SIMULATION D'UN SYSTÈME DE PROIES ET DE PRÉDATEURS

Table des matières

INTRODUCTION

L'objectif du projet est d'écrire une simulation de l'évolution d'un système de proies et de prédateurs prenant en compte, entre autres, leur mécanique de déplacement, de prédation et de reproduction.

La simulation s'effectue par pas de temps constant, dans un univers en 2D carré. Initialement, une certain nombre de proies et de prédateurs sont répartis dans cet univers, et à chaque pas de simulation, leur nouvelle situation est calculée et affichée graphiquement par le programme.

Les parties suivantes détaillent les fonctionnalités implantées et les choix de réalisation d'une simulation stabilisée sans extinction des proies ni des prédateurs. Dans la suite, un terme écrit en ***italique et en gras*** désigne une constante modifiable de la simulation.

LISTES DES FONCTIONNALITÉS RÉALISÉES

INITIALISATION

L'espace est représenté par un carré de taille ***SPACE_SIZE*** × ***SPACE_SIZE*** (ici 400 × 400) pixels.

Sur cet espace sont initialement répartis le même nombre de proies et de prédateurs, ainsi qu'un nombre inférieur de plantes. Les prédateurs sont répartis aléatoirement dans le coin bas-droit du carré, les proies aléatoirement dans le coin haut-gauche et les plantes aléatoirement sur l'intégralité de l'espace. Un tel choix a été fait pour minimiser l'effet des conditions initiales de répartition des populations de proies et de prédateurs sur la stabilisation de la simulation : en effet, si par malchance les proies se trouvaient trop bien réparties dans le carré, et ne pouvaient donc pas facilement se reproduire, elles s'éteignaient dès cinquante pas de simulations.

La simulation réalise ***600*** pas de simulation, à une fréquence de ***15*** pas par seconde. Ces valeurs n'entraînent pas de ralentissement ou de saturation de la simulation sur mon ordinateur personnel : pour un ordinateur moins puissant, il est possible de modifier la fréquence de mise à jour de la simulation dans la classe *Simulation*. On peut par exemple prendre une valeur de ***UPDATE_RATE_HZ*** égale à 1 et une valeur de ***DURATION_S*** égale à 600 pour ne mettre à jour la simulation qu'une fois par seconde.

MISE À JOUR DE LA SIMULATION

À chaque pas de simulation, la séquence suivante se produit :

- Les animaux trop vieux (ayant vécu respectivement ***quinze*** et ***vingt*** pas de simulation pour prédateurs et proies) ou ne s'étant pas nourris pendant trop de pas de simulation (respectivement ***huit*** et ***treize*** pas pour prédateurs et proies) meurent.

- Si le pas de simulation est un multiple de ***cinq***, il apparaît autant de plantes que le nombre actuel d'animaux dans la simulation divisé par cinquante. On peut expliquer biologiquement ce choix par le fait qu'une augmentation du nombre d'animaux entraîne une augmentation de l'engrais naturel disponible, et donc du nombre de plantes qui pousse.

- La vitesse (distance parcourue à chaque mise à jour de la simulation) des animaux est mise à jour en fonction de leur état actuel :

    - S’ils ont mangé pour la dernière il y a moins de ***trois*** pas de simulation, leur vitesse est maximale et égale à ***vingt***.
    - S’ils ont mangé pour la dernière il y a plus de ***trois*** mais moins de ***sept*** pas de simulation, leur vitesse est plus faible et égale à la vitesse de base ***dix***.
    - S’ils ont mangé pour la dernière il y a plus de ***sept*** pas de simulation, leur vitesse est minimale et égale à ***cinq***.
    - Si proies ou prédateurs ont mangé une ou plusieurs plantes au tour précédent, leur vitesse déterminée précédemment est divisée par deux.
    - Enfin, si une proie a un prédateur dans son champ de vision, sa vitesse est multipliée par deux. De même, si un prédateur a une proie dans son champ de vision, sa vitesse est également multipliée par deux.

- Les animaux se déplacent en suivant la spécification suivante :
    - Les proies fuient le plus proche des prédateurs dans leur champ de vision, ou avancent aléatoirement suivant un mouvement Brownien si aucun prédateur n'est dans leur champ de vision. Ce champ de vision est un carré de demi côté égal à deux fois la vitesse de la proie. Celui-ci s'arrête sur les bords de l'espace.

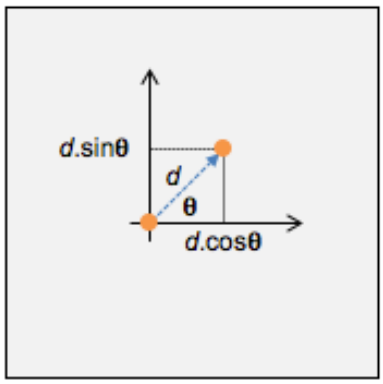


Illustration d'un déplacement Brownien de vitesse d et d'angle θ

    - Les prédateurs visent la plus proche des proies de leur champ de vision, ou continuent d'avancer en ligne droite dans la même direction si aucune proie n'est dans leur champ de vision. Ce champ de vision est un carré de demi côté égal à deux fois la vitesse du prédateur. Celui-ci s'arrête également sur les bords de l'espace.

- La simulation se connecte au serveur et échange des données concernant les animaux sortant du carré et entrant dans le carré selon le protocole défini ci-après. Dans le cas présent d'un serveur d'interconnexion simple ne gérant qu'une simulation, les proies et prédateurs sortant du carré réapparaissent de l'autre côté du même carré :

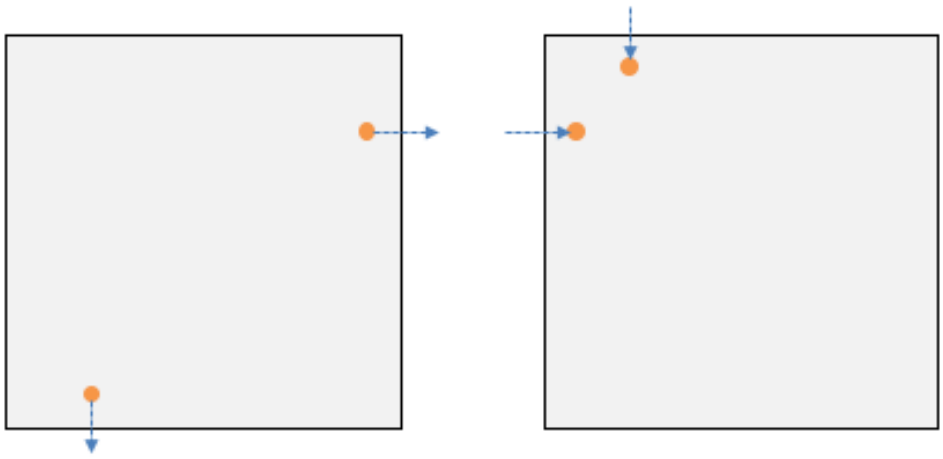


Illustration d'un animal sortant de l'espace

- Les prédateurs mangent les proies et les plantes se trouvant dans leur périmètre de prédation. Ce périmètre est un carré de demi côté ***dix***, qui continue de l'autre côté de l'espace par effet torique. Il n'y a pas de limite sur le nombre d'éléments qu'un prédateur peut manger par tour. Ensuite, les proies mangent les plantes se trouvant dans leur périmètre, qui est également un carré de demi côté ***dix*** qui continue de l'autre côté de l'espace par effet torique.

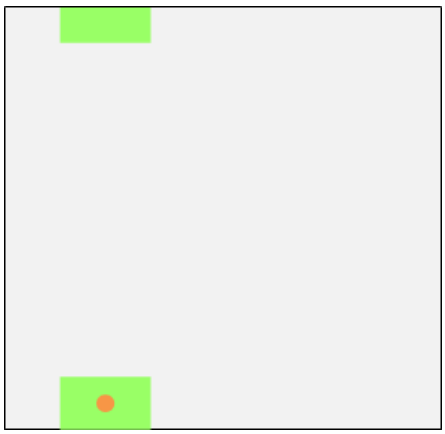


Illustration du périmètre de prédation d'un animal sur les bords de l'espace

- Les probabilités de reproduction des proies et des prédateurs sont mises à jour : elles sont modélisées par des exponentielles décroissantes du type $e^{-a_p \cdot N_p}$ avec a_p un paramètre différent pour les proies et les prédateurs et N_p le nombre de proies/de prédateurs. Un tel choix a été fait afin de réguler les explosions de population de proies et de prédateurs ; il s'explique biologiquement par une compétition plus forte pour la reproduction quand la population est plus importante, et donc un taux de reproduction plus faible en moyenne.

- Les prédateurs se reproduisent, puis les proies. Chaque couple n'engendre qu'un enfant. Pour qu'une reproduction ait lieu, les conditions suivantes doivent être réunies :

  - Un animal ne peut se reproduire que s'il a déjà vécu ***deux*** pas de simulation.
  - Un animal ne peut se reproduire qu'avec un animal se trouvant dans son périmètre de reproduction. Ce périmètre est un carré de demi côté ***dix***, qui continue de l'autre côté de l'espace par effet torique, comme pour la prédation.
  - Aucun des deux animaux impliqués dans la reproduction ne doit déjà s'être reproduit lors de ce pas de simulation.
  - S'il s'agit d'un couple de proies, aucun prédateur ne doit se trouver dans le champ de vision de ces proies. Ce champ de vision est comme précédemment un carré de demi côté égal à deux fois la vitesse de la proie, qui s'arrête sur les bords de l'espace.

- Les coordonnées de chaque cercle représentant proies, prédateurs et plantes sont mises à jour pour l'affichage graphique.

MODE D'EMPLOI DU SERVEUR ET DU CLIENT

Le lancement de la simulation est très simple. Après s'être placé dans le dossier M13/bin/, écrire dans deux terminaux différents :

```
java m13.SimpleServer
```

```
java m13.GraphicalDisplay
```

(il faut lancer le serveur avant de lancer la simulation)

La simulation est stable pour les paramètres de base utilisés (taille de la simulation, nombre initial de proies, de prédateurs et de plantes, valeur des différentes constantes) et avec des versions non modifiées des classes SimpleServer et GraphicalDisplay. Il est cependant possible de modifier toutes les ***constantes*** dans la classe Simulation. Le nombre d'éléments est lui modifiable à la ligne 50 de la classe GraphicalDisplay :

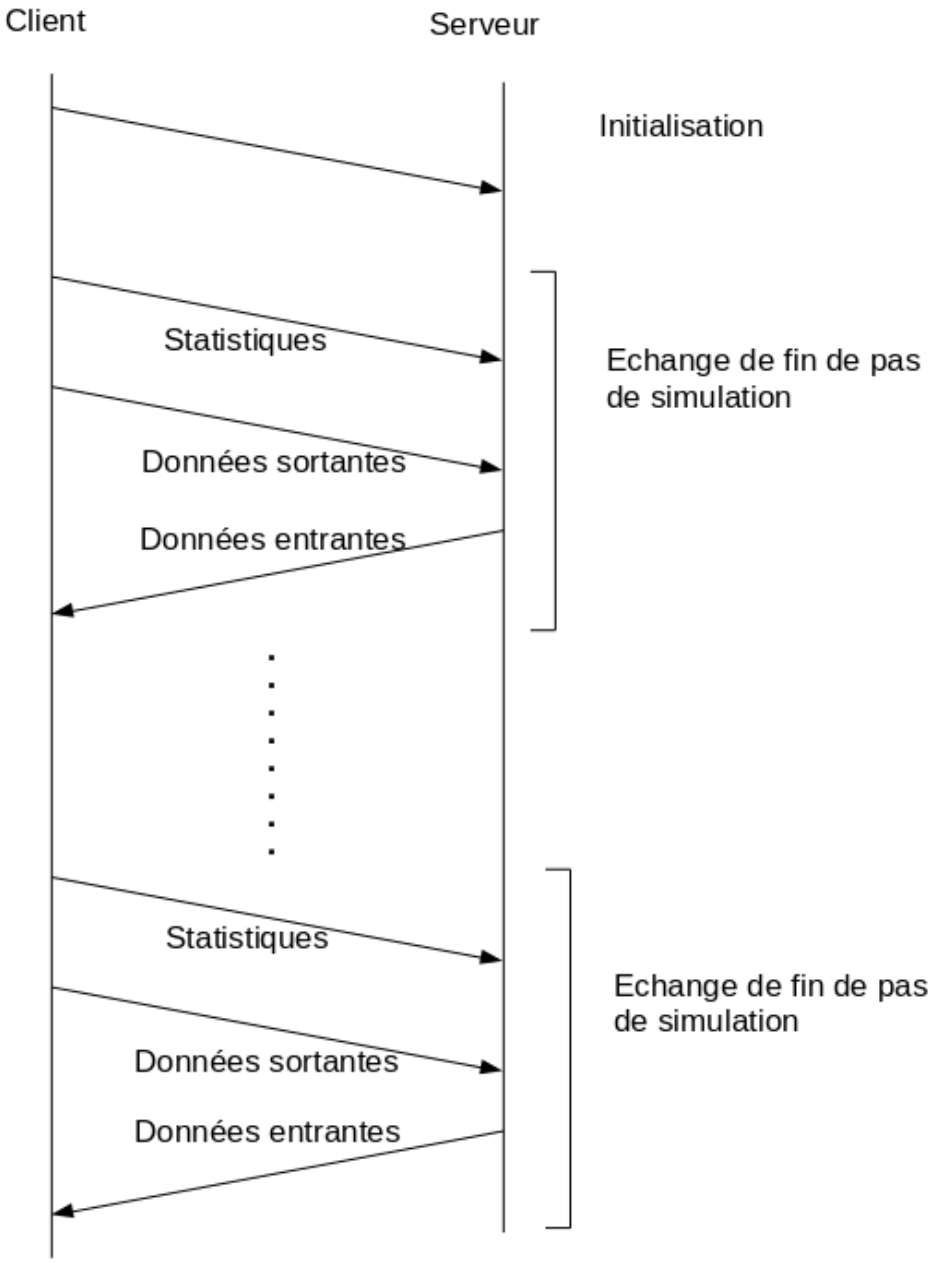
```
simulation = new Simulation(220);
```

Ce nombre initial d'éléments doit cependant rester supérieur ou égal à 22 pour ne pas entraîner d'erreur dans l'exécution de la classe Simulation.

PROTOCOLE DE COMMUNICATION ENTRE LE SERVEUR ET LE CLIENT

Le protocole client-serveur est tel que spécifié sur le LMS et implanté dans une version non modifiée de la classe SimpleServer fournie.

Le protocole est basé sur une connexion TCP sur le port 6789. Une session de simulation consistera en une initialisation, puis une suite d'échanges de données de simulation :



- L'initialisation est un envoi par le client de deux paramètres au serveur. Les paramètres

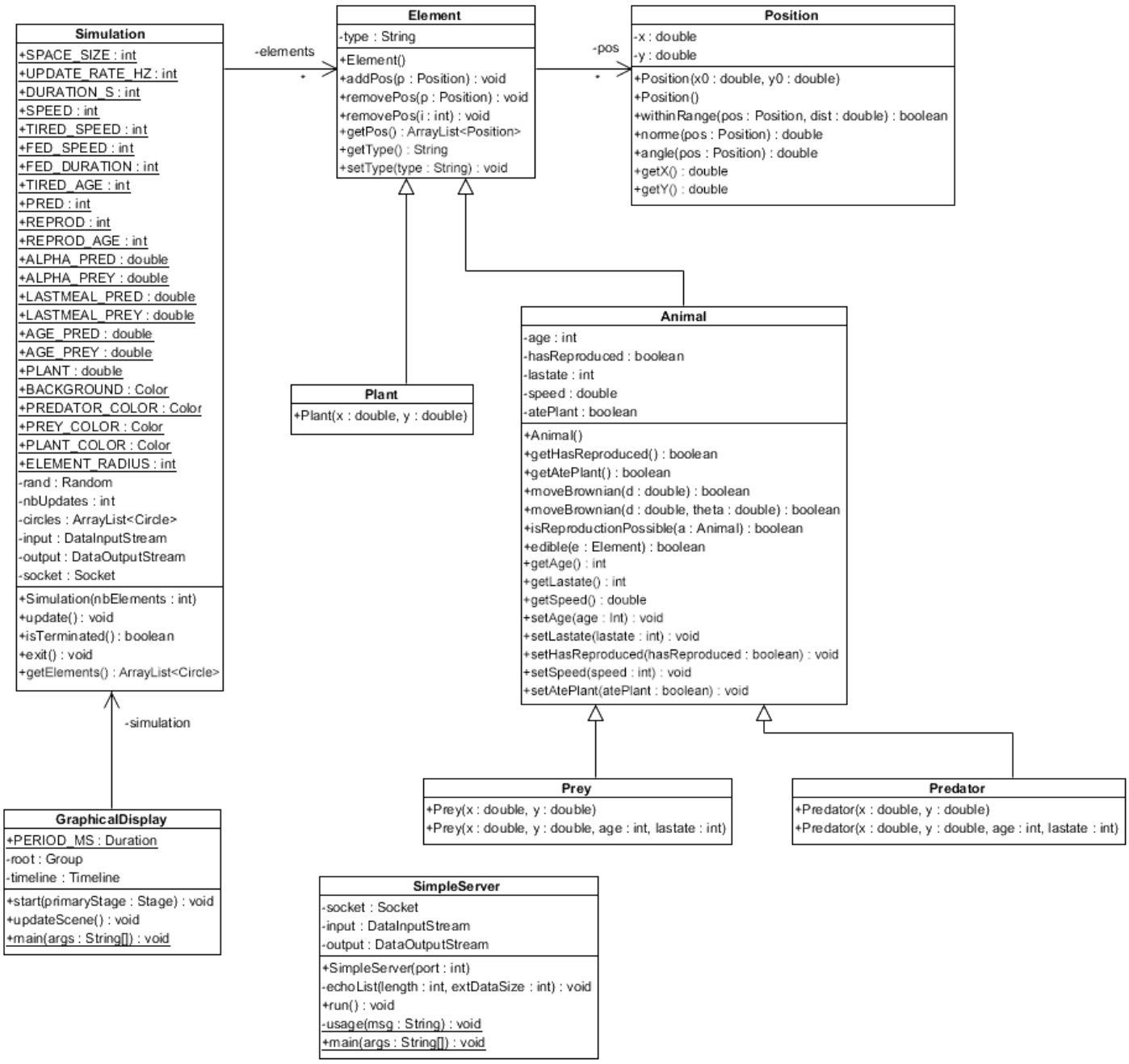
d'initialisation sont deux entiers associés respectivement aux tailles (en octets) des données additionnelles des proies et des prédateurs. Ces données additionnelles sont des données envoyées par le client en plus de et immédiatement après la position de chaque proie/prédateur. Les deux entiers envoyés ici sont 8, car pour chaque animal, on envoie au serveur deux entiers : son age et la dernière fois que celui-ci s'est nourri.

- Dans un pas de simulation, un échange de données de simulation consiste en :
    - D'abord un envoi par le client de statistiques de simulation. Il s'agit de deux entiers indiquant respectivement le nombre de proies et de prédateurs qui restent à l'intérieur du carré de simulation du client.
    - Puis le client communique les informations sur les animaux qui sortent du carré de simulation, selon le format suivant :
        - le nombre de proies qui sortent (nombre au format entier)
        - le nombre de prédateurs qui sortent (format entier)
        - la suite des données de chaque proie qui sort (suite éventuellement vide si aucune proie ne sort):
            - pour chaque proie le client transmet d'abord l'abscisse x dans le repère orthonormé de sa simulation, puis l'ordonnée y. Dans le repère orthonormé de la simulation du client, tout animal qui sort a au moins une des coordonnées en dehors de l'intervalle [0, 1].
            - puis les données additionnelles de la proie sous forme d'un tableau de bytes.
        - la suite des données de chaque prédateur, selon le même format que pour les proies.
    - Enfin le serveur répond au client avec des informations au même format :
        - le nombre de proies qui entrent dans le carré de simulation du client (nombre entier)
        - le nombre de prédateurs qui entrent (entier)
        - la suite des données de chaque proie qui entre (éventuellement vide si le nombre indiqué précédemment était 0). Les coordonnées sont toutes les deux dans l'intervalle [0,1], et le serveur fournit des données additionnelles sous forme de tableau de bytes de taille conforme au paramètre spécifié par le client pendant l'initialisation.
        - la suite des données de chaque prédateur, selon le même format que celui des proies.

On notera que l'échange de données entre le client et le serveur s'effectue plutôt sous forme de ping-

pong : chaque donnée envoyée par le client est instantanément renvoyée par le serveur. Ceci est conforme à l'implémentation du SimpleServer fourni mais pas exactement en accord avec la spécification fournie sur le LMS.

DIAGRAMME UML DU PROJET



Le projet comporte en tout 9 classes, dont 2 classes exécutables (GraphicalDisplay et SimpleServer), réunies dans le package m13. Les classes Animal et Plant héritent de la classe Element, et les classes Prey et Predator héritent de la classe Animal. GraphicalDisplay dépend de Simulation par son attribut simulation de multiplicité 1, Simulation dépend de Element par son attribut elements de multiplicité *, et Element dépend de Position par son attribut pos de multiplicité *.

SIMULATION

Simulation possède dix-huit constantes publiques, deux attributs statiques publics (***ALPHA_PRED*** et ***ALPHA_PREY***), sept attributs privés et cinq méthodes publiques. *Rand* est un générateur de nombres aléatoires utilisé lors de l'initialisation de la simulation, *nbUpdates* est un compteur du nombre de pas de simulation réalisés, *circles* est la liste des cercles à afficher par GraphicalDisplay et *input*, *output* et *socket* sont utilisés pour communiquer avec le serveur.

Le constructeur *Simulation* permet d'initialiser la simulation, *update* permet de réaliser un pas de simulation selon l'enchaînement décrit précédemment, *isTerminated* permet de vérifier si la simulation est terminée, *exit* de fermer la simulation et *getElements* d'obtenir la liste de cercles à afficher.

GRAPHICALDISPLAY

La classe GraphicalDisplay est telle que fournie sur le LMS. Elle possède une constante publique, deux attributs privés et deux méthodes publiques. (sans compter la méthode *main*) ***PERIOD_MS*** est la conversion de ***UPDATE_RATE_HZ*** en ms, alors que *root* et *timeline* sont utilisés pour l'affichage graphique.

Start permet de créer la simulation, de mettre en place les appels à *updateScene* automatiques et d'afficher la fenêtre de visualisation alors qu'*updateScene* permet de réaliser un pas de simulation.

SIMPLESERVER

SimpleServer est également tel que fourni sur le LMS. La classe possède trois attributs privés, deux méthodes publiques (plus le *main*) et deux méthodes privées. *Socket*, *input* et *output* permettent de communiquer avec le client.

Le constructeur *SimpleServer* permet de créer un serveur, *echoList* permet de renvoyer les données concernant chaque proie et chaque prédateur au serveur (position + données additionnelles), *run* permet de lancer le serveur et *usage* permet d'afficher un message d'erreur.

ELEMENT

Element est la super-classe de tous les éléments de la simulation. Ses attributs sont donc communs entre tous les éléments utilisés dans la simulation. La classe possède deux attributs privés et sept méthodes publiques. *Type* représente le type d'élément, afin de pouvoir différencier les plantes des proies et des prédateurs et les proies et les prédateurs entre eux quand on parcourt la liste des éléments dans Simulation. *Pos* est une ArrayList de positions qui représente toutes les positions qu'un élément a prises depuis sa création. Seul le dernier élément de la liste (position actuelle) est conservé lorsqu'un animal sort et rentre dans l'espace.

Le constructeur *Element* permet de créer un élément, *addPos* d'ajouter une position à sa liste de positions, et les deux versions de *removePos* d'enlever une position de sa liste de positions, avec l'indice ou la position elle-même. Les méthodes *get* permettent d'avoir accès à la valeur des attributs et *setType* permet de modifier le type d'un élément.

POSITION

La classe Position permet de représenter la position des divers éléments de la simulation dans le carré de simulation. Position possède deux attributs privés et sept méthodes publiques. Ses deux attributs x et y représentent respectivement les coordonnées x et y d'un point dans le plan cartésien. Ils sont tous les deux compris entre 0 et ***SPACE_SIZE***.

Les deux constructeurs *Position* permettent de créer une nouvelle position. (si aucun paramètre n'est passé au constructeur, la position créée est (0, 0)) *WithinRange* est la méthode utilisée pour déterminer si une position appartient au périmètre de prédation ou de reproduction carré d'une proie ou d'un prédateur. *Norme* détermine la norme du vecteur (this – pos), sans tenir compte du fait que le carré de simulation se replie sur les bords. La méthode est utilisée pour déterminer la proie/le prédateur le plus proche dans le champ de vision d'un animal. *Angle* renvoie l'angle entre le vecteur Ox et le vecteur (this – pos), et permet de déterminer la direction à suivre pour un animal pour suivre un autre ou le fuir, ou pour continuer en ligne droite. Les méthodes *getX* et *getY* permettent d'avoir accès aux valeurs de x et y.

ANIMAL

La classe Animal est une sous-classe de la classe Element et est la sur-classe de Prey et Predator. Cette classe permet de représenter les animaux, c'est-à-dire les éléments mobiles de la simulation. La classe possède cinq attributs privés propres et deux attributs hérités d'Element, sept méthodes publiques propres et quatre méthodes héritées. *Age* représente le nombre de pas de simulation vécus par un animal, *hasReproduced* est un booléen valant true si l'animal s'est reproduit dans ce pas de simulation et false sinon, *lastate* représente le nombre de pas de simulation écoulés depuis la dernière fois que l'animal s'est nourri, *speed* représente la distance parcourue par l'animal à chacun de ses déplacements et *atePlant* est un booléen valant true si l'animal a mangé une plante au tour précédent et false sinon.

Le constructeur *Animal* permet de créer un nouvel animal, les méthodes *get* permettent d'avoir accès aux valeurs des différents attributs et les méthodes *set* de modifier ces valeurs. Les deux méthodes *moveBrownian* permettent de déplacer un animal en ajoutant une position à sa liste de positions, l'une tirant aléatoirement l'angle dans l'intervalle $[0, 2\pi]$ et l'autre ayant un angle en paramètre. *IsReproductionPossible* vérifie si deux animaux de la même espèce réunissent toutes les conditions pour se reproduire (cf. déroulé d'un pas de simulation), et *edible* vérifie si un prédateur peut manger une proie/une plante ou si une proie peut manger une plante.

PREY & PREDATOR

Ces deux classes héritent de la classe Animal et n'ont que deux constructeurs propres chacun. Quand *age* et *lastate* ne sont pas spécifiés à la création d'une proie ou d'un prédateur, ils sont initialisés à 0.

PLANT

La classe Plant hérite d'Element et ne possède qu'un constructeur. Celui-ci permet de créer une nouvelle plante à la position (x, y).

CONCLUSION

La simulation ainsi réalisée intègre toutes les fonctionnalités de base demandées, ainsi que les enrichissements suivants : gestion de l'âge, du champ de vision et de la fatigue des animaux, ainsi que la gestion des ressources naturelles.

Quelques modifications vis-à-vis de la simulation de base se sont cependant trouvées être nécessaires afin de stabiliser la simulation, qui est très dépendante de deux facteurs : la répartition initiale des éléments dans l'espace et la probabilité de reproduction des animaux. L'influence de la répartition initiale a pu être limitée simplement par un groupement des proies d'un côté de l'espace et des prédateurs de l'autre à l'initialisation, mais maîtriser l'influence de la probabilité de reproduction requit plus de modifications. En effet, il a fallu séparer les probabilités de reproduction des proies et des prédateurs, modéliser une fonction modifiant la probabilité de reproduction de chaque espèce en fonction du nombre total de représentants de cette espèce et tester empiriquement pour quelle valeur du paramètre de cette fonction la simulation se stabilisait.

Un enrichissement supplémentaire qui pourrait être réalisé serait l'implantation d'un serveur d'interconnexion multiple permettant de gérer plusieurs clients à la fois. Les différentes simulations s'échangeraient donc des animaux lorsque ceux-ci sortiraient de l'espace de leur simulation.