Université de Lille 1 SCIENCES ET TECHNOLOGIES

Rapport de PJI - ProJet Individuel

Sujet 29 : Serious-Game basé sur la simulation multi-agents des ressources halieutiques

Université Lille1

Auteurs :
Antonin Carette
Julien Duthoit

Encadrant de CRIStAL :
Dr. Sébastien Picault

Encadrant de l'Agrocampus Ouest :
Dr. Christine Largouët

Version du 1er juin 2015

Table des matières

Remerciements

Nous tenons à remercier toutes les personnes qui ont accompagné le développement de ce projet, et qui nous ont aidé lors de la rédaction de ce rapport.

Nous remercions ainsi nos encadrants de projet, Mr Sébastien Picault, enseignant-chercheur au laboratoire CRIStAL de Lille, et Mme Christine Largouët, enseignant-chercheur à l'Agrocampus Ouest de Rennes, qui nous ont beaucoup aidés et conseillés à chaque étape du projet et du rapport.

Introduction

Lors d'un projet universitaire organisé dans notre Master Informatique à l'Université de Lille 1, nous avons eu l'occasion de travailler sur un projet de recherche basé sur la simulation multi-agents, proposé par deux enseignants-chercheurs en Informatique de l'Université de Lille 1 et de l'Université de Rennes.
Ceci a eu pour but d'approfondir notre connaissance dans le milieu de la Recherche, ainsi que nous confronter à la simulation multi-agents.

Contexte

Le projet a pour but la création d'un *serious-game* permettant à l'utilisateur de simuler un écosystème marin complexe sur lequel agit les conditions propres à celui-ci, mais aussi le comportement des agents implémentés. Le contexte du *serious-game* permettra ainsi de pouvoir informer et mettre en garde l'utilisateur sur les répercutions engendrées par l'activité de la pêche sur cet écosystème, comme par exemple les étudiants de l'Université de Rennes.

Problèmatique

La problématique du projet s'axe sur :

- la difficulté de la modélisation du système multi-agents et des ressources halieutiques,
- l'implémentation de cette modélisation,
- l'expérimentation et le paramètrage de l'ensemble de la modélisation.

Objectif

Pour répondre à la problématique, nous avons utilisé la plateforme de simulation Netlogo et l'extension développée dans l'équipe SMAC de CRIStAL : IODA.

Chapitre 1

Les systèmes multi-agents

Nous souhaitons modéliser un écosystème marin, où chaque espèce marine peut donc interagir avec une ou plusieurs autres, en fonction de ses paramètres. Si nous modélisons l'écosystème d'un point de vue programmation, nos espèces sont des objets interagissant les uns avec les autres par des méthodes, afin de pouvoir obtenir un comportement.
Mais, il est beaucoup plus intéressant de pouvoir laisser ces espèces décider par eux-mêmes quel comportement adopter, pour une certaine configuration de l'écosystème à un temps t.
C'est ce qui nous a permis de nous pencher sur une branche de l'intelligence artificielle dite "distribuée", où l'intelligence artificielle est répartie sur un nombre défini d'entités, appelée "systèmes multi-agents".

1.1 Qu'est-ce-qu'un "système multi-agents" ?

Les systèmes multi-agents (SMA) permettent de simuler et modéliser des comportements d'individus, appelés "agents", interagissant dans un certain environnement que l'on définira comme un écosystème.
Utilisés principalement dans la recherche en intelligence artificielle, ou combinés avec d'autres domaines scientifiques tels que la chimie ou encore la biologie, les SMA sont aussi très utilisés dans l'industrie (comme le Cinéma - pour la représentation des mouvements de foule - et dans le Jeu-Vidéo par exemple) ou encore la finance (le suivi des cours de la bourse en *trading*).
Afin de simuler les agents et leurs comportements, il est nécessaire (dans un souci d'efficacité et réutilisation d'outils pré-existants) d'avoir une plateforme bien spécifique à l'application que l'on souhaite faire, et au domaine associé. De nombreuses existent et se spécialisent dans différents domaines d'application. Pour notre projet, il a été décidé d'utiliser la plateforme de simulation **Netlogo** [1], avec l'extension

1. Voir ici : `https://ccl.northwestern.edu/netlogo/`

IODA.

1.2 Présentation de la plateforme de simulation Netlogo et de l'extension IODA

1.2.1 Netlogo

Netlogo est une plateforme de simulation libre et gratuite, créée en 1999. Programmée en Java et en Scala, elle a été influencé par le langage de programmation Logo quant au DSL [2] dont elle est pourvue. Très facile d'utilisation, elle a d'abord été conçue pour l'aide à la programmation dans l'éducation.
La simulation Netlogo permet de prendre en compte différents agents (les *turtles*) vivant dans un environnement composé de *patchs* - typiquement, des carrés. Il est essentiel de savoir que plusieurs *turtles* peuvent être disposés sur un même *patch*, mais que chaque agent est placé sur un seul *patch* à fois ! Ainsi, sauf modification de l'utilisateur, un agent sera disposé au milieu d'un *patch*, à distance $\frac{l}{2}$ (où l est la longueur d'un côté de celui-ci).

1.2.2 IODA

IODA [3] est une extension écrite en NetLogo, créée par deux membres de l'équipe SMAC de CRIStAL : Sébastien Picault et Philippe Mathieu. Son but est de simplifier un maximum le design et la réutilisation de simulations individu-centré, c'est à dire des simulations basées sur les conséquences globales résultant d'interactions locales entre les individus de la population du système.
Les principes de IODA sont les suivants :

1. une entité est un agent,
2. tout comportement est représenté par une règle (c'est à dire une interaction), et chaque règle est composée d'un déclencheur (optionnel), d'une condition ainsi que d'une action,
3. ces interactions sont affectées à des familles d'agents, exécutées par un moteur générique.

Ainsi, c'est le moteur générique IODA qui détermine quelles interactions peuvent avoir lieu lors d'un pas de temps. Voici une trace de son exécution (pour un pas de temps) :

2. Un DSL est un langage dédié.
3. IODA : *Interaction-Oriented Design of Agent simulations* - voir ici : `http://www.lifl.fr/SMAC/projects/ioda/ioda_for_netlogo/doc/IODA-NetLogo-Documentation-v2.3.html`.

1. les interactions de mises à jour (*UPDATE*) réalisables sont exécutées par chaque agent donné,
2. puis chaque agent susceptible d'agir sur les autres choisit une interaction de la façon suivante :
   (a) perçoit ses voisins (subissant les actions des autres agents),
   (b) filtre les cibles d'interactions qu'il peut effectuer,
   (c) évalue les déclencheurs et conditions de ces interactions,
   (d) sélectionne une des interactions réalisables (de priorité maximale),
   (e) exécute les actions correspondantes, présentes dans la déclaration de l'interaction.

La Figure 1 sert d'exemple quant au fonctionnement du moteur IODA.

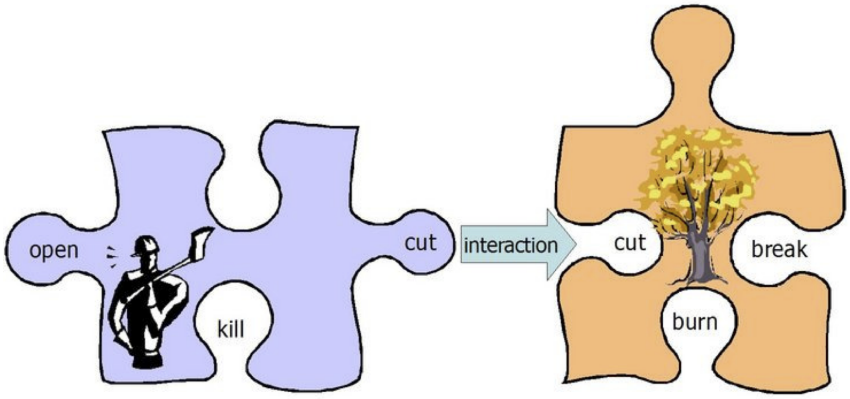

FIGURE 1 – Schéma récapitulatif quant au fonctionnement du moteur IODA.
Deux agents sont présents : l'agent *Woodcutter* et l'agent *Tree*. Ce premier est doté de 3 interactions agissant chacune sur un agent : *open*, *kill*, *cut* (sur un agent *Tree*), tandis que le second détient de même 3 interactions qu'il subira : *cut*, *burn* et *break*. Dans cette configuration, l'agent *Woodcutter* n'a que l'agent *Tree* parmi ses voisins - il préférera ainsi exécuter l'interaction *cut*, après que les déclencheurs et conditions aient été validés.

Chapitre 2

Le projet de *serious-game*

Notre projet s'axe sur plusieurs parties, ayant pour but d'amener à la création d'un *serious-game*, c'est à dire un jeu dont le premier objectif n'est pas le divertissement mais par exemple l'éducation ou l'exploration scientifique. Le projet se décompose en trois grandes étapes :

- la réalisation d'un modèle à une espèce et la détermination des briques de bases de la simulation(caractéristiques des poissons et leurs interactions),
- l'ajout de diverses espèces et d'interactions proies/prédateurs,
- la recherche d'un état d'équilibre afin de pouvoir stabiliser l'écosystème créé,
- enfin, l'ajout des pêcheurs et des ports.

L'intérêt d'utiliser la forme d'un *serious-game* est de pouvoir donner au joueur le pouvoir d'impacter la simulation, l'objectif étant une prise de conscience de l'impact de la pêche sur l'environnement, mais aussi du souci de la rentabilité. Le joueur doit réussir à mêler au mieux les deux pour réussir dans le jeu.

2.1 Rapide présentation du *serious-game*

La simulation sur laquelle se base le *serious-game* prend en compte plusieurs éléments. Il comprend les relations entre les proies et les prédateurs et la stabilité du système, les évènements ponctuels (tempêtes...) ainsi que l'impact de la pêche sur les populations marines selon les stratégies choisies.
Le joueur doit gérer son parc de bateaux avec différent types de bateaux et les envoyer pêcher selon divers critères. Son but étant de pêcher le plus de poissons tout en minimisant les pertes (comme le fuel par exemple), le joueur doit comprendre l'impact qu'a les bateaux et les stratégies de pêche sur l'écosystème.
Dans un second temps, chaque configuration possède un état stable dans lequel les poissons sont les plus nombreux et les mieux nourris. Dans le jeu, plus l'influence de la pêche sur cet environnement est élevée et moins le joueur gagnera de points.

2.2 Représentation des ressources halieutiques par un *serious-game*

La représentation des ressources halieutiques [1] par un *serious-game* nous permet d'avoir un "laboratoire virtuel", nous permettant de tester de nombreuses configurations.
Ces différentes configurations sont :

- le choix des espèces représentées,
- le choix des jeux de paramètres pour chaque espèce, chaque espèce étant représentée par sa consommation en nourriture, sa rapidité de reproduction, sa capacité à stocker de la nourriture et sa mortalité (en cas de manque de nourriture).

Chaque espèce possède aussi une stratégie de migration, représentant sa capacité à trouver de la nourriture et à se disperser, ou alors rester grouper.
Les différentes espèces de poissons sont représentées par sa masse (la **biomasse**). En effet celle-ci permet de représenter de manière simple et pertinente l'augmentation en taille des individus ainsi que l'apparition de nouveaux individus, ayant eux aussi leur masse propre.
Ainsi, chaque interaction des poissons revient à créer un flux de biomasse d'un agent à un autre, ou bien tout simplement à représenter l'énergie dépensée par un évènement particulier.

2.3 Apport du système multi-agents pour la modélisation

La plate-forme Netlogo spatialise les agents, ce qui permet de facilement gérer les déplacements d'un bateau d'un patch à un autre. Pour les poissons, chaque espèce a un seul et unique représentant (de l'intégralité des poissons de l'espèce présente) par case. Les différents agents agissent à chaque tour s'ils le peuvent avec les différents agents de la simulation, selon leurs besoins.
On peut aussi observer l'impact spatial avec la répartition des différents agents selon leurs stratégies, diverses pressions (comme la pêche) et événements.
Les différents agents ne peuvent interagir avec l'environnement et les autres agents que par des interactions dans notre modèle. Les agents représentant les poissons ont tous les mêmes interactions, ce qui permet de faciliter le développement en utilisant une interaction générique qui va être exécuté par tous les agents voulant

1. Les ressources halieutiques représentent l'art de la pêche et de la gestion des ressources aquatiques pêchées.

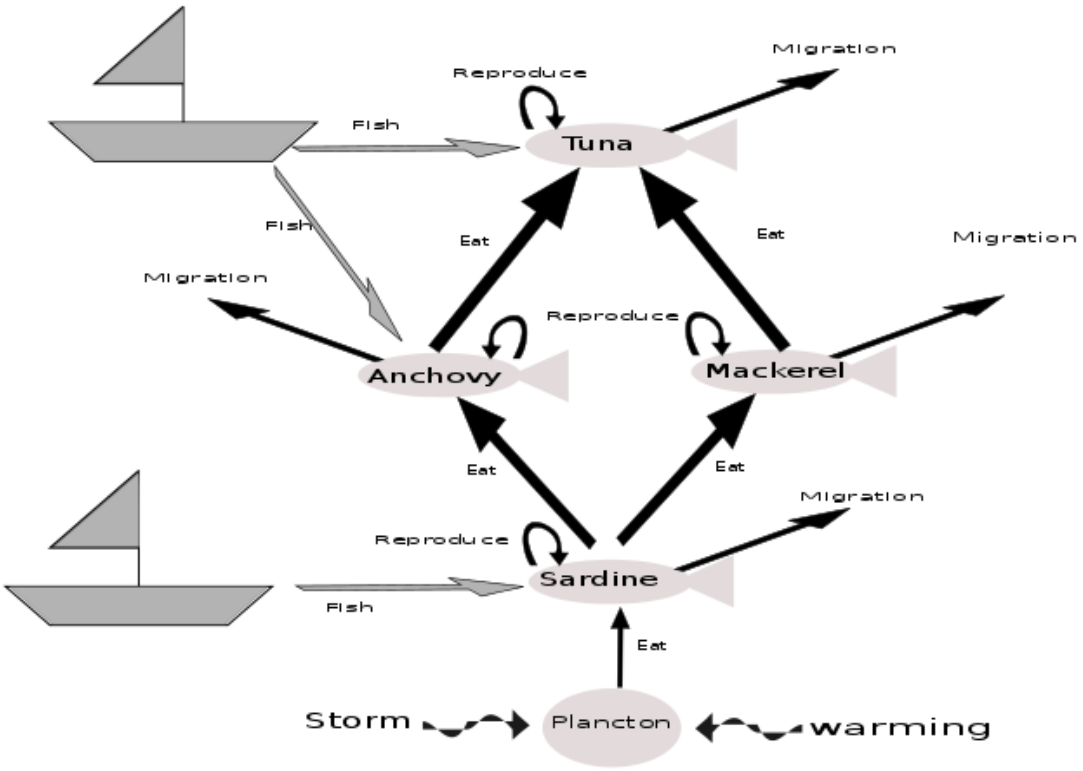

FIGURE 2 – Schéma représentatif des flux de biomasse

l'implémentant. Cela nous a ainsi permis de développer les interactions selon un modèle simple (le plancton - *Food* dans la simulation) et de généraliser vers le modèle complet avec différents agents proies/prédateurs.

Chapitre 3

Modélisation du projet

La modélisation du projet a été réalisée avec Netlogo et l'extension IODA.
L'évolution du projet a été constante sur toute sa durée, de l'implémentation des premiers agents, de leurs prédateurs (afin de pouvoir établir une chaîne alimentaire), et enfin des pêcheurs et de l'évaluation des scores.
Concernant l'utilisation des outils permettant la modélisation, nous avons dû nous former sur ceux-ci, Netlogo et IODA nous étant alors inconnu. Cette formation a ainsi débouché sur une compréhension générale du fonctionnement de la plateforme Netlogo et du moteur IODA, nous ayant permis de débuter le projet. La suite de celui-ci nous a ensuite permis d'expérimenter et de terminer cette formation.

3.1 Caractéristiques et comportements des agents

Chaque espèce marine sera représentée par un agent - appelé *breed* en Netlogo, caractérisant une sous-espèce de *turtles*[1]. Elle est représentée dans la modélisation par son abondance, c'est-à-dire la quantité de sa biomasse.

Attention ! Il est important de faire remarquer que l'on se servira de l'espace de nommage IODA pour expliciter au programmeur quel *breed* Netlogo exécute quelle primitive ! Ainsi, **nous parlons d'*agents* (en italique) comme d'une convention implicite de nommage permettant de dire que la primitive est exécutée par tout agent implémenté** (*Boats*, *Anchovies*, *Mackerels*, etc...).
On recense 3 types d'agents :

- **les proies** : le plancton (*Food* dans la modélisation),
- **les proies/prédateurs** : le maquereau, la sardine, etc...
- **les prédateurs** : le thon, le pêcheur.

Chaque type d'agents diffère par ses caractéristiques propres, mais aussi par l'im-

1. Il est important de notifier que la notion d'héritage n'existe pas pour les *breeds* Netlogo.

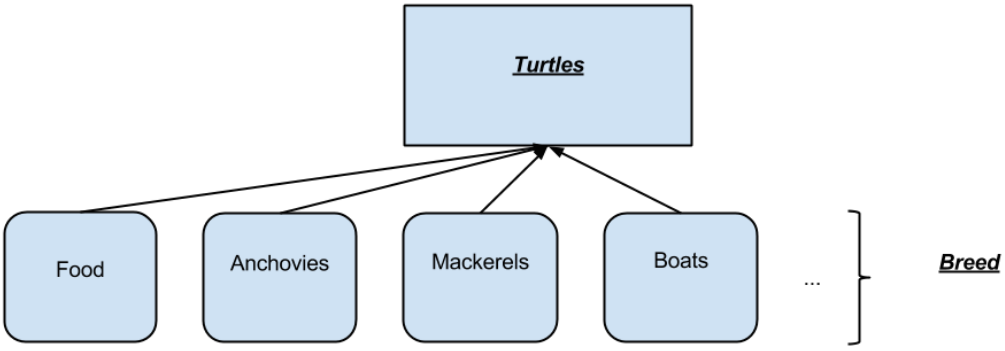

FIGURE 3 – Schéma récapitulatif quant à l'implémentation des *turtles* et des *breeds* pour Netlogo.

plémentation de ses comportements. Ainsi, chaque type (ou classe) d'agents a des caractéristiques et comportements différents des autres.
Par contre, chaque agent d'un même type détient les mêmes caractéristiques, à ceci près que les valeurs de ces paramètres peuvent être différentes (en fonction des conditions biologiques, de leur type de prédation, etc...). Ainsi, tous les agents proies/prédateurs partagent les mêmes caractéristiques.
Pour finir, chaque interaction sera implémentée dans un fichier d'interactions, tandis que sa déclaration, son agissement et sa priorité seront définies dans une matrice d'interactions.

3.1.1 Caractéristiques des agents

Les agents proies

Les proies ont un nombre réduit de caractéristiques, dû au fait qu'ils se situent au plus bas de la chaîne alimentaire.

Les agents proies/prédateurs

Les agents proies/prédateurs auront un grand nombre de caractéristiques, compte-tenu du fait qu'ils appartiennent aux deux grandes classes d'individus.

Les agents prédateurs

Les agents prédateurs, eux, sont encore divisé en deux grandes catégories :

- les agents prédateurs (pour les espèces marines),
- les agents prédateurs humains, agissant sur les espèces marines.

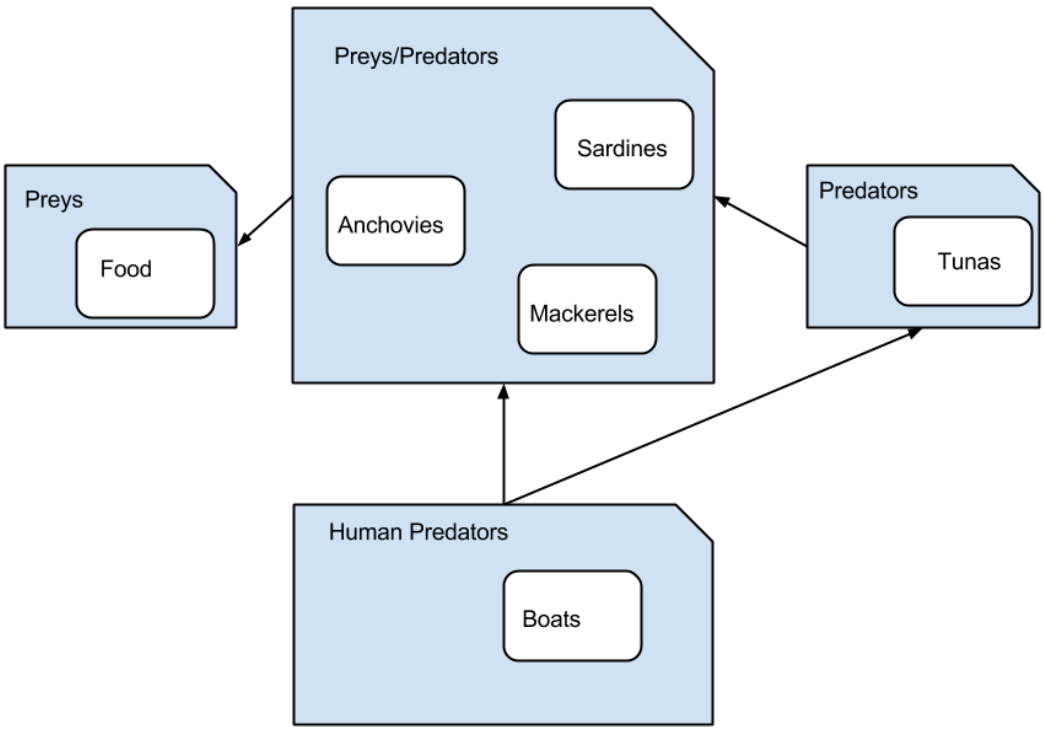

FIGURE 4 – Modélisation des différents *types* d'agents. La flèche signifie que l'espèce pointée est la proie de l'espèce pointeuse.

3.1.2 Les comportements des individus

Afin de faciliter la compréhension des matrices contenant les comportements des individus, nous nous appuierons d'abord sur un exemple simple : un écosystème ne contenant qu'une espèce marine prédatrice (*agent*) et qu'une espèce primitive marine (*Food*).

La matrice d'interactions

La priorité donnée aux comportements des agents est définie dans une matrice, dont les interactions avec IODA nous permettra de modifier à la volée et à volonté certains paramètres contenus dans cette dernière (par exemple, pour le changement de migration d'une espèce marine donnée).
Nous rappelons qu'**un seul comportement est exécuté** par agent sera exécuté pour un pas de temps (ou *tick*[2]).

La matrice d'interactions, présentée en Figure 6, montre plusieurs choses intéressantes et nécessaires à la compréhension de la simulation :

2. Un *tick* correspond à un pas de temps dans la simulation.

Properties	Preys	Preys/Predators	(Fishes) Predators	(Humans) predators
Biomass	True	True	True	False
Biomass requirement	False	True	True	False
Old biomass	True	True	True	False
Food maxi	True	True	True	False
Food stock	False	True	True	True
Mortality	False	True	True	False
Reproduction	False	True	True	False
Food consumption	False	True	True	False
Migration strategy	False	True	True	True

FIGURE 5 – Tableau récapitulatif concernant les grandes caractéristiques des différentes catégories d'agents.

sources	interactions	priority	targets	distances
agent	Migration	100	Food	1
agent	Reproduce	80	Food	0.5
agent	Eat	70	Food	0.5
agent	Die	20	UPDATE	//
agent	ComputeBiomassReq	10	UPDATE	//
agent	ComputeFoodMaxi	10	UPDATE	//
agent	ComputeOldBiomassF	0	UPDATE	//
agent	Digest	0	UPDATE	//
Food	Grow	0	UPDATE	//

FIGURE 6 – Un exemple simple de la matrice d'interactions pour une espèce marine (agent) et le plancton (*Food*). **Ex1 :** L'agent *agent* va adopter pour le *tick* **x** le comportement *Migration* avec une priorité de 100, sur un agent *Food* à distance 1 de lui. **Ex2 :** L'agent *agent* va adopter pour le *tick* **x** le comportement *Digest* avec une priorité 0.

- seuls les deux agents sont présents (colonne *breed*)
- plusieurs comportements sont définis pour l'espèce *agent* - chaque comportement possède une certaine priorité, et chaque agent cherche à réaliser une interaction de priorité la plus élevée possible (en tenant compte que toutes les conditions de ce comportement sont validées) [3] :
  - un comportement défini prenant une cible, sur une distance définie,
  - un comportement défini ne prenant aucune cible ; il sera donc défini comme une mise-à-jour pour chaque pas de temps
- un seul comportement est défini pour *Food* - il est intéressant de constater ici que même si le poids de réalisation est de 0, ce comportement sera défini comme priorité, car seul.

3. On rappelle ici que le moteur IODA est déterministe.

Le fichier d'interactions

Le fichier d'interactions permet de définir les comportements pris en compte dans la matrice d'interactions.
En prenant exemple sur la Figure 6, nous pouvons pour donner un exemple écrire le fichier d'interactions (voir Figure 7) correspondant.
La méthodologie d'écriture pour une interaction IODA est la suivante :

1. se poser la question "Qui (un agent) fait quoi (une interaction) ? Et avec Qui ?",
2. définir les interactions dans la matrice d'interactions, avec l'apport du déclenchement (le *trigger*), des conditions (*Conditions*) et des actions (*Actions*),
3. enfin, le codage des primitives.

Concrètement, chaque interaction est définie par le mot-clef ***interaction***, et se termine avec le mot-clef ***end***.
Une interaction est appelée automatiquement si le ***trigger*** est bien validé. Aussi, il faut que les conditions soient justifiées pour l'agent mais aussi pour sa cible si elle existe (appelée avec ***target :***). Chaque suite de conditions est ainsi composée d'une suite de disjonction de conditions, permettant de mettre-en-oeuvre l'interaction si au moins l'une des conditions est respectée !
Enfin, la dernière étape est l'appel des procédures et/ou fonctions à appeler dans le champ ***actions*** de l'interaction.

Cette section marque la fin de l'implémentation de la base de la modélisation.
Cette partie a été l'une des plus difficile à mettre en oeuvre mais aussi la plus contraignante, étant donné qu'il a fallu retranscrire avec précision chaque caractéristique et comportement pour un agent, en sachant que toute la suite du projet repose sur cette base d'implémentation. Ainsi, il a fallu prévoir les modifications qui ont dû être effectuées par la suite.

3.2 La création de la chaîne alimentaire

Pour la chaîne alimentaire, nous avons envisagé plusieurs scénarios :

- une proie peut être finale [4] ou non,
- un prédateur peut être une proie pour une autre espèce ou non,
- un prédateur peut manger une ou plusieurs proie(s),
- deux prédateurs peuvent entrer en compétition pour une seule proie.

Étant donné que l'utilisation de l'héritage (comme vu dans la Programmation Orientée Objet) n'existe pas en Netlogo, la modélisation de la chaîne alimentaire a été un peu plus difficile à mettre en oeuvre.

4. Une espèce est une proie finale si elle n'est pas prédatrice pour une autre espèce.

```
interaction Migration
    trigger     not:enoughFood?
    condition stillAlive? not:target:samePatch? not:food-migration? target:stillAlive?
    condition   stillAlive? target:enoughFood? food-migration? target:stillAlive?
    actions     migration   eat
end

interaction Reproduce
    trigger     canReproduce?
    condition   stillAlive? not:hungry? target:stillAlive?
    actions     reproduce eat
end

interaction Eat
    trigger     hungry?
    condition   stillAlive? target:stillAlive?
    actions     eat
end

interaction Die
    trigger     hungry?
    actions     die
end

...

interaction ComputeBiomassReq
    condition   stillAlive?
    actions     computeBiomassReq
end

;;FOOD
interaction Grow
    actions     grow
end
```

FIGURE 7 – Un extrait du fichier d'interactions, correspondant avec l'exemple de la Figure 6.

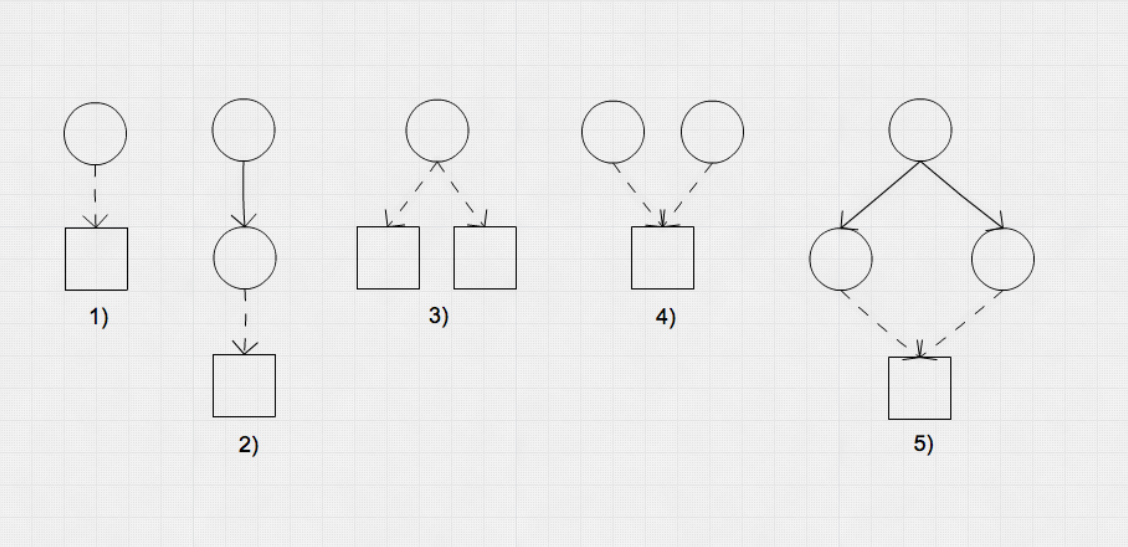

FIGURE 8 – 5 schémas permettant de comprendre les scénarios envisagés - les cercles représentants les proies, les carrés représentant les prédateurs.
1) Un prédateur cible une proie finale.
2) Un prédateur cible une proie (non-finale) qui elle cible une proie finale.
3) Un prédateur cible deux proies finales.
4) Deux prédateurs ciblent une proie finale.
5) Un prédateur cible deux proies non-finales qui, elles, ciblent une seule proie finale.

Chaque comportement ayant été implémenté pour une espèce générique (*agent*), nous avons dû redéclarer chaque comportement pour l'espèce en cours d'implémentation. Le problème qui se pose est alors de bien différencier l'utilisation d'une primitive pour : soit un prédateur, soit une proie. La spécification du type Proie/Prédateur de l'espèce pour un comportement a donc été implémentée dès que l'utilisation le permettait.

Breed	**Preys**	**Predators**	**Eat...**
Food	True	False	Nothing
Sardines	True	False	Food
Anchovies	True	True	Sardines
Mackerels	True	True	Sardines
Tunas	False (for Fishes)	True	Anchovies & Mackerels

FIGURE 9 – Tableau récapitulatif concernant l'implémentation des différents agents de la simulation.

La matrice d'interactions ainsi que la modélisation de la simulation ont ainsi été revues afin de permettre une meilleure compréhension et lisibilité du code (voir Figure 10).
Ainsi, une espèce proie **et** prédateur détenant le comportement *isAlive ?* implémen-

tera une méthode pour Proie ainsi qu'une autre pour Prédateur [5], chacune exécutant le comportement général *isAlive ?*.
Il faudra faire attention à la notion de Proie pour l'agent *Tunas*. En effet, aucune autre espèce marine n'est un prédateur pour cet agent, mais il ne faut pas oublier que chaque espèce marine est une proie pour les pêcheurs !

```
interaction Eat
  trigger PRED_hungry?
  condition PRED_stillAlive? target:PREY_stillAlive?
  actions PRED_eat
end

...

to-report agent::stillAlive?
  report biomass-f > 0
end

to-report mackerels::PRED_stillAlive?
  report agent::stillAlive?
end

to-report mackerels::PREY_stillAlive?
  report agent::stillAlive?
end
```

FIGURE 10 – Exemple de l'implémentation d'un comportement général pour une espèce proie/prédateur.

3.3 Le cas de la pêche

La pêche introduit deux nouveaux agents dans la simulation : les ports (*Harbours*) et les bateaux, ou pêcheurs (*Boats*).
Les ports influencent la création de bateaux, ainsi que l'espèce ou les espèces marine(s) à pêcher. Elles n'ont ainsi aucune influence directe sur chaque espèce marine, *a contrario* des bateaux influenceront directement la variation d'espèces marines via la pêche, selon le nombre de bateaux déployés et des espèces pêchées par ceux-ci. Ainsi, les pêcheurs sont qualifiés de *super-prédateurs*, car elles n'ont pas de prédateurs.
Comme ces derniers n'ont pas de prédateurs, il va falloir implémenter des contraintes permettant de réduire leur dominance sur la majorité de l'écosystème marin. Des contraintes (inspirées de celles réelles) sur les bateaux et sur les ports ont donc été implémentés :

5. Notre nomenclature voudra que chaque comportement exécuté pour une proie sera précédé par **PREY_**, et que chaque comportement exécuté pour un prédateur sera précédé par **PRED_**.

- chaque port a un nombre fixé de bateaux à déployer en mer,
- chaque port a un certain nombre d'espèces pris en compte quant à la pêche - ce nombre peut être compris entre *1* et n, n étant le nombre total d'espèces marines,
- chaque bateau est spécifique à un port,
- chaque bateau ne peut supporter une charge de biomasse supérieure à son seuil maximum fixé - il sera ainsi obligé de rentrer au port décharger sa cargaison,
- chaque bateau détient une réserve de fioul, initialisée au départ du port - si cette réserve atteint un seuil égal à la réserve nécessaire pour rentrer au port, celui-ci retourne décharger sa cargaison.

```
boats              ToFish                 100   anchovies   1
boats              ToFish                 100   mackerels   1
boats              ToFish                 100   sardines    1
boats              ToFish                 100   tunas       1
boats              MoveWithoutFishing     80
boats              GoToHarbour            80

harbours         ComputeFoodStock         10  UPDATE
harbours         ComputeBoatsAvailable    10  UPDATE

boats            VerifySailing            10  UPDATE
boats            DecrTick                 10  UPDATE
```

FIGURE 11 – Exemple de l'implémentation de la matrice d'interactions pour les agents *Harbours* et *Boats*.

Comme l'ensemble de ces nouveaux agents n'ont pas de comportements liés aux espèces marines déjà implémentés, la redéclaration des primitives n'est pas utile dans cette partie du projet. Nous avons ainsi défini de nouveaux comportements, propres aux ports et aux bateaux, afin de gérer au mieux l'ensemble des interactions avec le milieu marin, mais aussi entre les ports et les bateaux, voire même entre les bateaux aussi (voir Figure 11 et Figure 12).

3.4 L'expérimentation

C'est à partir de l'implémentation concernant la chaîne alimentaire qu'il a fallu expérimenter la simulation en testant et en modifiant chaque paramètre pris en compte lors de la modélisation car, en ayant introduit de nouveaux agents influençant directement les autres agents, nous avons totalement bouleversé l'écosystème - le réduisant dans un premier temps à un écosystème se réduisant au plancton, seul survivant de la modélisation après x tours.
Pour avoir un modèle crédible sur lequel tester l'impact de la pêche, il faut que

```
;DeployABoat -> make a boat to sail
interaction DeployABoat
    trigger boatsAvailableToSail?
    condition not:target:isSailing?
    actions deployABoat
end

;The harbour computing if he could sell his food stock
interaction ComputeFoodStock
    actions computeFoodStock
end

;The harbour computing number of boats available to sail
interaction ComputeBoatsAvailable
    actions computeBoatsAvailable
end

;;BOATS

;A boat will fish if he's sailing, he can sail again and if the prey is still alive
interaction ToFish
    condition   isSailing? isAvailableToSail? target:PREY_stillAlive?
    actions     toFish
end

;Interaction to move without fishing
interaction MoveWithoutFishing
    condition   isSailing? isAvailableToSail? not:target:PREY_stillAlive?
    actions     moveWithoutFishing
end

;A boat will go to harbour if he's sailing and if he's not available to sail
interaction GoToHarbour
    condition   isSailing? not:isAvailableToSail?
    actions     goToHarbour
end

;Interaction to verify he the boat is sailing
interaction VerifySailing
    condition   isSailing? isAvailableToSail?
    actions     verifySailing
end

;Interaction to decrement ticks
interaction DecrTick
    condition   isSailing? isAvailableToSail?
    actions     decrTick
end
```

FIGURE 12 – Extrait des comportements pour les agents *Harbours* et *Boats*.

l'écosystème soit viable sur le long terme. Ainsi, pour résoudre ce problème efficacement, Netlogo possède un outil d'analyse, appelé *BehaviorSpace* (exemple Figure 13), permettant de lancer la simulation n fois pour chaque jeu de paramètres choisi. Grâce à cet outil, nous avons pu mettre en place différents tests afin de trouver un équilibre.
Le fonctionnement de notre modèle imposait de tester espèce par espèce. Dans un premier temps, il a fallu determiner des paramètres à tester, l'utilisation d'un trop grand nombre de paramètres générant des expériences trop longues. Il a été préféré les trois critères de recherche suivant (voir le deuxième cadre de la Figure 13) :

- la mortalité,
- la consommation de nourriture,
- la reproduction.

Le système ne permettant pas la survie des différentes espèces, la recherche en temps de survie des différentes espèces était plus pertinente au départ, la finalité était de regarder la biomasse des différentes espèces. Ce qui a permis d'arriver à une méthodologie de recherche en plusieurs temps :

- la **recherche large** : très grande gamme de valeurs pour les paramètres, mais très écartées (pas de répétitions),
- la **recherche fine** : faible gamme de valeurs pour les paramètres et assez rapprochées (pas de répétitions),
- la **recherche profonde** : très faible gamme de valeurs, très rapprochées (grand nombre de répétitions).

Le but étant d'isolé des valeurs interessante puis de vérifier la robustesse de celle-ci. Un autre avantage du *BehaviorSpace* est qu'il permet aussi d'enregistrer dans un fichier des valeurs particulières (par exemple la biomasse des agents thons) et d'avoir la moyenne à chaque tour, permettant de connaitre l'espérance sur un grand nombre d'expériences (ce qui s'avère aussi utile pour le score).

La deuxième grande partie de l'experimentation permet de mesurer l'impact de la pêche sur un jeu de données connu, en comparant les bateaux critère par critère (spécialisé contre opportuniste, grand stock contre faible stock, faible autonomie contre grande autonomie, etc...). On observe d'un côté l'efficacité des différentes configuration pour connaître la rentabilité la plus forte et de l'autre, et de l'autre l'effet sur les populations de poissons.

3.5 Modélisation de l'interface graphique

En modélisant le projet avec Netlogo, nous disposions des différents outils de représentation graphique de Netlogo :

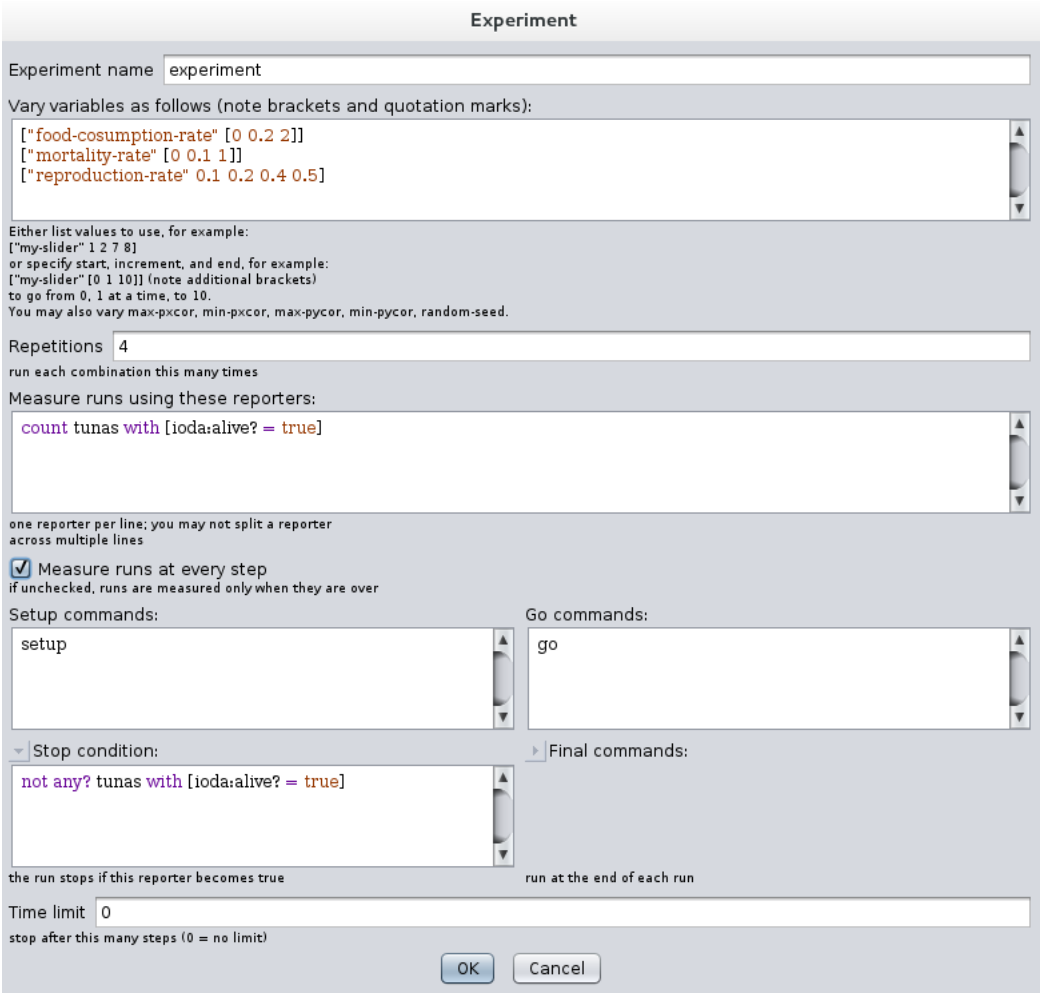

FIGURE 13 – Exemple de configuration du *BehaviorSpace*.

- une représentation dans l'espace (voir Figure 14),
- un outil permettant de tracer des courbes en temps réel (voir Figure 15),
- de multiple éléments graphiques permettant d'avoir un contrôle sur la simulation (voir Figure 16).

On peut voir sur la Figure 14 que la nourriture est représentée par une forme (ronde dans cette modélisation) d'une couleur plus ou moins intense en fonction de la quantité de nourriture, les agents poissons vivants sur le *patch* sont représentés par des poissons (la couleur variant selon l'espèce) et les bateaux par des bateaux. Cette vue permet d'avoir une idée de la représentation spatiale des espèces, et peut être complétée selon l'information précise concernant la présence d'une ou plusieurs espèce(s) (visualisation d'une espèce seulement, visualisation de l'espèce la plus présente par *patch*, etc...), comme on peut le voir sur la Figure 17.
Les courbes (voir Figure 15) permettent d'avoir une idée précise de la quantité de poissons en terme de biomasse et de la satisfaction de ceux-ci en nourriture, grâce à la connaissance du stock qui diminue avec sa raréfaction.
On a aussi accès à un grand nombre de boutons et de variables (voir Figure 16), qui

permettent de tester des configurations (régler les paramètres pour les expérimentations), d'avoir une influence sur l'environnement (croissance du plancton, etc...), ou d'interagir avec l'affichage (température, etc...)

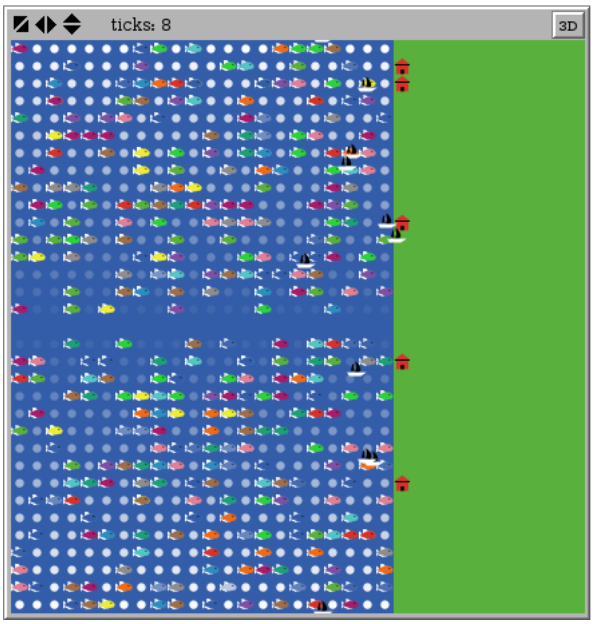

FIGURE 14 – Représentation graphique d'une simulation.

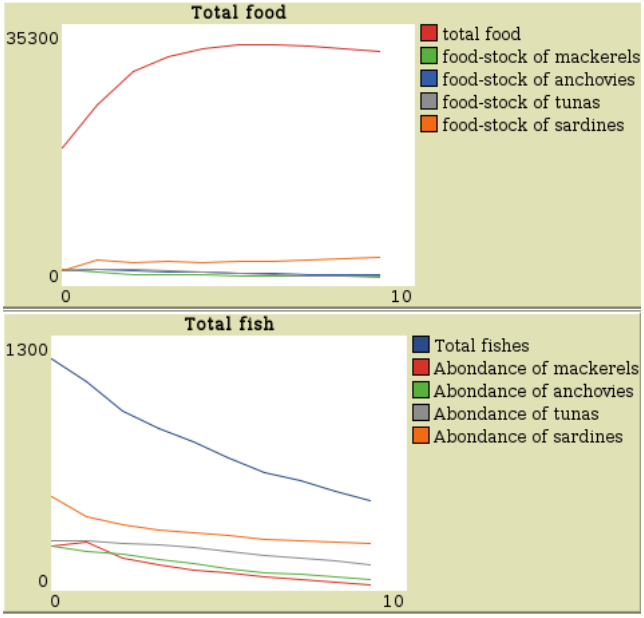

FIGURE 15 – Courbes représentant la biomasse ainsi que les stocks de nourriture.

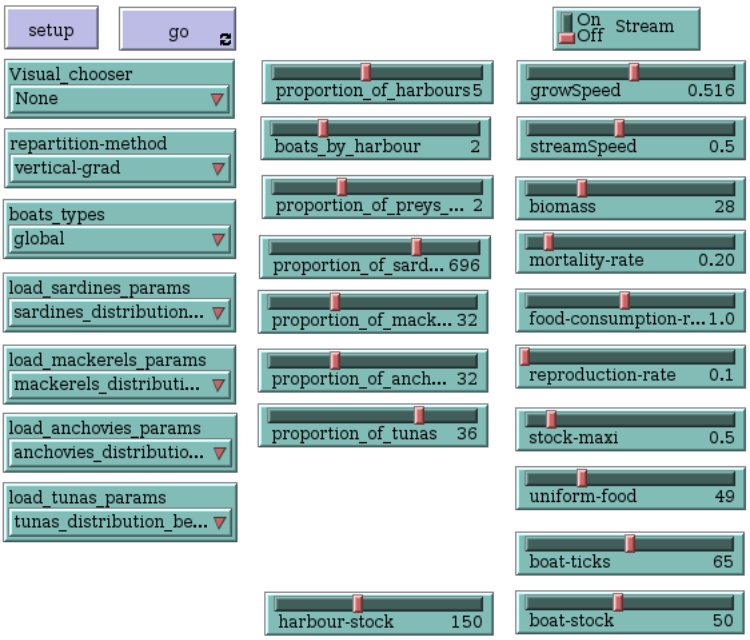

FIGURE 16 – Boutons de la simulation, permettant de modifier les paramètres de la simulation.

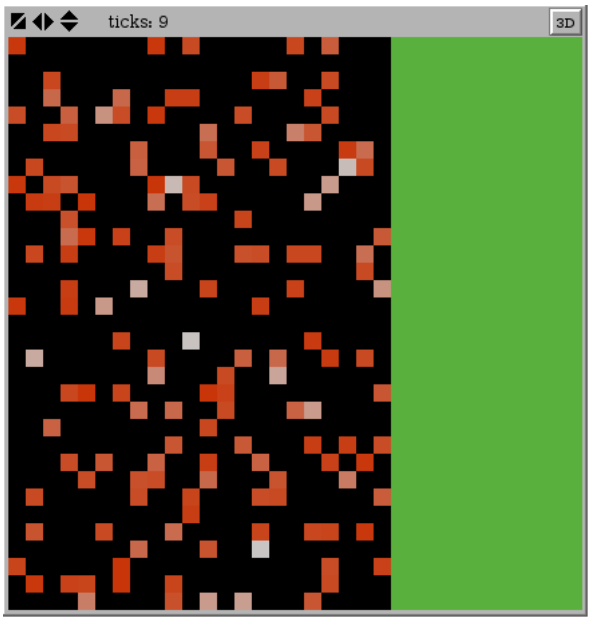

FIGURE 17 – Carte de temperature des thons.

Conclusion

En conclusion, la modélisation de l'écosystème marin via un système multi-agents, basé sur les ressources halieutiques, a été réalisée. Cette base logicielle a été testée et paramétrée afin de pouvoir la stabiliser pour laisser le champ libre au travail sur l'aspect "Jeu" du projet (via l'implémentation du système de score).
Cela nous a permit personnellement d'approcher et de comprendre les bases du système multi-agents, d'établir des procédures de tests complètes ainsi que de toucher à la plateforme Netlogo et l'extension IODA.

Il serai cependant intéressant de réfléchir à l'exportation du projet sur le Web, notamment grâce au fait que le projet s'adresse à un large public (qui ne détient pas forcément la plateforme logicielle pour pouvoir y jouer), ainsi qu'au travail que fait l'équipe de Netlogo sur l'exportation de sa plateforme en langage JavaScript.

Au final, ce sujet de recherche nous a d'ores et déjà apporté la compréhension de certaines notions, notions qui seront enseignées et approfondies dans la spécialité du Master que nous allons poursuivre : le Master MoCAD de l'Université de Lille 1.

Bibliographie/Webographie

[1] *Netlogo documentation.* `http://ccl.northwestern.edu/netlogo/docs/`.

[2] M.-O. Cordier, C. Largouët, and Y. Zhao. Model-checking an ecosystem model for decision-aid. In G.A. Papadopoulos, editor, *Proc. of the IEEE 26th Int. Conf. on Tools with Artificial Intelligence (ICTAI'2014)*, pages 539–543, 2014.

[3] J. Ferber. *Les systèmes multi-agents : vers une intelligence collective.* interEditions, 1995.

[4] Y. Kubera, P. Mathieu, and S. Picault. IODA : An interaction-oriented approach for multi-agent based simulations. *Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS)*, 23(3) :303–343, 2011.

[5] P. Mathieu and S. Picault. *IODA documentation.* `http://cristal.univ-lille.fr/SMAC/projects/ioda/ioda_for_netlogo/`.