Systèmes multi-agents.

Johan DUFAUX et Romain SOMMERARD

9 février 2016

https ://github.com/rsommerard/agentjandk

Table des matières

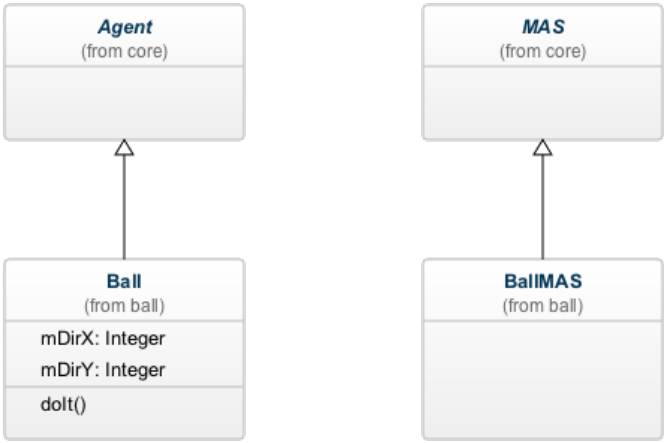
1 Core

Notre projet est composé de 4 packages.
— Core : contient les classes communes aux différents projets.
— Ball : contient les classes du projet Ball.
— Wator : contient les classes du projet Wator.
— Pacman : contient les classes du projet Pacman.

2 Bille

2.1 Architecture

Le package ball contient une classe Ball qui hérite de la classe Agent et une classe BallMAS qui sert à initialiser les billes.



2.2 Comportement

Une bille possède une direction en X et une direction en Y. A chaque tour :
— La bille récupère son voisinage de Moore.
— La bille observe la position sur laquelle elle désire aller.
— Si position est libre, la bille se déplace.
— Si la bille rencontre une autre bille, alors elle inverse sa propre direction et reste sur place pour ce tour.
— Si la bille rencontre un mur et se déplace en diagonale alors elle rebondit à 90°. Si elle se déplace verticalement ou horizontalement, elle rebondit à 180°.

2.3 Utilisation

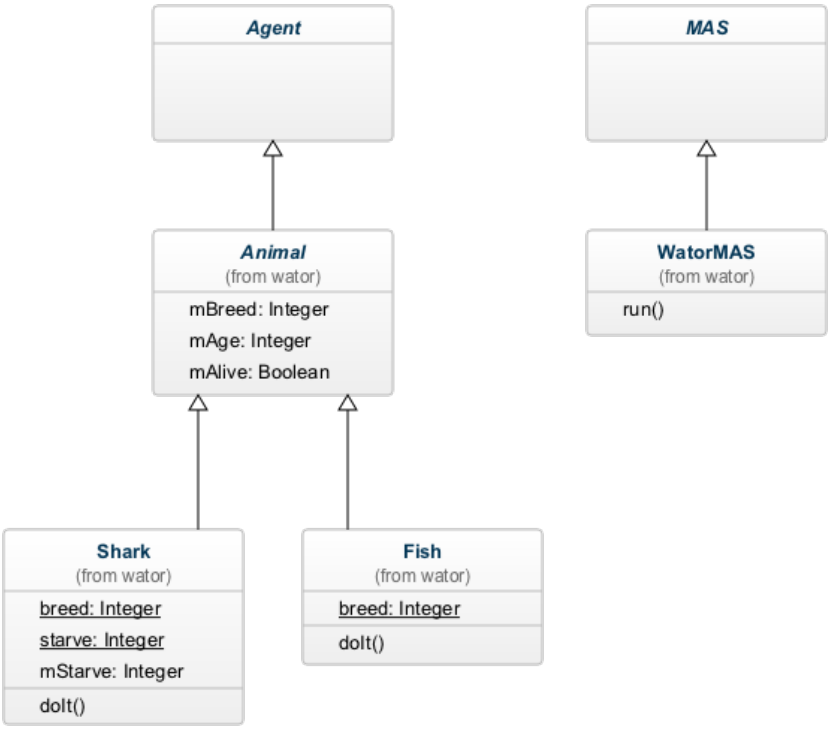
Pour tester le programme, il est possible de lancer directement le jar. Celui-ci est configuré par défaut avec les paramètres qui vont bien. Il est possible de spécifier les paramètres comme indiqué dans la documentation suivante.

```
Usage: java -jar ball.jar
  [-nbAgent <NumberOfAgent>] (default: 400)
  [-nbTurn <numberOfTurn>] (default: 1000000)
  [-width <width>] (default: 120)
  [-height <height>] (default: 120)
  [-agentSize <sizeOfAgent>] (default: 5)
  [-speed <speed>] (default: 50)
  [-seed <seed>] (default: random)
  [-grid] (default: false)
  [-equity] (default: false)
  [-toric] (default: false)
```

3 Wator

3.1 Architecture

Les classes du package Wator héritent des classes du package Core. Une nouvelle vue (CsvView) a été créée pour génerer le fichier CSV.
Les classes Shark et Fish représentent respectivement les poissons et les requins. Ces classes héritent d'une classe Animal qui hérite elle-même d'Agent.
Une nouvelle classe Système multi-agent WatorMAS est nécessaire pour gérer la reproduction des poissons et des requins. Elle étend de MAS qui contient la méthode run.



3.2 Comportement

3.2.1 Fish

Le comportement d'un poisson est le suivant :
— Son age et son taux de reproduction s'incrémentent.
— Il recupère son voisinage de mooore.
— Si le voisinage est plein, il ne fait rien et arrête son tour.
— Sinon Si sa reproduction a atteint le seuil, il se reproduit et un nouveau poisson né sur une case aléatoire à coté et le taux est remis à zéro.
— Il tente ensuite de se déplacer aléatoirement sur une case libre dans le voisinage.

3.2.2 Shark

Le comportement d'un requin est le suivant :
— Sa faim, son taux de reproduction et son age s'incrémentent.
— Il recupère son voisinage de mooore.
— Si sa faim a atteint son maximum. Il meurt et arrête son tour.
— Sinon il tente de manger un poisson situé aléatoirement dans son voisinage. S'il en mange un, il le tue.
— Ensuite, si son taux de reproduction a atteint la limite et qu'une case est libre à coté de lui, il se reproduit et un nouveau requin né sur une case aléatoire à coté. Le taux de reproduction est remis à zéro.
— Il tente ensuite de se déplacer aléatoirement sur une case libre dans le voisinage.

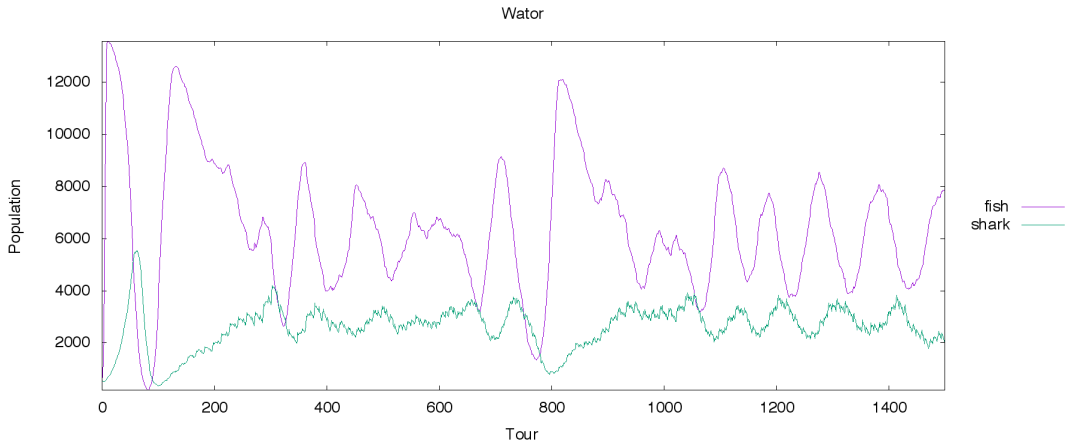
3.3 Utilisation

Pour tester le programme, il est possible de lancer directement le jar. Celui-ci est configuré par défaut avec les paramètres qui vont bien. Il est possible de spécifier les paramètres comme indiqué dans la documentation suivante.

```
Usage: java -jar wator.jar
  [-nbShark <NumberOfShark>](default: 500)
  [-nbFish <numberOfFish>](default: 500)
  [-fBreed <fishBreed>] (default: 1)
  [-sBreed <sharkBreed>] (default: 10)
  [-starve <SharkStarve>] (default: 15)
  [-nbTurn <numberOfTurn>] (default: 1000000)
  [-width <width>] (default: 120)
  [-height <height>] (default: 120)
  [-agentSize <sizeOfAgent>] (default: 5)
  [-speed <speed>] (default: 5)
  [-seed <seed>] (default: 0)
  [-grid] (default: false)
  [-equity] (default: true)
  [-noToric] (default: toric)
```

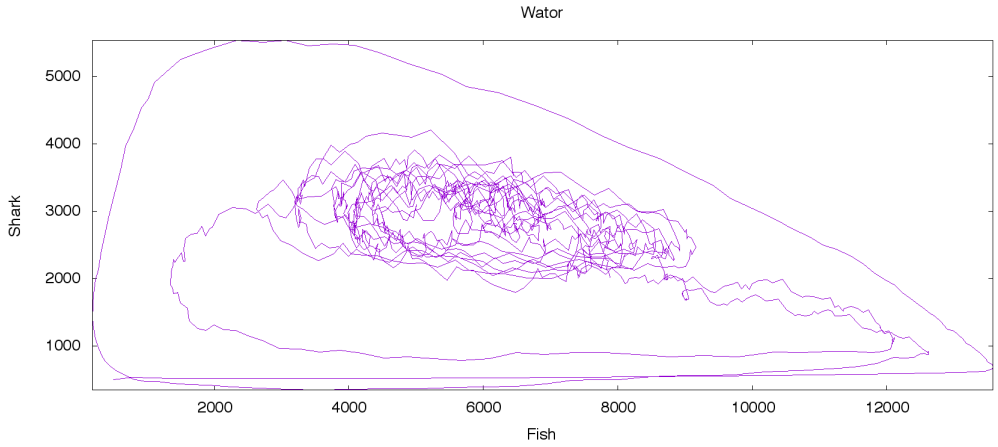
Pour vérifier la bonne implémentation des comportements des agents, nous avons généré les courbes suivantes :

FIGURE 1 – Nombre de Fish et Shark avec les paramètres par defaut



Comme on peut le voir, on a bien une inversion de phase entre la population des poissons et celles des requins. On peut aussi voir qu'à certain moment, la population des poissons est supérieure à celle des requins.

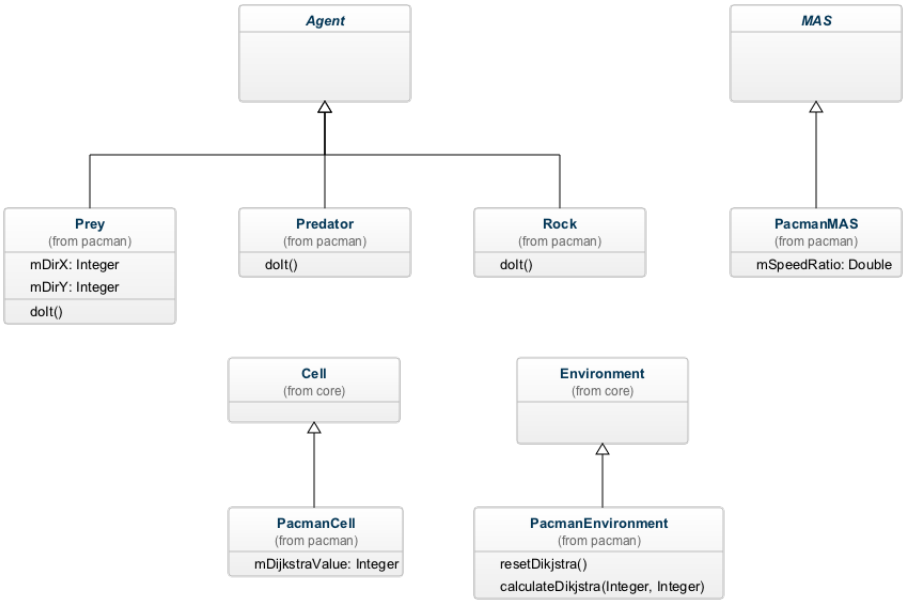
FIGURE 2 – Nombre de Fish en fonction du nombre de Shark avec les paramètres par defaut



4 Pacman

4.1 Architecture

Les classes du package Wator héritent des classes du package Core.
Les classes Prey, Predator et Rock représentent respectivement la proie, les prédateurs et les obstacles.
Une classe PacmanCanvas hérite de Canvas pour pouvoir afficher les chiffres Dijkstra.
La classe PacmanCell hérite de Cell pour contenir le nombre Dijkstra.
Environment est hérité par PacmanEnvironment pour pouvoir calculer Dijkstra sur l'espace.
PacmanMAS initialise la proie, les prédateurs et rocher et calcul le ratio de vitesse.



4.2 Comportement

Un paramètre speedRatio permet de gérer une différence de vitesse entre la proie et les prédateurs. Par exemple, pour un ratio $1.4 = 14/10 = 7/5$, la proie est avantagée. Ainsi, la proie parle tous les 5 tours et les prédateurs parlent tous les 7 tours. Ceci permet de prendre n'importe quel ratio mais peut provoquer un ralentissement du jeu. En effet dans l'exemple, de nombreux tours sont passés sans avoir d'action.

4.2.1 Prey

Un KeyListener dans le PacmanMAS est ajouté au canvas. La proie se déplace suivant les touches fléchés ou les touches qzsd. Si la touche est pressée, alors la direction est modifiée. Si la touche est relachée, alors la direction est remise à zéro.

4.2.2 Predator

Le comportement du predator est le suivant :
— Il recupère son voisinage de Moore.
— Il se déplace vers la cellule ayant la valeur de Dijkstra la plus faible. S'il y en a plusieurs, il choisit aléatoirement.

4.2.3 Rock

Les Rocher ont pour comportement de ne rien faire.
Comme ce sont des Agents, il est tout à fait possible de leur attribuer un comportement.

4.3 Utilisation

Pour tester le programme, il est possible de lancer directement le jar. Celui-ci est configuré par défaut avec les paramètres qui vont bien. Il est possible de spécifier les paramètres comme indiqué dans la documentation suivante.

```
Usage: java -jar pacman.jar
  [-nbPredator <NumberOfPredator>] (default: 3)
  [-nbRock <NumberOfRock>] (default: 500)
  [-width <width>] (default: 50)
  [-height <height>] (default: 50)
  [-agentSize <sizeOfAgent>] (default: 14)
  [-speedRatio <speedRatioWithPredator>] (default: 2)
  [-speed <speed>] (default: 100)
  [-seed <seed>] (default: random)
  [-grid] (default: false)
  [-equity] (default: true)
  [-dijkstra] (default: false)
```

5 Conclusion

Nous sommes fières de nos courbes concernant le projet Wator. Pour trouver les meilleurs paramètres, nous avons réalisé un script qui génère tous les graphiques pour des données de 1 à 18. Ceci nous a permis d'analyser et trouver les paramètres générant un graphique intéressant.
En somme, la programmation multi-agent est un très bon moyen de simuler des comportements de masses ou individuels. Nous avons pu découvrir la puissance des environnements comme NetLogo qui facilitent grandement le développement et permettent de directement se concentrer sur le comportement des agents plutôt que de l'architecture logicielle.