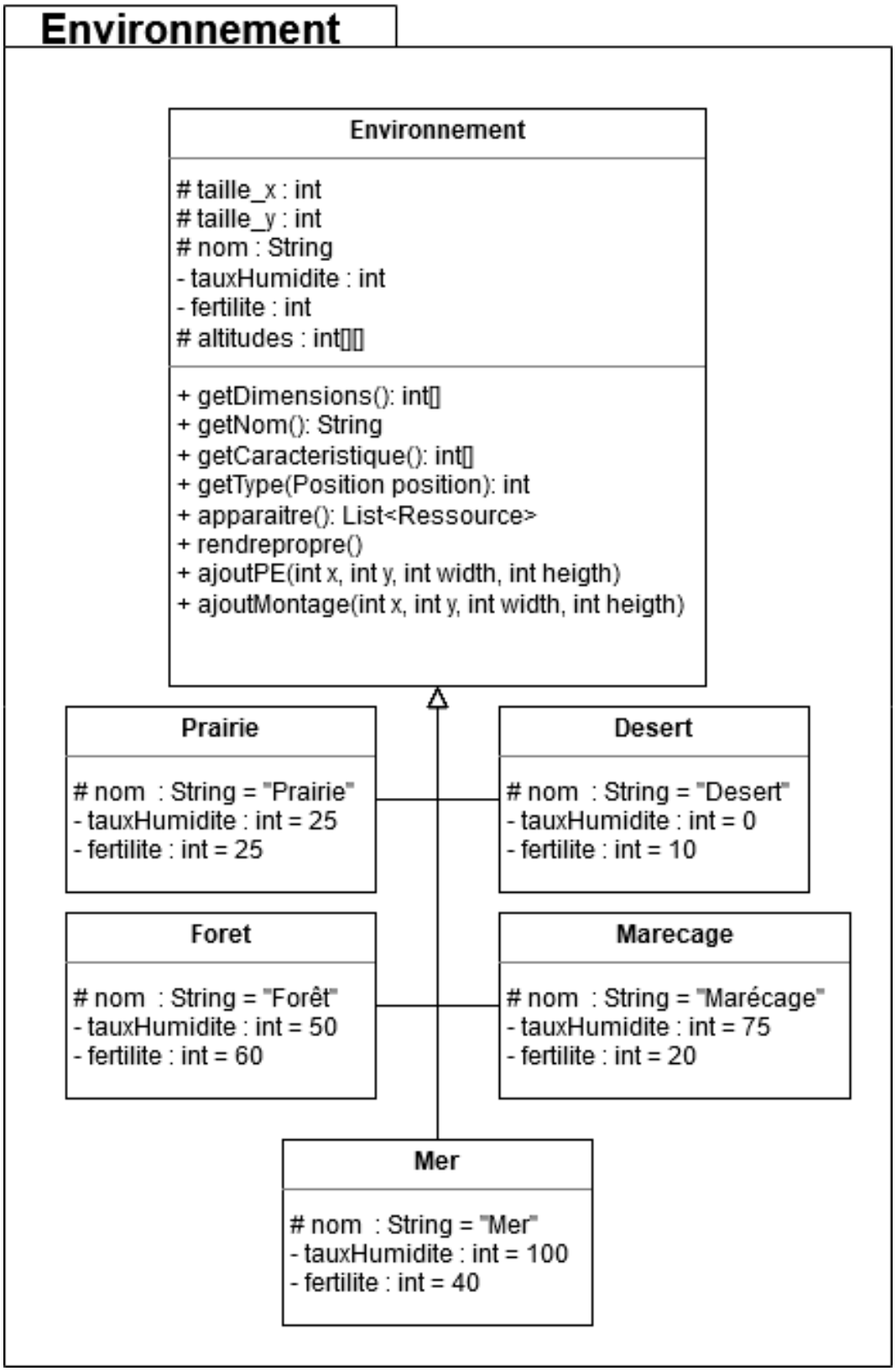

Figure 4 – Diagramme UML du package Environnement

6.3 Package Ressources

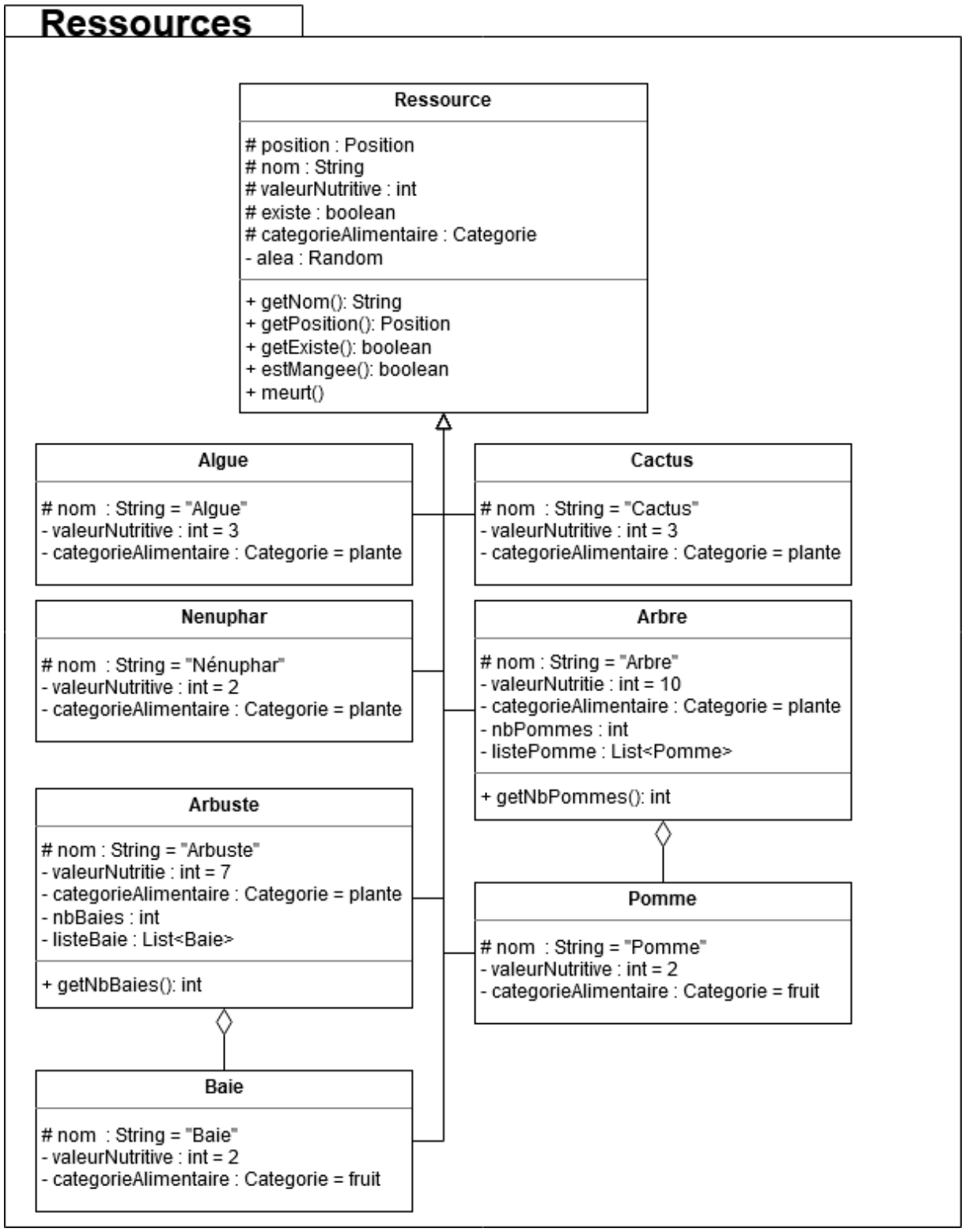

FIGURE 5 – Diagramme UML du package Ressource