Mini-projet Simulation d'un Ecosystème POO JAVA

BARRY Abdoulaye
SECK Aloyse Maliam

Encadreurs : Mr Fouilloux & Mr Chaussard
Année universitaire 2022-2023

19/05/2023

TABLE DES MATIERES

I. INTRODUCTION

Au sein de notre parcours de formation à l'école Sup Galilée, en première année du cycle ingénieur informatique, nous avons suivi un cours de programmation orientée objet qui nous a conduit à mener un projet de développement d'un écosystème simulé. Mettant en œuvre les principes du langage Java, ce projet se décline dans le présent rapport, qui témoigne de notre démarche de conception et d'implémentation d'un système complexe regroupant animaux, végétaux et zones interagissant selon des règles préalablement définies. Ainsi, ce document propose une synthèse approfondie de notre projet en mettant en exergue les choix conceptuels opérés, les fonctionnalités concrétisées ainsi que les résultats obtenus.

Ce rapport s'attache à dévoiler de manière exhaustive les tenants et aboutissants de notre projet de développement d'un écosystème simulé, réalisé avec une approche orientée objet en programmation (POO). Notre objectif primordial fut de concevoir et d'implémenter un système où la présence d'animaux, de végétaux dans des zones s'articule selon des règles rigoureusement établies. Ainsi, nous exposerons avec rigueur les décisions conceptuelles qui ont jalonné notre parcours, les fonctionnalités méticuleusement intégrées au système, ainsi que les résultats obtenus lors des différentes étapes. En outre, nous mettrons en lumière les diagrammes de classes, dépeignant avec précision la structure interne de notre système. Enfin, nous offrirons un compte rendu détaillé des simulations effectuées, permettant d'évaluer l'évolution de notre écosystème au fil du temps.

II. Organisation des fichiers et répertoires

L'organisation des fichiers et répertoires du projet de simulation d'écosystème est la suivante :

```
projet-simulation-ecosystem/
├── src/
│   ├── ecosystem/
│   │   ├── Zone.java
│   │   ├── Animal.java
│   │   ├── Insecte.java
│   │   ├── Oiseau.java
│   │   ├── Mammifere.java
│   │   ├── Carnivore.java
│   │   ├── Herbivore.java
│   │   ├── Sauterelle.java
│   │   ├── Chenille.java
│   │   ├── Biche.java
│   │   ├── Pigeon.java
│   │   ├── Aigle.java
│   │   ├── Lion.java
│   │   ├── AnnimalVolant.java
│   │   ├── Vegetal.java
│   │   ├── Arbre.java
│   │   ├── Vivace.java
│   │   ├── Iris.java
│   │   ├── Pivoine.java
│   │   ├── Chene.java
│   │   ├── ZonePleineException.java
│   │   └── NoWaterException.java
│   ├── view/
│   │   ├── Ecosystem.java
│   │   └── images/
│   │   ├── animal1.png
│   │   ├── animal2.png
│   │   ├── vegetal1.png
│   │   └── vegetal2.png
│   └── main/
│   └── MainNature.java
└── bin/
```

La structure de l'archive comprend un répertoire principal nommé projet-simulation-ecosystem. À l'intérieur de ce répertoire, nous trouvons un répertoire src contenant le code source de notre projet, ainsi qu'un répertoire bin où seront stockés les fichiers binaires générés lors de la compilation.

Dans le répertoire src, nous avons plusieurs sous-répertoires et fichiers Java. Le répertoire ecosystem regroupe les classes liées à l'écosystème et à ses composants tels que les zones, les animaux, les insectes, les oiseaux, les mammifères, les carnivores, les herbivores, etc. Chaque classe correspond à un concept spécifique de l'écosystème. Par exemple,

nous avons les classes Zone, Animal, Insecte, Oiseau, Mammifere, Carnivore, Herbivore, Sauterelle, Chenille, Biche, Pigeon, Aigle, Lion, AnnimalVolant, Vegetal, Arbre et Vivace.

Le répertoire view contient la classe Ecosystem, qui est responsable de l'affichage graphique de l'écosystème. De plus, il contient également un sous-répertoire images qui stocke les images des animaux et végétaux présents dans l'écosystème. Vous pouvez trouver des exemples d'images dans ce répertoire, tels que Lion.png, Biche.png, Iris.png et Chene.png.

Enfin, le répertoire main contient la classe MainNature, qui représente le point d'entrée du programme. C'est à partir de cette classe que l'exécution du programme démarre.

Cette structure de fichiers et répertoires permet une organisation claire du code source du projet, et facilite la gestion des différentes fonctionnalités de la simulation d'écosystème.

III. Diagramme de classes du système

a. Présentation du diagramme de classe

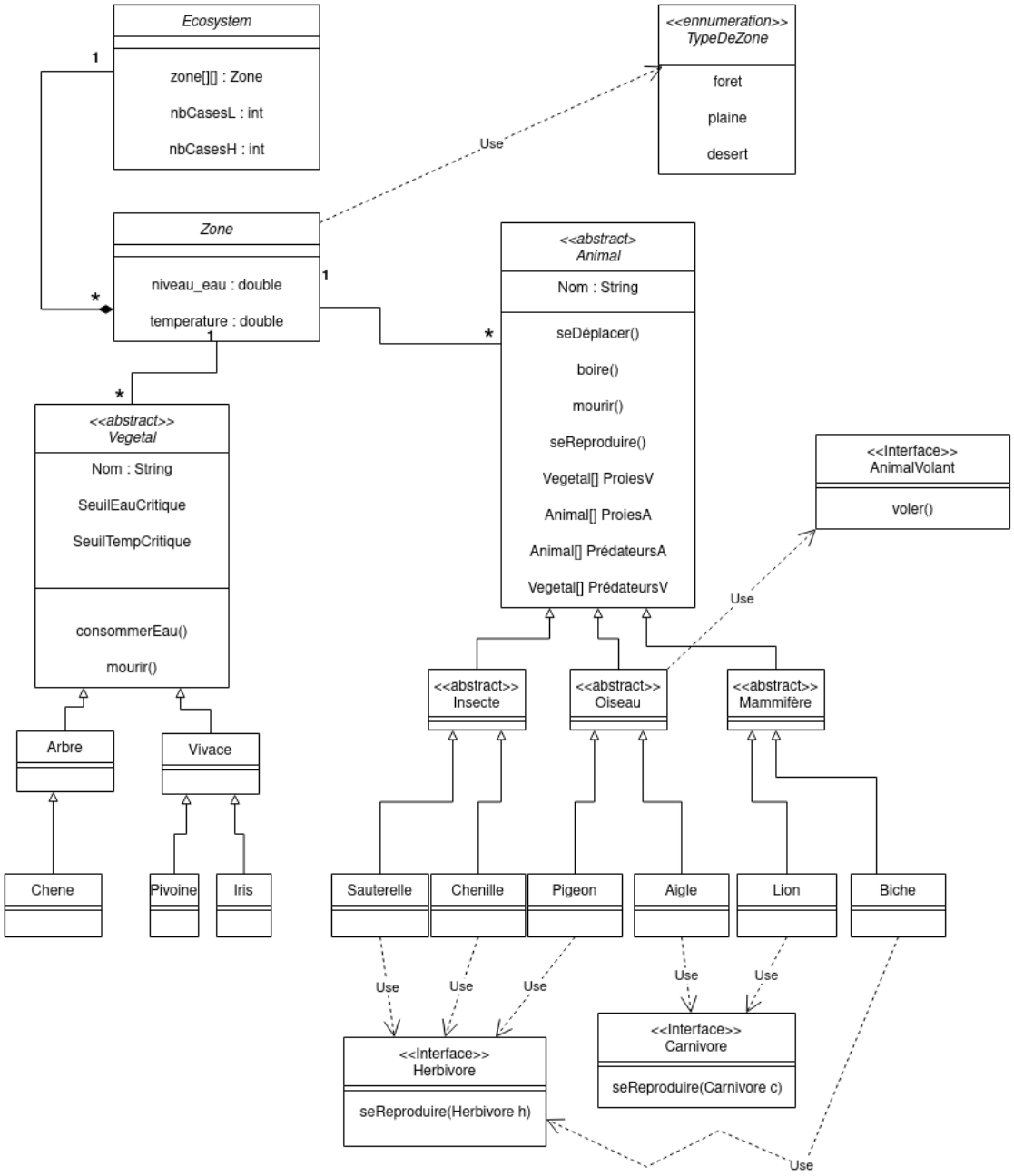


Figure 1 : Diagramme de Classes de l'écosystème

b. Explication des classes et les liens entre elles

Classe Ecosystem

La classe centrale, nommée "Ecosystem", joue un rôle prépondérant dans la simulation de l'écosystème. Elle utilise une matrice bidimensionnelle de type "Zone[][]" afin de représenter les différentes zones qui composent cet écosystème. Chaque zone de cette matrice intègre à la fois des animaux et des végétaux.

Voici une description exhaustive des principales fonctionnalités de la classe "Ecosystem":

La méthode "paintComponent(g)" est employée pour rendre visuellement l'écosystème dans la fenêtre graphique. Elle est invoquée généralement lorsqu'une actualisation ou un redessinage de la zone graphique est nécessaire. Cette méthode se charge de l'affichage des animaux, des végétaux et des autres éléments caractéristiques de l'écosystème dans cette fenêtre graphique.

La méthode "dessinerImage(g, imagePath, x, y, rayon)" permet d'effectuer le dessin d'une image en utilisant des coordonnées spécifiques (x, y) et un rayon donné. Elle est utilisée pour afficher les images représentant les animaux et les végétaux présents au sein de l'écosystème. L'argument "imagePath" représente le chemin vers le fichier image à afficher.

Les méthodes "addAnimal(i, j, animal)" et "addVegetal(i, j, vegetal)" ont pour objectif d'intégrer respectivement un animal et un végétal dans une zone précise de l'écosystème. Les paramètres "i" et "j" désignent les coordonnées de la zone dans la matrice, tandis que les variables "animal" et "vegetal" font référence aux instances des animaux et des végétaux à ajouter.

La méthode "updateAnimaux(i, j)" se charge de la mise à jour des animaux présents dans une zone donnée de l'écosystème. Elle vérifie les conditions requises pour la reproduction des animaux et ajuste les rayons de ces derniers en fonction de leur effectif. Généralement, cette méthode est invoquée régulièrement afin de simuler l'évolution temporelle de l'écosystème.

Les méthodes "reproduireProies(i, j, pourcentageReproduction)" et "reproduirePredateurs(i, j, pourcentageReproduction)" permettent la reproduction des proies et des prédateurs au sein d'une zone spécifique de l'écosystème. Elles utilisent un pourcentage de chance spécifié pour déterminer si la reproduction doit avoir lieu.

La méthode "mettreAJourAnimaux(i, j)" ajuste les rayons des animaux présents dans une zone donnée de l'écosystème en fonction de leur effectif. Cette méthode peut être utilisée pour appliquer des règles spécifiques de croissance ou de diminution des populations animales dans une zone particulière.

Classe Zone

La classe "Zone" assume la responsabilité de la gestion des zones spécifiques au sein de l'écosystème. Voici une description approfondie des principales fonctionnalités offertes par cette classe:

La classe "Zone" contient une liste de végétaux, de type ArrayList<Vegetal>, qui stocke les végétaux présents dans une zone spécifique de l'écosystème. Chaque élément de la liste "vegetaux" est une instance de la classe "Vegetal", représentant un végétal particulier. Cette liste permet une gestion dynamique des végétaux dans la zone, facilitant les opérations d'ajout, de suppression et de manipulation de ces entités végétales.

De même, la gestion des animaux dans la classe "Zone" est réalisée à l'aide d'une liste d'animaux, de type ArrayList<Animal>, qui stocke les animaux présents dans cette zone spécifique de l'écosystème. Chaque élément de la liste "animaux" est une instance de la classe "Animal", représentant un animal spécifique. Cette liste permet une gestion dynamique des animaux dans la zone, simplifiant les opérations d'ajout, de suppression et de manipulation de ces entités animales.

La classe "Zone" possède également un champ "niveauEau", de type double, qui représente la quantité d'eau disponible dans la zone. Ce champ permet de suivre et de mettre à jour l'approvisionnement en eau de la zone, élément essentiel à la survie des plantes et des animaux qui y résident. Le niveau d'eau peut être ajusté en fonction des interactions avec d'autres éléments de l'écosystème ou des conditions environnementales.

La méthode "changementZone()" permet de modifier le type de la zone en fonction de son niveau d'eau. Par exemple, si le niveau d'eau est inférieur à 250, la zone se transforme en désert. Si le niveau d'eau se situe entre 250 et 500, une forêt peut évoluer en plaine. Si le niveau d'eau est compris entre 500 et 1000, une plaine peut se transformer en forêt.

Les méthodes "addAnimal(Animal animal)" et "addVegetal(Vegetal vegetal)" permettent d'ajouter respectivement un nouvel animal ou un nouveau végétal à la zone, en levant une exception ZonePleineException qui nous permet de gérer de manière adéquate les situations où une zone ne peut plus accueillir de nouvelles entités en raison de contraintes de capacité.

Les méthodes "removeAnimal(Class<? extends Animal> animalClass)" et "removeVegetal(Class<? extends Vegetal> vegetalTrouve)" permettent de supprimer respectivement un animal et un végétal de la zone en fonction de leur classe. Elles effectuent une recherche de l'animal ou du végétal correspondant dans la liste, puis les suppriment s'ils sont présents.

Classe Animal

La classe abstraite Animal représente un animal dans l'écosystème. Elle contient plusieurs attributs tels que le nom de l'animal, son âge, son espérance de vie, son rayon, la quantité d'eau consommée et la quantité maximale d'eau consommable. La classe Animal possède également des listes pour stocker les proies végétales, les proies animales, les prédateurs et les prédateurs animaux de l'animal. Ces listes sont respectivement de type ArrayList<Vegetal> et ArrayList<Animal>.

La classe Animal contient des méthodes d'accès telle que getEsperanceDeVie() et getAge(). La méthode vieillir permet d'augmenter l'âge de l'animal de 1. La méthode mourir(Zone zone) est appelée lorsque l'animal meurt. Elle affiche un message indiquant

que l'animal est mort et supprime l'animal de la zone en utilisant la méthode removeAnimal() de la classe Zone.

La méthode abstraite seDeplacer(Ecosystem ecosystem, int i, int j) doit être implémentée par les classes dérivées d'Animal pour définir le comportement de déplacement de l'animal dans l'écosystème. De même, la méthode abstraite manger(Ecosystem eco, int i, int j) doit être implémentée par les classes dérivées d'Animal pour définir le comportement alimentaire de l'animal dans l'écosystème. Les méthodes moveAnimaux() et moveAnimal() sont utilisées pour déplacer les animaux d'une zone à une autre en utilisant les coordonnées des zones. Les méthodes boire(Zone zone) et utiliserEau() permettent à l'animal de boire de l'eau dans la zone donnée et d'utiliser l'eau consommée.

Classe Vegetal

La classe abstraite "Vegetal" joue un rôle essentiel dans la modélisation des végétaux au sein de notre écosystème simulé. Elle contient plusieurs attributs qui définissent les caractéristiques et le comportement des végétaux.

L'attribut "nom" représente le nom spécifique du végétal. Cet attribut permet d'identifier et de différencier les différentes espèces végétales présentes dans l'écosystème.

L'attribut "age" représente l'âge du végétal, exprimé en nombre d'années. Il permet de suivre la progression de l'âge du végétal au fil du temps de simulation.

L'attribut "esperanceDeVie" indique l'espérance de vie maximale du végétal, c'est-à-dire la durée de vie moyenne attendue pour cette espèce végétale spécifique. Il est important de noter que cet attribut peut varier d'une espèce à l'autre.

Le végétal possède également un attribut "rayon" qui représente sa taille, exprimée en unités de mesure appropriées. Ce paramètre permet de définir l'envergure spatiale du végétal dans l'écosystème.

Pour garantir la survie du végétal, nous avons l'attribut "seuilEauCritique" qui indique le niveau d'eau minimal en dessous duquel le végétal ne peut plus survivre. Si le niveau d'eau de la zone dans laquelle se trouve le végétal descend en dessous de ce seuil, des mesures doivent être prises pour assurer son approvisionnement en eau.

De même, les attributs "seuilTempCritiqueMin" et "seuilTempCritiqueMax" représentent respectivement les températures minimale et maximale pour lesquelles le végétal peut survivre. Ces seuils permettent de déterminer les conditions environnementales optimales pour la survie du végétal.

Le végétal consomme de l'eau pour son métabolisme, et cela est géré par les attributs "qteEauConsommee" et "maxEauConsommable". "qteEauConsommee" représente la

quantité d'eau déjà consommée par le végétal, tandis que "maxEauConsommable" indique la quantité maximale d'eau que le végétal peut consommer.

Pour assurer la survie du végétal, des méthodes telles que "consommerEau(Zone Z)" sont implémentées. Cette méthode permet au végétal de consommer de l'eau à partir de la zone spécifiée, en vérifiant d'abord la disponibilité de l'eau et en ajustant les quantités consommées en conséquence. Si le niveau d'eau est insuffisant, une exception de type "NoWaterException" est levée pour signaler le manque d'eau dans la zone.

Le végétal peut également utiliser l'eau consommée pour ses cellules grâce à la méthode "utiliserEau()". Cela reflète le processus biologique par lequel l'eau est utilisée à des fins vitales par le végétal.

En outre, la méthode "vieillir()" permet d'augmenter l'âge du végétal d'une année, en tenant compte de son processus de vieillissement au cours du temps de simulation.

Enfin, la méthode "mourir(Zone Z)" est appelée lorsque le végétal meurt. Elle supprime le végétal de la zone spécifiée en utilisant la méthode "removeVegetal()" de la classe "Zone", et affiche un message indiquant la mort du végétal.

Les Exceptions

Les classes ZonePleineException et NoWaterException sont spécialement conçues pour gérer les exceptions liées à notre projet d'écosystème simulé. Elles sont situées dans le répertoire "ecosystem" aux côtés des autres classes principales telles que Zone, Animal, Vegetal, etc.

La classe ZonePleineException est utilisée pour signaler une exception lorsqu'une tentative d'ajout d'un animal ou d'un végétal dans une zone déjà pleine est effectuée. Cela signifie que la capacité maximale d'entités de la zone est atteinte. L'utilisation de cette exception permet une gestion adéquate des situations où une zone ne peut plus accueillir de nouvelles entités en raison de contraintes de capacité.

D'autre part, la classe NoWaterException est utilisée pour signaler une exception lorsque la quantité d'eau disponible dans une zone est insuffisante pour soutenir la croissance des végétaux présents. Cette exception permet de gérer les cas où les végétaux nécessitent une quantité minimale d'eau pour survivre et se développer. En levant cette exception, nous pouvons prendre les mesures appropriées pour préserver l'équilibre de l'écosystème en fournissant une quantité d'eau suffisante.

IV. Structures de données

En ce qui concerne les structures de données, nous avons majoritairement utilisé des arraylists. Ce sont des structures assez flexibles et faciles d'utilisation ; elles nous ont permis de stocker la liste des végétaux et animaux présents dans une zone. Ce qui nous a facilité les opérations d'ajout ou de retrait d'éléments qui sont déjà intégré dans JAVA.

V. Jeux d'essais

c. Présentation des jeux d'essais

Dans cette partie, nous allons présenter les résultats d'une série de tests visant à illustrer le bon fonctionnement global de notre programme.
Pour commencer, l'une des règles fondamentales à respecter dans cet écosystème est le placement des animaux. En effet, au départ, l'écosystème est divisé en deux zones égales de forêts et de plaines puis les animaux et végétaux sont placés de manière aléatoire dans ces zones.
L'image qui suit illustre ce placement avec les forêts qui sont représentées par la couleur verte et les plaines par la couleur jaune.

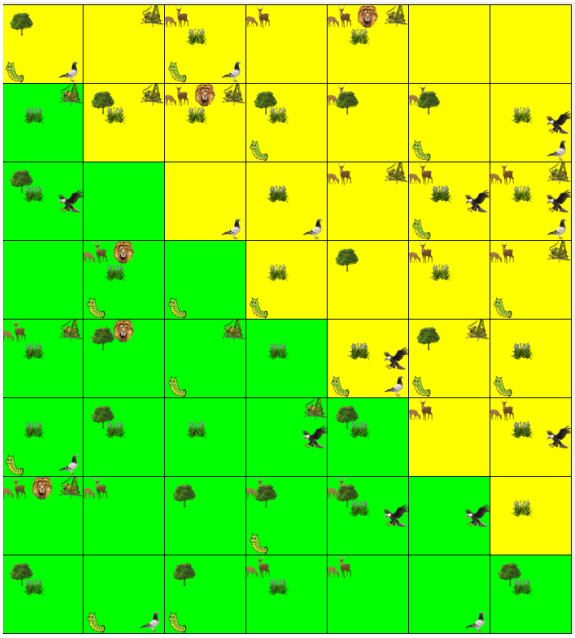
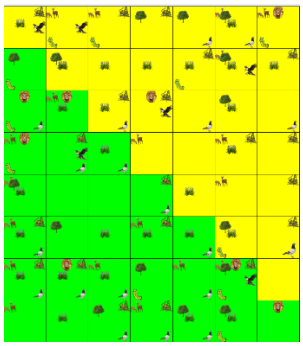
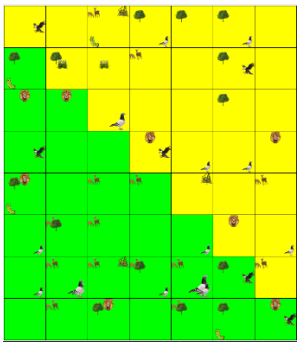


Figure 2 : Placement initial des animaux

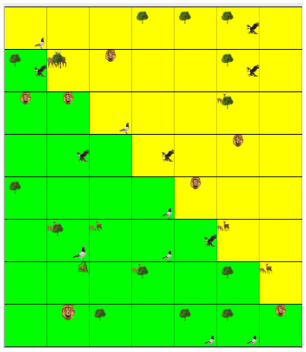
Une autre règle importante est le déplacement des animaux. En effet, au cours des itérations, les animaux doivent être capable de se déplacer de zones en zones pour chercher à se nourrir. Les images qui suivent illustrent l'exécution de la méthode ***moveAnimal()*** qui permet à un animal de se déplacer d'une zone à une autre. L'évolution est représentée sous 4 itérations. On remarque également que les animaux (spécialement les proies) diminuent au fil de ces itérations. Les images illustrent donc la prédation dans l'écosystème.



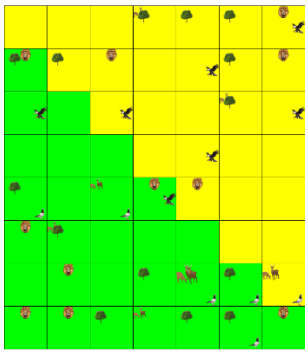
Itération 1



Itération 2



Itération 3



Itération 4

Figure 3 : Déplacement et prédation des animaux

Pour finir, une autre règle de cet écosystème concerne le changement de zone. Le principe est le suivant : une plaine peut se transformer en forêt ou vice versa en fonction des ressources en eau et des températures. Les forêts et plaines peuvent aussi se transformer en désert mais l'inverse est impossible.
L'image qui suit illustre ce changement de zone. Certaines forêts sont devenues déserts ; il en va de même pour les plaines.

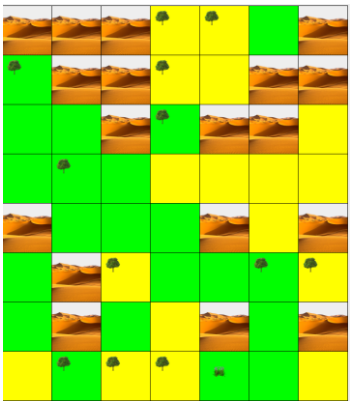


Figure 4 : Changement des zones

VI. Simulation et résultats

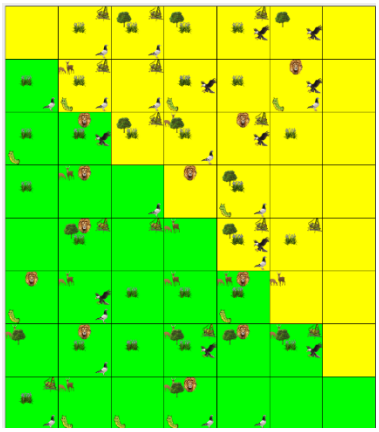
Pour comprendre le fonctionnement de notre écosystème, il est important de lancer une série de simulations qui serviront à prédire ou plutôt observer les différents changements que l'on peut avoir en fonction des instances définies au départ.

Ainsi la simulation consiste à définir des paramètres initiaux, puis à lancer les grandes règles de l'écosystème pour enfin noter l'évolution de ce dernier.

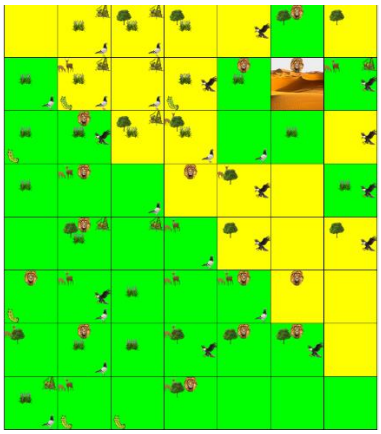
d. Paramètres initiaux des simulations et Résultats obtenus

Dans un premier temps nous allons lancer une simulation avec un pourcentage égal de proies et de prédateurs dans l'écosystème. Leur taux de reproduction est aussi le même pour cette simulation.

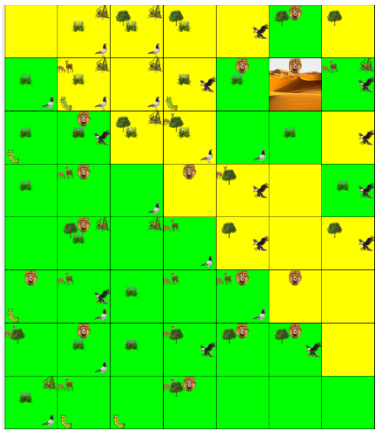
Les images suivantes montrent l'évolution de l'écosystème après un certain nombre d'itérations.



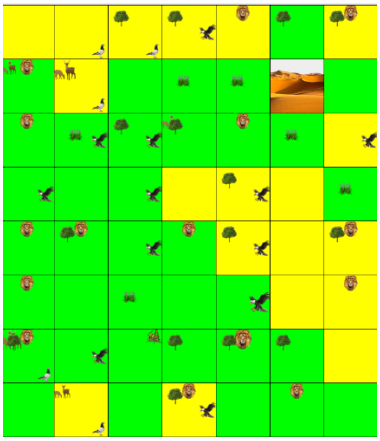
1



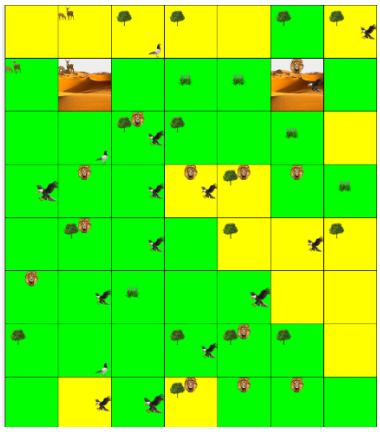
2



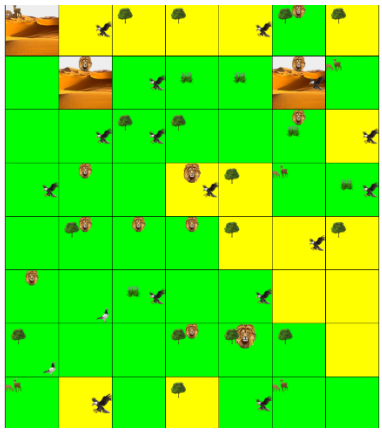
3



4



5



6

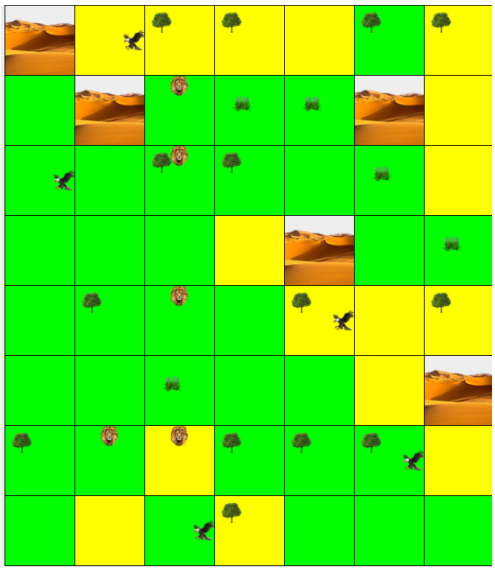
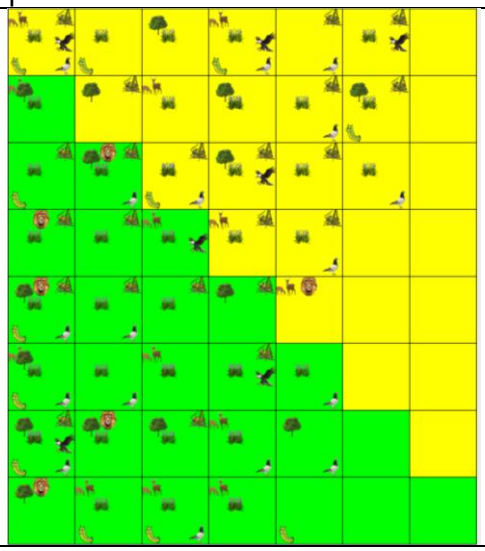


Figure 5 : 1ère simulation (Proies & Prédateurs en quantité égales)

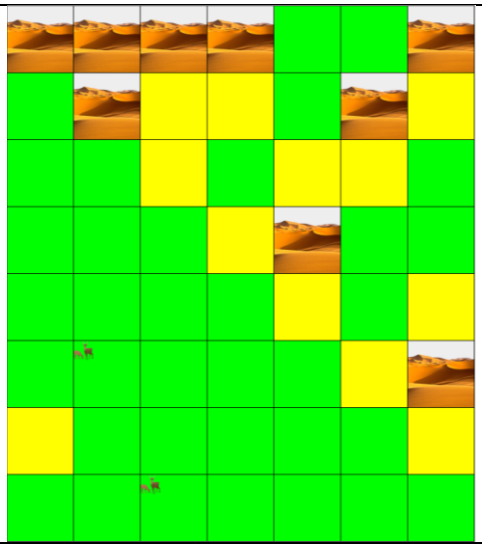
Cette simulation montre une diminution assez rapide du nombre de proies. Au bout de la 6ème itération, il ne reste plus que quelques proies. La dernière image montre l'avant dernière itération dans laquelle nous pouvons remarquer qu'il n'y a plus aucune proie. Les prédateurs finissent par mourir en raison de l'absence de nourriture.

Pour notre seconde simulation, nous choisissons d'attribuer une quantité deux fois plus importante de proies que de prédateurs soit 30% de proies et 15% de prédateurs.
Cette simulation mène à l'extinction de toutes les espèces, cependant elle survient plus lentement car les prédateurs ont une grande quantité de nourriture à disposition.

A la 3ème simulation, nous décidons d'augmenter le pourcentage des proies à 45.
Cette simulation montre que certaines proies réussissent à survivre sans être mangés par leurs prédateurs.



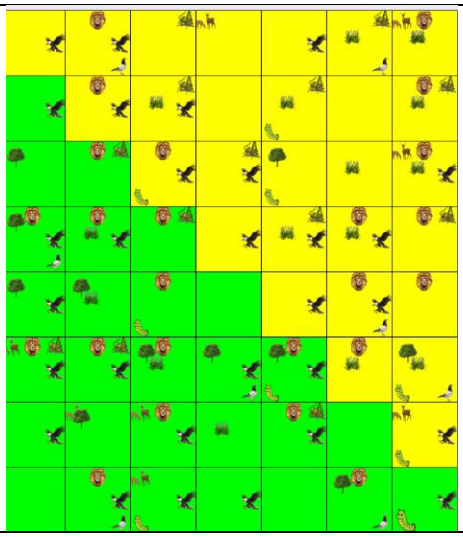
Etape Initiale



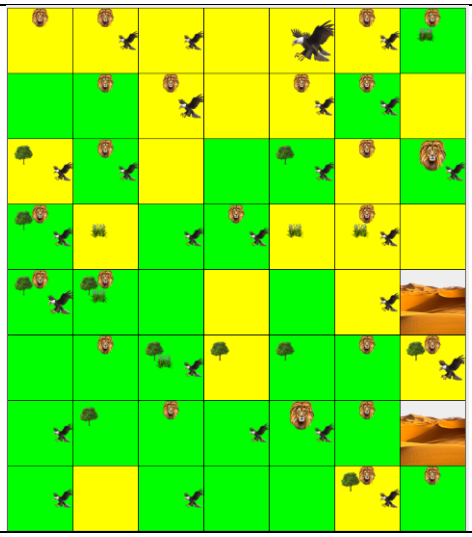
Etape Finale

Figure 6 : 3ème simulation (Proies-45% & Prédateurs-15%)

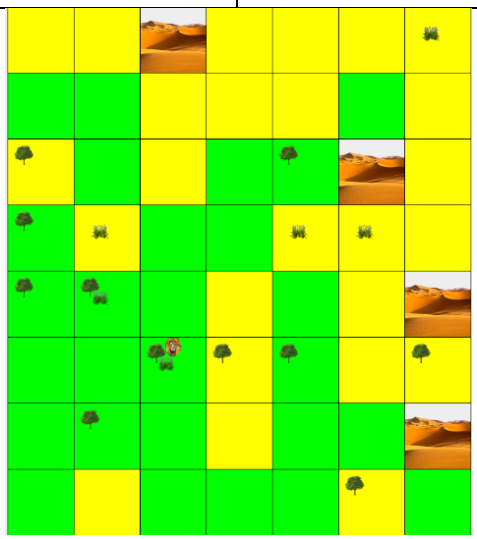
Maintenant, lançons une simulation avec un nombre plus important de prédateurs que de proies.



Etat initial



Etat intermédiaire



Etat final

Figure 7 : 4ème simulation (Proies-15% & Prédateurs-45%)

Comme on pouvait s'y attendre, le nombre de prédateurs étant plus important, les proies disparaissent assez vite ce qui entraîne une extinction de l'écosystème très rapide.

e. Résumé des résultats obtenus par rapport aux questions posées

En résumé, il est difficile d'obtenir un équilibre de l'écosystème où tous les prédateurs et proies vivent sans pour autant que cela entraîne une extinction des espèces.

En effet, attribuer un pourcentage plus élevé de proies peut ralentir l'extinction mais les prédateurs finissent souvent par manquer de nourriture et par conséquent mourir.
Cela peut également avoir l'effet inverse, les proies étant trop nombreuses dans l'écosystème, les prédateurs finissent par s'éteindre parce qu'ils sont en faible nombre et les proies atteignent leur espérance de vie puis meurent.

Il faut noter que pour compléter le fonctionnement de l'écosystème, il faudrait mettre en jeu d'autres règles comme la reproduction des végétaux que nous n'avons pas implémentée, les changements de températures au cours de la simulation et pleins

d’autres caractéristiques d’un écosystème qui sont difficiles à intégrer dans notre programme.

VII. Conclusion

Notre projet de développement d'un écosystème simulé s'est révélé être une expérience profondément enrichissante, bien que non dépourvue de défis. La conception et l'implémentation d'un système d'une telle complexité ont exigé un engagement soutenu et une approche proactive face aux problèmes rencontrés.

Parmi les difficultés auxquelles nous avons été confrontés, nous pouvons citer l'adaptation des dimensions des animaux en fonction de leur nombre dans une zone spécifique. Cette fonction s'est avérée être une étape cruciale pour ajuster dynamiquement le rayon des animaux en fonction de leur effectif au sein de la zone. Trouver une approche efficace pour mettre à jour ces dimensions tout en préservant la cohérence et la lisibilité du code a représenté un défi stimulant auquel nous avons dû faire face avec détermination.

Par ailleurs, nous avons dû surmonter des obstacles liés à la gestion des ressources de l'écosystème, tels que l'approvisionnement en eau des végétaux et des animaux, ainsi que la modélisation des interactions entre les différentes entités. La recherche de solutions

optimales à ces problématiques a exigé une analyse approfondie et une réflexion stratégique.

Malgré ces difficultés, notre travail d'équipe solide et notre engagement indéfectible nous ont permis de surpasser les obstacles et de progresser de manière significative. La collaboration étroite entre les membres de notre équipe, caractérisée par une communication efficace et une entraide mutuelle, a joué un rôle déterminant dans notre réussite.

En conclusion, ce projet de développement d'un écosystème simulé a été une expérience formatrice qui nous a permis d'appliquer nos connaissances en programmation orientée objet et de relever des défis concrets. Nous sommes fiers des résultats obtenus et des compétences que nous avons développées tout au long de ce processus. Nous sommes convaincus que cette expérience constituera une base solide pour aborder des projets plus complexes à l'avenir et pour continuer à nous épanouir en tant qu'ingénieurs informatiques compétents et aguerris.