Proposition d'une approche de modélisation du comportement d'espèces animales guidée par les métaheuristiques : vers de nouveaux outils d'aide à la compréhension du comportement animal

Chabi Affolabi Rodolpho Babatounde

UNIVERSITE DE CORSE - PASCAL PAOLI

ECOLE DOCTORALE ENVIRONNEMENT ET SOCIETE

UMR CNRS 6134 Science Pour l'Environnement

UAR CNRS 3514 STELLA MARE

Thèse présentée pour l'obtention du grade de

DOCTEUR EN INFORMATIQUE

Soutenue publiquement par

M. Chabi Affolabi Rodolpho BABATOUNDE

(chabibabatounde@gmail.com)

le 13 Mai 2024

Proposition d'une approche de modélisation du comportement d'espèces animales guidée par les métaheuristiques : vers de nouveaux outils d'aide à la compréhension du comportement animal

---

Directeurs :

- M. Antoine AIELLO, DR, Université de Corse
- M. Bastien POGGI, Dr, Université de Corse

Rapporteurs :

- M. Rémy COURDIER, Pr, Université de La Réunion
- M. David HILL, Pr, Université Clermont Auvergne

Jury :

- M. Angelo STEFENEL, Pr, Université de Reims Champagne-Ardenne
- M. Rémy COURDIER, Pr, Université de La Réunion
- M. David HILL, Pr, Université Clermont Auvergne
- M. Thierry ANTOINE-SANTONI, Pr, Université de Corse
- Mme. Oumaya BAALA, Dr-HDR, Université Technologique Belfort
- M. Antoine AIELLO, DR, Université de Corse
- M. Bastien POGGI, Dr, Université de Corse

A mon feu père René BABATOUNDE

Ta tragique disparition a beaucoup affecté la famille. Tout au long de ta vie, tu as su nous inculquer les valeurs morales et le goût du travail bien fait. Nous te promettons de mettre en pratique tous tes sages conseils. Repose en paix, papa chéri.

A ma mère Rissikatou SOUMANOU

Tu as toujours été là pour nous, aussi bien dans les moments difficiles que dans les moments heureux. Aucun de ces mots ne pourra exprimer suffisamment ma gratitude et mon amour pour toi. Tu peux être fier de ce que je suis devenu ! Merci pour cette éducation que papa et toi nous avez donnée.

Remerciements

Le couronnement de ce travail n'a été possible que grâce au concours direct ou indirect de certaines personnes qui à un moment donné, ont été présentes pour moi.

Je tiens sincèrement à exprimer ma gratitude envers :

— Les Professeurs Angelo Stefenel, Remy Courdier, David Hill et le Docteur Oumaya Baala pour avoir généreusement offert leur expertise et leur regard critique sur ce travail. Vos commentaires lors des réunions du Comité de Suivi, nos échanges enrichissants lors des colloques TERRINT, ainsi que votre participation au jury et la rédaction de vos rapports ont grandement contribué à améliorer la qualité de mes travaux de thèse.

— Le Professeur Thierry Antoine-Santoni et le Docteur Evelyne Vittori, en tant qu'acteurs souvent discrets, mais déterminants pour l'aboutissement de ce travail, je tiens à souligner le rôle essentiel que vous avez joué dans mon parcours académique. Votre soutien sans faille pendant mes études de master a contribué à forger la base qui a fait de moi un étudiant accompli. Aujourd'hui, en tant que chercheur, votre influence perdure, et je vous suis reconnaissant pour les nombreuses séances de travail, les visioconférences et les multiples relectures de ce document. Notre chemin ne s'arrête pas ici, et je suis résolu à continuer de bénéficier de votre guidance.

— Antoine Aiello, je tiens à te remercier pour ta confiance. Merci de m'avoir offert l'opportunité de rejoindre la remarquable équipe de STELLA MARE, une expérience qui m'a permis de progresser sur de nombreux plans. Cette incroyable aventure a débuté il y a plus de cinq ans déjà. En tant qu'alternant, doctorant, ou ingénieur, à chaque étape, j'ai eu la chance de bénéficier de ta confiance indéfectible. Je tiens à exprimer ma gratitude pour tout ce que tu as rendu possible, tant sur le plan professionnel que personnel.

— Bastien Poggi, toutes les pages de ce document ne suffiraient pas à exprimer ma gratitude envers toi. Tout a commencé, cette soirée mémorable du 15 novembre 2018 où nous avons discuté pour la première fois de mon projet de thèse, et depuis, notre collaboration n'a jamais faibli. À travers toi, j'ai trouvé un enseignant, un directeur de thèse, un ami et surtout un grand frère. Aujourd'hui, cette réalisation est aussi la tienne. Nous avons encore de nombreux projets à mener ensemble, et je suis impatient de les concrétiser.
Un merci spécial à Alexandra pour sa grande gentillesse.
Je n'oublie évidemment pas les copains, Pépin et surtout Maximux, la mascotte de l'équipe Metaheuristiques. Je l'avoue, Max est l'une des raisons qui me motivaient à faire le long trajet jusqu'à Ajaccio !
À tous, je vous envoie mes plus sincères salutations et remerciements.

— Toute l'équipe de l'École Doctorale, sous la direction d'Alain Muselli. Un remerciement tout particulier à David Moungar pour sa disponibilité et son soutien tout au long de ces années de thèse.

— Marie Laure Nivet et Paul-Antoine Bisgambiglia, pour leurs conseils scientifiques réfléchis et clairvoyants. Votre expertise a été extrêmement précieuse dans ma formation et le développement de ma recherche, et je vous suis reconnaissant pour votre soutien.

— L'ensemble du personnel de STELLA MARE, que ce soit de près ou de loin, chacun d'entre vous a apporté une contribution précieuse à la réalisation de cette thèse. Votre collaboration et votre soutien ont été inestimables, et les moments partagés lors des pauses café, des repas au CASONE et des soirées de l'AG illustrent parfaitement la belle équipe que nous formons.
Un remerciement particulier à Marie-France Pieri, Yannick Guaitella, Estelle Emanuelli, Pascal Rinaldi-Dovio, Jean-Sébastien Gualtieri, Salomé Ducos, Jean-François Andreani et bien sûr à toi Rémi Millot, mon «poto» toujours là pour moi.

— Mes collègues doctorants de l'UMR SPE, un immense merci pour ces moments partagés ! Ces deux années passées à vos côtés à Corte ont été une expérience marquante. La science peut véritablement compter sur cette jeunesse acharnée pour accomplir un travail remarquable. David, Marie, Anaïs, Céline, Eléa et Alberto, votre dynamisme et votre dévouement ont laissé une empreinte indélébile sur cette aventure. Même en tant qu'informaticien réputé pour être toujours enfermé dans son bureau, vous avez réussi à me faire vivre de merveilleux moments !
Merci à chacun de vous !

— Mon épouse Yemi Maria, ton amour, ta compréhension et tes encouragements ont été la source inépuisable de ma force et de ma persévérance. Merci d'avoir partagé ce chemin avec moi, d'avoir été ma complice et mon pilier. Ta présence a rendu cette étape bien plus significative. Je suis infiniment reconnaissant de t'avoir à mes côtés.
Reçois ici le fruit de tes sacrifices.

— Mon frère Brice Armand et mes sœurs Mireille et Nadine, je tiens à saluer la tendre complicité qui a toujours existé entre nous, une complicité qui nous a permis de surmonter tant d'obstacles. Que cette union se renforce davantage, afin que nous puissions relever ensemble les défis qui nous attendent.
Votre soutien et votre présence ont été des éléments essentiels de ma vie, et je vous exprime ma profonde gratitude.

— Rodrigue Worou et Eugène Akiyo, pour vos relectures minutieuses et vos corrections qui ont contribué à améliorer la qualité de ce document. Même les kilomètres qui nous séparent n'ont en rien altéré votre disponibilité à m'accorder de votre temps.

— Le Docteur Arouna Yessoufou, qui a toujours cru en moi et m'a constamment accompagné dans tous mes projets académiques, professionnels et personnels depuis ma tendre enfance jusqu'à ce jour.

— Toute ma famille et mes amis au Bénin, ce pays qui m'a vu naître et grandir. Je vous porte toujours dans mon cœur. Vous êtes la preuve que la distance n'arrête pas l'amour.

— Toutes les merveilleuses personnes que j'ai eu le privilège de rencontrer partout dans ce beau territoire Corse qui est aussi aujourd'hui chez moi.
Merci pour votre générosité et la camaraderie que vous m'avez offertes.

— À tous ceux que je n'ai pas cités individuellement, mais qui ont contribué, de près ou de loin, à la réalisation de ce projet.
Merci sincèrement à chacun d'entre vous pour votre contribution significative à la réussite de ce projet.

"Da u fondu di u me cori, vi ringraziu. Pudete esse fieru di voi, fieru d'avè cuntribuitu à a cuncretizazione di u me sognu d'infanzia. Quella hè di gran valuta per mè !"

Résumé

L'étude du comportement animal a toujours été un sujet d'intérêt, remontant même à la préhistoire, où les humains devaient comprendre les habitudes des espèces pour se protéger ou améliorer leur stratégie de chasse. De nos jours, la compréhension des comportements des animaux demeure une préoccupation, d'autant plus cruciale à l'ère actuelle, où les enjeux écologiques et économiques en lien avec des espèces d'intérêts pèsent lourdement sur notre société. Dans un contexte où les contraintes spatiales, temporelles, techniques et financières peuvent entraver l'étude directe du comportement animal, la modélisation et la simulation informatique ont grandement contribué à surmonter ces limites. La simulation de modèles offre la possibilité de prévoir le comportement d'espèces spécifiques et de tester divers scénarios, facilitant ainsi la prise de décisions. L'évolution rapide des techniques de modélisation, notamment avec l'avènement du Deep Learning, a considérablement renforcé l'efficacité des modèles de comportement. Cependant, ces modèles présentent souvent un défaut majeur : Plus ils sont précis, moins ils sont interprétables et explicables. De plus, leur mise en œuvre par un public non expert en informatique, tel que les biologistes, peut être laborieuse.

Pour répondre à cette problématique, cette thèse propose une nouvelle approche novatrice traitant la modélisation du comportement animal comme un problème d'optimisation. Cette méthode repose sur une base de données d'actions élémentaires dans laquelle il faut trouver les actions et paramètres optimaux reproduisant au mieux le comportement observé et décrit dans des données collectées au préalable, à l'aide de moyens technologiques tels que des capteurs ou des vidéos. La résolution de tels problèmes d'optimisation, a été faite avec des métaheuristiques, une classe de méthodes de résolution particulièrement efficace pour ce type de problème. Ainsi, nous avons proposé et développé l'approche ANIMETA, qui intègre un ensemble d'outils contribuant à la génération de modèles de comportement animal à la fois précis, interprétables et explicables.

Le système ANIMETA développé se compose de ANIMETA-MOD, un modèle générique conçu pour représenter le comportement animal par des actions élémentaires. Pour être simulé, il a été intégré dans un système multi-agents appelé ANIMETA-SMA, conçu à cet effet. L'outil ANIMETA-ENGINE, chargé de la génération des modèles proprement dite, utilise des métaheuristiques pour choisir des actions et des paramètres optimaux. L'interface entre ANIMETA-ENGINE et l'utilisateur, ainsi que d'autres systèmes informatiques, est assurée par les outils ANIMETA-HIM et ANIMETA-API. ANIMETA-HIM a été particulièrement conçu pour être simple, épuré et intuitif pour les utilisateurs. L'approche proposée ainsi que les outils développés à cet effet ont été vérifiés et validés avec quatre modèles différents, à savoir deux modèles expérimentaux, un modèle de comportement de cochon et un modèle généré à partir d'observations directes effectuées chez le Sciaena umbra. Les résultats issus des séries de tests subis par ces modèles montrent la concordance de ANIMETA avec d'autres plates-formes et démontrent sa rapidité comparée à d'autres méthodes. Cependant, quelques points restent à améliorer, notamment la durée d'optimisation qui pourrait être réduite en explorant des possibilités de parallélisation et en modifiant le processus d'évaluation des solutions. Malgré ces performances sous-optimales, ANIMETA offre quand même des résultats encourageants pour la modélisation du comportement animal. Pour cela, nos perspectives suggèrent l'amélioration des algorithmes, l'ajout de fonctions d'évaluation dédiées, et l'élargissement de la base de données d'actions élémentaires via une plateforme contributive.

Mots clés : Modélisation, simulation, Comportement animal, Métaheuristiques, Optimisation via simulation, Système multi-agents, Éthologie.

Abstract

The study of animal behavior has always been a subject of interest, dating back to prehistoric times when humans had to understand the habits of species to protect themselves or enhance their hunting strategies. Nowadays, understanding animal behavior remains a concern, especially in the current era where ecological and economic issues related to species of interest weigh heavily on our society. In a context where spatial, temporal, technical, and financial constraints can hinder direct study of animal behavior, computer modeling and simulation have greatly contributed to overcoming these limitations. Modeling simulations offer the possibility to predict the behavior of specific species and test various scenarios, facilitating decision-making. The rapid evolution of modeling techniques, particularly with the advent of Deep Learning, has significantly enhanced the efficiency of behavior models. However, these models often have a major drawback : the more accurate they are, the less interpretable and explainable they become. Moreover, their implementation by non-experts in computer science, such as biologists, can be laborious.

To address this issue, this thesis proposes an innovative approach treating animal behavior modeling as an optimization problem. This method relies on a database of elementary actions in which one must find optimal actions and parameters that best reproduce the observed behavior described in previously collected data using technological means such as sensors or videos. The resolution of such optimization problems has been done with metaheuristics, a class of resolution methods particularly effective for this type of problem. Thus, we proposed and developed the ANIMETA approach, which integrates a set of tools contributing to the generation of animal behavior models that are both accurate, interpretable, and explainable.

The developed ANIMETA system consists of ANIMETA-MOD, a prototype model designed to represent animal behavior through elementary actions. To be simulated, it has been integrated into a multi-agent system called ANIMETA-SMA, designed for this purpose. The ANIMETA-ENGINE tool, responsible for the actual generation of models, uses metaheuristics to select optimal actions and parameters. The interface between ANIMETA-ENGINE and the user, as well as other computer systems, is ensured by the tools ANIMETA-HIM and ANIMETA-API. ANIMETA-HIM was specifically designed to be simple, streamlined, and intuitive for users. The approach and tools developed for this purpose were verified and validated with four different models, namely two experimental models, a pig behavior model, and a model generated from direct observations of Sciaena umbra. The results from the test series undergone by these models show the concordance of ANIMETA with other platforms and demonstrate its speed compared to other methods. However, some points still need improvement, particularly the optimization duration that could be reduced by exploring possibilities for parallelization and modifying the solution evaluation process. Despite these few suboptimal performances, ANIMETA still offers encouraging results for animal behavior modeling. Therefore, our perspectives suggest improving certain algorithms, adding dedicated evaluation functions, and expanding the database of elementary actions through a collaborative platform.

Keywords : Modeling, Simulation, Animal Behavior, Metaheuristics, Simulation-based Optimization, Multi-Agent System, Ethology.

Table des matières

Introduction générale

Contexte de la recherche

Depuis les prémices de la réflexion humaine jusqu'à l'évolution continue de notre société, l'exploration du comportement animal a toujours captivé l'esprit de l'humain. Cette fascination pour les subtilités du comportement de l'animal a joué un rôle capital dans notre propre survie. En remontant à la préhistoire, nos ancêtres manifestaient déjà un intérêt pour les subtilités du comportement animal, car ceux-ci se devaient de comprendre les habitudes de différentes créatures pour déterminer quelles étaient les espèces à craindre et celles qui pouvaient être exploitées, que ce soit dans le cadre de la chasse, de l'élevage ou de la domestication (Giraldeau et al. 2015). À cette ère primitive, la compréhension du comportement des animaux constituait un atout crucial, puisqu'elle pouvait même être déterminante pour la survie d'une tribu entière.

De nos jours encore, ces animaux pris avec les écosystèmes auxquels ils appartiennent demeurent indispensables pour la survie de l'humanité, au regard des fonctions vitales qu'ils assurent. Elles vont de la pollinisation des cultures par les insectes à la régulation des maladies par des prédateurs naturels ; de la purification de l'eau par des organismes filtrants à la fourniture de nourriture et de médicaments issus de divers organismes.

Face aux enjeux écologiques de notre ère, et au regard de l'état de dégradation actuel de la biodiversité [1](EO Wilson et al. 1988 ; Descombe 2022), la compréhension du comportement des animaux est un sujet qui reste toujours d'actualité. Cette réalité exige des réponses novatrices et impératives pour la préservation de la diversité biologique (Ceballos et al. 2015) au risque de compromettre les fonctions vitales qu'elle assure. Dans cette quête de compréhension et parmi les techniques proposées par les différents acteurs (Tsibulsky et al. 2007 ; Ericsson et al. 2013 ; Butts et al. 2022), la modélisation du comportement animal émerge comme un outil essentiel pour créer des modèles capables de reproduire des

1. Les dernières données de la liste rouge mondiale de l'Union Internationale pour la Conservation de la Nature font état de plus de 42 000 espèces en danger d'extinction sur environ 1,2 million qui sont actuellement répertoriées

comportements d'animaux dans des scénarios définis selon les besoins afin d'anticiper l'impact d'un tel scénario dans la vie réelle. Par exemple, en simulant l'évolution de la population d'une espèce clé, nous pouvons mieux appréhender son impact potentiel sur la biodiversité et donc aider à orienter les décisions en matière de conservation et de gestion durable des ressources naturelles. La Corse, avec son littoral riche en espèces marines, ne se soustrait pas à ces préoccupations, d'autant plus que sa biodiversité joue également un rôle essentiel dans l'économie régionale. C'est donc face à ces défis que l'Université de Corse, à travers sa plateforme de recherche STELLA MARE, œuvre pour une compréhension approfondie des espèces présentes sur les côtes corses en mobilisant des approches novatrices issues de l'ingénierie écologique et de l'informatique.

L'Unité d'Appui et de Recherche 3514 (UAR) STELLA MARE [2] (Sustainable TEchnologies for LittoraL Aquaculture and MArine REsearch), créée en 2010 et labellisée en 2011 par le CNRS, s'investit activement dans des projets de recherche visant à gérer de manière intégrée les ressources halieutiques du littoral Corse, et de favoriser une aquaculture durable. Dans la poursuite de ces objectifs, une des actions majeures de l'UAR STELLA MARE est la restauration des écosystèmes dégradés ou en voie de l'être en relâchant dans les milieux naturels des individus produits en laboratoire. Pendant ces actions de restauration, une question fondamentale fréquemment posée par les scientifiques concerne le moment, le lieu, la manière et la quantité d'individus à relâcher pour obtenir un meilleur taux de survie et restaurer l'équilibre de l'écosystème en place. Pour répondre à ces questions, il a été mis sur pied au sein de l'unité le programme de recherche intutilé : « Modélisation Comportementale ». L'objectif de ce programme est de développer un Système d'Information dédié au suivi global des espèces halieutiques permettant à terme d'approfondir la compréhension du comportement des espèces marines. Ceci est particulièrement essentiel compte tenu de la complexité de certains écosystèmes et des contraintes qu'ils imposent. En réalité, il n'est pas toujours possible de créer des conditions d'observation viables, surtout dans des milieux où les interventions humaines sont naturellement limitées. Ainsi, le programme de recherches s'articule autour de deux axes majeurs :

— Axe 1 : La création d'un Système d'Information dédié au suivi global d'espèces halieutiques, basé sur les réseaux de capteurs sans fil (Manicacci et al. 2022) ;
— Axe 2 : Le développement et la validation de modèles de simulation comportementale basés sur les données recueillies .

C'est précisément dans le cadre de l'axe 2 que se situe notre projet de thèse.

Problématique de recherche

Le domaine de la modélisation et de la simulation du comportement animal a connu l'évolution de plusieurs techniques et moyens au fil du temps, permettant d'obtenir des prédictions de comportements d'une remarquable précision. Parmi ces techniques, qui vont de la création de modèles basés sur des équations à ceux élaborés à partir de données, les algorithmes d'apprentissage automatique (ou Deep learning) occupent une place prépondérante. Ils représentent une révolution significative dans le domaine de l'intelligence artificielle en général,

2. https://stellamare.universita.corsica

et plus particulièrement dans la modélisation et la simulation.

Cependant, un obstacle majeur entrave actuellement leur utilisation : la difficulté à comprendre et à expliquer les modèles générés. Cet état des choses soulève la question fondamentale de l'interprétabilité et de l'explicabilité des modèles (Mittelstadt et al. 2019 ; Linardatos et al. 2021).

En effet, l'interprétabilité concerne la compréhension générale d'un modèle, fournissant les informations nécessaires pour démontrer son fonctionnement. En revanche, l'explicabilité va au-delà en offrant des explications détaillées et compréhensibles, même pour un néophyte, concernant chaque décision prise par le modèle. En d'autres termes, l'interprétabilité relève de l'expertise, répondant à la question du « comment » le modèle prend ses décisions, tandis que l'explicabilité répond à la question « pourquoi ». Aujourd'hui, la plupart des modèles générés à partir des données utilisent des méthodes d'apprentissage automatique (deep learning de type « boîte noire »). Ces modèles offrent une grande précision dans leurs prédictions, mais demeurent souvent peu interprétables et peu explicables (Abdullah et al. 2021). De plus, leur mise en œuvre peut se révéler fastidieuse pour un public de biologistes. Dans le contexte de l'UAR 3514 STELLA MARE, où les chercheurs sont des biologistes et où une compréhension approfondie du comportement des espèces demeure nécessaire, ces méthodes de modélisation ne représentent pas la solution idéale.

Compte tenu de ce contexte et du cadre établi par le programme de recherche, la question fondamentale de notre réflexion est : *Comment pouvons-nous proposer aux biologistes un outil leur permettant de générer, à partir des données, des modèles de comportement animal à la fois précis, interprétables et explicables ?* Une réponse à cette question peut être tout à fait apportée par l'optimisation via la simulation (Andradóttir 1998).

Contributions

Dans nos travaux de recherche, nous avons apporté une réponse à cette question qui a consisté à la proposition d'une approche incrémentale et itérative, considérant la modélisation du comportement animal comme un problème d'optimisation à résoudre à l'aide de méthodes de la classe des métaheuristiques. Cette approche repose sur une hypothèse simple selon laquelle : « il serait possible de modéliser le comportement d'une espèce animale avec des actions élémentaires provenant d'autres espèces animales ». Concrètement, notre démarche consiste à constituer en un premier temps, une base de données d'actions élémentaires. Ainsi, lors du processus de la modélisation d'un comportement, nous cherchons ensuite une combinaison d'actions et le paramétrage associé. L'objectif est d'obtenir un modèle optimal reproduisant au mieux le comportement décrit dans le jeu de données de référence, préalablement collecté sur l'espèce à modéliser. Une particularité de cette approche est qu'elle inclut directement l'expert (le biologiste) dans le processus de modélisation pour «guider» l'optimisation et résoudre les problèmes d'équifinalité.

Pour ce faire au cours de cette thèse, nous avons :

— Proposé un modèle générique visant à représenter le comportement animal sous la forme d'une série d'actions élémentaires à exécuter dans un ordre spécifique. Pour simuler les modèles élaborés sur cette base, nous avons proposé également un simulateur dédié pour

exécuter les scénarios définis en fonction des séquences d'actions spécifiées. L'ensemble a été intégré dans un Système Multi-Agents développé à cet effet.

— Proposé une adaptation de quatre métaheuristiques phares selon leur spécificité, pour tirer parti de leurs mécanismes d'exécution afin de générer à la fois les séquences d'actions élémentaires et les paramètres associés. Il a s'agissait concrètement de personnaliser ces algorithmes en fonction des caractéristiques spécifiques de notre approche, favorisant ainsi la convergence vers des modèles optimaux.

— Proposé un ensemble d'outils composé d'une Interface Homme-Machine, d'une API et d'une plateforme de simulation pour exploiter convenablement l'approche proposée. Ces outils sont intuitifs et accessibles pour aider lors de la génération de modèles de simulation de comportement animal, ceci en conservant un niveau de précision acceptable selon les critères du modélisateur.

Organisation du document

Dans ce document, la présentation des travaux réalisés au cours de cette thèse est structurée en deux parties

La première partie explore la piste de la modélisation du comportement des animaux comme un problème d'optimisation. Nous y dressons un état de l'art sur l'étude du comportement animal, la modélisation et simulation et enfin les problèmes d'optimisation.

Le chapitre 1 aborde l'étude et la modélisation du comportement animal. Il commence par fournir un aperçu des étapes clés qui ont marqué l'évolution de l'étude du comportement au fil du temps. En parcourant ces étapes, il devient évident que les différents courants de pensée ont rencontré des limites, lesquelles peuvent être surmontées grâce à l'application de techniques de modélisation et de simulation. Le chapitre se poursuit donc par une introduction au domaine de la modélisation et de la simulation, particulièrement leur application spécifique aux comportements d'animaux. À partir des connaissances acquises et en accord avec notre problématique de recherche, nous concluons le chapitre en mettant en lumière notre hypothèse selon laquelle il est possible de générer des modèles explicables de comportement animal à l'aide des méthodes d'optimisation.

Dans le chapitre 2, l'attention se porte sur la résolution des problèmes d'optimisation, en mettant l'accent sur les métaheuristiques, une catégorie de méthodes reconnues pour leur efficacité dans la résolution de ce type de problème. En décrivant en détail le processus d'exécution des principales métaheuristiques largement reconnues, nous posons les bases nécessaires à l'application de celles-ci, conformément à la problématique énoncée dans le chapitre 1.

La deuxième partie est consacrée à la présentation de nos différentes propositions pour résoudre notre problématique. Elle présente donc ANIMETA, notre approche permettant la modélisation du comportement animal grâce aux métaheuristiques.

Dans le chapitre 3, nous présentons ANIMETA-MOD, le modèle générique de comportement animal dont le fonctionnement repose sur une série d'actions déterminées. Son architecture a été soigneusement élaborée pour s'aligner avec les exigences et les nuances des métaheuristiques pour une intégration dans le processus d'optimisation.

Dans le chapitre 4, nous dévoilons en détail notre approche dite de « Modélisation guidée par l'optimisation » dans le contexte de la génération de modèles de comportements d'animaux. Après avoir décrit le processus de modélisation, nous montrons comment elle peut être mise en œuvre avec les métaheuristiques. En guise d'illustration, nous avons choisi quatre métaheuristiques (algorithmes génétiques, de colonie de fourmis, des systèmes immunitaires par clonage et de recherche harmonique) pour lesquelles nous montrons, pour chacune, les adaptations nécessaires. Le chapitre se termine par la présentation de possibles fonctions d'évaluation utilisables.

Le chapitre 5 expose le processus de vérification et de validation de ANIMETA. Ce processus ayant porté sur plusieurs types de modèles a permis d'une part de s'assurer que l'approche a été implémentée et fonctionne comme elle a été pensée et conçue, et d'autre part il a permis d'évaluer ses performances et sa capacité à proposer des modèles qui reproduisent de façon optimale, les comportements attendus.

Le document se termine enfin par une conclusion qui fait une synthèse critique des travaux menés puis et présente des perspectives intéressantes qui peuvent être envisagées.

Première partie

Vers la modélisation du comportement des animaux comme la résolution d'un problème d'optimisation

CHAPITRE 1

Étude et modélisation du comportement animal

Sommaire

Introduction

Durant ces dernières décennies, la compréhension du comportement animal est devenue un défi majeur pour l'humanité, en particulier pour la communauté scientifique qui a besoin d'outils pour aider à la gestion des écosystèmes [1]. Disposer de tels outils permettant de mieux comprendre le comportement des animaux contribue manifestement à l'élaboration de stratégies de protection, mais aussi d'un autre côté pour les professionnels, il peut représenter un atout incontournable pour augmenter le rendement des unités de production.

Dans ce chapitre, nous mettons en évidence l'exploration du comportement des animaux et de sa compréhension, en commençant par une brève rétrospective des divers courants de pensée et des aspects multidisciplinaires de l'étude du comportement animal qui ont permis d'établir les fondations de la recherche moderne sur le comportement animal, avec une emphase particulière sur l'éthologie en raison de ses contributions essentielles dans le domaine. Cependant, étant principalement basée sur l'observation, l'éthologie atteint rapidement ses limites, ce qui conduit à l'intégration du domaine de la modélisation et de la simulation pour aller au-delà de ces limites.

1.1 Histoire de l'étude du comportement animal : de la philosophie à l'éthologie

L'intérêt que nous portons aujourd'hui à l'étude du comportement des animaux pourrait potentiellement remonter jusqu'aux origines de l'humanité et aurait d'ailleurs contribué d'une manière ou d'une autre à la survie et à l'évolution de l'espèce humaine. En effet, bien qu'étant primitif, l'homme se devait déjà de comprendre les animaux, leur environnement et leur comportement pour savoir quels animaux il pouvait chasser, pêcher ou domestiquer. Pour nous, il s'agit déjà là des prémices de l'étude du comportement animal.

Plus concrètement dans l'histoire, l'étude des animaux et de leur comportement commence avec Aristote [2] qui posa véritablement les bases de la psychologie et de la biologie des animaux. La théorie d'Aristote sur le vivant suppose que tous les êtres vivants ont une âme. Pour vivre, il faut alors avoir une âme, qui est le siège de la pensée. Il affirmera d'ailleurs dans son ouvrage *History of Animals*

> *«Les animaux ont naturellement une certaine faculté de participer à toutes les affections que l'âme peut éprouver, la prudence et l'audace, le courage et la lâcheté, la douceur et la cruauté, et tous les autres sentiments analogues. Il y en a même qui sont, dans une certaine mesure, susceptibles d'apprendre et de s'instruire, tantôt les uns par les autres, tantôt sous la main de l'homme, pourvu qu'ils aient le sens de l'ouïe, et non seulement tous ceux qui entendent les sons, mais ceux qui peuvent percevoir les différences des signes et les distinguer»*(Hilaire et al. 1883).

1. https://stellamare.universita.corsica
2. (384-322 av. J.-C.)

Cependant, Aristote apporte certaines clarifications concernant l'âme. Toutes les âmes ne sont pas équivalentes. Les végétaux possèdent une âme nutritive, les animaux ont une âme sensitive, tandis que les êtres humains possèdent une âme cognitive, intellectuelle ou réflexive. Ce point de vue a été à la fois soutenu par certains et rejeté par d'autres, ce qui a donné naissance à différents courants de pensée. Au fil du temps, ces courants ont évolué en réponse aux nouvelles connaissances, tels que le courant de Descartes, le darwinisme, la psychologie comparée, le canon de Morgan ainsi que l'éthologie. Chacun de ces courants, que nous présenterons par la suite, possède sa propre conception du comportement animal et les conceptualise en conséquence.

1.1.1 Le Comportement animal selon Descartes

Avant de se constituer en tant qu'un domaine de la science, l'étude du comportement animal est née au début du XVII^{ieme} siècle par des courants philosophiques qui montraient les rapports entre les animaux et l'Homme. Un des pionniers de ce courant fut René Descartes (1596-1650), un mathématicien et philosophe. Dans ses travaux visant à comprendre l'homme, il pense qu'étudier le comportement des animaux pouvait expliquer certains comportements de l'Homme. Selon sa conception, l'Homme et les animaux ont en commun un corps qui exécute de façon mécanique des actions. Ce qui les rend différents dans l'exécution de ces actions, c'est que l'un (l'Homme) est doté d'une âme qui lui donne la faculté de raisonner, d'apprendre et d'agir selon un processus décisionnel (Giraldeau et al. 2015). Cela implique ainsi que si l'on veut comprendre le comportement d'un humain, en particulier, les comportements élémentaires ou involontaires qui ne nécessitent pas un processus décisionnel, il suffira d'étudier le comportement des animaux. De base, l'idée de Descartes n'est pas de comprendre le comportement des animaux, puisque l'étude du comportement des animaux n'est qu'un passage pour comprendre le comportement de l'Homme. De cette façon de voir les choses du point de vue de Descartes, deux points sont particulièrement intéressants :

— Une espèce (comme une espèce animale) peut permettre de mieux comprendre le comportement d'une autre espèce (l'humain dans ce cas) ;
— Le comportement d'un animal peut être considéré comme une séquence d'actions élémentaires déclenchée «automatiquement» par des stimuli.

En passant de la perspective de Descartes à celle de Darwin avec le Darwinisme, l'attention se déplace de l'accent mis sur la séparation entre l'esprit humain et le monde animal vers une compréhension progressive de la continuité et de l'adaptation du comportement.

1.1.2 Le Darwinisme

Le darwinisme (Van Den Bergh 2018) de Charles Darwin (1809-1882) a été révélé dans sa publication intitulée «On the Origin of Species by Means of Natural Selection» (Darwin 1859). Dans cet ouvrage, il évoque la théorie dite de l'évolution selon laquelle, il existerait une continuité entre animaux et humains qui auraient des ancêtres communs et lointains. La différence entre animaux et humains serait alors due à une évolution de chacun d'eux pour répondre à leurs besoins et s'adapter à leur environnement. Cela voudrait donc dire que si l'un excelle dans une activité et l'autre non, et vice-versa, ce ne serait pas parce qu'il ne sait pas le faire, mais plutôt parce qu'il ne sait pas bien le faire. Il s'agirait d'une approche de nature quantitative et non qualitative. Ce que sait faire une espèce (animale ou humaine)

résulte d'une adaptation à son environnement dans un but ultime qui est sa propre survie et celle de son espèce plus généralement. En prenant le cas de l'intelligence humaine face à celle d'un animal pour illustrer ce principe, l'humain serait peut-être plus intelligent que l'animal, mais il n'est pas question que l'animal soit dépourvu de cette faculté.

De la perspective évolutionniste de Darwin, d'autres courants de pensée mettront en avant un autre aspect du comportement, à savoir celui de l'apprentissage, sur lequel s'appuieront la psychologie comparée et le behaviorisme.

1.1.3 La Psychologie comparée et Behaviorisme

George Romanes (1848-1894), disciple de Darwin, s'appuie sur les travaux de ce dernier, notamment ceux portant sur l'expression des émotions chez l'humain et les animaux (Darwin 1872), pour fonder sa conception de l'intelligence animale (Romanes 1882). En effet, Darwin, dans sa logique de l'évolution des espèces, a montré qu'il existe de nombreuses similitudes entre le comportement animal et humain. C'est précisément cette démarche comparative qui pousse Romanes à aborder la compréhension du comportement animal sous un angle purement anthropomorphique [3]. La psychologie comparée s'oppose ainsi à la théorie de Descartes en attribuant également aux animaux la capacité d'imiter et d'apprendre. Pour démontrer cette théorie, Romanes s'appuie sur des anecdotes, notamment celle d'un chat qui parvient à ouvrir la porte d'entrée d'une maison. Selon Romanes, *«l'usage coordonné qu'il faisait de ses pattes avant pour ouvrir la porte, tout en faisant usage d'une patte arrière pour la pousser, suffisait à Romanes pour conclure que ce comportement compliqué ne pouvait avoir été acquis autrement que par l'imitation des actions de son maître»* (Giraldeau et al. 2015). Ce sont là les fondements du behaviorisme [4], qui définit le comportement d'un individu comme un mécanisme fortement influencé par son environnement. Les agissements de cet individu sont donc le résultat des succès et des échecs qu'il rencontre dans son environnement. C'est encore une fois ce que révèle l'anecdote du chat qui ouvre une porte. Ce comportement ne pouvant pas être inné, il résulte alors d'un mécanisme d'apprentissage motivé certainement par des échecs antérieurs. Le comportement, selon le behaviorisme, est donc le résultat d'un processus (complexe) d'apprentissage par paliers successifs dans un environnement. Pour réaliser leurs expérimentations, les behavioristes créent un cadre d'apprentissage (le conditionnement) dans lequel les sujets sont placés. Sur la base des observations effectuées, ils peuvent ainsi tirer des conclusions ou en apprendre sur le comportement du sujet observé et, plus largement, sur son espèce.

Cette approche va être jugée «complexe» pour certains courants de pensée notamment celui porté par Conwy Lloyd Morgan qui mettra en avant lui la simplicité des explications des comportements.

1.1.4 Le canon de Morgan

Conwy Lloyd Morgan (1852-1936), un autre disciple de Darwin, s'oppose à l'approche anthropomorphique de la psychologie comparée de Romanes. Il fonde son opposition sur le fait que la psychologie comparée est très objectiviste. Elle a tendance à écarter délibérément les données subjectives pour ne s'en tenir qu'à ce qui est contrôlable par les sens. En effet,

3. Attribution aux animaux des réactions et des sentiments propres à l'espèce humaine.
4. «Behaviorisme» vient du mot anglais «behavior» qui signifie «comportement».

les résultats de Romanes ne reposaient pas sur une approche scientifique rigoureuse, mais plutôt sur des anecdotes. Morgan instaure donc son approche d'étude comportementale à travers un canon qui stipule que : «Un comportement ne doit pas être expliqué par un mécanisme mental complexe lorsqu'il peut tout aussi bien l'être par un mécanisme plus simple». (Sober 1998). Ce canon met ainsi en avant le principe de *parcimonie* dans l'étude du comportement. Un principe qui consiste à utiliser le minimum de causes élémentaires pour expliquer un phénomène. Néanmoins, il ne fait pas de son approche une vérité absolue, ce qui justifie d'ailleurs l'usage de *«tout aussi bien»* dans son canon. Cela signifie ainsi qu'il ne faut pas conclure immédiatement que l'on peut trouver d'emblée une explication simple, mais en réalité il s'agit de trouver une explication qui rend compte au mieux du phénomène (Giraldeau et al. 2015).

Après avoir exploré ces principes «simplistes» du canon de Morgan, nous nous orientons à présent vers l'éthologie qui se penche sur l'aspect naturel du comportement animal afin de mieux appréhender les intrications qui peuvent exister entre les animaux et leur écosystème.

1.1.5 L'éthologie, science de l'étude du comportement animal

L'éthologie est née au 20^{eme} siècle et se place comme la discipline scientifique rigoureuse qui fait une étude biologique du comportement. En effet, comme nous avons pu le constater jusqu'ici, les autres disciplines qui étudient le comportement, et en particulier le comportement des animaux, se focalisent plutôt sur les aspects philosophiques et psychologiques. Elle s'appuie quand même sur les autres disciplines, notamment le behaviorisme, pour lequel l'éthologie présente une méthodologie différente.

D'antan, les béhavioristes se focalisaient sur l'apprentissage des comportements, c'est-à-dire des comportements acquis, bien qu'ils admettent l'existence des comportements innés. Ils validaient leurs hypothèses à travers des expérimentations avec des sujets dans un conditionnement purement artificiel. L'éthologie, quant à elle, va étudier les comportements à travers une méthodologie basée sur l'observation dudit comportement dans son état naturel.

Dans la littérature, nous retrouvons déjà des travaux qui ont posé les bases de l'éthologie au sens naturiste de l'étude du comportement animal. Les pionniers de cette démarche sont principalement Charles Otis Whitman (1842-1910) et Wallace Craig (1876-1954) (Giraldeau et al. 2015). L'éthologie, telle qu'elle est formalisée comme science du comportement, prend effectivement corps à partir des travaux et des écrits de Konrad Lorenz (1903-1989) et Nikolaas Tinbergen (1907-1988), considérés comme les pères fondateurs de l'éthologie moderne. Ils ont posé les bases formelles de cette science. Prenant ses sources dans le darwinisme [5], l'éthologie est cette science qui étudie le comportement animal dans son état naturel en tenant compte des interactions entre lui et son environnement. L'éthologie vient en réaction au Behaviorisme de John Broadus Watson.

L'éthologie se débarrasse des stéréotypes anthropomorphiques (utilisés en psychologie comparée et dans le behaviorisme) en reposant sur des buts et méthodes propres à elle, fondés sur l'observation plutôt que sur l'expérimentation et le conditionnement (Tinbergen 1963). Le principe de stimulus-réaction promu par le behaviorisme est donc ainsi remis en cause. L'objectif n'est plus de voir l'action que provoque un stimulus, mais plutôt d'observer un comportement dans son état naturel et d'inférer les causes. L'éthologie ne cherche alors

5. Théorie d'après laquelle les espèces évoluent selon les lois de la sélection naturelle présentée à la section 1.1.2

pas à comprendre comment se comporte le sujet, mais veut plutôt comprendre pourquoi le sujet se comporte d'une manière ou d'une autre. Elle se considère à part entière comme une branche de la biologie. C'est ce que va exposer Tinbergen (**amy__les__2006** ; Bateson et al. 2013) pour expliquer les fondements de l'éthologie. Pour lui, l'éthologie se pose quatre questions, dont trois sont du même type qu'en biologie :

1. *Quelles sont les causes proximales des comportements ?*
   Un comportement observé chez un sujet peut être la réponse à des facteurs qui peuvent être endogènes ou exogènes. Prenons l'exemple d'un animal qui se met en situation de camouflage : la vue d'un prédateur pourrait amener le sujet à se camoufler. Il s'agit là d'un facteur externe à ce comportement. Ce même comportement pourrait également être hormonal dû à une situation de stress observée chez le sujet, lorsqu'il est en situation de stress. Il s'agit là d'un facteur interne. Dans le cas de certains comportements, les facteurs sont liés. Le facteur externe implique un facteur interne qui, à son tour, déclenche le comportement.
2. *Quelles sont les causes ultimes des comportements ?*
   L'éthologie s'intéresse à la fonction du comportement, autrement dit à son but. Étant fortement inspirée du darwinisme, l'éthologie relie les différents comportements des animaux à leur survie. Déterminer les causes ultimes d'un comportement revient à déterminer la valeur d'un comportement pour la survie de l'individu. Pour expliquer cela, prenons l'exemple donné dans (**amy__les__2006**) qui concerne les chants d'oiseaux. Quelle pourrait être la fonction ultime de ce comportement ? Les chants sont le plus souvent émis par des oiseaux mâles. Ces chants sont associés à deux fonctions principales : la première est la défense de leur territoire et la seconde est l'attraction d'une partenaire pour la reproduction (Nowicki et al. 2005). Ces deux fonctions visent un objectif ultime qui est d'assurer la survie de l'individu ou de l'espèce toute entière.
3. *Quelle est l'ontogenèse des comportements ?*
   Les éthologues font la distinction entre les comportements innés et les comportements instinctifs. Les comportements innés sont présents dès la naissance ou se développent spontanément sans nécessiter d'apprentissage. Par exemple, le réflexe de succion chez les mammifères est un comportement inné. Ce type de comportement a la particularité d'être constant, prévisible et commun à tous les individus d'une même espèce. Quant aux comportements instinctifs, ils sont également présents dès la naissance, mais ils sont souvent déclenchés par des stimuli et nécessitent parfois un apprentissage complémentaire. Chez les oiseaux par exemple, le comportement de nidification peut être considéré comme instinctif dans la mesure où ils savent relativement construire un nid ou bien ont préalablement observé la construction d'un nid, mais ils ne le construisent que lorsqu'ils ont à leur disposition les matériaux nécessaires et un emplacement approprié dans leur environnement. Déterminer l'ontogenèse d'un comportement consistera à identifier s'il est inné et transmis dans le patrimoine génétique, ou sinon à identifier comment il apparaît et à quel moment. En reprenant l'exemple des chants d'oiseaux, l'éthologue cherche à savoir si l'oiseau sait-il chanter ou l'apprend-il, et à quel moment cet apprentissage devient-il indispensable pour sa survie ?
4. *Quelle est la phylogenèse des comportements ?*
   Ici, on s'intéresse à comment le comportement est apparu au cours de l'évolution de

l'espèce. C'est sans doute la question la plus difficile à laquelle on peut répondre, car l'éthologie étant basée sur l'observation de l'instantané, il est difficile de remonter dans le temps pour se rendre compte de l'évolution d'un comportement.

En fin de compte, au travers de ces quatre questions, l'éthologie a pour objectif principal d'expliquer le «comment» et le «pourquoi» des comportements animaux. Le «comment» se rapporte aux causes immédiates et aux actions concrètes mises en œuvre pour réaliser ces comportements. Le «pourquoi», quant à lui, se focalise sur la fonction des comportements et leurs causes ultimes, c'est-à-dire les raisons évolutionnaires qui ont conduit à leur développement.

De la naissance de l'éthologie jusqu'à nos jours, l'étude des comportements animaux a connu des évolutions qui ont donné lieu à deux approches distinctes pour représenter et examiner ces comportements : l'éthologie objectiviste (ou classique), et l'éthologie fonctionnelle.

1.1.5.1 Éthologie objectiviste ou classique

Cette approche originelle de l'éthologie, proposée par Konrad Lorenz, instaure une hiérarchie des comportements en fonction de leur contribution à un but à atteindre. Cet aspect du comportement est mis en avant par Lorenz, qui fondera ses travaux sur l'étude de la cause du comportement. Cette approche éthologique de Lorenz, dite classique (ou causale ou objectiviste), introduit deux notions de base pour expliquer un comportement. Il s'agit du *Fixed Action Pattern (FAP)*[6] et du *Innate Releasing Mechanism (IRM)*[7] (Lorenz 1950).

L'IRM est un ensemble de stimuli qui déclenche un FAP, c'est-à-dire une unité de comportement. Elle est alors une cause et un FAP l'effet visible. Autrement dit, les IRM activent les FAP. Ce «principe de cause à effet» se différencie du behaviorisme par le fait qu'ici, les stimuli ne se limitent pas qu'à la perception de l'environnement, mais aussi aux causes endogènes, comme par exemple la motivation du sujet à réaliser cette action. Plus explicitement, lorsqu'un sujet reçoit un stimulus, celui-ci active un mécanisme de déclenchement inné[8] (IRM) qui aboutit à l'expression d'une unité de comportement (FAP). La manière dont cette unité de comportement s'exprime est définie par le mécanisme de déclenchement, en prenant en compte la force du stimulus, mais aussi d'autres facteurs comme les ressources disponibles (ex : motivation, énergie) et les facteurs qui inhibent.

Afin de faire comprendre donc son approche, Konrad Lorenz propose une illustration très célèbre faisant une analogie entre l'écoulement d'eau dans un modèle hydromécanique et le déclenchement d'un comportement (Coquelle 2005). Dans le modèle hydromécanique présenté à la figure 1.1, on peut voir un robinet T symbolisant une production d'énergie stockée dans le réservoir R. La quantité d'eau disponible dans le réservoir représente la quantité d'énergie (la ressource) disponible pour faire des actions. La valve V représente le mécanisme déclencheur et le ressort S représente les forces inhibitrices. La masse Sp symbolise la partie des stimuli de force variable provenant des perceptions. Les écoulements Tr selon le niveau G atteint symbolisent la manière (ou la force) dont le FAP activé s'exprime. Le

6. Traduction anglaise de «Modèle d'action fixe»
7. Traduction anglaise de «Mécanisme de déclenchement inné»
8. C'est-à-dire prédéfini

but visé par ce modèle est de montrer que le comportement observé est le résultat d'un mécanisme dit d'instinct qui combine des facteurs internes et externes.

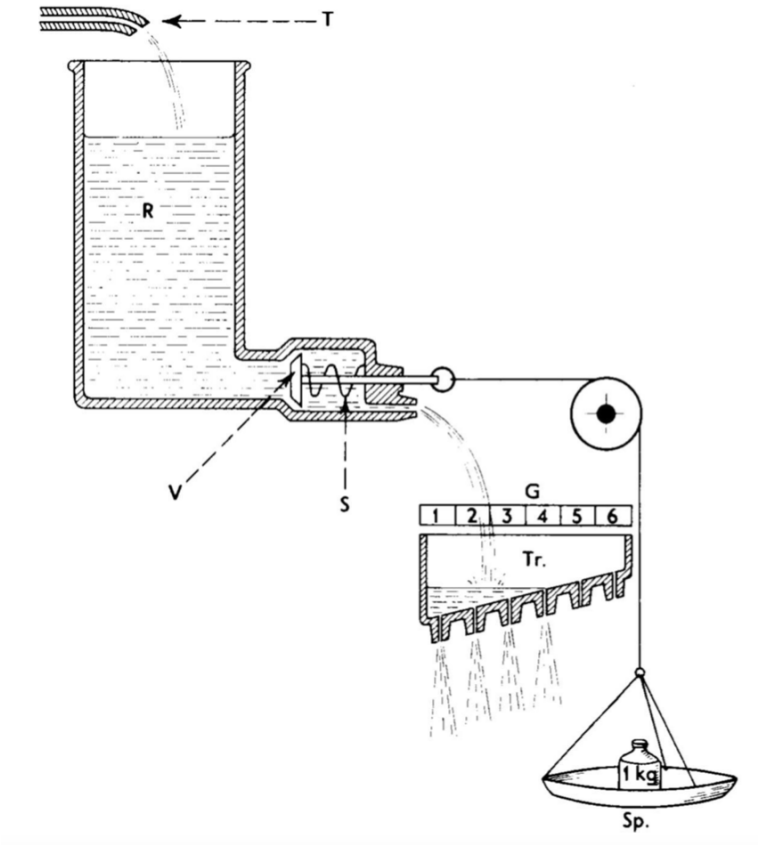


FIGURE 1.1 – Modèle d'activation hydromécanique de Lorenz (LORENZ 1950).

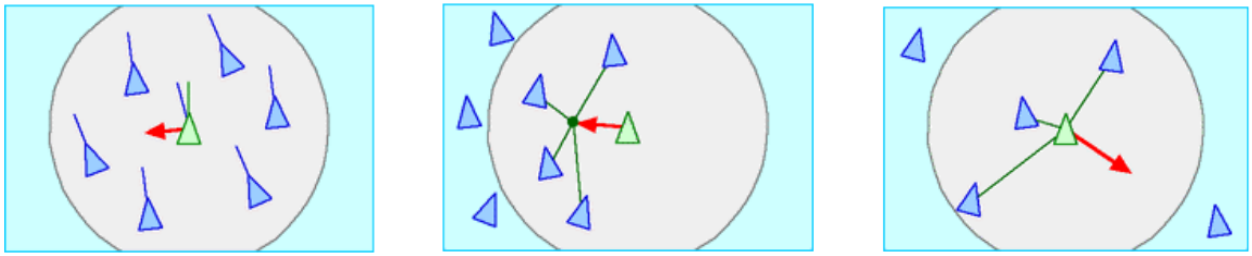
1.1.5.2 Éthologie fonctionnelle : L'éthologie d'aujourd'hui

Pendant plusieurs années, le modèle objectiviste de Lorenz est resté (et reste encore) le modèle de référence des études en éthologie. Mais il va très vite montrer ses insuffisances à cause de son organisation qui ne permettait pas d'appliquer le modèle à des organismes complexes. En effet, des comportements relativement simples peuvent bien se représenter avec ce modèle dans la mesure où ils sont déclenchés par des stimuli exclusifs connus, dont l'expression observable peut être décrite par une série d'actions. Dans la pratique, ce n'est pas toujours le cas. Il peut bien y avoir au même moment différents stimuli pouvant également déclencher différents comportements. Prenons l'exemple d'un sujet qui perçoit à la fois dans son champ de vision un(e) partenaire pour la reproduction, une proie et un prédateur. Ces trois stimuli déclenchent des comportements tout à fait différents, bien qu'ils concourent tous au but ultime de survivre et de perpétuer son espèce. Dans un tel contexte, un problème de «choix» se pose.

Pour régler ce problème, le modèle fonctionnel de l'éthologie d'aujourd'hui reprend les principes de l'éthologie objectiviste et propose une hiérarchisation des comportements (MCDOUGALL

1926 ; Baerends 1976). Le principe ici n'est pas de disposer à proprement parler d'une liste ordonnée de comportements déclenchés par niveau, mais plutôt de réguler l'expression des comportements déclenchables de sorte que l'expression observable qui en résulte soit le fruit de plusieurs processus parallèles dynamiques et chaotiques.

Un exemple illustrant parfaitement cette situation est celui du modèle BOIDS (Reynolds 1987). Il modélise le vol d'une nuée d'oiseaux [9] sur la base des observations naturelles. Le modèle décrit le comportement du groupe à partir du comportement de chacun des individus. Concrètement, un individu a la capacité de percevoir ses congénères et de se déplacer en fonction de leur position. Ce déplacement est décrit par trois «sous-comportements», l'alignement, la cohésion et la séparation (illustrés à la figure 1.2) dont l'expression conjointe définit la trajectoire finale.



(a) Alignement : chaque individu s'aligne sur ces proches voisins pour suivre le même chemin

(b) Cohésion : Les individus se rapprochent tout en gardant un écart pour rester groupés

(c) Séparation : Lorsque les individus sont trop proches, ils se séparent pour éviter la collision

Figure 1.2 – Comportement d'une nuée d'oiseaux décrit par le programme BOIDS

Qu'il s'agisse de l'éthologie classique connue aux origines ou bien de l'éthologie d'aujourd'hui, les deux approches sont basées sur l'observation et la description la plus détaillée possible. Cette description est consignée dans un document appelé «éthogramme».

L'éthogramme est un catalogue où sont consignés les comportements d'une espèce ou d'un sujet observé dans un contexte donné. Nous portons un intérêt particulier à l'éthogramme car, par sa fonction, il constitue une véritable source de connaissance sur le comportement de l'espèce qu'il décrit.

1.1.5.3 L'éthogramme

L'éthogramme encore appelé «répertoire de conduites» ou «répertoire comportemental» peut être défini comme un «*ensemble de toutes les unités de conduite possibles d'un individu dans son environnement naturel, ou l'inventaire systématisé de toutes ses règles naturelles de conduite*»(Cheyssac 2015). Il s'agit d'une description minutieuse des comportements d'une espèce en langage naturel.

Un éthogramme peut être général et présenter les principales activités de l'espèce, ou peut être spécifique en se focalisant sur les comportements sociaux, territoriaux, reproducteurs, parentaux, communicatifs, alimentaires ou locomoteurs. Dans la littérature, nous retrouvons trois classes d'éthogrammes, qui sont (selon (Bakeman et al. 2011) et cité par (Cheyssac 2015)) :

9. Le modèle peut être appliqué au déplacement des troupeaux ou des bancs de poissons.

— **Éthogramme global (ou complet)** : C'est l'éthogramme couramment rencontré. Il présente un (ou plusieurs) comportement général de l'espèce. Un éthogramme global est axé autour des points suivants :

  l'individu : Dans ce type d'éthogramme, on retrouve les comportements qu'a l'individu sur lui-même sans l'intervention d'une entité extérieure, tels que sa posture, sa locomotion ou son régime alimentaire.

  le social : Il s'agit ici de la description des comportements sociaux, c'est-à-dire ceux qui s'expriment avec les semblables de la même espèce. On y trouve par exemple les conduites nuptiales, l'imitation et la coopération.

  l'inter-spécificité : On s'intéresse ici au comportement vis-à-vis des espèces autres que la sienne. Il s'agit par exemple de la prédation, la symbiose ou le mutualisme.

— **Éthogramme partiel** : Ce type d'éthogramme est utilisé lorsque l'on doit descendre à un niveau de description plus bas qu'un comportement individuel. Il s'agit d'une partie d'un ensemble plus large d'observations et de descriptions comportementales. Un éthogramme global peut couvrir un large éventail de comportements, tandis qu'un éthogramme partiel se concentre sur un sous-ensemble spécifique de comportements d'intérêt. Par exemple, un éthogramme partiel peut être utilisé, pour décrire précisément le comportement de chasse d'une espèce

— **Éthogramme spécial** : Toujours dans le but de donner le plus de précision dans la description du comportement, l'éthogramme spécial va encore plus loin dans la description par rapport à l'éthogramme partiel. Il est utilisé en particulier pour décrire les «primitives» de comportements telles que la conduite de vocalisation (produire un son élémentaire).

Le tableau 1.1 présente un exemple d'éthogramme tiré de (Leach et al. 2011). Il décrit le comportement douloureux chez un lapin, utilisé pour déterminer l'intensité de la douleur de chaque séquence.

Table 1.1 – L'éthogramme du comportement douloureux chez un lapin

Comportement	**Description**
Tiquer	Mouvement rapide de la fourrure sur le dos
Réculer	Le corps se soulève sans raison apparente
Grimacer	Mouvement rapide de l'arrière dans un mouvement de bascule accompagné d'une fermeture des yeux et d'une déglutition.
Trébucher	Perte partielle d'équilibre
Chute	Perte totale d'équilibre lors d'un déplacement
Appuis	Abdomen poussé vers le sol, généralement avant de marcher
Cambrer	Cambrure complète du dos vers le haut
Frissonner	Contraction des muscles obliques du flanc
Trainer	Marcher à un rythme très lent

Comme nous pouvons le remarquer à travers cet exemple et la classification des éthogrammes, l'éthogramme vise à être le plus précis et le plus fidèle possible dans la description d'un comportement animal. Il se doit donc d'être :

— Exhaustif : l'éthogramme doit contenir toutes les catégories de comportements ou unités de conduite de l'espèce étudiée. Ce critère est d'autant plus important lorsque l'éthogramme est global.
— Précis : les comportements doivent être clairs et sans aucune ambiguïté pour permettre à tout observateur de reconnaître les comportements décrits.
— Homogène : l'éthogramme, en tant que catalogue de conduites, ne doit contenir que des comportements du même ordre. Il ne sera donc pas possible d'avoir à la fois dans un même éthogramme une description au niveau global, partiel ou spécial.
— Maniable : il doit être facilement utilisable.

Au fil de l'évolution du temps et des façons de penser, l'observation minutieuse du comportement animal a suscité de nouvelles pistes pour une compréhension plus accrue et plus précise du comportement des animaux. La modélisation et la simulation viennent donc à juste titre pour aller au-delà des limites de l'observation pure. Nous nous intéressons donc à ce domaine pour approfondir notre compréhension.

1.2 Modélisation et simulation du comportement animal

Avant d'aborder à proprement dit la modélisation et la simulation du comportement animal, intéressons-nous tout d'abord aux concepts qui fondent la discipline de la modélisation et de la simulation en informatique.

1.2.1 Concepts généraux de la modélisation et de la simulation

Les origines réelles du concept de la modélisation ne sont pas clairement identifiées puisque dans quasiment chaque étape cruciale du développement de l'humanité, il est possible de retrouver des traces de la modélisation. De façon littérale, la modélisation est le processus de «présentation sous la forme d'un modèle». Ce qu'on pourrait comprendre par-là, c'est que la finalité d'une modélisation c'est de partir d'un système réel pour en obtenir une représentation simplifiée, abstraite ou symbolique appelée modèle (Birta et al. 2019). Selon le contexte, et le cas d'utilisation, il peut s'agir d'un modèle physique pour servir de référence (exemple des modèles en coutures) ou comme dans notre contexte d'étude, un modèle de simulation pour simuler le fonctionnement d'un système au fil du temps. Modéliser et simuler fait donc intervenir ces trois éléments principaux que sont le système réel, le modèle et la simulation.

1.2.1.1 Le système réel

Le système est l'élément étudié durant le processus de la modélisation et de la simulation. Il est donc indispensable de bien comprendre de quoi il est question et de cerner les théories qui le régissent.

Le concept du système apparaît dans plusieurs domaines de la recherche et porte également des définitions propres à chacun d'eux. Dans le langage courant, un système est défini comme «un groupe organisé ou connecté de choses» [10], coordonnées par des règles en fonction d'un

10. https://www.oed.com/view/Entry/196665?redirectedFrom=system

but à atteindre. On peut l'illustrer par le corps humain qui est constitué de plusieurs organes et organismes qui contribuent à un but ultime qui est la survie. Sur un aspect un peu plus technique, une définition du système est proposée dans (Franceschini 2017) : *«Par système, on désigne un phénomène (...) complexe(...) ou plus généralement un système réel dont on souhaite comprendre le fonctionnement»*. Il met ainsi en lumière pour un système quatre points d'attention qui sont :

- *L'interaction* qui existe entre les entités qui forment le système.
- *L'environnement*, dans lequel et à travers lequel le système agit. Nous y reviendrons plus en détail dans la section 1.2.2.4.
- *La finalité*, c'est-à-dire, ce pour quoi le système est destiné. Parfois, la finalité n'est pas connue et peut être découverte par des études.
- *La synergie* qui fait référence au résultat qui émerge des actions élémentaires réalisées par chaque entité du système.

Lorsqu'un système est composé de plusieurs entités, il est dit « complexe » dans le sens où les entités qui le composent interagissent et parfois collaborent.

Un système complexe présente les propriétés suivantes (Ladyman et al. 2013) :

- *Non-linéarité* : Cette propriété signifie que le comportement qui résulte de l'exécution de chaque entité qui compose le système n'est pas superposable. Il ne suffit donc pas d'additionner ou de multiplier les résultats entre eux pour obtenir le résultat commun. Autrement dit, les changements dans les entrées ne produisent pas nécessairement des changements proportionnels dans les sorties.
- *Rétroaction (Feedback)* : Elle est étroitement liée à l'interaction que le système a avec ses voisins. Elle est mise en évidence lorsque l'interaction avec ses voisins à un instant t influe sur l'interaction qu'il a avec eux à un autre instant $t_1 > t$.
- *Ordre* : Caractérise le mécanisme de déclenchement du système. On parle souvent de pilotage du système, qui peut être effectué par le temps ou par des événements.
- *Robustesse et absence de contrôle central* : Le système complexe étant constitué de plusieurs entités, son fonctionnement est de fait distribué et non produit de manière centralisée. Il est donc stable en cas de perturbation.
- *Émergence* : C'est une notion forte, peut-être la plus forte des systèmes complexes. C'est la propriété par laquelle le fonctionnement de chaque entité du système complexe fait apparaître un fonctionnement qui pourrait s'apparenter à un comportement unique.
- *Organisation hiérarchique* : Il existe des niveaux d'organisation formant une hiérarchie de systèmes et de sous-systèmes. Le fonctionnement de chaque niveau inférieur est le fondement de l'émergence.

La figure 1.3 montre la représentation qu'on pourrait donner à un système. Dans cette représentation, le système est caractérisé par ses entrées, son processus et ses sorties.

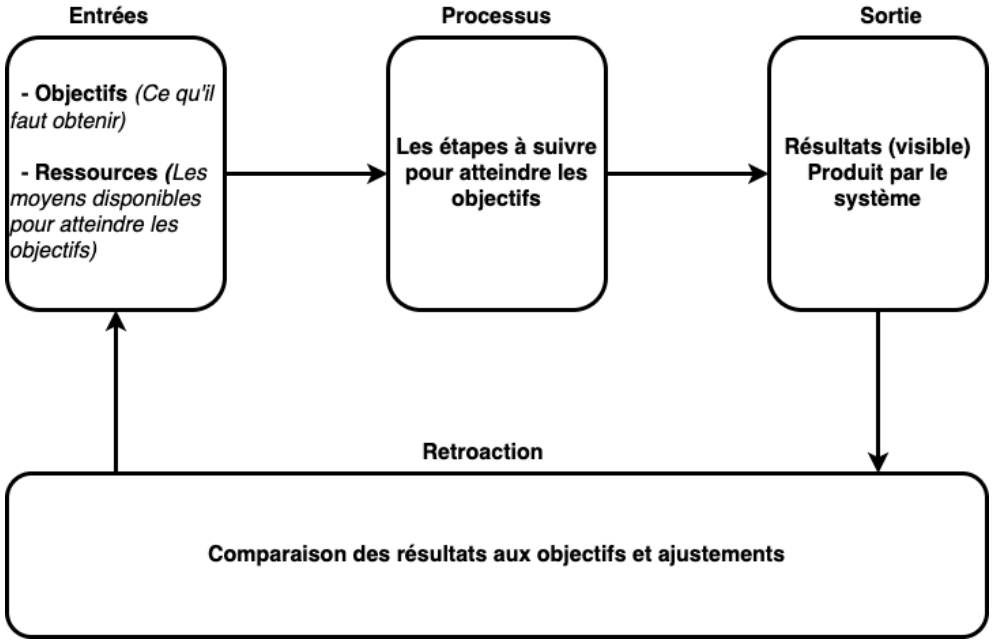


FIGURE 1.3 – Représentation possible d'un système

Les entrées désignent à la fois les objectifs et les ressources à disposition pour les atteindre. En reprenant notre exemple du corps humain en tant que système, une ressource pourrait être l'air, l'eau ou les aliments. *Le processus* désigne les étapes ou la démarche à suivre pour atteindre les objectifs définis en entrée. Toujours pour le corps humain, cela comprend se nourrir, se reposer, se soigner, etc. Les *sorties*, quant à elles, désignent les résultats qui proviennent du système en réponse aux entrées et au processus. Les résultats en sortie peuvent parfois être récupérés en tant qu'entrées pour être comparés aux objectifs afin de procéder à des ajustements, que ce soit au niveau des entrées ou du processus.

Les systèmes sont généralement classés en fonction des propriétés qu'ils présentent. Selon les descriptions des systèmes fournies dans (SHAWKI et al. 2015) et (CASSANDRAS et al. 2008), nous distinguons deux grandes catégories de systèmes : les systèmes statiques et les systèmes dynamiques. La classe des systèmes statiques regroupe les systèmes dont la sortie est indépendante des valeurs passées de l'entrée, contrairement à la classe des systèmes dynamiques où la sortie dépend généralement des valeurs passées en l'entrée. Il est également possible de former des sous-classes au sein de ces deux catégories principales en utilisant des propriétés telles que les règles d'évolution du temps et du pilotage, ou la nature des sorties, comme proposé par (POGGI 2014).

À la lumière de ces différents éléments du système, nous pouvons à présent nous tourner vers le modèle, qui logiquement découle du système lui-même.

1.2.1.2 Le modèle

Un modèle est la représentation simplifiée d'un système, qu'il soit statique ou dynamique. Il peut prendre différentes formes, qu'elles soient numériques ou symboliques. Dans le domaine de la mode, par exemple, le modèle est la présentation d'une création et sert principalement à donner un aperçu de ce que sera la création une fois sur le marché (Rabbinge et al. 1989). Ce concept de modèle est également largement utilisé dans les domaines de la science et de l'ingénierie, où il représente partiellement un système d'intérêt afin de comprendre son comportement réel ou d'évaluer différentes stratégies de fonctionnement (Ingalls 2001 ; Shannon 1998). De manière plus formelle, (Coquillard et al. 1997) propose la définition suivante pour un modèle : *«Pour un observateur B, un objet A∗ est un modèle d'un objet A dans la mesure où B peut utiliser A∗ pour répondre aux questions qu'il se pose sur A»*. Cette définition met en évidence deux éléments importants : l'observateur et les questions auxquelles il cherche des réponses.

— Modéliser un système passe par une phase d'observation où le modélisateur [11] observe le système à modéliser afin d'identifier et de recueillir des caractéristiques jugées suffisantes pour mettre en évidence le comportement observé. Ainsi, à aucun moment, le modèle ne pourrait reproduire le système réel au complet puisqu'il ne représente que certaines propriétés.

— Quant aux questions auxquelles il faut répondre, cela sous-entend qu'on modélise un système lorsque l'on a des questions à se poser sur celui-ci. Pour que le modèle soit alors capable de répondre aux questions, il se doit de reproduire le plus fidèlement possible le système réel et être capable de se trouver dans les conditions requises pour subir (réellement ou artificiellement) les mêmes actions que le système réel. En retour, il doit être en mesure de répondre aussi (réellement ou artificiellement) comme le système réel et produire une réponse. Il faut donc voir le modèle comme une entité pouvant être dans un ou plusieurs états (les conditions), qui reçoit des informations hétérogènes spécifiques (les entrées), en fait un traitement à partir de ses règles (fonction interne) pour produire une ou plusieurs réponses (les sorties), comme représenté par la figure 1.4.

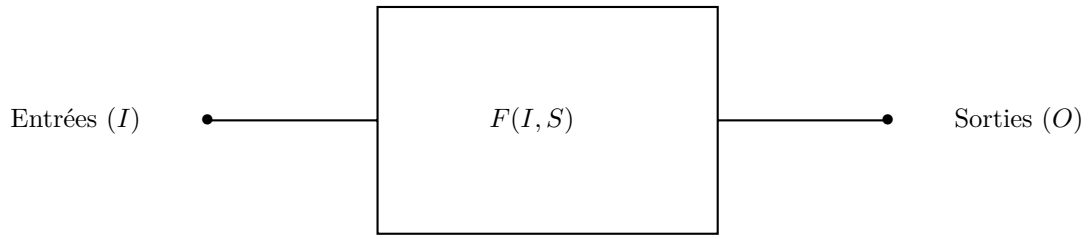


Figure 1.4 – Schéma synoptique d'un modèle : I est l'ensemble des entrées et O l'ensemble des sorties. F est la fonction interne qui produit la sortie à partir de I et de S, l'ensemble des états du modèle

La modélisation d'un système se fait en cinq grandes étapes (Frantz 1995) présentées à la figure 1.5.

11. Celui qui modélise un système

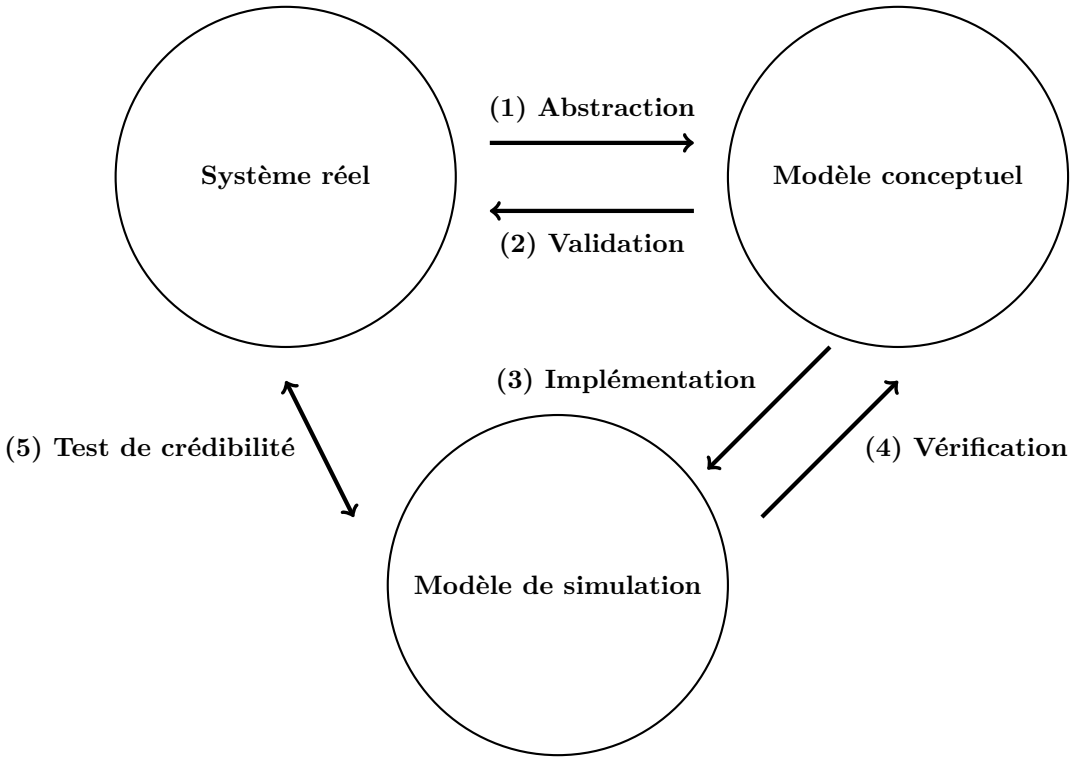


FIGURE 1.5 – Les cinq étapes du développement d'un modèle.

La première étape est l'abstraction : elle représente la façon dont le modélisateur voit le système réel, c'est-à-dire « comment il le pense ». C'est une phase cruciale pour le développement du modèle, puisqu'elle consiste à identifier les facteurs qui influencent le comportement étudié du système. L'identification de ces éléments dépend des questions auxquelles le modèle devra répondre. Nous appellerons le modèle obtenu à cette étape le « modèle conceptuel ». Couramment, les modèles conceptuels sont représentés par des schémas synoptiques ou par le formalisme UML (GENERO et al. 2011). Il n'est donc pas exclu de voir un modèle conceptuel présenté autrement.

La seconde étape est la validation du modèle conceptuel. Le modèle conceptuel est analysé pour déterminer s'il représente de manière suffisante le système réel. Généralement, à cette phase, on fait appel à un regard extérieur, celui de l'expert [12].

La troisième étape est l'implémentation du modèle conceptuel. Ici, on donne une forme au modèle. En fonction de l'analyse réalisée lors de la phase d'abstraction, un formalisme est choisi pour rendre le modèle conceptuel exécutable. Le modèle exécutable est appelé « modèle de simulation ».

S'ensuit la quatrième phase, qui est celle de la vérification. Le modèle est analysé pour vérifier si la forme exécutable est une représentation exacte du modèle conceptuel.

Enfin vient la cinquième et dernière phase appelée « test de crédibilité », qui, comme son nom l'indique, détermine si l'on peut croire aux réponses fournies par le modèle. Ici, le modèle est testé [13] pour observer les réponses apportées. Elles sont ensuite comparées aux

12. Celui qui connaît suffisamment le système du volet sur lequel on l'étudie et pour lequel le modèle est élaboré
13. c'est-à-dire simulé

réponses apportées par le système réel, mis dans les mêmes conditions. L'écart entre les réponses définit la crédibilité du modèle.

Principalement à la cinquième phase, la comparaison n'est possible que lorsque le modèle de simulation est testé dans une configuration bien définie lors de la simulation.

1.2.1.3 La simulation

La simulation est le processus par lequel on exécute un modèle. Afin de mieux l'expliquer, faisons une analogie avec des situations de la vie quotidienne. Si l'on considère le modèle comme une partition de musique, la simulation serait le processus qui permet de la jouer. En effet, lors de la simulation, le modèle est configuré et placé dans un contexte spécifique pour réagir et produire en sortie des résultats que devrait produire le système réel s'il était placé dans les mêmes conditions. Cette définition s'explique encore plus aisément lorsque l'on considère la définition littérale du mot « simulation », qui signifie « faire comme » : le modèle va faire comme le système réel. La figure 1.6 montre la simulation d'une nuée d'oiseaux en vol (BOIDS) [14], extraite de (Lebar Bajec et al. 2007). Les triangles noirs représentent les oiseaux, le sommet indiquant le cap.

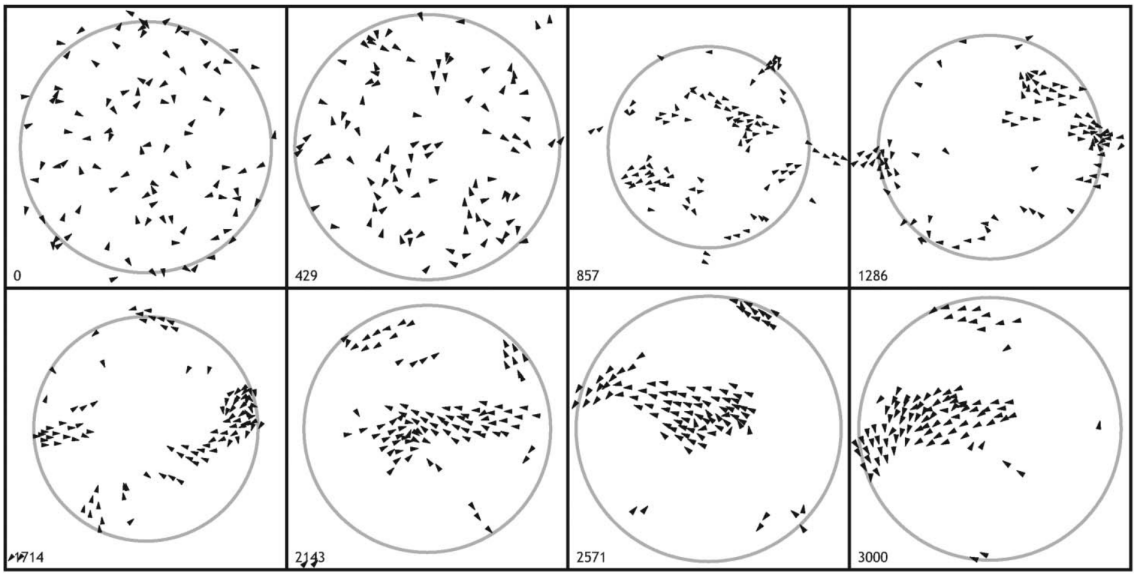


Figure 1.6 – Simulation d'une nuée d'oiseaux

Le fait de pouvoir réaliser des simulations qui permettent de « faire comme » un système réel représente un atout majeur, car cela offre la possibilité de réaliser deux choses principales sur lesquelles il est important de se concentrer.

1. Lorsque l'on dispose d'un modèle, on peut le simuler et contourner les contraintes spatiales et temporelles. En effet, lorsque l'on effectue une simulation, les échelles sont réduites, y compris celles du temps et de l'espace. C'est d'ailleurs l'un des avantages majeurs de la simulation. Grâce à cette caractéristique, le concepteur du modèle peut

14. Programme informatique de vie artificielle, simulant le comportement d'un essaim d'oiseaux en vol (Reynolds 1987)

rapidement se projeter dans le temps et l'espace afin d'identifier des points critiques qui pourraient survenir dans le système réel. C'est le cas, avec les modèles qui permettent aujourd'hui de prédire la météo à un endroit précis et à un moment donné en se projetant dans le temps et l'espace.

2. Le modèle étant la représentation du système, on peut lui faire subir différents scénarios sans craindre une détérioration du système réel. Cet atout est notamment utilisé dans les cas où l'on souhaite réaliser des tests. Par exemple, il est possible de simuler et étudier les stratégies d'évacuation d'un bâtiment en cas d'incendie (Kasereka et al. 2018), sans bien sûr disposer réellement de ce bâtiment et encore moins y mettre le feu. La simulation offre ainsi l'avantage d'envisager plusieurs scénarios et de les simuler pour évaluer à la fois leur efficacité et identifier les points critiques à améliorer.

Grâce à ces atouts, la modélisation et la simulation sont largement utilisées dans de nombreux domaines tels que la médecine et la santé (Dong et al. 2012 ; Manolis et al. 2011), les sciences sociales (**varenne_les_2010**), la biologie (Laubenbacher et al. 2013), l'écologie (Taylor et al. 2007), la sécurité civile (Filippi et al. 2011) et bien d'autres. Leur contribution se situe dans l'anticipation des situations de crise ou dans la proposition de solutions adaptées pour la gestion de ces situations.

Pour réaliser les simulations, il faut disposer d'un simulateur capable d'interpréter et de produire les résultats appropriés en fonction des états et de la fonction interne du modèle. Selon l'approche de modélisation choisie, le simulateur peut réaliser des simulations pilotées par le temps ou par des évènements (Özgün et al. 2009).

Dans une simulation pilotée par le temps, les changements dans le système interviennent en fonction de l'évolution du temps. Ce temps est soit continu, soit discret. On parle de temps continu lorsque la variable «temps» change continuellement et sans interruption. Cela signifie que pour tout $t \in \mathbb{R}$, il est possible de trouver une valeur de temps intermédiaire entre deux instants jusqu'à des quantités infinitésimales. On parle de temps discret, lorsque le temps est divisé en des instants distincts. Pour chacun de ces instants spécifiques, une observation des changements dans le système est faite.

Quant à la simulation pilotée par les évènements, elle se focalise sur les évènements. Les changements dans le système sont effectués lorsque des évènements interviennent. Ainsi, il n'est pas nécessaire d'avoir une horloge qui déclenche des changements à des pas de temps, mais plutôt de rester «attentif» aux évènements. La simulation pilotée par les évènements est particulièrement rapide à mettre en œuvre tout en offrant une facilité à expliquer les résultats obtenus (Zeigler et al. 2000 ; Michel 2004). De plus, elle a généralement un temps d'exécution inférieur à celui de la simulation pilotée par le temps.

La figure 1.7 résume les relations qui existent entre le système, le modèle et le simulateur.

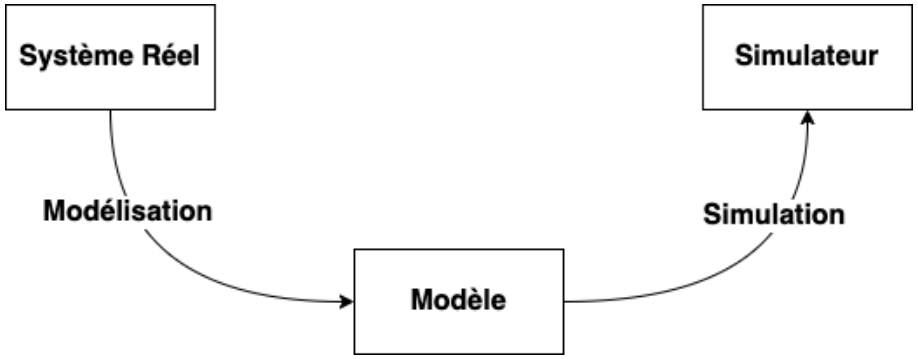


FIGURE 1.7 – Théorie de la modélisation et simulation

Comme nous pouvons le remarquer, la modélisation n'implique pas nécessairement l'utilisation d'un modèle mathématique ou informatique. Il existe donc de facto plusieurs approches pour modéliser un système, à condition de disposer d'un simulateur capable d'exécuter le modèle. Ces approches se présentent sous différentes formes, allant des plus simples aux plus complexes, ou des plus génériques aux plus spécialisées. Le tableau 1.2 propose un aperçu sur des exemples d'approches de modélisation et de simulation des systèmes complexes rencontrées dans la littérature avec chacune des qualités et des faiblesses qui leur sont propres.

TABLE 1.2 – Tableau comparatif d'exemples d'approches de modélisation et simulation

Approche	Description	Forces	Limites
Équation différentielle (CHAACHOUA et al. 2006)	Décrit les variations continues d'une quantité au fil du temps.	Précision pour les (systèmes continus).	Mise en œuvre difficile pour les systèmes complexes non linéaires.
Automates à états finis (HOPCROFT et al. 2001)	Modèle avec un ensemble d'états et conditions de transition entre eux.	Simple et intuitive pour décrire des systèmes séquentiels.	Difficile à mettre en œuvre quand le nombre d'états et/ou de transitions est grand.
Chaînes de Markov (HAN et al. 2021 ; HC YANG et al. 2005)	Processus semblable aux automates à états finis, mais où les transitions sont définies par des probabilités.	Modèle stochastique simple pour des prédictions à court terme.	Nécessite une connaissance précise des probabilités de transition.
Arbres décisionnels (QUINLAN 1990 ; KOTSIANTIS 2013)	Structure arborescente basée sur des règles conditionnelles .	Facile à interpréter et bien adapté pour des décisions binaires.	Difficile à mettre en œuvre pour des systèmes dans lesquels il existe des relations entre les variables.
Formalisme DEVS (ZEIGLER et al. 2000 ; MICHEL 2004)	Modèle formel pour la modélisation et la simulation de systèmes (discrets et continus) dirigés par les évènements.	Produit (rapidement) à la simulation des résultats assez précis.	Peut nécessiter des connaissances avancées en modélisation formelle.
Agents (AKASHAH et al. 2020 ; AXELROD 1997)	Modélisation basée sur des entités autonomes interagissant dans un environnement.	Modélise des interactions complexes entre entités.	Difficile à paramétrer correctement.
Régression (SARANG 2023)	Techniques statistiques pour modéliser la relation entre les variables, en trouvant la meilleure approximation.	Fonctionne bien lorsque la relation est linéaire.	Perd en efficacité lorsque les relations entre variables sont de plus en plus complexes
Réseaux de Neurones artificiels (NICHOLS et al. 2018 ; JANIESCH et al. 2021)	Modèle basé sur le fonctionnement du cerveau, utilisé pour apprendre des modèles à partir de données.	Capable de modéliser des relations complexes.	Requiert de grandes quantités de données et de puissance de calcul. Résultats difficiles à expliquer

Dans cet univers très large d'approches de modélisation et particulièrement dans le contexte de notre recherche visant à modéliser et simuler le comportement des animaux, une approche émerge comme particulièrement prometteuse et mérite une exploration approfondie. Il s'agit de l'approche basée sur les agents. Contrairement aux approches conventionnelles qui décrivent le système comme une collection de variables et de règles, l'approche basée sur les agents envisage les entités individuellement en tant qu'acteurs autonomes capables de collecter des informations, de communiquer et d'agir pour s'adapter à leur environnement. Ces caractéristiques semblent remarquablement refléter la manière dont les animaux interagissent avec leur environnement, et c'est pourquoi nous souhaitons l'explorer.

1.2.2 Le paradigme agent

La modélisation basée sur les agents est l'un des paradigmes de modélisation qui connaît un succès remarquable en raison de la pertinence du concept «agent» dans des domaines divers et variés (Matthews et al. 2007 ; Barbati et al. 2012 ; Mehdizadeh et al. 2022).

1.2.2.1 L'agent

Étant donné que nous parlons du paradigme agent, il est crucial d'abord de bien comprendre ce qu'est un agent afin de bien comprendre le paradigme, car ce qui est souvent reproché aux approches basées sur les agents, c'est leur manque de crédibilité vis-à-vis des autres domaines de l'informatique (Michel 2004). Partons de la définition littérale de l'agent. En effet, littéralement, un agent peut désigner tout et n'importe quoi à la fois. On parle par exemple d'agent secret, d'agent pathogène, d'agent de sécurité, etc. Ainsi, un agent peut tout à fait prendre le sens d'«objet» d'un point de vue littéral.

En raison de la forte hétérogénéité des définitions, trouver une définition sur laquelle toutes les disciplines pourraient s'entendre apparaît comme une énigme difficile à résoudre. Lors d'un atelier [15], Carl Hewitt va même qualifier d'embarrassante la question «Qu'est-ce qu'un agent» ? (Wooldridge et al. 1995). Ainsi, plutôt que de proposer une définition formelle et unanime de ce qu'est un agent, il faut plutôt voir l'agent comme une manière de penser un système informatique. Dans (Wooldridge et al. 1995), deux notions sont proposées pour expliquer ce qu'est un agent. L'une est dite «faible» et l'autre est dite «forte».

La notion faible désigne par le terme agent, tout système ayant les caractéristiques suivantes :

- *une autonomie* : lui permettant d'agir sans aucune intervention extérieure directe (l'humain ou un autre agent)
- *une sociabilité* : lui permettant une interaction avec les autres agents de son environnement, via des moyens de communication bien spécifiques. Cette interaction peut également s'étendre jusqu'à l'humain via une IHM.
- *une réactivité* : lui permettant de percevoir son environnement et d'en recevoir des informations ou des ordres. L'agent devra les prendre en compte et y répondre au moment opportun.

15. Thirteenth International Workshop on Distributed AI.

— *une proactivité* : lui permettant d'agir aux signaux reçus de son environnement. Il peut aussi avoir des réactions qui résultent d'une stratégie pour atteindre un objectif défini.

À partir de cette description, nous pouvons considérer un agent comme : «toute entité réelle ou virtuelle, ayant un objectif (ou non), en fonction duquel il perçoit son environnement, analyse les informations collectées et communique avec d'autres agents pour agir en conséquence, tout ceci en toute autonomie». Cette définition que nous illustrons par la figue 1.8 se rapproche assez bien de celles proposées par (Genesereth et al. 1994 ; Russell et al. 2002) et plus récemment par (Franceschini 2017).

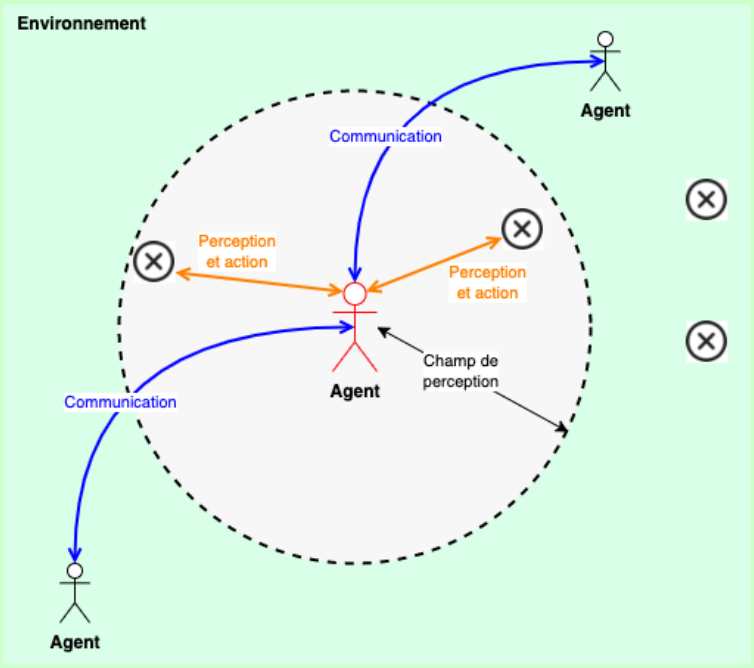


Figure 1.8 – Représentation d'un agent et ses interactions avec son environnement

Bien qu'assez explicite, la notion faible reste quand même assez controversée. Pour beaucoup de chercheurs, un agent ne devrait pas avoir une définition aussi large. L'utilisation de ce terme devrait être très cadrée, spécifiquement pour des systèmes informatiques qui, en plus de posséder les propriétés présentées dans la notion faible, sont mis en œuvre à l'aide de concepts qui s'appliquent aux humains. Ils précisent cependant que cette approche ne devrait pas se confondre avec de l'anthropomorphisme [16].

Maintenant que nous avons une compréhension suffisante de ce qu'est l'agent, intéressons-nous à présent aux différents types d'agents qui jouent des rôles essentiels dans ce contexte.

1.2.2.2 Les types d'agents

Les agents sont le plus souvent classés suivant leur fonctionnement interne. La limite entre le type d'agent et son architecture interne est très étroite. Notre objectif étant de rester le

16. Concept (philosophique) qui donne une apparence humaine à des animaux ou des objets. Ce concept est très présent dans les séries d'animation où l'on donne de la voix, une posture et un style de vie humaine à des animaux ou des objets.

plus explicite possible, nous faisons bien cette différence. Nous présentons donc ici les types d'agents, et dans la section 1.2.2.3, les principales architectures.

Nous distinguons deux principaux types d'agents : les agents réactifs et les agents cognitifs.

1.2.2.2.1 Agents réactifs

Le fonctionnement des agents réactifs est basé sur des relations entre les perceptions et les actions à partir d'une cartographie directe. Ainsi, l'agent réagit mécaniquement à des stimuli suggérés par les perceptions. Les agents réactifs n'ont formellement pas de buts précis, mais cela ne veut pas dire qu'ils sont dépourvus d'intelligence. En effet, des comportements individuels peuvent émerger pour former un comportement collectif d'une intelligence remarquable. Tel est principalement le cas pour les colonies d'animaux ou les essaims. On parle d'intelligence collective(Leimeister 2010). Un des exemples les plus illustratifs est celui des essaims de fourmis. Une fourmi à elle seule n'est a priori pas pourvue d'une intelligence [17], mais lorsqu'elle se retrouve en colonie, les actions de chaque individu font émerger une organisation intelligente qui œuvre pour la survie et la pérennité de l'espèce.

1.2.2.2.2 Agents cognitifs

Par opposition à l'agent réactif, l'agent cognitif inclut un processus de prise de décision qualifié de «intelligent» ou de «rationnel» (Michel 2004). Ce processus lui permet de réaliser seul des opérations complexes telles que la mémorisation, l'analyse et le raisonnement. La propriété principale qui caractérise un agent cognitif est la mémoire. Grâce à celle-ci, l'agent :

— Dispose d'une base de connaissances constituée de croyances et d'objectifs qui lui permettent de raisonner et de prendre des décisions sur les actions à réaliser. Pour un agent placé dans un environnement où il y a des proies et des prédateurs, l'intention serait de survivre. En termes de croyance, nous pouvons en identifier deux. La première est de savoir que les prédateurs sont dangereux et qu'il faut les éviter. La seconde est de savoir que les proies sont utiles pour la survie et qu'il ne faut pas s'en méfier.

— Dispose d'une représentation de l'environnement. L'agent peut garder en mémoire une représentation de son environnement qui pourrait intervenir dans son processus de prise de décision. En reprenant l'exemple précédent, l'agent peut garder en mémoire les zones où se trouvent les proies et les prédateurs. Cette information lui permet d'agir de manière optimale pour atteindre son objectif qui est la survie.

La mémoire dont est doté l'agent cognitif lui permet de réaliser des tâches (non stéréotypées) qui résultent d'un raisonnement individuel.

Néanmoins, l'agent cognitif n'est pas systématiquement celui à utiliser, car il présente également des contraintes. Avec l'intégration de la mémoire, une contrainte spatiale est automatiquement ajoutée. De plus, étant donné qu'il y a désormais un processus de prise de décision, le temps de réaction de l'agent est rallongé.

Le fonctionnement global d'un agent est clairement déterminé par son type, ce qui reflète la manière dont il opère généralement. Ce mode opératoire découle des choix successifs effectués par l'agent au cours de son exécution. Les modalités de ces prises de décision sont elles

17. En considérant l'intelligence comme la capacité de prendre des décisions individuelles

gouvernées par l'architecture interne de cet agent. Dans la littérature scientifique examinée, diverses architectures sont mentionnées, et nous en exposons les principales.

1.2.2.3 Principales architectures agent

L'architecture d'un agent représente son organisation interne. Il définit comment est construit le processus de prise de décision appelé *la délibération* (Pnueli 1986). Nous retrouvons dans la littérature plusieurs architectures d'implémentation des agents (Chong et al. 2007). Pour la plupart, elles sont inspirées de l'architecture de subsomption et de l'architecture BDI.

1.2.2.3.1 Architecture de subsomption (Brooks 1983)

L'architecture de subsomption est liée aux agents réactifs. La délibération est basée sur une relation directe entre les perceptions et les actions. Un agent a qui implémente l'architecture de subsomption est défini par :

$$a =< \Pi, \Gamma, \upsilon >$$

Π ensemble perceptions de l'agent (ce qu'il peut savoir) ;
Γ ensemble des modules comportementaux de l'agent (ce qu'il peut faire) ;
$\upsilon : \Pi \rightarrow \Gamma$ une application telle que $\forall\ \gamma \in \Gamma\ ;\ \exists\ \pi \in \Pi\ /\ \upsilon(\pi) = \gamma\ ;$

Le principe ici d'associer de façon classique, les perceptions à des blocs pondérés par une priorité. Les blocs accèdent tous en parallèle aux données et sont capables de contrôler le système par eux-mêmes. On obtient ainsi une hiérarchisation entre les actions avec un déclenchement à la priorité la plus grande (voir figure 1.9) L'algorithme 1 décrit ce fonctionnement.

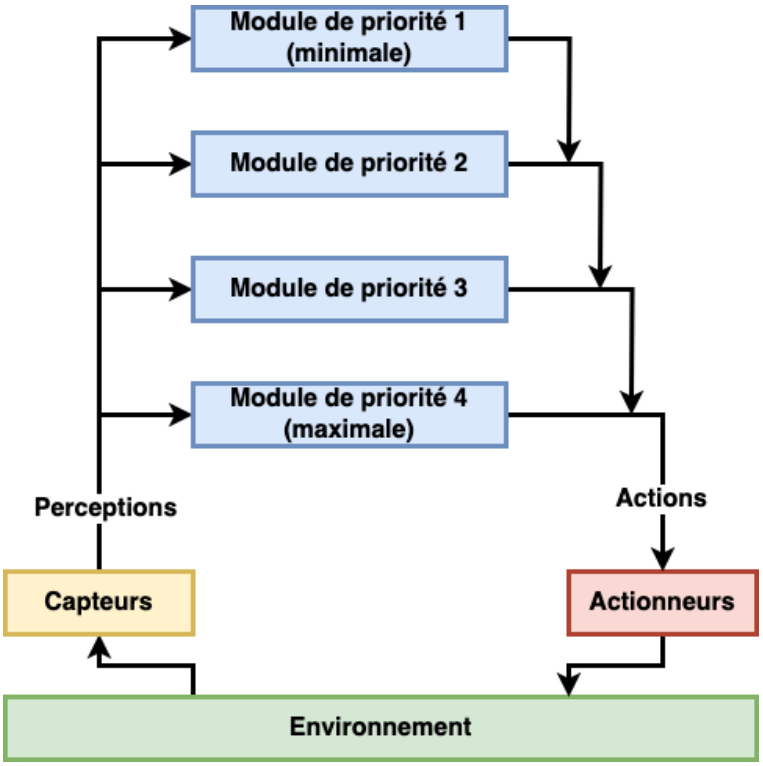


FIGURE 1.9 – Architecture de subsomption

Algorithme 1 : Algorithme d'exécution de l'architecture de subsomption

Données : Π (Perceptions), Γ (Modules de comportement)
tant que *agent en vie* **faire**
  list_modules ← [ ] ▷ Debut du cycle;
  inputs ← perceptions() ▷ L'agent perçoit son environnement ;
  pour *chaque* $p \in inputs$ **faire**
    module ← $v(p)$ ▷ En se basant sur Γ et Π;
    ajouter(module, list_modules) ;
  fin
  moduele_a_executer = max(list_modules) ;
  executer(moduele_a_executer) ▷ Fin du cycle ;
fin

Cette architecture est souvent utilisée dans le domaine de la robotique pour les robots dont les actions ne nécessitent pas de longs processus de décision, comme se déplacer. Elle a été d'ailleurs mise en œuvre dans (STEELS 1990) pour l'étude d'une mission d'exploration d'une planète lointaine, dans l'objectif de collecter des échantillons de roches et de minéraux. L'emplacement des échantillons n'est pas connu à l'avance, mais on sait qu'ils ont tendance à être regroupés. La représentation est donnée à travers le diagramme d'activité de la figure 1.10.

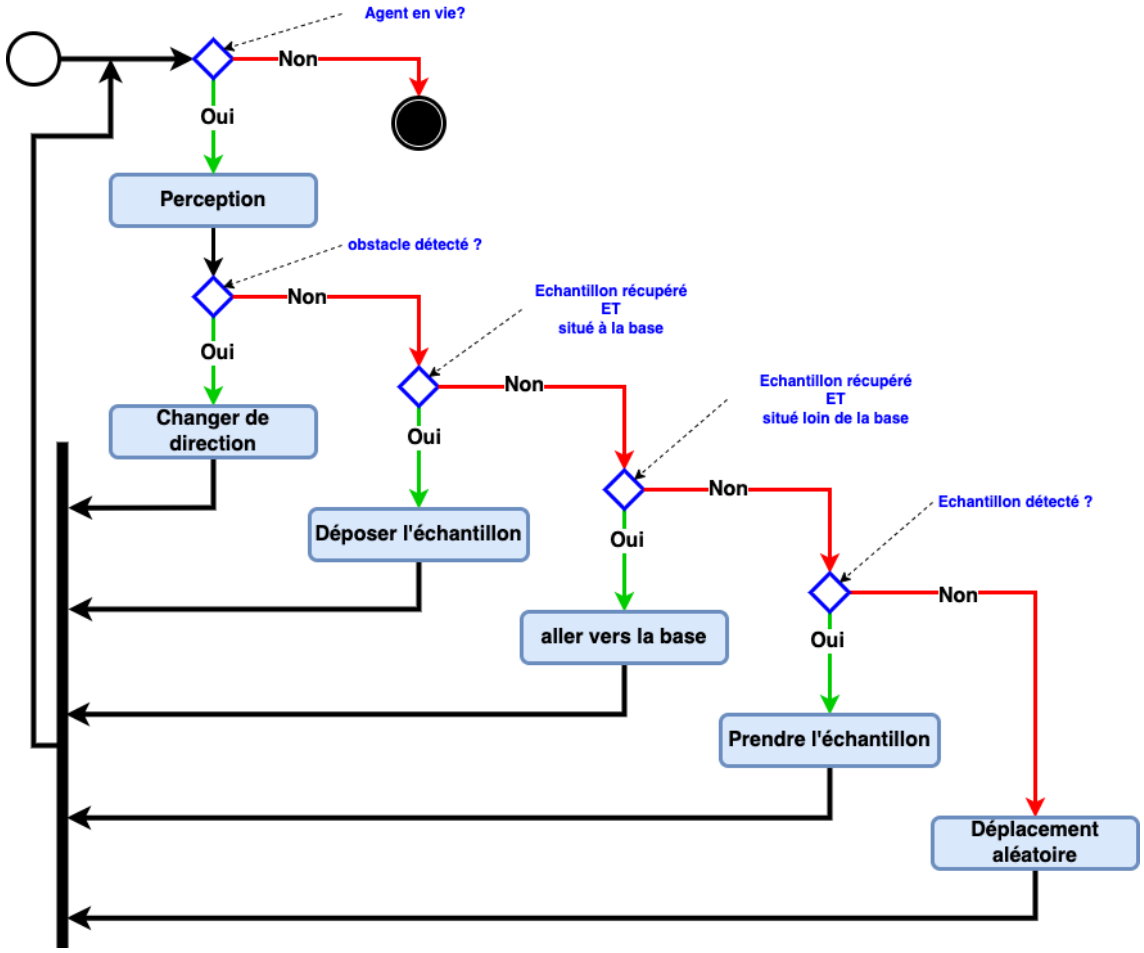


FIGURE 1.10 – Diagramme d'activité du robot explorateur de planète de Steel

L'extrême modularité de l'architecture de subsomption permet de faciliter sa maintenance, car il est très facile d'ajouter, de modifier ou de supprimer des actions et les associations entre perceptions et actions. Cette simplicité implique malheureusement d'autres limitations, car bien qu'elle soit efficace pour décrire des comportements très simples, elle est difficilement applicable à des actions plus complexes. De plus, toutes les conditions de déclenchement doivent être planifiées de sorte que les règles soient mutuellement exclusives et qu'il n'y ait pas de conflits de règles.

1.2.2.3.2 Architecture Beliefs-Desire-Intentions (BDI) (Rao et al. 1992)

L'architecture *Beliefs-Desire-Intentions*[18] des agents tire ses origines des travaux réalisés sur le raisonnement pratique humain (BRATMAN et al. 1988). Elle est basée sur un système de raisonnement pratique qui permet à l'agent de planifier des actions à réaliser dans un environnement dynamique (CHONG et al. 2007). Elle repose sur trois éléments logiques appelés *états mentaux* (ANTHONY et al. 2014) :

18. Croyances-Désirs-Intentions en Français

— *Les croyances* représentent des informations que l'agent possède sur tout ce qui se trouve autour de lui. Elles peuvent tout à fait être incomplètes et incorrectes (Costantini et al. 2006). Cet ensemble de connaissances est mis à jour en temps réel à la fois par les perceptions et les actions de l'agent (Guerra-Hernández et al. 2005). Remarquons qu'ici nous parlons des connaissances que l'agent a de ce qui se trouve autour de lui, plutôt que de parler de son environnement, car comme nous le verrons dans la section 1.2.2.4, l'environnement peut être assimilé au contexte physique dans lequel se trouve l'agent.

— *Les désirs* constituent les objectifs à atteindre par l'agent, c'est-à-dire la finalité de son fonctionnement ou encore les tâches à accomplir. On parle ici d'objectifs (aims en anglais) lorsque les désirs peuvent ne pas être exclusifs. Dans le cas contraire, on parle plutôt de *buts* (goals en anglais).

— *Les intentions* sont des schémas d'agissement pour lesquels l'agent s'engage à réaliser. Généralement, il s'agit d'un plan, une séquence d'actions dans un certain ordre que l'agent exécute pour tenter d'atteindre un ou plusieurs de ses objectifs formulés par ses désirs. À priori, elles ne sont pas prédéfinies et sont constituées pendant la simulation. Dans certaines variantes de l'architecture BDI, lorsque toutes les intentions possibles pour satisfaire un désir ont été mises en œuvre sans succès, le désir peut être déclaré *irréalisable* et retiré de la liste (Georgeff et al. 1989).

Le fonctionnement d'un agent a qui implémente une architecture BDI est basé sur le traitement d'une file d'attente d'évènements mis à jour continuellement par les perceptions. Initialement, l'agent a est défini de la façon suivante :

$$a =< \Omega, \Delta, I, P, \Phi >$$

Ω : ensemble (vide ou non) des croyances ;
Δ : ensemble non vide des désirs ;
I : ensemble vide des intentions ;
P : file d'attente des évènements à traiter. Elle est constituée des perceptions ;
Φ : ensemble non vide d'actions élémentaires que peut exécuter l'agent.

Comme illustré par la figure 1.11, lorsqu'une perception parvient à l'agent, elle est ajoutée dans P comme un nouvel évènement à traiter. L'exécution démarre par le traitement de cette file d'attente. Chaque perception $p \in P$ est intégrée dans Ω dans la mesure où elle apporte une nouvelle information qui peut soit se traduire par une nouvelle croyance, soit modifier une croyance existante. À partir de cet instant, ces nouvelles croyances disponibles peuvent désormais être mises à contribution pour l'accomplissement des désirs de l'agent.

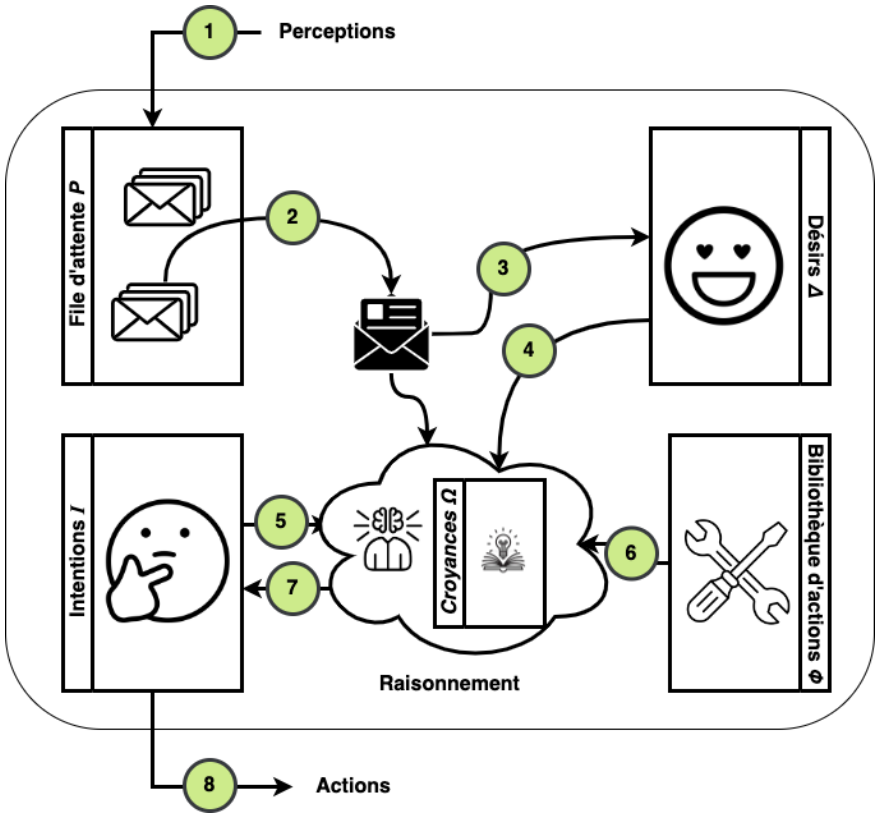


FIGURE 1.11 – Architecture BDI

(1)Ajout de perception dans la file d'attente. (2) Récupération de chaque perception. (3) Mise à jour des désirs. (4)(5)(6) Utilisation des désirs, des intentions et de la bibliothèque d'actions dans le processus de délibération. (7) Mise à jour des intentions. (8) Exécution des actions.

Pour ce faire, les désirs de l'ensemble Δ concernés par les croyances dans Ω sont identifiés et récupérés. Vient ensuite la phase de formulation des intentions qui, comme susmentionné, représentent la stratégie à mettre en œuvre pour tenter de satisfaire les désirs. Il s'agit là de la phase de délibération à proprement dite. En effet, compte tenu de Ω et de Δ, si I est vide, des intentions sont formulées à partir des actions élémentaires de Φ. Sinon, les intentions déjà existantes sont mises à jour (ajout/suppression/permutation d'actions). Le processus est terminé par l'exécution des actions qui résultent des intentions. Tout ce fonctionnement est résumé par l'algorithme 2.

Il faut noter qu'initialement, l'architecture BDI n'intègre pas de mécanismes d'apprentissage, mais cette piste est de plus en plus explorée depuis quelques années (PHUNG et al. 2005 ; GUERRA-HERNÁNDEZ et al. 2005).

Algorithme 2 : Algorithme d'exécution d'une architecture BDI

```
Données : Ω (Croyances), Δ (Désirs), I (Intentions), P (Perceptions), Φ (Actions élémentaires)
tant que agent en vie faire
    ▷ Début du cycle;
    inputs ← perceptions()        ▷ L'agent perçoit son environnement ;
    pour chaque p ∈ inputs faire
        ajouter p dans P;
        Ω ← Intégrer(p, Ω) ;
    fin
    Δ ← mise_a_jour(Δ, Ω) ;
    I ← mise_a_jour(I, Δ) ;
    i = choisir_intention(I, Δ, Ω) ;
    tant que (i non vide) faire
        π ← premier_élément_de(i) ;
        exécuter(π) ;
        inputs ← perceptions;
        pour chaque p ∈ inputs faire
            ajouter p dans P;
            Ω ← Intégrer(p, Ω) ;
        fin
        Δ ← mise_a_jour(Δ, Ω) ;
        I ← mise_a_jour(I, Δ) ;
        i = choisir_intention(I, Δ, Ω) ;
    fin
    ▷ Fin du cycle
fin
```

Tout au long du développement qui a été fait jusqu'à ce stade, nous avons fréquemment parlé de l'environnement. Il devient de plus en plus évident que l'environnement revêt une importance fondamentale dans le domaine de la modélisation et de la simulation et particulièrement pour les agents.

En effet, il occupe une place centrale en influençant directement les éléments en interaction, en fournissant le cadre dans lequel les processus se déroulent, et en exerçant une influence significative sur les résultats et les comportements observés au sein du système modélisé ou simulé. Par conséquent, reconnaître le rôle essentiel de l'environnement contribue à une meilleure compréhension et représentation des phénomènes étudiés, ainsi qu'à la création de modèles et de simulations plus fidèles à la réalité.

1.2.2.4 L'environnement

Lorsqu'on aborde une approche de modélisation basée sur les agents, il est assez rare de voir l'environnement dissocié de cette perspective (Weyns et al. 2005). Il devient particulièrement manifeste que donner une place prépondérante à l'environnement est une nécessité, étant donné que les perceptions et les actions qu'un agent doit effectuer sont façonnées par le contexte environnant dans lequel il évolue. C'est ce qui se passe effectivement dans le fonctionnement des colonies de fourmis. Les individus de la colonie déposent des phéromones dans l'environnement où ils se situent, en fonction de l'accessibilité ou non à des endroits

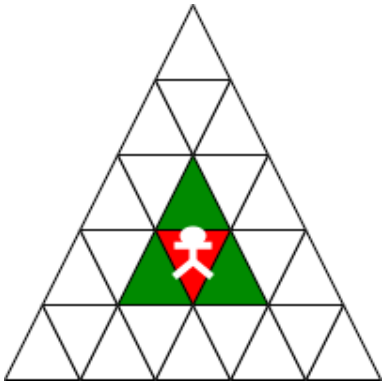
(ex : flaque d'eau, mur). Les autres individus, pour retrouver leur chemin, perçoivent à leur tour les phéromones laissées par les autres.

Malgré le fait que l'on puisse comprendre aisément par-là que l'agent et l'environnement ne peuvent être dissociés, cette idée ne fait pas toujours l'unanimité au sein de la communauté des approches basées sur les agents. Pour certains, les agents peuvent se passer d'un tel environnement et interagir directement entre eux. Selon (Michel 2004), cette façon de penser n'est pas différente de celle qui ne dissocie pas l'environnement de l'agent, puisque pour communiquer, l'agent purement communicant devra utiliser un support de communication qui possède des propriétés d'un environnement. Ce raisonnement sera renforcé également par (Weyns et al. 2005), qui établit l'importance de l'environnement et le présente sous deux aspects (Franceschini 2017) : la structure de l'environnement et la dynamique de l'environnement.

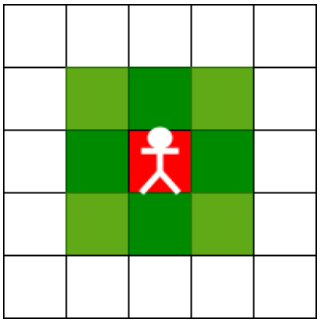
1.2.2.4.1 Structure de l'environnement

Sur le plan structurel, l'environnement est un espace virtuel qui régit le comportement de l'agent sur le plan physique (déplacement, topologie). Comme tout espace, il peut être continu ou discret.

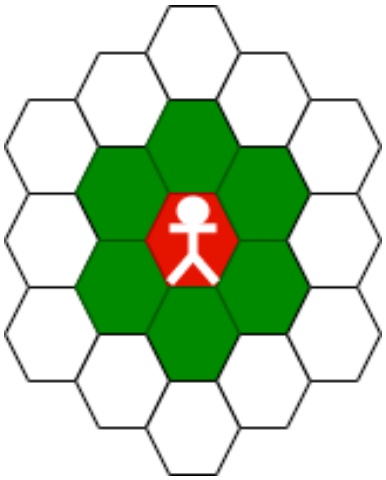
- L'environnement est dit continu lorsqu'il est défini sur l'ensemble des réels. Sur le plan géométrique, il est possible d'avoir des points infinitésimaux et donc des positions très fines. Cette structure est utilisée notamment lorsqu'une très grande précision est nécessaire.
- L'environnement discret est principalement utilisé et rencontré dans les travaux ayant porté sur la modélisation et la simulation des agents (ex :(Helbing 2012)). C'est une conception simplifiée de l'espace continu. Ici, on retrouve principalement deux types d'environnement : l'environnement cellulaire et l'environnement sous forme de graphe. Ce qui les distingue, c'est le nombre de voisins directs qu'un agent peut avoir. Les environnements cellulaires ont un nombre limité de voisins, contrairement aux environnements sous forme de graphe qui peuvent en avoir plusieurs (voir figure 1.12d). Cependant, les algorithmes pour explorer ceux-ci ont une complexité cyclomatique plus grande, ce qui peut être très coûteux en temps et en ressources lors de la simulation (McCabe 1996 ; Ebert et al. 2016). La figure 1.12 présente quelques représentations d'environnements discrets. Elle met en évidence les voisins directs qu'un agent peut avoir selon chaque type d'environnement. La figure 1.12b représente un environnement cellulaire formé par des carrés. Ici, il n'est possible d'avoir que quatre cellules directement voisines de l'agent. C'est la représentation la plus courante. Dans certains cas, l'agent peut se déplacer en diagonale, ce qui permet d'avoir huit cellules voisines. La figure 1.12c représente un environnement cellulaire formé par des hexagones. Le nombre de cellules directement voisines est de six.



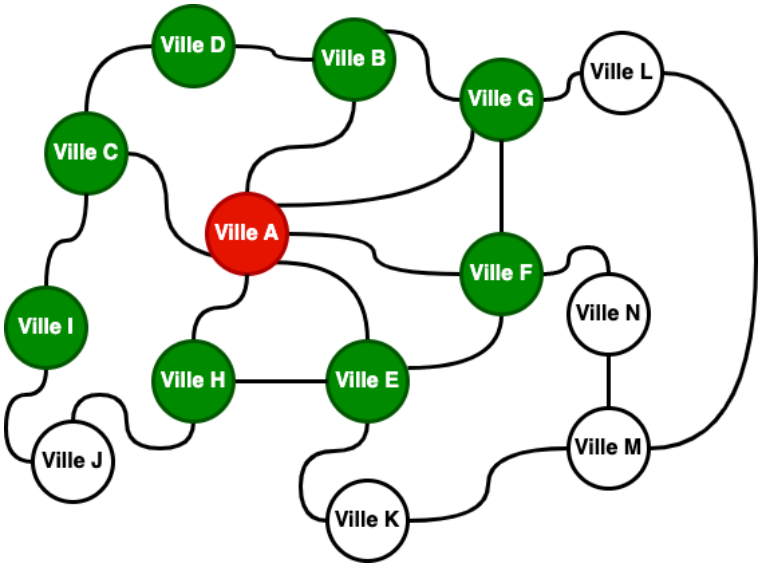
(a) Environnement cellulaire avec des cellules triangulaires (3 cellules voisines)



(b) Environnement cellulaire avec des cellules carrées (4 ou 8 cellules voisines)



(c) Environnement cellulaire avec des cellules hexagonales (6 cellules voisines)



(d) Environnements discrets : Représentation sous forme de graphe (8 nœuds voisins dans ce cas)

FIGURE 1.12 – Structure physique des environnements discrets : Représentation cellulaire

1.2.2.4.2 Dynamique de l'environnement

Au-delà de sa structure, l'environnement peut avoir des propriétés qui peuvent aussi influencer le comportement de l'agent. C'est donc un aspect à ne pas négliger et à prendre en compte lors de la modélisation d'un système. En effet, un environnement ne reste pas nécessairement figé dans le temps et peut être amené à évoluer [19]. L'évolution d'un environnement peut se faire dans deux cas :

19. On parle d'environnement statique (ne change pas sans l'intervention d'un agent) et d'environnement dynamique

— *Cas 1 : Actions d'agents* : Cas où les agents, par leurs actions, vont opérer directement des changements sur l'environnement. Ils peuvent par exemple changer sa structure (création/suppression de liens entre des nœuds dans un environnement sous forme de graphe), le dégrader (creuser un trou) ou même lever/ajouter des contraintes physiques.

— *Cas 2 : Règles endogènes* : L'évolution de l'environnement peut être régie par des règles définies. Ces règles permettent souvent de prendre en compte des facteurs naturels tels que la régénération de la végétation, le cycle de l'eau (pluie-évaporation-infiltration), la dissipation et l'évaporation des phéromones ou des odeurs.

Pour profiter pleinement des avantages qu'offre la modélisation avec les agents, il faut réaliser leur simulation avec plusieurs autres agents afin de leur permettre d'interagir de manière cohérente. On forme ainsi un système multi-agents (SMA) où les agents par leurs propres comportements peuvent, collaborer dans un même environnement.

1.2.2.5 Les Systèmes multi-agents

Comme nous l'avons expliqué jusqu'ici, les agents sont des entités qui tirent leur force de leur aptitude à collaborer. Un système multi-agents (SMA), comme on pourrait le comprendre par sa désignation, est tout simplement un ensemble formé de plusieurs agents. Puisque l'agent ne peut se dissocier de son environnement (Weyns et al. 2005), on pourrait donc dire qu'un système multi-agents est une extension de la technologie des agents dans laquelle un ensemble d'agents autonomes agit dans un environnement pour atteindre des objectifs communs ou spécifiques. Cela se fait en coopérant ou en rivalisant, en partageant ou non des connaissances entre eux (Balaji et al. 2010). Les agents qui constituent le SMA peuvent être différents les uns des autres, par leur type, par leur organisation structurelle ou encore par la connaissance qu'ils ont de l'environnement.

Faire collaborer un tel ensemble hétérogène d'éléments impose de définir (voire redéfinir) certains concepts propres aux agents qui caractérisent par conséquent le SMA. Il s'agit entre autres de l'environnement, des agents eux même, des relations entre les agents, les mécanismes de communication, les opérations réalisables et leur ordonnancement. Ces points sont détaillés ci-après.

1.2.2.6 L'environnement dans un SMA

En plus de prendre en compte les contraintes structurelles et dynamiques de l'environnement évoquées dans la section 1.2.2.4, il devient essentiel, dans le contexte d'un Système Multi-Agents (SMA), de développer une modélisation de l'environnement qui garantisse la mise à disposition de toutes les ressources nécessaires pour accueillir et superviser l'ensemble des agents en présence. Cette approche requiert une attention particulière aux détails afin de créer un environnement propice à l'interaction. Par exemple, lorsqu'on modélise un environnement (discret) pour une simulation SMA, il est utile de définir si dans une même case, plusieurs agents peuvent s'y retrouver au même moment.

1.2.2.6.1 Les agents

Contrairement à un système constitué d'un seul agent, dans un SMA, le comportement des agents n'est plus seulement influencé par l'environnement mais aussi par les autres agents qui y figurent. Par exemple, si un agent seul dans un environnement peut aller dans n'importe quelle direction, lorsqu'il se trouve dans un SMA, il devra désormais vérifier avant si la cellule est vide ou simplement si le déplacement est possible. La multiplicité des agents implique parfois leur diversité. Nous distinguons généralement dans un SMA deux types d'agents :

— *Les agents actifs* sont les entités actives du SMA. Ils ont la capacité d'agir directement dans le système et le modifier (ex : déplacer, développer, mourir, etc.).

— *Les agents passifs* (ou simplement objets) n'ont pas la capacité d'agir directement sur le système. Ils peuvent seulement être perçus, créés, détruits et modifiés par les agents actifs. Ces types d'agent sont généralement utilisés pour donner une certaine configuration à l'environnement [20].

1.2.2.6.2 Les relations entre les agents

Si les agents sont mis ensemble pour collaborer, c'est qu'il existe des relations entre eux. En d'autres termes, c'est la réponse à la question «que peuvent-ils faire les uns aux autres ?». Les réponses possibles à cette question font partie intégrante des SMA et devront être prises en compte dans la modélisation. Ces relations s'illustrent assez bien lorsque l'on souhaite modéliser un SMA constitué de proies et de prédateurs. Les proies peuvent se faire dévorer par des prédateurs.

1.2.2.6.3 Les mécanismes de communication

Modéliser un SMA afin que les agents puissent collaborer, requiert la mise en place de mécanismes de communication. On parle du *langage de communication des agents* (Singh 2003).

En fonction des objectifs et des relations entre agents, les mécanismes de communication doivent être efficaces [21] et répondre aux questions suivantes (Luncean et al. 2015) :

— Quel support ou canal de communication utiliser ?

— Comment indiquer le(s) destinataire(s) ?

— Comment le message est représenté et normalisé ?

— Que doit contenir le message ?

— Quand et comment le message doit-il être généré et transmis ?

— Comment le destinataire reçoit le message ?

20. Il ne s'agit pas ici d'une configuration au sens informatique mais plutôt d'une disposition physique comme par ajouter un cloisonnement pour former des zones

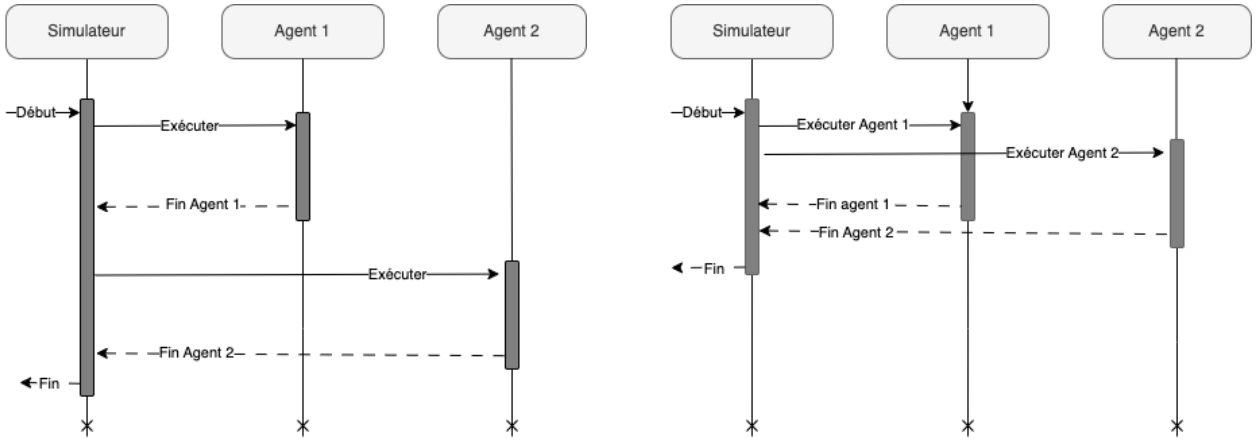
21. selon (Balaji et al. 2010), il faut éviter les communications inutiles ou redondantes

1.2.2.6.4 Les opérations

Les opérations s'inscrivent logiquement dans la suite des éléments précités. Communiquer, collaborer et atteindre les objectifs se font au moyen des opérations spécifiques qui caractérisent le SMA.

1.2.2.6.5 L'ordonnancement

Les SMA ont un lien étroit avec les systèmes distribués (Misik et al. 2016). Ils partagent également certaines problématiques, notamment en ce qui concerne l'ordonnancement et l'accès aux ressources. L'ordonnancement dans les SMA désigne l'ordre dans lequel les agents sont activés (Bittencourt et al. 2018). Cela signifie que l'issue de la simulation peut être modifiée en fonction de l'ordre d'activation des agents. Prenons l'exemple de la figure 1.14 qui illustre la simulation d'un modèle proie-prédateur où la proie a la capacité de se dupliquer. Nous constatons que pour les mêmes modèles, l'issue de la simulation n'est pas la même [22]. Dans la simulation montrée par la figure 1.14a où le prédateur est activé en premier, à la fin de la simulation (t=2) nous avons dans l'environnement deux proies et un prédateur. Alors que dans la simulation de la figure 1.14b au même instant, nous avons 5 proies et un prédateur. Dans les travaux que nous avons examinés portant sur les SMA, l'activation des agents est généralement effectuée de manière aléatoire (Coquelle 2005 ; Misik et al. 2016). Cependant, d'autres méthodes d'activation existent. Elles sont basées sur l'ordre d'ajout(Rasmussen et al. 2008) ou la durée du processus(Beisel et al. 2011). Une fois activés, les agents peuvent agir de manière synchrone ou asynchrone. Dans le cas de l'approche synchrone, un agent activé termine son cycle avant que l'agent suivant ne soit activé, tandis que dans l'approche asynchrone, dès qu'un agent est activé, le suivant est activé sans attendre la fin du cycle de l'agent précédemment activé. Le schéma de la figure 1.13 montre la différence d'exécution avec une approche synchrone et une approche asynchrone.

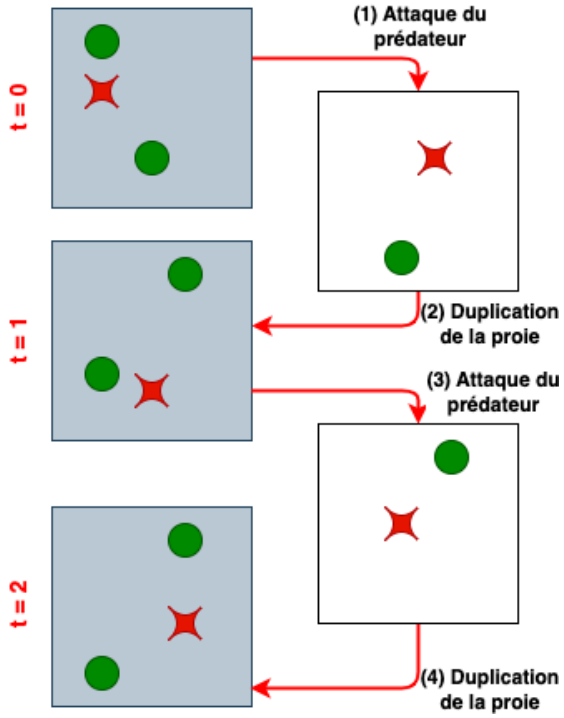


(a) Exécution de deux agents de façon synchrone

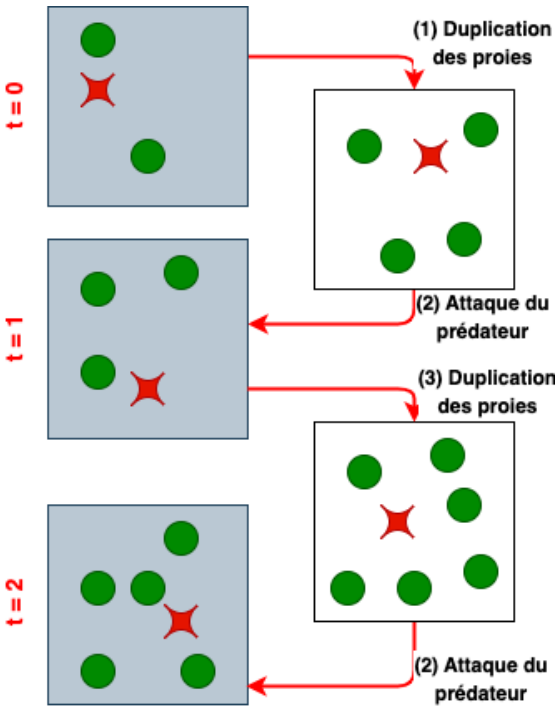
(b) Exécution de deux agents de façon asynchrone

Figure 1.13 – Exécution synchrone et asynchrone de deux agents

22. Ces aspects sont de plus en plus étudiés dans le cadre de la reproductibilité et de la répétabilité des résultats



(a) À chaque instant de cette simulation, l'agent «prédateur» est activé en premier avant les agents «proies». À la fin de la simulation, il reste autant d'agents qu'au début.



(b) À chaque instant de cette simulation, l'agent «proies» est activé en premier avant l'agent «prédateur». À la fin de la simulation, on a beaucoup plus de proies qu'au début.

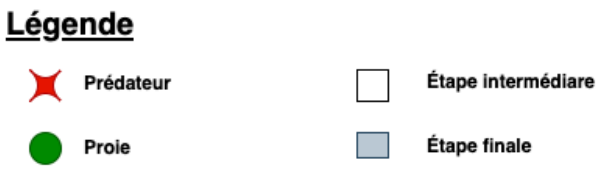


FIGURE 1.14 – Exemple d'issues possibles d'une simulation selon l'ordre d'activation des agents

1.2.2.7 Exemples de plateformes de modélisations des agents

Avec les multiples applications que les agents permettent, de plus en plus de travaux se sont intéressés à leur modélisation, en apportant des contributions dans la manière de concevoir, d'organiser et d'exécuter les agents. Ainsi, de nos jours, pour modéliser et simuler des agents, nous disposons d'une large gamme d'outils qui se distinguent les uns des autres soit par leur programmation, soit par leur facilité de mise en œuvre. Nous présentons ici des exemples probants des plateformes de modélisation et de simulation des SMA, en mettant l'accent sur les plus utilisées et les plus récentes.

1.2.2.7.1 NetLogo

Considérée de nos jours comme la plateforme de simulation la plus utilisée, NetLogo (Wilensky 1999 ; Tisue et al. 2004) est un logiciel libre et open source, disponible pour les principaux systèmes d'exploitation. Avec un langage de programmation dédié, son interface graphique et à sa documentation complète, NetLogo affiche clairement son objectif principal qui est d'être un outil éducatif facilitant la conception et l'utilisation des modèles (Al-Sharawneh et al. 2009). Les principaux domaines d'utilisation de NetLogo sont la sociologie, la biologie, la médecine, la physique, la chimie, les mathématiques, l'informatique, l'économie et la psychologie sociale (Allan 2009). Avec l'utilisation croissante du langage Python ces dernières années, il existe également des variantes de NetLogo en Python (Jaxa-Rozen et al. 2018).

1.2.2.7.2 Swarm

Swarm est une bibliothèque open source de modélisation et de simulation multi-agents conçue spécifiquement pour des applications de vie artificielle et l'étude des complexités. La première version a été mise à disposition du public en 1997, mais les premières versions stables n'ont été disponibles qu'en 2009. Développée en Objective-C et en Java, elle est compatible avec les plateformes UNIX et Windows. Swarm fournit un ensemble d'outils d'implémentation de simulation et d'analyse des modèles à évènements discrets. Swarm est basé sur un composant appelé «Model», lui-même composé d'un objet appelé «Swarm» (essaim) qui est une collection d'agents avec un planning d'évènements sur ces agents ; et un autre composant appelé «Observer» qui observe la simulation et l'affiche (Iba 2013). Swarm est principalement utilisé dans les domaines des sciences naturelles et sociales, du commerce et du marketing, de l'économie et des systèmes de planification des transports et de la mobilité (Abar et al. 2017).

1.2.2.7.3 Repast

Repast, pour *REcursive Porous Agent Simulation Toolkit* (North et al. 2006), est quasiment une implémentation dans le langage Java de la bibliothèque Swarm qui était originellement implémentée en Objective-C. La proximité entre Swarm et Repast se trouve à la fois dans leur philosophie et dans leur apparence, mais Repast ne met pas en œuvre les essaims. Bien que la modélisation et la simulation nécessitent des connaissances du langage Java de la part de l'utilisateur, depuis 2022, des extensions [23] permettent l'implémentation rapide de modèles simples sans nécessiter une grande expérience en programmation. Repast se positionne comme une plateforme qui apporte sa contribution dans le domaine des sciences sociales et comprend des outils spécifiques à ce domaine (Allan 2009).

1.2.2.7.4 MASON

MASON (Luke et al. 2005) (pour Multi-agents Simulator Of Neighbourhoods / Networks) propose une alternative plus simplifiée à Repast qui optimise les temps d'exécution des agents. Dans sa conception, il reste généraliste et propose une suite d'outils pouvant être

23. https://repast.github.io/

utilisés dans plusieurs domaines plutôt que d'être spécifiques à un domaine particulier. Ainsi, on peut aller de la simulation d'une entité unique (comme un robot) à des systèmes plus complexes tels que les essaims. Tout comme les précédents, il permet d'exécuter des modèles à évènements discrets. La contribution de MASON se situe principalement dans son fonctionnement modulaire, permettant de séparer la simulation de la visualisation et de suspendre une simulation sur une machine pour la reprendre sur une autre.

1.2.2.7.5 GAMA

GAMA (Amouroux et al. 2009) (pour GIS Agent-based Modeling Architecture) est une plateforme open source développée en 2007 par l'*Institut de la Francophonie pour l'Informatique* de l'époque [24] et met un accent particulier sur les données d'information géographique (Taillandier et al. 2012). En effet, ces dernières années, de plus en plus de problématiques sont liées à des données géographiques, comme les réseaux de distribution ou de transport, ou encore plus récemment avec les réseaux de contacts et de transmission de la COVID-19. Étant basé sur Java, il est compatible avec Windows, Linux et Mac OSX. GAMA se positionne ainsi comme un outil permettant de réaliser des simulations beaucoup plus proches de la situation réelle et d'effectuer des analyses spatiales plus approfondies, en intégrant les Systèmes d'Information Géographique (SIG). Une autre de ses forces est son langage de programmation intuitif (GAML) accessible aux non-informaticiens, dont la prise en main est possible en 10 minutes [25].

1.2.2.7.6 Mesa

Mesa (Kazil et al. 2020) est née à la suite d'un constat clair et apporte la solution. En effet, comme nous l'avons vu jusqu'à présent, il existe une multitude de plateformes pour modéliser et simuler des agents. Mais aucune d'entre elles n'est basée à l'origine sur Python [26], qui est depuis quelques années un langage très en vogue en raison de sa rapidité d'exécution et de sa simplicité syntaxique. De la même façon, elles nécessitent toutes une installation (parfois longue) avec de nombreuses dépendances. Mesa se positionne ainsi comme la première plateforme implémentée en Python et pouvant s'exécuter dans un navigateur. L'architecture de Mesa repose sur trois modules, à savoir son module de modélisation, son module d'analyse et son module de visualisation. Ils intègrent chacun, des fonctions prédéfinies utiles pour réaliser rapidement les simulations. Le module de modélisation sert à construire le modèle (les agents et leur environnement), le module d'analyse fournit des outils paramétrables pour collecter les données de simulation, et le module de visualisation contient une suite de bibliothèques pour visualiser dans un navigateur les simulations avec les données. Depuis 2022, il existe une extension de Mesa appelée *Mesa-Geo* qui répond au besoin croissant d'intégrer des SIG dans les simulations (B Wang et al. 2022).

24. Aujourd'hui, il est appelé Institut de la Francophonie pour l'Innovation (https://ifi.vnu.edu.vn/)
25. selon le site web du concepteur https://gama-platform.org/
26. À la date de création de Mesa

1.2.2.7.7 AgentPy

Comme la plupart des plateformes de modélisation et de simulation des systèmes Multi-Agents, AgentPy (Foramitti 2021) est open-source et apporte sa contribution à la simplification du processus de création et d'analyse des modèles pour des applications scientifiques. Cette simplification commence déjà par l'installation en une seule commande du gestionnaire de paquets pip (PyPI). Il est également accessible via des plateformes interactives telles que IPython, IPySimulate et Jupyter. AgentPy présente plusieurs similitudes avec Mesa, présenté précédemment, mais la principale différence réside vraiment dans la simplification de leur mise en œuvre. Une comparaison est proposée à l'adresse https://agentpy.readthedocs.io/en/latest/comparison.html [27].

Pour tous ces outils de modélisation et de simulation des SMA, un résumé est présenté dans le tableau 1.3.

27. Consulté le 01/O7/2023

TABLE 1.3 – Comparaison des plateformes de simulations basées sur les agents

Plateforme	**Langage**	**Type d'agent**	**Points forts**	**Points faibles**
NetLogo	Logo	Réactif et Cognitif	Grande communauté ; documentation complète ; prise en main facile utile dans un cadre pédagogique	Nécessité d'apprendre un langage dédié ; peut mobiliser beaucoup de ressources machine
Swarm	Objective-C et Java	Réactif et Cognitif	Multi plateformes ; forte scalabilté ;	Documentation limitée ; moins en moins utilisée ; modélisation basée sur le concept d'essaim
Repast	Java	Réactif	Flexibilité ; disponibilité de modèles prédéfinis et paramétrables ; outils d'analyse selon le domaine d'application	Documentation limitée, Interface utilisateur peu intuitive
MASON	Java	Réactif	Prise en main facile ; Modélise plusieurs types d'agents (prédéfinis) ; Reste performant pour de grands nombres d'agents.	Interface peu convival (not user-friendly)
GAMA	Java	Réactif et Cognitif	Fonctionnalités avancées pour SMA ; Interface graphique très conviviale ; Accepte les Systèmes d'Information Géographiques (SIG)	Mise à jour lourde et récurrente, Mobilise beaucoup de ressources machine ; Apprentissage et prise en main complexe.
Mesa	Python	Réactif	Simple pour apprendre la modélisation SMA ; Documentation détaillée ; Temps de simulation réduit	Pas d'interface de programmation, Très basique et rigide (difficile à faire évoluer ou à adapter à la modélisation du comportement Animal)
AgentPy	Python	Réactif	Simple pour apprendre la modélisation SMA ; Interopérable	Pas d'interface de programmation, Très basique et rigide

Après avoir réalisé l'examen minutieux des diverses approches de modélisation et de simulation, en mettant particulièrement l'accent sur le paradigme axé agent, il est désormais approprié de recentrer notre discussion sur le contexte spécifique de notre étude, à savoir celle de la modélisation du comportement animal et d'apporter des réponses à une question que l'on pourrait se poser. C'est «pourquoi modéliser le comportement animal et particulièrement en écologie comme ça l'est dans notre contexte d'étude ?». Cette question nous amène donc à présenter les intérêts d'une telle approche.

1.2.3 Intérêts à modéliser le comportement animal

D'un premier point de vue, il n'est pas évident de donner une réponse précise et toute faite à cette question compte tenu de son très large champ. Comme nous avons pu le voir dans la section 1.1, chaque discipline ou chaque courant de pensée a ses propres motivations et ses propres méthodes. De notre position, nous pensons pouvoir y répondre en considérant deux aspects :

Face aux enjeux écologiques et économiques de notre ère, disposer d'un outil permettant de reproduire virtuellement le comportement d'un animal afin de se projeter dans le temps et dans l'espace représente un atout majeur. Cet atout peut principalement se remarquer sur deux axes.

Premièrement, sur le plan écologique, nous ne sommes plus sans savoir que de nos jours certaines espèces sont menacées de disparition. En effet, les derniers chiffres font état de 28% des espèces qui sont menacées [28]. En améliorant la compréhension du comportement (par exemple, le cycle de vie ou de reproduction, des principaux prédateurs, etc.), il serait possible pour les experts et les décideurs de proposer des stratégies de préservation ou de restauration pour lesquelles l'efficacité peut être directement vérifiée par la simulation.

Deuxièmement, sur le plan économique, certaines espèces sont des leviers clés dans les économies locales et régionales. Elles représentent une part importante du chiffre d'affaires des professionnels. Mieux comprendre le comportement de ces espèces permettrait de les produire en quantité et en qualité accrues, ce qui aurait un impact encore plus grand sur l'économie.

1.2.4 Méthode de modélisation du Comportement animal

Comme présenté dans la figure 1.5, concevoir un modèle, soit-il celui d'un comportement animal passe aussi par les mêmes étapes que sont l'abstraction, la validation, l'implémentation, la vérification et le test de crédibilité. Néanmoins, une attention particulière est à porter à la phase d'abstraction et la phase d'implémentation, qui requièrent un certain contact rapproché avec l'animal et de disposer de certaines connaissances à son sujet.

— La phase d'abstraction est souvent très laborieuse et longue pour des raisons apparemment simples, mais qui posent des contraintes difficiles à lever. L'abstraction en elle-même est déjà l'étape où le concepteur identifie la partie (ou la fonction) du sujet d'étude à modéliser. L'animal, pouvant être considéré comme un système complexe [29], il n'est pas toujours aisé d'isoler une partie et de la décrire. Dans les cas où cela se fait, il demeure quand même une nécessité d'avoir certaines connaissances sur l'animal,

28. https://www.iucnredlist.org
29. Système formé de différentes entités qui effectuent des actions élémentaires

généralement issues des éthogrammes ou des observations faites directement dans des contextes précis. Observer un animal dans son milieu naturel ou même in vitro sur une longue durée sans perturber son comportement naturel reste une grosse contrainte pour modéliser le comportement animal. L'emblématique photo de Konrad Lorenz à la figure 1.15 est assez révélatrice des difficultés à pouvoir s'approcher au plus près des animaux pour les observer. Heureusement, aujourd'hui, avec la démocratisation des réseaux de capteurs sans fil et le développement de nouveaux moyens de communication, il est possible de disposer de moyens technologiques pour le suivi et l'observation des espèces d'intérêt, quel que soit leur milieu de vie.

FIGURE 1.15 – Konrad Lorenz, un des pères fondateurs de l'éthologie avec ses oies préférées. Photo tirée de (GADAGKAR 2021), ©Willamette Biology, CC BY-SA 2.0

— Lors de la phase d'implémentation du modèle conceptuel obtenu après abstraction, il est judicieux d'avoir du recul sur les objectifs et le contexte de la modélisation. Concrètement à cette phase, il s'agira de se demander qui seront les utilisateurs du modèle et quelle réponse ceux-ci attendent de la question à laquelle le modèle est destiné à répondre. Les éléments de réponse à ces deux questions permettront de choisir la bonne approche de modélisation et le bon outil d'implémentation. En effet, comme résumé dans le tableau 1.2 et le tableau 1.3, les approches de modélisation n'ont pas les mêmes niveaux de précision dans les réponses apportées et ne sont pas non plus présentées de la même manière.

— Dans les phases de vérification et de test de crédibilité, les mêmes problématiques que celles de la phase d'abstraction pourraient se poser, dans le sens où il faut aller au

contact de l'animal pour comparer les résultats produits par le modèle conçu et ce qu'il en est réellement chez l'animal lui-même.

Ces trois points que nous venons d'énumérer tendent à montrer le caractère crucial de l'observation dans la modélisation du comportement animal. L'animal par sa nature et l'environnement dans lequel il vit ne permet pas toujours de faire des observations sur des durées raisonnables. Par exemple pour l'UAR CNRS 3514 STELLA MARE qui étudie les espèces du milieu marin, il n'est pas toujours possible de faire des plongées sous-marines sur de longues durées pour observer les espèces d'intérêts, à cause de certaines profondeurs hostiles à l'humain ou simplement, il est limité par l'autonomie des bouteilles d'air. Pour cela, il devient parfois impératif de mettre en place une bonne partie d'instrumentalisation qui devra se faire bien avant la phase d'abstraction à proprement dite.

Une fois que toutes ces précautions sont prises, le modélisateur peut se lancer à proprement dit dans la conception de son modèle.

En nous basant sur les références, nous avons pu constater que cette modélisation du comportement animal suscite un grand intérêt pour la communauté scientifique, à voir le nombre de travaux portant sur cette thématique. Cette multiplicité de travaux a permis d'avoir aujourd'hui plusieurs façons de modéliser et de simuler le comportement animal. Nous en présenterons celles qui correspondent le plus à notre cadre de recherche dans la section suivante.

1.2.5 Approches de modélisation et simulation du comportement animal

Traditionnellement, la modélisation et la simulation de la dynamique du comportement des animaux se font au moyen des techniques spécifiques de mathématiques qui sont adoptées et mises en œuvre directement dans un langage de programmation. Au fil des évolutions et des adaptations réalisées est apparue une nouvelle notion, celle de l'***animat***, introduite par Stewart W. Wilson (SW Wilson 1986). Une définition du terme *animat* est proposée dans (Coquelle 2005). Selon Coquelle, l'animat désigne *«tout modèle calculatoire reproduisant un comportement animal autonome, ce qui inclut les modèles internes ainsi que les modèles observés d'un point de vue externe»*. Il faut comprendre de cette définition que toute représentation informatique d'un animal ou toute représentation informatique agissant comme un animal est un animat. Selon son concepteur, un animat peut être basiquement défini par quatre points :

1. Des senseurs représentant ses entrées qui lui permettent de recevoir des signaux (des informations) venant de lui ou de son environnement ;
2. Des actionneurs qui dotent l'animat de la capacité à agir sur l'environnement et de le modifier ;
3. Parmi les signaux perçus, il y en a qui sont spéciaux et déclenchent des actions spécifiques, et d'autres qui n'ont aucun effet. Dans un état d'absence de signaux (spéciaux ou non), il peut aussi déclencher des actions ;
4. Enfin, il agit tant à l'extérieur avec des opérations internes de sorte à optimiser le traitement des signaux spéciaux pour réduire l'occurrence des signaux spéciaux.

Cette définition du modèle d'animat reste très générique, mais pose les bases d'un domaine à part entière dans les sciences de l'étude du comportement animal. Le modèle d'animat va bien sûr connaître des évolutions, comme le modèle proposé par Roger K. Thomas (Thomas 2001) et cité par (Coquelle 2005). Bien que l'auteur ne parle pas explicitement de l'animat [30], dans le modèle qu'il propose, il distingue trois principales parties. Une partie extérieure appelée *«External Observable»* qui concerne toutes les interactions directes entre l'animat et l'environnement ; une partie interne appelée «Internal Observable» qui met en évidence les interactions propres à l'organisation interne de l'animat, et enfin une «interface» pour lier les deux parties. Cette organisation de l'animat lui permet d'exprimer les comportements de la façon suivante :

L'animat reçoit des stimulations appelées «External Observable Antecedent» (EOA) depuis l'environnement à travers des senseurs. Ces stimulations sont transmises à l'intérieur de l'animat sous la forme d'un «Internal Observable Antecedent» (IOA) et reçues par une interface. En fonction de l'IOA, l'interface ordonne l'action correspondante en générant dans l'organisme une IOC (Internal Observable Consequent) dont la détection par les actionneurs engendre une réponse (External Observable Consequent ou EOC) sur l'environnement. La figure 1.16 illustre cette représentation qu'on pourrait donner à un animat ainsi que les interactions qu'il peut avoir avec son environnement.

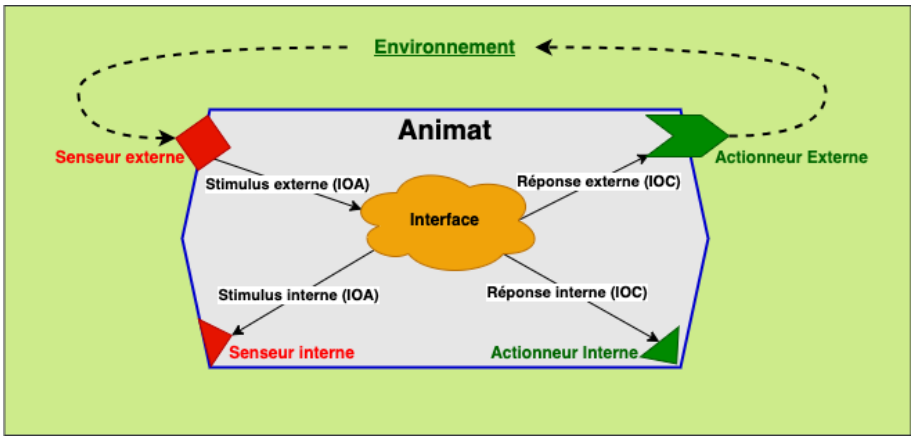


Figure 1.16 – Représentation d'un animat selon Roger K. Thomas cité par (Coquelle 2005)

30. Il utilise les termes «Concepts en science du comportement»

Depuis son apparition avec les travaux de Stewart W. Wilson, l'utilisation de la notion d'animat a contribué à la création de nouvelles connaissances dans l'étude du comportement animal grâce à des travaux qui ont permis de modéliser et de simuler le comportement animal. Nous passons ici en revue quelques-unes parmi les plus significatives.

1.2.5.1 Formalisme DESIRE pour la modélisation et simulation du comportement animal

Le formalisme DESIRE [31] (Korn 1996) est conçu pour développer des modèles de simulation des systèmes à base de connaissances pour des tâches de raisonnement complexes. Cette définition fait de lui un outil idéal pour modéliser et simuler le comportement animal. Au-delà du fait qu'il répond au principe de parcimonie prôné par le canon de Morgan pour la modélisation du comportement animal, des extensions relativement simples et naturelles du formalisme DESIRE adaptées aux tâches des systèmes multi-agents ont été proposées (FMT Brazier et al. 2000). La modélisation du comportement avec le formalisme DESIRE (F Brazier et al. 1996) se compose de trois types de connaissances que sont : le composant du processus, le composant des connaissances et la relation entre le composant du processus et le composant des connaissances.

— *Le composant de processus* définit les processus pertinents du système identifié, avec différents niveaux d'abstraction. Ils sont représentés sous forme de composants, avec des informations d'entrée et de sortie pour chacun. Les composants de processus peuvent être composés d'autres composants de processus ou des composants primitifs, capables de représenter les différents niveaux d'abstraction et d'exécuter des tâches spécifiques. La manière dont les processus à un niveau d'abstraction sont composés est appelée composition. La composition de processus offre donc une approche structurée pour décrire les interactions et les relations entre les différents niveaux d'abstraction des processus, facilitant ainsi la gestion et la compréhension du système complexe modélisé.

— *Les composants des connaissances* sont des structures qui représentent les différentes connaissances à différents niveaux d'abstraction. Elles sont clairement définies dans les niveaux d'abstraction de bas niveau et peuvent être même cachées dans les niveaux plus élevés.

— *La relation entre le composant du processus et le composant des connaissances* Chaque processus dans une composition de processus utilise des structures de connaissances. Les structures de connaissances utilisées pour chaque processus sont définies par la relation entre la composition de processus et la composition de connaissances.

Un cas d'utilisation du formalisme DESIRE pour modéliser et simuler le comportement des animaux est présenté dans (Jonker et al. 2001) qui modélise différents types de comportements, allant du comportement purement réactif au comportement proactif, social et adaptatif. Un exemple est présenté dans la figure 1.17 montrant comment un modèle générique d'agent (à un niveau d'abstraction macro) est graphiquement représenté.

31. DEsign and Specification of Interacting REasoning components

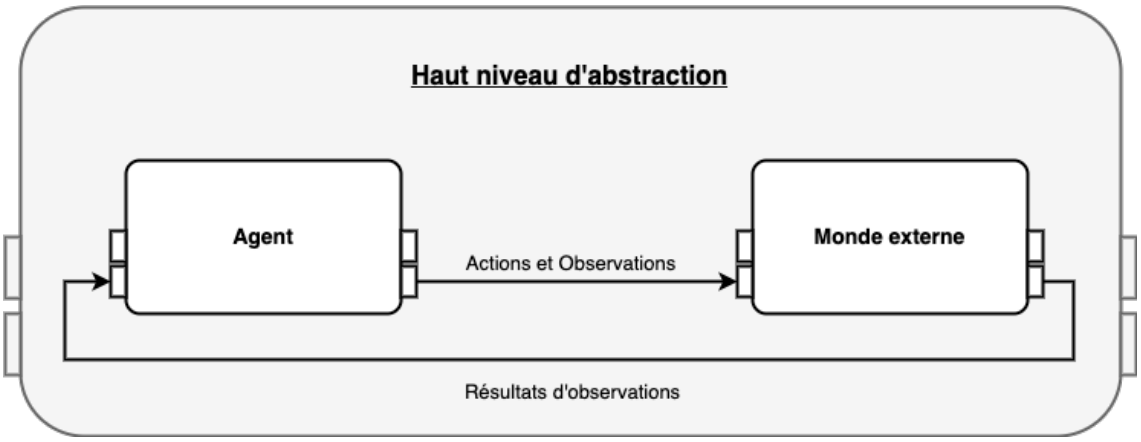


FIGURE 1.17 – Modèle d'agent générique pour un comportement purement réactif.

Dans cet exemple, il s'agit d'un modèle pour un comportement purement réactif, où chacune des compositions « agent » et « monde externe » est définie par ses propres composants de processus, ses propres composants de connaissance et les relations entre ces deux types de composants.

En somme, ce qu'il faut retenir de ce formalisme, c'est qu'il est défini de façon générique pour modéliser des systèmes qui répondent à des règles et des connaissances en prenant en compte la possibilité d'avoir divers niveaux d'abstraction. Il permet ainsi de modéliser le comportement des animaux avec différents niveaux de complexité. Même si la conception des modèles est relativement simple, leur simulation est peu intuitive.

1.2.5.2 Ethomodelisation

L'«Ethomodelisation» est le fruit des travaux de thèse de doctorat en informatique de Alexis Drogoul(DROGOUL 1993) . Cette approche est un compromis entre les multiples modèles de comportement qui existaient en ces moments (modèle tropisme [32], modèle behavioriste, modèle de Lorenz). Le modèle de comportement en ethomodélisation est basé sur les systèmes multi-agents (agents réactifs) où l'agent exécute des tâches. Chaque tâche est déclenchée par un ou plusieurs stimuli particuliers, qu'ils soient internes ou externes. Une tâche est composée d'une séquence de «primitives de comportement» qu'elle exécute lorsqu'elle est déclenchée. La figure 1.18 montre l'architecture d'un agent en ethomodélisation. On peut remarquer que l'agent est capable de recevoir des stimuli internes, mais aussi des stimuli venant de son environnement. À ces stimuli, l'agent «réagit» par l'exécution des tâches correspondantes qui elles-mêmes sont définies par des primitives de comportement (illustrées ici par le cercle, le carré et le triangle).

32. Modèle selon lequel les causes et les conséquences du comportement animal seraient semblables à celui de l'humain, autrement dit, il s'agit de donner une représentation «humaine» à l'animal

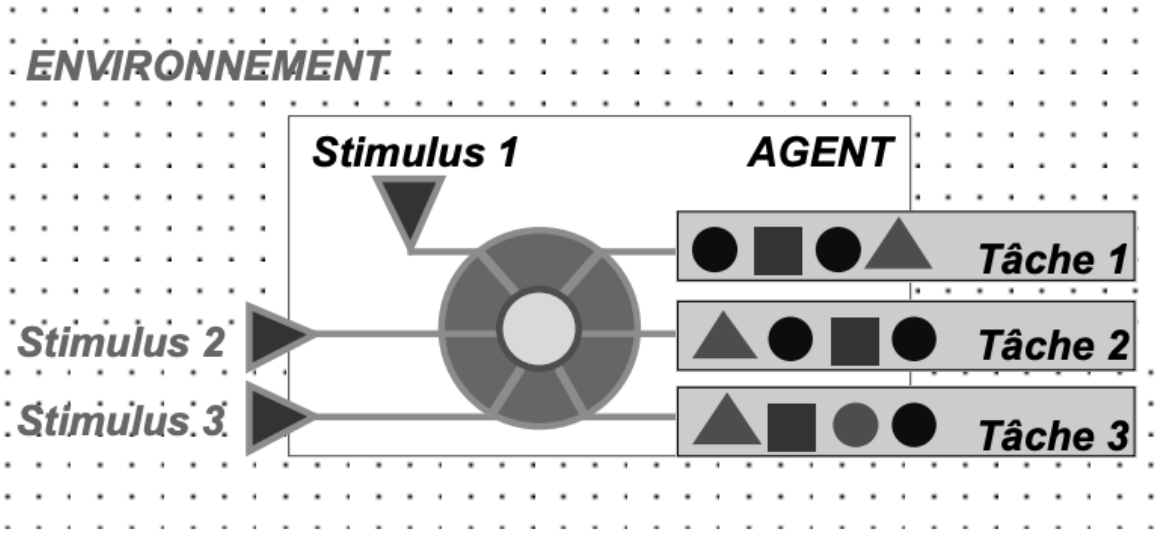


FIGURE 1.18 – Architecture fonctionnelle d'Ethomodélisation (DROGOUL 1993)

1.2.5.3 behavioRis

Développé dans le cadre des travaux de Coquelle Ludovic (COQUELLE 2005), behavioRis est un outil de simulation de modèles éthologiques basés sur une plateforme multi-agents et qui s'appuie sur les connaissances en éthologie (les éthogrammes en particulier). Cet outil a été mis sur pied afin de rendre accessible la simulation des modèles éthologiques à un public non «informaticien», notamment aux éthologues afin qu'ils puissent décrire et simuler les comportements décrits dans un éthogramme indépendamment du modèle d'étude. L'outil behavioRis est assez «généraliste» et offre la possibilité de décrire le comportement à simuler à l'aide d'un langage dédié pour son prototypage. À cet effet, l'architecture de l'agent repose sur un «animat [33]» qui inclut les modèles internes comme les modèles observés d'un point de vue externe. La particularité de behavioRis est qu'il utilise la logique floue pour exprimer les règles de comportement manipulant les données imprécises. La figure 1.19 montre l'architecture fonctionnelle d'un animat dans Behavioris. L'animat perçoit son environnement grâce à des composants «Perception» qui déclenchent à leur tour des comportements (Composant «behaviour») en fonction de l'état interne de l'animat(composant Internal state). Chaque comportement déclenche une liste d'actions motrices exécutées par des contrôleurs(composant Controller).

33. Tout modèle calculatoire reproduisant un comportement animal autonome

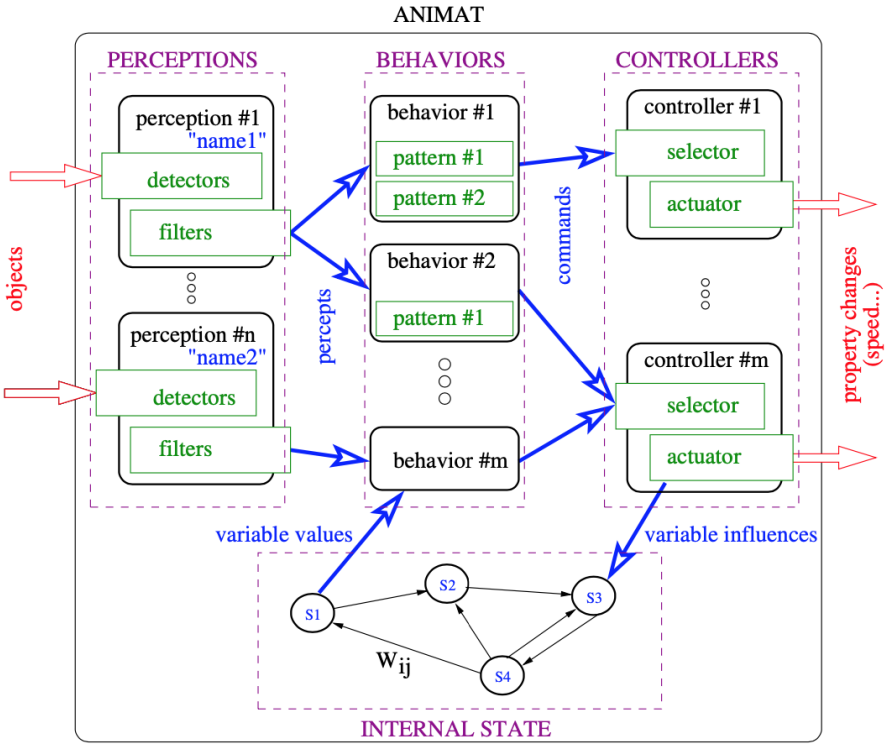


FIGURE 1.19 – Architecture fonctionnelle de Behavioris (COQUELLE 2005)

1.2.5.4 Apprentissage automatique pour la modélisation du mouvement des animaux

Les différentes méthodes de modélisation et de simulation du comportement animal susmentionnées sont toutes orientées vers l'inférence des comportements, plutôt que vers leur prédiction à proprement dit. Dans un contexte pareil, les algorithmes d'apprentissage automatique et d'apprentissage profond (Machine Learning et Deep Learning) constituent des outils idéaux pour répondre à ce besoin de prédire le comportement des animaux, mais ils sont rarement utilisés à cette fin (WIJEYAKULASURIYA et al. 2020). Néanmoins, nous trouvons dans la littérature quelques travaux qui s'y sont intéressés (VALLETTA et al. 2017). Deux grandes catégories semblent se dégager de ces travaux, la première, majoritaire, concerne des outils d'analyse et de classification des comportements observés, tandis que la seconde concerne la conception de modèles pour prédire les comportements.

Dans la catégorie des outils de classification, on retrouve plusieurs travaux ayant porté sur le comportement d'espèces d'insectes, d'oiseaux, de poissons ou plus généralement de mammifères (MARTISKAINEN et al. 2009 ; SCHANK et al. 2011 ; CARROLL et al. 2014 ; ASHWOOD et al. 2022). Parmi la multitude de travaux présents dans la littérature, deux vont principalement nous intéresser en raison de leur conception, de leur philosophie et de leurs données d'entrée.

— Le premier est présenté dans (GRÜNEWÄLDER et al. 2012). Cette approche utilise des techniques d'apprentissage automatique pour concevoir un modèle de classification des

comportements clés des guépards, tels que l'alimentation, le déplacement et le repos. La conception du modèle repose sur des données de déplacement d'individus équipés de colliers GPS pendant environ un an. L'objectif d'un tel modèle est d'identifier les différences saisonnières et de genre dans l'activité quotidienne et les heures d'alimentation. Cette approche est relativement hybride, car elle combine un SVM (Support Vector Machine) (Deris et al. 2011 ; Apsemidis et al. 2020) avec une chaîne de Markov cachée (Mor et al. 2021) pour réaliser la classification.

— *DeepEthogram* (Bohnslav et al. 2021) est une méthode modulaire utilisant un réseau de neurones artificiels. Cette méthode a pour objectif de détecter et de classer automatiquement les comportements spécifiés par l'utilisateur dans une séquence vidéo. Le processus implique un modèle d'apprentissage en profondeur supervisé qui, avec un minimum de données d'entraînement fournies par l'utilisateur, est capable de prendre une vidéo et un ensemble de comportements préalablement définis, pour générer un modèle d'apprentissage profond binaire.
De façon traditionnelle, la classification des comportements dans une vidéo se fait en visualisant toute la vidéo pour identifier et compter le temps passé par l'individu dans un état ou un autre. Dans DeepEthogram, une première phase d'entraînement consiste à l'utilisateur de fournir au logiciel de petites séquences de vidéos contenant les comportements d'intérêt. Ces comportements sont étiquetés dans les vidéos à l'aide d'une interface utilisateur, comme le montre la figure 1.20. Au terme de l'entraînement, le modèle conçu est capable d'identifier chaque comportement et d'estimer le temps passé dans cet état avec une précision supérieure à 90 %, ce qui correspond à une performance humaine de niveau expert.

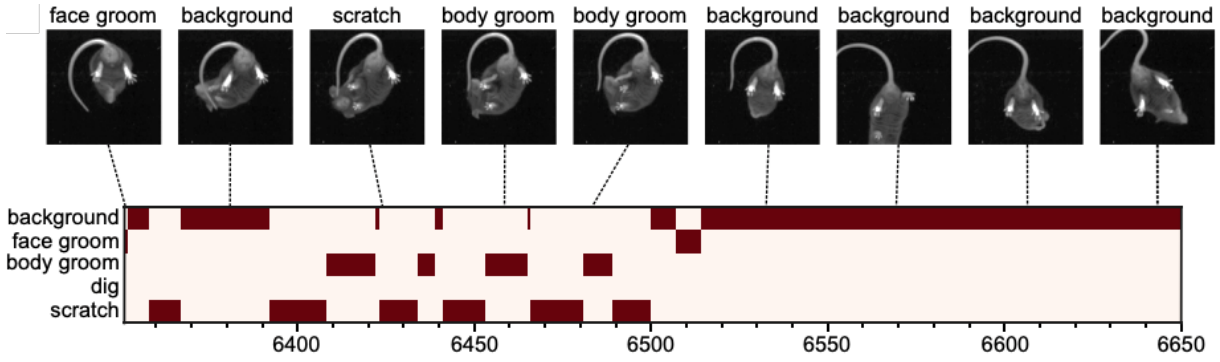


Figure 1.20 – Étiquetage de vidéo dans DeepEthogram (Bohnslav et al. 2021)

Dans ce même style d'utilisation, nous avons aussi l'outil *DeepLabCut* (Mathis et al. 2018) qui offre des méthodes de tracking d'un individu ou d'un ensemble de zones d'intérêts dans une vidéo. Il repose également sur des réseaux neuronaux profonds.

Ces deux approches présentées sont d'un apport considérable et permettent de gagner du temps pour les experts du comportement animal dans l'établissement des éthogrammes. Elles permettent d'identifier rapidement et précisément des comportements spécifiques dans un ensemble de données (positions GPS, vidéos). Cependant, un flou demeure quant aux critères qui caractérisent l'identification de ces comportements dans tout ce flux de données.

Dans la seconde catégorie qui est celle des modèles de prédiction du comportement des animaux, nous avons trouvé très peu de travaux à ce sujet. Les principaux que nous avons identifiés sont les suivants.

— Dans (Browning et al. 2018), les chercheurs se sont intéressés à la prédiction du comportement de plongée chez les « oiseaux de mer ». Pour ce faire, les auteurs ont opté pour un modèle d'apprentissage profond supervisé entraîné par des données GPS. La validation des prédictions réalisées sur des espèces telles que le guillemot de troïl, le cochevis d'Europe et le pingouin torda a abouti à une conclusion cruciale sur la possibilité de réaliser efficacement ces prédictions avec seulement les données GPS pour ces espèces.

— Dans quasiment le même contexte que (Browning et al. 2018), les travaux dans (Wijeyakulasuriya et al. 2020) ont consisté à la conception de modèles utilisant des algorithmes d'apprentissage automatique et d'apprentissage profond pour prédire le mouvement des fourmis et des mouettes. Prédire précisément la trajectoire des animaux est une tâche qui peut s'avérer complexe et laborieuse pour une multitude de raisons. Pour cela, l'approche proposée ici est assez novatrice. La prédiction de la trajectoire à proprement dite se fait en deux étapes. La première étape consiste à prédire d'abord l'état comportemental dans lequel se trouve l'animal étudié (ex : recherche de nourriture, repos, errance, etc.). Partant de cette première prédiction, la deuxième étape prédit, en fonction de cet état, la vitesse et la direction du déplacement.

Le passage en revue de toutes ces approches, méthodes et outils de modélisation et de simulation du comportement animal, que ce soit pour les classifier ou pour les prédire concrètement, montre que c'est un champ qui est très exploré par les spécialistes du comportement animal. C'est un domaine d'application dont les résultats ont permis des avancées notables dans la compréhension du comportement des animaux.

Toutefois, une question primordiale se pose notamment sur les techniques qui utilisent l'apprentissage automatique, en particulier l'apprentissage profond, pour lesquelles des résultats très précis sont obtenus, mais il n'est toujours pas possible d'expliquer concrètement ces résultats obtenus, contrairement aux méthodes algorithmiques ou d'inférence pour lesquelles les résultats peuvent s'expliquer par des règles. Il s'agit là de la notion d'interprétabilité et d'explicabilité.

1.3 Les méthodes d'optimisation, un levier pour des modèles explicables ?

La question de l'interprétabilité et de l'explicabilité dans les modèles informatiques occupe une place cruciale dans le processus de modélisation, particulièrement pour les scientifiques de l'UAR CNRS 3514 STELLA MARE dans le sens où elle est essentielle pour comprendre le fonctionnement des modèles et favoriser la confiance lors de leurs utilisations.

1.3.1 Comprendre le modèle par l'Interprétabilité et l'Explicabilité

Comme nous avons pu nous en apercevoir tout au long de ce chapitre, lorsque l'on modélise un système, on le fait pour mieux le comprendre et prendre des décisions. Cette recherche de connaissance a ouvert le champ à la conception d'une multitude d'approches et de méthodes, des plus simples aux plus complexes, pour la modélisation. Sans surprise, certaines d'entre elles permettent d'obtenir des résultats remarquables. Mais bien qu'ils soient très significatifs et satisfaisants, une question à laquelle il faut impérativement trouver une réponse se pose : comment comprendre et expliquer une décision prise à l'aide d'un modèle ? Jusqu'à quel niveau peut-on « faire confiance » à un modèle ? On pourrait comparer cette situation à celle d'un médecin qui a un traitement efficace contre une maladie, mais ne sait pas comment et pourquoi cela fonctionne. En tant que patients, serions-nous prêts à prendre ce traitement ? Une autre situation serait celle de quelqu'un qui, en observant les pilotes, a acquis la capacité de piloter un avion de manière experte, mais on ne sait pas comment il y parvient ni ce qui motive chacune de ses actions. Serions-nous prêts à monter dans un avion piloté par cette personne ? Dans ces deux situations, pour qu'un processus soit accepté par le grand public, il faut disposer de suffisamment d'éléments de compréhension. On parle d'interprétabilité et d'explicabilité des modèles (Mittelstadt et al. 2019).

Dans la littérature, nous retrouvons plusieurs définitions proposées pour ces deux concepts. Concernant l'interprétabilité, deux définitions reviennent couramment(Linardatos et al. 2021).

— Dans (Doshi-Velez et al. 2017), l'interprétabilité est définie comme *«la capacité d'expliquer ou de présenter en termes compréhensibles à un humain»*
— Pour (Miller 2019), l'interprétabilité c'est *«le degré de compréhension de l'information par l'utilisateur»*

À partir de ces deux définitions, nous nous apercevons que l'interprétabilité fait référence à la capacité d'apporter des informations nécessaires et suffisantes à un humain pour comprendre le fonctionnement du modèle. C'est donc un champ réservé aux experts de la modélisation ou des données, dont la finalité est de fournir des informations sur le raisonnement interne qui conduit aux résultats fournis par le modèle. Lorsque le modèle est peu (voire pas) interprétable, il est dit *«opaque»* ou *«boîte noire»*. Ces dénominations se justifient par le fait qu'il n'y a pas de transparence dans le processus interne du modèle (Burrell 2016).

L'explicabilité quant à elle consiste à fournir à un néophyte ou à un technophile des informations qui permettent de rendre compréhensibles les résultats fournis par un modèle, le tout dans un vocabulaire (ou langage) simple, accessible et suffisamment compréhensible (Gilpin et al. 2018 ; Miller 2019).

Ce qu'il faut retenir de ces différentes définitions, c'est que l'interprétabilité se focalise sur ce qui se passe à l'intérieur du modèle et répond aux questions *comment le modèle procède-t-il pour prendre une décision qui conduit à ses résultats ?* L'explicabilité, de son côté, œuvre pour l'acceptabilité des résultats du modèle. Elle vise à répondre à un humain quelconque qui poserait la question *pourquoi le modèle produit ses résultats ?* L'explicabilité ne peut donc pas se défaire de l'interprétabilité, puisqu'elle en est de fait la première étape pour la rendre possible. Ces deux concepts concourent donc à un même but, qui est de comprendre le fonctionnement et les décisions du modèle.

Il existe de nombreux modèles qui sont entièrement interprétables en raison des approches

de modélisation utilisées pour les construire. On peut citer par exemple les équations différentielles, les chaînes de Markov, les automates finis. Ils se basent sur des règles précises ou des raisonnements probabilistes qui permettent de justifier clairement l'obtention ou non d'un résultat. Il n'en reste pas moins que certaines approches basées sur les données, notamment la régression (linéaire, logistique, softmax), sont également entièrement interprétables (Li 2022). En revanche, ce n'est pas toujours le cas de toutes les autres méthodes d'apprentissage automatique, en particulier les réseaux de neurones (deep learning (Deng et al. 2014)), qui, malgré leur efficacité face à des problèmes de plus en plus complexes, restent difficiles à comprendre. Plus un modèle est précis, plus il est difficile à expliquer (Abdullah et al. 2021) comme illustré par la figure 1.21. Ce qui peut paraître paradoxal, c'est qu'ils sont utilisés dans des secteurs cruciaux tels que la santé et les politiques de gestion, où la compréhension est fondamentale.

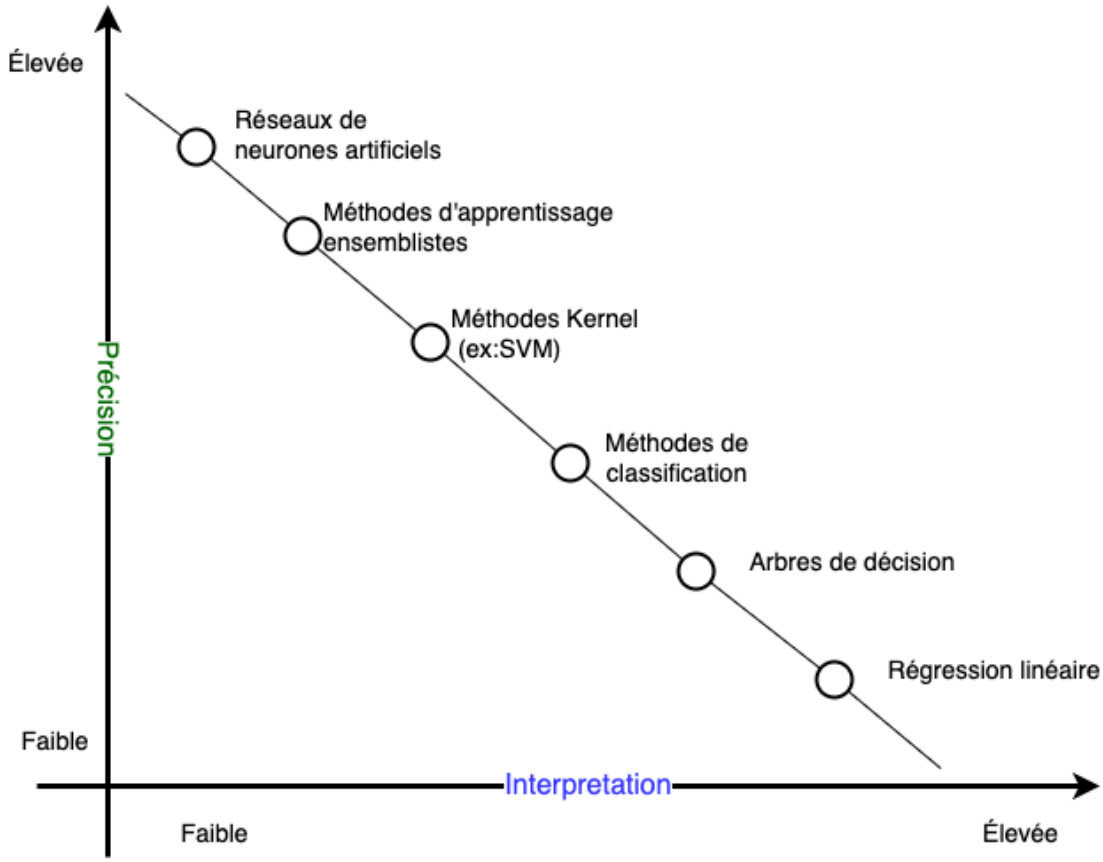


Figure 1.21 – Explicabilité des modèles en fonction des approches de modélisation (Abdullah et al. 2021)

Lorsqu'il s'agit de modéliser le comportement des animaux, les points cruciaux que nous avons identifiés concernent principalement l'existence d'une multitude d'approches, chacune apportant ses avantages et ses inconvénients. Ces modèles varient en fonction du contexte, et souvent, il faut trouver un compromis délicat entre simplicité, explication et précision. D'un côté, certains modèles sont relativement simples à implémenter, mais sont relativement peu précis. D'un autre côté, des modèles plus complexes offrent une meilleure précision, mais leur

opacité rend difficile leur explication. Ainsi, il est impératif de trouver une approche adéquate qui permet de générer des modèles à la fois explicables et précis. Les méthodes d'optimisation se profilent comme une solution prometteuse pour relever ce défi en cherchant à équilibrer ces aspects, offrant ainsi une perspective encourageante pour la modélisation du comportement animal.

1.3.2 Modéliser le comportement des animaux comme un problème d'optimisation

Les travaux significatifs en lien avec la modélisation du comportement animal que nous avons parcourus l'ont souvent considéré comme une série d'actions déclenchées par des stimuli (Tang et al. 2010 ; Coquelle 2005 ; Drogoul 1993). À partir de ce constat, si nous prenons également en considération le comportement d'un animal comme tel et que nous disposons d'un ensemble d'actions connues dans lequel nous parvenons à identifier la séquence d'actions appropriée[34], qui corresponde à celle de l'animal en question, nous pourrions alors réussir à élaborer un modèle de son comportement. Cependant, une question se pose : comment pouvons-nous constituer cet ensemble de base d'actions connues dans un contexte où nous voulons justement acquérir plus de connaissances sur le comportement à modéliser ? Une piste intéressante pour contourner cette contrainte peut être trouvée dans la conception que René Descartes avait du comportement humain. Selon lui, l'étude du comportement des animaux pouvait aider, voire permettre de comprendre, le comportement humain. Il utilisait ainsi les connaissances acquises chez un être vivant (l'animal) pour créer de nouvelles connaissances concernant un autre être vivant (l'humain). En émettant nous-mêmes l'hypothèse selon laquelle *«il serait possible de modéliser le comportement d'une espèce à partir des connaissances acquises chez d'autres»*, nous pourrions envisager d'utiliser des actions élémentaires connues observées chez d'autres espèces pour constituer l'ensemble d'actions dans lequel nous devrions trouver la bonne séquence d'actions.

Cette seule hypothèse ne saurait être suffisante pour modéliser efficacement le comportement d'une espèce à partir des connaissances acquises chez d'autres espèces, dans la mesure où les comportements peuvent parfois différer. Pour mieux comprendre ce que nous évoquons, supposons que nous sommes dans le cadre de la modélisation du déplacement d'une espèce A, connue pour sa lenteur dans son déplacement, et que l'ensemble d'actions à disposition pour le modéliser provient d'une espèce B, connue quant à elle pour sa rapidité de déplacement par rapport à l'espèce A. Il ne serait pas évident dans ce cas précis que notre hypothèse soit directement valable. Néanmoins, on peut toutefois se dire qu'il suffirait de prendre le déplacement de cette espèce B et limiter sa vitesse à une valeur proche de celle de l'espèce A. Notre première hypothèse pourrait donc être renforcée par une seconde, qui est : *«Il serait possible de retrouver chez différentes espèces un même comportement avec variations.* Ces variations pourraient être par exemple le paramétrage ou l'ordre d'exécution.

En combinant ces deux hypothèses, on pourrait donc dire que si nous souhaitons modéliser le comportement d'une certaine espèce A, nous pourrions nous baser sur un ensemble Δ d'actions élémentaires provenant d'autres espèces, dans lequel nous pouvons trouver une combinaison C^* d'actions provenant de Δ de manière à ce que C^* soit constitué d'actions

34. c'est-à-dire les bonnes fonctions comportementales

qui reproduisent le comportement réel de A. Pour que ces actions puissent reproduire au mieux le comportement réel de l'espèce A, il est nécessaire qu'elles soient paramétrables. Ainsi, générer un modèle pour cette espèce revient à trouver, d'une part, la combinaison idéale d'actions, et d'autre part, trouver le bon paramétrage pour que l'ensemble reproduise au mieux le comportement attendu. Une illustration de cette façon de générer un modèle est proposée par la figure 1.22. Le comportement attendu pour l'espèce modélisée est défini par les actions $A_5^{p=2}$; $A_3^{p=3.6}$; $A_1^{p=14.5}$ dans cet ordre, avec respectivement comme paramètre les valeurs 2 ; 3.6 ; et 14.5. Dans l'ensemble d'actions, les combinaisons $C_1 = (A_1^{p=15}; A_4^{p=0.3}; A_2^{p=1.1})$; $C_2 = (A_5^{p=3}; A_2^{p=3.3}; A_1^{p=14.6})$; et $C_3 = (A_4^{p=2.7}; A_1^{p=11.9}; A_6^{p=6.14})$ peuvent être formées avec ces paramètres associés. Parmi ces propositions, C_2 est celle qui se rapproche le plus de la combinaison attendue. Elle sera donc le modèle généré et présenté au modélisateur.

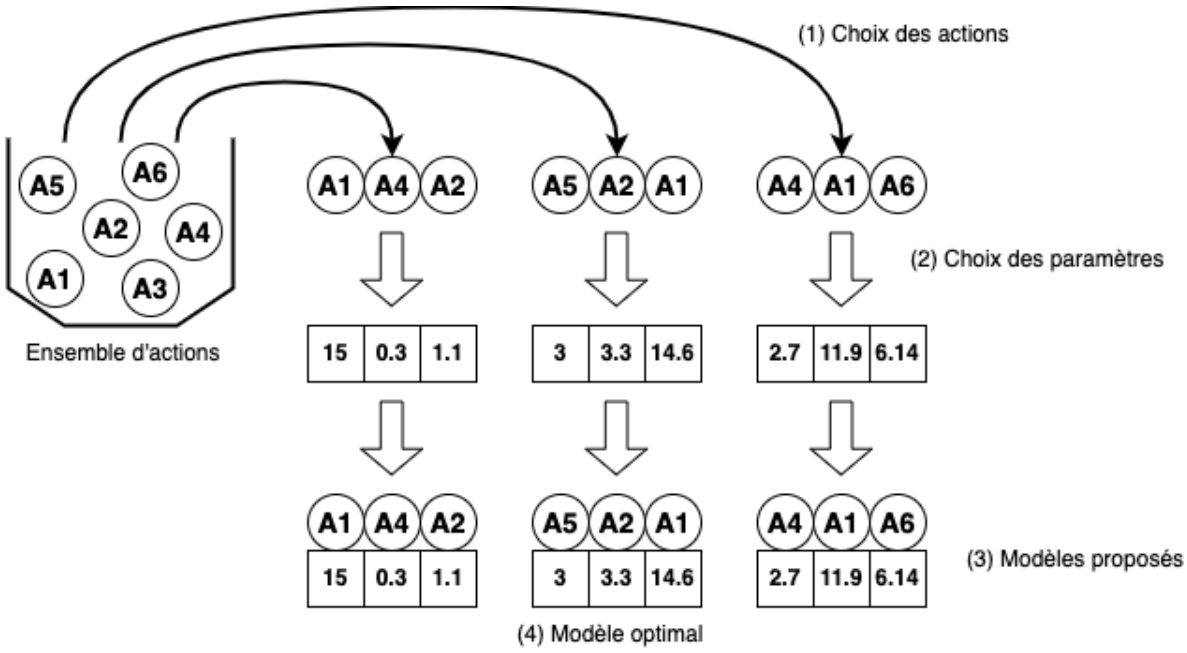


Figure 1.22 – Hypothèse de modélisation guidée par optimisation

Notre objectif étant désormais de trouver une combinaison optimale d'actions et des paramétrages optimaux, nous nous retrouvons face à un problème d'optimisation pour trouver la meilleure combinaison d'actions et déterminer les paramétrages optimaux. Considérer la génération de modèle comme un problème d'optimisation résolvable par les métaheuristiques apporte une solution concrète à la problématique d'interprétabilité et d'explicabilité à laquelle les concepteurs et les utilisateurs du modèle sont souvent confrontés. En effet, le modèle généré étant constitué d'actions connues, il rend son exécution transparente pour le concepteur et l'utilisateur. Il devient ainsi possible d'expliquer clairement comment et pourquoi le modèle produit les résultats observés. De plus, le fait que les actions élémentaires proviennent de modèles connus ou d'éthogrammes réputés pour leur niveau de précision dans les explications renforce notre conviction que le modèle final obtenu par notre approche est fortement explicable.

Un autre profit que nous pouvons tirer de cette façon de modéliser le comportement des animaux est que celle-ci pourrait permettre de générer des modèles même pour des espèces pour lesquelles très peu ou pas de connaissances sont disponibles. Concrètement, étant donné que la génération de modèle peut se baser sur d'autres actions, il est possible de construire un modèle pour une espèce dont nous ne disposons pas de connaissances complètes à travers un processus simple. Ledit processus consistera d'abord à trouver, parmi l'ensemble des actions connues, la combinaison qui correspond le mieux aux observations effectuées. Étant donné que le modèle est considéré comme explicable, il peut ensuite être analysé par un expert (biologiste ou éthologue) qui pourra le valider ou l'invalider. L'intervention de l'expert permet aussi de traiter les cas d'équifinalité. Une situation d'équifinalité (Von Bertalanffy 1956) se produit lorsqu'il existe de multiples solutions, pour résoudre un même problème. Dans notre contexte, il s'agira des situations où une combinaison d'actions trouvée produit des résultats similaires à celui observé in situ, mais qui n'est pas une combinaison qui reflète véritablement le comportement de l'animal modélisé.

Conclusion

Au cours de ce chapitre, nous avons exploré le domaine de la modélisation du comportement animal sous différents angles, qu'ils soient philosophiques, biologiques ou informatiques. Nous avons constaté que chaque approche de modélisation et de simulation se concentre sur des aspects spécifiques ; ce qui entraîne souvent un compromis. Cependant, malgré cette diversité, les différentes tâches associées à ces approches, telles que la mise en œuvre, l'exploitation des modèles et leur configuration, restent souvent fastidieuses et basées sur des intuitions et des hypothèses propres à chaque expert. De plus, il arrive fréquemment que des connaissances précises sur le système à modéliser fassent défaut, et les experts peuvent avoir des connaissances limitées, ce qui peut introduire des biais. Face à cette réalité, une possible solution pourrait être trouvée dans le domaine de l'optimisation en considérant la modélisation du comportement animal comme un problème d'optimisation à résoudre.

CHAPITRE 2

Résolution de problèmes d'optimisation

Sommaire

Introduction

Dans les différentes actions du quotidien de l'humain, celui-ci recherche une certaine efficacité relative à des objectifs propres. Il peut s'agir de se déplacer en prenant le chemin le plus court, de conclure une affaire en faisant le profit maximum ou encore de réaliser quelque chose avec le moindre coût. Avec l'évolution des systèmes et des contraintes de plus en plus grandes, il devient également de plus en plus difficile de prendre en compte tous les facteurs pouvant influencer une décision et raisonner dans des délais acceptables. C'est bien là aussi l'une des difficultés que peuvent rencontrer les experts de la modélisation et particulièrement de la modélisation du comportement animal. En effet, comme nous l'avons présenté dans le chapitre 1, à différentes étapes de la conception d'un modèle, le choix d'une méthode et de la configuration associée peut être très fastidieux surtout lorsque ces modèles font intervenir un nombre important de variables ayant des relations non linéaires. De plus, pour certains, le paramétrage est réalisé de manière intuitive, sur la base de suppositions propres à chaque concepteur. Étant donné que l'on ne dispose justement pas de connaissances précises sur le système réel, ces suppositions peuvent être mises en cause.

Face à cet état de choses, une solution à ces contraintes peut être apportée par une discipline à cheval entre les mathématiques et l'informatique appelée *«Optimisation par ordinateur»* [1] (XS Yang et al. 2011). Dans ce chapitre, nous présentons ce qu'est un problème d'optimisation ainsi que ses principales méthodes de résolution par ordinateur. Nous découvrirons la notion de complexité des problèmes d'optimisation, puis nous nous focalisons sur les métaheuristiques, une catégorie de méthodes d'optimisation particulièrement efficace face à des problèmes de grandes complexités.

2.1 Problèmes d'optimisation

De façon standard, un problème d'optimisation est défini de la façon suivante :

$$\begin{aligned} &\textbf{Variables} : && X \in \Omega \\ &\textbf{Objectif} : && f : \Omega \to \mathbb{R} \\ & && x \to f(x) \\ &\textbf{Contraintes} : && D \subseteq \Omega \end{aligned} \tag{2.1}$$

Minimiser ou maximiser $f(x)$ tel que $D \subseteq \mathbb{R}^k$.

— x est appelé variable d'optimisation ou variable de décision ;
— k est la dimension du problème c'est-à-dire le nombre de variables de décision ;
— Ω est appelé *«espace de recherche»* ;
— D est l'ensemble des contraintes du problème ;
— f est appelée *«fonction d'évaluation»* qui à chaque variable associe une valeur dans $\mathbb{R}$.

Illustrons cette définition par un exemple simple. Supposons un commerçant qui doit récupérer en une seule fois des conteneurs de marchandises avec son camion qui transporte

1. «computational optimization» en anglais.

au maximum 15 tonnes. Arrivé au port, il se retrouve face aux conteneurs suivants :

— Un conteneur A d'une tonne qui génère un bénéfice de 1 000 euros ;
— Un conteneur B d'une tonne qui génère un bénéfice de 2 000 euros ;
— Un conteneur C de deux tonnes qui génère un bénéfice de 2 000 euros ;
— Un conteneur D de quatre tonnes qui génère un bénéfice de 10 000 euros ;
— Un conteneur E de douze tonnes qui génère un bénéfice de 4 000 euros.

Quels conteneurs devra choisir le commerçant pour faire le maximum de bénéfice, tout en tenant dans les limites de son camion ?[2].

Répondre à cette question revient à résoudre un problème d'optimisation. Dans ce cas précis, avec de simples raisonnements, on pourrait trouver des solutions réalisables par énumération de toutes les possibilités et choisir la meilleure après les avoir évaluées. Mais cette façon de faire est de moins en moins possible lorsque le nombre de possibilités est grand. Il faudra alors formuler le problème sous une forme mathématique pour la résoudre.

Résoudre un problème d'optimisation revient à chercher les extrémums de la fonction f.
Soit une fonction $f : I \subset \mathbb{R}^n \to \mathbb{R}$

— $x \in \mathbb{R}^n$, $f(x)$ est un **extremum (minimum ou maximum) local** de f sur I si :

$x \in I$ et $\exists r \in I$ tel que $f(r) \leq f(x)$ (on parle de minimum local) **ou**

$x \in I$ et $\exists r \in I$ tel que $f(r) \geq f(x)$ (on parle de maximum local)

— $x \in \mathbb{R}^n$, $f(x)$ est un **extremum (minimum ou maximum) global** de f sur I si :

$x \in I$ et $\forall r \in I$ tel que $f(x) \leq f(r)$ (on parle de minimum global) **ou**

$x \in I$ et $\forall r \in I$ tel que $f(x) \geq f(r)$ (on parle de maximum global)

Des exemples d'extrémums locaux et d'extrémums globaux sont montrés dans la figure 2.1.

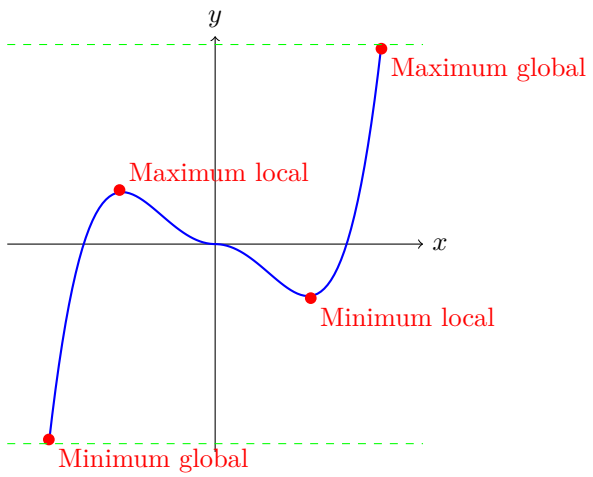


FIGURE 2.1 – Extrémums locaux et globaux d'une courbe

2. Problème du sac-à-dos : https://fr.wikipedia.org/wiki/Probl%C3%A8me_du_sac_%C3%A0_dos (09/01/2024 à 11 :07)

On dit qu'une solution x_{opt} est une solution optimale à un problème d'optimisation défini par l'équation 2.1 lorsque x_{opt} respecte toutes les contraintes de D et que $\forall x \in \Omega, f(x_{opt}) \leq f(x)$ lorsqu'il s'agit d'une minimisation ou $f(x_{opt}) \geq f(x)$ lorsqu'il s'agit d'une maximisation. Dans les cas où le problème est multi-variables, $X = [x_1, x_2, ..., x_n]$ est un vecteur des variables d'optimisation.

Comme nous pouvons le remarquer dans l'exemple du commerçant à la section 2.1, lorsque le nombre de possibilités (c'est-à-dire l'espace de recherche) est grand, la résolution du problème devient relativement coûteuse tant sur le plan temporel que spatial. La résolution du problème devient donc difficile, voire impossible (Weise et al. 2009). Cette difficulté s'identifie aisément à travers l'exemple suivant : Supposons un bus qui doit parcourir $n = 2$ stations $x_1 et x_2$. Dans ce cas, seulement deux itinéraires $I_1 = (x_1 - x_2)$ et $I_2 = (x_2 - x_1)$ sont possibles. Lorsque n passe à 3, six itinéraires sont possibles : $I_1 = (x_1 - x_2 - x_3), I_2 = (x_1 - x_3 - x_2), I_3 = (x_2 - x_1 - x_3), I_4 = (x_2 - x_3 - x_1), I_5 = (x_3 - x_1 - x_2), I_6 = (x_3 - x_2 - x_1)$, soit 3!. Lorsqu'on passe à $n = 4$, $n = 10$ ou $n = 100$, on se retrouve respectivement à $4! = 24$, $10! = 3628800$, $100! = 0,933.10^{158}$. Le problème devient très vite hors de portée.

Le niveau de difficulté de résolution d'un problème d'optimisation est donc lié à la complexité des calculs mis en œuvre (Papadimitriou 2003). Qu'est-ce qui fait alors que la résolution d'un problème est plus complexe qu'une autre et comment mesure-t-on cette complexité ? Une réponse est apportée dans (Cook 2007) par la notation «Big-O» (Landau 2005 ; Bachmann 2005) où la complexité est calculée en fonction du nombre d'opérations O, k une constante positive et de la taille de l'entrée n. Le tableau 2.1 présente quelques exemples de calcul de complexité.

Table 2.1 – Calcul de complexité par la notation «Big-O»

Notation	**Type de la Complexité**	**Exemple**
$O(1)$	Constante	Changer un élément dans un tableau
$O(log\ n)$	Logarithmique	Recherche binaire
$O(n)$	Linéaire	Déterminer la taille d'une liste
$O(n^k)$	Polynomial	Parcourir un graphe. Ex : DFS (Tarjan 1972)
$O(k^n)$	Exponentiel	SAT Solver (Gong et al. 2017)
$O(n!)$	Factoriel	Opérations récursives

2.2 Classification des problèmes d'optimisation

Dans la littérature, on retrouve principalement trois manières de classer les problèmes d'optimisation. La première se base sur le type des variables de décision, la deuxième est basée sur l'objectif du problème et la troisième repose sur la complexité du problème déterminé par la méthode «Big-O».

2.2.1 Classification selon les variables de décisions

Dans la classification selon les variables de décisions, nous distinguons deux types de problèmes d'optimisation. Les problèmes à variables discrètes, et les problèmes à variables continues (DRÉO et al. 2003).

2.2.1.1 Optimisation à variables discrètes

L'optimisation à variables discrètes, ou simplement l'optimisation discrète traite des problèmes où les variables de décision sont exclusivement des valeurs entières et forment un sous-ensemble de $\mathbb{N}$ ($x \in \Omega \mid \Omega \subset \mathbb{N}$). Parfois, x peut ne pas être un élément de $\mathbb{N}$ et représenter un objet tel qu'un conteneur dans l'exemple du commerçant présenté à la section 2.1. On parle toujours dans ce cas d'optimisation discrète (ou combinatoire) (RABBOUCH et al. 2023), dans la mesure où on associe des valeurs entières aux objets (BLUM et al. 2001). Un exemple d'association qui peut être réalisée est présenté dans le tableau 2.2. Si une solution pour le commerçant est $ConteneurA + ConteneurC + ConteneurD$, elle peut être représentée par le triplet $(1, 3, 4)$.

Parmi les exemples de problèmes à variables discrètes, nous pouvons citer le problème du voyageur de commerce (LAPORTE 1992), les problèmes de routage (SIMSIR et al. 2019) et les problèmes de planification (LEGRAIN et al. 2014).

TABLE 2.2 – Association d'objets à des valeurs entière dans le cadre d'une optimisation combinatoire

Objets	**Valeur associée dans** $\mathbb{N}$
Conteneur A	1
Conteneur B	2
Conteneur C	3
Conteneur D	4
Conteneur E	5

2.2.1.2 Optimisation à variables continues

Dans l'optimisation à variables continues, les variables de décision peuvent prendre toutes les valeurs de l'intervalle sur lequel le problème est défini. Généralement, cet intervalle Ω est un sous-ensemble de $\mathbb{R}$, voire les deux peuvent même être confondus. Comme exemple d'optimisation à variables continues, supposons un industriel qui souhaite produire un emballage cylindrique pour conditionner des produits d'un volume $v = C^{ste}$. Pour réduire les coûts de matières premières et utiliser une quantité de papier raisonnable, il cherche à trouver le couple $x = (r, h)$ (avec r le rayon de base et h la hauteur du cylindre) qui lui permet d'avoir une surface minimale à couvrir avec l'étiquette tout en maintenant le volume $v = k$. La figure 2.2 montre trois solutions possibles pour ce problème que sont les couples (r_1, h_1) (r_2, h_2) et (r_3, h_3). Comme nous pouvons le constater, r et h peuvent prendre différentes valeurs de $\mathbb{R}^*$, par exemple $59,910503$ et $22,051994$, ou encore 29. Ici, l'espace de recherche Ω est $\mathbb{R}^{*2}$, un sous-ensemble de $\mathbb{R}^2$.

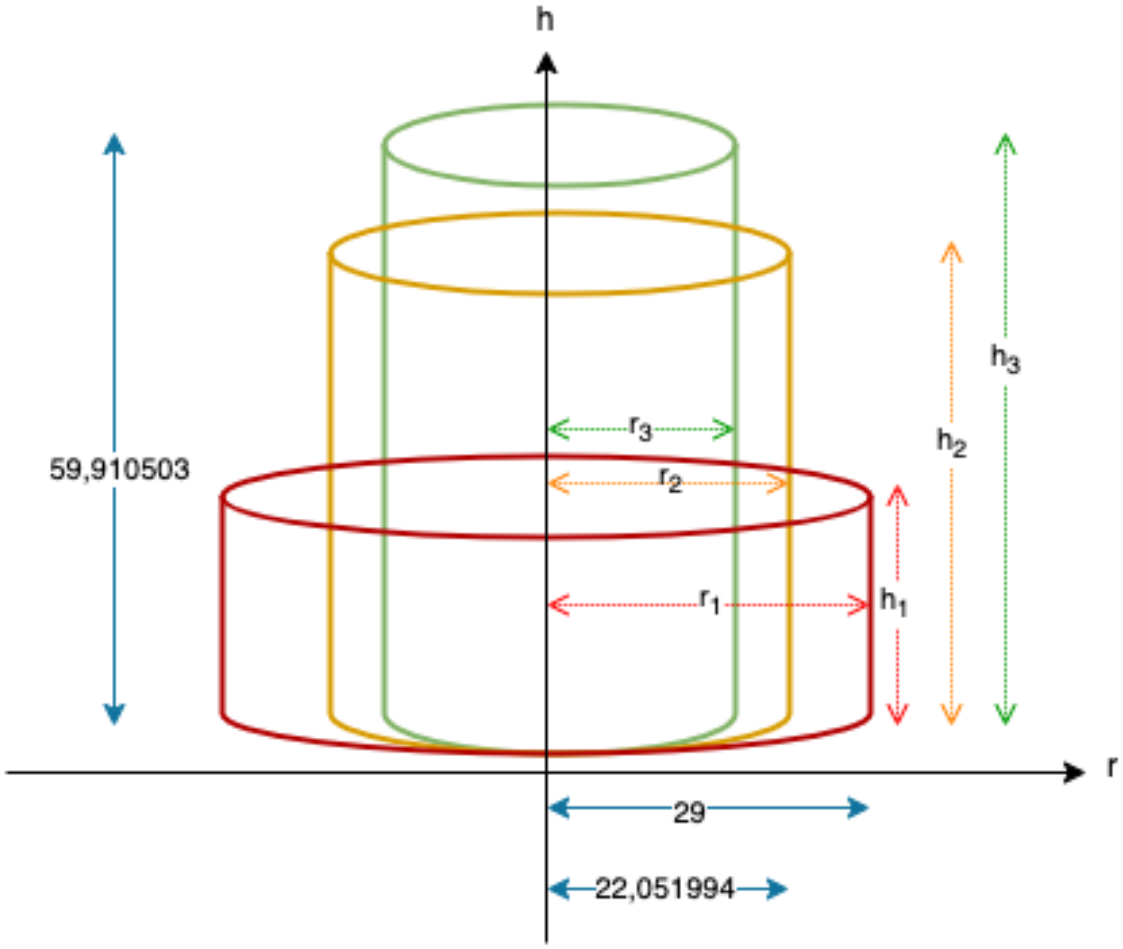


FIGURE 2.2 – Solutions possibles pour un problème à variables continues

2.2.2 Classification selon les objectifs

Un objectif, au sens d'un problème d'optimisation, est le résultat «espéré» à la fin du processus. L'objectif d'un problème est modélisé et indiqué par sa «fonction objectif», destinée à être minimisée ou maximisée selon le cas. Sur cette base, nous distinguons deux classes de problèmes d'optimisation en utilisant les objectifs comme critère de classification : les problèmes mono-objectifs et les problèmes multi-objectifs.

— *Les problèmes mono-objectifs*, comme leur nom l'indique, ont un objectif unique, donc une seule «fonction objectif». L'exemple de la section 2.2.1.2 est un problème mono-objectif avec comme seul objectif de minimiser la surface définie par l'unique fonction $f : 2\pi r(r + h)$ sous la contrainte $\pi r^2 h = v$. Il est important de faire remarquer la différence entre l'objectif et les variables de décision. Bien que le problème ait plusieurs variables de décision (r et h), il reste un problème mono-objectif par sa fonction f.

— *Les problèmes multi-objectifs* (COLLETTE et al. 2011), quant à eux, ont plusieurs objectifs. Résoudre un problème multi-objectifs revient à trouver une solution qui satisfait

tous les objectifs. Parfois, ces objectifs peuvent être contradictoires [3]. Dans ces cas, la résolution du problème consiste à trouver une solution optimale qui fait un bon compromis des objectifs (ex : front de pareto (Gunantara 2018)). Une autre approche de résolution de ce type de problèmes est de le reformuler en un problème mono-objectif avec une fonction objectif qui combine tous les autres ou en changeant les contraintes [4].

2.2.3 Classification selon la complexité

Portée par la théorie des complexités, cette classification se base sur le temps de résolution des problèmes déterminée la notation Big-O. En réalité, ce n'est pas une notation dédiée exclusivement aux problèmes d'optimisation, mais il s'agit d'une notation utilisée pour les problèmes (informatiques) en général.

2.2.3.1 Les problèmes de la Classe P

Les problèmes de la classe P sont ceux qui peuvent être résolus par une machine de Turing déterministe (Turing 1936 ; Papadimitriou 2003). On entend par machine de Turing déterministe une machine qui fait un seul calcul à la fois [5]. Une telle machine permet de résoudre les problèmes dans des temps polynomiaux. Cette classe caractérise donc l'ensemble des problèmes que l'on peut résoudre dans des temps polynomiaux, c'est-à-dire que le temps de calcul est de la forme $O(n^k)$ pour tout $k \in \mathbb{N}^+$ (avec k une constante positive et n la taille de l'entrée).

2.2.3.2 Les problèmes de la classe NP

La classe NP *(Non-déterministe Polynomial)* regroupe les problèmes qui peuvent être résolus par une machine de Turing non-déterministe(Turing 1936 ; Kozen 1976). On peut voir une telle machine comme étant capable d'exécuter en parallèle un nombre fini d'alternatives [6]. La résolution de ces problèmes prend un temps exponentiel, mais la vérification des solutions trouvées peut être faite dans un temps polynomial. Ces problèmes sont souvent comparés à des énigmes qui peuvent être très difficiles à résoudre, mais une fois la réponse trouvée, elle peut être vérifiée plus rapidement. Dans cette catégorie de problèmes NP, nous distinguons deux sous-catégories qui sont les NP-Complet et NP-Difficile.

Les problèmes de la classe *NP-Complet* respectent les critères suivants :

— Il est possible de trouver une solution en un temps polynomial acceptable [7] ;
— Tous les problèmes de la classe NP se ramènent à celui-ci via une réduction polynomiale ; cela signifie que le problème est au moins aussi difficile que tous les autres problèmes de la classe NP.

Les problèmes de la classe *NP-Difficile* quant à eux sont les problèmes NP pour lesquels il n'est pas facile de vérifier qu'une solution est acceptable.

3. Exemple : Maximiser la distance à parcourir en voiture et minimiser le temps de parcours. Or, on sait que la distance parcourue est proportionnelle au temps de parcours

4. Minimiser le temps de parcours et maximiser la distance parcourue peut être résumé en un seul objectif qui serait de trouver la vitesse optimale, une variable qui prend en compte à la fois la distance et le temps

5. Nos ordinateurs sont des machines déterministes

6. À ce jour, une telle machine est purement théorique

7. l'acceptabilité est relative à celui qui veut résoudre le problème

La figure 2.3 présente un récapitulatif de la classification des problèmes d'optimisation qui ont été présentés.

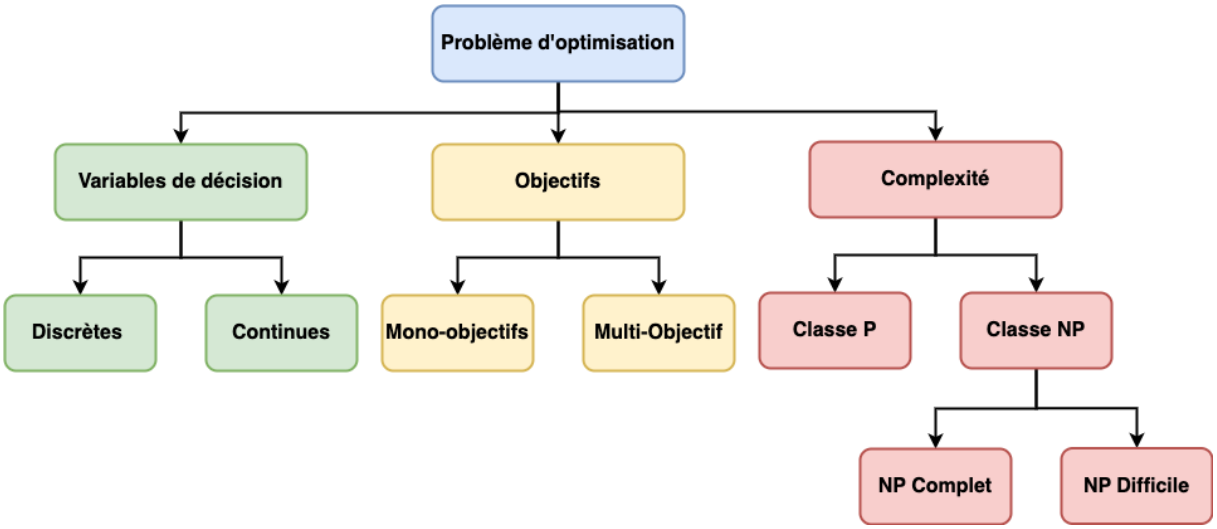


FIGURE 2.3 – Classification des problèmes d'optimisation

Après avoir examiné les diverses catégories de problèmes d'optimisation, il est maintenant judicieux de s'intéresser aux méthodes de résolution qui ont été développées pour les résoudre.

2.3 Méthodes de résolution de problèmes d'optimisation

Lorsque nous prenons la classification des problèmes selon leur complexité, nous pouvons voir que des deux grandes catégories, il y en a une qui regroupe des problèmes pour lesquels il y a la certitude de les résoudre (les problèmes de la classe P résolus avec une machine de Turing déterministe) et d'autres pour lesquels il n'existe pas un algorithme qui garantit de trouver la meilleure solution, mais une fois qu'une solution est proposée, sa validité et sa justesse peuvent être vérifiées. Cette différenciation des problèmes permet de facto d'avoir deux grandes classes de méthodes de résolution de problèmes d'optimisation : les méthodes exactes et les méthodes approchées (figure 2.4).

Les méthodes exactes sont utilisées pour la résolution des problèmes de la classe P et certains problèmes de la classe NP-Complet de petite instance. Elles regroupent l'ensemble des méthodes qui explorent de façon «intelligente» et systématique tout l'espace de recherche afin de trouver une solution optimale si celle-ci existe, d'évaluer et de garantir son optimalité(JOURDAN et al. 2009 ; GOGNA et al. 2013). Dans cette catégorie, les méthodes les plus utilisées sont la programmation linéaire (KORTE et al. 2010), la méthode simplex (EISELT et al. 2007), l'Algorithme A*(FOEAD et al. 2021) et les méthodes de la famille «Branch and X» (BARNHART et al. 1998 ; MORRISON et al. 2016 ; TAWARMALANI et al. 2005). Même si ces méthodes permettent de résoudre efficacement certains problèmes, elles sont connues pour être très coûteuses en temps et ne peuvent pas être appliquées à des problèmes de grande complexité qui en plus peuvent être non linéaires. C'est là qu'intervient la seconde catégorie de méthodes, c'est-à-dire les méthodes approchées (ou méthodes probabilistes).

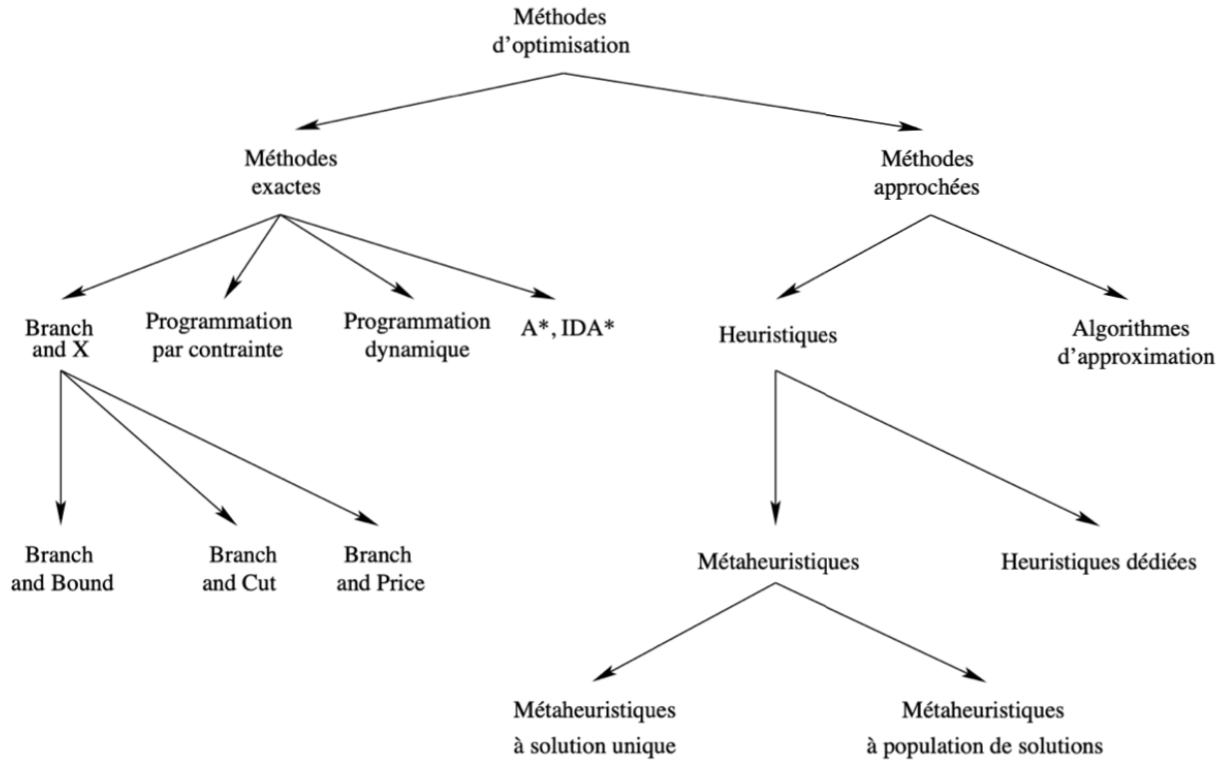


FIGURE 2.4 – Une (possible) classification hiérarchisée des méthodes d'optimisation

Les méthodes approchées se basent sur des raisonnements aléatoires et probabilistes pour trouver des solutions les plus proches possible de la solution «exacte». Ces méthodes sont particulièrement efficaces pour les problèmes complexes, notamment NP-Complet puisqu'elles exploitent une de leurs propriétés pour proposer des solutions. Il s'agit de la propriété de vérification des solutions en temps raisonnables. En effet, dans un processus itératif et incrémental, les méthodes approchées vont proposer des solutions qui pourront être vérifiées et améliorées (si besoin) jusqu'à trouver une solution satisfaisante dans des temps raisonnables. La qualité d'une méthode approchée va donc se calculer par rapport à l'écart obtenu entre sa solution et l'optimale. Évidemment, cette approche ne garantit pas que la solution trouvée soit l'optimum global. Pour cela, l'optimalité de la solution est définie par des critères définis au départ. On parle ici de «conditions d'arrêt». Généralement, il s'agit d'un nombre maximum d'itérations ou d'une valeur seuil de la «fonction objectif» ou encore lorsque sa valeur garde une certaine stabilité. Depuis leur apparition à la deuxième moitié du 20^{eme} siècle, les métaheuristiques ont connu un développement remarquable, comme le témoigne le nombre de publications qui est en constante croissance (POP et al. 2022 ; JOURDAN et al. 2009). Ce succès est dû à leur capacité à apporter des solutions satisfaisantes à une large gamme de problèmes d'optimisation de taille (BOISSON 2008), le tout dans des temps raisonnables (GOGNA et al. 2013). Cette caractéristique des méthodes métaheuristiques nous intéresse particulièrement dans le cadre de ces travaux.

2.4 Résolution de problèmes d'optimisation par les métaheuristiques

Encore appelées «Algorithmes inspirés par la nature[8]», les métaheuristiques sont des algorithmes qui tirent leur inspiration du fonctionnement de divers systèmes qui nous entourent, tels que la physique, la sociologie, la biologie et particulièrement le règne animal. L'idée à l'origine de la conceptualisation des métaheuristiques était de fournir à la communauté scientifique des stratégies générales de résolution de problèmes (Nesmachnow 2015).

Pour résoudre les problèmes d'optimisation auxquels ils sont soumis, ces algorithmes se basent sur un processus itératif qui permet la convergence vers une solution jugée satisfaisante. Généralement, les algorithmes partent d'une ou plusieurs propositions de solutions aléatoires qu'ils améliorent à l'aide de composants ou opérateurs spécifiques à chaque algorithme. Ces opérateurs sont pour la plupart inspirés des comportements collectifs ou d'autres mécanismes de la nature, ce qui leur vaut la dénomination «Algorithmes inspirés par la nature» (Zang et al. 2010).

Au cours des trois dernières décennies, les métaheuristiques ont connu un succès remarquable dans des domaines tels que la finance (Domínguez et al. 2017), la gestion de la production (Ferreira et al. 2008), la santé (Kaur et al. 2023), les télécommunications et les applications informatiques (Gendreau et al. 2010 ; Boussaïd et al. 2013 ; Juan et al. 2015) lorsqu'il s'agit de trouver rapidement des solutions satisfaisantes à des problèmes de grande complexité sans rester bloqués dans des optima locaux (Gogna et al. 2013 ; Elshaer et al. 2020 ; Roni et al. 2022). Pour parvenir à ce résultat, les métaheuristiques reposent sur deux concepts clés : la *diversification* et l'*intensification*. La diversification consiste à explorer l'espace de recherche pour identifier des zones qui peuvent potentiellement contenir des solutions optimales. L'intensification vient après la diversification pour exploiter les zones identifiées, soit en trouvant une solution optimale, soit en améliorant sa qualité.

2.4.1 Phases d'exécution d'une métaheuristique.

Depuis l'apparition des méthodes à métaheuristiques, un très grand nombre d'algorithmes ont été créés. Autant qu'ils sont, leur exécution peut se résumer en cinq phases principales que sont la génération, l'évaluation, l'analyse, la sélection et la mise à jour (Younis et al. 2022). Elles sont résumées par le schéma de la figure 2.5.

2.4.1.1 La génération aléatoire

Une ou plusieurs solutions sont générées et sauvegardées dans la mémoire de travail de l'algorithme. Cette phase consiste à fournir la solution ou le groupe de solutions de base, à améliorer tout au long du processus d'optimisation. C'est donc une phase cruciale et déterminante puisque la solution finale qui sera trouvée dépend fortement de la qualité de celle(s) générée(s) ici. La solution ou le groupe de solutions devra impérativement respecter toutes les contraintes du problème et couvrir une grande partie de l'espace de recherche. Lors des phases de *diversification*, la génération aléatoire peut être mise en œuvre pour proposer de nouvelles solutions.

8. Parfois on retrouve le terme «Bio-inspirés», bien que la source d'inspiration soit plus large

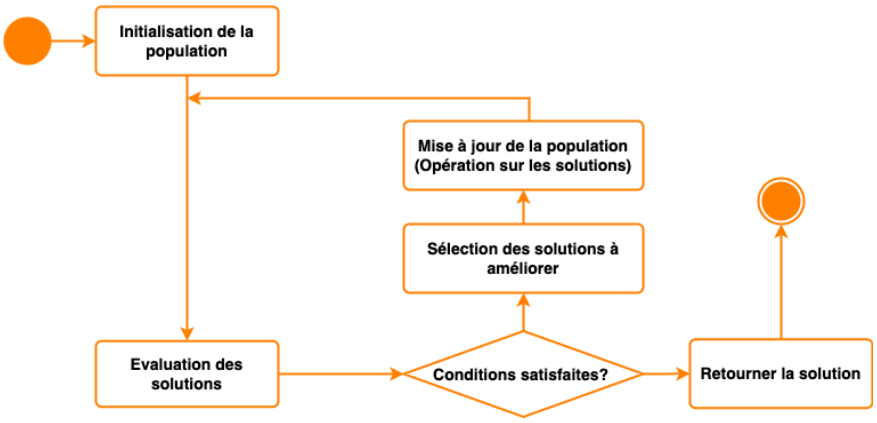


FIGURE 2.5 – Phases d'exécution d'un algorithme de métaheuristique

Un autre aspect à prendre en compte lors de la phase de génération de solutions est la représentation des solutions. En effet, selon le type de problème à résoudre et l'algorithme utilisé, il faut trouver une représentation qui permet à la fois de garder l'intégrité de la solution et d'être compatible avec les différentes opérations que l'algorithme doit effectuer (nous y reviendrons plus en détail dans la section 2.4.3).

En reprenant une fois encore l'exemple du commerçant de la section 2.1, une génération de solution possible est illustrée par la figure 2.6. Trois solutions aléatoires $x_1 = (A, C, E)$, $x_2 = (B, C, D)$ et $x_3 = (A, C, D)$ sont générées en faisant un tirage aléatoire dans l'espace de recherche formé par les conteneurs A, B, C, D et E.

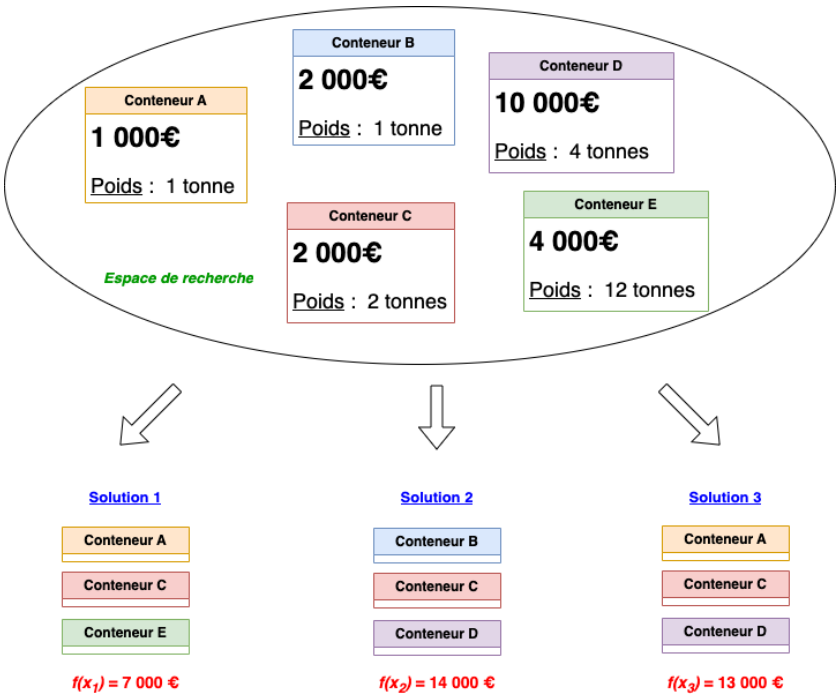


FIGURE 2.6 – Exemple de génération de solutions

2.4.1.2 L'évaluation

Comme son nom peut l'indiquer, cette phase permet d'évaluer la qualité d'une solution vis-à-vis du problème. Il s'agit d'une fonction établie sur la base de la fonction «objectif») $f : \Omega \to \mathbb{R}$ (avec Ω l'espace de recherche) telle que $\forall x \in \Omega, f(x) = y$. La valeur de y pour une solution x est appelée « fitness » ou « score », c'est-à-dire l'adaptation de la solution pour le problème. Dans le cas d'une minimisation, on cherche à trouver la valeur la plus petite possible de y, et dans le cas d'une maximisation, c'est la valeur la plus grande qui est recherchée.

La phase d'évaluation intervient à la suite de la phase de génération des solutions ou de mise à jour (à voir plus loin) pour évaluer chacune des nouvelles solutions. Une relation d'ordre est ainsi créée entre les solutions pour identifier les bonnes et les moins bonnes.

2.4.1.3 L'analyse

Une fois la « qualité » des solutions connue à la phase d'évaluation, vient ensuite la phase d'analyse. C'est une phase de délibération où deux cas sont possibles. Le premier cas est l'interruption du processus si l'une des conditions d'arrêt définies au préalable est atteinte (par exemple : le nombre maximum d'itérations est atteint ou la valeur seuil du fitness est franchie). La meilleure solution à ce stade est retournée, c'est la fin de l'exécution de l'algorithme.

Dans le second cas, lorsqu'aucune condition d'arrêt n'est atteinte, l'algorithme suit son cours et passe à la phase suivante qu'est la sélection.

2.4.1.4 La sélection

Dans cette phase, les solutions susceptibles d'être améliorées ou qui peuvent contribuer à trouver la solution optimale sont identifiées et retenues pour la suite du processus. Dans la littérature, on retrouve plusieurs techniques de sélection, dont la plus courante est la technique probabiliste. En effet, à l'aide de l'évaluation d'une solution, on détermine sa probabilité d'être retenue. Plus une solution est adaptée au problème, plus sa probabilité d'être retenue est grande (Behera 2020). Contrairement à la technique dite déterministe où seules les meilleures solutions sont retenues, la technique probabiliste permet de donner une « chance » aux solutions moins bonnes d'être améliorées ou même de participer à l'amélioration d'autres solutions en assurant une forme de diversification.

2.4.1.5 La Mise à jour

Considérée comme le cœur du processus d'exécution des métaheuristiques, la mise à jour est la phase pendant laquelle les solutions retenues lors de la sélection sont améliorées. Cette phase correspond à l'*intensification* et est propre à chaque métaheuristique. Son rôle est de faire varier les solutions existantes pour en créer de nouvelles. Ces différentes variations sont effectuées au moyen d'opérateurs qui sont aussi spécifiques à chaque algorithme. Un exemple de mise à jour d'une solution est montré à la figure 2.7, où une solution x est mise à jour pour donner une solution x' avec un score différent. Il est important de noter que pour certains algorithmes, lors de la phase de mise à jour, de nouvelles solutions peuvent être générées.

Cette opération correspond à une nouvelle diversification des solutions et permet d'éviter le blocage des algorithmes dans des optima locaux.

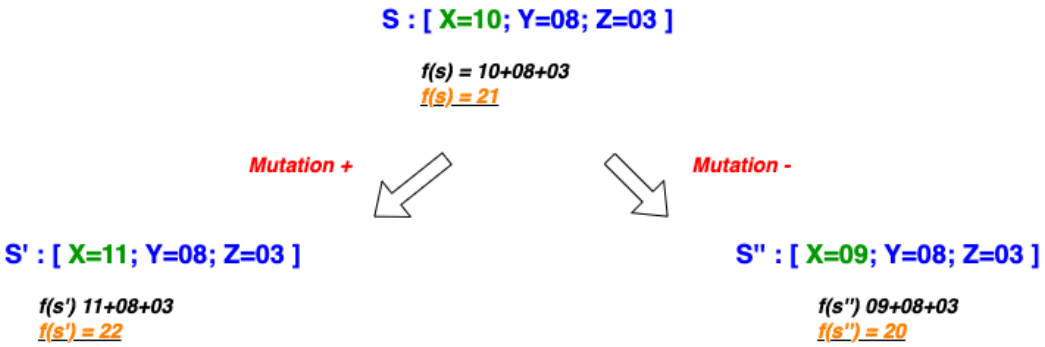


FIGURE 2.7 – Mise à jour d'une solution : exemple de mutation dans un algorithme génétique.

2.4.2 Classification des métaheuristiques

Comme la majorité des concepts que nous avons présentés jusqu'ici, il existe différentes façons de classer les algorithmes métaheuristiques. Les principaux critères utilisés se basent sur le nombre de solutions traitées, la source d'inspiration, la fonction objectif, le nombre de voisins ou structures, et l'utilisation de l'historique de recherche (GOGNA et al. 2013 ; POP et al. 2022). Un résumé de ces critères est proposé à la figure 2.8.

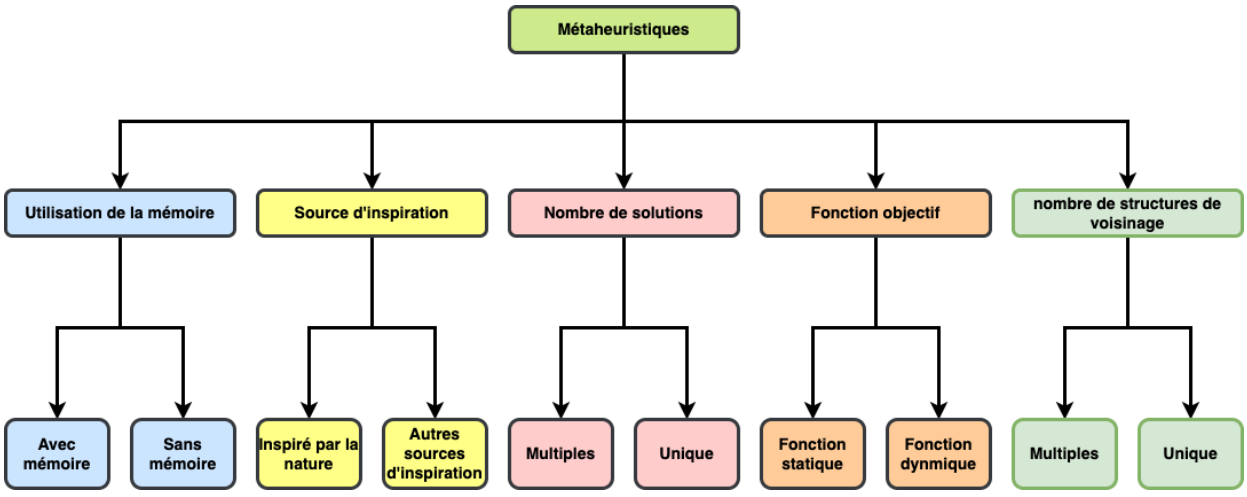


FIGURE 2.8 – Classification des algorithmes à métaheuristiques

En choisissant comme critère de classification le nombre de solutions utilisées dans l'algorithme, nous distinguons deux catégories : les métaheuristiques à solution unique ou « S métaheuristics [9] » et les métaheuristiques à multiples solutions ou « P métaheuristics [10] ».

9. S pour single, c'est-à-dire « unique »
10. P pour Population Based, c'est-à-dire « basé sur une population »

2.4.2.1 Catégorie des «S metaheuristics»

La recherche de solutions optimales avec les « S metaheuristics » [11] s'effectue avec une unique solution qui est conservée et remplacée en mémoire. Les algorithmes de cette classe partent d'une solution unique, créée aléatoirement dans la phase de génération de solution. Cette solution de base est « perturbée » pour produire d'autres solutions lors de la phase de mise à jour (Talbi 2013). Comme nous l'avons précisé précédemment, cette phase peut également voir l'apparition de nouvelles solutions. Parmi toutes les solutions générées, celle qui a le meilleur score est choisie lors de la phase de sélection pour remplacer la solution unique en mémoire. Ces méthodes sont généralement performantes à l'intensification sur des problèmes avec peu d'optima locaux.

2.4.2.2 Catégorie des «P metaheuristics»

La catégorie des « P metaheuristics » [12] regroupe les algorithmes qui traitent et gardent en mémoire plusieurs solutions réparties sur l'espace de recherche. Elle est de loin la catégorie la plus utilisée et la plus connue, étant donné le grand nombre d'algorithmes qui la constituent (Agushaka et al. 2023). Ici, bien que spécifique à chaque algorithme, la proposition de nouvelles solutions lors de la mise à jour se fait généralement par la combinaison des solutions existantes ou sur la base des informations recueillies au cours des itérations passées. Les algorithmes de cette catégorie sont particulièrement meilleurs sur la diversification et l'extraction en cas de blocage dans un optimum local.

Dans la communauté des métaheuristiques, la classification basée sur le nombre de solutions est largement utilisée pour présenter les différents algorithmes pour résoudre les problèmes. Nous faisons de même pour présenter les principaux algorithmes de métaheuristiques les plus cités dans la littérature (Omidvar et al. 2022).

2.4.3 Exemples de métaheuristiques

Le nombre d'algorithmes à métaheuristiques étant très grand et ne cessant de croître (Roni et al. 2022 ; Pop et al. 2022), cette présentation ne prétend pas être exhaustive et ne se focalise que sur les plus connus. Le tableau 2.3 fait un résumé de ces algorithmes que nous présentons dans cette section.

11. S = Single (unique)
12. P = population

TABLE 2.3 – Exemples d'algorithmes à métaheuristiques

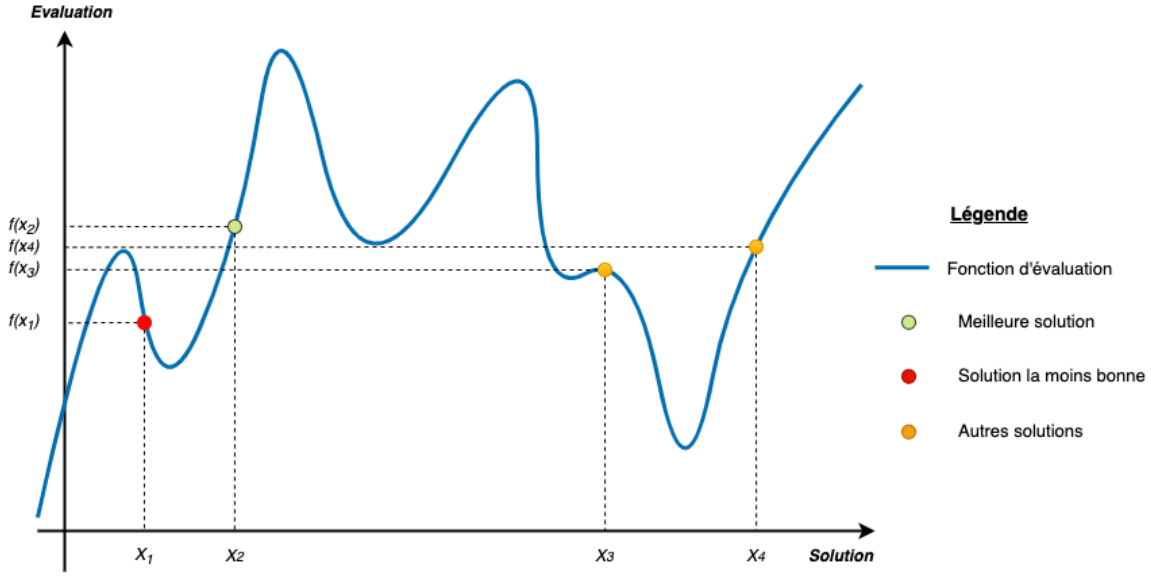
Catégorie	Algorithme	Source d'inspiration
S metaheuristics	Hill Climbing	-
	Recherche tabou	Mémoire humaine
	Recuit simulé	Métallurgie / Verrerie
P metaheuristics	Algorithme génétique	Génétique
	Algorithme immunitaire	Système immunitaire
	Recherche harmonique	Musique
	Colonie de fourmis	Colonie de fourmis

Présentons dans un premier temps ces algorithmes de la catégorie des «S metaheuristics».

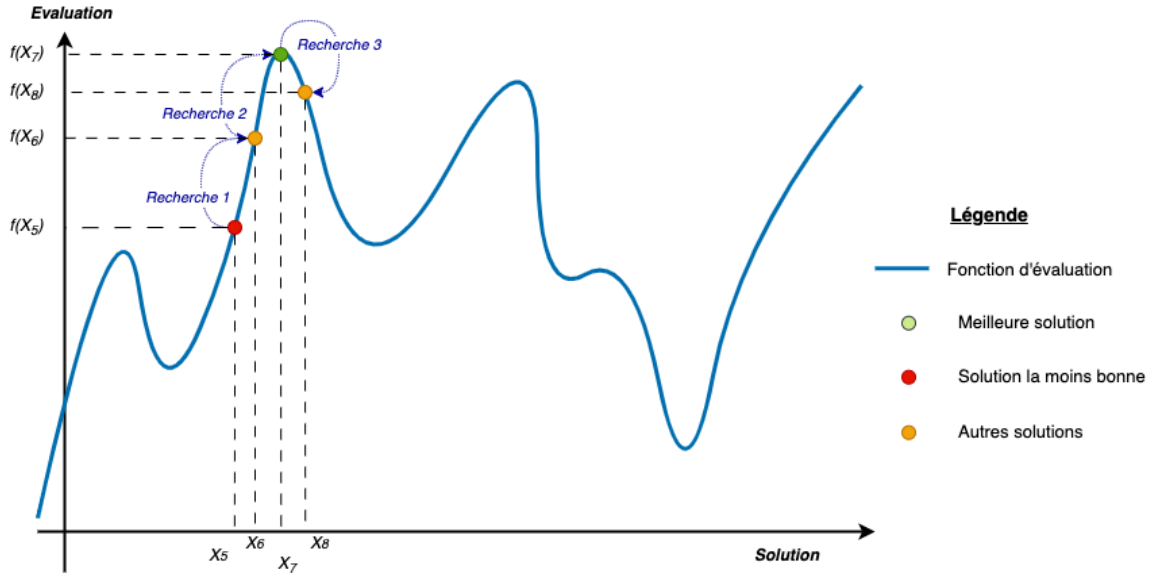
2.4.3.1 Algorithme de la recherche par descente (Hill Climbing)

La recherche par descente ou recherche locale (MLADENOVIĆ et al. 1997) est l'une des premières métaheuristiques à être implémentée pour la résolution de problèmes d'optimisation. Le principe ici est de générer de manière aléatoire plusieurs solutions à l'initialisation, les évaluer puis les trier par ordre croissant (pour les problèmes de minimisation) ou décroissant (pour les problèmes de maximisation). À la fin de ce tri, la solution qui présente le meilleur score (car elle est de la classe des «S Metaheuristics») est retenue. Ensuite, le processus itératif démarre, dans lequel des solutions voisines à la solution retenue sont générées et évaluées également. Si une meilleure solution est trouvée, elle remplace la solution retenue. L'algorithme 3 présente le processus d'optimisation et l'illustre avec la figure 2.9. À la figure 2.9a, c'est l'initialisation. Les solutions x_1, x_2, x_3 et x_4 sont aléatoirement générées puis évaluées. À cette étape, nous avons $f(x_2) > f(x_4) > f(x_3) > f(x_4)$. x_2 donc choisie puis améliorée progressivement comme le montre la figure 2.9b jusqu'à ce qu'une solution moins bonne soit trouvée (x_8 dans ce cas). Dans cette situation, la solution précédente (x_7) est alors retournée comme solution optimale.

La principale insuffisance de cet algorithme reste le risque élevé de blocage sur un optimum local. Cette limite est due au fait que la diversification n'est réalisée que lors de la phase d'initialisation à partir de laquelle dépend le résultat final.



(a) Quatre solutions générées à l'initialisation



(b) Phase d'intensification : Amélioration de la solution x_2

FIGURE 2.9 – Illustration de l'algorithme de Hill Climbing

Algorithme 3 : Algorithme de recherche par descente (Hill Climbing)

```
Données : pas
solution ← générer_solution();
évaluer(solution);
tant que NON condition_arrât faire
    solutions_générées ← générer_solutions_voisines(solution) ;
    pour chaque voisin dans solutions_générées faire
        évaluer(voisin);
        si voisin mieux que solution alors
            solution ← voisin ;
    fin
fin
retourner solution
```

Il est en effet important de prendre en compte le principe du voisinage, selon lequel les nouvelles solutions sont générées. Il s'agit du degré de perturbation à appliquer à la solution de base pour produire de nouvelles solutions. C'est le «pas» de la recherche et il constitue un paramètre de configuration à part entière de cet algorithme. Une bonne configuration de ce paramètre permet à l'algorithme d'être efficace. En effet, lorsque le «pas» est grand, il est possible de manquer la solution optimale et de se retrouver bloqué dans une boucle infinie, tandis que lorsque le «pas» est petit, le temps de recherche est plus long.

2.4.3.2 Algorithme de recherche tabou

La recherche tabou (Glover 1986) reprend la manière dont la recherche locale cherche l'optimum, avec cette fois la particularité d'utiliser explicitement l'historique de la recherche. Cet aspect permet de refaire la diversification et donc de sortir des optima locaux.

L'algorithme dispose d'une mémoire dans laquelle il enregistre les n dernières solutions trouvées. À chaque itération, l'algorithme s'interdit d'exploiter une solution tant qu'elle se trouve dans ladite mémoire, telle une mémoire à court terme, similaire à celle de l'humain dont l'algorithme s'inspire. Cette approche est particulièrement intéressante dans la recherche, car elle évite de revenir à des zones déjà explorées et même d'accepter des solutions moins bonnes que celles déjà trouvées, en espérant qu'elles pourraient conduire à un optimum global lors de l'intensification. L'algorithme 4 est celui de la recherche par tabou. La figure 2.10 applique cet algorithme pour rechercher la valeur 100. À l'itération 1, la meilleure valeur est 96, elle est ajoutée en un premier temps dans la mémoire. S'en suit la recherche du voisinage qui a conduit aux solutions 94 et 98. La solution 94 étant moins bonne que la solution courante contrairement à la solution 98 qui est retenue puis ajoutée à la mémoire à son tour. À l'itération 2, des solutions 96 et 100 sont générées aux voisines de la solution 98. La solution 96 étant déjà en mémoire, elle est rejetée et la solution 100 attendue est trouvée.

Le paramètre de configuration de l'algorithme de recherche tabou est la taille n de la mémoire, c'est-à-dire le nombre maximal de solutions récentes à enregistrer. Si n est petit, la recherche se concentrera sur de petites zones de l'espace de recherche, avec le risque de revisiter à plusieurs reprises des solutions déjà explorées (globalement). Dans le cas où n est grand, une grande partie de l'espace de recherche sera explorée, mais le temps de traitement sera rallongé (Boussaïd et al. 2013). Dans certaines variantes de l'algorithme, la valeur de

n peut être modifiée de manière dynamique pendant l'exécution (Battiti et al. 1994).

Algorithme 4 : Algorithme de recherche par tabou

```
Données : n (taille de la mémoire)
mémoire_tabou ← [ ];
solution ← generer_solution();
évaluer(solution);
ajouter(solution, mémoire_tabou);
tant que NON condition_arret faire
    solutions_générées ← générer_solutions_voisines(solution) ;
    tant que solutions_générées ∈ mémoire_tabou faire
        solutions_générées ← générer_solutions_voisines(solution) ;
    fin
    pour chaque voisin dans solutions_générées faire
        évaluer(voisin);
        si voisin mieux que solution alors
            solution ← voisin ;
        ajouter(solution, mémoire_tabou);
        suppression_FIF0_(mémoire_tabou);
    fin
fin
retourner solution
```

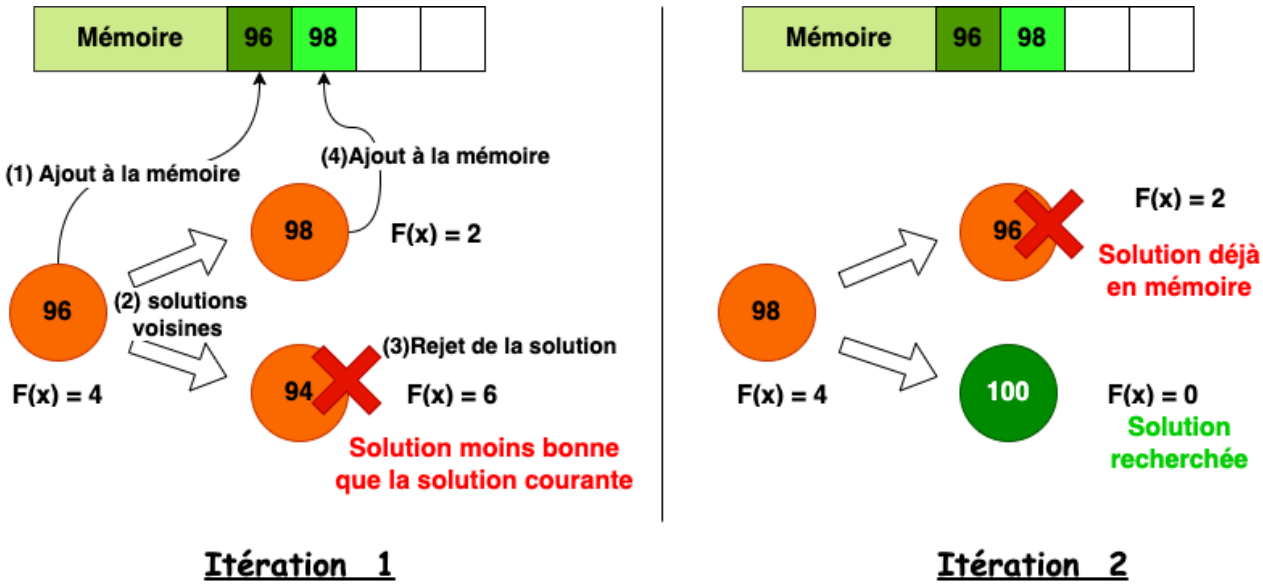


Figure 2.10 – Exemple de recherche par tabou

2.4.3.3 Algorithme du recuit simulé

L'algorithme du recuit simulé a été présenté dans (Kirkpatrick et al. 1983). Cet algorithme est inspiré du processus de refroidissement utilisé en métallurgie pour obtenir des objets à l'état solide cristallin à partir d'un matériau visqueux. En effet, lorsqu'un matériau est chauffé, les constituants du matériau sont fortement en mouvement. Lorsque le matériau est refroidi, les atomes se déplacent moins, se stabilisent et forment une structure solide. La structure solide obtenue dépend de la manière dont le matériau visqueux a été refroidi. Un refroidissement rapide conduit à une structure amorphe, tandis qu'un refroidissement lent conduit à une structure cristalline.

En transposant ce processus à la résolution d'un problème d'optimisation, l'algorithme du recuit simulé repose sur deux paramètres principaux :

— Une valeur $T^{max} \in \mathbb{R}$ considérée comme la température initiale T_{max} du matériau lorsqu'il est chauffé ;

— Une vitesse de refroidissement $\Delta T = C^{ste}$ tel que $T_2 - T_1 = \Delta T$ avec $(T_1, T_2) \in \mathbb{R}^2$ des températures à deux instants successifs.

Comme présenté dans l'algorithme 5, au démarrage, une solution est générée et évaluée, puis un processus itératif débute, au cours duquel des solutions voisines sont générées et comparées à la solution courante. La meilleure solution est alors conservée pour remplacer la solution courante. À ce stade, l'algorithme du recuit simulé peut être assimilé à une recherche par descente avec des pas progressifs jusqu'à atteindre un état de stabilité. La différence réside dans la capacité du recuit simulé à sortir des optimums locaux. En effet, lorsqu'une solution voisine moins bonne que la solution courante est générée, la probabilité p (probabilité de Boltzmann) de la conserver est calculée de la manière suivante :

$$p = e^{-\frac{f(x)-f(x')}{T}} \tag{2.2}$$

avec $f(x)$ le score de la solution courante, $f(x')$ le score de la solution voisine et T la température. A chaque itération, la température T est mise à jour.

Algorithme 5 : Algorithme du recuit simulé

```
Données : Tmax (Température initiale), ΔT (Vitesse de refroidissement)
solution ← générer_solution( );
T ← Tmax;
évaluer(solution);
tant que NON condition_arrêt faire
    voisin ← générer_solution_voisine(solution) ;
    si voisin mieux que solution alors
        solution ← voisin ;
    sinon
        probabilité ← calculer_probabilité(solution, voisin, T) ;
        solution ← choix(voisin, probabilité)      ▷ solution = voisin selon la probabilité obtenue;
    fin
    T ← T - ΔT                                     ▷ Mise à jour de la température
fin
retourner solution
```

Cet algorithme permet de faire de la diversification et de s'extraire des optimums locaux.

Cependant, il peut être coûteux en termes de temps et d'espace. Une alternative pourrait donc être les algorithmes de la catégorie des «P métaheuristiques», qui sont par définition basés sur plusieurs solutions.
Après cette présentation des «S métaheuristiques», présentons maintenant les algorithmes de la catégorie des «P métaheuristiques».

2.4.3.4 Algorithme génétique

L'algorithme génétique pourrait apparaître comme la métaheuristique basée sur les populations la plus connue, tant il est utilisé(Dokeroglu et al. 2019), et tant il existe de variantes(Katoch et al. 2021). C'est un algorithme d'inspiration biologique apparu en 1975 (Holland 1992) qui s'appuie sur la théorie de l'évolution de Charles Darwin, ce qui lui vaut d'ailleurs d'appartenir à la classe des algorithmes évolutionnaires (**petrowski_les_2014**) [13]. Selon cette théorie, les espèces vivantes subissent perpétuellement, au fil des générations, des transformations génétiques pour s'adapter à leur environnement. Seules les espèces qui réussissent à s'adapter survivent, tandis que les autres meurent. Cette analogie est reprise par l'algorithme génétique qui considère le problème d'optimisation comme l'environnement auquel les individus d'une espèce avec une population de taille n devront s'adapter en subissant des transformations au fil des générations. Cette adaptation peut s'évaluer sur la base de la fonction objectif. Comme dans le phénomène biologique, les individus avec un meilleur score seront croisés entre eux pour donner naissance à de nouveaux enfants, tandis que ceux ayant un score moins bon vont disparaître de la population et être remplacés par de nouveaux individus. Le tableau 2.4 présente l'analogie qui peut être faite entre les concepts de la génétique et des algorithmes génétiques.

Un algorithme génétique est principalement caractérisé par l'encodage de la solution, l'opérateur de sélection, l'opérateur de croisement et l'opérateur de mutation (Lambora et al. 2019).

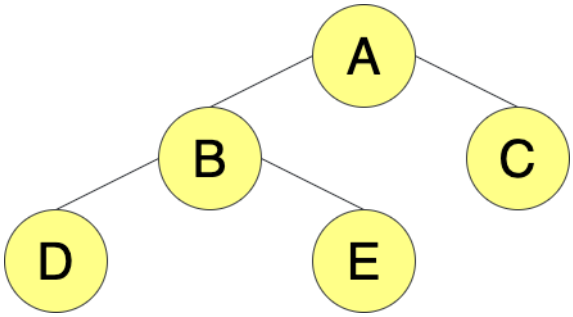
Table 2.4 – Analogie entre la génétique et la métaheuristique

Génétique	**Algorithme génétique**
Individu	Solution
Chromosome (génotypes)	Ensembles des variables formant une solution
Gêne	Variable de décision
Adaptation	Score (Fitness)
Sélection naturelle	Diversification
Mutations	Diversification
Croisement	Intensification

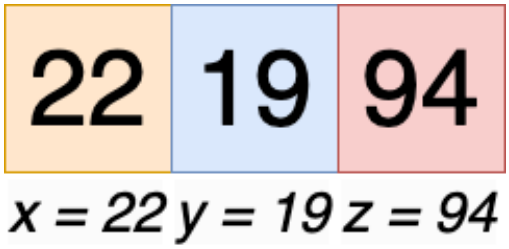
— ***Représentation de la solution*** : Tout au long du processus d'optimisation, les solutions sont appelées à être manipulées et à subir des transformations qui contribuent à leur amélioration vers l'optimum. Pour cela, les solutions doivent avoir une représentation structurelle qui permet de subir ces transformations sans «alourdir» le processus [14].

13. Si l'on utilise comme critère de classification le mode de fonctionnement de l'algorithme
14. Exemple : Éviter la perte d'informations, éviter d'explorer à nouveau des espaces déjà exploités

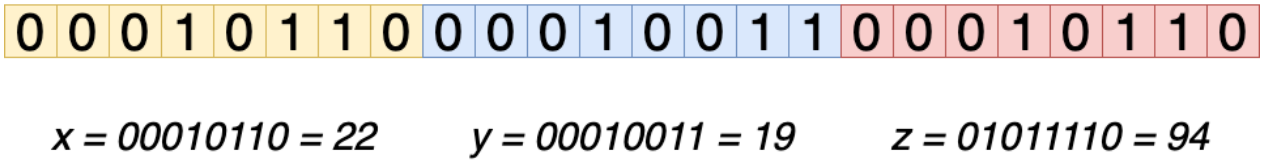
Dans la littérature, nous pouvons également trouver plusieurs façons de représenter les solutions. Le choix entre l'une ou l'autre dépend des variables de décision (et de leur type). Néanmoins, les plus couramment rencontrées sont la représentation sous forme réelle (figure 2.11b), la représentation sous forme binaire (figure 2.11c) et la représentation sous forme d'un arbre (figure 2.11a).
Les solutions représentées comme telles se prêtent bien aux différentes transformations propres à l'algorithme génétique, mais ne peuvent pas être évaluées dans l'état. À cet effet, il est généralement implémenté un décodeur associé à la représentation de la solution pour partir de la forme encodée, vers une forme directement utilisable par la fonction d'évaluation.



(a) Représentation sous forme d'arbre d'une solution avec 5 variables de décision A, B, C, D et E



(b) Représentation en valeur réelle d'une solution avec 3 variables de décision x, y et z



(c) Représentation en valeur binaire d'une solution avec 3 variables de décision x, y et z

FIGURE 2.11 – Exemples de représentations de solution pour un algorithme génétique

— ***Opérateur de sélection*** : Dans le monde du vivant, source d'inspiration de l'algorithme génétique, la survie d'une espèce dépend de la qualité des individus qui se reproduisent. Il en est de même dans la recherche de solutions par l'algorithme, qui dépend également du score des «parents» se reproduisant entre eux. Dans la nature, certaines espèces se reproduisent uniquement avec les meilleurs géniteurs de leur population (notamment chez les primates), tandis que d'autres espèces se reproduisent de façon totalement aléatoire (comme certains poissons qui dispersent leurs gamètes dans l'environnement, rendant ainsi la fécondation aléatoire). Tout comme dans la nature, il peut aussi exister différentes façons de sélectionner les solutions à faire évoluer. La plus connue est la technique de la «Roulette» (GOLDBERG 1989), qui associe à chaque individu de la population une probabilité d'être sélectionné, proportionnelle à leur score.

Cette probabilité est calculée selon l'équation 2.3, avec $P(x)$ la probabilité de l'individu

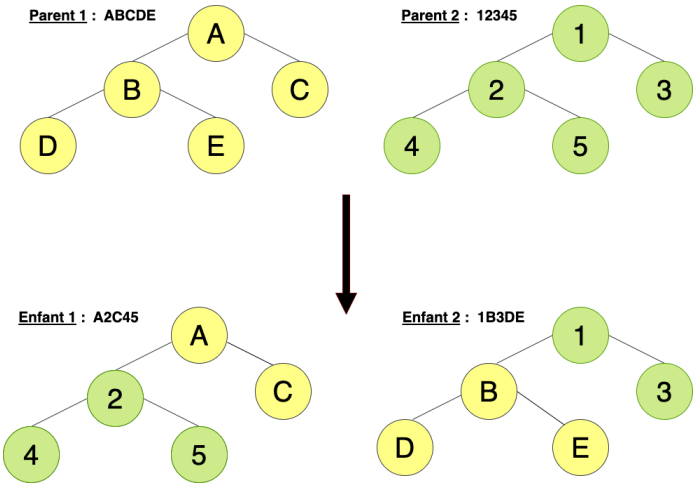
x d'être sélectionné, x_i la solution à la position i de la population, n la taille de la population et f la fonction d'évaluation.

$$P(x) = \frac{f(x)}{\sum_{i=1}^{n} f(x_i)} \tag{2.3}$$

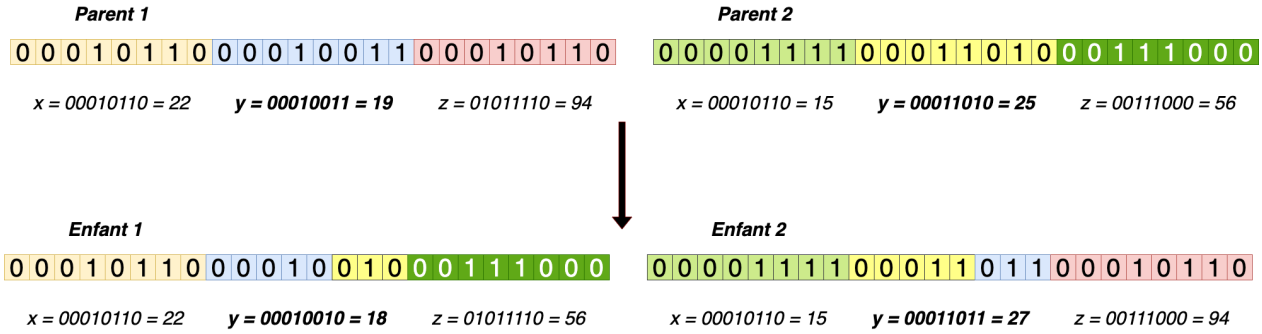
Par cette méthode, les solutions ayant les meilleurs scores ont plus de chance d'être sélectionnées. Elle permet également de garder une certaine diversité de solutions.

— ***Opérateur de croisement*** : L'opérateur de croisement propose de nouvelles solutions à partir des solutions sélectionnées dans la phase précédente. C'est donc l'opérateur qui permet aux couples de «parents» d'échanger entre eux des informations de leurs génomes, c'est-à-dire les variables de décision. Pour chaque parent sélectionné, des points de croisement sont choisis. Ces points de croisement représentent les positions où les segments seront recomposés pour créer de nouveaux «enfants», comme illustré dans la figure 2.12. Cette figure montre 2 exemples : un premier avec une représentation sous la forme d'un arbre et le second sous la forme binaire. Le croisement de la figure 2.12a concerne deux solutions de bases (les parents 1 et 2) avec respectivement comme valeur $[ABCDE]$ et [12345]. En choisissant les points B et 2 comme points de croisement, nous obtenons deux nouvelles solutions (les enfants 1 et 2) avec des génotypes différents des parents ($[A2C45]$ pour l'un et $[1B3DE]$ pour l'autre). Quant au croisement de la figure 2.12b, le procédé est similaire. Nous sommes partis de deux solutions «parents» ayant respectivement $[x = 22, y = 19, z = 94]$ et $[x = 15, y = 25, z = 56]$ pour obtenir deux solutions «enfants» différentes de génotype $[x = 22, y = 18, z = 56]$ et $[x = 15, y = 27, z = 94]$.

— ***Opérateur de mutation*** : Les mutations génétiques dans le monde vivant sont souvent des «erreurs» qui s'introduisent dans la constitution du génome des «enfants». Ce principe est également repris par l'algorithme génétique et peut être considéré comme une diversification. En effet, au fil des croisements, les individus de la population tendent à avoir la même constitution et donc à se bloquer dans un optimum local. En introduisant quelques perturbations par ce phénomène de mutation, on intègre dans la population de nouveaux individus avec un génotype différent, et donc un score différent. Ce type d'individu peut être de meilleure qualité ou non. Néanmoins, au-delà de diversifier la population, il pourrait participer, au fil des croisements, à créer un individu de meilleure qualité. Des exemples de mutations sont présentés dans la figure 2.13. Il est important de préciser que la mutation d'un «enfant» n'est pas systématique. Elle se fait au moyen d'une probabilité P_c généralement faible.



(a) Croisement de deux solutions représentées sous forme d'arbres



(b) Croisement de deux solutions représentées sous forme binaire

FIGURE 2.12 – Croisements avec différentes représentations de solution

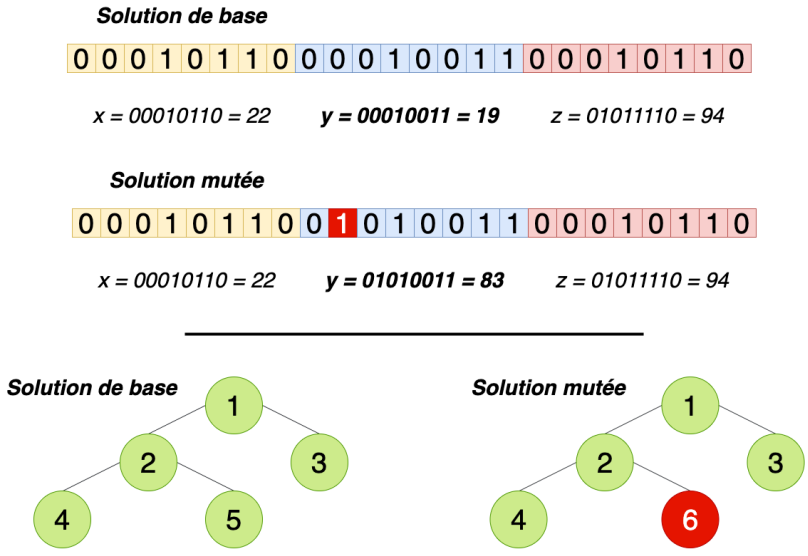


FIGURE 2.13 – Mutations avec les deux représentations de solution (binaire et arbres)

Tous ces opérateurs mis bout à bout permettent de trouver des solutions pour quasiment tout type de problèmes d'optimisation comme décrit par l'algorithme 6. Comme on peut le comprendre par l'algorithme 6, les paramètres tels que la taille n de la population et la probabilité de mutation P_c influencent fortement l'atteinte ou non d'un optimum global.

Algorithme 6 : Algorithme génétique

```
Données : n (Taille de la population), n+(Nombre de nouveau), nx(Nombre de croisement)
population ← générer_solution(n)                    ▷ Générer n solutions aléatoire;
évaluer(population)                    ▷ Évaluer chaque élément de la population;
tant que NON conditon_arrêt faire
    pour i ← 1 ; à i < nx ; faire
        parent1, parent2 ← choisir_parent(population) ;
        enfant1, enfant2 ← croiser(parent1, parent2) ;
        évaluer(enfant1, enfant2) ;
        ajouter(enfant1, enfant2, population) ;
        p1 ← calculer_probabilité_mutation(enfant1) ;
        mutation1 ← muter(enfant1, p1)              ▷ Mutation selon la probabilité p1;
        si mutation1 réussie alors
            évaluer(mutation1) ;
            ajouter(mutation1, population) ;
        p2 ← calculer_probabilité_mutation(enfant2) ;
        mutation2 ← muter(enfant2, p2)              ▷ Mutation selon la probabilité p2;
        si mutation2 réussie alors
            évaluer(mutation2) ;
            ajouter(mutation2, population) ;
    fin
    uniformiser(population)                ▷ Réduire la taille de la population;
    nouveaux ← générer_solution(n+) ;
    ajouter(nouveaux, population) ;
    trier(population) ;
    solution ← meilleur(population) ;
fin
retourner solution
```

2.4.3.5 Algorithme des systèmes immunitaires artificiels

En se référant à la classification proposée, l'algorithme des systèmes immunitaires artificiels (Farmer et al. 1986) se retrouve dans la classe des algorithmes bio-inspirés. Il s'inspire du système immunitaire des organismes dans leur manière de s'adapter et de faire face à l'agression d'un corps étranger. En effet, les organismes sont constamment agressés par des agents pathogènes. Pour contrer ce phénomène, les animaux vertébrés en particulier sont dotés d'un système immunitaire capable d'identifier et éliminer l'agent pathogène. Ce phénomène biologique a inspiré divers algorithmes pour la résolution de problèmes d'optimisation, dont celui du clonage par sélection (De Castro et al. 2001). Cet algorithme traduit la capacité des systèmes immunitaires à créer une très grande variété de cellules et de molécules susceptibles de reconnaître et d'éliminer spécifiquement un nombre important d'envahisseurs étrangers. L'analogie qui est faite entre le système biologique et la résolution d'un problème d'optimisation est présentée dans le tableau 2.5.

TABLE 2.5 – Analogie entre système immunitaire biologique et système immunitaire artificiel

Système immunitaire biologique	**Système immunitaire artificiel**
Système immunitaire	Ensembles des solutions
Anticorps	Solution
Affinité	Score (Fitness)
Clonnage	Intensification
Mutation	Diversification

En se basant sur cette analogie, le processus d'optimisation à proprement dit est présenté par l'algorithme 7 : Au début du processus d'optimisation, un nombre n d'anticorps est aléatoirement généré. Cette génération faite de manière aléatoire correspond aux anticorps dans la « mémoire » d'un système immunitaire et du côté de l'optimisation, il s'agit de l'initiation. Chaque anticorps généré est évalué pour mesurer son affinité au problème. En fonction de celle-ci, les anticorps avec les meilleurs scores sont clonés en grandes quantités tandis que ceux qui sont « moins bons » sont très peu ou pas du tout clonés. Les clones obtenus subiront à leur tour une série de mutations, elle aussi proportionnellement à l'adaptation. Cette étape correspond à la phase d'intensification des métaheuristiques. Lorsque de fortes perturbations sont appliquées, on parle d'hyper-mutation. Pour assurer un certain niveau de diversité, de nouveaux anticorps peuvent être générés aléatoirement et évalués de la même manière. L'exemple de la figure 2.14 illustre l'utilisation de l'algorithme pour la résolution d'un problème. La solution recherchée est le couple (x, y) tel que $4x - 2y = 0$. La fonction d'évaluation ou d'adaptation est $f(x, y) = |(4x - 2y)|$. Il s'agira donc de minimiser $f(x, y)$.

Algorithme 7 : Algorithme des systèmes immunitaires artificiels de clonage par sélection

```
Données : n (nombre d'anticorps dans le système immunitaire initiale)
système_immunitaire = créer_anticorps_aléatoirement(n) ;
évaluer(système_immunitaire)                    ▷ Évaluation de chaque solution ;
tant que NON condition_arrêt faire
    pour chaque anticorps dans systême_immunitaire faire
        clonnes= clonner(anticorps) ;
        pour chaque clonne dans clonnes faire
            clone = muter(clonne)                          ▷ Ou Hyper-muter ;
            évaluer(clone);
            ajouter(clone, systeme_immunitaire);
        fin
    fin
    uniformiser(système_immunitaire) ;
    nouveaux ← créer_anticorps_aléatoirement(n+) ;
    ajouter(nouveaux, système_immunitaire) ;
    trier(système_immunitaire) ;
    solution ← meilleur(système_immunitaire) ;
fin
retourner solution
```

Le succès de l'algorithme des systèmes immunitaires artificiels de clonage par sélection repose évidemment sur l'efficacité de l'opérateur de mutation, mais aussi sur la taille n de la population. Ce facteur taille peut parfois se révéler très coûteux, car plus n est grand, plus il faut d'espaces pour stocker les solutions et plus d'opérations sont à effectuer. Considérons une optimisation à l'aide de l'algorithme des systèmes immunitaires artificiels de clonage par sélection avec une taille $n = 10$ et supposons également qu'en moyenne deux clones sont créés pour chaque anticorps. À chaque itération, on se retrouve donc à évaluer et traiter 20 nouveaux anticorps, ce qui pourrait affecter considérablement l'efficience de l'algorithme. Un autre aspect qui affecte les performances de l'algorithme est que lorsque n est grand, cela accroît la probabilité de créer au sein de la population, des clones similaires ou déjà traités dans les itérations passées.

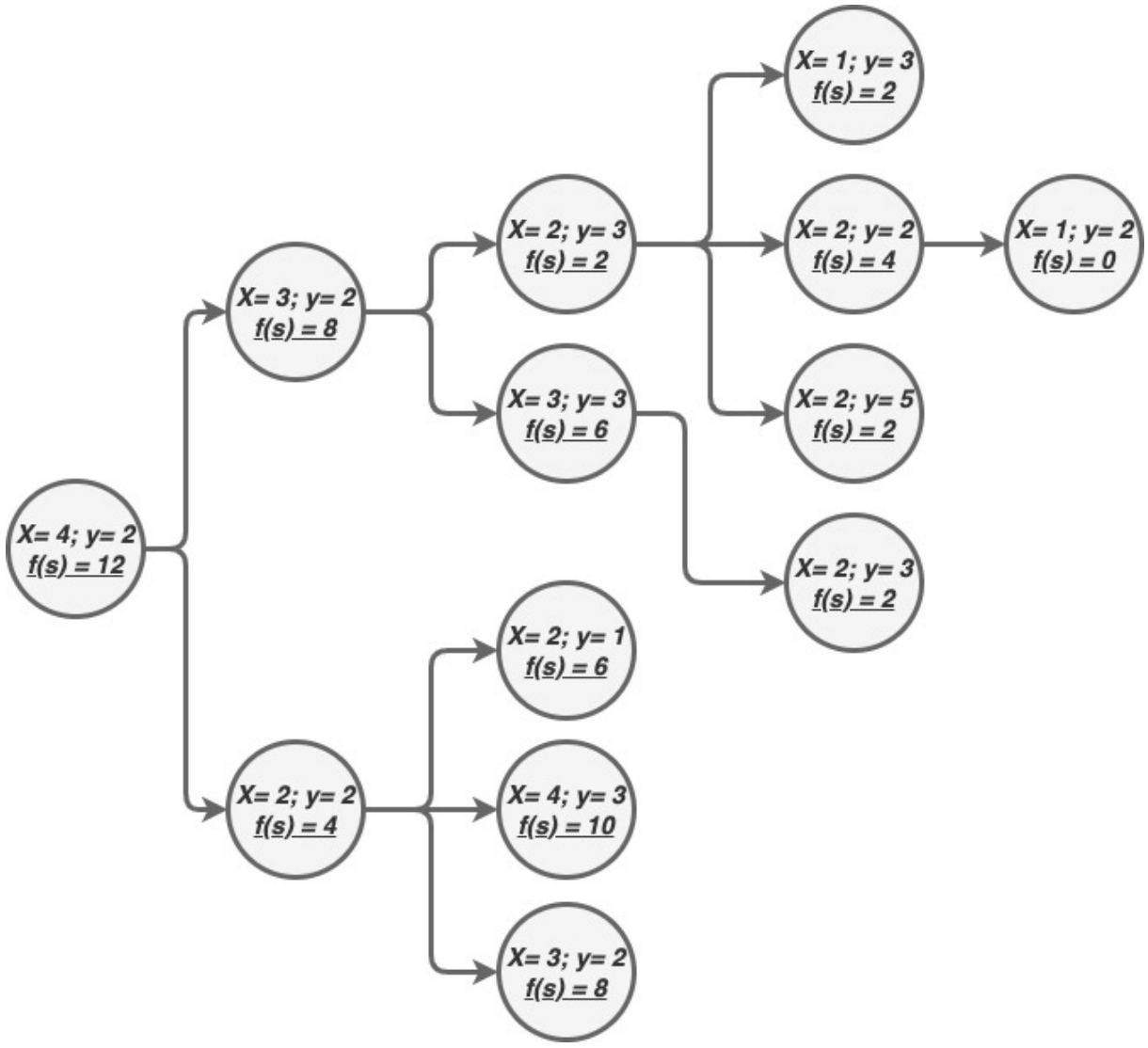


FIGURE 2.14 – Mutations avec différentes représentations de solution

2.4.3.6 Algorithme de colonie de fourmis

Le règne animal, en particulier la famille des insectes, est incontestablement une grande source d'inspiration pour les ingénieurs et les chercheurs. Il est très surprenant de voir ces êtres apparemment « faibles », « dépourvus » d'intelligence, faire preuve d'une organisation remarquable d'où émerge une intelligence collective. Ces différentes façons de faire sont fréquemment empruntées par l'humain pour résoudre des problèmes qui lui sont spécifiques. C'est d'ailleurs pour cela qu'on parle de biomimétisme [15], un concept sur lequel sont basées plusieurs métaheuristiques.

Le fonctionnement des colonies de fourmis va inspirer l'algorithme d'optimisation qui porte le même nom (Dorigo et al. 1996). En effet, pour chercher une source de nourriture, les fourmis de la colonie explorent divers chemins, généralement pris aléatoirement. Lors de cette exploration, chaque fourmi dépose sur son chemin un marqueur chimique que sont les phéromones. La première fourmi qui trouve une source de nourriture et revient au nid est donc logiquement celle qui a trouvé le chemin le plus court entre le nid et la source de nourriture. La figure 2.15 illustre par un exemple simple ce processus. Le tableau fait le bilan de la quantité de phéromones disponible sur chaque portion du chemin de cet exemple.

Les fourmis F_1 et F_2 entament leur exploration et quittent ensemble le nid N à un instant t_0. À t_1, F_1 se trouve au point A et F_2 se trouve au point C. Chacune y dépose la même quantité de phéromones, comme indiqué par le tableau 2.6. À t_2, F_2 se trouve au point S, la source de nourriture, pendant que F_1 se trouve encore entre le point A et le point B. F_2 peut donc se retourner au nid pendant que F_1 continue son exploration. À t_3, F_1 atteint le point B pendant que F_2 est sur le chemin du retour au point C, en renforçant en même temps la quantité de phéromones sur cette portion. À t_4, F_1 atteint enfin le point S et F_2 est toujours sur le chemin du retour entre le point C et le nid N. Finalement, à t_5, F_2 arrive au nid pendant que F_1 entame son chemin de retour et se trouve au point B. De facto et par répétition, le chemin le plus chargé en phéromones est le plus court. Ces phéromones serviront de stimuli aux autres fourmis qui quittent le nid, dans le choix du chemin à emprunter.

Table 2.6 – Quantités de phéromones sur le chemin

Portion de chemin	t_0	t_1	t_2	t_3	t_4	t_5
N-A	0	1	1	1	1	1
A-X	0	0	1	1	1	1
X-B	0	0	0	1	1	1
B-S	0	0	0	0	1	2
N-Y	0	1	1	1	1	2
Y-C	0	1	1	1	2	2
C-S	0	0	1	2	2	2

Ce principe est repris pour proposer un algorithme qui résout les problèmes d'optimisation. Le tableau 2.7 présente l'analogie entre les colonies des fourmis et l'algorithme qu'il a inspiré. Il découle de nature que cet algorithme est adapté pour la résolution des problèmes

15. Le biomimétisme consiste à s'inspirer des propriétés essentielles (par exemple des formes, compositions, processus, interactions) d'un ou plusieurs systèmes biologiques, pour mettre au point des procédés et des organisations permettant un développement durable des sociétés. https://www.ecologie.gouv.fr/biomimetisme

où il est recherché les plus courts chemins, mais il peut bien sûr résoudre d'autres types de problèmes en procédant à des adaptations. Ces adaptations concernent généralement la méthode d'exploration des chemins possibles.

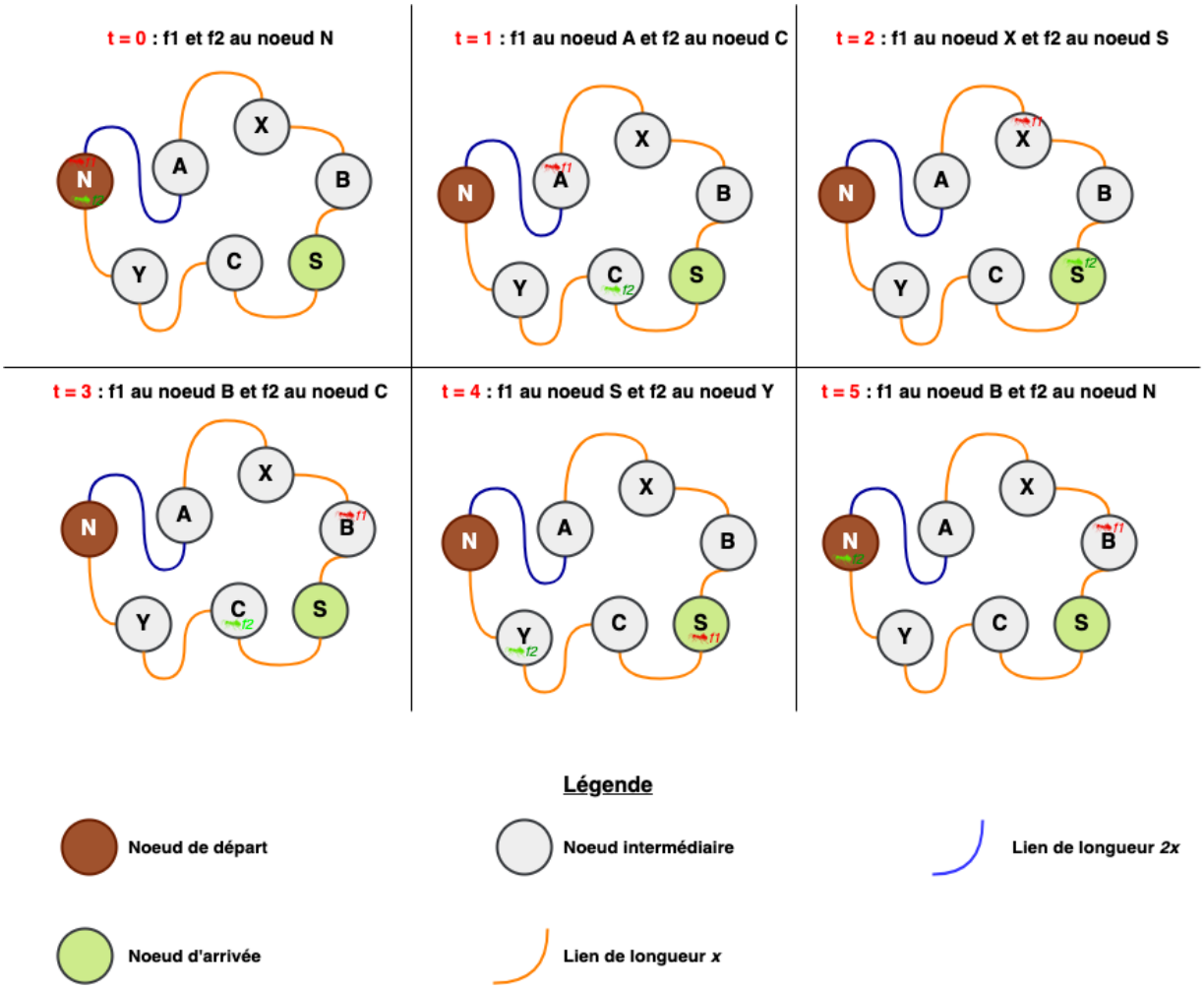


FIGURE 2.15 – Exemple de parcours de chemins par des fourmis

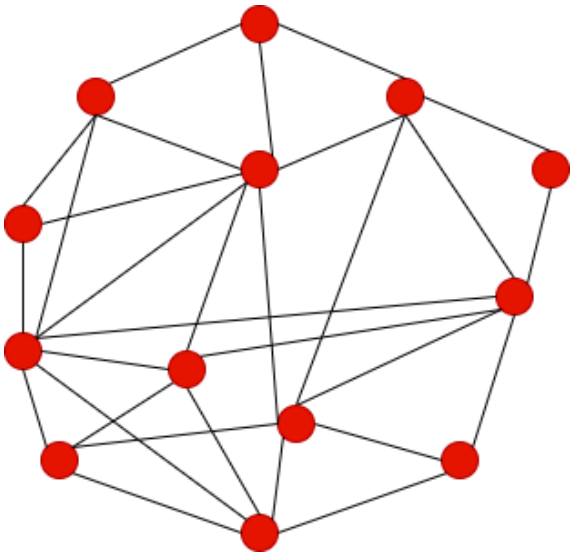
TABLE 2.7 – Analogie entre une colonie de fourmis et l'algorithme de même nom

Colonie de fourmis	**Algorithme d'optimisation**
Environnement	Espace de recherche
Nid	Nœud de démarrage
Source de nourriture	Nœud terminal
Phéromones	poids des chemins
Chemin parcouru	Solution
Coût du chemin	Fitness

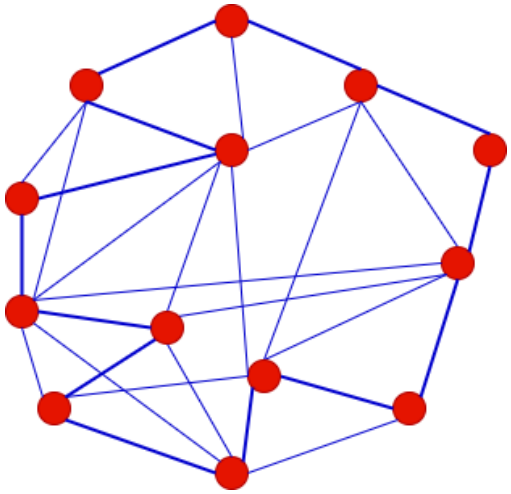
Le principe de cet algorithme décrit dans l'algorithme 8, est de créer des fourmis artificielles qui devront explorer un espace de recherche dans lequel ils seront placés. Généralement, cet espace est modélisé par un graphe où les nœuds (le nid, les nœuds intermédiaires et les nœuds terminaux) représentent les étapes d'une solution. Ainsi, au départ, une quantité n de fourmis artificielles explore l'espace de recherche en passant de nœud en nœud et en mémorisant ces étapes, jusqu'à atteindre un nœud terminal. Au nœud terminal, la solution qui découle du parcours est construite (en se référant aux étapes mémorisées) puis évaluée. Proportionnellement à la qualité de la solution obtenue, la quantité de phéromones à déposer sur le chemin est déduite et associée à l'arc du graphe, sous la forme d'un « poids ». Il s'agit là de la diversification. À ce stade, les portions de chemin ayant conduit à de bons scores ont des poids plus grands que les autres. Ce poids permet aux autres fourmis de choisir les chemins à explorer, et donc les solutions à intensifier. En effet, à chaque nœud, la fourmi artificielle effectue un choix sur le chemin à suivre. Le poids du nœud agit comme stimulus pour les autres fourmis dans le choix d'un chemin ou d'un autre. Ce choix de chemin peut se faire au niveau supérieur, c'est-à-dire en choisissant celui qui a le plus de phéromones, ou de manière probabiliste selon la formule :

$$P(x) = \frac{(\rho_x)}{\sum_{i=1}^{n} \rho_i} \tag{2.4}$$

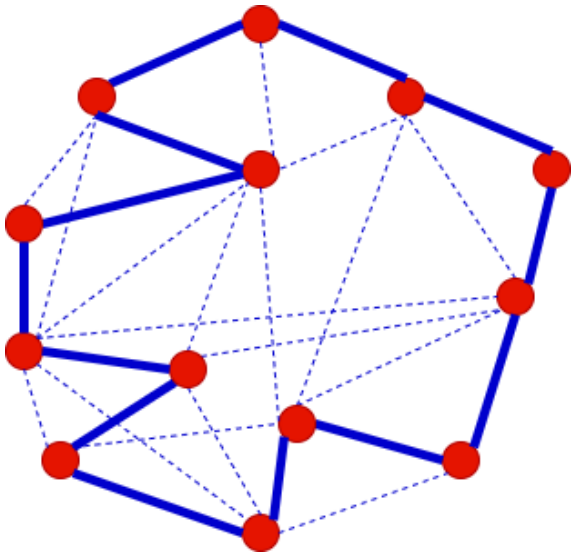
où $P(x)$ est la probabilité du chemin x avec une quantité de phéromones ρ_x d'être choisi parmi les n chemins qui partent du nœud. Le choix probabiliste est tout de même recommandé pour ne pas rester bloqué dans un optimum local, mais aussi pour permettre d'explorer les chemins les moins probables. Un mécanisme d'évaporation des phéromones au fil des itérations peut également être mis en œuvre pour inhiber encore davantage les chemins les moins probables. Ce mécanisme se matérialise par la décrémentation du poids des chemins. La figure 2.16 illustre la résolution d'un problème d'optimisation du voyageur de commerce en utilisant l'algorithme de colonie de fourmis. L'état initial est représenté par le graphe de la figure 2.16a. À ce stade, aucun chemin n'est parcouru ni aucune phéromone n'est déposée. À la figure 2.16b nous avons un stade intermédiaire où les fourmis ont parcouru déjà le graphe. Les chemins les plus explorés sont beaucoup plus chargés en phéromones. Enfin à la figure 2.16c qui est l'état final, le chemin optimal qui se dégage est formé par les portions les plus chargées en phéromones.



(a) État initial : Graphe représentation les chemins qui lient chaque point



(b) Étape intermédiaire : Les fourmis explorent différents chemins. Chaque fourmi dépose une quantité de phéromones proportionnelle à la qualité du chemin parcouru.



(c) État final : Le meilleur chemin se dessine par une grande quantité de phéromones. L'évaporation inhibe les mauvais chemins.

FIGURE 2.16 – Résolution d'un problème du voyageur de commerce par l'algorithme de colonie des fourmis

Algorithme 8 : Algorithme de colonie des fourmis

```
Données : n (nombre de fourmis dans la colonie)
graphe ← Initialiser_graphe();
▷ Si le graphe est fourni, il directement utilisé, si non il est généré à partir de l'espace de recherche ;
tant que NON condition_arrêt faire
    chemins ← [ ];
    pour chaque fourmi dans la colonie (n fourmis au total) faire
        chemin_fourmi ← [ ];
        noeud = noeud_de_départ() ;
        ajouter(noeud, chemin_fourmi)                    ▷ Ajouter le noeud au chemin;
        tant que noeud n'est pas un noeud terminal faire
            noeud = choisir_noeud_suivant(noeud, graphe)   ▷ selon les phéromones et probabilité ;
            ajouter(noeud, chemin_fourmi) ;
        fin
        ajouter(chemin_fourmi, chemins);
    fin
    pour chaque chemin dans chemins faire
        solution ← construire_solution(chemin)   ▷ Construire la solution correspondante au chemin;
        score ← évaluer(solution) ;
        mise_a_jour(graphe, chemin, score)      ▷ Quantité de phéromones à ajouter selon le score;
    fin
    évaporation_phéromones() ;
    solution_finale ← meilleur(graphe) ;
fin
retourner solution_finale
```

2.4.3.7 Algorithme de la Recherche Harmonique

Les algorithmes d'optimisation par métaheuristiques ne s'inspirent pas seulement du monde animal, mais parfois et surtout ces dernières années, des algorithmes s'inspirent de plus en plus des processus propres à l'humain (Rai et al. 2022 ; Dehghani et al. 2022). L'algorithme de la recherche harmonique en est le parfait exemple. Il tire son origine de la façon dont les musiciens, et particulièrement les musiciens de jazz, improvisent rapidement une musique en variant leur façon de jouer, jusqu'à obtenir une bonne harmonie d'un point de vue esthétique (Geem et al. 2001 ; Geem 2006). Dans la réalité, lorsque des musiciens jouant différents instruments doivent jouer ensemble dans un orchestre, ils doivent s'accorder mutuellement de manière à ce qu'ils jouent une mélodie harmonieuse appréciée par le public, bien que chaque musicien ait sa façon à lui de jouer de son instrument. Ils vont donc devoir puiser dans leur mémoire des portions de musique puis les varier afin de créer l'harmonie avec les autres.

En appliquant ce processus aux métaheuristiques, l'analogie du tableau 2.8 peut être faite. Cette analogie conduisant à l'algorithme 9 de la recherche harmonique a été proposée dans (Geem et al. 2001). Au début, l'algorithme est initialisé avec les paramètres suivants :

— n : la taille de la mémoire, correspondant au nombre d'harmoniques ;
— p_s et p_{aj}, $\in ([0,1])^2$, des probabilités dites respectivement de «sélection» et d'«ajustement» ;
— $t_{aj} \in]0, 100]$ un pourcentage.

L'exécution à proprement dite commence par la génération aléatoire de n harmoniques évaluées et stockées dans la mémoire tel que décrite par la matrice de l'équation 2.5.

Il s'agit là bien évidemment de la phase de diversification. Si les harmoniques générées ne permettent pas d'atteindre une condition d'arrêt, il s'en suit le processus d'amélioration des harmoniques, c'est-à-dire l'intensification.

$$\begin{bmatrix} x_1^1 & x_1^2 & ... & x_1^m \\ x_2^1 & x_2^2 & ... & x_3^m \\ ... & ... & ... & ... \\ x_n^m & x_n^m & ... & x_n^m \end{bmatrix} \tag{2.5}$$

avec m le nombre de variables de décisions.

TABLE 2.8 – Analogie entre une performance musicale et l'optimisation

Performance musicale	**Algorithme d'optimisation**
Harmoniques possibles	Espace de recherche
Harmonique	Solution
Sonorité jouée par un musicien / instrument	variable de décision
Note	valeur d'une variable de décision
Improvisation	Perturbation d'une solution

En utilisant une distribution de probabilité uniforme, un nombre x compris entre $[0, 1]$ est généré puis comparé à p_s, une probabilité dite de sélection. La valeur de x obtenue définit le procédé par lequel de nouvelles harmoniques sont créées :

— **Si** $x \leq p_s$, alors les nouvelles harmoniques sont créées à partir de celles déjà présentes dans la matrice de mémoire en opérant des choix aléatoires. Considérons le cas de la matrice de mémoire de l'équation 2.6.

$$\begin{bmatrix} A & E & W \\ B & F & X \\ C & G & Y \\ D & H & Z \end{bmatrix} \tag{2.6}$$

Les harmoniques $[BHY]$; $[CFX]$ et $[DEW]$ sont des exemples qui peuvent être obtenus aléatoirement. Les nouvelles harmoniques ainsi obtenues sont, elles aussi, évaluées à leur tour. Celles qui présentent un bon score sont insérées dans la matrice de mémoire pour remplacer l'harmonique la moins bonne. Ce que l'on peut remarquer à travers ce procédé, c'est que les séquences de l'harmonique optimale qui peuvent être obtenues se trouvent évidemment dans la matrice de mémoire depuis sa création. Afin d'éviter un blocage dans cet optimum (qui peut être local), des perturbations peuvent être effectuées selon une probabilité d'ajustement définie par p_{aj}. Les variables de décision sont alors modifiées par un taux d'ajustement de $t_{aj}\%$ en plus ou en moins.

— **Sinon si** $x > p_s$, alors la matrice de mémoire n'est pas considérée. La nouvelle harmonique est générée aléatoirement, renforçant ainsi la diversification. De même que dans le cas précédent, si la nouvelle harmonique présente un bon score, elle remplace la moins bonne de la matrice de mémoire.

Algorithme 9 : Algorithme de la recherche harmonique

```
Données : n (taille de la mémoire), sp_s (probabilité de sélection), p_aj (probabilité d'ajustement),
          t_aj taux d'ajustement
matrice_de_mémoire ← [ ];
pour i allant de 0 à n faire
    harmonique ← générer_aléatoirement_harmonique( ) ;
    évaluer(harmonique);
    ajouter(harmonique, matrice_de_mémoire) ;
fin
tant que NON conditon_arret faire
    x ← generer_un_nombre_entre(0, 1) ;
    si x ≤ p_s alors
        nouvelle_harmonique ← créer_harmonique(matrice_de_mémoire) ;
        y ← générer_un_nombre_entre(0, 1) ;
        si y ≤ p_aj alors
            nouvelle_harmonique ← appliquer_perturbations(nouvelle_harmonique, t_aj);
    sinon
        nouvelle_harmonique ← générer_aléatoirement_harmonique( ) ;
    fin
    évaluer(nouvelle_harmonique);
    si nouvelle_harmonique a un bon score alors
        mettre_a_jour(matrice_de_mémoire, nouvelle_harmonique);
    trier(matrice_de_mémoire) ;
    solution ← meilleur(matrice_de_mémoire) ;
fin
retourner solution
```

Conclusion

À travers ce chapitre, nous avons exploré le domaine de l'optimisation et de la résolution de problèmes en mettant l'accent sur les métaheuristiques, des méthodes qui offrent des approches flexibles et efficaces pour trouver des solutions de qualité dans des espaces de recherche vastes et difficiles à explorer. L'une des principales forces des métaheuristiques réside dans leur capacité à échapper aux pièges des optimums locaux et à trouver des solutions proches de l'optimum global dans des temps raisonnables. Ces algorithmes sont capables d'explorer rapidement l'espace de recherche tout en évitant de rester bloqués dans des solutions sous-optimales. De plus, leur aspect générique permet de les appliquer à une large gamme de domaines et de situations, des problèmes d'optimisation combinatoire aux tâches d'optimisation continues. En explorant les métaheuristiques, nous avons trouvé la possibilité intéressante de les utiliser pour la résolution des problèmes d'optimisation qui concouraient à la modélisation du comportement des animaux.

Deuxième partie

ANIMETA - Une interface pour les métaheuristiques au service de la modélisation de comportements d'animaux

CHAPITRE 3

Proposition d'un modèle générique de comportement animal

Sommaire

Introduction

Ce chapitre s'inscrit dans la continuité des deux précédemment présentés, mais il apporte également un nouvel élément capital pour la résolution de notre problématique de recherche, qui consiste in fine à proposer aux biologistes (et à un public non expert en informatique) des modèles de comportements génériques, intuitifs, simples à implémenter et explicables, qu'ils soient construits manuellement ou automatiquement. Le passage en revue de la littérature nous a révélé qu'il existe un grand nombre d'approches de modélisation du comportement animal, des plus simples aux plus complexes, qui répondent à des besoins spécifiques.

Dans cet espace diversifié, opter pour une approche particulière nous oblige à faire également face à ses limites. Dans une telle situation, une solution serait de choisir une approche qui fait un compromis, mais cette solution reste problématique, car aucune des approches explorées ne nous permet d'atteindre notre objectif ultime. À la lumière de toutes ces contraintes, nous avons opté pour la définition d'un modèle de comportement propre à nos objectifs, comme l'ont fait également d'autres chercheurs, notamment (Drogoul 1993) et (Coquelle 2005).

À cet effet, nous présentons dans ce chapitre notre proposition de modèle générique de comportement animal dénommé ANIMETA-MOD. Nous verrons comment ce modèle, c'est-à-dire l'animat, est constitué, comment il fonctionne, et comment il peut être simulé.

3.1 ANIMETA-MOD, modèle générique compatible à l'intégration des métaheuristiques

La définition de notre modèle générique repose sur un ensemble d'hypothèses et de postulats recueillis de notre revue de littérature. Il s'agit entre autres de :

— *«Le comportement d'un animal ne saurait rester figé tout au long de sa vie»* : Cette pensée est mise en avant par le darwinisme, où les comportements de base sont appelés à évoluer. Nous pouvons considérer que cette même pensée a été confirmée, mais sous une autre forme, avec les courants béhavioristes qui prônent la prise en compte de la notion d'apprentissage dans le comportement des animaux et, par conséquent, leur évolution ;

— *«Le comportement d'un animal peut tout aussi bien être expliqué par des mécanismes simples sans faire appel à un processus complexe»* : Cette théorie est développée par le canon de Morgan avec le principe de parcimonie. Ce principe sera d'ailleurs utilisé par (Drogoul 1993) pour proposer un modèle de comportement animal.

— *«Les comportements d'animaux peuvent être définis par un ensemble de stimuli qui déclenchent des unités de comportement selon un principe de cause à effet»* : C'est ce que dit l'éthologie objectiviste proposée par Lorenz avec les FAP et les IRM. Cette même théorie a été en partie utilisée par (Coquelle 2005) pour proposer un modèle de simulation du comportement animal avec les composants «Perception» et des composants «Comportement» qui peuvent représenter respectivement les IRM et les FAP.

— *«Il est possible de définir une hiérarchisation des comportements avec un niveau de priorité»* : Nous tirons cette conclusion de «l'éthologie d'aujourd'hui» (modèle fonctionnel de l'éthologie).

Sur la base de ces éléments et de notre recherche, qui s'effectue dans un contexte éthologique (où l'étude du comportement se fait à l'échelle de l'individu), le modèle générique dénommé ANIMETA-MOD que nous proposons utilise une approche centrée sur l'individu (Grimm 2019) et se présente comme une entité placée dans un environnement où il a la capacité de le percevoir et d'y agir, faisant lui-même partie intégrante dudit environnement. Il est doté d'un ensemble de comportements évolutifs ordonnés selon leur priorité, où chacun d'entre eux est déclenché par des stimuli exclusifs. Ainsi, deux comportements ne peuvent pas s'exprimer simultanément. Chaque comportement à son tour est lui-même constitué d'actions élémentaires dont l'exécution entraîne des changements sur l'animat lui-même et/ou son environnement. Pour assurer cette fonction, l'animat est identifié par un type et trois principaux groupes fonctionnels, dont les rôles sont les suivants :

— Le premier groupe fonctionnel est appelé «groupe des perceptions». Il permet à l'animat de percevoir les informations de son environnement ainsi que celles qui le concernent ;
— Le deuxième groupe fonctionnel est appelé «groupe des désirs». Il s'agit de l'ensemble des unités de comportement de l'animat déclenché en fonction des perceptions rapportées par le groupe prévu à cet effet ;
— Le troisième groupe fonctionnel est appelé «tâche par défaut». Il représente un groupe d'action qui est exécuté si aucune unité de comportement n'est déclenchée.

Cette organisation structurelle est fortement inspirée de l'architecture BDI retrouvée dans la modélisation basée sur les agents (voir section 1.2.2.3.2). Le choix d'un modèle de type « réactif » repose principalement sur le principe de parcimonie mis en avant par Morgan (Sober 1998), dont la mise en œuvre, dans des travaux, a permis d'obtenir des résultats intéressants (Jonker et al. 2001 ; Coquelle 2005 ; I Boumans 2017). Au-delà de ces éléments présentés, le modèle réactif est une approche intéressante pour la validation de nos hypothèses.

Bien que le modèle soit assez réactif, il dispose de mécanismes permettant de renforcer ou d'inhiber certains comportements en fonction des expériences. Cette manière de procéder peut paraître comme une forme d'apprentissage, mais nous la considérons plutôt comme un mécanisme d'habituation ou d'assimilation qui confère à l'animat la capacité de réagir plus ou moins rapidement à un stimulus et d'enclencher le comportement correspondant. De plus, lorsqu'un comportement est déclenché, il peut être interrompu au profit d'un autre comportement plus important ou prioritaire. Tous ces éléments font de notre animat, un modèle qui n'est pas purement réactif.

La figure 3.1 montre les différents composants qui forment le modèle de comportement que nous proposons. Il s'agit des composants *Perception*, *Sense* (dans le groupe des perceptions), *Desire* (dans le groupe des désirs), puis enfin *Task* et *Action* (dans le groupe de la tâche par défaut).

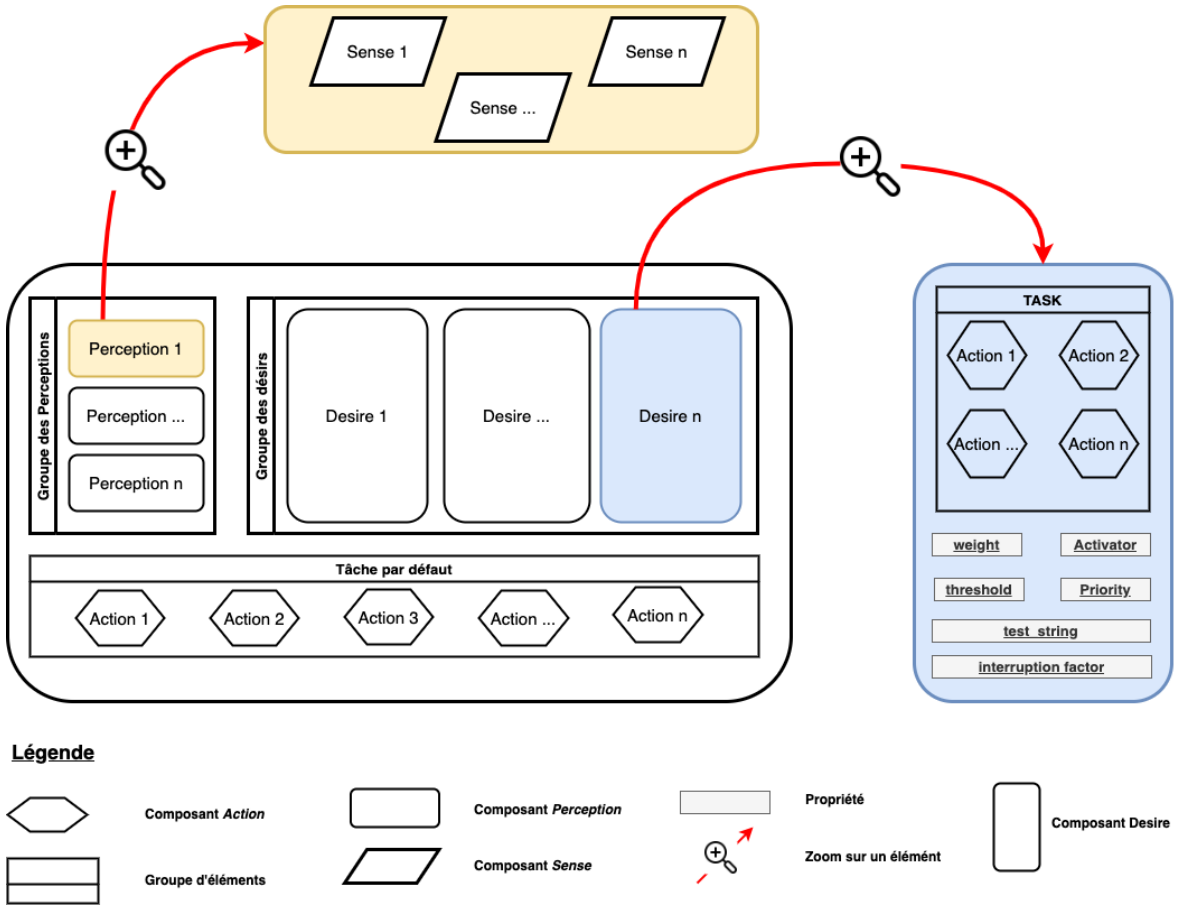


FIGURE 3.1 – Architecture fonctionnelle d'un animat dans ANIMETA-MOD

3.1.1 Le composant *Perception*

Dans un contexte réaliste, les composants *Perception* peuvent être considérés comme les sens de l'animal dans la mesure où ils lui permettent de prendre des informations sur son environnement. Cependant, contrairement à un sens (ex :la vue, l'odorat, etc.) qui a pour objectif de récupérer en une fois une information, de l'analyser et d'y détecter des éléments, le composant *Perception* est défini pour détecter directement l'élément spécifié. Illustrons ce que nous exprimons là avec le sens de la vue chez un animal qui lui permet de voir son environnement. À l'aide de ce seul et unique sens, il collecte tous les éléments visuels de son environnement qui sont ensuite traités par une autre entité (le cerveau) pour tirer des informations comme les distances, les formes, les couleurs et bien d'autres. Appliquer ce même procédé pour ANIMETA-MOD pourrait s'avérer très coûteux et complexe puisque l'animat est tenu de collecter tous les éléments de l'environnement et les traiter un à un, y compris ceux qui n'influencent pas son comportement. Pour cette raison, l'animat n'est constitué que des composants *Perception* qui ne détectent que les éléments qui peuvent intervenir dans son comportement.

Supposons un animat qui devra détecter dans son environnement un objet k de couleur c, où se trouvent plusieurs objets k de couleurs différentes. L'animat sera donc constitué d'un composant *Perception* p qui ne détectera que les objets k de couleur c, les autres étant exclus automatiquement. Chaque détection effectuée par un composant *Perception* est accompagnée d'une valeur qui représente la force de la détection du composant. Cette valeur est calculée avec une expression mathématique définie par le concepteur du modèle lui-même avec une syntaxe définie à cet effet. Cette syntaxe est présentée plus loin et plus en détail dans le tableau 3.2.

Afin donc de définir de façon précise avec une certaine granularité, les composants *Perception* sont définis au moyen d'autres composants appelés *Sense*.

3.1.2 Le composant *Sense*

Il s'agit ici de la plus petite unité pour définir une perception pour le modèle [1]. Il définit chaque condition de détection qui intervient dans la perception. Par exemple, supposons une espèce conceptuelle qui a pour comportement de se mettre au repos lorsqu'il fait nuit et dès qu'elle aperçoit les commodités nécessaires. Pour définir cette situation par un composant *Perception* à l'aide des composants *Sense*, il faudra en utiliser deux. Le premier va se charger de détecter dans son environnement lesdites commodités, et le second va lui se charger de détecter s'il fait nuit. L'ensemble de ces détections conjuguées forme la détection de la perception.

Un composant *Sense* est défini par :

— une *cible* correspondant au type de l'objet à détecter dans l'environnement. L'animat lui-même étant dans cet environnement, il peut tout aussi bien être une cible. Dans l'exemple présenté précédemment, la cible du composant *Sense* qui porte sur les commodités pourrait être les types «nid» ou «paille».

— un *angle* définissant un secteur circulaire où le composant *Sense* fera sa détection. Cela signifie simplement que seuls les objets qui se trouvent dans le secteur sont testés par rapport aux conditions définies (voir figure 3.3). La définition de l'angle n'est pas requise lorsque la cible est l'animat lui-même.

— un *rayon* qui, associé à l'angle, définit le secteur circulaire et la portée du composant *Sense* (Alcazar 2011), comme illustré par la figure 3.3. La définition de l'angle n'est pas requise lorsque la cible est l'animat lui-même.

— une *expression* représentant une formule de calcul de la force de la détection faite par le composant *Sense*. Elle peut être assimilée à une fonction $\gamma : A \rightarrow \mathbb{R}$ (avec A l'ensemble des attributs de l'animat) qui renvoie une valeur représentant la force correspondante. Tout comme pour le composant *Perception*, sa définition est laissée au soin du concepteur selon ses besoins et du contexte de modélisation. Elle utilise également la même syntaxe présentée dans le tableau 3.2.

1. Il est important de faire la différence lorsque nous utilisons simplement le mot «perception» et lorsque nous parlons du composant *Perception*. Le mot désigne la perception au sens littéral et le composant désigne l'ensemble formé par l'expression de la force des composants *Sense*

Grâce à la combinaison des composants *Sense* pour former les composants *Perception*, il devient possible de doter notre animat de la capacité de détecter une information très précise dans l'environnement. On parlera donc de perception «valide» lorsque ce que décrit son composant *Perception* est détecté. Cette validité se fait au moyen de l'opération logique «ET», c'est-à-dire que le composant *Perception* n'est valide à un instant donné que si tous les composants *Sense* sont valides [2].

Finalement, en guise de résultat, lorsqu'une perception est valide, elle renvoie les éléments détectés, l'élément le plus proche (sa position et sa distance) et bien sûr la force de la perception calculée.

La figure 3.2 présente formellement, à travers un diagramme de classe UML, les relations entre les composants *Perception* et *Sense*.

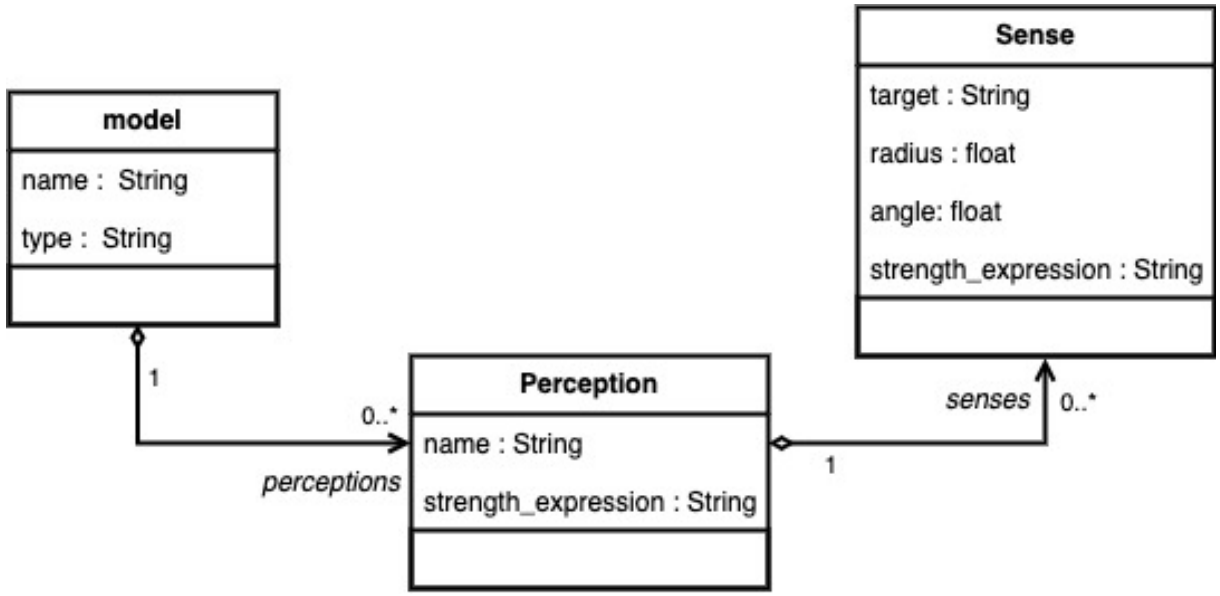


FIGURE 3.2 – Diagramme des classes UML des composants *Perception* et *Sense*

À titre d'exemple pour comprendre le fonctionnement des composants *Perception* et des composants *Sense*, considérons un autre animat qui déclenche un comportement de chasse s'il a faim et qu'il voit une source de nourriture (ou proie) identifiée par le type «nourriture» dans un rayon de 3 mètres avec un angle de vue de 90°. La force de détection du composant *Perception* est égale au produit du nombre de proies détectées par la valeur de sa variable interne représentant son niveau d'énergie. Une telle perception peut être modélisée par deux composants *Sense*. L'un pour connaître le niveau d'énergie de l'animat lui-même et l'autre pour détecter les proies. La configuration de chaque composant *Sense* est présentée dans le tableau 3.1. Les expressions utilisées sont établies sur la base du tableau 3.2. Remarquons que pour le composant *Sense* défini pour récupérer la valeur de la variable «energy», les valeurs de l'angle et du rayon ne sont pas définies puisqu'elles ne sont pas nécessaires quand le composant a pour cible l'animat lui-même. La figure 3.3 illustre la zone de perception du composant *Sense* portant sur les proies. Dans cette zone définie par l'angle α et le rayon

2. On conviendra qu'un composant *Sense* ou un composant *Perception* est valide lorsqu'il détecte un objet conformément à sa description

r se trouvent deux objets, la proie et le prédateur. Dans ces conditions, seul l'objet N°2 représentant la proie sera détecté par le composant *Sense* puisqu'il est le seul qui respecte toutes les conditions imposées par la cible, le rayon et l'angle.

Quant à la force de détection du composant *Perception*, elle est définie par l'expression $1.$2 qui veut dire le produit des résultats des composants *Sense* 1 et *Sense* 2.

TABLE 3.1 – Exemple de définition d'un composant *Perception* par deux composants *Sense*

Attribut	**Sense 1** (niveau d'énergie)	**Sense 2** (proie)
Cible	@	Proie
Angle	Non définie	90
Rayon	Non définie	3
expression	@.energy	$.len

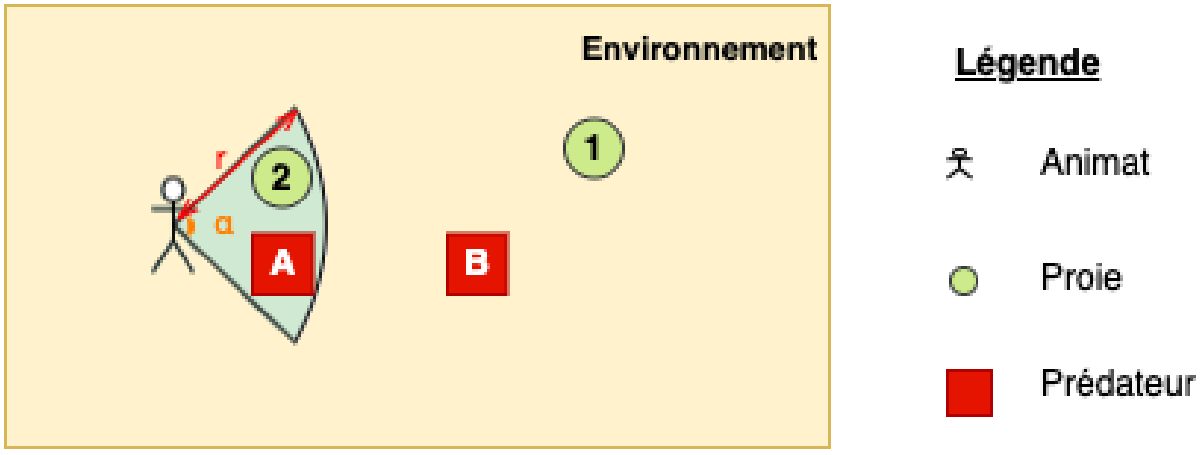


FIGURE 3.3 – Illustration de la zone de détection d'un composant *Sense*

3.1.3 Le composant *Task*

Le composant apparaît à la fois dans le groupe des désirs et dans le groupe de la tâche par défaut. Son rôle et son organisation s'inspirent particulièrement de (DROGOUL 1993). Le composant *Task* représente dans sa globalité une tâche exécutée par l'animat pour atteindre un objectif défini. Cette tâche est en réalité une liste d'actions élémentaires exécutées dans un certain ordre et dans un certain temps par l'animat, pour agir sur son environnement ou sur lui-même. Si, par exemple, l'objectif pour l'animat est de se nourrir, la tâche à réaliser pourrait être constituée successivement des actions «chasser», «attraper la proie», «transporter la proie» et «manger». L'exécution complète de ces actions permet d'atteindre l'objectif qui est de se nourrir. Cette description nous permet déjà de comprendre que le composant *Task* est le composant «opérationnel» de l'animat.

Les différentes actions qui constituent le composant *Task* sont elles-mêmes des composants que nous appellerons tout simplement *Action*, et que nous présenterons plus tard dans la section 3.1.4.

Au-delà des actions élémentaires qui le composent, un composant *Task* est également défini par sa durée, qui correspond à la somme des durées de chaque action élémentaire.

3.1.4 Le composant *Action*

L'existence de ce composant est motivée par les explications données pour le composant *Task*. Le composant *Action* représente une action de base (action fondamentale) que peut réaliser l'animat. Ces composants sont indépendants les uns des autres lorsqu'ils sont mis ensemble dans un composant *Task*. Cela veut dire que l'exécution de l'un ne conditionne pas celle de l'autre. Il peut exister plusieurs actions élémentaires prédéfinies. Elles sont toutes définies par leur nom et la durée de l'opération qu'elles réalisent. Néanmoins, il y a la possibilité d'avoir d'autres paramètres supplémentaires qui leur sont propres pour réaliser ces opérations.

3.1.5 Le composant *Desire*

La structure de ANIMETA-MOD étant inspirée de l'architecture BDI des systèmes agents, nous reprenons le concept du désir de cette architecture en y ajoutant des spécificités propres à notre contexte d'étude, qui est celui du comportement animal. Partons d'un exemple concret qui a fortement inspiré la définition du composant *Desire*.

Considérons le comportement de la parade nuptiale chez certaines espèces. Ce comportement est intégré dans l'ensemble des comportements que peuvent avoir les individus de cette espèce, mais la volonté d'adopter ce comportement n'apparaît que lorsqu'il y a certains stimuli, tels que la période de reproduction ou la présence d'un partenaire de reproduction. À ce stade, il ne s'agit que d'une volonté, mais le comportement lui-même n'est pas encore effectif. Il ne devient effectif que si le comportement en cours est moins important que le comportement de reproduction. Dans le cas où il s'agit, par exemple, d'un comportement déambulatoire, il est interrompu au profit du comportement de reproduction qui peut alors s'exécuter. Toutefois, il peut également être interrompu à son tour lorsqu'un autre comportement plus important est déclenché par une perception. C'est ce qui se produirait si l'individu exécute sa parade nuptiale et qu'il perçoit un danger approchant. Par analogie à cette façon de faire de certaines espèces, nous avons défini le composant *Desire* en le considérant comme un ensemble qui représente une unité de comportement, mais qui embarque tous les éléments nécessaires pour son déclenchement et son accomplissement. Il intègre donc, à cet effet, l'ensemble des stimuli (dans le cas de ANIMETA-MOD, les composants *Perception*) susceptibles de le déclencher.

Comme on peut souvent le trouver dans les implémentations de l'architecture BDI (Caillou et al. 2017), le désir est un objectif défini que l'agent cherche à atteindre, ou pour lequel il cherche un compromis s'il en existe plusieurs. Lorsque l'objectif est atteint, il est retiré de la liste ou transformé en une croyance (*Belief*). Ce procédé ne pourrait pas nous permettre de reproduire convenablement les comportements des animaux. Pour cela, plutôt que de définir

le composant *Desire* comme des objectifs qui sont retirés lorsqu'ils sont atteints, nous donnons la possibilité de passer d'un état à un autre durant la simulation. Ils peuvent prendre les états suivants :

— L'état *«activable»* correspond à l'activation d'une ou plusieurs perceptions liées au composant *Desire*. Lorsqu'il se trouve dans cet état, cela indique que les conditions sont réunies pour créer la volonté d'atteindre un objectif. Dans notre exemple, cet état correspond à l'état où se trouve l'animal lorsqu'il perçoit (au sens du composant *Perception*) à la fois un(e) partenaire et la période de reproduction.

— L'état *«activé»* correspond à la présence du composant dans la file d'attente des comportements à exécuter. Lorsqu'un composant *Desire* est dans cet état, il n'est pas encore mis en œuvre. Cet état correspond, dans notre exemple, à l'apparition réelle de la volonté de se reproduire. Lors de l'ajout du composant *Desire* dans la file d'attente des comportements à exécuter, il est accompagné des conditions de son activation, c'est-à-dire le résultat de la perception à son origine. Cette information peut être utile plus tard lors de la gestion de l'ordre d'exécution. En effet, étant donné que plusieurs désirs ne peuvent pas être exécutés simultanément, si un composant *Desire* D_2 est activé et qu'un autre composant *Desire* D_1 est encore en cours d'exécution, le temps que D_1 soit satisfait, il est possible que les conditions d'activation de D_2 aient changé au point que le besoin de le satisfaire ne soit plus nécessaire. Voilà pourquoi il est important de conserver la condition d'activation pour vérifier si elle est toujours valable.

— L'état *«actif»*, quant à lui, correspond à la mise en œuvre concrète du désir. Lorsqu'un composant *Desire* est *actif*, l'animat exécute (tant que possible) les actions nécessaires pour l'accomplissement de l'objectif formulé par le composant.

NB : Pour désigner désormais un composant Desire, nous préciserons l'état (activable, activé, ou actif) dans lequel il est pour des raisons de compréhension.

Ainsi, le comportement de l'animat se résume finalement à une alternance entre l'exécution et l'interruption des composants *Desire*, étant donné que toutes les opérations de l'animat s'y réfèrent. Afin d'assurer pleinement cette orchestration de manière optimale, un composant *Desire* est défini par :

— un nom unique (*name*) qui sert à l'identifier.

— un poids (*weight*), un entier naturel non nul qui définit l'importance du comportement que définit le composant par rapport aux autres composants *Desire*. Plus ce nombre est grand, plus il est important par rapport aux autres et donc a plus de chance d'être priorisé à l'exécution. En nous référant à l'exemple précédent, si les composants *Desire* du comportement de reproduction et du comportement de fuite d'un danger sont tous deux à l'état «activé», ce sera le composant *Desire* du comportement de fuite qui passera à l'état «actif» puisqu'il serait défini avec un poids plus grand.

— un coefficient (*interruption_factor*), un réel compris entre 0 et 1 qui représente la susceptibilité d'interrompre la réalisation du composant *Desire* lorsqu'il est dans l'état «actif». Dans notre exemple le comportement de la déambulation aura un coefficient grand, car il est celui qui est le plus susceptible d'être interrompu. Concrètement,

comme cela a été le cas dans (IJMM Boumans et al. 2016), le coefficient peut être vu comme la probabilité d'interrompre le composant lorsqu'un autre plus important ou prioritaire devra passer à un état *actif*, afin d'éviter des interruptions systématiques.

— une fonction d'activation (*activator*) qui est une expression utilisée pour déterminer la force du stimulus qui déclenche le composant *Desire*. Il s'agit bien ici de la force du stimulus, qu'il ne faudrait pas confondre avec force la perception. La force du stimulus qui déclenche un comportement peut faire intervenir d'autres facteurs (exemple la motivation interne) comme l'ont souligné (Lorenz 1950) et (Drogoul 1993). La définition de l'expression de la fonction d'activation utilise elle aussi la syntaxe du tableau 3.2.

— un seuil (*threshold*), également un nombre réel, définit le niveau que devra franchir la valeur retournée par une fonction d'activation pour que le composant *Desire* passe de l'état «activable» à l'état «activé». Ce seuil permet de ne pas avoir un modèle purement réactif qui déclenche systématiquement un comportement lorsqu'un stimulus est présent.

— une fonction d'évaluation des priorités (*priority*) qui est elle aussi une expression définie à partir de la syntaxe du tableau 3.2 pour déterminer la priorité du composant *Desire* lorsqu'il est à l'état *actif* par rapport aux autres de la file d'attente qui sont dans l'état *activé*.

— une fonction (*test_function*) booléenne qui permet de tester si le désir est satisfait ou non. Ce test est effectué pour mettre à jour la file d'attente des composants *Desire* activés. Sa définition utilise elle aussi la syntaxe du tableau 3.2.

— une tâche (un composant *Task*) telle que décrite dans la section 3.1.3. Il s'agit d'une liste ordonnée de composants *Action* que l'animat devra exécuter pour atteindre l'objectif défini par le composant *Desire*.

Toutes ces descriptions faites sur les composants qui forment le modèle type de comportement que nous proposons nous permettent de les représenter plus formellement à travers le diagramme des classes UML de la figure 3.4.

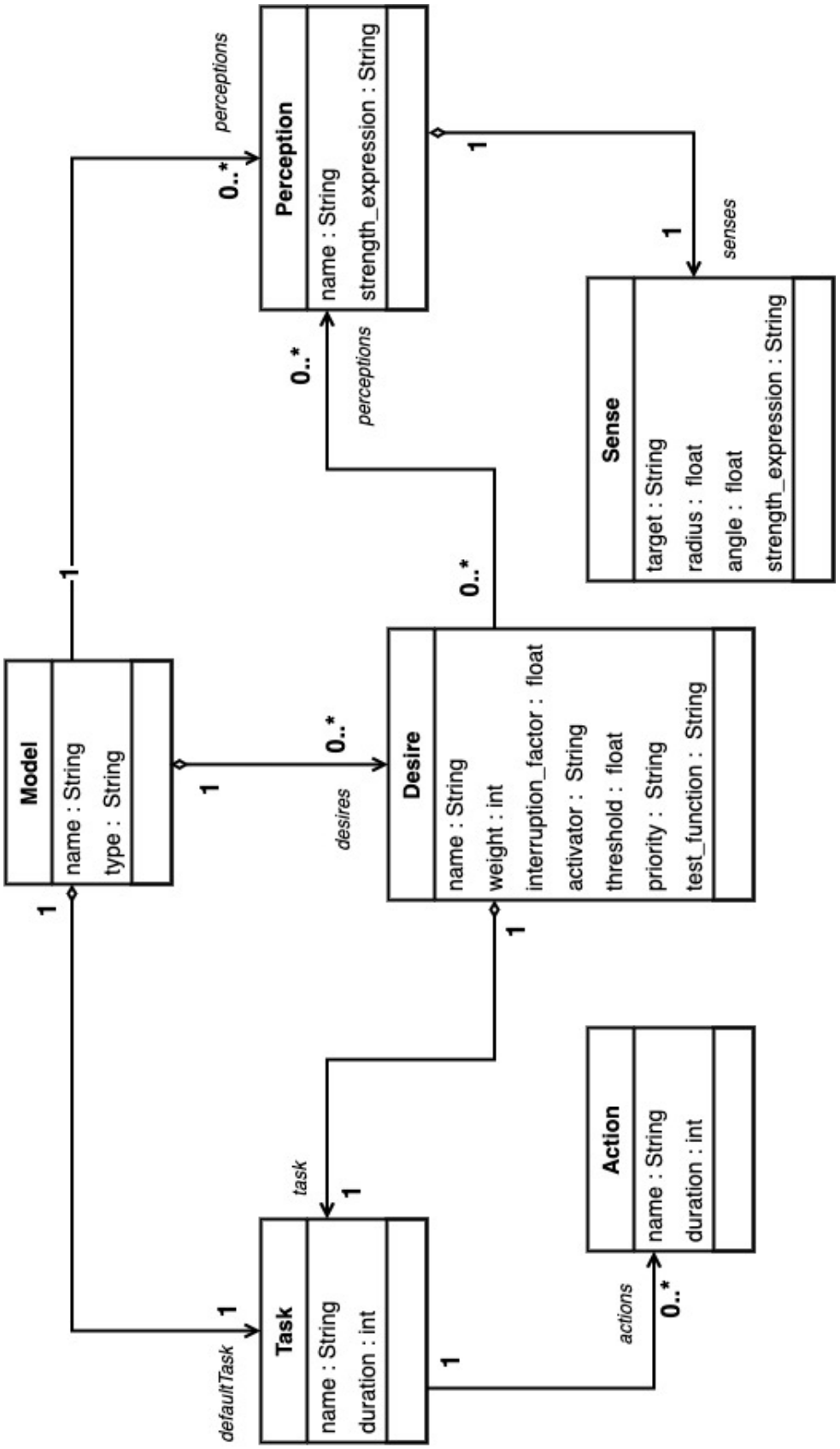


FIGURE 3.4 – Diagramme des classes UML décrivant un animat dans ANIMETA-MOD

TABLE 3.2 – Syntaxe de définition d'expression dans les composants *Sense*, *Perception* et *Desire*

Symbole	**Correspondance**	**Exemple**	
		Expression	Description
@	L'animat	@.enegry	accède à la propriété *«energy»* de l'animat
π	Un composant *Perception*	$\pi.closer\_agents\_position$	accède à la position de l'élément le plus proche détecté par un composant *Perception*
$	Un composant *Sense*	$.closer_distance	Accède à la distance qu'il y a entre l'animat et l'élément le plus proche détecté par le composant *Sense*
Δ	Un composant *Desire*	$\Delta.weight * \pi.agents\_len$	Accède et multiplie entre eux, le poids du composant *Desire* et le nombre d'éléments détectés par le composant *Perception*
€	L'environnement	€.timer	Accède à l'horloge de l'environnement. Il peut être vu dans la réalité comme l'heure qu'il est à cet instant

En proposant ANIMETA-MOD pour représenter le comportement animal, il reste compréhensible (interprétable et explicable), pour le biologiste puisque tout le processus lui est transparent. À Chaque instant de la simulation, il est possible de connaître les conditions dans lesquelles se trouve l'animat, comprendre le choix qui est fait en fonction de ces conditions et comprendre donc la réaction de l'animat à l'aide des actions clairement définies par les composants *Desire* et la tâche par défaut.

L'utilisation de ANIMETA-MOD pour représenter le comportement animal se distingue par sa compréhensibilité, son interprétabilité et sa transparence, autant pour les biologistes que pour tout autre observateur. À chaque instant de la simulation, il est possible de connaître au complet, les conditions dans lesquelles se trouve l'animat, de comprendre les choix qui sont faits en fonction de ces conditions, et donc d'expliquer la réaction de l'animat grâce aux actions clairement définies par les composants *Desire* et le composant *Task* de la tâche par défaut. Les composants *Perception* permettent de capturer les informations de l'environnement, les composants *Desire* décrivent les objectifs et les actions possibles, et le composant *Scheduler* coordonne toutes ces données de manière explicite et traçable. Les biologistes et les observateurs peuvent ainsi avoir une visibilité complète sur le processus de prise de décision de l'animat.

3.1.6 L'environnement

La modélisation de l'environnement est la représentation que nous donnons à celui-ci pour simuler notre modèle de comportement animal. Deux aspects sont à prendre en compte particulièrement. Le premier définit comment il est organisé et le second définit comment il est structuré.

3.1.6.1 Organisation de l'environnement

Pour rester dans la logique de notre objectif de généricité et de simplicité dans la modélisation du comportement animal, nous optons pour une représentation basique de l'environnement. À cet effet, nous adoptons une modélisation similaire à celle présentée dans les travaux de (IJMM Boumans et al. 2016) et (Drogoul 1993) pour modéliser l'environnement comme un espace dans lequel peuvent se trouver plusieurs éléments de différents types. La particularité est que tous ces éléments sont modélisés sur la base du modèle de comportement présenté à la section 3.1 pour des raisons de cohérence. Le rôle que jouera chacun de ces éléments de l'environnement est défini par le comportement décrit par ses composants *Desire* et *Task*. Ainsi, un élément dans l'environnement peut être un mur, une plante ou un animat qui n'est pas directement le sujet étudié. Par exemple, dans le cadre de l'étude du comportement d'un carnivore dans son environnement (comme illustré dans la section 3.2 en exemple), bien que les herbivores et les plantes ne soient pas le sujet principal, ils devront être modélisés comme des animats et placés dans l'environnement, car leur présence est nécessaire. Dans le cas où nous souhaitons placer un mur dans l'environnement, le mur serait défini comme un élément sans aucun composant *Perception* ni *Desire*. Sa tâche par défaut serait tout simplement de ne rien faire.

En procédant de cette manière pour tous les types d'objets à placer dans l'environnement, nous parvenons à constituer un environnement complexe, propice à l'étude d'un comportement. À ce stade de nos travaux, nous ne modélisons pas toutes les propriétés physiques qui peuvent exister dans un environnement réel, comme le vent ou la luminosité. Actuellement, seule une horloge est définie, et elle est incrémentée à chaque pas de temps.

Bien que conscients que ces propriétés peuvent influencer certains comportements (par exemple, la propagation des phéromones par le vent), nous proposons comme alternative de les modéliser également comme des éléments avec le comportement approprié. Prenons le cas de la diffusion des phéromones. Elles peuvent être modélisées par des particules qui sont des éléments dotés d'un comportement autonome.

3.1.6.2 Structure de l'environnement

En ce qui concerne l'aspect structurel de l'environnement, la même question concernant la représentation la plus adaptée au contexte d'étude se pose. Pour y apporter une réponse, un premier choix devra être fait entre un espace continu et un espace discret. Les points forts et les points faibles de chacun d'eux ont déjà été mis en avant dans la section 1.2.2.4.1. Ils concernent principalement la précision et la fluidité des mouvements. À ce stade de nos travaux, le besoin n'est pas du côté de la précision ou de la fluidité des mouvements dans la simulation, mais il se trouve plutôt du côté de la validation de l'approche que nous proposons.

Un espace discret semble donc le mieux adapté pour notre contexte. Sa simplicité et le peu de fonctionnalités qu'il implémente favorisent une meilleure analyse des comportements. Néanmoins, si le besoin d'utiliser un espace continu s'impose, le changement peut se faire aisément.

Maintenant que nous avons présenté cette nouvelle manière générique de modéliser le comportement des animaux, il est temps d'illustrer à travers un exemple concret. Cette démonstration nous permettra de mieux saisir comment cette approche peut être mise en œuvre.

3.2 Exemple de modélisation d'un comportement animal avec ANIMETA

Le modèle utilisé pour illustrer notre approche est du type «Proie-Prédateur» que nous appellerons «Modèle A». Il met en œuvre un écosystème reproductible dans un environnement contrôlé. Le modèle A présente un cadre dans lequel un écosystème abstrait est créé avec trois catégories distinctes d'individus. Le comportement des individus de chaque catégorie est décrit par le pseudo-éthogramme représenté par le tableau 3.3 :

TABLE 3.3 – Pseudo-ethogramme des catégories d'individus du modèle A

Catégorie	Rôles	Description
Plantes	Proie	— Cette catégorie se situe au bas de la chaîne alimentaire. Les individus de cette catégorie ne peuvent pas se déplacer.
Herbivores	Proie et prédateur	— La catégorie «Herbivore» se situe en dessous de la catégorie des «plantes» dans la chaîne alimentaire. — Un individu de la catégorie des «herbivores» peut se déplacer, percevoir les «plantes» dans un rayon de 5 unités de longueur (u.l.) et s'en nourrir lorsqu'il a faim. La condition de faim peut être identifiée par un niveau d'énergie inférieur à 100 points. — Lorsqu'un qu'un individu de la catégorie des herbivores se nourrit d'une plante, son niveau d'énergie est augmenté de 2 points. — Les individus de la catégorie «Herbivore» peuvent percevoir leur environnement dans un angle 360°. — À chaque déplacement, un individu perd une quantité d'énergie proportionnelle à la distance parcourue, à raison d'une unité d'énergie par u.l. — Lorsque le niveau d'énergie d'un individu atteint 0, il meurt. — Lorsqu'un individu perçoit jusqu'à une distance de de 10 u.l des individus de la catégorie «carnivore», il les évite.
Carnivores	Prédateur	— La catégorie des carnivores se situe au sommet de la chaîne alimentaire. — Les individus peuvent se déplacer à la recherche de «herbivore» pour se nourrir lorsqu'ils ont faim. La condition de faim peut être identifiée par un niveau d'énergie inférieur à 150 points. — Lorsqu'un qu'un individu se nourrit d'un herbivore, son niveau d'énergie est augmenté de 3 points. — Le niveau d'énergie d'un individu est augmenté de 3 points lorsqu'il se nourrit d'un «herbivore». — A chaque déplacement, il perd 1,5 unité d'énergie par u.l. — Lorsque le niveau d'énergie d'un individu atteint 0, il meurt.

Reproduire ce modèle en utilisant notre approche revient à représenter les connaissances du tableau 3.3 pour chaque catégorie conformément à l'architecture fonctionnelle d'un animat. Dans ANIMETA, toutes les entités sont représentées par des animats, il s'agit donc de construire chaque catégorie comme un animat.

Pour ce modèle, nous avons trois animats à créer : l'animat de type «plante», l'animat de type «herbivore» et l'animat de type «carnivore». Pour chacun d'eux, nous devons constituer les différents composants de l'architecture fonctionnelle d'un animat, en nous basant sur les informations tirées du pseudo-éthogramme du tableau 3.3. Nous utilisons pour cela une méthode descendante, c'est-à-dire partir des composants principaux (le groupe des perceptions, le groupe des nœuds et la tâche par défaut) vers les composants de bas niveau (les paramètres).

3.2.1 Catégorie «carnivore»

De l'analyse des informations apportées par le tableau 3.3, nous pouvons déduire que les animats de type «carnivore» n'agissent directement que sur les animats de type «herbivore». Cette action, qui se traduit par la consommation de l'animat de type «herbivore», est exclusivement déclenchée par un composant «Perception» que nous pouvons définir par les deux composants «Sense» suivants :

— Sense 1 : Permet de prendre en compte l'état interne de l'individu pour savoir s'il a un besoin de se nourrir.
— Sense 2 : Vient en complément au Sense 1 car, pour que l'action de «consommer» un individu de type «herbivore» soit effective, il faut qu'il soit perçu. Alors, ce sens a donc pour cible tout individu de type «herbivore» se trouvant dans le champ de perception de l'individu de type «carnivore».

Le composant *Perception* de l'individu peut donc être modélisé comme présenté dans le tableau 3.4. Le composant *Sense* 1 porte sur l'individu lui-même ($target = @$) [3]. Il teste la condition booléenne ($boolean\_expression = True$) pour déterminer si l'animat a besoin de se nourrir, c'est-à-dire s'il détecte que le niveau d'énergie est inférieur à 150 ($@.energy < 150$). Il retourne comme résultat le niveau d'énergie de l'animat.

Le composant *Sense* 2 porte sur tous les autres individus du type «herbivore» ($target = herbivore$) qui se trouvent dans le champ de vision défini par un angle de 180° et par un rayon de 10 u.l. Ici, il ne s'agit pas d'une condition qui est testée ($boolean\_expression = False$). Si le composant «Sense 2» est activé, il renvoie comme résultat le nombre d'individus du type «herbivore» identifiés dans le champ de vision ($\$.len$).

La force de la perception elle-même est définie par la formule :

$$force = \frac{1}{Sense1} + Sense2$$

Lorsqu'elle est écrite comme expression, on a (1/\$1) + \$2. Cette expression permet d'avoir une grande force de perception lorsque le niveau d'énergie (représenté par *Sense* 1) est bas ou lorsque plus d'herbivores (représentés par *Sense* 2) sont perçus.

3. Le dictionnaire des caractères spéciaux utilisés dans la définition des expressions est présenté dans le tableau 3.2

TABLE 3.4 – Configuration du composant *Perception* d'un individu *Carnivore*

Attribut	**Propriétés**	
name	percept_herbivorous	
Sense 1	*target*	@
	boolean_expression	True
	angle	Null
	radius	Null
	strength_expression	@.*energy* < 150
Sense 2	*target*	herbivorous
	boolean_expression	False
	angle	180
	radius	10
	strength_expression	$.*len*
strength_expression	(1/(1 ∗ $1)) + $2	

Pour ce qu'il en est des composants *Desire*, de l'analyse du tableau 3.3 nous ne pouvons qu'en déduire un seul : celui qui exécute la «consomation» d'un «herbivoire». La configuration donnée à ce composant *Desire* est présentée par le tableau 3.5 et le tableau 3.6.

TABLE 3.5 – Configuration du composant *Desire* d'un individu *Carnivore*

Attribut	**Propriétés**
name	carnivorous_desire_eat_herbivorous
threshold	0
weight	1
test_string	@.*energy* > 150
interruption_factor	1
activator	$\pi.strength$
priority	1
perceptions	[carnivorous_perception]
Task	(Voir tableau 3.6)

TABLE 3.6 – Configuration du composant *Task* du composant *Desire* de l'individu *Carnivore*

Attribut	**Propriétés**		
name	eat_herbivorous_task		
Actions	Eat	*duration_cycle*	1
		gain	10
		move_cost	1,5

Ce qu'il faut remarquer en particulier dans cette configuration, ce sont les valeurs attribuées à threshold, weight, interruption_factor et test_string, puis ensuite au composant Task.

La valeur threshold est maintenue à 0 pour signifier qu'il n'y a pas de seuil à franchir. Dès que la perception est valide, elle peut déclencher le composant *Desire*. weight est mis à 1 (valeur par défaut) car puisque l'animat n'a pas plusieurs composants *Desire*, le poids n'est pas déterminant. Il est donc gardé à la valeur minimale. Il en est de même pour la valeur de interruption_factor, qui est aussi mise à 1 car il ne peut pas y avoir d'interruption du composant *Desire* par un autre. La test_string étant l'expression de la condition qui permet de tester si le désir est «satisfait», elle vérifie si le niveau a atteint les 150. La fonction d'activation est maintenue à $\pi.strength$ pour signifier que la force de la perception est utilisée directement pour être comparée au seuil de stimulation.

Quant à la tâche associée au désir, elle est composée d'une seule action «Eat», configurée de manière à ajouter 3 unités d'énergie lorsqu'elle est exécutée, et pour chaque unité de temps de déplacement vers la cible, 1,5 unité d'énergie est prélevée, conformément à la description du pseudo-éthogramme.

Concernant la tâche par défaut, rien de particulier n'a été précisé dans le pseudo-éthogramme. Pour cela, nous choisissons de définir une tâche par défaut avec une seule action, qui est un déplacement aléatoire. Cette action est configurée avec une probabilité de bouger (ou de rester immobile) mise à 0,5.

3.2.2 Catégorie «Herbivore»

Après analyse des informations communiquées par le tableau 3.3, nous identifions pour les individus de la catégorie des «Herbivores» deux perceptions : une pour détecter la «faim» (niveau d'énergie bas et les individus de la catégorie des plantes) et une autre perception pour détecter les prédateurs (les individus de la catégorie des carnivores).

La perception pour détecter la faim est semblable à celle décrite par le tableau 3.4 pour les individus de la catégorie des «Carnivores», avec quelques changements présentés dans le tableau 3.7. Les changements concernent particulièrement le niveau d'énergie (Sense 1) et la zone de perception définie par l'angle et le rayon.

La configuration du second composant «Perception» pour détecter les carnivores est présentée par le tableau 3.8. L'unique composant «Sense» qu'il comporte porte sur tout individu de type «Carnivore» qui se trouve dans le secteur circulaire défini par un rayon de 5 u.l et dans un angle de 360°. La force de la perception est celle renvoyée par l'unique composant «Sense», qui est ici le nombre d'éléments détectés.

TABLE 3.7 – Configuration du premier composant *Perception* d'un individu *Herbivore*

Attribut	**Propriétés**	
name	percept_plant	
Sense 1	*target*	@
	angle	Null
	radius	Null
	strength_expression	@.*energy* < 100
	boolean_expression	True
Sense 2	*target*	plant
	angle	360
	radius	5
	strength_expression	$.*len*
	boolean_expression	False
strength_expression	(1/(1 ∗ $1)) + $2	

TABLE 3.8 – Configuration du second composant *Perception* d'un individu *Herbivore*

Attribut	**Propriétés**	
name	percept_carnivorous	
strength_expression	$1	
Sense	*target*	carnivorous
	angle	360
	radius	10
	strength_expression	$.*len*
	boolean_expression	False

Les individus de la catégorie des herbivores peuvent être modélisés avec deux composants «Desire», un pour assurer le désir de se nourrir et l'autre pour assurer le désir de fuir les prédateurs.

Le composant *Desire* qui permet à l'individu de se nourrir est semblable à celui des individus de la catégorie des carnivores, comme présenté dans le tableau 3.9. Au-delà du composant *Perception* associé et des expressions de test qui changent pour ce composant *Desire*, la différence majeure se trouve au niveau du poids (weight). Puisque les individus de type «Carnivores» ont plusieurs désirs, il est possible qu'un désir moins important soit interrompu par un autre. Se nourrir étant généralement moins «urgent» que fuir un prédateur, nous mettons le poids à la valeur minimale, qui est 1. C'est dans ce sens aussi qu'une grande valeur est attribuée à l'attribut interruption_factor, afin de rendre plus probable l'interruption de ce comportement au profit de l'autre, qui est de fuir le danger.

TABLE 3.9 – Configuration du composant *Desire* permettant à un individu *Herbivore* de se nourrir

Attribut	**Propriétés**
name	herbivorous_desire_eat_plant
threshold	0
weight	1
test_string	$@.energy >= 100$
interruption_factor	0.8
activator	$\pi.strength$
priority	1
perceptions	[percept_plant]
Task	(Voir tableau 3.10)

TABLE 3.10 – Configuration du composant *Task* du composant *Desire* (nourrir) de l'individu *Herbivore*

Attribut	**Propriétés**		
name	avoid_carnivorous_task		
Actions	Eat	*duration_cycle*	1
		gain	5
		move_cost	1

Le second composant *Desire*, présenté dans le tableau 3.11, a un poids plus élevé par rapport au premier en raison de son importance dans la liste des comportements de l'individu. Étant moins probable d'être interrompu, nous choisissons une petite valeur pour l'attribut interruption_factor. Le désir est défini pour fuir un prédateur identifié, et il n'est satisfait que si la distance entre l'individu et le prédateur le plus proche est supérieure à 10 u.l, comme défini dans l'attribut test_string. En ce qui concerne la tâche associée au composant *Desire*, elle se compose d'une seule action élémentaire, «fuir» (Runaway).

Nous définissons la tâche par défaut comme un déplacement aléatoire avec une probabilité de mouvement de 0,7 afin de rendre les individus un peu plus actifs.

TABLE 3.11 – Configuration du composant *Desire* permettant à un individu *Herbivore* de fuir un individu de type «carnivorous»

Attribut	**Propriétés**
name	herbivorous_desire_avoid_carnivorous
test_string	$dist\_away\_from\_agent(\pi.closer\_agent) >= 10$
activator	$dist\_away\_from\_agent(\pi.closer\_agent) < 10$
perceptions	[percept_carnivorous]
weight	2
threshold	True
interruption_factor	0
priority	0
Task	(Voir tableau 3.12)

TABLE 3.12 – Configuration du composant *Task* du composant *Desire* (fuir) de l'individu *Herbivore*

Attribut	**Propriétés**		
name	eat_plant_task		
Actions	Runaway	*duration_cycle*	1
		move_cost	1

3.2.3 Catégorie «Plant»

Les individus de la catégorie «Plant» n'ont pas un comportement particulier. Tout comme des plantes réelles, ils ne peuvent pas se déplacer. Dans la définition de l'animat, aucun composant *Desire* ni aucun composant *Perception* n'est nécessaire. Par contre la tâche par défaut devra être définie (puisqu'elle est obligatoire) mais avec une action vide.

La manière dont les concepts clés du modèle sont définis leur octroie la propriété d'être générique et donc d'offrir la possibilité de l'implémenter et d'en faire usage sous divers paradigmes. Néanmoins certains s'y prêtent mieux que d'autres selon les objectifs visés par la modélisation.

3.3 Intégration de ANIMETA-MOD dans un Système Multi-Agents

Le contexte éthologique de notre étude nous impose d'étudier le comportement de l'animal de manière individuelle. C'est d'ailleurs l'une des raisons pour lesquelles nous avons proposé cette représentation de l'animat, qui permet de modéliser de manière individuelle le comportement d'un animal placé dans un environnement indépendant avec ses propres attributs et capacités. Néanmoins, il peut arriver que le comportement individuel soit influencé par un comportement collectif (LEBAR BAJEC et al. 2007). De tels comportements ne peuvent évidemment pas être étudiés efficacement en se focalisant exclusivement sur l'individu. Le support sur lequel notre animat sera implémenté devra donc permettre de gérer de manière indépendante chaque entité, d'isoler un individu ou de le placer dans un cadre collectif.

Parmi les travaux portant sur la modélisation du comportement animal et les approches de modélisation que nous avons étudiés dans la littérature (voir (ORD et al. 2005 ; TANG et al. 2010 ; VALLETTA et al. 2017)), l'approche par les agents semble la plus adaptée. Elle offre une grande flexibilité dans l'implémentation et la gestion des entités, en parfaite adéquation avec les besoins de notre étude pour de multiples raisons. Parmi celles-ci, on peut citer :

— Premièrement, les agents, étant par définition «autonomes», se prêtent aisément à la contrainte d'implémenter pour chaque entité un comportement spécifique avec des attributs tout aussi spécifiques. On peut ainsi isoler et étudier un individu ou analyser les comportements collectifs qui émergent des comportements individuels.

— Deuxièmement, un SMA est un environnement indépendant dans lequel se trouvent des agents qui agissent en toute autonomie en fonction des informations collectées. Ceci correspond parfaitement à notre contexte d'étude. La simulation d'un SMA est assez

similaire au fonctionnement du règne animal. C'est donc une approche plus réaliste et puissante pour comprendre le comportement des animaux, car elle permet de prendre en compte les comportements émergents résultant de l'interaction des agents et des dynamiques à plusieurs niveaux. Ce caractère est un atout majeur pour l'approche que nous proposons, qui vise à inclure l'expert dans le processus de génération de modèles. De surcroît, le fait de voir un ensemble d'agents agir dans un environnement est très ressemblant à un ensemble d'individus d'une espèce. Cette ressemblance rend la simulation intuitive pour l'expert, qui est ainsi plus à même de confirmer ou d'infirmer des observations.

— Troisièmement, leur implémentation demeure assez simple et peu laborieuse, avec un fonctionnement suffisamment clair. En effet, lorsqu'ils sont implémentés comme un modèle réactif, l'algorithme exécuté est connu et reste analysable. De plus, c'est une approche qui a prouvé son efficacité dans des cas d'études similaires au nôtre (Drogoul 1993 ; Coquelle 2005 ; IJMM Boumans et al. 2016).

Compte tenu de ces raisons, nous pensons également que l'approche basée sur les agents est optimale pour implémenter la conception de l'animat telle qu'elle a été décrite. Dans la suite de cette étude, il faudra donc considérer l'animat comme un agent «autonome» qui implémente les composants cités dans la section 3.1 et qui est en constante interaction avec son environnement.

En optant pour cette approche basée sur les agents, il est donc nécessaire de choisir une plateforme de modélisation de systèmes multi-agents adaptée. Comme cela a été présenté précédemment dans la section 1.2.2.7, il existe un grand nombre de plateformes, chacune avec ses particularités. Compte tenu de notre contexte d'étude, nos critères de choix pour la plateforme sont principalement les suivants :

— La *flexibilité* pour descendre au niveau de programmation le plus bas possible afin de procéder, si besoin, à des ajustements dans le paradigme ;
— La *compatibilité avec les plateformes (systèmes d'exploitation) et l'optimisation des ressources* pour permettre une large utilisation sans contrainte particulière sur le matériel ;
— La *rapidité* d'exécution des simulations, car, par nature, toute approche de modélisation vise des simulations optimisées. Au-delà de cet aspect, nous aurons à réaliser plusieurs simulations lors du processus de génération de modèles. Le nombre de simulations peut croître très rapidement en fonction du nombre d'itérations nécessaires pour générer un modèle.
— La *qualité des données* produites en sortie de simulations et la facilité de leur traitement.

Dans le tableau 3.13 nous évaluons quelques plateformes susceptibles d'être utilisées pour implémenter l'animat (Abar et al. 2017) en leur attribuant un score compris entre 1 et 5.

Table 3.13 – Comparaison des plateformes utilisables pour implémenter l'animat (noté sur 5)

Plateformes	Flexibilité	Comptabilité	Rapidité	Qualité des données	Moyenne
NetLogo	3	5	4	4	***4***
MESA	3	3	5	4	***3,75***
Gama	4	4	3	4	***3,75***
AgentPy	3	3	5	3	***3,5***
Swarm	2	4	4	3	***3,25***

L'exploitation du tableau 3.13 révèle que chacune des plateformes présentées peut être utilisée d'une manière ou d'une autre pour l'implémentation de l'animat. Cependant, aucune d'entre elles ne satisfait tous les critères de choix. Face à ce constat, nous optons pour le développement *ex nihilo* d'un système conçu sur la base de nos besoins spécifiques pour l'agent, sa simulation et le traitement des données.

3.4 Exécution de la simulation du modèle

À la simulation, le fonctionnement du modèle est dirigé par un composant appelé *Scheduler*, un anglicisme pour désigner un *«planificateur»* ou un *«ordonnanceur»*. Son rôle dans le modèle est de coordonner son activité pendant toute la durée de l'exécution. Il déclenche le processus d'acquisition des informations de l'environnement via les composants *Perception*, traite l'information collectée et déclenche les réactions correspondantes. Tout comme un gestionnaire de tâches, il gère la file d'attente des composants *Desire*, en les activant ou en interrompant les actions en temps voulu et selon les besoins. Le composant *Scheduler* intègre également une horloge interne pour la gestion du temps, un facteur qui peut fortement influencer le choix d'une action ou d'une autre.

L'exécution du modèle à la simulation est orchestrée par le composant *Scheduler*. Elle s'effectue de manière cyclique à travers cinq étapes que sont : la collecte des informations, la formulation des désirs, la mise à jour des désirs, la prise de décision et l'exécution de l'action. Elles sont décrites par le diagramme d'activité UML présenté dans la figure 3.5.

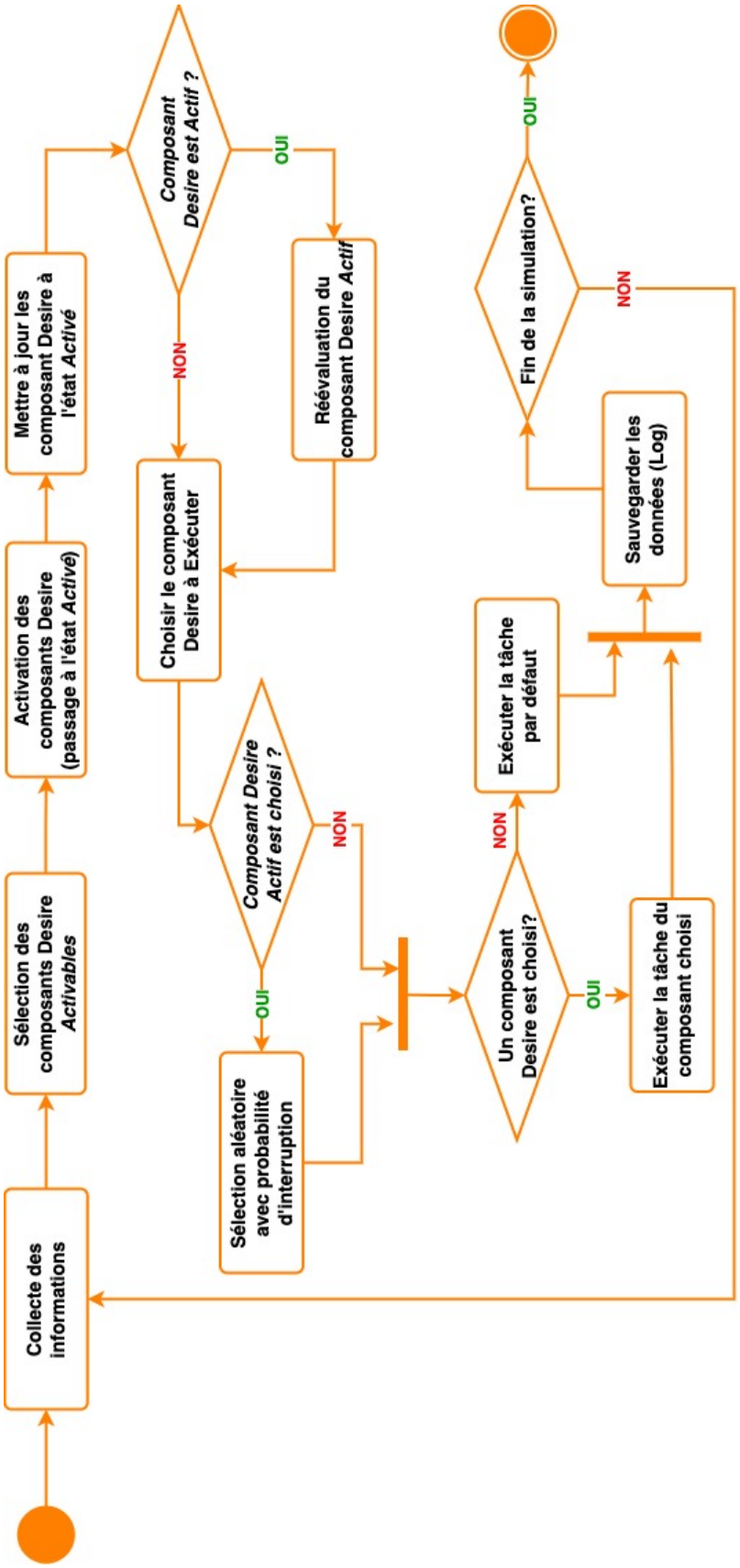


FIGURE 3.5 – Diagramme d'activité UML décrivant le fonctionnement d'un animat.

3.4.1 La collecte des informations

La routine d'un animat commence par la collecte d'informations provenant de son environnement. Cette étape, fondamentale pour lui permettre d'agir en fonction de ce qu'il perçoit dans l'univers qui l'entoure, repose sur l'utilisation de chacun de ses composants *Perception*. Concrètement, le composant *Scheduler* prend en charge cette tâche en parcourant successivement chacun des composants *Perception* du modèle. Pour chaque composant *Perception*, il procède à leur test afin de déterminer s'ils sont valides à cet instant précis. Il est important de rappeler qu'un composant *Perception* est considéré comme valide lorsque tous ses composants *Sense* réalisent la détection spécifiée dans leur description.

Lorsqu'un composant *Perception* est jugé valide, le composant *Scheduler* le récupère et l'utilise pour identifier les composants *Desire* qui peuvent être déclenchés. Autrement dit, il détermine les composants *Desire* qui sont susceptibles de passer à l'état activable, puis finalement à l'état activé. Le processus de passage à l'état activable est relativement simple : le composant *Scheduler* vérifie si le composant *Perception* valide figure dans la liste des déclencheurs de chaque composant *Desire*. Si c'est le cas, le composant *Desire* passe à l'état activable.

La phase suivante est celle du passage à l'état activé. À ce stade, la fonction d'activation propre à chaque composant *Desire* est utilisée pour calculer une valeur qui sera comparée au seuil défini (propriété *threshold*). Si cette valeur dépasse le seuil, le composant *Desire* est activé. Il est alors ajouté à une file d'attente, enregistré avec les résultats de la perception, et classé en fonction de son poids par rapport aux autres composants *Desire* actifs.

L'ensemble de ces informations actualisées est essentiel pour permettre à l'animat d'avoir une compréhension en temps réel de son environnement. De plus, cela lui permet d'ajuster ses objectifs à la phase suivante de la routine, qui est la mise à jour des composants *Desire*.

3.4.2 Mise à jour des composants *Desire*

Les composants *Desire* d'un animat peuvent subir des changements d'état en réaction à l'évolution de leur environnement. Pour illustrer cela, prenons un exemple : à un moment donné t, un animat active un composant *Desire* dans le but de se diriger vers une proie. Cependant, si la proie prend la fuite avant d'être atteinte à un moment ultérieur t', le composant *Desire* associé à cette perception précisément, n'a plus de raison de rester actif. L'animat doit donc prendre en compte ce genre de situations lors de la mise à jour de son état interne.

C'est précisément à cette étape que le composant *Scheduler* intervient. Il évalue la validité de chaque composant *Desire* activé en se basant sur la fonction de test qui lui est associée. Cette fonction de test est une fonction booléenne, ce qui signifie qu'il existe deux scénarios possibles : si le composant est toujours considéré comme valide selon la fonction de test, il est maintenu dans la file d'attente ; cependant, s'il n'est plus valide, il est simplement retiré de la file d'attente.

Une fois que toutes ces mises à jour sont effectuées, l'animat dispose désormais d'une perception actualisée de son environnement et de ses objectifs. À ce stade, il doit prendre une décision sur la conduite à adopter en fonction de la situation actuelle, en s'engageant dans un processus de prise de décision.

3.4.3 La prise de la décision

Le processus de délibération, qui consiste à prendre une décision concernant le comportement que l'animat doit adopter à chaque instant, se décline en deux volets distincts.

Le premier volet s'applique lorsqu'aucun composant *Desire* n'est actif. Dans cette situation, le choix du composant *Desire* à exécuter est relativement simple. Il repose sur la file d'attente des composants *Desire* activé, qui à cette étape est mise à jour et triée en fonction du poids attribué à chaque composant. En conséquence, le composant *Desire* ayant le poids le plus élevé est sélectionné pour être exécuté.

Le second volet concerne les situations où un composant *Desire* est déjà en cours d'exécution. Dans ce cas, le choix du composant prioritaire ne se fait plus automatiquement en fonction de sa priorité dans la file d'attente, car le composant en cours d'exécution peut toujours être pertinent. Ainsi, une réévaluation de la priorité du composant *Desire* actif est nécessaire.

Cette réévaluation s'effectue en se référant à la fonction d'évaluation des priorités qui a été définie préalablement. Si aucune fonction d'évaluation des priorités n'a été définie, une fonction par défaut suivante est utilisée pour cette réévaluation.

$$f(d) = \frac{d_w + p_s}{(d_{td} - d_{tc})} \tag{3.1}$$

avec d le composant Desire actif, d_w son poids, p_s la force de la perception, d_{td} la durée totale de son composant, la durée d'exécution déjà écoulée d_{tc}.

En définissant ainsi la fonction d'évaluation des priorités, elle prend en compte plusieurs facteurs déterminants pour le choix. Elle permet à la fois :

- La priorisation de la durée d'exécution : Elle permet de donner la priorité aux composants *Desire* qui ont des tâches de courte durée par rapport à ceux qui ont des tâches plus longues. Cela garantit une meilleure réactivité de l'animat dans des situations où des actions rapides sont nécessaires.
- La priorisation du poids du composant : Les composants *Desire* ayant un poids important sont privilégiés par rapport à ceux dont le poids est moindre.
- La priorisation de la stimulation : Les composants *Desire* fortement stimulés sont favorisés par rapport à ceux qui le sont moins. Cela signifie que les actions en réponse à des stimuli forts ou urgents sont mises en avant ; ce qui peut être crucial dans des situations critiques.

Si un nouveau composant *Desire* est évalué comme prioritaire selon ces critères, l'interruption du composant *Desire* actuellement en cours d'exécution n'est pas systématique. La décision d'interruption ou de poursuite de l'exécution du composant *actif* est déterminée par un tirage aléatoire, où la probabilité d'interruption est définie par le facteur d'interruption (*interruption_factor*).

Si, malgré ce processus, aucun composant *Desire* n'est sélectionné, l'animat exécute alors la tâche par défaut.

L'ensemble du processus de délibération portant sur le composant *Desire* qui sera exécuté est présenté par l'algorithme 10.

Algorithme 10 : Algorithme de prise de décision exécuté pas le composant *Scheduler*

```
Données : Δ (ensemble des composants Desire de l'animat) ; timer (temps fourni par l'horloge)
queue ← [ ]                                  ▷ file d'attente du composant Scheduler;
tache ← tache_par_defaut ;
tant que animat en vie faire
    ▷ (1) Collecte des informations ;
    perceptions_détectées ← teste_des_perceptions ;
    pour chaque p dans perceptions_détectées faire
        desire_activables ← récuperer_desires(p, Δ)   ▷ Composants Desires de Δ activables par p;
        pour chaque d dans desire_activables faire
            si d.force_activation ≥ d.seuil alors
                ajouter(d, queue)                     ▷ le composant Desire d est activé;
        fin
    fin
    ▷ (2) Mise à jour des composants Desire ;
    si un composant Desire est actif alors
        si le Desire est actif est satisfait alors
            désactiver Desire est actif ;
        sinon
            réévaluer(Desire_actif, time)             ▷ Réévalaluer son importance à cet instant;
    ▷ Mise à jour de la file d'attente ;
    pour chaque dx dans queue faire
        si dx satisfait alors
            retirer(dx, queue);
        sinon
            réévaluer(dx) ;
    fin
    Trier(queue) ;
    ▷ (3) Prise de décision ;
    si queue vide alors
        si un composant Desire est actif alors
            Desire ← Desire_actif ;
        tache ← Desire.tache ;
    sinon
        Desire_choisi ← choisir(queue)   ▷ On choisi dans la file d'attente, Desire avec le plus grands
          poids;
        si un composant Desire est actif alors
            reevaluer(Desire_actif) ;
            si Desire_actif estprioritaire Desire_choisi alors
                x ← generer_un_nombre_entre(0, 1) ;
                si x ≤ Desire_actif.interruption_factor alors
                    Desire ← Desire_choisi ;
                sinon
                    Desire ← Desire_actif ;
            tache ← Desire.tache ;
    action ← choisir_action(tache, timer);
    ▷ (4)Exécution de l'action ;
    exécuter(action) ;
fin
```

3.4.4 L'exécution de l'action

Après avoir pris une décision à l'étape précédente, le composant *Scheduler* dispose désormais d'une tâche à exécuter, qui peut être soit celle d'un nouveau composant *Desire* activé, soit celle d'un composant *Desire* déjà activé et en cours d'exécution (continuation), ou encore il peut s'agir de la tâche par défaut. Chacune de ces tâches est composée d'actions qui doivent être exécutées les unes après les autres, dans un ordre spécifié.

Cependant, il est important de noter que chaque action a sa propre durée d'exécution ; ce qui signifie qu'elles ne se déroulent pas toutes simultanément ni de la même manière. Pour coordonner ces exécutions, le composant *Scheduler* fait usage d'une horloge interne qu'il intègre. Cette horloge est incrémentée à chaque cycle. Il compte le temps d'exécution des actions et temps d'exécution de la tâche elle-même. Il s'assure ainsi que chaque action soit exécutée au moment opportun, en respectant les délais et l'ordre préétablis.

Lorsque cette étape est achevée, cela marque la fin d'un cycle d'exécution pour l'animat, et l'horloge interne est mise à jour en conséquence.

À partir de ce moment, les données de simulation pour cet instant précis peuvent être sauvegardées. L'ensemble des données sauvegardées tout au long de la simulation sera finalement restitué à la fin du processus de simulation, permettant ainsi d'analyser et d'évaluer le comportement et les performances de l'animat dans son environnement.

Conclusion

Dans ce chapitre, nous avons introduit un nouveau modèle de comportement animal. Il s'inscrit dans notre démarche de modélisation du comportement animal en tant que problème d'optimisation, une proposition qui nécessite un modèle compatible avec l'intégration de métaheuristiques. Le modèle se compose des composants suivants :

Le composant *Perception* dont le rôle est de doter l'animat des capacités lui permettant d'acquérir des informations de son environnement, y compris de lui-même.

Le composant *Desire* représente les unités de comportement. La définition de ces types de composants permet au modélisateur de spécifier les mécanismes de déclenchement de l'unité de comportement, ainsi que l'ensemble des actions réalisées lorsque cette unité est déclenchée. Cela se fait au moyen de deux autres composants, *Task* et *Action*.

Avec cette définition du modèle, une proposition d'intégration dans les systèmes multi-agents a été formulée. Cette initiative a conduit au développement d'une plateforme de modélisation dédiée accompagnée d'un simulateur également dédié.

Ces éléments posent ainsi les bases nécessaires pour la mise en œuvre de notre approche de modélisation guidée par les métaheuristiques.

CHAPITRE 4

Modélisation du comportement animal guidée par les métaheuristiques

Sommaire

Introduction

Dans les chapitres 1 et 2 faisant office d'état de l'art, nous avons pu constater que la modélisation du comportement animal peut être très contraignante et fastidieuse, depuis les phases de préparation jusqu'aux phases opérationnelles, sans oublier la phase de conception proprement dite. Même si ces contraintes sont souvent levées, cela se fait au prix d'un compromis entre précision et explicabilité.

Fort de ces éléments, nous avons proposé une nouvelle approche de modélisation du comportement animal. Cette approche repose sur des principes historiques et philosophiques apparus dans l'évolution de l'étude du comportement animal à la fois du côté de la biologie et de l'informatique. Elle s'appuie sur les connaissances variées (éthogrammes, modèles existants, équations) existantes sur différentes espèces afin de générer des modèles entièrement explicables.

Cette manière de représenter le comportement animal nous permet d'envisager une nouvelle approche de génération des modèles. Cette approche repose sur des méthodes d'optimisation, particulièrement les métaheuristiques. Dans un processus incrémental et itératif guidé par un expert, ces méthodes cherchent à trouver la combinaison optimale d'actions élémentaires qui reproduit au mieux le comportement de l'animal, représenté par les données de références fournies par l'utilisateur.

Avec cet ensemble de concepts que nous proposons, nous pensons pouvoir créer un outil complet d'aide à la compréhension du comportement animal. Cet outil pourra proposer des modèles pour des espèces dont la connaissance est souvent incomplète du fait de leur complexité ou de la rareté des données qui leur sont liées. Cette démarche repose sur une perspective analogue à celle de René Descartes, qui avançait qu'il était possible de comprendre le comportement d'un être vivant (l'homme) en se basant sur l'observation d'un autre être vivant (un animal). Dans cette optique, nous pensons qu'il est envisageable de modéliser le comportement d'une espèce animale en utilisant les informations et les données disponibles concernant une espèce similaire. Par exemple, si nous disposons d'une compréhension approfondie du comportement d'une espèce A, nous pouvons utiliser ces connaissances comme point de départ pour mieux comprendre le comportement d'une espèce B, même si nos informations sur cette dernière sont limitées.

Dans ce chapitre, nous présentons cette approche en commençant par exposer les éléments qui l'ont motivée et justifiée. Ensuite, nous décrivons le processus proposé pour la modélisation guidée par les métaheuristiques, en mettant en évidence ses principaux composants.

4.1 Modélisation guidée par l'optimisation

L'approche de modélisation guidée par optimisation que nous proposons est représentée par la figure 4.1.

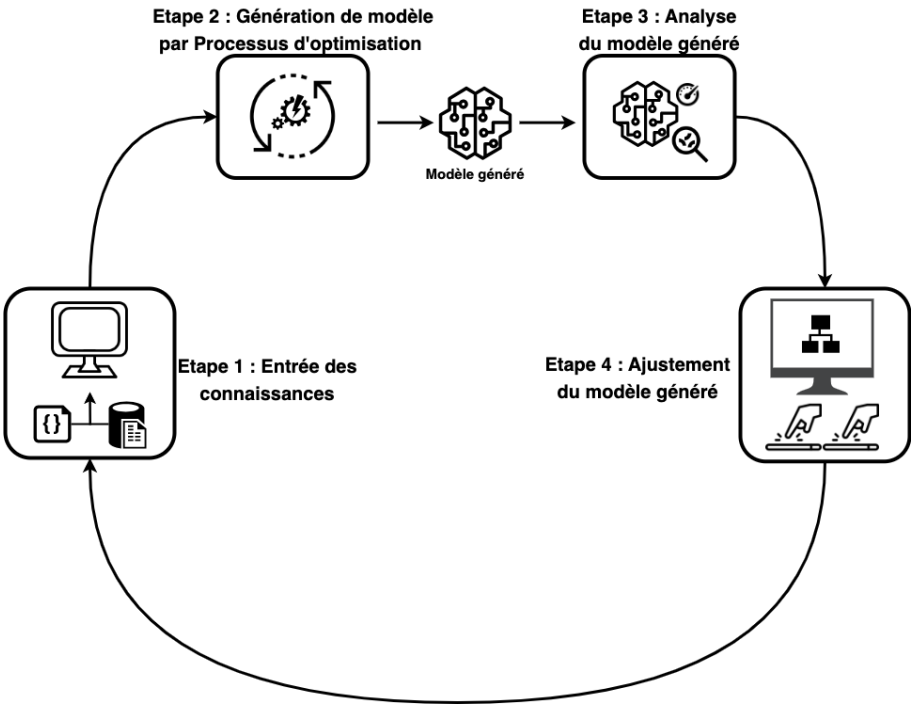


FIGURE 4.1 – Approche proposée pour la modélisation guidée par optimisation

4.1.1 Description du processus de modélisation

L'approche proposée repose sur un processus itératif et incrémental composé de quatre étapes principales que sont : l'entrée des connaissances, la génération de modèle par optimisation, l'analyse et l'ajustement du modèle généré.

4.1.1.1 Étape 1 : Entrée des connaissances

Il s'agit de l'étape où le concepteur du modèle introduit dans le système les connaissances et les données dont il dispose sur l'espèce à modéliser. Ces connaissances s'étendent sur deux volets :

1. Les informations qui renseignent sur le comportement de l'espèce à modéliser. Elles sont principalement extraites des éthogrammes, ou ce sont des règles de comportement tirées d'observations faites. Concrètement, il peut s'agir d'un ensemble d'actions possibles ou non, ou d'un intervalle réalisable pour les paramètres d'une action. Pour illustrer cela plus concrètement, considérons cet exemple : supposons que nous cherchons à modéliser le comportement d'une espèce de poisson. À partir des éthogrammes existants ou des observations, nous pouvons déjà identifier un ensemble d'actions possibles que ce

poisson peut faire et de la même manière ce qu'il ne peut pas faire. Par exemple, nous savons bien que les poissons ne peuvent pas voler, ce qui veut dire qu'ils ne peuvent pas se déplacer dans les airs comme le font les oiseaux ou les insectes. Par conséquent, toute action potentielle liée au vol peut être exclue des informations à entrer à cette étape. C'est également la même situation pour ce qu'il en est des intervalles réalisables pour les paramètres d'une action. Il est également tout à fait établi par exemple qu'une tortue ne pourrait aller à des vitesses allant jusqu'à 80km/h comme certains félins. À cet effet, dans le cadre de la modélisation du comportement de cette espèce, l'intervalle de vitesse peut alors être précisé.

L'entrée de ces données par l'utilisateur a un double objectif. Premièrement, les informations entrées sont utiles pour guider le processus de génération de modèle et contribuent à l'obtention d'un modèle avec un niveau de précision amélioré. Deuxièmement, elles réduisent considérablement le temps de calcul puisque l'espace de recherche se trouve réduit. L'entrée de ce type de connaissance n'est pas obligatoire pour générer un modèle. Si aucune connaissance n'est disponible, la génération du modèle devra prendre en compte tout l'espace de recherche défini par toutes les actions de l'ensemble, de même que pour tous les paramètres qui leur sont associés. De toute évidence, cela affectera les performances du processus de génération de modèle, car la complexité serait accrue.

2. Le deuxième volet concerne les données liées directement au comportement que nous cherchons à modéliser. Ces données revêtent une importance cruciale, car elles jouent un rôle central dans l'évaluation de la similarité entre le modèle généré et l'espèce réelle qu'il représente. Contrairement au premier volet qui concerne les connaissances d'ordre général qui elles sont facultatives, ici les données de comparaison sont obligatoires et incontournables. Les données dont nous parlons ici peuvent se présenter sous différentes formes. Elles peuvent consister en une série temporelle enregistrant l'évolution d'une caractéristique de l'espèce (ex : tailles, hormones, population) ou encore des données de géolocalisation (coordonnées GPS collectées à l'aide de réseaux de capteurs sans fil (Handcock et al. 2009 ; Karunanithy et al. 2022)). Étant donné que ces données servent de référence pour évaluer la qualité du modèle généré, il est vivement recommandé de veiller à ce qu'elles soient les plus représentatives et les plus exhaustives possibles, afin de garantir une évaluation pertinente du modèle généré.

Une fois que toutes les connaissances et données disponibles sont entrées dans le système, on passe à l'étape suivante de génération de modèle.

4.1.1.2 Étape 2 : Génération de modèle par optimisation

Cette étape revêt une importance cruciale dans notre approche, car elle est chargée de générer un modèle à l'aide de méthodes de résolution de problèmes d'optimisation. Ici, sont implémentés divers algorithmes d'optimisation que sont spécifiquement dans notre cas les métaheuristiques. Le processus de résolution du problème d'optimisation se déroule de manière itérative, en suivant les étapes définies dans notre approche (figure 4.1). En utilisant les

connaissances entrées à l'étape précédente [1], nous créons plusieurs combinaisons d'action à partir d'une base de données préalablement constituée, qui regroupe des actions élémentaires extraites d'éthogrammes, de modèles existants ou d'observation.

Chaque combinaison ainsi formée est simulée puis comparée (évaluée par rapport) à des données de référence directement à l'aide d'une fonction d'évaluation définie à cet effet. On parle d'optimisation via simulation (Andradóttir 1998), une approche dite d'«essais-erreurs » qui permet de tester plusieurs scénarios (Baccouche 2012). La combinaison formée (une solution) est ensuite améliorée itérativement, suivant la règle de la métaheuristique utilisée jusqu'à ce que nous parvenions à trouver la combinaison optimale d'actions avec les paramètres optimaux qui correspondent le mieux aux données de référence. À ce stade, ce modèle trouvé est explicable et peut être utilisé pour effectuer des simulations. Cependant, pour garantir sa cohérence et sa validité, une troisième étape est prévue, entièrement dédiée à son analyse.

4.1.1.3 Étape 3 : Analyse du modèle généré

La génération de modèles de comportements d'animaux avec cette approche que nous proposons se fait de façon entièrement automatisée, sur la base des informations fournies par le concepteur, qu'il s'agisse de connaissances pour guider l'optimisation ou de données de références pour l'évaluation des modèles générés à chaque itération. Il est vrai que le fait de laisser la génération du modèle totalement à la machine limite l'introduction éventuelle de biais par l'humain. Néanmoins, cela ne saurait être une raison pour l'écarter du processus de génération de modèle, mais bien au contraire. En fin de compte, c'est l'humain qui utilisera le modèle généré, et donc il est le mieux placé pour juger de la cohérence et de la pertinence des résultats produits. C'est précisément l'objectif de cette troisième étape. Le modèle généré est soumis à l'expertise humaine de la personne qui possède une connaissance approfondie du comportement de l'espèce modélisée.

Le modèle généré est soumis à l'appréciation de l'humain qui est ici l'expert du comportement de l'espèce modélisée. Le modèle généré étant explicable, l'humain dispose de toutes les informations nécessaires pour confirmer ou infirmer le modèle qui lui est présenté sous un format intuitif et qui facilite son analyse. Cette phase d'analyse peut conduire à la conservation de certaines parties du modèle tout en éliminant d'autres (par exemple, en ne conservant que les actions jugées cohérentes), ou elle peut susciter des idées pour des recherches futures. En plus de confirmer ou d'infirmer le modèle, cette étape permet également à l'expert de détecter et de résoudre d'éventuels problèmes d'équifinalité, c'est-à-dire des situations où différentes combinaisons d'actions produisent des résultats similaires.

À la fin de cette phase, deux scénarios se présentent :

— Le modèle généré satisfait les attentes du concepteur ou de l'expert : Dans ce cas il est sauvegardé dans un format directement utilisable.
— Dans le cas contraire, des modifications sont identifiées, et cela nécessite le passage à une quatrième et dernière étape du processus.

1. si elles sont disponibles puisque leur saisie n'est pas obligatoire

4.1.1.4 Étape 4 : Ajustement du modèle généré

Cette étape est une continuation de la précédente. Ici, les modifications identifiées peuvent être concrètement mises en œuvre à l'aide d'une interface homme-machine qui facilite les ajustements souhaités. À la fin de cette étape, nous obtenons un pseudo-modèle qui peut contribuer, de différentes manières, à l'acquisition de nouvelles connaissances sur l'espèce. C'est d'ailleurs pour ça que nous qualifions notre méthode d'approche «d'aide à la compréhension» du comportement animal. Les connaissances de base introduites à l'étape 1 sont désormais plus riches. Il s'agit là de l'aspect incrémental de notre approche, car à la fin de cette itération, nous avons atteint un niveau de connaissances plus avancé que celui du départ. Un nouveau cycle (itération) peut ensuite être initié à partir de l'étape 1, en prenant en compte les nouvelles connaissances acquises.

L'exécution de plusieurs itérations de l'ensemble de ces étapes permet de produire en sortie un modèle explicable, dont la construction s'est déroulée de façon incrémentale avec l'aide de l'expert.

Pour concrétiser notre proposition d'approche de modélisation et mettre en œuvre chacune des étapes présentées, nous devons développer une architecture dédiée qui repose sur des composants et des concepts fondamentaux que sont le composant *PartialElement* et le module d'optimisation.

4.1.2 Le composant «PartialElement»

Rappelons-le une fois encore que l'objectif principal de l'approche que nous proposons est de permettre la génération de modèles éthologiques explicables, basés sur des connaissances pouvant être complètes, partielles ou même inexistantes. Étant donné que ces connaissances peuvent provenir de diverses sources et revêtir différentes formes, il est nécessaire de les uniformiser. Dans cette optique, nous introduisons un composant générique appelé *PartialElement*.

Le composant *PartialElement* est défini sur la base de la représentation de l'animat afin d'assurer une cohérence dans la conception du modèle. Son rôle est de permettre la représentation partielle des composants de l'animat pour lesquels les données sont incomplètes. Le composant *PartialElement* est un élément générique qui peut donc être spécialisé pour chaque composant spécifique de l'animat, comme présenté dans le tableau 4.1. Les composants *PartialElement* proposés sont présentés par le diagramme UML des classes de la figure 4.2. En les définissant ainsi, les composants de *PartialElement* sont capables de fournir des indications utiles pendant le processus de génération de modèles.

Pour illustrer cette idée, prenons l'exemple de la génération d'un modèle de fuite d'une espèce quelconque. Supposons que nous sachions que cette espèce fuit en cas de détection d'un prédateur, mais nous ne connaissons pas la valeur exacte de sa vitesse de fuite. Cependant, grâce à des observations faites dans le milieu naturel, nous pouvons estimer un intervalle dans lequel se situe cette vitesse. Lors de la définition du composant *Desire* décrivant ce comportement de fuite, nous pouvons le définir comme un *PartialElement*, spécialisé en *PartialDesire* en y indiquant l'intervalle pour la vitesse. La valeur exacte de la vitesse sera déterminée pendant de la phase d'optimisation du processus de génération du modèle. Cet exemple souligne également l'importance d'inclure un expert du comportement étudié dans

TABLE 4.1 – Correspondance de chaque composant du modèle avec les composants *PartialElement*

Composant de l'animat	**Équivalent en *PartialElement***
Model (Animat)	PartialModel
Desire	PartialDesire
Task	PartialTask
Action	PartialAction
Paramètres	PartialKnowledge

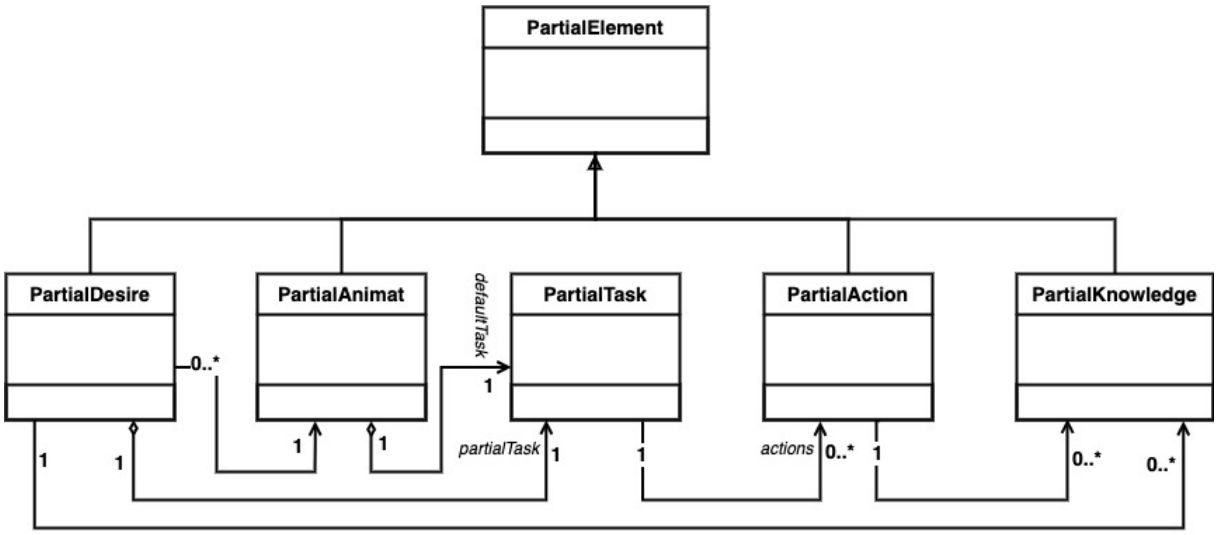


FIGURE 4.2 – Diagramme des classes UML des composants *PartialElement*

la boucle itérative de génération de modèle, étant donné son expertise, il est le mieux placé pour confirmer ou infirmer la valeur trouvée.

Selon le composant qu'on souhaite représenter partiellement, on peut apporter des précisions diverses. Concrètement, pour définir un composant incomplet de l'animat, on peut utiliser :

4.1.2.1 Composant *PartialKnowledge*

Un composant *PartialKnowledge* est utilisé pour définir un paramètre dans la configuration d'un composant de l'animat. Le composant *PartialKnowledge* est défini par les éléments suivants :

— Un *type* qui spécifie l'ensemble dans lequel le paramètre sera recherché, que ce soit l'ensemble des entiers, des réels, une matrice (entière ou réelle). Si le type n'est pas défini, l'ensemble des nombres réels est considéré par défaut.

— Les bornes inférieures ou supérieures d'un intervalle dans lequel la valeur doit être recherchée. La définition des deux bornes n'est pas obligatoire ; seule celle qui est définie

sera prise en compte. Par défaut, l'intervalle est $]-100,00[$.

4.1.2.2 Composant *PartialAction*

Le composant *PartialAction* est utilisé pour définir un composant *Action*. il est défini par les éléments suivants :

- Une *durée* qui peut accepter deux types de valeurs. Soit il s'agit d'un nombre pour définir directement la durée d'exécution de l'action lorsque celle-ci est connue, soit il s'agit d'un composant *PartialKnowledge* qui précise dans quel ensemble chercher la durée si elle n'est pas connue.
- Une *liste de composants PartialKnowledge* qui fournit des précisions sur les paramètres nécessaires à la configuration de l'action. Lorsqu'un paramètre est connu, sa valeur est renseignée directement. Sinon, il est représenté par un composant *PartialKnowledge*.

4.1.2.3 Composant *PartialTask*

Le composant *PartialTask* pour définir un composant *Task*. Le composant *PartialTask* est défini par :

- Une *liste de PartialAction* pour spécifier les actions qui composent la tâche. Cette liste n'est pas nécessairement utilisée strictement, et l'ordre des actions importe peu. Elle représente simplement les actions potentielles dont nous avons connaissance et qui peuvent intervenir dans la tâche. Les conditions d'utilisation de cette liste sont définies par les autres propriétés du composant *PartialTask*. Il n'est évidemment pas obligatoire de la définir, tout comme d'ailleurs les autres propriétés d'un composant *PartialElement*.
- Un *entier positif* précise le nombre d'actions qui composent la tâche. Si ce nombre n'est pas connu, il peut être défini par un composant *PartialKnowledge*. Cette propriété définit en partie les conditions d'utilisation de la liste des composants *PartialAction*. Si la taille de la liste est supérieure au nombre défini, alors les actions seront choisies dans la liste. Sinon, toutes les actions de la liste seront utilisées et complétées si besoin par d'autres actions provenant de la base de données d'actions.
- Une *table* pour associer une action de la liste à une position précise dans la séquence de la tâche. Cela permet de fixer l'ordre d'exécution des actions, ce qui est important, comme cela a été démontré dans la section 1.2.2.5.
- Une *valeur* booléenne précisant si les actions dans la liste doivent être utilisées exclusivement, sans faire appel à d'autres actions de la base de données.
- Une *valeur* booléenne précisant si une même action peut apparaître plusieurs fois dans la séquence ou non.

4.1.2.4 Composant *PartialDesire*

Le composant *PartialDesire* pour définir un composant *Desire*. Le composant *PartialDesire* est défini par :

— Trois paramètres qui représentent chacun le poids, le seuil et le coefficient d'interruption du composant *Desire*. Comme tout paramètre, si la valeur est connue, elle est affectée directement, sinon elle est décrite par un composant *PartialKnowledge*.

— Un composant *partialTask* pour définir la tâche associée au composant *Desire*.

Avec l'aide de ces différents composants de type *PartialElement*, nous offrons aux concepteurs de modèles la possibilité de fournir les connaissances disponibles sur l'espèce à modéliser, qu'elles soient complètes ou non, tout en restant cohérent avec la représentation proposée pour l'animat.

Considérons la modélisation de la tâche par défaut d'un animat, pour laquelle nous savons qu'en l'absence de toute stimulation, son action par défaut est de rester immobile pendant un certain laps de temps dont nous ne connaissons pas la durée exacte. Ce que nous savons, c'est que ladite durée ne peut pas dépasser 5 unités de temps. Cette tâche par défaut (*Default Task*) pourra être représentée conformément au diagramme d'objet UML de la figure 4.3.

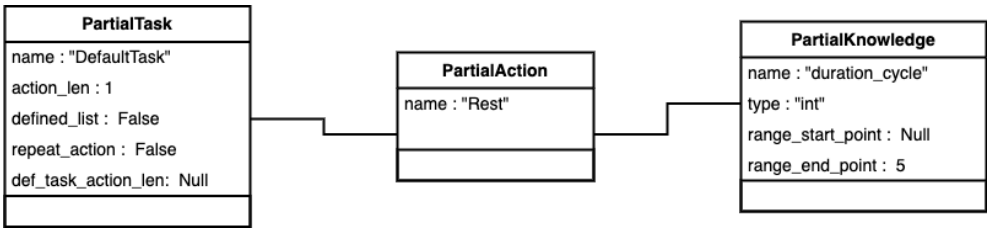


Figure 4.3 – Un exemple de représentation partielle d'une tâche avec un diagramme d'objet UML

L'information incomplète est la durée de l'action. La tâche par défaut est donc représentée par un composant *PartialTask*. À l'intérieur de celui-ci, on trouve un composant *PartialAction* qui représente l'action *Rest*, avec une durée paramétrée par un composant *PartialKnowledge*. Ce dernier, à son tour, est défini comme un entier avec une valeur maximale de 5.

Avec ces éléments en considération, concevoir un modèle impliquerait que le concepteur traduise les connaissances qu'il possède sur l'animal dans le format spécifié par l'architecture fonctionnelle (la figure 3.1). Lorsque des informations sont manquantes, le composant concerné est remplacé ou complété par un composant *PartialElement* correspondant. C'est là qu'intervient l'autre élément fondamental de notre approche qui a à charge de déterminer les éventuelles parties manquantes. Il s'agit du module d'optimisation.

4.1.3 Le module d'optimisation

Dans la hiérarchie de l'approche proposée, le module d'optimisation intervient à l'étape 2, qui est celle de la génération du modèle, en étant d'ailleurs l'unique acteur de cette étape. Pour générer les modèles, le module d'optimisation intègre des outils et algorithmes nécessaires à leur construction «intelligente». La figure 4.4 présente les trois entités qui forment le module d'optimisation ainsi que les interactions qui existent entre elles. Ces trois sous-modules sont le *Parser*, l'«optimiseur» et le module d'évaluation.

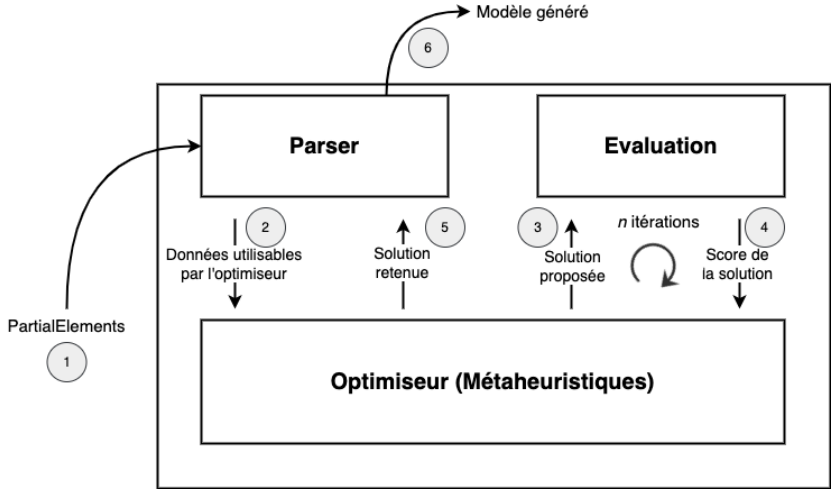


FIGURE 4.4 – Schéma synoptique du module d'optimisation

4.1.3.1 Le module «Parser»

Il s'agit d'une entité modélisée comme une classe qui joue deux rôles fondamentaux : la vérification des données d'entrée et leur interprétation dans la perspective de les rendre utilisables dans le processus de génération de modèle.

En effet, comme nous avons pu le voir dans la section 4.1.2 avec les composants *PartialElement*, les données entrées sont sous un format, parfois incomplet, et bien qu'ils soient unifiés, leur définition est laissée au soin de l'utilisateur. Dans ces conditions, des erreurs peuvent s'y insérer et des composants peuvent être incorrectement définis.

Le premier rôle du module «Parser» est de procéder à la validation de l'animat partiellement construit par l'utilisateur. Plus spécifiquement, le module Parser est implémenté comme une classe avec des méthodes qui lui permettent, pour chaque composant de l'animat, de vérifier s'il est exploitable dans le processus de génération. Il identifie et renvoie les parties incorrectes. En vérifiant ainsi chaque composant de l'animat, on parvient par conséquent à vérifier la validité de l'animat dans sa globalité.

Le second rôle, qui n'est pas moins important, joué par le composant «Parser», est celui de l'interprétation, voire de la traduction des données entrées par l'utilisateur. La nécessité d'avoir une telle fonction est constamment présente, car l'utilisateur insère lui-même des données. Le format dont il se sert est conçu pour lui être accessible et compréhensible pour faciliter la conception du modèle. Cependant, ce format n'est pas directement exploitable par la machine pour construire le modèle attendu. De même, lorsque la machine termine son processus, elle doit le rendre dans un langage compréhensible pour l'humain. Indubitablement, il est nécessaire d'avoir un intermédiaire entre l'humain et la machine, capable de rendre l'information accessible aux deux parties.

Une fois que les données ont été vérifiées, validées et traduites dans un format utilisable pour la construction des modèles, le processus d'optimisation peut alors commencer avec le module «optimiseur».

4.1.3.2 L'optimiseur

L'optimiseur englobe les méthodes d'optimisation, dites métaheuristiques, qui sont utiles pour résoudre le problème d'optimisation afin de générer le modèle. Pour remplir ce rôle, l'optimiseur s'appuie sur deux modules : le module des métaheuristiques compatibles avec la génération de modèles et le module d'évaluation des modèles (les solutions) générés.

L'implémentation de métaheuristiques adaptées à ce type de problèmes d'optimisation est l'une des principales contributions de nos travaux. Nous la présentons plus en détail dans la section 4.2. En ce qui concerne l'évaluation des modèles générés, elle est réalisée par le module d'évaluation.

4.1.3.3 Le module d'évaluation

Par définition, les métaheuristiques sont des méthodes qui partent d'une ou plusieurs propositions qu'elles cherchent à améliorer. Pour qu'une proposition puisse être améliorée, il est essentiel d'être en mesure d'évaluer sa réponse au problème posé. C'est précisément le rôle du module d'évaluation, qui attribue un score à chaque modèle généré au cours de l'exécution des métaheuristiques. Pour ce faire, le module d'évaluation doit impérativement simuler chaque modèle soumis à son évaluation afin de comparer les résultats de la simulation aux données de références (passée à l'étape 1 de la figure 4.1), en utilisant les critères définis par la fonction d'évaluation.

Schématiquement, le module d'évaluation pourrait être considéré comme une entité qui reçoit en entrée une proposition formulée par une métaheuristique, et renvoie en sortie un score qui représente l'adaptation de ladite proposition par rapport au problème posé. Dans le module d'évaluation, on retrouve trois composants principaux : un adaptateur, un simulateur et des fonctions d'évaluation, comme illustrés dans la figure 4.5.

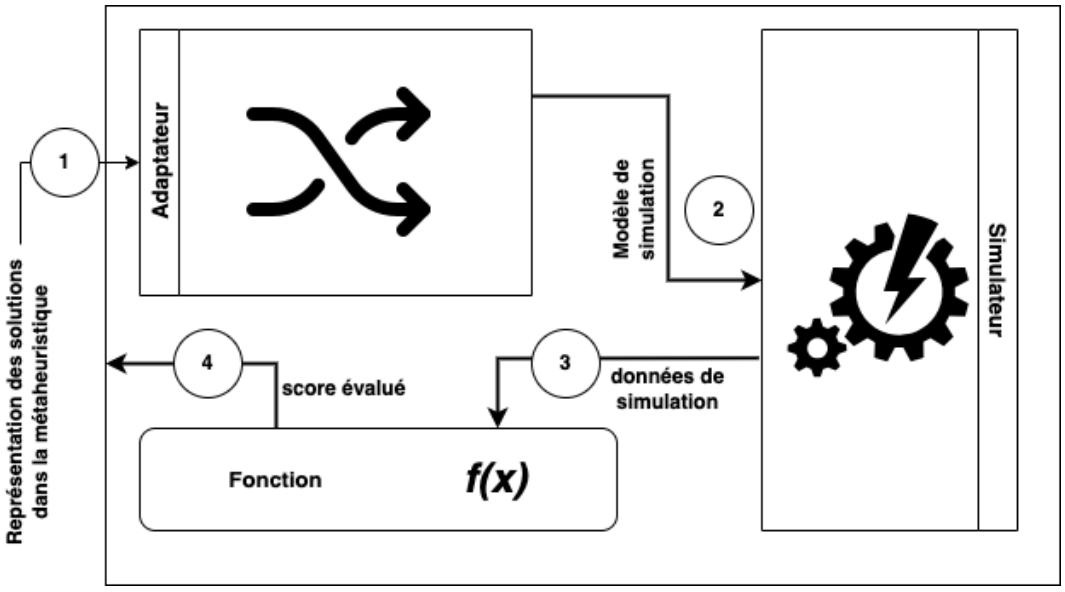


Figure 4.5 – Schéma synoptique du module d'évaluation

Comme nous l'avons présenté dans la section 2.4.1.1 et comme nous le verrons dans la section 4.2, où nous décrivons l'implémentation des métaheuristiques, les propositions peuvent avoir des représentations spécifiques propres à chaque algorithme (1). Ces différentes représentations de solutions ne peuvent pas être directement simulées. Ainsi, l'adaptateur a pour rôle de traduire ces représentations en modèles qui peuvent être simulés par le simulateur (2). À la fin de la simulation, le résultat (brut) est transmis à l'une des fonctions d'évaluation implémentées (3). Le score obtenu à l'issue de la comparaison avec les données de référence est ensuite renvoyé à la métaheuristique pour la poursuite du processus d'optimisation (4).

Tout au long de cette section, nous avons présenté notre proposition visant à construire de manière itérative et incrémentale des modèles de comportement animal. Compte tenu de la multiplicité de modules et de composants qui constituent notre approche, nous proposons à la figure 4.6 un schéma qui les hiérarchise et illustre les liens qui les relient. Elle montre que l'approche proposée se déroule en 4 étapes.

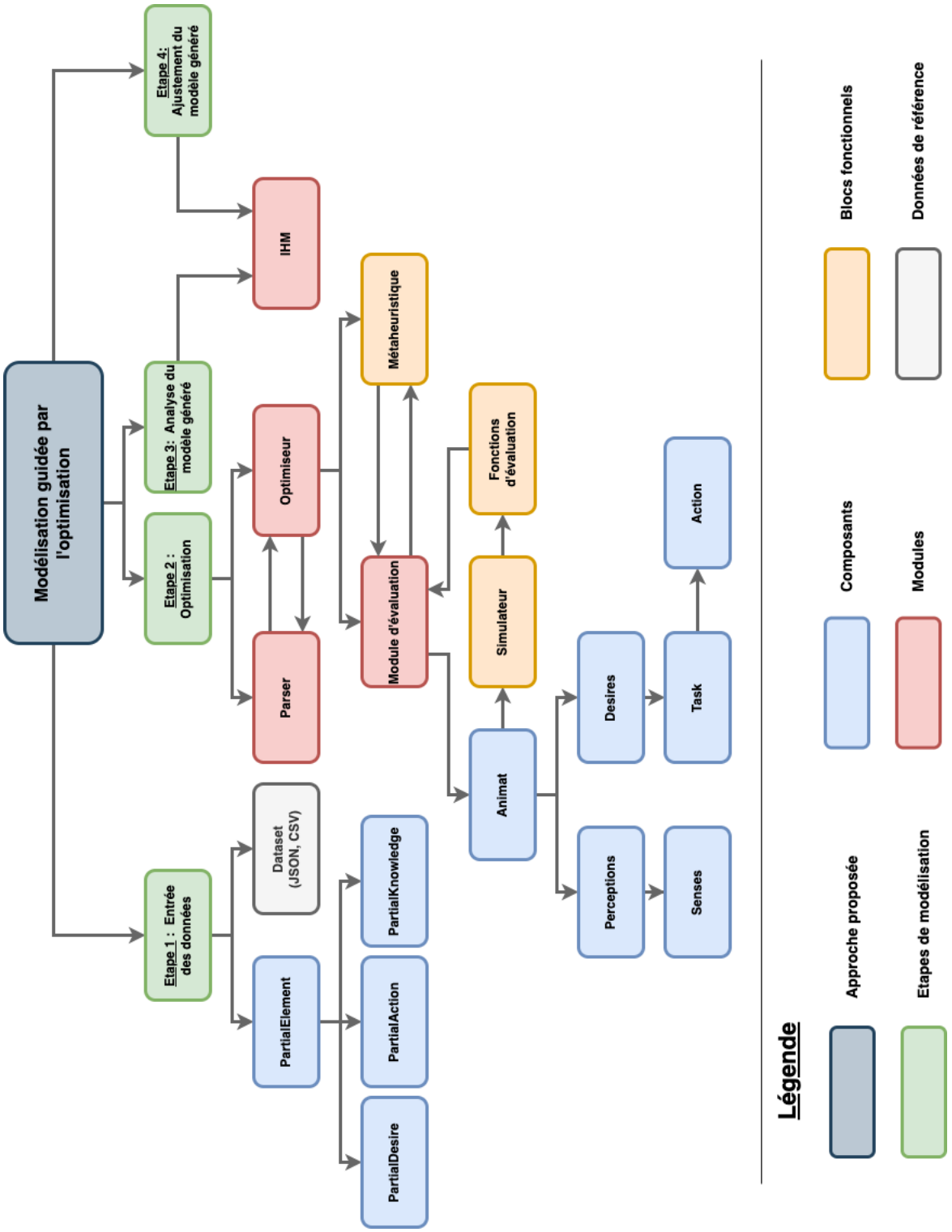


FIGURE 4.6 – Composants constituant l'approche proposée et les liens entre eux

— À l'étape 1, le concepteur entre les connaissances et les données dont il dispose. D'une part, il s'agit des connaissances partielles renseignées au moyen des composants *PartialElement*, et d'autre part, il s'agit des données de référence collectées souvent à partir de réseaux de capteurs sans fil.

— À l'étape 2, le processus d'optimisation est exécuté. Il repose sur deux modules principaux, à savoir le *Parser* et l'*Optimiseur*. Le *Parser* se charge de rendre les données exploitables pour chaque niveau en fonction des besoins, tandis que l'*Optimiseur* génère les modèles à l'aide de métaheuristiques. Chaque modèle est généré sous forme d'un animat qui est ensuite simulé et évalué respectivement avec le simulateur et les fonctions d'évaluation.

— À l'étape 3, le modèle généré est présenté à l'expert et soumis à son analyse à travers une IHM conçue à cet effet.

— À l'étape 4, les éventuels ajustements identifiés à l'étape 3 sont effectués toujours à travers l'IHM. Le pseudo-modèle obtenu à ce stade peut à son tour connaître des améliorations avec une nouvelle itération.

L'une des composantes essentielles de notre approche étant l'utilisation des métaheuristiques dans un contexte de génération de modèle de comportement animal, nous détaillons dans la section suivante les adaptations réalisées dans ce cadre.

4.2 Adaptation des métaheuristiques pour la génération de modèles de comportements d'animaux

Dans la section 2.4 nous avons présenté les métaheuristiques comme méthode de résolution de problèmes d'optimisation. Ils ont suffisamment prouvé leur efficacité pour des problèmes de grande complexité, ceci dans plusieurs domaines d'application (Poggi et al. 2022 ; Gogna et al. 2013 ; Simsir et al. 2019 ; Ezugwu et al. 2021).

En explorant toutes ces applications, nous remarquons que l'utilisation la plus courante qui est faite des métaheuristiques porte sur des problèmes d'optimisation où les résultats obtenus sont numériques. Dans les autres cas où les résultats attendus ne sont pas directement numériques, ils sont souvent ramenés à des résultats numériques (Gopalakrishnan 2013). À partir de ce constat, deux grands groupes d'utilisation des métaheuristiques semblent se dégager. Soit il s'agit de problèmes avec des variables à chercher sur un sous-ensemble de $\mathbb{R}$, soit ils sont à chercher dans un sous-ensemble de $\mathbb{N}$. Ce constat pourrait bien être considéré comme une conséquence directe et évidente de l'essence même des métaheuristiques, considérées comme des «méthodes d'optimisation des fonctions numériques» (Çimen et al. 2023 ; XS Yang 2011).

Cela est corroboré par le processus de validation d'une nouvelle métaheuristique, où les fonctions mathématiques utilisées ont un ensemble d'arrivée correspondant à un sous-ensemble de $\mathbb{R}$ (Wong et al. 2019) [2]. De plus, le déroulement des algorithmes eux-mêmes, notamment les opérateurs utilisés dans les phases d'intensification, s'appliquent généralement à des valeurs numériques. Prenons l'exemple de la métaheuristique inspirée par la conduite

2. Dans la littérature, on parle de fonction de Benchmark

des troupeaux d'éléphants (GG Wang et al. 2015). La mise à jour des propositions de solutions se fait conformément à la fonction décrite par l'équation 4.1, qui est typiquement applicable à des valeurs continues. Bien que des variantes pour l'optimisation discrète soient proposées, elles restent essentiellement basées sur la même philosophie et ne sont applicables qu'à des valeurs numériques.

$$x_{new,ci,j} = x_{ci,j} + \alpha(x_{best,ci} - x_{ci})r \tag{4.1}$$

où $x_{new,ci,j}$ et x_{ci} représentent respectivement la nouvelle position et l'ancienne position de la solution j dans l'ensemble ordonné ci des solutions proposées. $x_{best,ci}$ est la meilleure solution de ci et $(\alpha, r)^2 \in ([0,1])^2$ des paramètres de l'algorithme.

En ramenant l'utilisation des métaheuristiques dans notre contexte d'étude, nous sommes confrontés à une problématique majeure : l'application des métaheuristiques pour générer des modèles de comportement. En effet, le problème d'optimisation que nous cherchons à résoudre pour obtenir un modèle est un problème qu'on peut qualifier de mixte, à mi-chemin entre un problème combinatoire (discret) et un problème continu. La partie combinatoire du problème correspond à la formation des combinaisons d'actions, tandis que la partie continue elle, consiste à trouver le paramétrage associé à chaque action. Dans un tel cas, il n'est pas possible d'utiliser directement les métaheuristiques. Une solution consisterait à aborder le problème étape par étape en résolvant d'abord la partie combinatoire pour trouver les bonnes actions, ensuite la partie continue pour déterminer le bon paramétrage des actions choisies. Malheureusement, cette approche reviendrait à effectuer une optimisation locale et ne permettrait donc pas de trouver l'optimum global.

En plus de cet aspect hiérarchique (multi-niveaux») du problème, nous rencontrons une particularité exceptionnelle liée au nombre de variables qui peut différer pour des propositions de solutions d'un même problème. La figure 4.7 présente un exemple de génération de modèle qui met en évidence cette situation.

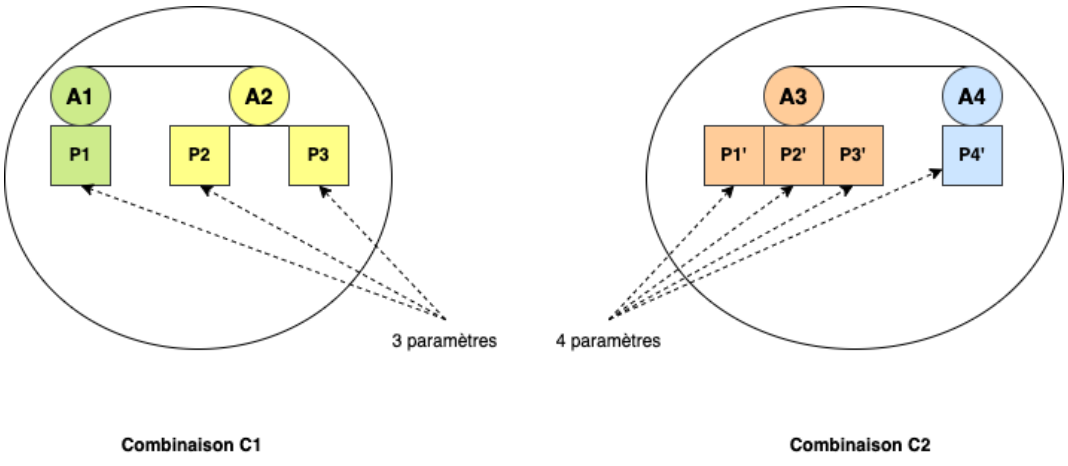


Figure 4.7 – Exemple de proposition de solutions avec des nombres de paramètres différents

Les combinaisons C_1 et C_2 sont obtenus par un même problème. Elles sont respectivement

constituées des couples d'actions (A_1, A_2) et (A_3, A_4). A_1, A_2, A_3 et A_4 acceptent respectivement 1, 2, 3 et 1 paramètre(s). Ainsi, la combinaison C_1 nécessite un paramétrage avec trois variables à optimiser, tandis que la combinaison C_2 en nécessite quatre. On a donc d'un côté 3 paramètres contre 4 de l'autre.

Compte tenu de toutes ces inadéquations, il est nécessaire d'adapter les métaheuristiques pour résoudre le problème d'optimisation devant aboutir à la génération d'un modèle de comportement animal. Pour ce faire, il est essentiel de redéfinir certains concepts spécifiques à chaque métaheuristique. En nous basant sur les classifications couramment utilisées dans la littérature, nous sélectionnons un algorithme par catégorie. Dans la catégorie des algorithmes évolutionnaires, nous optons pour l'algorithme génétique et l'algorithme de recherche harmonique. Dans la catégorie des algorithmes de colonies, nous choisissons l'algorithme de colonie de fourmis. Enfin, dans la catégorie des algorithmes bio-inspirés, nous optons pour l'algorithme des systèmes immunitaires par clonage.Le choix de ces algorithmes en particulier s'explique par leur utilisation répandue dans les problèmes d'optimisation.

4.2.1 Adaptations de l'algorithme génétique

L'algorithme génétique est sans aucun doute la métaheuristique la plus couramment utilisée en raison de son adaptation à un grand nombre de problèmes d'optimisation. Cependant, dans notre contexte d'étude, pour mettre en œuvre cet algorithme, il est nécessaire de redéfinir principalement deux concepts clés à savoir la représentation de la solution et l'opérateur de croisement.

4.2.1.1 La représentation de solution

Il existe plusieurs méthodes pour représenter les solutions lorsque l'on implémente un algorithme génétique. La plus courante parmi elles consiste à représenter l'ensemble des variables de décision sous forme d'une matrice de nombres binaires ou de nombres réels. Dans ces cas, le nombre de variables est connu et fixe, ainsi que la position de chaque variable dans la matrice ; ce qui facilite l'échange de portions de matrice lors de la phase de croisement.

Cependant, dans notre contexte, nous devons non seulement représenter des valeurs non numériques (ex : les actions), mais la taille et la position des variables ne sont pas fixes. Par conséquent, l'échange de portions de matrice devient soit impossible, soit aboutit à la génération d'enfants «incohérents» vis-à-vis du problème décrit par les composants *PartialElement*. Pour illustrer un cas d'enfants «incohérents», prenons l'exemple d'une portion de solution composée d'actions, comme indiqué dans la figure 4.8. L'action A_1, qui avait initialement deux paramètres, se retrouve après croisement avec trois actions, tandis que l'action A_2, qui avait trois paramètres, se retrouve avec deux.

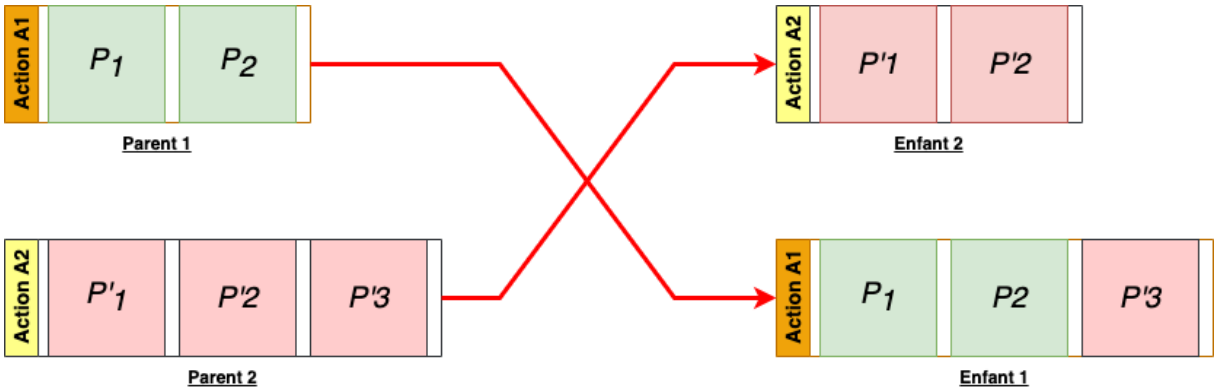


FIGURE 4.8 – Exemple de croisement donnant lieu à des solutions incohérentes vis-à-vis du problème

La solution apportée à cette problématique consiste à adopter une représentation sous forme d'arbre, comme illustré dans la figure 4.9. L'animat est représenté par un arbre hiérarchisé avec une racine d'où partent trois branches.

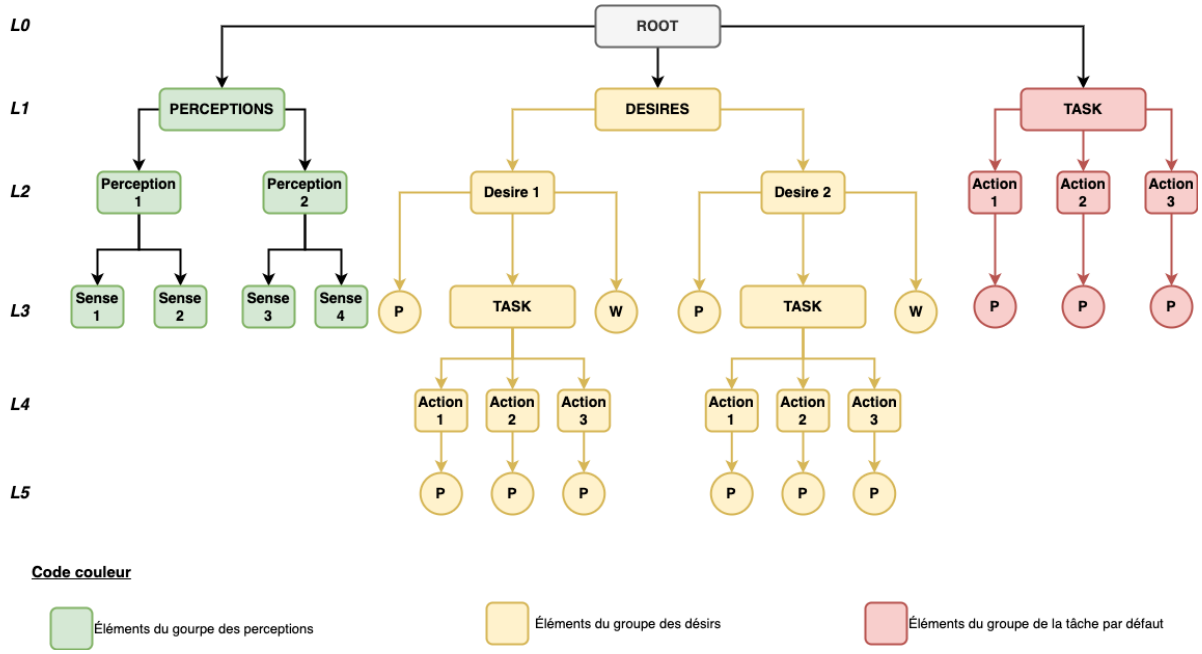


FIGURE 4.9 – Représentation d'une solution dans la version adaptée de l'algorithme génétique

— La première branche représente le groupe des composants *Perception*. Elle est reliée à un nœud de niveau 1 *(L1)* qui a pour successeurs les composants *Perception* de l'animat, eux-mêmes des nœuds de niveau 2 *(L2)*. Chaque composant *Perception* est directement associé à ses composants *Sense*.

— La deuxième branche conduit directement à un nœud de niveau 1 *(L1)* qui représente la tâche par défaut de l'animat. Une tâche étant une collection d'actions, les successeurs du nœud de niveau 2 *(L2)* représentent chacune des actions de la tâche. L'ordre d'exécution

des actions est défini par le parcours en longueur à partir du nœud «tâche». De même, les paramètres de chaque action sont les successeurs de niveau 3 *(L3)* du nœud qui leur est directement associé.

— La troisième branche représente le groupe des composants *Desire*. Le nœud de niveau 1 *(L1)* représente le groupe des composants. Chaque successeur de niveau 2 *(L2)* représente un composant *Desire* de l'animat. Étant composé de paramètres (poids, seuil, coefficient d'interruption) et d'une tâche, le nœud représentant le composant *Desire* a des successeurs directs qui représentent chacun de ces éléments. Tous ces éléments sont de niveau 3 *(L3)*. Le nœud représentant la tâche, a lui la particularité de posséder des nœuds avec deux niveaux supplémentaires que les actions de niveau 4 *(L4)* et leurs paramètres de niveau 5.

Le choix d'une nouvelle représentation sous forme d'arbre ajoute une couche supplémentaire d'encodage et de décodage au processus d'optimisation classique. L'encodage consiste à traduire les solutions en arbre, tandis que le décodage consiste à traduire les arbres en un modèle pouvant être simulé. Malgré cela, cette représentation apporte des solutions à toutes les contraintes posées par l'utilisation de l'algorithme génétique dans notre contexte d'étude.

Effectuer un croisement de solutions avec cette représentation revient donc à échanger des branches et des nœuds entre elles, comme présenté dans l'exemple de la figure 4.10. La représentation en arbre permet aussi de faciliter la représentation des solutions dans leur diversité. Même si les solutions ont des tailles différentes, grâce à la structure en arbre, nous pouvons toutes les représenter et identifier chaque nœud en fonction du type et des niveaux. Par exemple, nous saurons que tous les éléments de niveau 5 sont des paramètres d'une action, ou qu'un composant *Task* de niveau 1 représente la tâche par défaut. Ces informations sont essentielles pour la phase de croisement gérée par l'opérateur de croisement.

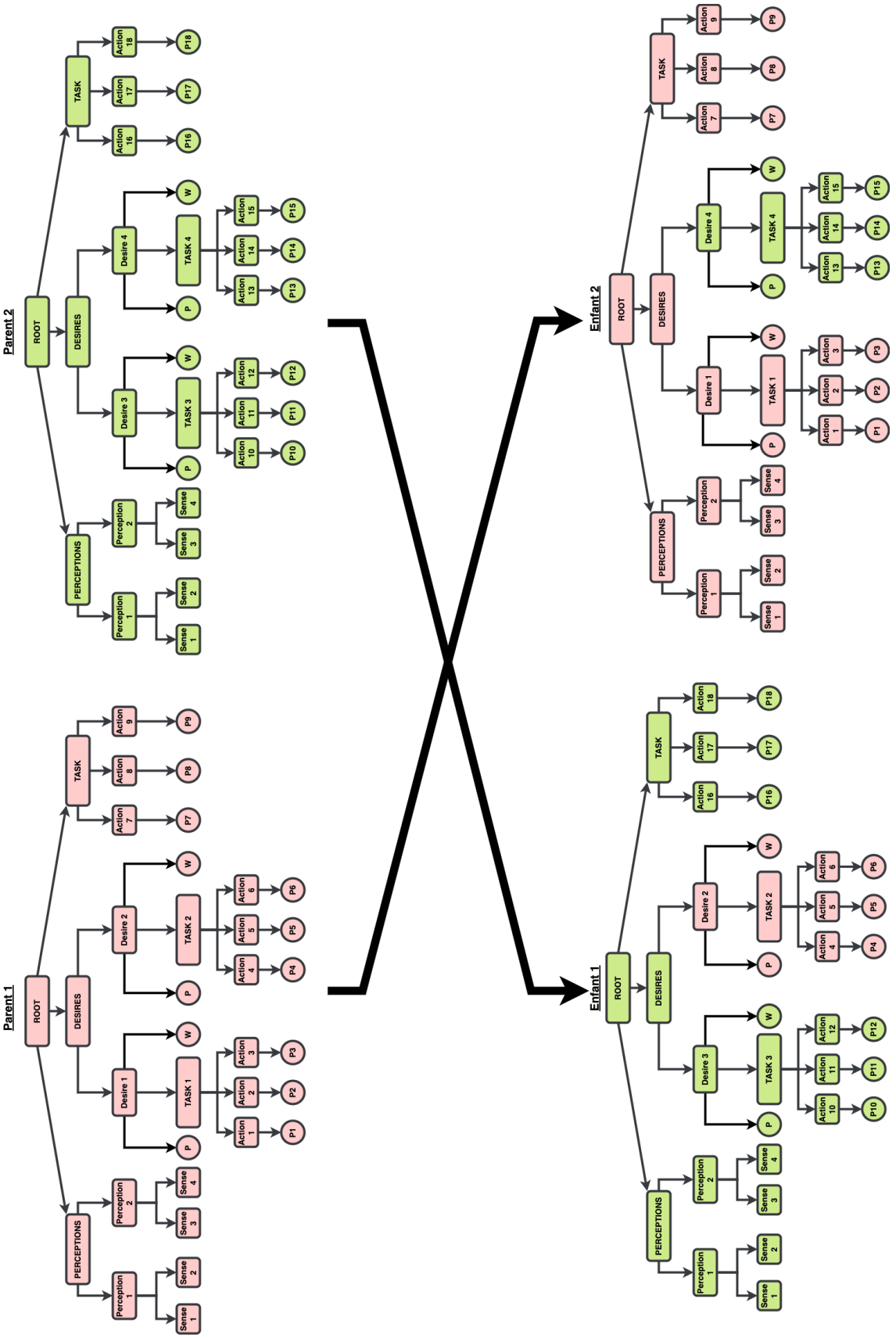


FIGURE 4.10 – Exemple croisement par échange de branche

4.2.1.2 L'opérateur de croisement

L'opérateur de croisement définit comment les croisements sont effectués, et donc comment les propositions de solutions sont améliorées. Dans notre cas, en représentant la solution sous forme d'arbre, le croisement se résume à un échange de branches. Cependant, malgré sa simplicité apparente, deux problèmes se posent.

1. Comment assurer la cohérence des «enfants» créés par rapport au problème après échanges de branches ?
2. Comment s'assurer que les nouveaux «enfants» sont «viables», c'est-à-dire être des animats qui peuvent être utilisés pour une simulation ?

Les solutions à ces deux questions sont fournies par deux niveaux de vérification que nous avons intégrés dans l'opération de croisement.

Le premier niveau consiste à établir des règles de croisement afin d'éviter de créer des combinaisons qui ne correspondent pas à un animat, par exemple un animat avec des composants *Perception* à la place des composants *Desire*. Les règles que nous avons définies sont les suivantes :

— *Les composants Perception et Desire ne sont pas échangés* : Cette règle répond à un besoin d'efficacité afin d'éviter des croisements inutiles et de limiter le temps de calcul. En effet, les composants *Perception* et *Desire* sont fournis par l'utilisateur dès le début du processus et ils restent les mêmes pour toutes les propositions de solutions. Il serait donc inutile de procéder à des échanges, car cela conduirait à des «enfants» identiques aux «parents».
— Seuls les nœuds de même type sont choisis comme points de croisement : cette règle permet d'éviter de créer des combinaisons qui n'ont pas de sens et qui ne représentent rien dans la logique d'exécution d'un animat.
— Les nœuds représentant les fonctions de test des composants *Desire* ne peuvent pas être modifiés afin d'éviter qu'un composant ne se retrouve avec une fonction de test qui n'est pas en adéquation avec les actions réellement effectuées.

Ces trois règles veillent à la cohérence structurelle des propositions de solutions après les croisements. À la fin de cette première vérification, nous avons l'assurance que chaque solution peut être simulée et donc évaluée.

Le deuxième niveau de vérification est inspiré du principe de la létalité de certains gènes. Un gène létal est un gène qui, lorsqu'il est présent dans le génotype d'un individu, cause des dommages graves et conduit généralement à la mort de l'individu. En d'autres termes, ces individus naissent, mais ne sont pas viables. Dans notre contexte, cela pourrait correspondre au cas où les propositions de solutions sont cohérentes avec la logique d'un animat en général, mais ne sont pas spécifiquement adaptées à l'animat à modéliser. Cela se produit notamment lorsque les nœuds «paramètre» sont utilisés et qu'ils sont remplacés par des valeurs qui ne correspondent pas à leur intervalle de définition.

Prenons l'exemple d'une tâche décrite par l'utilisateur comme étant composée de deux actions A_1 et A_2, chacune ayant un paramètre dans l'intervalle $[0, 1]$ et $[10, 100]$ respectivement. Lorsque ces nœuds sont échangés lors d'un croisement, nous obtenons une nouvelle proposition où l'action A_1 a un paramètre dans l'intervalle $[10, 100]$ et l'action A_2 dans l'intervalle $[0, 1]$. Cette nouvelle proposition est valide du point de vue d'un animat, mais elle n'est pas

cohérente pour ce problème spécifique. Pour gérer ces cas de croisement, nous procédons à un «contrôle de létalité» des solutions proposées. Ce contrôle consiste à vérifier si chaque solution proposée reste conforme à la description fournie par l'utilisateur en entrée c'est-à-dire, conforme aux composants *PartialElement*.

En tenant compte de tous ces facteurs, l'exécution de cet algorithme génétique adapté à notre problématique nécessite un certain nombre de paramètres, résumés dans le tableau 4.2. Ces paramètres peuvent être définis par l'utilisateur ou initialisés automatiquement par un générateur de paramètres implémentés dans l'algorithme.

TABLE 4.2 – Paramètres requis pour l'exécution de l'algorithme génétique

Paramètre	**Description**	**Domaine**
Taille de la population n	Définit le nombre de solutions qui constituent la population.	$n \in \mathbb{N}^{*+}$
Nombre de croisements n_c	Définit le nombre maximal de points d'échange de branches, lors des croisements. Avant chaque croisement le nombre exact est généré aléatoirement dans l'intervalle défini par le paramètre.	$n_c \in \mathbb{N}^{*+}$ (n_c doit être inférieur au nombre de variables de décision).
Nombre de nouveaux n^+	Définit le nombre de nouvelles solutions introduites dans la population à chaque itération.	$n^+ \in \mathbb{N}^+; n^+ < n$
Nombre d'itérations n_{it}	Définit le nombre maximal d'itérations d'exécution de l'algorithme.	$n_{it} \in \mathbb{N}^{*+}$
Probabilité de mutation p_m	Définit la probabilité de procéder à la mutation d'une nouvelle solution	$p_m \in [0, 1]$
Score limite f_{opt}	Définit le score seuil à atteindre ou à franchir pour qu'une solution soit considérée comme optimale.	$f_{opt} \in \mathbb{R}$

4.2.2 Adaptations de l'algorithme de colonie de fourmis

Le principe de résolution de problème d'optimisation par l'algorithme de colonie de fourmis consiste à faire parcourir un graphe (généré ou fourni) par des fourmis artificielles. Le chemin résultant de ce parcours est une solution au problème d'optimisation. Dans notre contexte d'étude, des adaptations spécifiques doivent être réalisées, notamment au niveau de la construction du graphe et de son parcours par les fourmis artificielles.

4.2.2.1 Construction du graphe

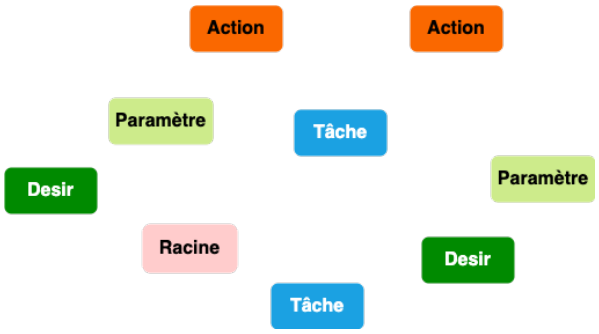
Comme décrit dans la section 2.4.3.6, la résolution effective du problème d'optimisation par l'algorithme de colonie de fourmis dépend fortement du graphe parcouru par les fourmis artificielles. Dans notre contexte, où le graphe n'est pas initialement fourni (comme c'est généralement le cas avec cet algorithme), nous devons le générer de manière à couvrir autant que possible l'espace de recherche de sorte qu'il maintienne une complexité acceptable.

Pour construire le graphe, nous générons aléatoirement un certain nombre d'animats, conformément à la description fournie par l'utilisateur à l'aide des composants *PartialElement*. Chacun de ces animats est décomposé en nœuds qui constitueront le graphe. À ce stade, il n'y a aucune connexion entre les nœuds. Leur création est conditionnée par des

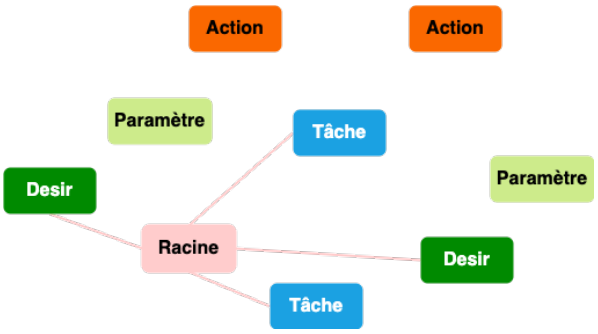
règles que nous devrons établir. En effet, tous les nœuds ne peuvent pas être connectés directement, car cela ne respecterait pas la logique d'exécution de l'animat. Par exemple, il ne peut pas y avoir de lien direct entre une tâche et une perception. Cela n'a pas de sens pour l'animat puisque dans son architecture fonctionnelle, il n'existe pas de lien entre un composant *Perception* et un composant *Task*. L'ensemble des règles établies pour réaliser les connexions sont présentées dans le tableau 4.3. Une illustration de la construction du graphe est proposée dans la figure 4.11.

TABLE 4.3 – Règles de connexion entre nœuds

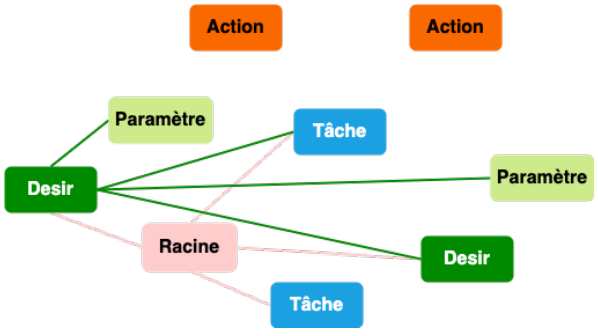
Nœud de départ	Nœud d'arrivée	Description
Racine	Task	Permet à une fourmi qui part de la racine d'accéder à des nœuds de tâche pour former la tâche par défaut
	Desire	Les fourmis qui partent de la racine ont accès aux nœud des composants *Desire*
Desire	Task	Lien pour constituer la tâche associée au composant *Desire*
	Paramètres	Accès aux paramètres pour constituer les paramètres (poids, seuil, coefficient) du composant *Desire*
Task	Action	constitution des actions de la tâche
Action	Paramètre	constitution des paramètres de l'action



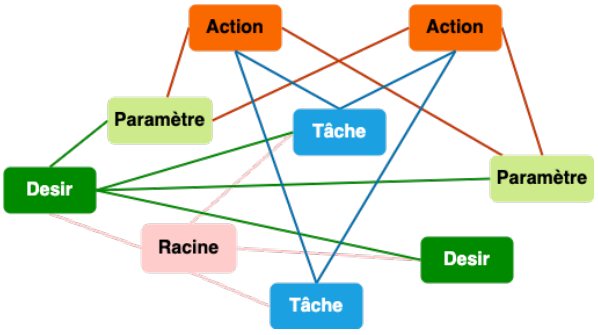
(a) Nœuds de base, provenant des solutions aléatoirement générées



(b) Etape 1 : connexion du nœud Racine aux nœuds tâches et aux nœuds désirs



(c) Étape 2 : connexion du nœud «Desire» aux nœuds «Task» et aux nœuds de «paramètre»



(d) Étape 3 et 4 : connexion des nœuds «Tasks» et «Action» respectivement aux nœuds «Action» et «Paramètre»

FIGURE 4.11 – Construction progressive de graphe pour l'algorithme de colonie des fourmis

Ainsi, pour construire ce graphe, nous générons aléatoirement un certain nombre d'animats, conformément à la description faite par l'utilisateur du modèle partiel. Chacun de ces animats est décomposé en nœuds qui constitueront le graphe. À ce stade, aucune connexion n'existe encore entre les nœuds, et il faudra les mettre en place en suivant certaines règles. En effet, tous les nœuds ne peuvent pas être connectés directement afin de respecter la logique d'exécution de l'animat. Par exemple, il ne peut pas y avoir de liens entre une tâche et une perception, car cela ne correspond à aucune logique de l'animat. Les règles que nous suivons pour réaliser les connexions sont présentées dans le tableau 4.3. Une illustration de la construction du graphe est proposée dans la figure 4.11. À la figure 4.11a nous avons les nœuds qui sont simplement placés. Aucune connexion n'est encore faite. À la figure 4.11b, la règle de connexion du nœud *Racine* est appliquée. Le nœud *Racine* est connecté aux nœuds *Task* et *Desire*. À la figure 4.11c, la règle de connexion des nœuds *Desire* est appliquée. Tous les nœuds *Desire* sont connectés aux nœuds *Task* et *Paramètre*. Enfin à la figure 4.11d les règles des nœuds *Task* et *Action* sont appliquées.

En construisant le graphe de cette manière, nous couvrons au mieux l'espace de recherche défini par le problème, et nous avons également l'assurance que toutes les solutions proposées sont logiques du point de vue d'un animat. Cependant, un problème persiste, celui de la cohérence des propositions avec les composants *PartialElement* définis par l'utilisateur. La solution à ce problème sera apportée par le fonctionnement des fourmis artificielles, notamment la manière dont elles parcourent le graphe.

4.2.2.2 Parcours du graphe par les fourmis artificielles

La construction d'une proposition de solution est faite par le parcours du graphe qu'effectuent les fourmis artificielles. Dans cette réalité, les liens dans le graphe représentent la possibilité pour une fourmi de passer d'un nœud à un autre. Pour cela, des règles de parcours générales sont définies dans l'implémentation des fourmis artificielles (ex : parcours d'un type de composant vers un autre). Par exemple, lorsque la fourmi a terminé la construction du composant *Task* de la tâche par défaut, elle sait, grâce aux règles de parcours intégrées, qu'elle doit maintenant adopter une autre stratégie spécifique à la construction des composants *Desire*. En plus de ces règles, nous ajoutons une étape intermédiaire où la fourmi vérifie les chemins possibles en se basant sur le composant *PartialElement* du nœud courant.

L'algorithme 11 décrit comment une fourmi artificielle explore le graphe pour construire une solution. Concrètement, l'exploration du graphe, correspondant à la construction d'une solution, consiste à trouver, à chaque nœud, le prochain nœud (voir algorithme 12) jusqu'à atteindre un nœud final. Le choix des nœuds suivants se fait en tenant compte de la quantité de phéromones déposées sur les chemins reliant les nœuds. Afin de diversifier de temps en temps les propositions et d'éviter un blocage éventuel sur une solution, nous introduisons dans l'algorithme une probabilité p_{choix} (élevée) qui détermine la manière dont le choix est effectué. Un nombre $q_0 \in [0, 1] \subset \mathbb{R}^+$ est généré aléatoirement.

— si $q_0 \leq p_{choix}$, alors le choix est fait en tenant compte de la quantité de phéromones ;
— si $q_0 > p_{choix}$, alors le choix est fait aléatoirement parmi les nœuds possibles.

Pour l'exécution de cette version de l'algorithme adaptée à notre problématique, les para-

mètres requis sont résumés dans le tableau 4.4.

Algorithme 11 : Algorithme de la fonction *explorer_nœud* d'une fourmi artificielle

```
Données : n_x (nœud courant) ; chemin (collection de nœuds parcourus)
ajouter(n_x, chemin) ;
si n_x est un nœud paramètre alors
    retourner chemin;
sinon si n_x est un nœud «Action» alors
    nb_params ← recuperer_nombre_parametre(n_x) ▷ Nombre de paramètres requis pour l'action ;
    pour n allant de 1 à nb_params faire
        nœud_suivant ← recuperer_nœud_suivant(n_x) ;
        explorer_nœud(nœud_suivant, chemin) ;
    fin
sinon si n_x est un nœud «Desire» alors
        ▷ La fourmi construit ici le chemin pour les paramètres poids, seuil et coefficient puis la tâche
    nœuds_paramètre ← récupérer_successeurs_(paramètre)  ▷ Nœuds suivants (des paramètres) ;
    nœud_poids = nœud_suivant(n_x, nœuds_paramètres)      ▷ Nœud suivant pour le poids;
    explorer_nœud(nœud_poids, chemin) ;
    nœud_seuil = nœud_suivant(n_x, nœuds_paramètres)      ▷ Nœud suivant pour le seuil;
    explorer_nœud(nœud_seuil, chemin) ;
    nœud_coefficient = nœud_suivant(n_x, nœuds_paramètres)  ▷ Nœud suivant pour le coefficient;
    explorer_nœud(nœud_coefficient, chemin) ;
    nœud_suivant ← récupérer_nœud_suivant(n_x);
    explorer_nœud(nœud_suivant, chemin) ;
```

Algorithme 12 : Algorithme de la fonction *récupérer_nœud_suivant*

```
Données : n_x (nœud courant) ; p_choix(Probabilité) ; graphe(Graphe à explorer) ;
suivant ← NULL ;
q_0 ← générer_nombre_aléatoire(0, 1);
si q_0 > p_choix alors
    suivant ← choix_aléatoire_suivant(n_x, graphe) ;
sinon
    liens ← récupérer_liens(n_x)                  ▷ Tous les chemins qui partent de n_x;
    trier(liens)                  ▷ Trier les liens en fonction de la quantité de phéromones;
    lien_max ← récupérer_max(liens) ;
    suivant = nœud_arrivée(lien_max)                  ▷ Nœud d'arrivée du lien;
partialElement ← récupérer_partialElement(n_x)      ▷ partialElement associé au nœud n_x;
vérification ← vérifier(n_x, suivant, partialElement)  ▷ Vérification de la validité du nœud suivant ;
tant que vérification = Faux faire
    suivant ← récupérer_nœud_suivant(n_x) ;
    vérification ← vérifier(n_x, suivant, partialElement) ;
fin
```

TABLE 4.4 – Paramètres requis pour l'exécution de l'algorithme de colonies de fourmis

Paramètre	Description	Domaine
nombre de solutions n	Définit le nombre de solutions aléatoires utilisées pour la construction du graphe.	$n \in \mathbb{N}^{*+}$
Nombre de fourmis n_f	Nombre de fourmis qui partent du nœud «Racine». Il correspond au nombre d'exploration faite	$n_f \in \mathbb{N}^{*+}$
Score limite f_{opt}	Définit le score seuil à atteindre ou à franchir pour qu'une solution soit considérée comme étant optimale.	$f_{opt} \in \mathbb{R}$
Probabilité p_{choix}	Définit la probabilité pour qu'un nœud suivant soit choisi selon la quantité de phéromone sur les chemins qui partent du nœud courant.	$p_{choix} \in [0, 1]$

4.2.3 Adaptations de l'algorithme des Systèmes immunitaires par clonage

Le principe de cet algorithme est de partir de plusieurs solutions qui sont clonées et mutées dans un processus itératif. Dans notre contexte d'étude, aucune adaptation particulière n'est nécessaire pour utiliser cet algorithme. Étant donné que par principe il n'y a pas de recombinaison, le risque d'obtenir des propositions non conformes à la logique de l'animat est considérablement réduit. De plus, il n'est pas nécessaire d'avoir une représentation de solution spécifique, car les mutations peuvent s'effectuer directement sur le modèle en modifiant ou en remplaçant les composants concernés.

Le déroulement de l'algorithme consiste initialement à générer plusieurs solutions de manière aléatoire. Ensuite, dans un processus itératif, des perturbations sont effectuées sur les composants à l'aide d'opérateurs définis à cet effet. Le nombre de clones créés et le degré de «perturbation» apporté à une solution peuvent être déterminés par l'une des méthodes suivantes :

— La méthode probabiliste identifie en fonction de leur adaptation, les solutions susceptibles d'évoluer. Elle utilise l'équation 4.2 pour déterminer, selon le cas, le nombre de clones, le nombre de composants à muter ou l'écart entre le composant d'origine et le composant muté.

$$Y = X.\frac{1}{score} \tag{4.2}$$

avec Y le résultat attendu (ex : nombre de clones) ;
X une valeur de référence liée à Y (ex : taille de la population) ;
score l'adaptation de la solution.

— La méthode empirique consiste à utiliser les données collectées lors de plusieurs expériences réalisées avec l'algorithme pour définir une correspondance entre le score d'une solution et le nombre de clonages ou de mutations qu'elle doit subir.

— La méthode stochastique consiste à définir de manière aléatoire le nombre de mutations et de clonages dans un intervalle préalablement défini.

Comme opérateurs effectuant lesdites perturbations, nous avons :

— L'opérateur de mutation de paramètres :
Les perturbations qui peuvent être apportées à un paramètre sont soit une incrémentation de sa valeur, soit une décrémentation, en utilisant une valeur Δ_x variable en fonction de l'affinité de la solution. On peut déterminer Δ_x en utilisant l'équation 4.2.

— L'opérateur de mutation du composant *Action*
Il est possible de réaliser deux types de mutations sur un composant *Action*. Le premier type consiste à modifier les paramètres de l'action, ce qui est effectué par l'opérateur de mutation de paramètres décrit précédemment. Le deuxième type de mutation consiste à remplacer entièrement l'action par une autre. Dans ce cas, une nouvelle action est générée de manière aléatoire pour remplacer l'ancienne conformément au composant *PartialAction* qui le défini.

— L'opérateur de mutation du composant *Task*
Le composant *Task* étant exclusivement composé de composants *Action*, ses mutations ne peuvent naturellement concerner que les composants *Action*. Trois possibilités de mutations sont envisageables :

1. changer l'ordre des composants *Action* ;
2. ajouter ou de remplacer des composants *Action* dans le *Task* existant ;
3. plus radicalement, générer un nouveau composant *Task* avec de nouvelles actions et de nouveaux paramètres.

— L'opérateur de mutation du composant *Desire* :
Ici, nous avons un composant mixte qui est composé à la fois de paramètres et d'une tâche. Les possibilités de mutations pour ce composant sont également mixtes, ce qui implique l'utilisation à la fois de l'opérateur de mutation du composant *Task* et de l'opérateur de mutation des paramètres.

Lorsque ces perturbations sont effectuées, il est important de veiller à ce que les nouvelles propositions de solutions soient en accord avec le problème d'origine. Nous avons donc intégré dans les opérateurs de perturbation une fonction de validation p. Pour qu'un composant subisse effectivement une mutation, l'opérateur se réfère au composant *PartialElement* qui le décrit (s'il existe) afin de vérifier si le composant muté correspond aux critères définis par l'utilisateur. Si le composant muté satisfait les critères, la mutation est validée. Sinon, elle est réessayée un certain nombre de fois, défini comme un paramètre de l'algorithme. Dans le but de diversifier la population, de nouvelles propositions de solutions sont introduites à chaque itération.

La figure 4.12 illustre le processus de recherche de solution avec cette variante de l'algorithme des Systèmes immunitaires par clonage. Elle montre comment une partie de la solution peut subir des perturbations pour générer de nouvelles propositions de solutions. En appliquant le changement d'actions (1), l'action 4 est remplacée par l'action 3 ; en appliquant la mutation de paramètres (2), le couple de paramètres (P=8, P=2) est devenu (P=6, P=4) et en appliquant la permutation d'actions (3), les postions des actions 4 et 1 sont permutées.

Les paramètres nécessaires à l'exécution de l'algorithme sont résumés dans le tableau 4.5.

TABLE 4.5 – Paramètres requis pour l'exécution de l'algorithme immunitaire de clonage par sélection

Paramètre	Description	Domaine
Nombre d'anticorps n	Définit la taille de la population «d'anticorps »	$n \in \mathbb{N}^{*+}$
Nombre de nouveaux n^+	Définit le nombre de nouvelles solutions introduites dans la population à chaque itération(optionnel).	$n^+ \in \mathbb{N}^+; n^+ < n$
Nombre d'itérations n_{it}	Définit le nombre maximal d'itérations d'exécution de l'algorithme.	$n_{it} \in \mathbb{N}^{*+}$
nombre de tentatives de clonage $n_{attempt}$	Définit le nombre de tentatives de clonage (invalides) qui peuvent être faites sur une solution.	$n_{attempt} \in \mathbb{N}^+$
Score limite f_{opt}	Définit le score seuil à atteindre ou à franchir pour qu'une solution soit considérée comme optimale.	$f_{opt} \in \mathbb{R}$

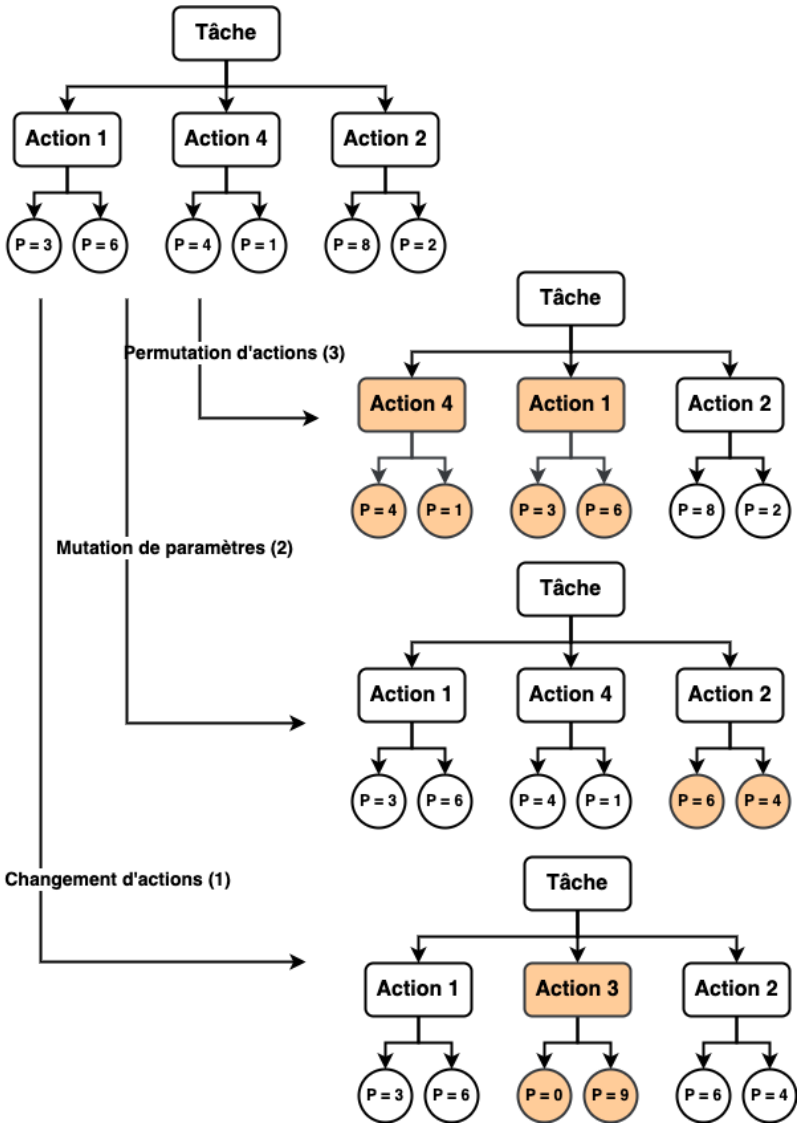


FIGURE 4.12 – Mutations de solution avec l'algorithme immunitaire de clonage par sélection

4.2.4 Adaptation de l'algorithme de la recherche harmonique

L'algorithme de recherche harmonique présente des similitudes frappantes avec les algorithmes génétiques et les algorithmes immunitaires de clonage par sélection. Ces ressemblances sont particulièrement visibles lorsque l'on examine les mécanismes d'échanges d'informations dans le cas des algorithmes génétiques, ainsi que le processus de remplacement ou de mutation dans le cas des algorithmes immunitaires de clonage.

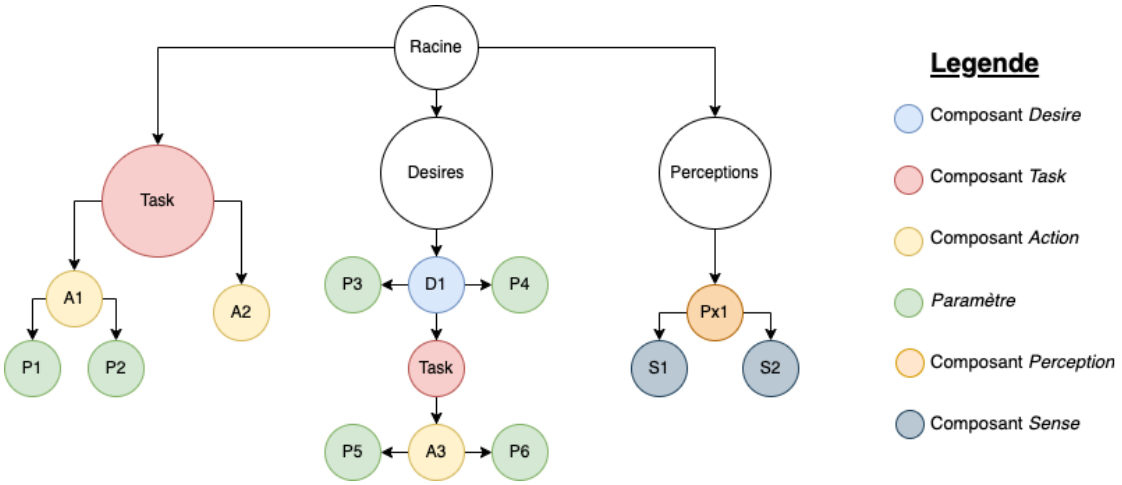
Il est tout à fait intéressant de constater ces éléments conceptuels communs qui peuvent influencer notre manière d'aborder les contraintes liées à la mise à jour des solutions par cet algorithme. En tenant compte du déroulé normal de cet algorithme, nous identifions en priorité deux contraintes : celle de la définition de la meilleure représentation des solutions et celle de la conceptualisation des opérations de mise à jour qui leur sont associées.

4.2.4.1 Représentation des solutions

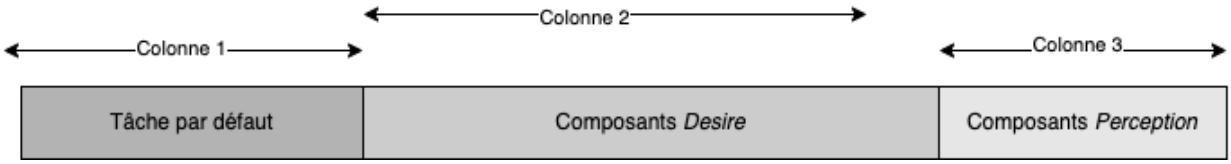
Particulièrement pour l'algorithme de recherche harmonique, une représentation matricielle (n lignes par m colonnes) à toute la population est imposée. Cependant, cette représentation n'est réalisable que sous certaines conditions :

1. Si chaque solution peut être exprimée sous forme d'un vecteur, c'est-à-dire comme une ligne dans une matrice.
2. Si tous les vecteurs ont une taille identique m, correspondant au nombre de colonnes dans la matrice.

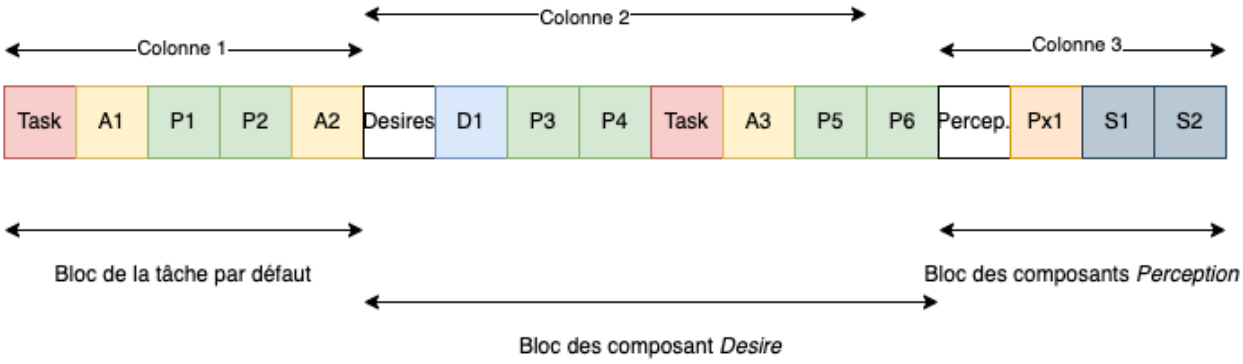
Pour la première condition, lorsque la solution est à l'origine représentée sous forme d'arbre (exemple de la figure 4.13a), la transformation suggérée consiste à «déployer» l'arbre en fonction des nœuds principaux qui émergent à partir de la racine. Concrètement, cela implique de prendre les blocs de perception, les blocs de désirs et le bloc de la tâche par défaut, puis de les «étaler» et de les disposer les uns à la suite des autres comme illustré dans la figure 4.13c.
Pour la seconde condition, uniformiser la taille des représentations, présente un défi substantiel. En effet, comme nous avons pu le faire remarquer plusieurs fois jusqu'ici, les solutions candidates pour un même problème peuvent différer en taille. Ainsi donc, l'application de la transformation proposée peut conduire à des représentations avec des lignes de longueurs variables. Le moyen trouvé pour surmonter ce nouvel obstacle consiste à ne pas déployer tous les blocs, mais plutôt à se limiter aux éléments de base de l'arbre initial, à partir de la racine. Par conséquent, toutes les solutions candidates seront représentées avec une taille fixe $m = 3$ comme illustré dans la figure 4.13b. À première vue, cela pourrait sembler induire une perte d'information ou restreindre les possibilités d'intensification du processus. Toutefois, pour contourner cet obstacle, des mécanismes adaptés sont définis dans l'opérateur de mise à jour, permettant ainsi de maintenir l'efficacité de l'approche.



(a) Solution de base représentée sous la forme d'un arbre



(b) Transformation de la solution de base en une solution à taille fixe ($m = 3$)



(c) Solution de base «déployée»

FIGURE 4.13 – Représentations des solutions utilisées pour l'algorithme de la recherche harmonique

4.2.4.2 Opérateur de mise à jour

La mise à jour avec la recherche harmonique peut se faire de deux façons. Une méthode classique typique à l'algorithme consiste en la création de nouvelles harmoniques à partir des harmoniques existantes, et l'autre consiste simplement à procéder à des perturbations sur une harmonique choisie. Dans l'exécution de l'algorithme, le choix entre l'une ou l'autre des

méthodes est effectué à chaque mise à jour en fonction d'une probabilité P_{aj} définie comme un des paramètres de l'algorithme.

Globalement, l'opérateur de mise à jour proposé fonctionne en deux étapes :

— Étape 1 : Choix des harmoniques candidates

Dans la littérature, on retrouve diverses techniques pour sélectionner les harmoniques candidates, dont bon nombre sont similaires à celles utilisées dans les algorithmes génétiques. Ces méthodes s'appuient en grande partie sur la qualité des solutions candidates, par exemple en sélectionnant les k meilleures dans la matrice de population, ou en adoptant la technique de la roulette. D'autres approches se contentent d'un choix aléatoire des candidats.

Dans notre approche, nous avons choisi de mettre en œuvre ces deux techniques, avec un choix probabiliste pour l'une ou l'autre sur la base d'une probabilité également définie comme paramètre. En adoptant cette stratégie, nous visons à accomplir deux objectifs majeurs. Tout d'abord, en sélectionnant les meilleures harmoniques, nous nous assurons de générer de nouvelles solutions à partir des plus performantes ; ce qui peut accélérer la convergence du processus d'optimisation. De plus, en incorporant un certain degré d'aléatoire dans notre choix, nous préservons un niveau significatif de diversité au sein de notre ensemble de solutions candidates.

— Étape 2 : Proposition d'une nouvelle harmonique

Comme mentionné précédemment, nous avons deux possibilités pour créer de nouvelles harmoniques.

— En ce qui concerne la première possibilité de créer de nouvelles harmoniques à partir d'autres harmoniques existantes, il est crucial de garantir un échange cohérent des blocs de manière à obtenir une harmonique finale «cohérente» du point de vue de la structure d'un animat (comprenant une tâche par défaut, un groupe de perceptions et un groupe de désirs). À ce stade, le problème de cohérence ne devrait normalement pas se poser, car selon la représentation de la solution, les harmoniques sont composées d'un nombre fixe de blocs ($m = 3$). En effectuant des échanges bloc par bloc, la cohérence devrait être maintenue, mais il existe un risque considérable de rester bloqué dans un optimum local. Par conséquent, notre proposition consiste à choisir aléatoirement l'une des harmoniques candidates, puis à remplacer certains de ses blocs (et/ou leurs constituants) par des blocs provenant d'autres harmoniques candidates, de manière similaire à une opération de croisement dans l'algorithme génétique. Le nombre Y de remplacements à effectuer dépend de la qualité de l'harmonique choisie et est déterminé de la même manière que celle présentée dans l'équation 4.2.

Pour effectuer ces remplacements, nous reprenons les harmoniques dans le format indiqué dans la figure 4.13c. Pour chaque remplacement y parmi les Y, un élément de l'harmonique candidate est sélectionné pour être remplacé par un autre élément appartenant apparemment au même type de bloc chez une autre harmonique. Par exemple, si l'élément choisi est la valeur du poids d'un composant *Desire*, il sera remplacé par une autre valeur de poids du même composant *Desire* provenant d'une autre harmonique. Cette approche permet de contourner efficacement la contrainte d'uniformisation des tailles tout en évitant de rester bloqué dans un optimum local. L'exemple présenté dans la figure 4.14 illustre comment une nouvelle solution candidate peut se former à partir d'un ensemble d'harmoniques.

— Enfin, en ce qui concerne la dernière méthode concernant la perturbation d'une solution de base, le processus est similaire à une mutation, à l'instar de l'algorithme des systèmes immunitaires par clonage décrit dans la section 4.2.3. Pour ce faire, une harmonique candidate est sélectionnée aléatoirement parmi l'ensemble de la population. Proportionnellement à sa qualité, le même processus de mutation présenté dans l'algorithme des systèmes immunitaires par clonage est appliqué.

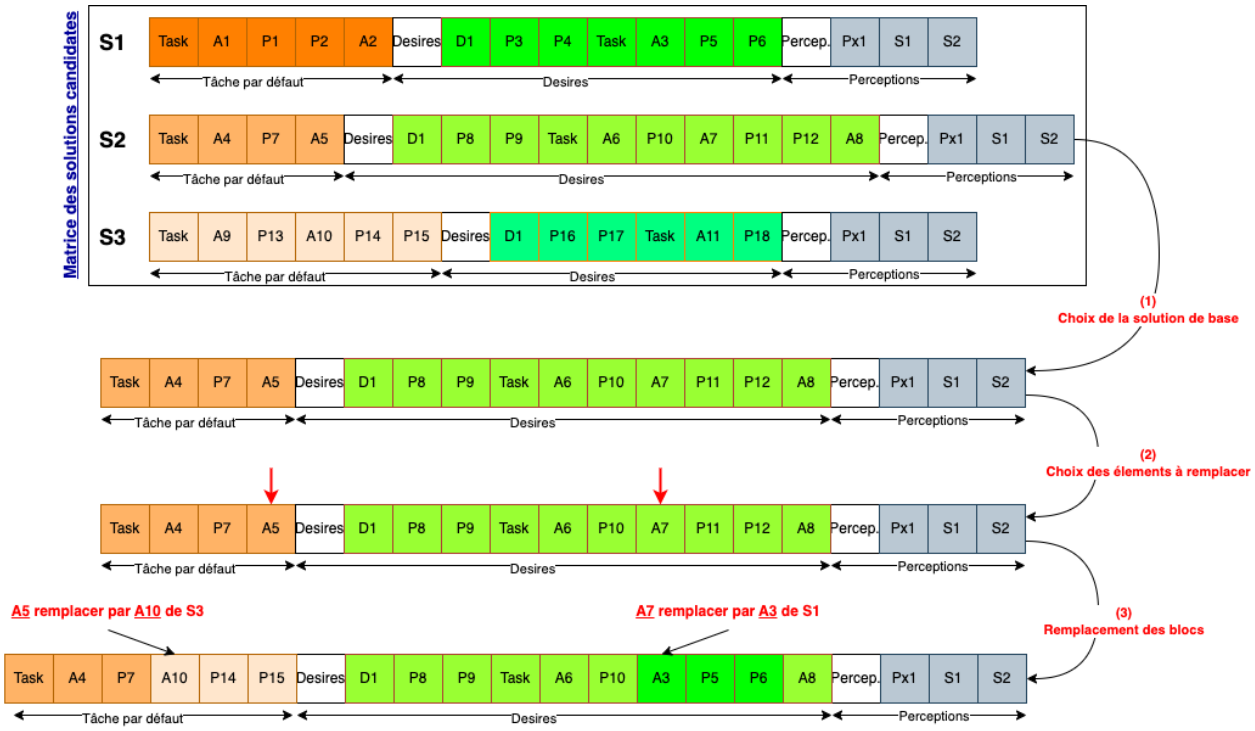


FIGURE 4.14 – Création d'une nouvelle harmonique à partir de la matrice des solutions candidates

Pour exécuter cette adaptation de l'algorithme de la recherche harmonique de manière adéquate, les paramètres énumérés dans le tableau 4.6 sont nécessaires.

TABLE 4.6 – Paramètres requis pour l'exécution de l'algorithme de la recherche harmonique

Paramètre	Description	Domaine
nombre d'harmonique n	Définit le nombre d'improvisations initiales.	$n \in \mathbb{N}^{*+}$
Probabilité sp_s	Définit la probabilité de créer une harmonique à partir des harmoniques existantes (intensification) ou une toute nouvelle improvisation (diversification)	$n_f \in \mathbb{N}^{*+}$
Probabilité p_{aj}	si avec sp_s, il a été choisi de mettre à jour une harmonique existante, p_{aj} définit la probabilité que la dite harmonique soit mise à jour à partir des autres harmoniques ou de procéder à de simples «pertubations»	$f_{opt} \in \mathbb{R}$
Taux t_{aj}	Définit le degré de pertubation à apporter à une harmonique. Elle peut être considérée comme la probabilité que l'harmonique subisse une hypermutation.	$t_{aj} \in [0, 1]$
Probabilité p_{choix}	Définit la probabilité de choix de la technique de sélection d'harmoniques entre la technique de la roulette et la technique stochastique. p_{aj}.	$p_{choix} \in [0, 1]$

La puissance de toutes ces métaheuristiques, réside dans leur capacité à générer des solutions et à les faire évoluer en fonction de leur adaptation au problème à résoudre. L'évaluation des solutions proposées apparaît également donc comme un élément fondamental dans l'exécution des métaheuristiques en général et particulièrement dans notre approche proposée.

4.3 Fonctions d'évaluation de modèles de comportement animal

Elle est tout à fait déterminante pour la réussite d'un processus d'optimisation en ce sens où elle évalue la qualité de la solution proposée, qualité selon laquelle, les améliorations correspondantes sont apportées par les algorithmes.

Il est important de faire clairement la différence entre la fonction «objectif» et la fonction d'évaluation qui sont souvent confondues mais sont deux termes distincts jouant des rôles tout aussi différents.

La fonction d'évaluation attribue une valeur numérique à chaque solution proposée, en fonction des critères spécifiques du problème à résoudre. La fonction «objectif» quant à elle définit formellement l'objectif à atteindre dans le problème considéré. Elle est généralement définie en fonction des variables du problème et peut être de nature différente en fonction du type de problème (minimisation, maximisation, etc.). En résumé, la fonction d'évaluation est utilisée par l'algorithme de métaheuristique pour évaluer la qualité des solutions, tandis que la fonction objectif définit l'objectif global à atteindre dans le problème.

Avec le succès remarquable des méthodes d'apprentissage automatique, on trouve aujourd'hui dans la littérature une diversité de méthodes pour mesurer les performances des modèles (HYNDMAN et al. 2006 ; JADON et al. 2022). Dans le tableau 4.7, nous présentons quelques-unes de ces méthodes (BOTCHKAREV 2018).

Comme on peut le remarquer, ces fonctions d'évaluation se basent sur les trajectoires [3]

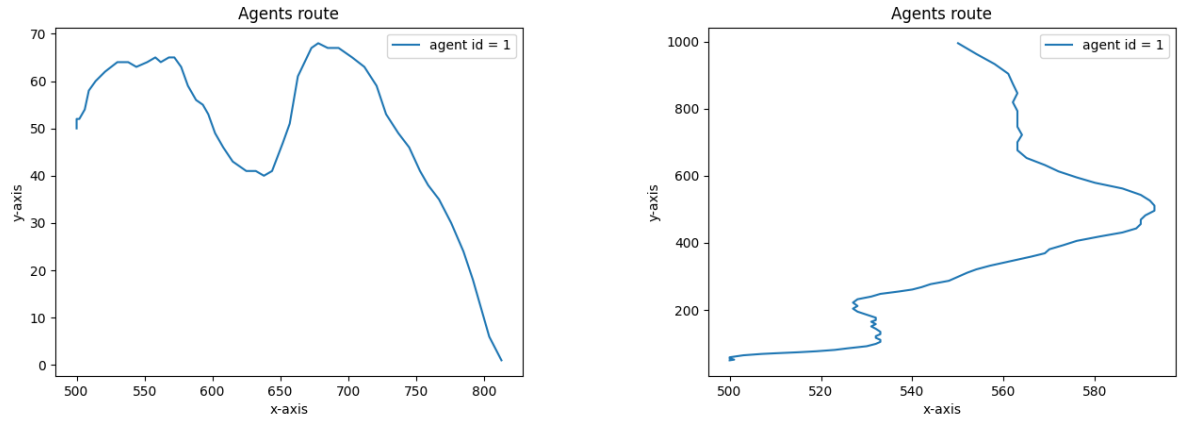
3. On parle de Path-based fitness

TABLE 4.7 – Quelques méthodes d'évaluation de performance de modèles

Méthodes	**Description**	**Equation**
Distance Euclidienne *(DE)*	Mesure la distance dans l'espace euclidien. Elle est recommandée dans lorsqu'on cherche à mieux comprendre les relations entre les variables.	$DE = \sqrt{\sum(y' - y)^2}$ Avec y' la prédiction et y la valeur réelle.
Erreur quadratique moyenne *(MSE)*	Mesure la moyenne quadratique des différences entre les prédictions et les valeurs réelles, également appelées résidus. Elle est utile dans un contexte où on veut minimiser les grandes erreurs par rapport aux petites.	$MSE = \frac{1}{n}\sum(y' - y)^2$ Avec n le nombre de prédictions, y' la prédiction et y la valeur réelle.
Racine carrée de l'erreur quadratique moyenne *(RMSE)*	Comme l'indique sa dénomination, elle est étroitement liée à l'erreur quadratique moyenne, car c'est la racine carrée de cette dernière. Elles apportent une interprétation plus intuitive des erreurs dans la même unité que la variable cible.	$RMSE = \sqrt{\frac{1}{n}\sum(y' - y)^2}$ Avec n le nombre de prédictions, y' la prédiction et y la valeur réelle.
Coefficient de détermination ou R^2 *(R2)*	Évalue la performance d'un modèle en termes de sa capacité à reproduire les résultats observés.	$R2 = 1 - \frac{\sum(y-\overline{y})^2}{\sum(y-\widehat{y})^2}$ Avec y la valeur réelle, $\overline{y}$ la moyenne des valeurs réelles, $\widehat{y}$ la prédiction,
Erreur moyenne en pourcentage absolu (MAPE)	Valeur absolue du pourcentage d'erreur entre les valeurs observées et prévues. C'est une mesure utile lorsque les erreurs relatives ont plus de sens que les erreurs absolues.	$MAPE = \frac{1}{n}\sum\lvert\frac{y-y'}{y}\rvert$ Avec n le nombre de prédictions, y' la prédiction et y la valeur réelle.

(SAHIN et al. 2016) et évaluent la similarité entre la trajectoire de référence et la trajectoire prédite. Cependant, dans notre contexte d'étude, il n'est pratiquement pas possible de prédire parfaitement le comportement des animaux pour plusieurs raisons. Comme nous l'avons décrit jusqu'ici, le comportement de l'animal dépend de ses propriétés individuelles, de son état interne et de l'environnement, qui ne peuvent pas toujours être estimés à partir des données de référence disponibles. De plus, le comportement peut être intrinsèquement stochastique (IM et al. 2020).

Pour illustrer cela, prenons l'exemple des trajectoires présentées dans la figure 4.15, qui ont été générées lors de deux simulations différentes du même modèle. Si l'on choisit l'une comme référence et l'autre comme prédiction, on constate facilement que, bien que les modèles soient les mêmes, l'utilisation des métriques présentées dans le tableau 4.7 ne refléterait pas suffisamment la similitude des résultats.



(a) (b)

FIGURE 4.15 – Trajectoire décrite par un animat lors de deux différentes simulations

Cela soulève alors une question cruciale : *comment évaluer un modèle de comportement d'une espèce animale* ? Jusqu'à présent, très peu de travaux se sont penchés sur cette question. Cependant, une réponse semble être apportée par(IM et al. 2020), qui proposent des métriques pour évaluer les modèles de comportement à savoir la méthode *«n-step»* et la méthode de la distribution d'erreur.

4.3.1 n-step prediction error

La méthode n-step prediction error consiste à examiner l'erreur de prédiction du modèle sur $n > 1$ étapes. Contrairement à la méthode de l'erreur quadratique moyenne qui mesure l'écart point à point, la méthode n-step réalise n simulations et calcule l'erreur minimale parmi ces échantillons. L'erreur minimale est déterminée par l'équation 4.3 :

$$err_{xt}^{m} := min(||x_t^m - x_t||) \tag{4.3}$$

où x_t est la valeur de la variable dans les données de référence à l'instant t et x_t^m est la prédiction à l'instant t lors de la simulation m ($m \in [1, n] \subset \mathbb{N}^{*+}$).

Cette approche d'évaluation semble apporter des réponses concrètes au problème soulevé par l'exemple de la figure 4.15. Cependant, elle reste assez complexe, car elle nécessite la réalisation de n simulations.

4.3.2 Distance d'histogramme

Il s'agit d'une mesure utilisée pour évaluer la similarité ou la divergence entre deux histogrammes (CHA et al. 2002 ; MA et al. 2010). Cette distance peut être déterminée de plusieurs manières, parmi lesquelles nous avons la distance de Bhattacharyya, la distance Chi carré et la distance de Hellinger. Les formules mathématiques pour calculer chacune de ces distances sont présentées dans le tableau 4.8. Ces mesures sont inspirées de la manière dont les biologistes comparent quantitativement le comportement des animaux. Dans le cadre de notre

étude, elles nous permettront de comprendre si les modèles de simulation sont adéquats dans différents scénarios ou conditions.

Plus la distance d'histogramme est faible, plus les distributions sont similaires. Une distance d'histogramme de 0 indiquerait une parfaite similarité, tandis qu'une distance de 1 indiquerait une divergence maximale.

Table 4.8 – Formules mathématiques de calcul de distances d'histogrammes

Distance	**Formule mathématique**
Distance de Bhattacharyya	$D_{\text{bhattacharyya}} = -\ln\left(\sum_i \sqrt{p_i \cdot q_i}\right)$ Avec p_i et q_i représentent les fréquences normalisées des intervalles correspondants dans les histogrammes du A et du Groupe B.
Distance de Chi carré	$D_{\chi^2} = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$ Avec O_i l'observation (la fréquence) dans la i-ème classe de l'histogramme réel, E_i l'espérance (la fréquence attendue) dans la i-ème classe de l'histogramme simulé et n le nombre total de classes ou d'intervalles dans l'histogramme.
Distance de Hellinger	$D_{\text{Hellinger}} = \sqrt{\frac{1}{2}\sum_{i=1}^{n}(\sqrt{p_i} - \sqrt{q_i})^2}$ Avec p_i la probabilité associée à l'événement i dans la première distribution, q_i la probabilité associée à l'événement i dans la deuxième distribution, n le nombre total d'événements ou de classes.

Le passage en revue de toutes ces approches d'évaluation nous a permis de nous rendre compte que peu de travaux se penchent sur l'évaluation à proprement dite des modèles de comportements, en particulier ceux qui concernent les animaux. Pour les quelques-uns que nous avons pu voir, chacune d'elles a ses points forts et ses points faibles, et généralement, leur précision augmente proportionnellement avec leur complexité. Compte tenu de ce constat, nous proposons d'implémenter la distance euclidienne, l'erreur moyenne en pourcentage absolu, la méthode n-step, et la distance d'histogramme afin de laisser l'utilisateur choisir selon les objectifs de sa modélisation.

Conclusion

En conclusion de ce chapitre, nous avons exposé en détail notre approche qui aborde la génération de nouveaux modèles de comportements en utilisant des connaissances complètes ou partielles sur l'espèce étudiée. En combinant une analyse approfondie des pré-requis et des connaissances construites dans les domaines de la modélisation, de la simulation et de l'optimisation par les métaheuristiques, nous sommes parvenus à proposer une approche qui considère la modélisation du comportement animal comme un problème d'optimisation pouvant être résolu à l'aide des métaheuristiques.

Pour atteindre cet objectif, nous avons proposé une description du comportement animal par des composants appelés *PartialElement* compatibles avec les métaheuristiques dans le cadre d'un processus d'optimisation. De plus, nous avons adapté les principaux algorithmes des métaheuristiques pour surmonter les contraintes et limites qu'ils imposent. Ces adaptations visent à intégrer dans ces algorithmes notre concept de «modélisation guidée par l'optimisation».

Cependant, dans le souci de suivre une démarche scientifique rigoureuse, notre approche nécessite une vérification et une validation approfondies afin d'évaluer sa robustesse et son applicabilité dans des scénarios réels. En vérifiant la cohérence de la mise en œuvre de l'approche proposée et en la confrontant à des cas d'application, notre objectif dans le prochain chapitre est de consolider sa crédibilité et d'identifier les domaines où des améliorations pourraient être apportées.

CHAPITRE 5

Vérification et validation de ANIMETA

Sommaire

Introduction

Dans les chapitres 4 et 3 précédents, nous avons présenté différents concepts pouvant nous permettre de générer automatiquement des modèles de comportement d'espèces animales à l'aide des méthodes d'optimisation, notamment les métaheuristiques. Ces concepts ont posé les bases pour la solution novatrice proposée en réponse à la problématique soulevée par nos recherches. En référence aux deux concepts phares sur lesquels se fonde cette solution (les animats et les métaheuristiques), nous avons dénommé l'outil qui met en œuvre cette approche de solution *ANIMETA* [1].

ANIMETA est donc un ensemble d'outils qui peut être mis en œuvre par un expert du comportement des animaux pour générer des modèles de comportement de ces animaux et ainsi aider l'expert à mieux les comprendre. Pour ce faire, ANIMETA s'appuie sur deux principaux concepts que nous avons définis : une nouvelle architecture d'animat avec le simulateur associé et une approche de génération de ce type de modèle pour des espèces animales (présentés dans la section 3.1 et dans la section 4.1).

Dans ce chapitre, nous présentons premièrement comment les différents concepts de ANIMETA ont été implémentés, c'est-à-dire son architecture, le langage utilisé, l'environnement de travail ainsi que l'organisation du code. Deuxièmement, nous présentons le processus qui a été mis en œuvre pour vérifier l'implémentation des concepts, puis nous terminons en présentant quelques applications de ces concepts pour générer des modèles d'espèces animales en guise de validation de l'approche proposée.

5.1 Architecture de la suite d'outils ANIMETA

L'un des principaux objectifs de nos travaux est de rendre la modélisation du comportement animal simple et accessible pour un public qui n'est pas nécessairement expert en programmation et qui, le plus souvent, est réticent à l'installation de logiciels pouvant éventuellement perturber leur environnement de travail [2]. Pour cela, ANIMETA est subdivisé en trois parties : une interface Web (ANIMETA-HIM), une partie opérationnelle (ANIMETA-ENGINE) et une interface de programmation d'application (ANIMETA-API) liant les deux premières parties.

5.1.1 ANIMETA-HIM

ANIMETA-HIM est une interface homme-machine, pensée et conçue pour être épurée et intuitive pour l'utilisateur. Deux fonctions sont assurées par ANIMETA-HIM : permettre à l'utilisateur, d'une part, de concevoir et de simuler des modèles «ANIMETA» et, d'autre part, de fournir les informations et données nécessaires pour générer un modèle à l'aide des métaheuristiques. Pour ce faire, ANIMETA-HIM reste dans la continuité de la représentation de l'animat présentée dans la section 4.1.2, en fournissant les différents composants qui forment un animat. Ainsi, concevoir un modèle revient pour l'utilisateur à construire l'arbre qui correspond à l'animat à simuler en se servant des différents composants disponibles sur l'interface. Pour faciliter cette construction d'arbre, ANIMETA-HIM est conçu à l'aide du

1. Nom issu de la combinaison des mots « ANImat » et « METAheuristique »
2. ex : conflit de version avec un autre outil ; utilisation des ressources de la machine

framework GoJS [3] selon le principe du «Drag and Drop» [4]. La même philosophie est mise en œuvre également pour la génération des modèles à l'aide des méthodes d'optimisation. Comme nous l'avons vu dans la section 4.1.2, générer le modèle revient à décrire un modèle partiel (*PartialElement*) qui est utilisé dans un processus d'optimisation. ANIMETA-HIM fournit là aussi l'ensemble des composants utiles pour la construction du modèle partiel. La figure 5.1 montre une capture d'écran de ANIMETA-HIM.

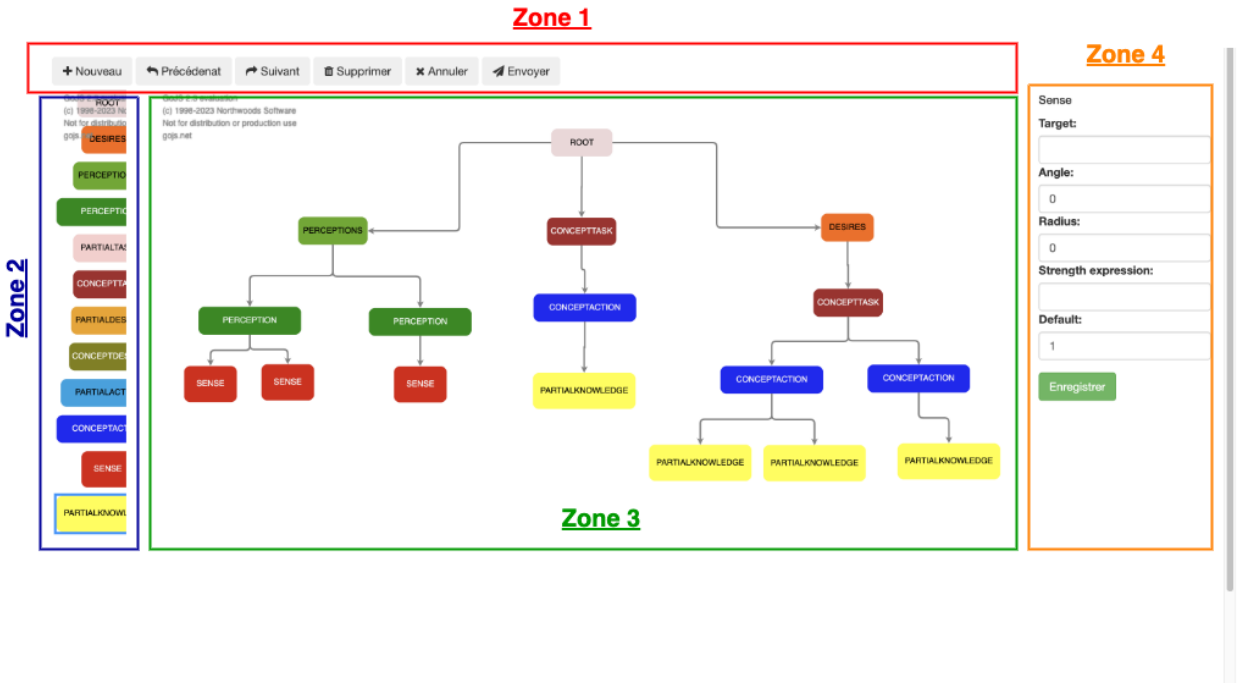


FIGURE 5.1 – Capture d'écran de ANIMETA-HIM

La **zone 1** correspond à la barre d'outils, où l'utilisateur peut trouver différentes actions qui peuvent être réalisées, telles que créer/ouvrir un projet, lancer une simulation, récupérer les données d'une simulation, etc.

La **zone 2** est la boîte à outils, où l'on retrouve les différents composants nécessaires à la construction à la fois d'un modèle complet pouvant être simulé, ainsi que les composants pour construire un modèle partiel.

La **zone 3** est l'espace de travail, où l'arbre est construit. Les composants de la zone 2 y sont placés (par «Drag and Drop») et connectés entre eux.

La **zone 4** est le volet de paramétrage, qui sert à éditer les paramètres de chaque composant de l'espace de travail. Un double-clic sur un composant de l'espace de travail fait apparaître dans le volet de paramétrage un formulaire avec les différents paramètres dudit composant.

L'implémentation de ANIMETA-HIM a été réalisée avec des technologies web (HTML5, CSS3, Javascript et GoJS) pour être accessible en ligne. Le choix de cette technologie permet

3. Site web officiel de GoJS : https://gojs.net/latest/index.html
4. Méthode consistant à utiliser une souris, un pavé tactile ou un écran tactile pour déplacer un élément graphique d'un endroit à un autre sur l'écran (voir https://fr.wikipedia.org/wiki/Glisser-d%C3%A9poser)

de rendre ANIMETA accessible à un large public et, surtout, de lever la contrainte d'installation sur la machine de l'utilisateur. Ceci implique donc de déplacer les fonctionnalités principales vers un nœud central avec de ressources matérielles suffisantes.

5.1.2 ANIMETA-API

Il s'agit ici d'une couche intermédiaire qui fait le lien entre l'utilisateur et le côté « Back-end » du projet. C'est donc une interface de programmation d'application dont le rôle est de recevoir les requêtes venant de l'utilisateur, par exemple via ANIMETA-HIM, les analyser, les sauvegarder et les soumettre pour traitement. Une fois le traitement terminé, elle récupère les résultats et les présente à l'utilisateur.

ANIMETA-API est donc une API REST (Fielding 2000) développée en utilisant le framework Sails.js [5], basé sur la plateforme Node.js [6]. Ce framework se démarque particulièrement des autres de sa catégorie par sa robustesse et sa facilité de mise en œuvre [7]. Le requêtage d'une fonction de traitement de ANIMETA se fait au moyen d'une URL spécifiquement définie à cet effet. Pour des raisons de sécurité, l'authentification à l'API est faite au moyen de jetons (tokens).

Grâce à cette API, le champ d'utilisation de ANIMETA se trouve davantage élargi puisqu'il devient ainsi possible de l'intégrer dans d'autres outils existants, s'inscrivant ainsi dans l'atteinte de l'objectif qui est d'aider au mieux les biologistes à mieux comprendre le comportement des animaux.

5.1.3 ANIMETA-ENGINE

Comme son nom pourrait le suggérer, il s'agit de l'élément central de tout le projet ANIMETA. Dans la suite logique de ce qui a été dit précédemment, ANIMETA-ENGINE est la couche « back-end » en charge du traitement des données du projet. On pourrait ainsi comprendre par là qu'il s'agit de l'implémentation concrète du cœur du projet ANIMETA. Implémenté en langage Python (version 3.9.6) sous l'environnement de développement PyCharm [8], ANIMETA-ENGINE inclut les éléments nécessaires pour traiter les données reçues de ANIMETA-API, simuler les modèles et exécuter le processus d'optimisation avec les métaheuristiques.

En nous basant sur ces fonctionnalités, ANIMETA-ENGINE se décompose en plusieurs paquets, comme illustré par son arborescence à la figure 5.2.

5. Site web officiel de Sails.js : https://sailsjs.com/
6. Site web officiel de Node.js : https://nodejs.org/fr
7. Avec Sails.js, les composants du framework sont créés par de simples lignes de commandes
8. https://www.jetbrains.com/fr-fr/pycharm/

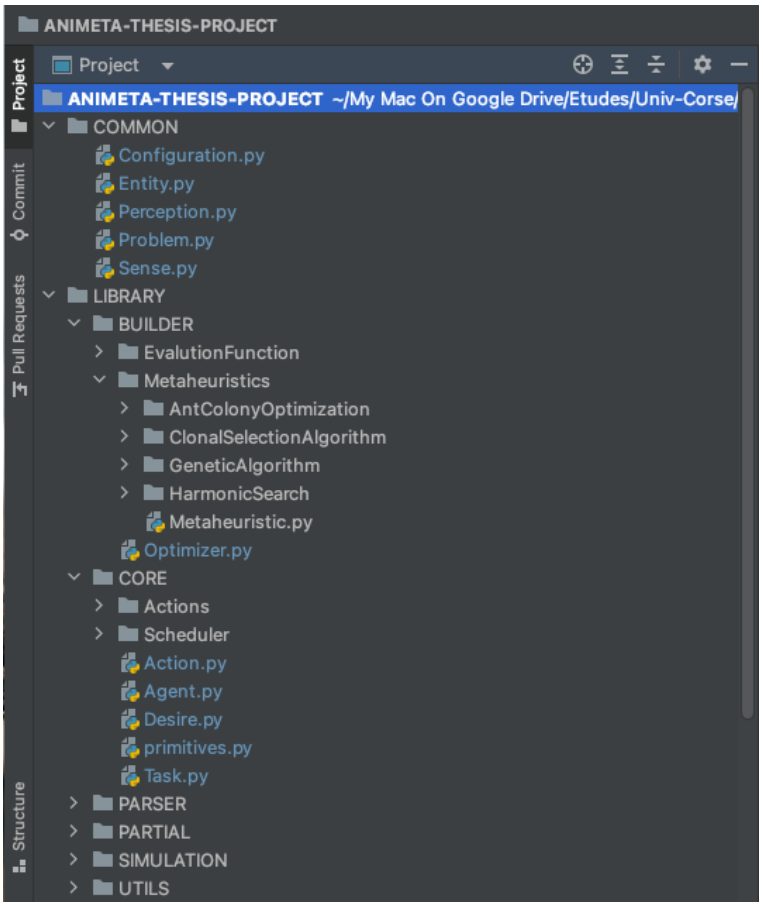


FIGURE 5.2 – Arborescence du ANIMETA-ENGINE

Le paquet «COMMON» regroupe les composants qui sont communs à toutes les fonctionnalités de ANIMETA-ENGINE. On y retrouve, par exemple, la définition des classes des composants *Perceptions* et *Senses*, qui sont les mêmes qu'il s'agisse d'une modélisation, d'une simulation ou d'une génération de modèles. Le paquet «LIBRARY» assure les différentes fonctions de ANIMETA-ENGINE. Il se compose de paquets tels que :

— Le paquet «BUILDER» est celui qui s'occupe de la génération de modèles. Il contient l'implémentation des différentes métaheuristiques (algorithmes, représentation de solution) ainsi que les différentes fonctions d'évaluation.

— Le paquet «CORE» est formé des différents composants de l'architecture fonctionnelle de l'Animat, tels que les composants *Desire*, *Task* et *Action*. Il s'agit des composants «complets», directement exécutables.

— Le paquet «PARSER» assure le passage fidèle des informations entre les différents niveaux de fonctionnalité. Il est composé de l'ensemble des fonctions et méthodes chargées de traduire les informations d'une couche à une autre. Il s'agit, par exemple, de passer du format JSON brut passé en entrée à une instance d'objet utilisable dans le processus d'optimisation ou encore de récupérer une instance d'un composant de l'architecture fonctionnelle de l'animat (présentée dans la section 3) à partir d'un composant *PartialElement* correspondant.

— Le paquet «PARTIAL» est composé des différents composants de type *PartialElement* utilisés pour définir le modèle partiel, indispensable pour la génération de modèles.
— Le paquet «SIMULATION» regroupe l'ensemble des outils nécessaires pour simuler un modèle ANIMETA. Les principaux constituants sont les classes gérant l'environnement, la collecte des données lors de la simulation et la réalisation de la simulation à proprement dite.
— Le paquet «UTILS», quant à lui, fournit des fonctions élémentaires pour le bon fonctionnement de ANIMETA-ENGINE, comme l'importation automatique des paquets selon les besoins de l'exécution en cours.

Les liens et les échanges qui existent entre ces trois parties du projet ANIMETA sont illustrés dans la figure 5.3.

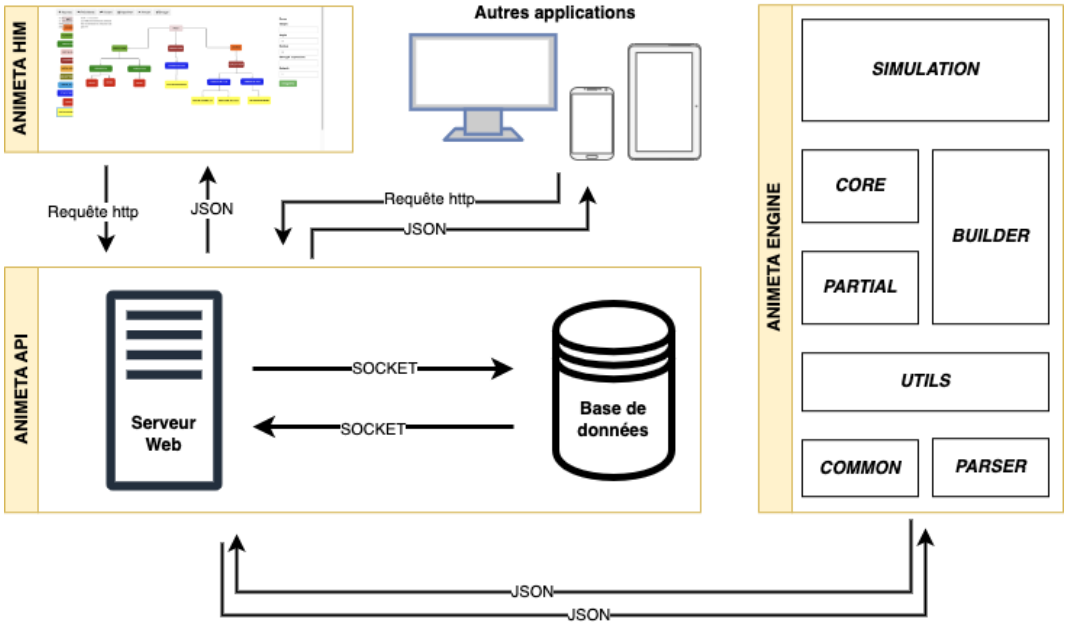


FIGURE 5.3 – Liens et échanges entre ANIMETA-HIM, ANIMETA-API et ANIMETA-ENGINE

En franchissant ainsi l'étape de l'implémentation, nous pouvons désormais passer à celle de la vérification et de la validation (OBERKAMPF et al. 2010) afin de confirmer sa pertinence et sa valeur. En effet, ANIMETA aspire à être un outil complet d'aide à la compréhension du comportement animal en s'appuyant sur les apports de la modélisation et de la simulation. C'est bien pour cela que nous avons proposé une nouvelle façon de modéliser le comportement animal (voir la section 3.1). Nous avons donc logiquement développé le simulateur associé à cette approche de modélisation. De plus, nous avons également proposé une approche de modélisation et de simulation qui s'appuie sur l'architecture proposée pour l'animat, en vue de générer des modèles de simulation. Il est donc manifestement indispensable de vérifier et de valider ces propositions à travers un examen minutieux des processus et une comparaison avec la réalité.

Avant de continuer dans nos explications, nous pensons qu'il est important d'apporter des clarifications sur ce qu'est la vérification et ce qu'est la validation, deux termes qui ont

tendance à être confondus :

— Vérifier un système logiciel, c'est s'assurer que celui-ci se comporte de la manière dont il a été défini puis programmé, et qu'il produit le résultat correspondant à ce fonctionnement. Il est généralement basé sur la logique, pour fournir des preuves solides. Il existe de nombreuses techniques de vérification, moins formelles, la plus populaire étant le test(Cardoso et al. 2021).

— Valider un système logiciel, en particulier un modèle, c'est évaluer à quel point il réagit de la même manière que le système réel.

En d'autres termes, vérifier un modèle revient à répondre à la question « est-ce que la simulation correspond vraiment à la description faite par le modèle ? » et valider répond à la question « est-ce que le modèle représente bien la réalité ? ».

Dans notre cas ici, notre objectif est double. D'une part, il s'agira de s'assurer que notre approche est fonctionnelle et capable de rendre fidèlement, à travers la simulation, la description faite dans le modèle. D'autre part, il s'agira de s'assurer que les modèles de comportement générés se rapprochent le plus des comportements réels.

5.2 Vérification de la suite ANIMETA

Compte tenu des fonctions principales de ANIMETA, nous effectuons une vérification en deux parties. La première concerne la vérification du modèle générique de comportement animal proposée et de son simulateur associé. La seconde partie, quant à elle, porte sur le processus de génération de modèle. Pour ce faire, nous avons proposé une méthode de vérification propre à notre contexte d'étude. Cette méthode tire ses fondements de (Kraibi et al. 2019) et de (Chen et al. 2022), des travaux qui ont mis en œuvre une technique similaire.

5.2.1 Méthode de vérification utilisée

L'ultime attente que nous avons pour une méthode de vérification d'une telle approche de modélisation est de s'assurer que les résultats obtenus à l'issue de chaque simulation et de chaque génération de modèle sont conformes à ce qu'ils devraient être, tout en étant en accord avec la réalité. En nous inspirant du principe de parcimonie, notre procédé repose sur des vérifications simples et élémentaires. L'hypothèse ici est qu'en vérifiant chaque composant implémenté, nous vérifions ainsi l'ensemble du système logiciel. En d'autres termes, si chaque composant fonctionne comme prévu et aux moments prévus, le système peut également être considéré comme fonctionnel. Puisque notre système a principalement deux fonctionnalités : d'un côté la modélisation et la simulation, et de l'autre la génération de modèles, la vérification s'effectue également en deux parties, chacune pour une fonctionnalité spécifique.

5.2.1.1 Vérification de la modélisation et de la simulation

Pour procéder aux vérifications, nous avons opté pour un modèle simple expérimental comme modèle de référence. Ce choix se justifie par deux raisons principales. Tout d'abord, ce modèle est simple ; ce qui signifie qu'il a une complexité réduite et qu'il est plus facile

de comprendre son comportement. Cela nous permet de détecter efficacement des anomalies lors du processus de vérification.

Ensuite, le fait qu'il s'agisse d'un modèle expérimental fictionnel offre la possibilité de définir nous-mêmes le comportement voulu du modèle, d'accentuer ou d'atténuer la vérification pour une partie particulière, et d'augmenter progressivement les niveaux de complexité. Cela nous donne une grande flexibilité pour adapter le modèle aux différents aspects que nous souhaitons évaluer.

Le modèle expérimental utilisé décrit un animat placé dans un environnement où se trouvent plusieurs sources de nourriture. L'animat se déplace de façon aléatoire dans cet environnement jusqu'à ce qu'il ait le désir de se nourrir. Ce désir de se nourrir est modélisé par une valeur seuil de la quantité d'énergie dont dispose l'animat (on prendra pour seuil la valeur 100). À chaque déplacement, l'animat perd une quantité d'énergie proportionnelle à la distance parcourue, soit une unité d'énergie par unité de distance. Lorsqu'il se nourrit, il gagne cinq unités d'énergie par élément consommé. Cette description, bien que sommaire, apporte des informations précises sur le comportement de l'animat. On peut donc déjà la qualifier de pseudo-éthogramme.

En ramenant le pseudo-ethogramme à un modèle ANIMETA, l'animat est constitué de :

— Un composant *Perception* pour détecter la «faim» et une source de nourriture ;
— Un composant *Desire* déclenché par la perception et qui exécute la tâche conduisant à la consommation de la nourriture ;
— Une tâche par défaut représentant le déplacement à aléatoire.

Une description détaillée de ces composants ainsi que leur configuration est présentée par le diagramme d'objet UML de la figure 5.4.

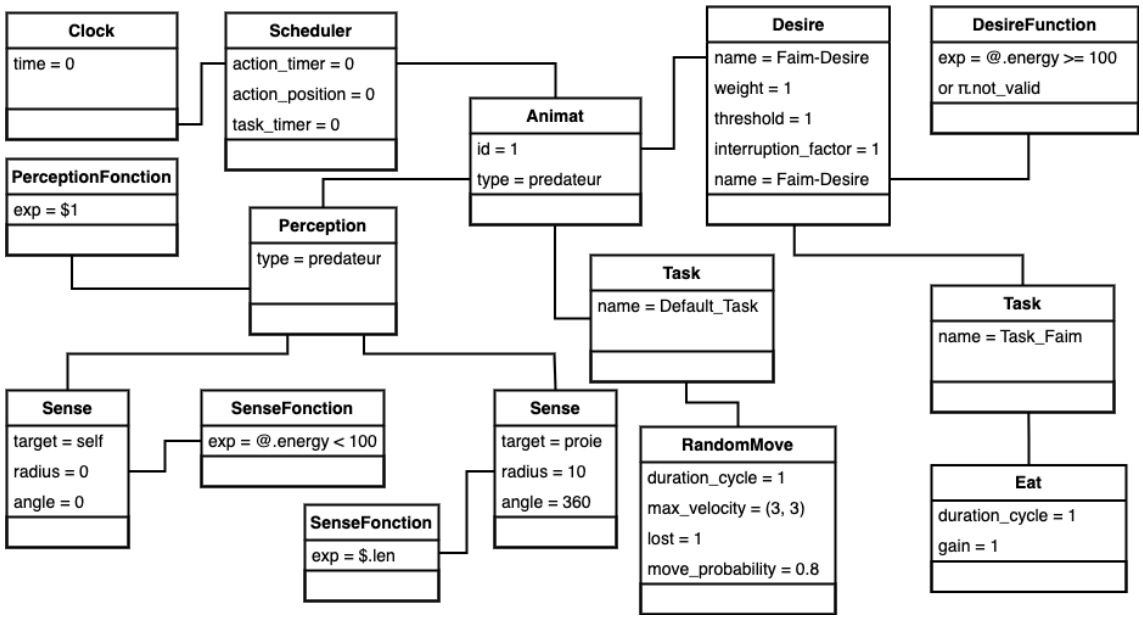


Figure 5.4 – Diagramme d'objet UML du modèle expérimental pour la vérification

Globalement, pour chaque fonctionnalité testée, l'approche méthodique de la vérification s'est déroulée en cinq (5) étapes essentielles minutieusement suivies afin d'assurer la fiabilité et la cohérence de notre approche.

Étape 1 : Génération de diagramme d'activité et de séquence UML.

Comme expliqué précédemment, la vérification nécessite d'abord une bonne connaissance du système. À cet effet, nous commençons chaque vérification par l'élaboration d'un diagramme d'activité puis un diagramme de séquence pour représenter visuellement, selon nos connaissances, le flux d'exécution. Ce diagramme nous sert de guide tout au long de la procédure de test, car il représente ce dont on attend de l'exécution et permet d'identifier les chemins critiques et les zones susceptibles d'engendrer des erreurs.

Étape 2 : Introduire des écritures en console dans le Code.

En nous basant sur les diagrammes élaborés à l'étape 1, nous ajoutons des lignes supplémentaires dans les portions de code correspondant aux séquences. Ce code supplémentaire permet d'afficher en console des messages pour indiquer l'exécution effective de la fonction, afficher l'identifiant unique de l'instance en cours d'exécution, les propriétés au début et à la fin, ainsi que les valeurs d'entrée et les valeurs retournées en fin d'exécution.

Afficher ainsi en console ces informations permet de vérifier en temps réel les valeurs des variables et des objets pour identifier tout comportement inattendu ou erroné. D'autre part, cette étape permet de vérifier la cohérence des résultats attendus par rapport aux entrées fournies.

Étape 3 : Préparation du test.

Ici, nous préparons les éléments essentiels pour mener à bien les tests envisagés, ce qui inclut notamment la création des jeux de données indispensables à nos expérimentations. Plus particulièrement, dans le cadre de la vérification de la phase de modélisation et de simulation, notre démarche a consisté à mettre en œuvre le modèle théorique que nous avions préalablement proposé en utilisant ANIMETA-HMI.

Étape 4 : Exécution du processus à tester

Une fois que les éléments nécessaires à l'exécution du test sont mis à disposition à l'étape 3, nous procédons ici à l'exécution du processus que nous souhaitons tester. Dans notre cas, pour la vérification de la partie modélisation et simulation de ANIMETA, il s'agit de l'exécution d'une série de simulations pour lesquelles les résultats produits sont collectés et sauvegardés.

Étape 5 : Analyse et comparaison des données résultats

Cette dernière étape consiste à comparer les résultats obtenus (affichage en console, comportements observés, données collectées) aux résultats attendus (la suite logique

définie dans les diagrammes de séquence et d'activité). Il s'agit ici de mener une analyse approfondie pour confirmer ou infirmer la justesse de l'approche. Cette comparaison nous permet également de mettre en lumière les points forts et quelques limites de notre approche, ce qui nous aide à apporter des ajustements et des améliorations là où cela est nécessaire.

En procédant ainsi, nous pouvons vérifier la cohérence de notre approche et ainsi nous assurer que le système répond aux exigences spécifiées.

5.2.1.2 Vérification de la génération de modèles

Pour ce qui est de la vérification de la génération de modèle, le même protocole que celui utilisé pour vérifier la modélisation et la simulation a été appliqué. Pour chaque fonctionnalité testée, un diagramme de séquence UML a été élaboré, des messages écrits en console ont été ajoutés, l'exécution prévue a été effectuée, et les données ont été analysées. La différence fondamentale se trouve ici au niveau de l'étape 3 qui concerne la donnée de test, qui n'est plus un modèle complet pouvant être directement simulé, mais plutôt un modèle partiel avec un jeu de données de références.

Pour obtenir ce type de données, nous avons repris le modèle présenté précédemment et l'avons rendu « partiel » en considérant simplement que le paramétrage de l'action de la tâche par défaut n'est pas connu. Le diagramme d'objet UML de la figure 5.5 illustre ce modèle partiel utilisé. Nous pouvons remarquer sur le diagramme qu'en guise de tâche par défaut, ce n'est plus un composant *Task* qui est utilisé, mais un composant *PartialTask*. Cela s'explique simplement par le fait que la tâche n'est pas complète puisque l'action qui la constitue aussi n'est pas complète. C'est d'ailleurs ce qui justifie le fait que l'action soit définie elle aussi par un composant *PartialAction*.

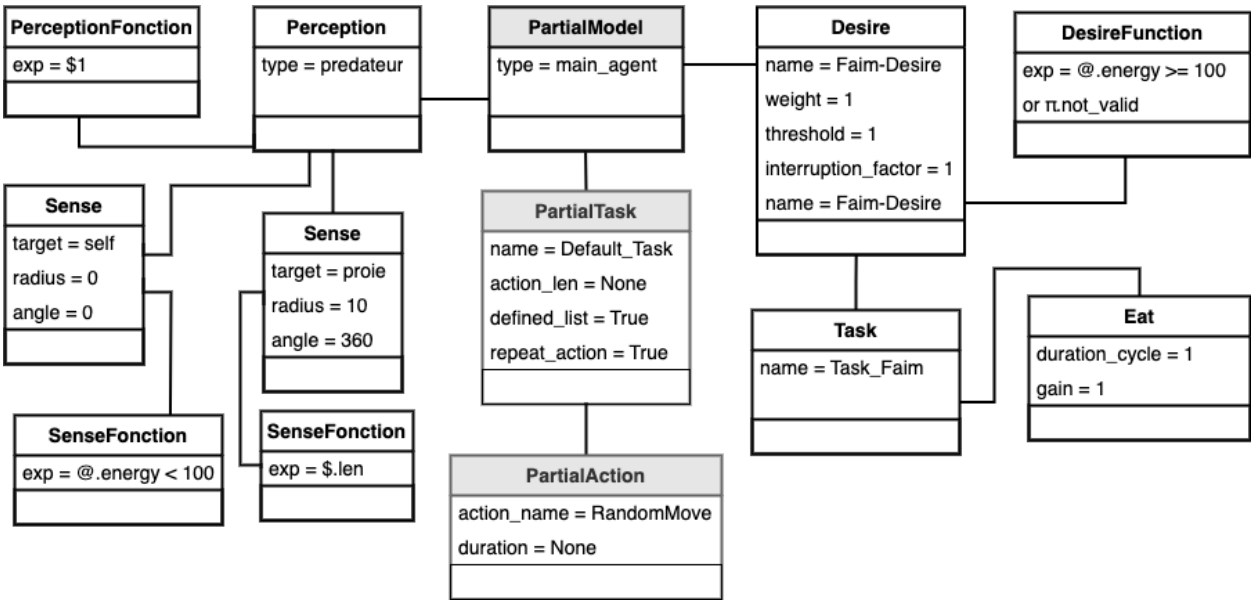


Figure 5.5 – Diagramme d'objet UML du modèle partiel utilisé pour vérification de la génération de modèles

En ce qui concerne le jeu de données de référence nécessaire pour générer le modèle, nous utilisons celles collectées lorsque nous avons réalisé la série de simulations à l'étape 4, pendant le processus de vérification de la modélisation et de la simulation. Il est important de préciser qu'ici, la finalité n'est pas de contrôler l'exactitude du modèle généré, mais plutôt de vérifier si le processus de génération fonctionne comme nous l'avons pensé et conçu.

5.2.2 Résultats obtenus à la vérification

Comme annoncé précédemment, l'objectif principal des vérifications réalisées est de s'assurer que nous avons un système qui fonctionne comme nous l'avons pensé et conçu. Le protocole étant le même qui a été appliqué pour les deux groupes de vérifications réalisées (modélisation-simulation et génération de modèles), et au risque de nous répéter, nous trouvons judicieux de présenter les résultats obtenus en deux autres volets en ayant comme point de vue les grands blocs : en premier lieu, le volet de l'interfaçage avec les utilisateurs, et en second lieu, le volet du «moteur» de ANIMETA.

5.2.2.1 L'Interfaçage avec les utilisateurs

Conformément au protocole mis en place, nous avons débuté nos vérifications par l'élaboration de diagrammes de séquences UML en guise de référence (étape 1 du protocole de vérification). Le diagramme de la figure 5.6 présente le diagramme conçu à cet effet. Il représente le flux d'exécution du processus de conception et de simulation d'un modèle du côté de ANIMETA-ENGINE. Sur la figure 5.7, il s'agit du diagramme de séquence de la partie de génération de modèles à l'aide des métaheuristiques. Les parties qui concernent la création du modèle partiel et la simulation n'y figurent pas entièrement, car il s'agit des mêmes que celles présentées dans le diagramme de la figure 5.6.

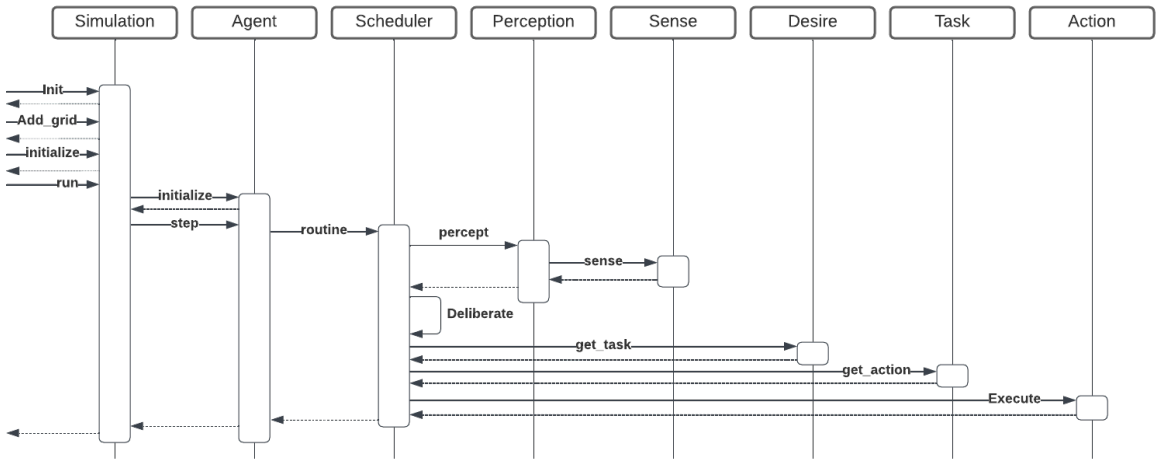


FIGURE 5.6 – Diagramme de séquence UML du processus de génération d'un modèle

En nous référant à ces diagrammes, les tests proprement dits ont été réalisés à la fois sur une même machine et sur un serveur dans un environnement virtuel PyCharm (version 2023.2). Les caractéristiques de la machine et du serveur sont résumées dans le tableau 5.1.

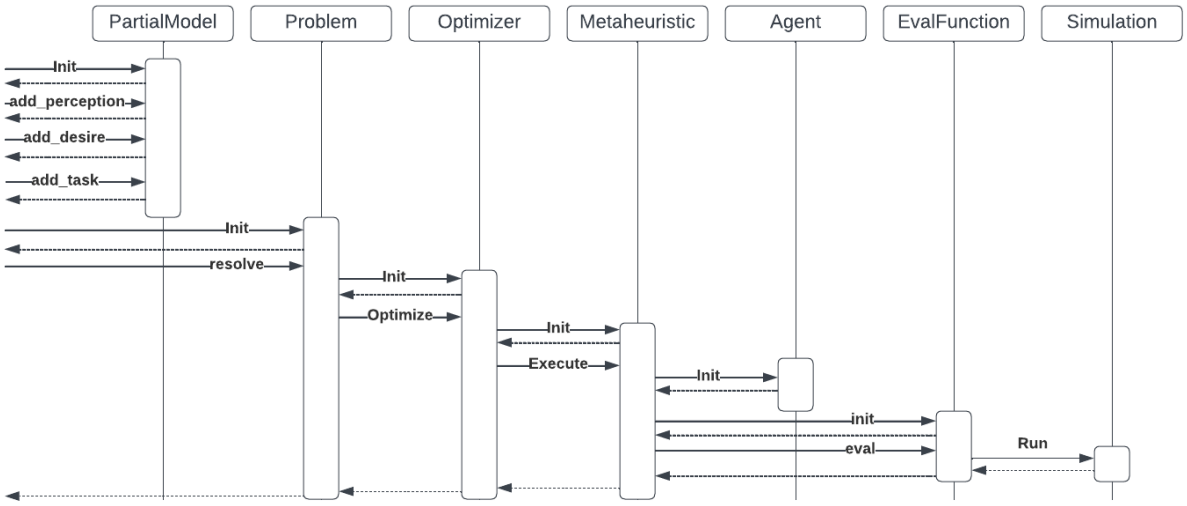


FIGURE 5.7 – Diagramme de séquence UML du processus de génération d'un modèle

TABLE 5.1 – Configuration de l'environnement de test

Caractéristique	**Ordinateur**	**Serveur**
Type	Laptop	VPS
Système d'exploitation	macOS Moterey 12.6.7	Ubuntu 20.04.5 LTS
Mémoire Vive	16 Go, 2133 MHz LPDDR3	2 Go
Mémoire de stockage	512 Go SSD SATA	20 Go SSD SATA
Processeur	3,3 GHz Intel Core i7 double cœur	3,0 GHz 2 vCore

Les modules concernés par cette première série de tests sont ANIMETA-HIM et ANIMETA-API, puisqu'ils interviennent dans l'interfaçage avec les différents utilisateurs.

— Lors de la vérification de ANIMETA-HIM, l'objectif est de s'assurer qu'il garantit la création de modèles cohérents vis-à-vis de l'architecture. À cet effet, nous avons tenté d'établir des connexions entre tous les différents types de composants. Le résultat attendu pour ce test était de vérifier si les connexions acceptées étaient réellement possibles vis-à-vis de ANIMETA. Le tableau 5.2 présente les résultats obtenus à l'issue de ce test. On peut constater qu'il a été possible de connecter un composant *PartialKnowledge* à un composant *Sense* et à un composant *Task*. Or cela ne correspond pas à l'architecture fonctionnelle d'un animat. Cette erreur ainsi identifiée a été corrigée avant la poursuite des tests. D'un autre côté, nous avons également vérifié que lorsqu'un modèle est conçu via ANIMETA-HIM, il est restitué fidèlement au format JSON qui sera transmis à ANIMETA-API. Pour chaque composant du modèle conçu, nous l'avons identifié dans le contenu JSON, vérifié son emplacement (hiérarchie), vérifié la présence de tous les attributs ainsi que les valeurs associées. Cette opération a été un succès complet.

TABLE 5.2 – Résultats des tests de connexion de composants à composants

	Model	**Perceptions**	**Perception**	**Sense**	**Desires**	**Desire**	**Task**	**Action**
Model	Échec	Succès	Échec	Échec	Succès	Échec	Succès	Échec
Perceptions	Succès	Échec	Succès	Échec	Succès	Succès	Échec	Échec
Perception	Échec	Succès	Échec	Succès	Échec	Échec	Échec	Échec
Sense	Échec	Échec	Succès	Échec	Échec	Échec	Échec	Échec
Desires	Succès	Succès	Échec	Échec	Échec	Succès	Succès	Échec
Desire	Échec	Succès	Échec	Échec	Succès	Échec	Succès	Échec
Task	Succès	Échec	Échec	Échec	Échec	Succès	Échec	Succès
Action	Échec	Échec	Échec	Échec	Échec	Échec	Succès	Échec

	Partial Model	**Perceptions**	**Perception**	**Sense**	**Partial Desires**	**Partial Desire**	**Partial Task**	**Partial Action**	**Partial Knowledge**
PartialModel	Échec	Succès	Échec	Échec	Succès	Échec	Succès	Échec	Échec
Perceptions	Succès	Échec	Succès	Échec	Succès	Succès	Échec	Échec	Échec
Perception	Échec	Succès	Échec	Succès	Échec	Échec	Échec	Échec	Échec
Sense	Échec	Échec	Succès	Échec	Échec	Échec	Échec	Échec	Échec
PartialDesires	Succès	Succès	Échec	Échec	Échec	Succès	Succès	Échec	Échec
PartialDesire	Échec	Succès	Échec	Échec	Succès	Échec	Succès	Échec	Succès
PartialTask	Succès	Échec	Échec	Échec	Échec	Succès	Échec	Succès	Échec
PartialAction	Échec	Échec	Échec	Échec	Échec	Échec	Succès	Échec	Succès
Partial Knowledge	Échec	Échec	Échec	Succès	Échec	Succès	Succès	Échec	Échec

— La vérification de ANIMETA-API a principalement porté sur la conservation et la restitution fidèle des données traitées. Comme illustré dans la figure 5.3, les données traitées par ANIMETA-API sont non seulement différentes, mais elles proviennent également de sources différentes. Une attention particulière a donc été portée aux niveaux où des échanges ou des transformations de données se font.

Le premier niveau concerne les requêtes adressées à ANIMETA-API. Évidemment, ces requêtes ne proviendront pas uniquement de ANIMETA-HIM, pour lesquelles nous avons déjà certifié, après des tests, que les données envoyées sont cohérentes. Les requêtes peuvent provenir d'autres systèmes souhaitant intégrer ANIMETA et pourraient donc envoyer des données incohérentes. En conséquence, l'objectif central de cette vérification était de s'assurer du bon fonctionnement de la fonction de vérification des requêtes. Ainsi, à l'aide de l'application POSTMAN [9], toutes les requêtes ont été traitées avec différentes données d'entrée, qu'il s'agisse de données manquantes, tronquées, voire inexistantes. Pour toutes ces données, la fonction de vérification des requêtes de ANIMETA-API a systématiquement rejeté les requêtes mal formées.

Le second niveau concerne le flux de données entre l'API et la base de données [10]. Il s'agit ici de vérifier si les modèles (complets ou partiels) reçus sont correctement sauvegardés dans la base de données et s'ils sont fidèlement reconstitués. Cette vérification a été menée avec succès pour la totalité des tests effectués. Les données reconstituées depuis la base de données sont identiques en tout point aux données envoyées au départ. La figure 5.8 montre un exemple de ce cycle. Sur la figure 5.8a l'on présente un extrait du document JSON envoyé à ANIMETA-API, tandis que sur la figure 5.8b l'on montre le JSON reconstitué à partir de la base de données. Il est important de noter que l'organisation du document JSON à l'origine n'est pas exactement la même après reconstitution. La structure initiale est imposée par l'outil de développement, tandis que la structure après reconstitution est adaptée pour un traitement dans ANIMETA-ENGINE.

9. https://www.postman.com/
10. Base de données servant de stockage pour l'API. Elle est automatiquement définie par le framework Sails.Js

```
{
    "category": "PARTIALMODEL",
    "type": "Agent",
    "children": [{
            "category": "PERCEPTIONS",
            "text": "PERCEPTIONS",
            "children": [{
                "category": "PERCEPTION",
                "text": "PERCEPTION",
                "name": "P1",
                "strength_expression": "$1+3",
                "children": [{…
                }]
            }]
        },
        {
            "category": "DESIRES",
            "text": "DESIRES",
            "children": [{
                "category": "PARTIALDESIRE",
                "text": "PARTIALDESIRE",
                "name": "MonDesire",
                "weight": "",
                "threshold": "2",
                "interruption_factor": "1",
                "test_string": "true",
                "children": [{…
                    },
                    {
                        "category": "PERCEPTION",
                        "value": "P1"
                    },
                    {
                        "category": "PARTIALTASK",
                        "text": "PARTIALTASK",
                        "name": "",
```

```
{
    "perceptions": [
        {
            "senses": [
                {
                    "target": "prey",
                    "radius": 10,
                    "angle": 180,
                    "strength_expression": "2",
                    "default": "",
                    "category": "SENSE"
                }
            ],
            "name": "P1",
            "strength_expression": "$1+3",
            "category": "PERCEPTION"
        }
    ],
    "partialDesires": [
        {
            "perceptions": [
                "P1"
            ],
            "id": 10,
            "name": "",
            "weight": {
                "name": "weight",
                "type": "",
                "range_start_point": "1",
                "range_end_point": "10",
                "probability": 0,
                "category": "partialKnowledge"
            },
            "threshold": 2,
            "interruption_factor": 1,
            "test_string": "true",
```


(a) Document JSON généré par ANIMETA-HIM (b) Document JSON reconstruit par ANIMETA-API

FIGURE 5.8 – Exemple de données reçues, sauvegardées et reconstruites

5.2.2.2 Exécution des processus centraux de ANIMETA : ANIMETA-ENGINE

Ayant déjà à disposition le diagramme de séquence, la vérification de ANIMETA-ENGINE a concrètement débuté par l'étape 2, qui consiste à insérer des lignes de code supplémentaires pour afficher des informations à propos de l'exécution. Les principales informations affichées sont :

— L'identifiant unique de l'objet en cours d'exécution : Puisqu'il peut y avoir plusieurs instances de la même classe qui peuvent être créées, connaître l'identifiant unique de chaque instance permet de savoir quel objet exécute quelle fonction ;

— Le message «*Début/Fin d'exécution de [nom_de_la_fonction] par [nom_de_la_classe] : Identifiant unique = [Identifiant_Objet*» pour indiquer l'entrée et la sortie effective dans la séquence ;

— Les propriétés et les attributs de l'objet : le comportement de l'objet étant lié à ces attributs, les connaître est indispensable pour la vérification ;
— Les valeurs d'entrée (valeurs passées en paramètre) et les valeurs de sortie (valeurs retournées) s'il y en a. C'est le message le plus indicatif, puisqu'il permet de se rendre directement compte de la bonne exécution ou non de la fonction.

Lorsque nous avons réalisé les tests, qu'il s'agisse de ceux de la simulation de modèle ou de ceux de la génération de modèle, les messages insérés sont effectivement apparus dans la console, comme l'illustre la figure 5.9.

```
Agent predator created ID = 4565367248
      Action "RandomMove" Created ID = 4565367344
      Task named default_task created
         Action RandomMove ( id: 4565367344 ) added to Task with ID = 4565367392
      Task default_task  (id: 4565367392 ) added to Agent
Grid instance created. Size = 100 x 100 . ID = 4565367488
Simulation instance created ID = 4565367536
Grid (id: 4565367488 ) added to Simulation instance with id= 4565367536
Agent main_agent (id: 4485924944 )added to simulation
Agent prey (id: 4565367008 )added to simulation
Agent predator (id: 4565367248 )added to simulation
Simulation instance (id: 4565367536 ) Started
    Initializing main_agent in simulation with id= 4485924944
    Initializing prey in simulation with id= 4565367008
    Initializing predator in simulation with id= 4565367248
        Simulation running... ( time: 1 )
            Agent ( id: 4565368304 ) start his routine ...
             energy=500; name=Not defined; type=main_agent; position=(50, 50); orientation=None; velocity=(0, 0); acceleration=(0, 0); force=None;
                Getting perception named Faim id: 4565368544
                    Getting sense id: 4565368784
                    Getting sense id: 4565405808
                Getting perception named SeePredator id: 4565368688
                    Getting sense id: 4565406048
                Getting perception named Repos id: 4565405904
                    Getting sense id: 4565406288
                Deliberating...
                Executing actions...
                    Executing action name Sleep
            Agent ( id: 4565368304 ) Finish his routine !
             energy=500; name=Not defined; type=main_agent; position=(50, 50); orientation=None; velocity=(0, 0); acceleration=(0, 0); force=None;
```

Figure 5.9 – Affichage en console des messages de contrôle

La comparaison de l'ordre d'apparition des messages avec les différents diagrammes de séquence nous a permis de constater que la séquence attendue avait été respectée. Néanmoins, la remarque principale qui a été faite est qu'il y a des fonctions qui sont appelées à plusieurs reprises, notamment les fonctions d'instanciation d'objet. Cela nous a permis de nous rendre compte que parfois, plusieurs instances de classes sont créées, alors qu'une seule pourrait être utilisée.

À l'issue de tous ces tests, nous pouvons retenir en somme que notre protocole de vérification en cinq étapes s'est avéré essentiel pour garantir le bon fonctionnement de notre programme. La combinaison d'un diagramme d'activité détaillé, d'écritures en console, de valeurs de contrôle et de simulations a permis de détecter et de corriger les erreurs potentielles à chaque étape du processus de développement. Cette approche rigoureuse nous a permis de confirmer que l'implémentation que nous avons faite correspond parfaitement à l'approche proposée. Cela a accru notre confiance quant aux résultats que nous obtiendrons de ladite implémentation. Cependant, jusque là, elle ne garantit pas que les résultats obtenus sont

conformes à la réalité. En effet, notre seule certitude à ce stade, c'est que l'implémentation produit un résultat conforme à la façon dont nous avons pensé l'approche. Il reste toujours à savoir si ces résultats sont conformes à la réalité. Pour ce faire, nous avons procédé à la validation de l'approche avec divers types de modèles. Des modèles théoriques et des modèles réels avec divers niveaux de complexité. Tout comme pour la vérification, la validation s'est faite en deux parties. La validation de la modélisation et de la simulation, puis la validation de la génération de modèles.

5.3 Validation de la modélisation et simulation avec ANIMETA

Nos validations ont consisté à procéder à plusieurs tests qui nous permettent d'évaluer à quel point les résultats obtenus sont conformes à la réalité. De ce fait, il est obligatoire d'avoir à disposition des données réelles pour faire la comparaison, ce qui n'est pas toujours aisé à obtenir. Compte tenu de cette contrainte, pour chaque validation, nous optons, selon les possibilités, pour des données collectées directement chez des espèces réelles ou des données provenant de la simulation de modèles de comportement.

Pour nous assurer que notre approche permet de modéliser et d'obtenir, à la suite de leur simulation, des résultats cohérents avec la réalité, nous optons pour deux différents cas d'applications : un modèle d'espèce théorique et un modèle d'espèce réelle. Le choix d'un modèle théorique s'explique par les mêmes raisons que celles évoquées dans la section précédente. Quant au choix d'un modèle d'espèce réelle, il s'inscrit concrètement dans l'objectif de la validation, qui est d'évaluer la similitude des résultats de simulations avec le système réel. Les différentes comparaisons sont faites au moyen d'outils de visualisation et d'analyse statistique.

5.3.1 Validation avec un modèle expérimental de type «Proie-Prédateur»

Ce premier modèle de validation est le modèle présenté au chapitre 3, section 3.2, pour illustrer notre modèle générique du comportement animal. Il crée un environnement simulé où des interactions peuvent exister entre les différentes catégories d'agents qui le constituent. Cette simulation vise à s'assurer que notre modèle générique est capable de produire des résultats similaires aux modèles étudiant la dynamique des écosystèmes.

L'implémentation de ce modèle avec les différents individus a permis de tester notre approche de modélisation sur un modèle qui intègre différents types d'agents avec différents comportements. La figure 5.10 montre une capture d'écran d'une simulation réalisée avec le modèle.

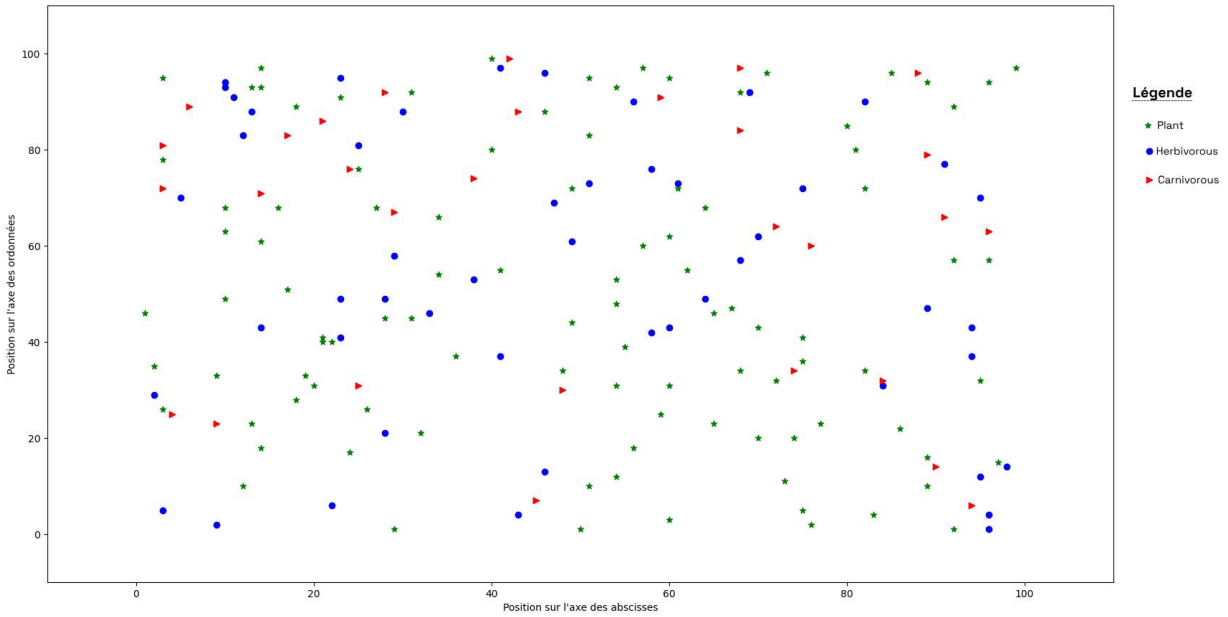


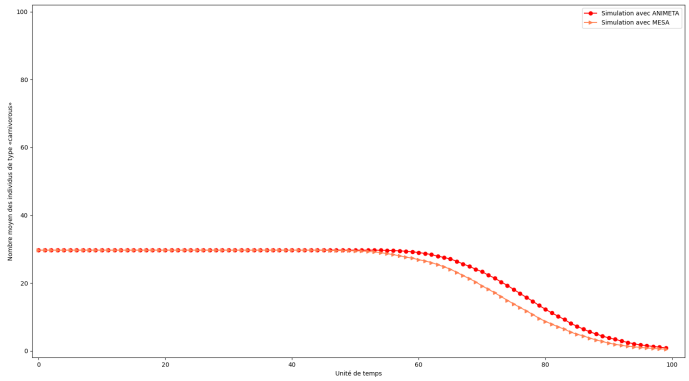
FIGURE 5.10 – Simulation du modèle décrit avec l'approche ANIMETA

Notre objectif dans cette phase de validation étant d'évaluer les différences qui peuvent apparaître entre une simulation réalisée avec l'approche ANIMETA et les autres approches de modélisation et de simulation présentées dans la section 1.2.2.7, il est indispensable d'implémenter ce même modèle dans au moins l'une de ces autres approches. La plateforme MESA a donc été choisie à cet effet en raison de sa proximité avec ANIMETA. En effet, ces deux approches partagent des principes très similaires au niveau de la structure, à savoir l'architecture de l'agent (actions réalisées par un ordonnanceur), l'organisation de l'environnement (environnement discret défini par des grilles), la gestion du temps (temps discret piloté par une horloge) et le langage d'implémentation (Python 3).

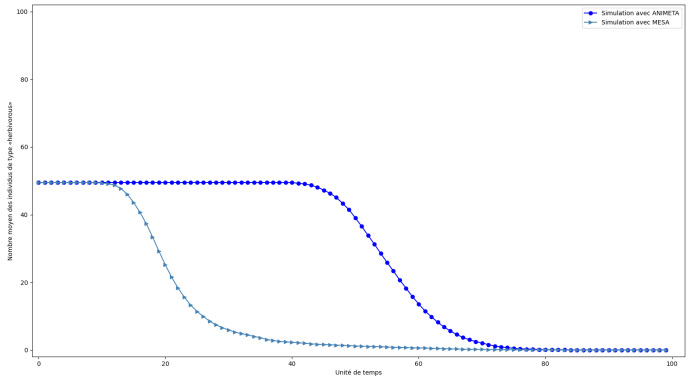
L'implémentation de notre modèle avec la plateforme MESA n'a pas été sans contraintes. Elle a été très laborieuse et très coûteuse en temps. Il a fallu écrire entièrement le code pour chaque type d'agent dans le modèle, ce qui n'est pas le cas pour ANIMETA qui permet de définir son modèle en quelques clics depuis ANIMETA-HIM.

Afin de faire les comparaisons, nous avons réalisé pour chaque approche une série de 100 simulations pour lesquelles nous avons collecté les données. Chaque simulation met en scène 100 agents de type «Plantes», 50 agents de type «Herbivores» et 30 agents de type «Carnivores».

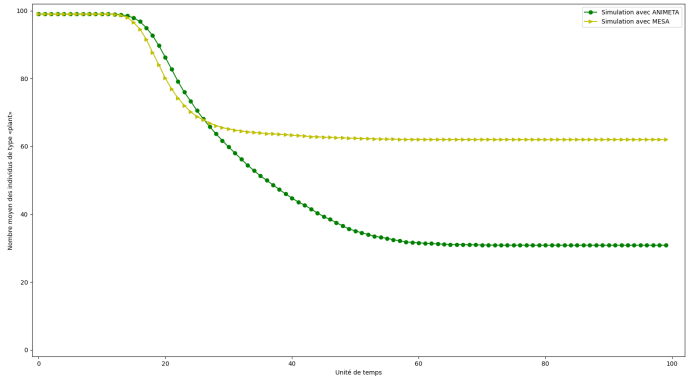
Dans cet ensemble de données collectées, le paramètre qui nous intéresse dans notre analyse est la taille de la population de chaque type d'agent. À cet effet, pour chaque approche et à chaque instant, nous avons calculé la valeur moyenne de la taille de la population de chaque type d'agent. Les résultats obtenus sont présentés dans la figure 5.11.



(a) Évolution moyenne de la population des agents «Carnivorous»



(b) Évolution moyenne de la population des agents «Herbivorous»



(c) Évolution moyenne de la population des agents «Plant»

FIGURE 5.11 – Évolution moyenne des populations d'agents

La figure 5.11a montre l'évolution moyenne de la population des agents de type «Carnivorous» pour la simulation avec ANIMETA et MESA. Nous pouvons constater visuellement que les deux courbes ont la même allure et sont quasiment confondues. Ce constat visuel est confirmé numériquement par les métriques du tableau 5.3 avec une erreur quadratique moyenne de 3,49 et une erreur maximale de 4,40. À l'échelle du modèle simulé, ces résultats sont parfaitement acceptables.

TABLE 5.3 – Métriques de comparaison de l'évolution de la population de «Carnivorous»

Métrique	**Valeurs**
Erreur Quadratique Moyenne (MSE)	3.485
Racine carrée de l'Erreur Quadratique Moyenne (RSME)	1.867
Erreur Médiane Absolue	0.168
Erreur Maximum	4.376
Distance de Hausdorff	3.038

Sur la figure 5.11b, nous avons l'évolution moyenne de la population des agents «Herbivorous» pour chaque plateforme. Nous constatons également qu'ici, les deux courbes semblent avoir la même allure, mais avec un important décalage (voir tableau 5.4).

TABLE 5.4 – Métriques de comparaison de l'évolution de la population de «Herbivorous»

Métrique	**Valeurs**
Erreur Quadratique Moyenne (MSE)	633.44
Racine carrée de l'Erreur Quadratique Moyenne (RSME)	25.168
Erreur Médiane Absolue	5.584
Erreur Maximum	47.198
Distance de Hausdorff	33.577

En effet, pour la simulation avec la plateforme MESA, la décroissance de la population commence autour de la quinzième unité de temps pour s'annuler peu avant la soixantième unité de temps, tandis que pour la simulation avec ANIMETA, la décroissance commence beaucoup plus tard, autour de la quarantième unité de temps, pour s'annuler peu avant la quatre-vingtième unité de temps.

Deux raisons peuvent expliquer l'apparition de ce décalage dans les résultats de simulation :

— La possibilité de mémorisation de ANIMETA : Comme décrit dans la section 3, lors de la simulation d'un animat, plusieurs composants *Desire* peuvent être activés, c'est-à-dire ajoutés à une file d'attente qui est déroulée selon la priorité de chaque élément qui la constitue. En ramenant ce fonctionnement à notre cas d'application, il permet à un agent de type «Herbivorous» qui perçoit à un instant un agent «Carnivorous» et un agent «Plant», de fuir l'agent «Carnivorous» et de revenir après se nourrir de l'agent «Plant». Ceci permet à l'agent de survivre plus longtemps avant de mourir. Compte

tenu des contraintes imposées par l'architecture d'un agent dans MESA, nous n'avons pas pu implémenter cette fonctionnalité de la même manière.

— Les positions initiales des agents au début de la simulation : Prenons l'exemple de la figure 5.12. Dans cet exemple, la plupart des agents de type herbivorous sont plus proches des agents de type carnivorous et plus éloignés de ceux de type Plant. Dans ces conditions, les agents de type herbivorous sont plus exposés à la prédation (ce qui peut rapidement entraîner une baisse de leur population) et ont moins de chances de se nourrir. Sans apport d'énergie, ils meurent rapidement. De plus, cette proximité avec les agents de type carnivorous déclenche plus souvent le comportement de fuite au détriment de celui de se nourrir. Finalement, ils perdent beaucoup plus d'énergie qu'ils n'en gagnent et meurent rapidement.

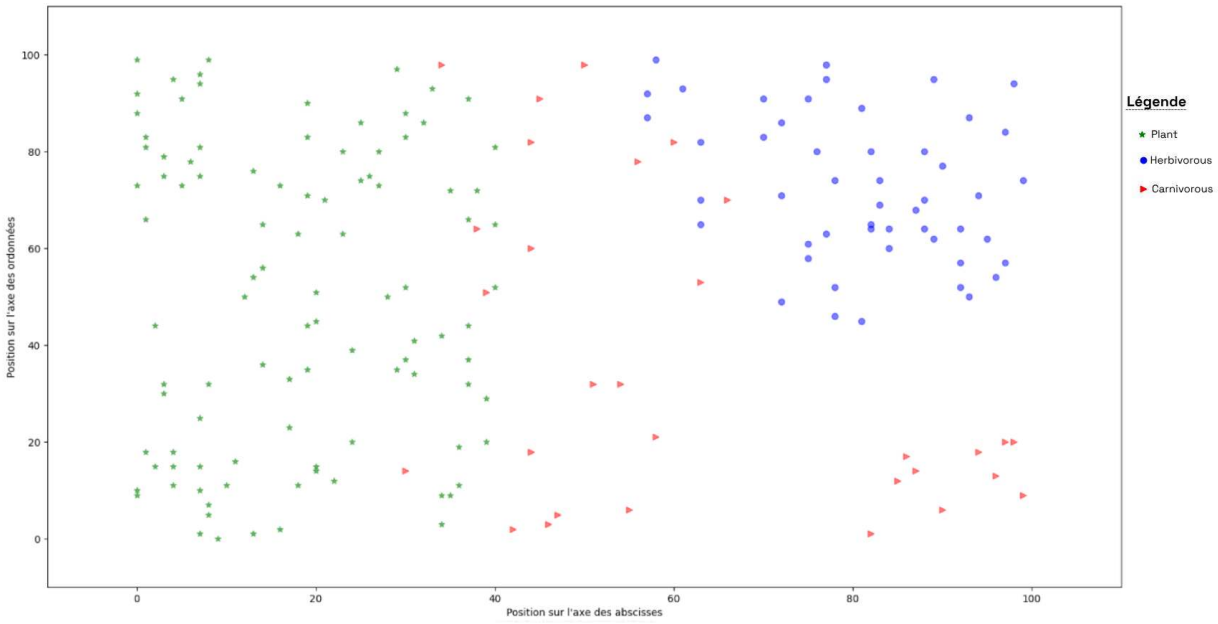
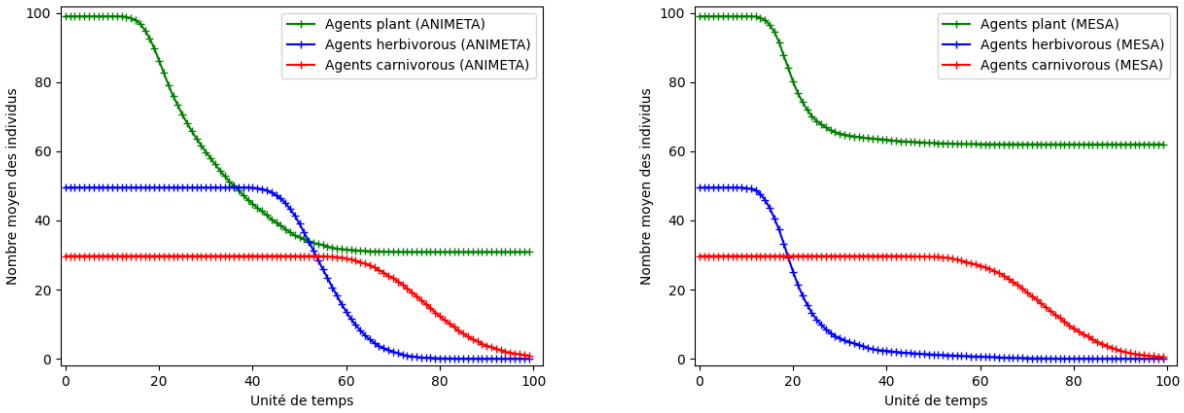


FIGURE 5.12 – Exemple de position initiale des agents à la simulation

Un décalage similaire est également observé dans la figure 5.11c, qui montre l'évolution moyenne de la population des agents de type «Plant». La population commence à décroître un peu avant la vingtième unité de temps pour les deux approches. En revanche, elle se stabilise à environ 70 individus pour la plateforme MESA, tandis qu'elle se stabilise à environ 35 individus pour ANIMETA. Cette observation découle logiquement de l'évolution de la population des agents «Herbivorous». En effet, sur la figure 5.13, lorsque l'on superpose l'évolution de la population des agents «Herbivorous» et des agents «Plant», nous remarquons que la population des agents «Plant» se stabilise au fur et à mesure que la population des agents «Herbivorous» diminue.



(a) Population des agents simulés avec ANIMETA

(b) Population des agents simulés avec MESA

FIGURE 5.13 – Évolution de la population des agents simulés avec ANIMETA et MESA

De tout ce qui précède, ce que nous pouvons retenir de cette expérimentation, c'est que les résultats de simulation avec l'approche ANIMETA, d'une part, correspondent à ceux attendus pour la simulation d'un tel modèle, et d'autre part, sont proches de ceux obtenus avec d'autres approches, notamment celui de la plateforme MESA que nous avons utilisée à titre de comparaison. La conclusion majeure que nous pouvons tirer à ce stade, c'est qu'en utilisant l'approche ANIMETA, nous pouvons modéliser et simuler convenablement un modèle de comportement d'animaux et obtenir des résultats qui correspondent à ceux attendus. Toutefois, le décalage (bien que justifié) observé entre les résultats nous suggère de pousser plus loin notre validation sur un autre modèle où les raisons évoquées pour le décalage n'influencent pas les résultats. Pour cela, nous réalisons une autre validation dans la section suivante.

5.3.2 Validation avec un modèle de comportement de morsure de queue chez les cochons

Ce second modèle de comportement nous sera particulièrement utile pour analyser les hypothèses émises à la suite de la validation du modèle présenté précédemment.

5.3.2.1 Présentation du modèle

Le modèle que nous utilisons pour cette validation est tiré de (IJMM BOUMANS et al. 2016) et modélise le comportement de morsure de queue chez les cochons. Nous l'appellerons «Modèle B». Il présente pour nous plusieurs intérêts, car son fonctionnement présente des similitudes avec celui d'ANIMETA (par exemple : hiérarchisation des comportements, seuil de déclenchement, motivation interne, etc.), favorisant ainsi une meilleure comparaison des résultats. Au-delà de cet aspect, le modèle implémenté directement par ses concepteurs dans NetLogo 5.1.0 est disponible en téléchargement [11].

11. https://www.comses.net/codebases/5010/releases/1.3.0/ consulté le 10 Octobre 2023 à 11h16

Le modèle B met en œuvre dans un enclos un groupe de 10 cochons élevés dans des systèmes d'élevage intensifs stériles. Les cochons sont représentés par des agents qui ont la possibilité d'adopter six différents comportements, à savoir : le sommeil, le repos, l'alimentation, l'exploration, le mouvement et la morsure de queue. Ces comportements sont déterminés par les variables *sleeping_drive*, *resting_drive*, *biting_drive*, *exploring_drive* et *feeding_drive* qui évoluent tout au long de la simulation selon les règles précisées dans le tableau 5.5. La description complète de chaque comportement est rapportée dans la documentation officielle, comme indiqué dans le tableau 5.6.

TABLE 5.5 – Mise à jour des variables par unité de temps selon le comportement

Variable	**Mise à jour des variables par unité de temps selon le comportement**					
	Agent *pig* en cours d'exécution					
	Sommeil	Repos	Alimentation	Exploration	Mouvement	Morsure
exploring_drive	-0,10	0,17	0,05	-	0,17	-
feeding_drive	0,09	0,09	-1,11	0,18	0,18	0,18
sleeping_drive	-0,20	0,17	0,21	0,21	0,21	0,21
biting_drive	-0,06	-	-	0,276	-	-0,09
	Agent *pig* perçu					
	Sommeil	Repos	Alimentation	Exploration	Mouvement	Morsure
biting_drive	-	-	-	-	-	0,05
sleeping_drive	-	-	-	-	-	-0,20

TABLE 5.6 – Description des comportements réalisable par les agents «cochon»

Unité de comportement	**Actions réalisés**
L'alimentation	Le cochon se déplace vers un espace d'alimentation, se nourrit, et met à jour ses variables.
Le sommeil	Le cochon se rapproche du cochon le plus proche avec la plus grande valeur de *sleeping_drive* et met à jour ses variables selon le tableau 5.5.
L'exploration	Le cochon choisit aléatoirement une direction et essaie de se déplacer sur une distance de 2 unités. S'il réussit, il met à jour ses variables.
La morsure de queue	Le cochon se déplace vers le cochon le plus proche, le choisit comme victime, le mord, se déplace aléatoirement vers une autre position, et met à jour ses variables.
Le mouvement	Le cochon se déplace aléatoirement vers une zone libre, puis met à jour ses variables.
Le repos	Le cochon se rapproche du cochon le plus proche et met à jour ses variables.

Ces agents évoluent dans un environnement (enclos) constitué par les cochons eux-mêmes et un espace où ils vont se nourrir, comme représenté par la figure 5.14. Chacun des 4 comportements possibles par l'agent est décrit dans le tableau. Ces comportements étant hiérarchisés, la figure 5.15 montre comment, à chaque pas de temps, le choix du comportement à exécuter est fait.

FIGURE 5.14 – Capture d'écran d'une simulation du modèle B : sol en béton (gris), un espace d'alimentation (noir) et dix cochons (IJMM BOUMANS et al. 2016).

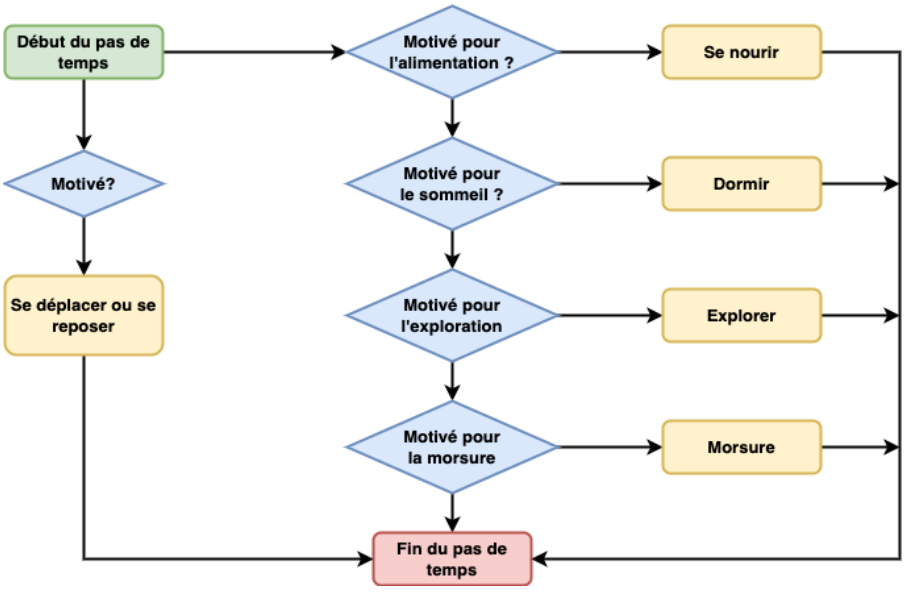


FIGURE 5.15 – Diagramme d'exécution des comportements par chaque agent du modèle B à chaque pas de temps.

5.3.2.2 Construction du modèle

Le modèle B met en œuvre un seul type d'agent, les cochons. En analysant la description des comportements de ce type d'agent, nous pouvons regrouper les six unités de comportement en cinq, en combinant le repos et le mouvement dans une seule unité de comportement. En effet, l'action conjuguée de ces deux unités correspond à l'action élémentaire «RandomMove» disponible dans ANIMETA. Ainsi, pour modéliser l'animat «cochon» (pig), nous aurons à créer 4 composants *Desire* pour quatre des unités de comportement et un composant *Task* en tant que tâche par défaut qui implémente la cinquième unité de comportement, comme décrit ci-dessous.

Unité 1 : L'alimentation (Feeding)

Le déclenchement de ce comportement est conditionné par un seul composant *Perception* (*feeding_perception*) défini dans le tableau 5.7. Il se compose de deux composants *Sense*. Le premier permet de récupérer l'état interne (niveau de faim, défini par feeding_drive) et le second vérifie si une source de nourriture (Feeding_space) est présente dans la zone de perception. La force de la perception (*strength_expression = $1*) est exclusivement celle du composant *Sense* qui renvoie la valeur de feeding_drive.

Table 5.7 – Configuration du composant *Perception* feeding_perception d'un agent *Pig*

Attribut	**Propriétés**	
name	feeding_perception	
Sense 1	*target*	@
	boolean_expression	False
	angle	Null
	radius	Null
	strength_expression	@.*feeding_drive*
Sense 2	*target*	Food
	boolean_expression	False
	angle	360
	radius	10
	strength_expression	1
strength_expression	$1	

Le comportement d'alimentation déclenché par la perception *feeding_perception* est défini par le composant *Desire* décrit dans le tableau 5.8. Il passe à l'état activable (c'est-à-dire ajouté à la file d'attente) lorsque la force de la perception franchit le seuil défini (propriété *activator*). Dans le modèle d'origine, comme dans le composant *Desire*, le seuil est 0. En nous référant au diagramme de la figure, nous pouvons constater que le comportement d'alimentation est celui ayant le poids le plus important. À cet effet, nous lui attribuons la valeur 4. Tous les autres composants *Desire* devront donc avoir un poids inférieur à 4.

La tâche associée à ce comportement peut être définie par deux actions élémentaires de ANIMETA, à savoir *GetCloser* (se rapprocher) et *Eat* (consommer). L'agent va donc se rapprocher de la cible et la consommer. Cependant, de façon native, l'action *Eat* fait

disparaître la cible concernée. Comme ici, il ne s'agit pas de faire disparaître la source de nourriture, nous allons donc configurer l'attribut *kill* avec la valeur *False* pour obtenir le résultat attendu. Cette configuration montre une fois encore la flexibilité qu'offre ANIMETA dans la modélisation. Cette flexibilité va nous permettre également d'ajouter des instructions pour mettre à jour certaines variables de l'agent, telles que définies dans le modèle d'origine, mais qui ne sont pas nativement prises en compte par les actions élémentaires. Ceci se fait au moyen du composant *Unimplemented*, qui est aussi, par polymorphisme, une action que l'utilisateur définit lui-même par des instructions de la syntaxe de ANIMETA (voir tableau 3.2). Pour les actions *GetCloser* et *Unimplemented*, remarquez que la durée (duration_cycle) est mise à 0 pour indiquer que ce sont des actions qui s'exécutent en même temps que l'action *Eat*.

Table 5.8 – Configuration du composant *Desire* du comportement d'alimentation

Attribut	**Propriétés**
name	feeding_desire
test_string	$@.feeding\_drive <= 0$
activator	$\pi.strength > \Delta.threshold$
perceptions	[feeding_perception]
weight	4
threshold	1
interruption_factor	1
priority	1
Task	(Voir tableau 5.9)

Table 5.9 – Configuration du composant *Task* du composant *Desire* du comportement d'alimentation

Attribut	**Propriétés**		
name	feeding_desire_task		
Actions	GetCloser	*duration_cycle*	0
		max_attempt	10
	Unimplemented	*duration_cycle*	0
		instructions	@.exploration_drive += 0.05, @.feeding_drive += -1.11, @.sleeping_drive += 0.21, @.feeding += 1
	Eat	*duration_cycle*	1
		gain	0
		move_cost	0
		kill	False

Unité 2 : Le sommeil (Sleeping)

Le déclenchement de cette unité de comportement est réalisé par la perception *sleeping_perception*. Tout comme le composant *Perception* présenté précédemment, le principe est le même, c'est-à-dire 2 composants *Sense*, dont l'un pour récupérer la variable *sleeping_drive* et l'autre pour percevoir un autre agent de type cochon (avec la plus grande

valeur de *sleeping_drive*) duquel il va se rapprocher pour le sommeil. ANIMETA, avec sa flexibilité, offre une fois encore la possibilité de préciser cette particularité de la perception grâce à l'attribut *min_max*. La manière dont ce composant *Perception* est donc défini est présentée par le tableau 5.10.

TABLE 5.10 – Configuration du composant *Perception* sleeping_perception d'un agent *Pig*

Attribut	**Propriétés**	
name	sleeping_perception	
Sense 1	*target*	@
	boolean_expression	False
	angle	Null
	radius	Null
	strength_expression	@.*sleeping_drive*
Sense 2	*target*	Pig
	boolean_expression	False
	angle	360
	radius	10
	strength_expression	1
min_max	sleeping_drive	
strength_expression	$1	

Le composant *Desire* associé à cette perception est décrit dans le tableau 5.11. Le principe reste le même que pour le composant *Desire* défini précédemment. Le seuil et la condition d'activation sont similaires. La différence se trouve au niveau du poids (weight) qui est mis ici à 3, conformément à la description de la figure 5.15. Quant aux actions qui forment la tâche à réaliser, nous en avons deux. L'action *GetCloser* pour se rapprocher d'un compère pour le sommeil et une action *Unimplemented* pour définir la mise à jour à effectuer sur les variables internes. Afin de préciser que l'action *GetCloser* se réalise en fonction de l'agent ayant la plus grande valeur de *sleeping_drive*, son attribut *target* est défini comme *maximum_agent*.

TABLE 5.11 – Configuration du composant *Desire* du comportement du sommeil

Attribut	**Propriétés**
name	sleeping_desire
test_string	@.*sleeping_drive* <= 0
activator	$\pi.strength > \Delta.threshold$
perceptions	[sleeping_perception]
weight	3
threshold	0
interruption_factor	1
priority	1
Task	(Voir tableau 5.12)

TABLE 5.12 – Configuration du composant *Task* du composant *Desire* du comportement du sommeil

Attribut	**Propriétés**		
name	sleeping_desire_task		
Actions	Unimplemented	*duration_cycle*	0
		instructions	@.exploration_drive += -0.10, @.feeding_drive += 0.09, @.sleeping_drive += -0.20, @.biting_drive += -0.06, @.sleeping += 1
	GetCloser	*duration_cycle*	1
		max_attempt	10
		target	*maximum_agent*

Unité 3 : L'exploration (Exploration)

Contrairement aux deux précédentes unités de comportement, la perception associée à celle-ci n'est définie que par un seul composant *Sense* qui informe sur la variable *exploration_drive*. En ce qui concerne le composant *Desire*, il reste dans le même style que les deux précédentes. Ce qui change principalement, c'est la tâche associée, que nous décrivons dans le tableau 5.13. Elle est ici constituée de l'action native *RandomMove*, configurée pour reproduire le comportement attendu, puis d'une action *Unimplemented*, configurée elle aussi pour mettre à jour les variables.

TABLE 5.13 – Configuration du composant *Task* du composant *Desire* du comportement d'exploration

Attribut	**Propriétés**		
name	sleeping_desire_task		
Actions	Unimplemented	*duration_cycle*	0
		instructions	@.feeding_drive += 0.18, @.sleeping_drive += 0.21, @.biting_drive += 0.276, @.exploration += 1
	RandomMove	*duration_cycle*	1
		max_attempt	10
		max_velocity	$(2, 2)$

Unité 4 : La morsure (Biting)

Bien qu'étant le comportement principalement étudié par ce modèle, dans ANIMETA, sa conception reste tout à fait simple et similaire aux trois autres précédemment définies. Conformément à la description de ce comportement, la perception qui peut le déclencher est conditionnée par les deux éléments que sont la valeur de la variable *biting_drive* et la vue d'un autre cochon comme victime pour la morsure. Ces deux éléments sont définis et présentés par les composants *Sense* du tableau 5.14.

TABLE 5.14 – Configuration du composant *Perception* biting_perception d'un agent *Pig*

Attribut	**Propriétés**	
name	biting_perception	
Sense 1	*target*	@
	boolean_expression	False
	angle	Null
	radius	Null
	strength_expression	@.*biting_drive*
Sense 2	*target*	Pig
	boolean_expression	False
	angle	360
	radius	10
	strength_expression	1
strength_expression	$1	

Le composant *Desire* associé à cette perception est présenté dans le tableau 5.15. Il comporte quelques particularités, notamment pour le seuil de déclenchement et les actions de la tâche. En effet, dans (IJMM BOUMANS et al. 2016), il est précisé que pour représenter les variations individuelles, le seuil de déclenchement est défini aléatoirement, basé sur une distribution normale avec une moyenne de $0,5$ et une déviation standard de $0,05$. Ces critères sont définis dans le composant *Desire* au moyen de la classe native *Random_initializer* paramétrée à cet effet. En ce qui concerne les actions de la tâche, la particularité se trouve dans le déroulement même de l'action de morsure. En effet, ici, elle affecte à la fois le mordeur et la victime. Dans ANIMETA, les actions s'effectuant de base sur l'agent qui perçoit, nous utilisons une autre action native appelée *Reverse_Unimplemented*, qui a la même fonction que *Unimplemented* utilisée précédemment, à la seule différence que les cibles sont inversées [12]. Ainsi, dans la tâche, nous avons trois actions :

— L'action *GetCloser* qui permet à l'agent de se rapprocher de l'autre agent choisi comme cible ;
— L'action *Unimplemented* pour mettre à jour les variables de l'agent «mordeur»
— L'action *Reverse_Unimplemented* pour mettre à jour les variables de l'agent «mordu».

TABLE 5.15 – Configuration du composant *Desire* du comportement de la morsure

Attribut	**Propriétés**
name	biting_desire
test_string	$@.biting\_drive <= 0$
activator	$\pi.strength > \Delta.threshold$
perceptions	[biting_perception]
weight	1
threshold	$Random\_initializer(min = 0, max = 1, mean = 0.5, deviation = 0.05)$
interruption_factor	1
priority	1
Task	(Voir tableau 5.16)

12. La différence entre *Unimplemented* et *Reverse_Unimplemented* se trouve au niveau d'agent qui subit l'action. Dans *Unimplemented*, c'est l'agent lui même qui subit l'action alors que dans *Reverse_Unimplemented* il s'agit de l'agent perçu

TABLE 5.16 – Configuration du composant *Task* du composant *Desire* du comportement de la morsure

Attribut	**Propriétés**		
name	biting_desire_task		
Actions	Unimplemented	*duration_cycle*	0
		instructions	@.feeding_drive += 0.18, @.sleeping_drive += 0.21, @.biting_drive -= 0.09, @.bite += 1
	Reverse_Unimplemented	*duration_cycle*	0
		instructions	@.sleeping_drive -= 0.20, @.biting_drive += 0.05, @.bitten += 1
	GetCloser	*duration_cycle*	1
		max_attempt	10

En ce qui concerne le poids de cette tâche, elle est mise à 1 car selon la description elle représente l'unité de comportement de moindre poids (voir figure 5.15).

Conformément à la figure 5.15, lorsque l'agent n'est pas motivé (dans ANIMETA c'est lorsqu'aucun composant *Desire* n'est activé), celui-ci déplace ou se repose avec respectivement les probabilités 0,14 et 0,86. Ce comportement peut être modélisé dans ANIMETA comme tâche par défaut avec l'action native *RandomMove* et une action *Unimplemented* pour la mise à jour des variables tel que défini par le tableau 5.17.

TABLE 5.17 – Configuration du composant *Task* pour la tâche par défaut.

Attribut	**Propriétés**		
name	Pig_default_task		
Actions	Unimplemented	*duration_cycle*	0
		instructions	@.exploration_drive += 0.17, @.feeding_drive += 0.135, @.sleeping_drive += 0.19, @.resting += 1 ; @.movement += 1
	RandomMove	*duration_cycle*	1
		move_probability	0.14
		max_attempt	10
		max_velocity	$(2, 2)$

Avec tous ces éléments mis bout à bout, nous construisons entièrement le modèle basé sur les agents du *cochon* dans le cadre de l'étude de son comportement de morsure de queue. Ce comportement étant influencé par l'environnement dans lequel se trouve le cochon, nous devons également le modéliser conformément à la description faite dans (IJMM BOUMANS et al. 2016). Rappelons-le, l'environnement est formé par un enclos dans lequel se trouvent un espace d'alimentation et d'autres cochons. Dans notre modélisation, l'élément qui nous manque, c'est l'espace d'alimentation. Étant donné que dans ANIMETA, tous les éléments sont des animats, nous pouvons donc représenter cet espace par un ensemble d'animats (de type *«Food»*) perceptibles par les cochons. Ce type d'animat sera inactif et n'implémentera

aucun composant *Perception* ni aucun composant *Desire*. Il n'aura pas non plus de tâche par défaut.

5.3.2.3 Résultats obtenus

Le modèle décrit a été implémenté et simulé avec ANIMETA, comme le montre la figure 5.16. Ainsi, pour faire notre validation, nous avons réalisé une série de 100 simulations sur 720 unités de temps pour lesquelles toutes les données ont été collectées. Le nombre de simulations et le temps de simulation sont exactement les mêmes que ceux utilisés dans (IJMM Boumans et al. 2016) qui représentent 24 heures (1 unité de temps correspond à 2 minutes). Ils sont jugés représentatifs pour une journée de vie d'un cochon. Dans le cadre de ces travaux, le modèle a été utilisé pour répondre à trois questions de recherche qui sont :

1. Quel est le temps moyen que les cochons passent pour chaque comportement ?
2. Quelles sont les proportions de groupes de cochons (neutre, victime, mordeur, victime et mordeur) qui se forment ?
3. À quels moments développent-ils ces comportements au fil du temps ?

La validation de notre modèle dans ANIMETA sera alors de comparer nos résultats obtenus à ceux obtenus dans (IJMM Boumans et al. 2016).

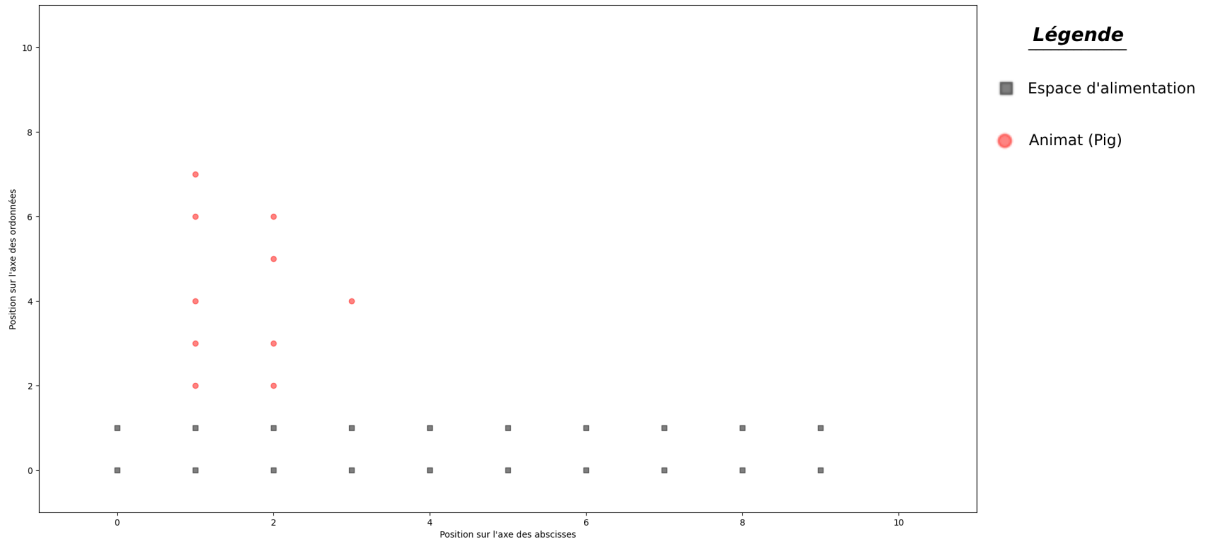


Figure 5.16 – Capture d'écran de la simulation du modèle de (IJMM Boumans et al. 2016) avec ANIMETA.

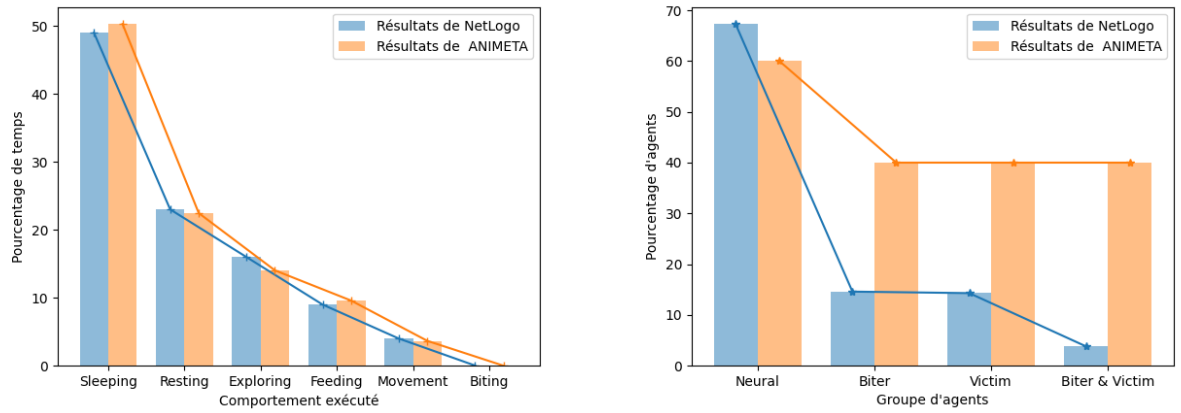
Pour répondre à la première question de recherche, nous avons déterminé, à partir des données collectées dans nos simulations, les valeurs moyennes de temps pour l'ensemble des agents «cochon» pour chaque comportement. Ces valeurs ont été représentées sur un même diagramme que celles du modèle d'origine, comme illustré dans la figure 5.17a. Pour des raisons de clarté de nos explications, nous désignerons le modèle ANIMETA par le terme «modèle A» et celui de (IJMM Boumans et al. 2016) par le terme «modèle B».

Visuellement, nous remarquons une grande similitude entre les résultats du modèle A et ceux du modèle B, qui lui est implémenté dans NetLogo. Le temps moyen (arrondi) du

comportement de sommeil est de 49% pour le modèle B et de 50% pour le modèle A. Pour le comportement de repos, il est de 23% pour le modèle B et le modèle A. Il est de 16% pour le modèle B et de 14% pour le modèle A lorsqu'il s'agit du comportement d'exploration. Pour le comportement d'alimentation, le temps moyen passé par les agents est de 9% pour les 2 modèles. S'agissant du comportement de déplacement (mouvement), il est également le même pour les deux modèles à savoir de 4%. Enfin, pour le comportement principalement étudié, qui est celui de la morsure de queue, le temps passé par les agents du modèle B est de 0,03%, dans une fourchette de 0% à 0,08% de comportement de morsure de queue. Pour le modèle A, il est de 0,06%.

Lorsque nous déterminons l'écart entre les deux résultats en utilisant l'erreur quadratique moyenne, nous obtenons la valeur $1,33$. En nous basant sur la fourchette de 0% à 0,08% obtenue pour le comportement de morsure dans le modèle B et que nous la ramenons à l'échelle de l'ensemble des comportements étudiés, nous pouvons juger les résultats comme étant en accord avec ceux attendus. Nous pouvons donc tirer une première conclusion partielle, qui est que ANIMETA permet bien de répondre à la première question de recherche.

La deuxième question de recherche étant d'analyser la proportion des groupes de cochons qui se forment, nous avons exploité les résultats de nos simulations, pour déterminer les quatre groupes ayant fait l'objet de cette étude qui se forment. Lesdits groupes sont : le groupe des neutres (neutral), le groupe des mordeurs (biter), le groupe des victimes (victim) puis le groupe des mordeurs et des victimes (biter & victim). Les résultats des simulations (série de 100) (IJMM Boumans et al. 2016) montrent qu'en moyenne, 67,3% des cochons sont restés neutres, 14,6% sont des mordeurs, 14,3% sont des victimes et 3,8% sont devenus à la fois mordeurs et victimes. La comparaison de ces résultats à ceux que nous avons obtenus est présentée par le graphique de la figure 5.17b.



(a) Pourcentage du temps par unité de comportement

(b) Répartition des catégories d'agents

Figure 5.17 – Comparaison des résultats de ANIMETA avec ceux de NetLogo

Nous remarquons qu'une différence palpable entre nos résultats moyens et ceux du modèle d'origine. Cette différence est d'autant plus remarquable lorsque l'on prend séparément les résultats de chaque simulation comme montré par la figure 5.18. Nous remarquons que d'une simulation à une autre, les résultats peuvent être très différents les uns des autres, mais aussi

différents des résultats du modèle d'origine. Néanmoins, les résultats de la simulation montrés à la 5.18b semblent être proches de ceux attendus. À la lumière de ces constats, il apparaît que le modèle tel qu'implémenté dans ANIMETA ne permet pas de répondre avec assurance à cette question de recherche.

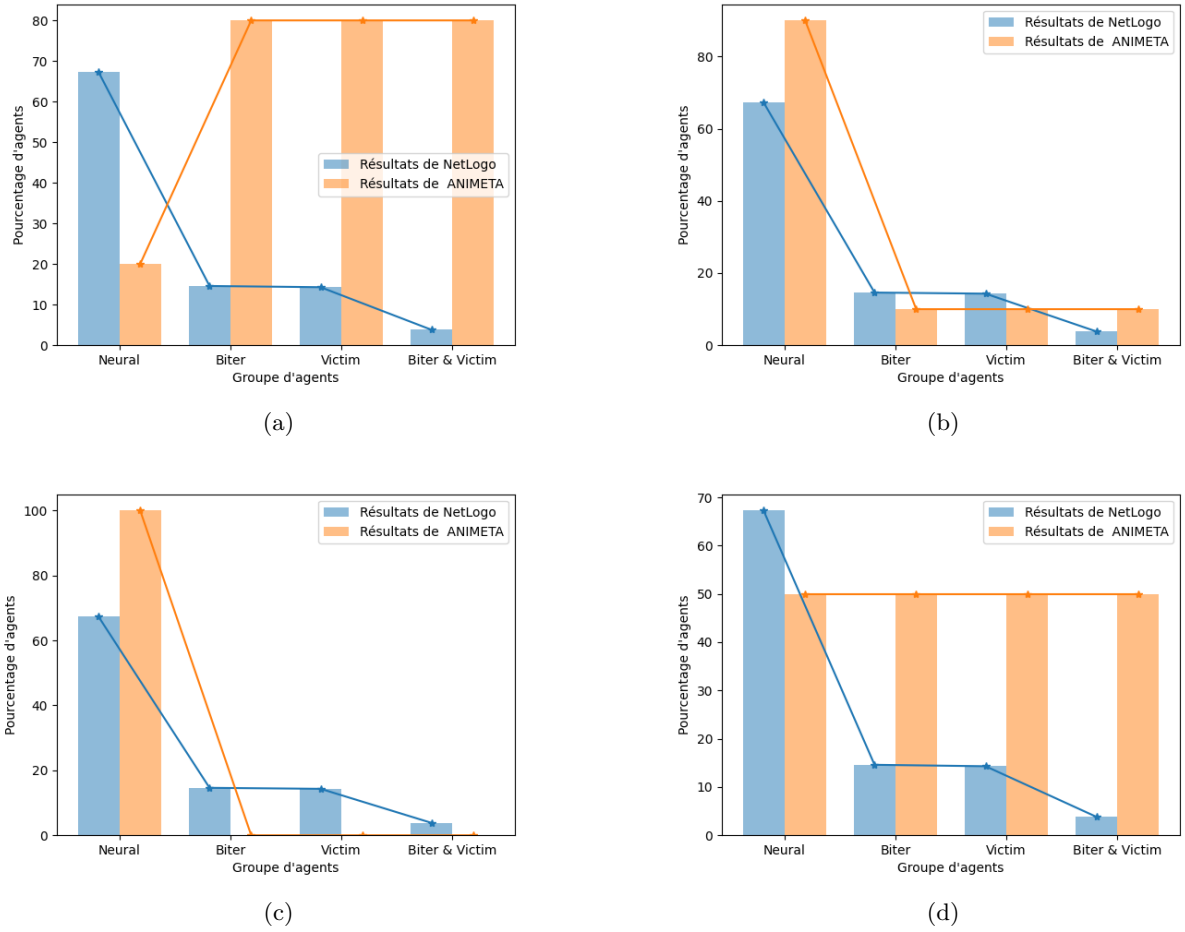
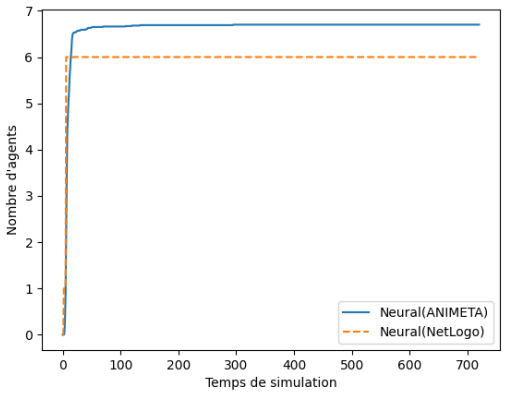
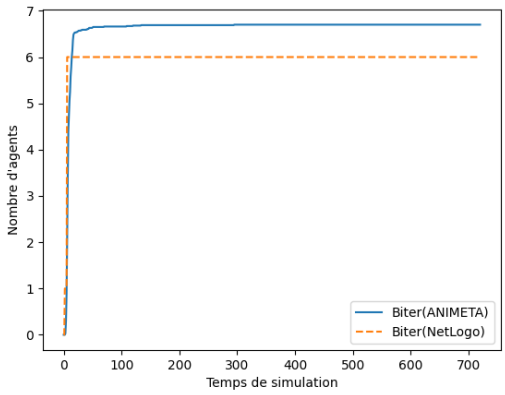


FIGURE 5.18 – Comparaison de quelques résultats de simulation de ANIMETA aux résultats de (IJMM BOUMANS et al. 2016)

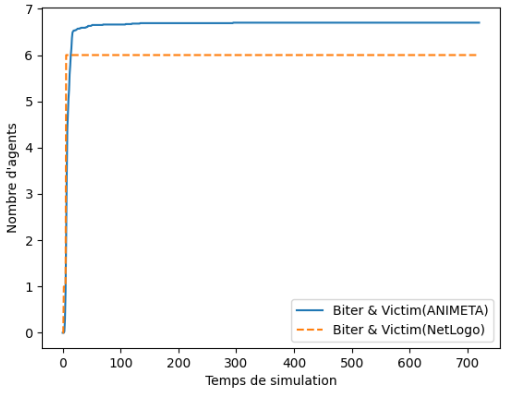
En comparant également les résultats des deux approches portant sur les moments où les cochons développent ces différents comportements au fil du temps (voir figure 5.19), pour répondre à la troisième question de recherche, nous remarquons ici que nos résultats sont en meilleure adéquation avec ceux attendus. Ceci se confirme par les courbes des résultats de ANIMETA qui ont les mêmes tendances que celles du modèle d'origine. Les écarts moyens pour chaque groupe de cochon sont présentés dans le tableau 5.18. Ces valeurs, ramenées à l'échelle de notre étude, voudront donc dire que nos résultats sont exacts sur la moyenne à un 1 agent près. Nous pouvons donc dire que la modélisation avec ANIMETA permet également d'apporter des réponses raisonnables à la troisième question de recherche de (IJMM BOUMANS et al. 2016).



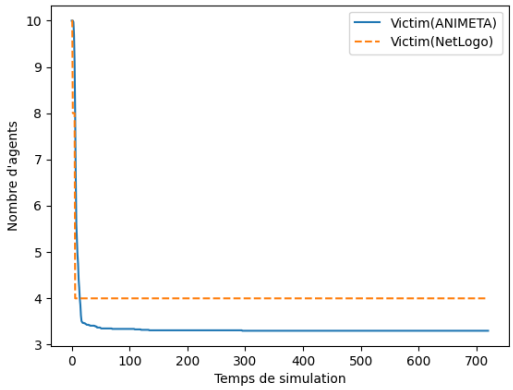
(a) Comparaison de l'évolution pour le groupe des agents mordeurs



(b) Comparaison de l'évolution pour le groupe des agents victimes de morsures



(c) Comparaison de l'évolution pour le groupe des agents à la fois mordeurs et victimes



(d) Évolution pour le groupe des agents neutres

FIGURE 5.19 – Comparaison de l'évolution du développement des groupes d'agents

TABLE 5.18 – Configuration du composant *Desire* du comportement de la morsure

Groupes	**Ecart Moyen**
Mordeurs (Biter)	0.505
Victimes (Victim)	0.505
Mordeurs et victimes (Biter & Victim)	0.505
Neutres (Neutral)	0.518

En résumé, cette série de validations de notre approche de modélisation a livré des résultats très encourageants. Nous avons pu constater que cette approche fonctionne efficacement, fournissant des prédictions qui s'alignent de manière satisfaisante avec la réalité, que ce

soit dans le cadre de modèles théoriques ou lorsqu'elle est appliquée à des cas concrets. Toutefois, il est important de garder à l'esprit qu'il existe encore des situations (comme celle de la deuxième question de recherche de (IJMM Boumans et al. 2016)) où ANIMETA ne peut garantir une précision totale, mais avec quand même une certaine similitude. Il est donc essentiel de rester conscient de ces limites de cette modélisation, qui constituent des perspectives intéressantes pour ces travaux. En fin de compte, elles posent des bases significatives pour une meilleure exploitation dans le processus de génération de modèle, qui est aussi une partie importante de tout le projet ANIMETA.

5.4 Validation de la génération de modèles avec ANIMETA

Après nous être rassuré lors de la phase de vérification que ANIMETA fonctionne comme il a été conçu et implémenté, il est à présent convenable de procéder à une validation approfondie qui garantit ainsi la solidité des fondements sur lesquels reposent l'approche présentée. Deux espèces ont été le sujet de notre validation à savoir le Sciaena umbra (Corb) et une espèce conceptuelle.

5.4.1 Validation avec des juvéniles de corbs (Sciaena umbra)

Le Sciaena umbra, communément appelé Corb en Corse, est une espèce de poisson qui vit dans les eaux côtières de la Méditerranée, dont les côtes corses font partie de son habitat privilégié. Ce poisson est reconnaissable par sa silhouette élancée, arborant une coloration argentée avec des reflets dorés, tandis que sa tête et son dos présentent souvent des teintes plus sombres. Le corb est un poisson emblématique d'intérêt écologique, économique et patrimonial pour la Corse, où il est considéré aujourd'hui comme menacé d'extinction par l'UICN(Ducos 2023). Heureusement, le corb est une espèce dont la plateforme STELLA MARE a réussi à maîtriser et est aujourd'hui capable de produire en écloserie plus de 80 000 de juvéniles par an, dont une bonne partie est relâchée dans le milieu naturel.

Figure 5.20 – Juvénile de l'espèce Sciaena umbra ©Salomé Ducos

Ces juvéniles ont fait l'objet d'une étude portée sur leur réponse de fuite suite à un stimulus. Cette étude, réalisée par (Ducos 2023), s'inscrit dans le cadre de l'évaluation de leur performance d'évasion afin de proposer des conditions de relâcher qui favorisent leur survie.

5.4.1.1 Présentation de l'expérimentation de (Ducos 2023)

L'expérimentation a été faite avec 182 juvéniles de Sciaena umbra, répartis en quatre classes de taille distinctes (40-50 mm, 50-60 mm, 60-70 mm et 70-80 mm). Les poissons ont été soumis à une stimulation mécanique dans une arène expérimentale de 35 x 35 cm, avec une profondeur d'eau d'environ 7 cm (voir figure 5.21).

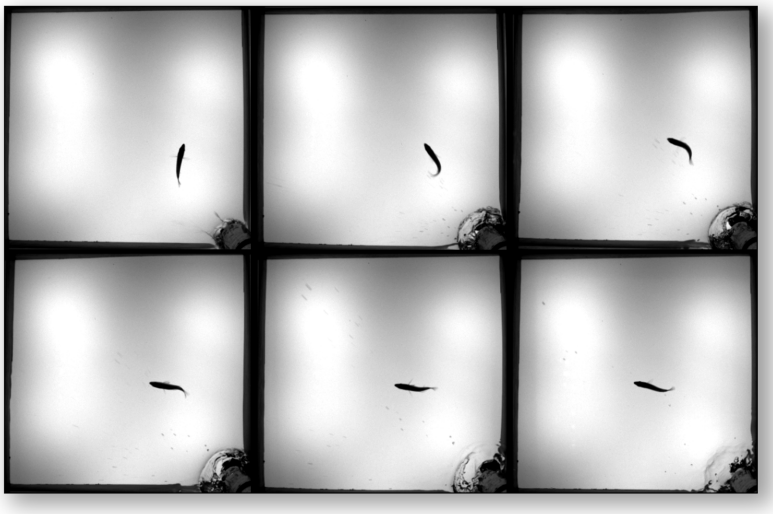


FIGURE 5.21 – Enclos avec des juvéniles de corbs filmés par une caméra ©Salomé Ducos

Chaque poisson a été stimulé mécaniquement avec la chute soudaine d'un objet suspendu à un électroaimant à l'extrémité d'un tube en PVC opaque, permettant l'identification précise du moment de la stimulation en fonction de la première onde perceptible résultant de l'impact de l'objet sur la surface de l'eau. Les réponses d'évasion ont été enregistrées par une caméra à 300 images par seconde. Les variables comportementales et locomotrices ont été étudiées, notamment la réactivité, la latence avant la réponse d'évasion, la durée de la réponse d'évasion, la distance cumulative, la vitesse maximale et la durée moyenne de la réponse d'évasion. À la suite de chaque stimulation mécanique, les individus ont été catégorisés comme «répondants» ou «non-répondants». La réactivité, exprimée en pourcentage, ainsi que la latence avant le déclenchement de la réponse d'évasion, ont été minutieusement évaluées pour chaque individu. De plus, la durée totale de la réponse d'évasion a été mesurée. Ces analyses ont été conduites en tenant compte des classes de taille et des conditions d'élevage des poissons.

5.4.1.2 Génération d'un modèle de réponse de fuite de juvénile de corb avec ANIMETA

Les comportements observés dans les vidéos peuvent être modélisés dans ANIMETA, par deux unités de comportement que sont :

— Une tâche par défaut : ici, comme dans la plupart des cas, elle est modélisée par un déplacement aléatoire, donc l'action RandomMove.
— Un composant *Desire* : Le cadre posé par le modèle à générer nous apporte des informations qui seront utiles dans notre modélisation. En effet, (Ducos 2023) mesure le

temps de latence (temps entre la simulation et la réponse de fuite) et la durée de la réponse. Cette situation, ramenée à ANIMETA, pourrait être modélisée par une tâche exécutant successivement deux actions :

— L'action élémentaire *Void* représente une action qui ne fait « rien ». Son objectif est de mettre une pause dans l'exécution de l'unité de comportement pour un certain temps, sans changer les paramètres internes de l'animat. La diversité des individus peut être représentée dans l'action *Void* par une loi normale de moyenne $m = 0$ et d'écart type ϵ en tant que paramètre à déterminer. La valeur choisie par tirage aléatoire dans cette distribution est multipliée par la durée de base de l'action à laquelle le résultat est ajouté.

— L'action élémentaire *Runaway* comprend plusieurs paramètres qui nous intéressent, notamment la durée de fuite (*duration_cycle*) et l'accélération maximale (*acceleration_max*). Dans le cadre de l'expérimentation, nous avons ajouté un décorateur à la classe pour inclure deux autres paramètres : *move_probability* et *diversity*. Le paramètre *move_probability* représente la probabilité qu'un poisson réponde à la simulation, tandis que *diversity* représente l'écart type de la loi normale avec une moyenne $m = 0$.

Les paramètres du composant *Desire* ne seront pas utiles dans le cas de cette modélisation puisqu'il n'y a qu'un seul composant *Desire*. Le paramètre *threshold*, quant à lui, peut être directement mis à 1 pour signifier que la fuite est déclenchée dès qu'il y a au moins un stimulus.

En intégrant toutes ces informations dans ANIMETA, nous obtenons le modèle partiel de la figure 5.22. Ce modèle partiel devra donc être complété avec des valeurs optimales trouvées au cours du processus d'optimisation.

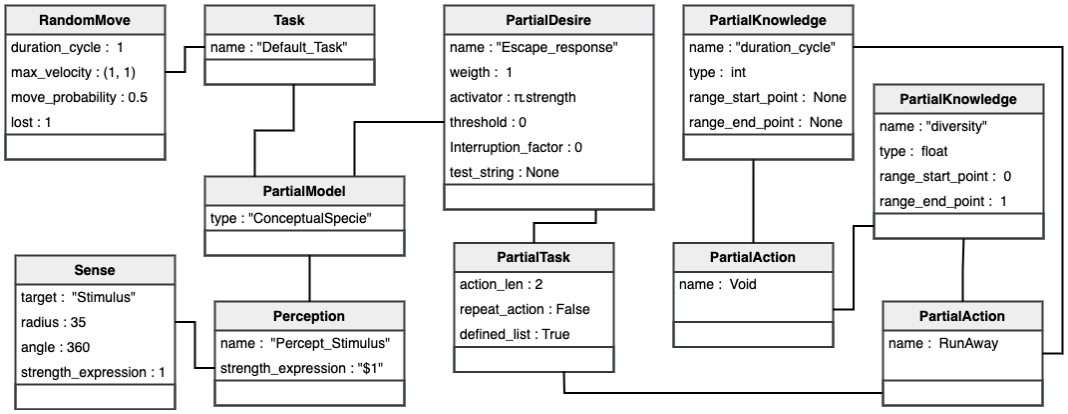


FIGURE 5.22 – Diagramme d'objet du modèle partiel de réponse de fuite chez les juvéniles de corbs

5.4.1.3 Résultats obtenus

Le paramétrage pour réaliser l'optimisation, ainsi que les résultats obtenus à l'issue de celle-ci, sont résumés dans le tableau 5.19. La fonction d'évaluation utilisée est l'Erreur quadratique moyenne, qui mesure, pour le modèle et pour les données de référence, l'écart entre la distance parcourue, la vitesse moyenne, le temps de latence et la durée de la réponse.

TABLE 5.19 – Résultats obtenus à la génération de modèles de réponse de fuite

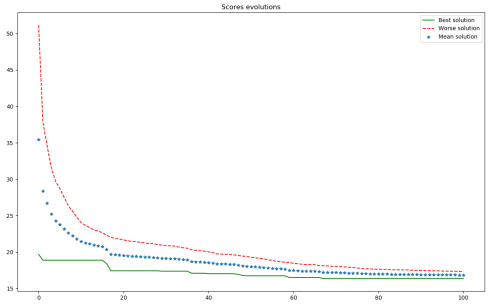
Algorithme	Paramétrage	Solution Obtenue	Itérations	Évaluations	Durée
Génétique *(Détails sur les paramètres au tableau 4.2)*	nb. individus (n) : 100 nb. croisements (n_c) : 10 nb. nouveaux (n^+) : 5 nb. itération (n_{it}) : 100 Proba. mutation (p_m) : 0.20 Score limite (f_{opt}) : 10	**Action Void** duration_cycle : 14 ϵ : 0.07 **Action Runaway** duration_cycle : 3 acceleration_max : 2 u.l/u.t^2 move_probability : 0.62 diversity : 0.18	Exécutées : 100 Solution : 67	Best : 16.37 Wrose : 17.35	02 :05 :29
Harmonique *(Détails sur les paramètres au tableau 4.6)*	nb. harmoniques (n) : 100 Proba. improvisation (sp_s) : 0.8 Proba. d'ajustement (p_{aj}) : 0.8 Taux d'ajustement (t_{aj}) : 0.2 Proba. technique (p_{choix}) : 0.5 nb. itération (n_{it}) : 100 Score limite (f_{opt}) : 10	**Action Void** duration_cycle : 10 ϵ : 0.50 **Action Runaway** duration_cycle : 4 acceleration_max : 2 u.l/u.t^2 move_probability : 0.04 diversity : 0.27	Exécutées : 100 Solution : 69	Best : 16.78 Wrose : 21.55	00 :50 :59
Immunitaire *(Détails sur les paramètres au tableau 4.5)*	nb. anticorps (n) : 100 nb. nouveaux (n^+) : 10 nb. tentative ($n_{attempt}$) : 4 nb. itération (n_{it}) : 100 Score limite (f_{opt}) : 10	**Action Void** duration_cycle : 14 ϵ : 0.07 **Action Runaway** duration_cycle : 2 acceleration_max : 1 u.l/u.t^2 move_probability : 0.63 diversity : 0.18	Exécutées : 100 Solution : 66	Best : 16.38 Wrose : 18.99	06 :04 :55
Fourmis *(Détails sur les paramètres au tableau 4.4)*	nb. solutions (n) : 100 nb. fourmis (n_f) : 100 proba. choix (p_{choix}) : 0.8 Score limite (f_{opt}) : 10	**Action Void** duration_cycle : 15 ϵ : 0.82 **Action Runaway** duration_cycle : 1 acceleration_max : 6u.l/u.t^2 move_probability : 0.11 diversity : 0.12	Exécutées : 100 Solution : 38	Best : 20.08 Wrose : 32.05	02 :31 :45

Le meilleur modèle obtenu à l'issue de cette optimisation est celui obtenu par l'algorithme de la recherche harmonique. Il modélise la réponse de fuite chez les juvéniles de corb avec deux actions :

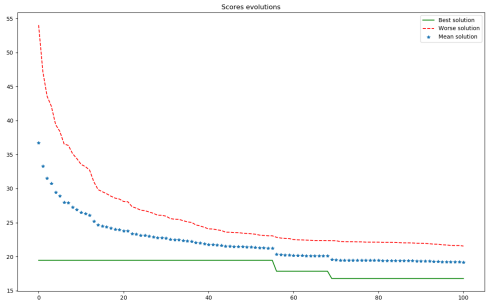
— L'action *Void* représente le temps de latence et est configurée pour une durée de 14 u.t. La diversité des individus est représentée par une loi normale de moyenne $m = 0$ et d'écart type $\epsilon = 0.07$.

— L'action *RunAway* est paramétrée avec une durée de 3 u.t. La probabilité de l'agent de répondre ou non au stimulus est de 0.62, et la durée de l'action est de 2 u.t. L'accélération maximale pour la réponse de fuite est de 2 u.l/u.t^2. La diversité est représentée par une loi normale de moyenne $m = 0$ et d'écart type 0.18 (*diversity*).

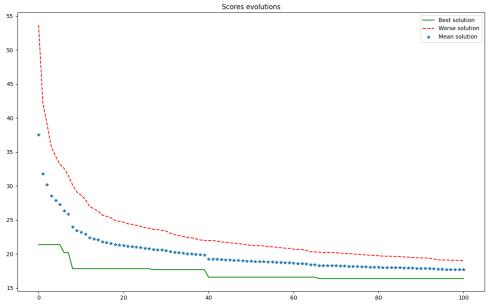
L'élément le plus frappant au vu de ces résultats est le score des solutions optimales. Il reste encore assez élevé par rapport au score optimal attendu. De plus, ils sont quasiment les mêmes et sont obtenus dès les premières itérations et ne changeront pas jusqu'à la fin comme l'indique la figure 5.23.



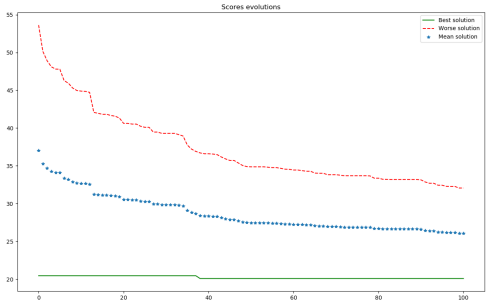
(a) Algorithme génétique



(b) Algorithme de la recherche harmonique



(c) Algorithme immunitaire



(d) Algorithme de colonie des fourmis

FIGURE 5.23 – Évolution des scores au fil des itérations pour chaque algorithme

L'explication possible pour ces résultats est que les algorithmes n'ont peut-être pas pu proposer de meilleures solutions, atteignant ainsi l'optimum global possible. Cependant, l'algorithme des colonies de fourmis a montré son inefficacité face à ce problème, car la solution finale trouvée présente un score encore élevé.

Néanmoins, en examinant de plus près les données du modèle obtenu, nous remarquons que, en moyenne, les résultats semblent assez proches de ceux présents dans le jeu de données de référence, avec une variation de 2 à 3 unités pour chaque variable observée (voir figure 5.24), avec une meilleure adaptation pour la durée de la réponse.

En effectuant une comparaison plus approfondie à la figure 5.25, des données produites par le meilleur modèle obtenu avec les données de référence, nous remarquons que la dynamique reste quand même maintenue, avec des données qui couvrent en majorité les mêmes intervalles. Il ne serait évidemment pas possible d'avoir exactement les mêmes répartitions de données, puisque le modèle généré n'est pas tout à fait parfait, comme en témoigne son score.

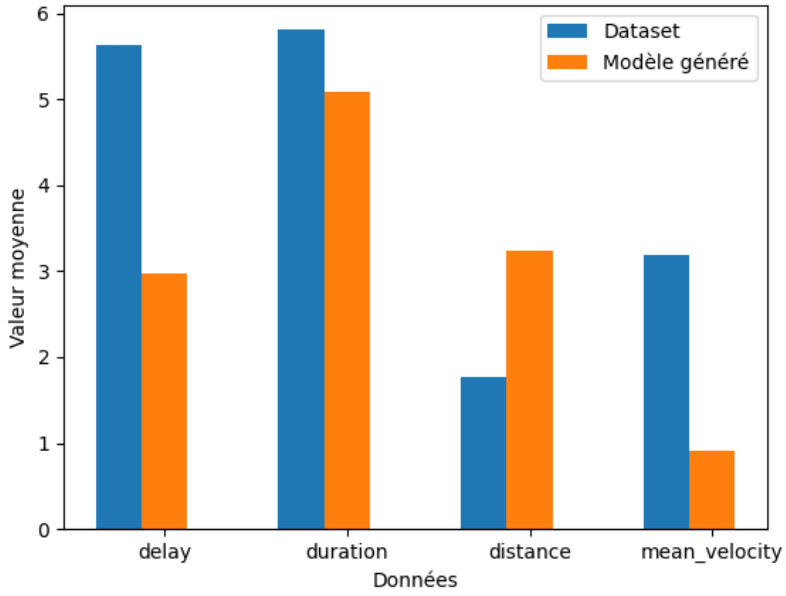
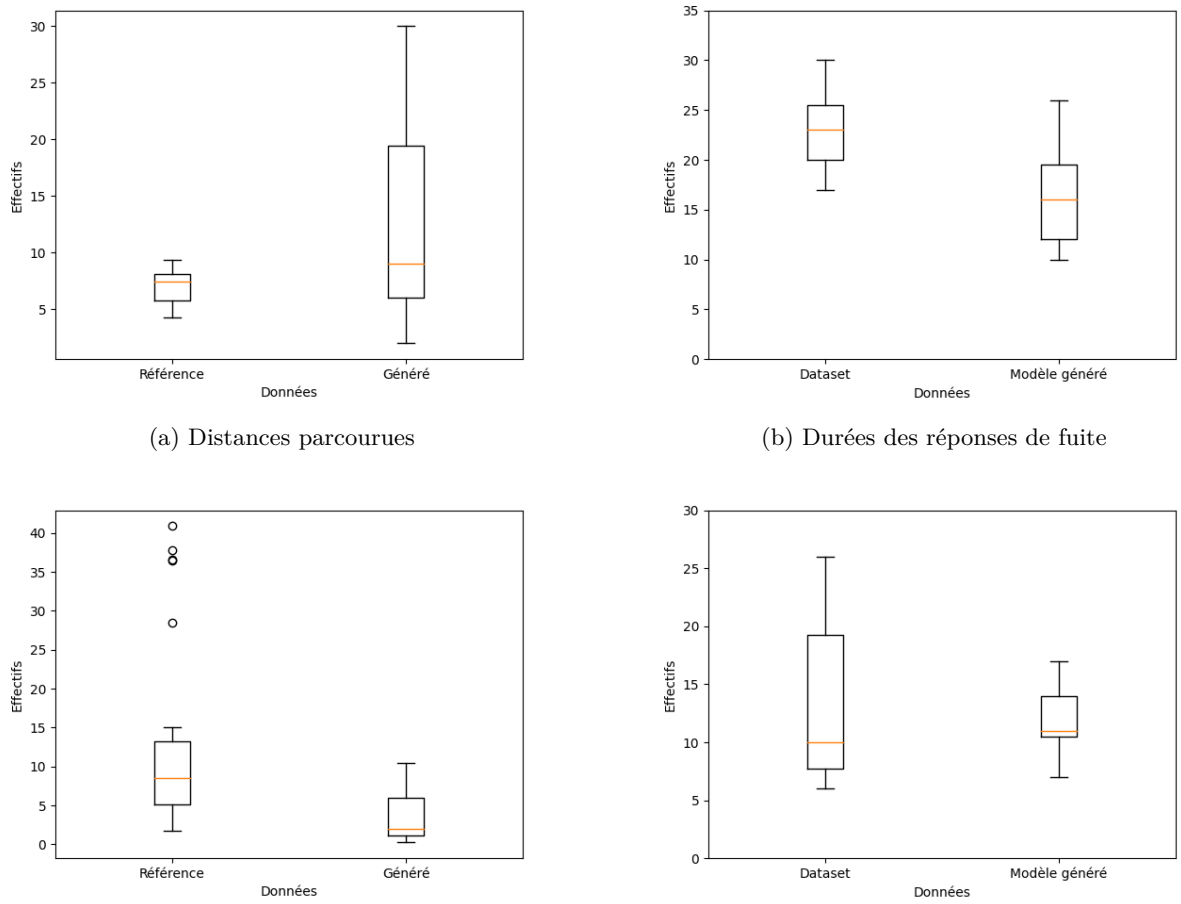


Figure 5.24 – Comparaison des valeurs moyennes des variables étudiées

Une hypothèse qui pourrait expliquer ces différences est que les variables étudiées pourraient ne pas être suffisantes pour modéliser de manière précise la réponse de fuite. Une autre piste à envisager serait la qualité des données de référence. En effet, on peut d'ailleurs voir sur la figure 5.25c quelques points aberrants.

D'autres facteurs qui ne sont pas pris en compte dans notre étude, et pour lesquels nous n'avons pas de données, tels que l'alimentation ou l'acclimatation dans le milieu, pourraient permettre d'améliorer la précision du modèle généré comme l'a précisé (Ducos 2023). Pour cela, nous aurions besoin de connaître tous les facteurs qui influencent un comportement, ainsi que les données qui leur sont associées. Dans cette situation, une espèce imaginaire pour laquelle nous aurions une totale maîtrise serait l'idéal pour valider l'approche que nous proposons.



(a) Distances parcourues

(b) Durées des réponses de fuite

(c) Vitesses moyenne de la fuite

(d) Temps de latence

FIGURE 5.25 – Comparaison statistique des données du modèle généré

5.4.2 Validation avec une espèce conceptuelle

5.4.2.1 Présentation et conception du modèle expérimental

Au vu des résultats obtenus avec l'expérimentation sur les corbs, nous optons, dans ce second cas d'application, pour un modèle expérimental d'une espèce conceptuelle. Le choix d'un tel modèle que nous appellerons «Modèle C» s'inscrit toujours dans la logique d'avoir un plein contrôle sur le modèle et de générer autant de données de divers types que possible. Le modèle utilisé ici est une forme avancée du modèle de type « proie-prédateur » présenté dans le chapitre 3, section 3.2. Ici, nous avons également trois catégories d'agents, à savoir : Prédateur (predator), espèce théorique (ConceptualSpecie) et plante (Plant). La description de leurs unités de comportement est faite dans le tableau 5.20.

TABLE 5.20 – Description des unités de comportement de modèle C

Catégorie	Description
Predator	— Lorsque leur niveau d'énergie est inférieur à 300, ils se nourrissent des agents de la catégorie «ConceptualSpecie». Le niveau d'énergie de l'individu est augmenté de 1000 points à chaque agent consommé. — Ils peuvent percevoir les éléments dans un champ de vision décrit à partir de leur position par un secteur circulaire d'angle 90° et de rayon 10 *U.l*. — Ils passent la pluspart de leur temps à déambuler dans leur environnement avec une vitesse maximale de 5 $u.l/u.t$. Le mouvement à chaque instant est conditionné par une probabilité de 0,3. — À chaque mouvement déambulatoire, il perd 1,5 unité d'énergie par u.l. — Lorsque le niveau d'énergie d'un individu atteint 0, il meurt.
ConceptualSpecie	— Ces agents passent la totalité de leur temps à se reposer et à essayer de vivre le plus longtemps possible, d'une part en faisant des épisodes de sommeil (la nuit ou en journée) lorsque l'environnement y est favorable. — Sur une journée, les agents font un sommeil entre 22h00 et 4h00 si l'environnement le leur permet (par exemple, s'il n'y a pas de prédateurs dans leur entourage). Ils font un court sommeil entre 14h00 et 16h00 à condition d'avoir un niveau d'énergie supérieur à 300 et d'avoir dans leur voisinage au moins un de leurs pairs. Il est évident qu'en milieu réel, ces actions ne peuvent pas se réaliser aussi ponctuellement, mais pour les besoins de l'expérimentation, nous allons les considérer comme telles. — Le niveau d'énergie est augmenté de 250 et de 500 chaque heure de sommeil respectivement pour la sieste et le sommeil prolongé. — Les agents sont capables de percevoir les prédateurs autour d'eux (360°) jusqu'à une distance de 5 *u.l*. Lorsque c'est le cas, ils les évitent de sorte à se maintenir à une distance supérieure à 10 *u.l*. Dans ce comportement de fuite, ils perdent 1.5 unité d'énergie par unité de longueur parcourue. — Pour survivre, lorsqu'ils ont un niveau d'énergie inférieur à 200, ils se nourrissent des plantes qui sont à moins d'une distance de 10 *u.l*. Chaque plante consommée fait augmenter le niveau d'énergie de 100. Ils arrêtent de se nourrir lorsque leur niveau d'énergie atteint 700. Dans cette quête de nourriture, ils perdent 0.25 unité d'énergie par unité de longueur parcourue. — Lorsque les agents ne sont pas stimulés pour aucun des comportements présentés, ils adoptent un comportement déambulatoire qui se manifeste soit par un déplacement aléatoire à une vitesse maximale de 3 $u.l/u.t$, soit par la formation d'une colonne [13] dont le mouvement, également aléatoire, est dirigé par un leader.
Plantes	— Cette catégorie se situe au bas de la chaîne alimentaire. Les individus de cette catégorie ne peuvent pas se déplacer.

La première étape de la validation a consisté à générer des données de référence, «la matière première » pour le processus. Pour ce faire, nous avons construit le modèle avec les outils de ANIMETA, avons simulé et sauvegardé avec les résultats produits. En nous inspirant du modèle B (IJMM Boumans et al. 2016), les simulations sont réalisées sur 720 unités de temps, soit 24 heures. À partir de l'analyse de la description du tableau 5.20, nous aurons à modéliser les trois animats de types « Plante », « ConceptualSpecie » et « Predator », sachant que le type « ConceptualSpecie » est celui qui fait l'objet de notre analyse.

— Pour les agents de type «Plant», aucun composant *Desire* ni aucun composant *Perception* n'est nécessaire, et leur tâche par défaut ne comporte aucune action.

— Les agents de type « Predator » peuvent être modélisés avec un seul composant *Perception* qui déclenche le comportement leur permettant de se nourrir des agents de type « ConceptualSpecie ». Il est composé de deux composants *Sense*, l'un qui vérifie la condition liée à l'énergie (énergie < 300) et l'autre qui détecte la présence d'un agent de type « ConceptualSpecie » dans le champ de perception. À ce composant *Perception* est associé un composant *Desire* dont la tâche est constituée uniquement de l'action élémentaire « Eat ». Quant à la tâche par défaut, elle est constituée de l'action élémentaire « RandomMove ».

— Les agents de type « ConceptualSpecie » sont les plus complexes de ce modèle. L'analyse du tableau 5.20 nous révèle une diversité de comportements soumise à des conditions variées. Nous pourrons les modéliser avec 6 unités de comportement, dont 5 composants *Desire* et une tâche par défaut pour représenter respectivement l'évitement des prédateurs, l'alimentation, le sommeil prolongé, la sieste, la formation de troupeau et le comportement déambulatoire. La définition formelle suivant le modèle *ANIMETA* est présentée en annexe 5.4.2.6.2. Néanmoins, nous décrivons ci-dessous le choix des différents paramètres de la configuration.

1. L'évitement des prédateurs (nous l'appellerons «*lifesaving*») est modélisé par un composant *Desire* avec le plus grand poids. Étant donné que nous avons 5 composants *Desire* au total pour modéliser les agents, le poids prend donc la valeur 5. Le composant *Desire* est déclenché par un composant *Perception* configuré de sorte qu'il puisse détecter tout agent de type «Predator» situé au plus à une distance de 5 u.l. Lorsqu'un composant *Desire* est activé (c'est-à-dire ajouté à la file d'attente), il n'est désactivé que lorsque l'agent s'éloigne du prédateur d'au moins 10 u.l.

2. L'alimentation (nous l'appellerons «*feeding*») est modélisée par un composant *Desire* dont le poids est 4. Il est déclenché par un composant *Perception* qui détecte tous les agents «Plant» jusqu'à une distance de 10 u.l. et lorsque le niveau d'énergie passe en dessous de 200. La désactivation du composant *Desire* est effective lorsque l'agent atteint la satiété (énergie > 700).

3. Le sommeil prolongé («*sleeping*») est modélisé par un composant *Desire* avec un poids mis à 3, car nous pensons qu'il est moins important que les deux précédents dans la survie de l'animat. Son déclenchement est conditionné par un composant *Perception* qui récupère l'objet «Timer» de l'environnement lors de la simulation. Cet objet peut être assimilé à une horloge et est donc capable de donner à tout

moment le temps de simulation. Le composant *Desire* est donc activé à l'heure du sommeil (22h00, soit 420 u.t dans la simulation). Il est désactivé lorsqu'arrive l'heure du réveil, c'est-à-dire 04h00 (120 u.t).

4. Le composant *Desire* pour la sieste («*napping*») est semblable à celui du sommeil prolongé, si ce n'est le poids et les conditions de déclenchement qui changent. Le poids ici est fixé à 2. Le déclenchement quant à lui est conditionné par la validité d'une perception qui vérifie la condition du temps (14h00 à 16h00), celle de l'énergie et celle de la présence d'un congénère.
5. La formation de troupeau («*following*») est le composant le moins important. Son poids est mis à 1. Il est déclenché par un composant *Perception* qui vérifie la condition de l'énergie et détecte la présence des congénères jusqu'à une distance de 10 u.l. La désactivation du composant *Desire* s'effectue lorsque la condition de l'énergie n'est plus vérifiée.
6. Enfin, la tâche par défaut est quant à elle modélisée par un composant *Task* avec la seule action «RandomMove», qui est l'action de déplacement aléatoire. Le composant est configuré à cet effet pour reproduire le comportement décrit dans le tableau 5.20.

Une capture d'écran d'une des simulations ayant servi à générer les données de référence est présentée à la figure 5.26.

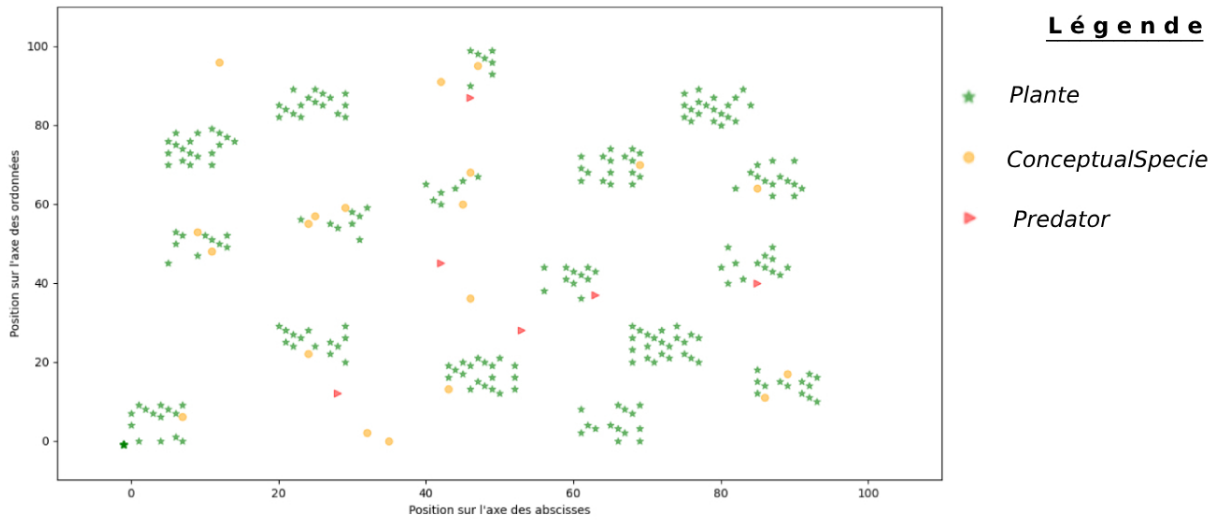


FIGURE 5.26 – Capture d'écran de la simulation du modèle de validation

Tout au long de cette validation, notre objectif sera de minimiser l'écart entre les données de référence et les résultats produits par les modèles générés au cours du processus. Pour cela, différents scénarios ont été proposés pour évaluer les performances de ANIMETA.

Divers scénarios ont été conçus pour exploiter les données générées précédemment, dans le cadre de la validation de notre approche. Chaque scénario a été méticuleusement élaboré afin de répondre à des questions spécifiques.

5.4.2.2 Scénario 1 : Paramétrage d'une action

5.4.2.2.1 Présentation du problème

Ce scénario vise à valider la capacité de ANIMETA à trouver la configuration optimale pour les actions du modèle. Concrètement, ce scénario pourrait correspondre à un cas d'utilisation où le biologiste connaît l'action que réalise l'animal étudié, mais pas les paramètres qui le caractérisent. Supposons donc que c'est le cas pour notre animal théorique, pour lequel nous ne connaissons pas le paramétrage de l'action de la tâche par défaut. La tâche par défaut devra donc être modélisée par un composant *PartialElement*, donc par un composant *PartialTask*. La figure 5.27 présente la représentation partielle de la tâche par défaut de l'agent.

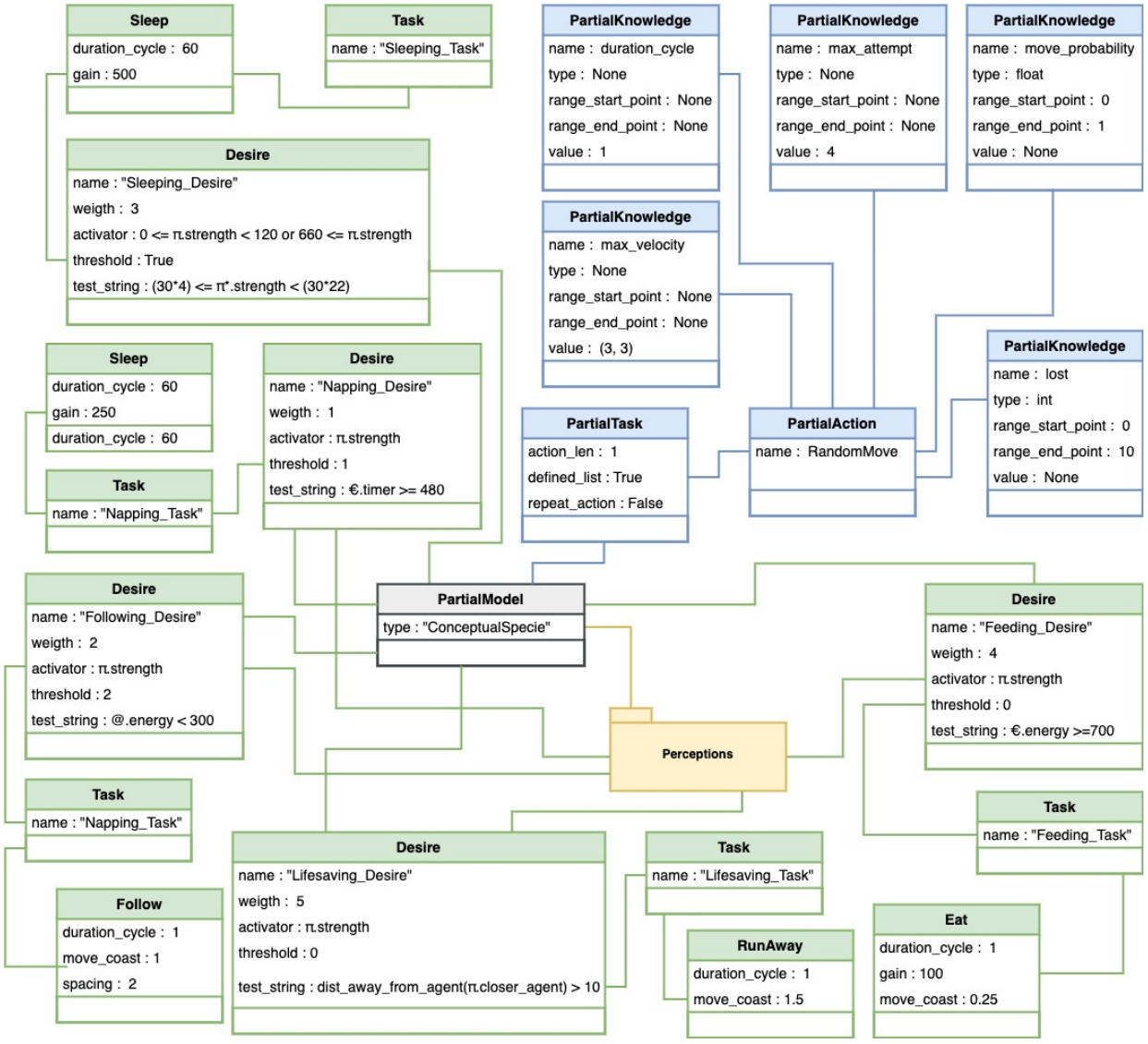


FIGURE 5.27 – Diagramme d'objet du modèle partiel (scénario 1)

L'attribut *defined_list* du composant *PartialTask* de la tâche par défaut est configuré avec la valeur « True » pour signifier au processus d'optimisation que l'espace de recherche se limite exclusivement à la liste des actions définies. Cette liste ici est formée par un composant *PartialAction* défini pour représenter l'action élémentaire *RandomMove* (name = RandomMove). Elle est complétée ensuite par une collection de composants *PartialKnowledge* qui représentent les connaissances à disposition. Sur les 5 paramètres que prend l'action élémentaire *RandomMove*, nous supposerons que 3 sont connus (duration_cycle, max_velocity, max_attempt) et précisés par les composants *PartialKnowledge*. Tous ces détails sont intégrés via *ANIMETA-HMI* comme le montre la figure 5.28. On y voit le bloc de la tâche par défaut formé par des composants de type *PartialElement*.

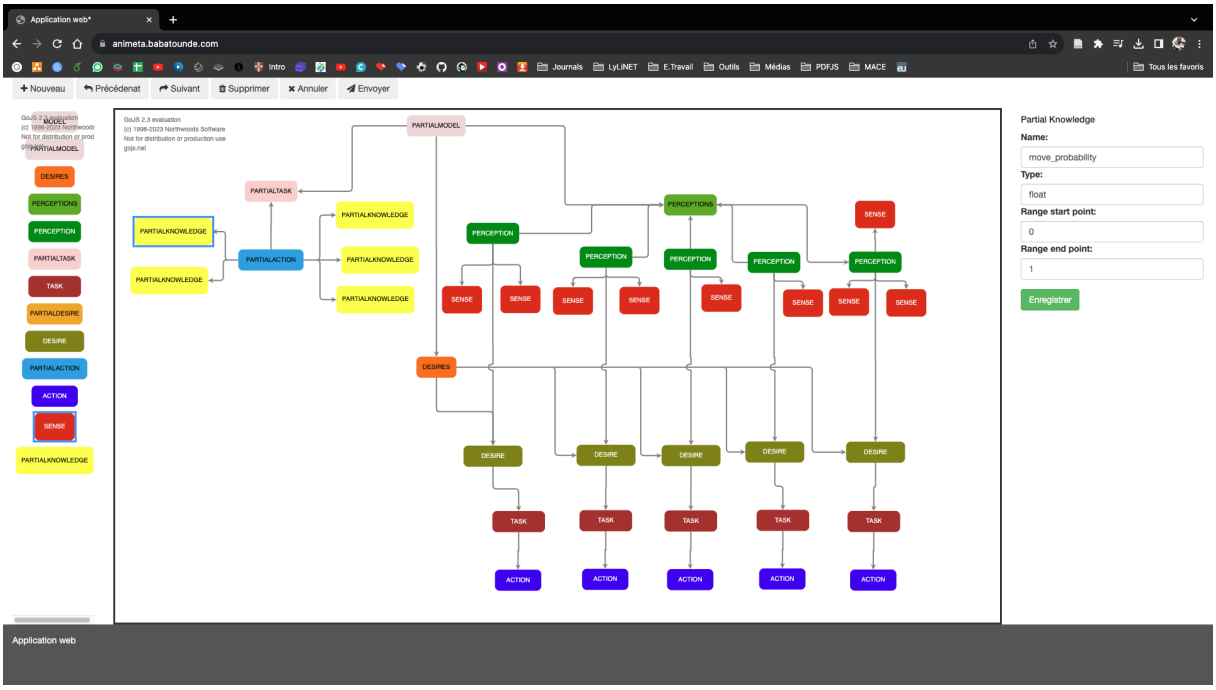


FIGURE 5.28 – Capture d'écran de ANIMETA-HMI lors de la conception du modèle partiel

5.4.2.2.2 Condition d'exécutions et résultats obtenus

Le modèle partiel de ce scénario présente deux inconnus (lost, move_probability) qui sont des paramètres de l'action *RandomMove*. Le tableau 5.21 présente les conditions de l'optimisation, notamment le paramétrage de chaque algorithme et un résumé des résultats obtenus pour chacun d'eux. Les solutions présentées dans ce tableau (de même que ceux des autres scénarios à venir) sont les meilleures que nous avons obtenues sur une série de 10 optimisations. Le choix pour ce paramétrage s'explique par l'espace de recherche qui n'est pas grand. Il serait donc inutile d'avoir plusieurs propositions de solution par itération. En ce qui concerne la fonction d'évaluation utilisée, il s'agit ici de la racine carrée de l'erreur quadratique moyenne. Elle mesure l'écart entre le couple quantité d'énergie - position prédit et celui des données de référence.

TABLE 5.21 – Conditions et résultats de l'optimisation au scénario 1

Algorithme	Paramétrage	Solution attendue	Solution Obtenue	Itérations	Évaluations	Durée
Génétique *(Détails sur les paramètres au tableau 4.2)*	nb. individus (n) : 10 nb. croisements (n_c) : 2 nb. nouveaux (n^+) : 2 Proba. mutation (p_m) : 0.20 nb. itération (n_{it}) : 100 Score limite (f_{opt}) : 20	lost : 1 move_probability : 0.5	lost : 1 move_probability : 0.6	Exécutées : 12 Solution : 12	Best : 18.01 Wrose : 33.68	00 :31 :17
Harmonique *(Détails sur les paramètres au tableau 4.6)*	nb. harmoniques (n) : 10 Proba. improvisation (sp_s) : 0.8 Proba. d'ajustement (p_{aj}) : 0.8 Taux d'ajustement (t_{aj}) : 0.2 Proba. technique (p_{choix}) : 0.5 nb. itération (n_{it}) : 100 Score limite (f_{opt}) : 20	lost : 1 move_probability : 0.5	lost : 7 move_probability : 0.08	Exécutées : 8 Solution : 8	Best : 19.88 Wrose : 53.96	00 :30 :05
Immunitaire *(Détails sur les paramètres au tableau 4.5)*	nb. anticorps (n) : 10 nb. nouveaux (n^+) : 2 nb. tentative ($n_{attempt}$) : 4 nb. itération (n_{it}) : 100 Score limite (f_{opt}) : 20	lost : 1 move_probability : 0.5	lost : 3 move_probability : 0.17	Exécutées : 3 Solution : 3	Best : 19.97 Wrose : 30.38	00 :37 :23
Fourmis *(Détails sur les paramètres au tableau 4.4)*	nb. solutions (n) : 10 nb. fourmis (n_f) : 10 proba. choix (p_{choix}) : 0.8 Score limite (f_{opt}) : 20	lost : 1 move_probability : 0.5	lost : 0 move_probability : 0.01	Exécutées : 100 Solution : 6	Best : 54.06 Wrose : 95.13	01 :55 :30

La figure 5.29 compare sur un même graphique, les données de références et les données produites par le modèle généré. Nous remarquons effectivement que les courbes sont très similaires, même s'il n'existe pas vraiment une superposition totale au niveau des trajectoires décrites. Cette divergence pourrait s'expliquer par les déplacements qui peuvent être très aléatoires, surtout dans le comportement de fuite. La consommation d'énergie étant liée à ces différents mouvements, ce fait pourrait également expliquer l'écart au niveau de la courbe d'énergie. Néanmoins, en se référant à la figure 5.30 qui montre les proportions d'actions réalisées, nous remarquons que la dynamique est maintenue.

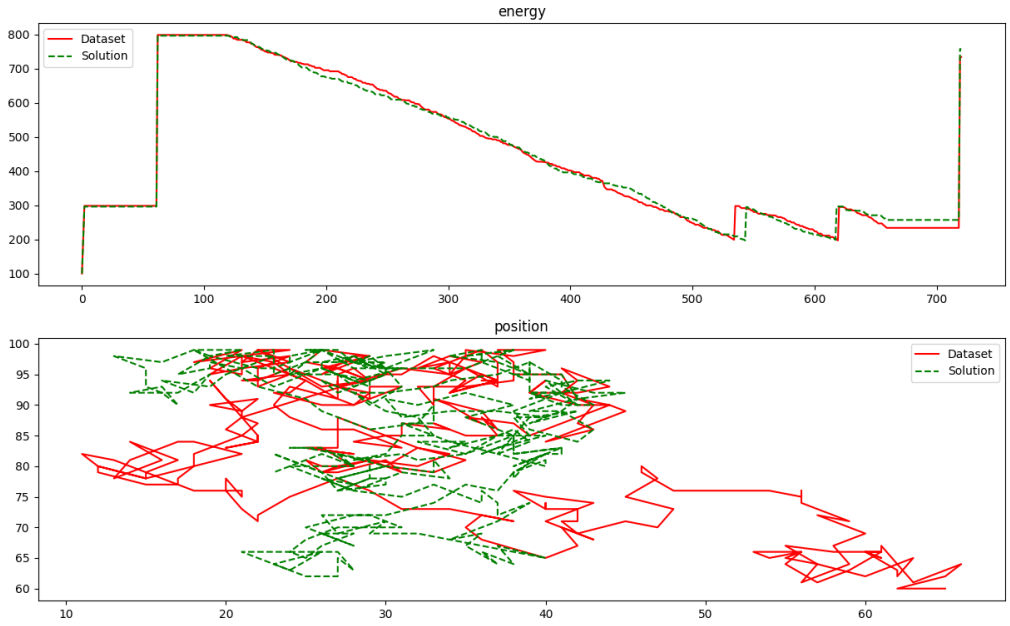


FIGURE 5.29 – Variation des niveaux d'énergie et des positions lors de la simulation des deux modèles

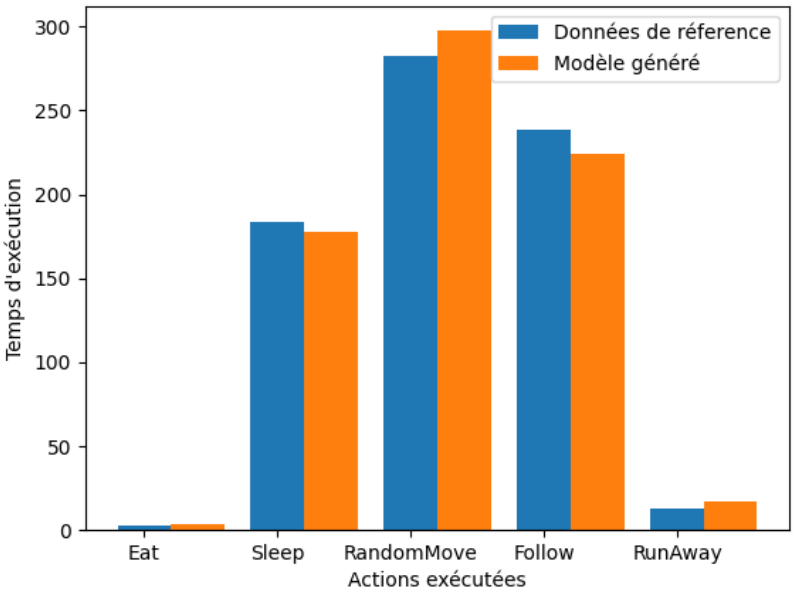


FIGURE 5.30 – Comparaison entre les actions réalisées par le modèle de référence et le modèle généré

Ces résultats ont surtout mis en lumière une des problématiques de notre recherche, celle de l'équifinalité. Prenons l'exemple des résultats obtenus avec l'algorithme de la recherche harmonique. Nous remarquons bien que la solution trouvée présente une bonne évaluation face au problème, mais les paramètres trouvés sont relativement éloignés de ceux attendus. Dans de telles situations, seul le biologiste avec son expertise peut confirmer ou infirmer les résultats obtenus.

À partir des résultats que présente l'algorithme de colonie de fourmis dans ce scénario, combinés à ceux obtenus à la section 5.4.1.3, nous faisons un constat capital. Celui de l'inefficacité de cet algorithme pour la résolution de ce type problème. Cette inefficacité peut s'expliquer par le fait que le graphe parcouru par les fourmis artificielles dépend des solutions générées aléatoirement au début de l'optimisation. L'espace de recherche est donc limité à celui-ci. Concrètement, cela voudra donc dire que si toutes les portions qui forment la solution optimale ne sont pas présentes dès le début, il n'est pas possible de la trouver à l'issue de l'optimisation.

5.4.2.3 Scénario 2 : Action et paramétrage manquants

5.4.2.3.1 Présentation du problème

Dans la continuité du scénario 1 où notre objectif est de valider la capacité de notre approche à trouver le paramétrage optimal pour une action élémentaire, nous passons ici à une étape supérieure qui va consister à trouver à la fois l'action manquante et son paramétrage. Nous montons ainsi en complexité puisque l'espace de recherche est beaucoup plus large et ne se limite pas qu'à un espace numérique. Nous avons donc un problème d'optimisation mixte dont une partie porte sur un problème combinatoire (trouver une action dans l'ensemble des actions) et l'autre partie porte sur un problème continu (trouver le paramétrage de l'action). Dans la réalité, ce scénario pourrait s'apparenter à un cas où le biologiste connaît les mécanismes de déclenchement d'un comportement, mais pas la série d'actions que réalise l'animal dans ces conditions. Pour cela, nous allons supposer dans ce scénario que les mécanismes de déclenchement du comportement d'alimentation sont connus, mais pas les actions réalisées. Autrement dit, la configuration du composant *Desire* est connue, mais pas la tâche associée. Ce modèle partiel peut donc être représenté par le diagramme d'objet de la figure 5.31.

Remarquez sur la figure, que le composant *PartialDesire* concerné est associé à une tâche de type *PartialTask* qui peut apporte plusieurs informations supplémentaires comme le nombre maximal d'actions à avoir pour la tâche (la valeur 1 dans ce cas). Concrètement, en termes de problème d'optimisation, nous aurons donc 4 inconnues qui sont l'action (*Eat*) et les valeurs de ses 3 paramètres (duration_cycle, move_cost, gain).

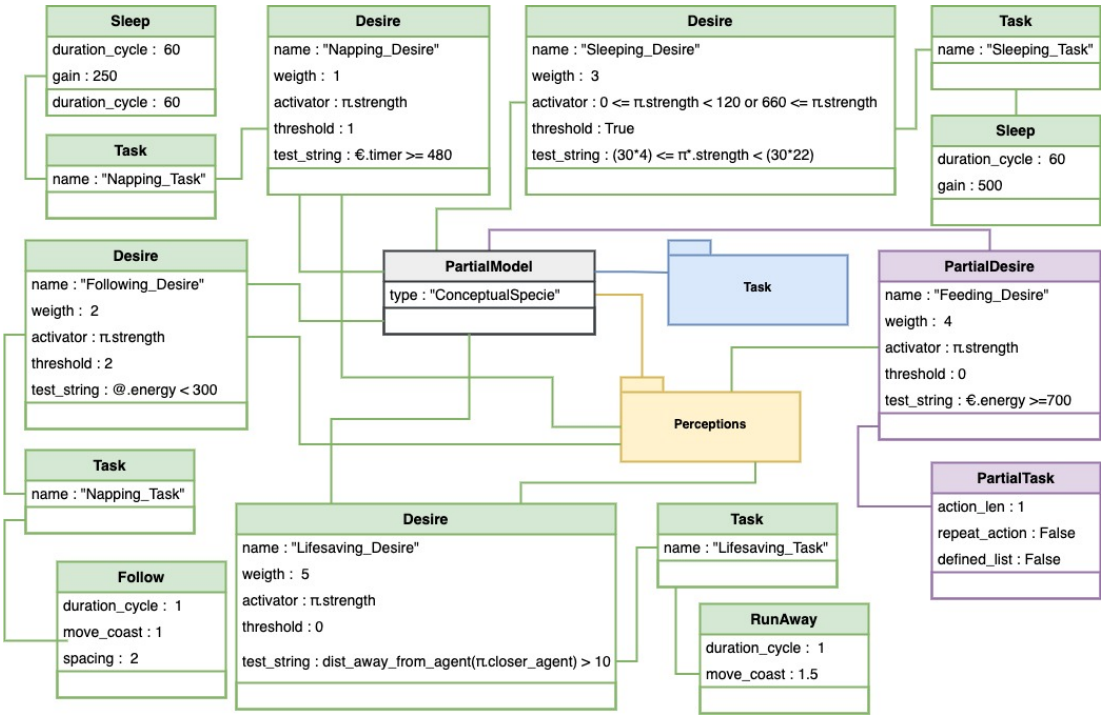


FIGURE 5.31 – Diagramme d'objet du modèle partiel (scénario 2)

5.4.2.3.2 Condition d'exécutions et résultats obtenus

Le tableau 5.22 présente les conditions de l'optimisation et les résultats obtenus à l'issue de la résolution du problème. Le même paramétrage que celui du scénario 1 a été maintenu. Nous ne jugeons pas nécessaire d'y apporter des changements puisque le nombre d'inconnus ne connaît pas une grande variation (4 dans ce scénario et 3 dans le précédent), même si les domaines de recherche sont plus étendus.

En analysant donc les résultats obtenus à l'issue de cette optimisation, nous remarquons que le temps de calcul a connu une augmentation significative par rapport au temps de calcul dans le scénario 1. Cette augmentation est tout à fait logique puisque l'espace de recherche est plus grand, et les valeurs recherchées sont de plus en plus précises. C'est le cas, par exemple, de la variable *move_coast* pour laquelle la valeur 0.25 doit être trouvée dans un espace de recherche qui concerne par défaut [14] tous les nombres réels de l'intervalle $[-100, 100]$.

En ce qui concerne le score des solutions obtenues, il n'y a pas de pertes significatives par rapport aux solutions du scénario 1. Dans certains cas, comme celui de l'algorithme génétique, nous avons obtenu des solutions avec de meilleurs scores. Ces résultats renforcent les acquis du scénario 1 sur la capacité de notre approche à générer des modèles optimaux qui conservent leur précision lorsque le domaine de recherche change et s'élargit. Toutefois, l'inefficacité de l'algorithme de colonie de fourmis demeure. Pour cela, nous continuerons le reste de la validation avec les trois algorithmes qui ont montré leur efficacité jusqu'ici, à savoir l'algorithme génétique, l'algorithme de la recherche harmonique et l'algorithme immunitaire par clonage.

14. c'est-à-dire lorsqu'il n'est pas précisé par un composant PartialKnowledge

TABLE 5.22 – Conditions et résultats de l'optimisation au scénario 2

Algorithme	Paramétrage	Solution attendue	Solution Obtenue	Itérations	Évaluations	Durée
Génétique *(Détails sur les paramètres au tableau 4.2)*	nb. individus (n) : 10 nb. croisements (n_c) : 2 nb. nouveaux (n^+) : 2 nb. itération (n_{it}) : 100 Proba. mutation (p_m) : 0.20 Score limite (f_{opt}) : 20	action : Eat duration_cycle : 1 gain : 100 move_coast : 0.25	action : Eat duration_cycle : 1 gain : 95 move_coast : 1	Exécutées : 46 Solution : 46	Best : 16.79 Wrose : 25.73	02 :36 :14
Harmonique *(Détails sur les paramètres au tableau 4.6)*	nb. harmoniques (n) : 10 Proba. improvisation (sp_s) : 0.8 Proba. d'ajustement (p_{aj}) : 0.8 Taux d'ajustement (t_{aj}) : 0.2 Proba. technique (p_{choix}) : 0.5 nb. itération (n_{it}) : 100 Score limite (f_{opt}) : 20	action : Eat duration_cycle : 1 gain : 100 move_coast : 0.25	action : Eat duration_cycle : 1 gain : 91 move_coast : 0.32	Exécutées : 34 Solution : 34	Best : 17.21 Wrose : 40.59	02 :54 :50
Immunitaire *(Détails sur les paramètres au tableau 4.5)*	nb. anticorps (n) : 10 nb. nouveaux (n^+) : 5 nb. tentative ($n_{attempt}$) : 4 nb. itération (n_{it}) : 100 Score limite (f_{opt}) : 20	action : Eat duration_cycle : 1 gain : 100 move_coast : 0.25	action : Eat duration_cycle : 1 gain : 82.58 move_coast : 0	Exécutées : 5 Solution : 5	Best : 19.30 Wrose : 41.22	02 :48 :28
Fourmis *(Détails sur les paramètres au tableau 4.4)*	nb. solutions (n) : 10 nb. fourmis (n_f) : 10 proba. choix (p_{choix}) : 0.8 Score limite (f_{opt}) : 20	action : Eat duration_cycle : 1 gain : 100 move_coast : 0.25	action : Sleep duration_cycle : 1 gain : 89	Exécutées : 100 Solution : 9	Best : 62.01 Wrose : 218.27	04 :16 :39

5.4.2.4 Scénario 3 : Paramètres de configuration et tâche d'un composant *Desire* manquants

5.4.2.4.1 Présentation du problème

Ce scénario est complémentaire au scénario 2. En effet, même si les mécanismes de déclenchement sont connus par le biologiste, il peut subsister des cas où certains paramètres, tels que la priorité des tâches, le poids ou les conditions d'interruptions, ne sont pas connus. Pour cela, nous reprenons le scénario 2 en y ajoutant cette fois-ci, comme données manquantes, le poids, la priorité et le facteur d'interruption. Ce qui change ici par rapport à la figure 5.31 est le composant *Desire*, qui est remplacé par celui de la figure 5.32. Puisque par principe, nous savons que le poids ne peut être qu'un entier naturel et qu'il aura une valeur maximale de 5, nous pouvons intégrer cette information par le biais d'un composant *PartialKnowledge*. Il en est de même pour la variable *interruption_factor*, pour laquelle on sait s'il s'agit d'un nombre réel compris entre 0 et 1. Enfin, le seuil (*threshold*) a la valeur *None* pour signifier qu'aucune information à son propos n'est connue. On se retrouve donc au total avec 7 inconnus qui sont :

1. Le poids du composant *Desire* représenté par un composant *PartialKnowledge* qui précise les informations connues ;
2. La probabilité d'interruption de la tâche représentée elle aussi par un composant *PartialKnowledge* ;
3. Le Seuil de déclenchement ;
4. L'action (*Eat*) de la tâche associée au composant *Desire* ;
5. Le paramètre duration_cycle de l'action de l'action *Eat* ;
6. Le paramètre gain de l'action de l'action *Eat* ;
7. Le paramètre move_coast de l'action de l'action *Eat* ;

Puisqu'ici l'espace de recherche s'élargit davantage, il faut aussi apporter quelques modifications au paramétrage des algorithmes pour une meilleure efficacité. C'est particulièrement le cas pour l'algorithme de la recherche harmonique et l'algorithme immunitaire, qui sont des algorithmes dont l'efficacité dépend fortement de la taille des populations et/ou du nombre de nouvelles solutions produites à chaque itération. Ce nouveau paramétrage est choisi de façon empirique compte tenu des multiples tentatives réalisées. Il est donc tout à fait discutable et peut être réglé avec diverses méthodes proposées par (Huang et al. 2020) et (Joshi et al. 2020).

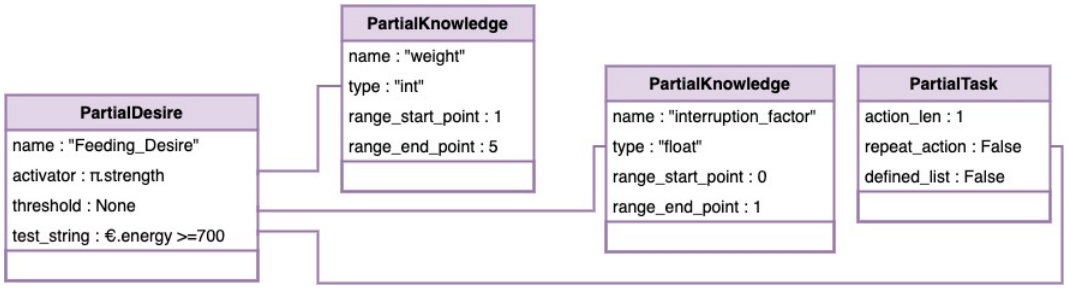


Figure 5.32 – Diagramme d'objet du composant *PartialDesire* du scénario 3

5.4.2.4.2 Condition d'exécutions et résultats obtenus

Le récapitulatif de l'optimisation est présenté dans le tableau 5.23. L'évaluation des modèles générés par les différents algorithmes montre des scores relativement constants par rapport aux scénarios précédents. Une fois encore, les résultats de ce scénario confortent notre validation quant à la capacité de la suite ANIMETA à générer des modèles qui conservent leur précision pour des espaces de recherche de plus en plus larges du point de vue du problème d'optimisation. Sur la figure 5.33, où nous montrons sur un même graphique les résultats de simulation du modèle généré et ceux du modèle de référence, nous constatons effectivement qu'il y a peu d'écarts entre les points. Néanmoins, une augmentation drastique des temps d'optimisation est remarquée sur l'ensemble des algorithmes. Ceci s'explique par les modifications apportées aux paramétrages des algorithmes, particulièrement la taille de la population. En effet, chaque solution de l'optimisation étant simulée puis évaluée, le temps total pour proposer un modèle est proportionnel au nombre de solutions générées durant l'entièreté du processus. Pour donner une idée de la durée de l'optimisation, il est important de savoir que l'évaluation d'une seule solution dure en moyenne 2 minutes et 40 secondes.

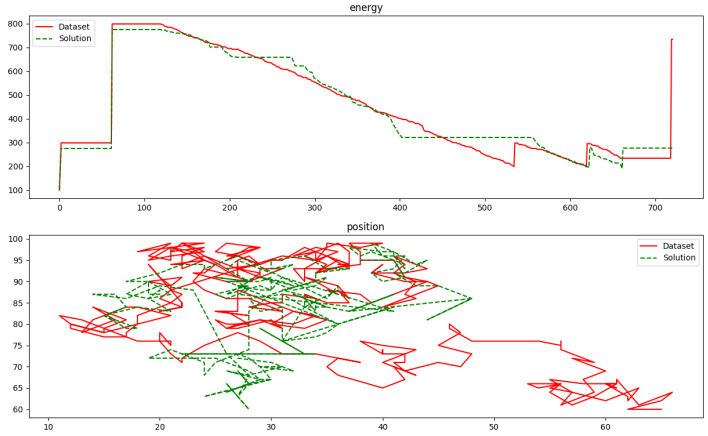


FIGURE 5.33 – Comparaison des données du modèle généré au scénario 3

Une fois encore, ces résultats montrent l'importance d'inclure le biologiste dans le processus de génération de modèle et de lui fournir des outils d'analyse adaptés. Il en va de l'efficacité de notre approche. En effet, dans le modèle de référence, la valeur du seuil est 0. Cela voudrait donc dire que, quelle que soit la valeur retournée par le composant *Perception*, le composant *Desire* peut être déclenché. Or, toutes les valeurs trouvées par les différents algorithmes sont toutes des valeurs réelles [15]. À la vue de ces résultats et en partant du principe que le biologiste peut faire cette analyse et déduire que le seuil ne peut être que 0, l'espace de recherche se restreint et nous gagnons donc en temps d'optimisation. Cette hypothèse a été validée en introduisant cette nouvelle information pour une nouvelle optimisation avec le même paramétrage. Le temps d'optimisation est passé de 09 heures 04 minutes et 30 secondes à 08 heures 16 minutes et 24 secondes.

15. Valeurs réelles cherchées dans l'intervalle [-100, 100]

TABLE 5.23 – Conditions et résultats de l'optimisation au scénario 3

Algorithme	Paramétrage	Solution attendue	Solution Obtenue	Itérations	Évaluations	Durée
Génétique *(Détails sur les paramètres au tableau 4.2)*	nb. individus (n) : 100 nb. croisements (n_c) : 10 nb. nouveaux (n^+) : 5 nb. itération (n_{it}) : 100 Proba. mutation (p_m) : 0.20 Score limite (f_{opt}) : 20	weight : 4 threshold : 1 interruption_factor : 0 action : Eat duration_cycle : 1 gain : 100 move_coast : 0.25	weight : 4 threshold : 0.7 interruption_factor : 0 action : Eat duration_cycle : 1 gain : 88 move_coast : 0.19	Exécutées : 44 Solution : 44	Best : 17.99 Wrose : 60.61	09 :04 :30
Harmonique *(Détails sur les paramètres au tableau 4.6)*	nb. harmoniques (n) : 100 Proba. improvisation (sp_s) : 0.8 Proba. d'ajustement (p_{aj}) : 0.8 Taux d'ajustement (t_{aj}) : 0.2 Proba. technique (p_{choix}) : 0.5 nb. itération (n_{it}) : 100 Score limite (f_{opt}) : 20	weight : 4 threshold : 1 interruption_factor : 0 action : Eat duration_cycle : 1 gain : 100 move_coast : 0.25	weight : 4 threshold : 0.92 interruption_factor : 0 action : Eat duration_cycle : 1 gain : 93 move_coast : 0.26	Exécutées : 89 Solution : 89	Best : 18.01 Wrose : 60.75	18 :21 :32
Immunitaire *(Détails sur les paramètres au tableau 4.5)*	nb. anticorps (n) : 100 nb. nouveaux (n^+) : 10 nb. tentative ($n_{attempt}$) : 4 nb. itération (n_{it}) : 100 Score limite (f_{opt}) : 20	weight : 4 threshold : 1 interruption_factor : 0 action : Eat duration_cycle : 1 gain : 100 move_coast : 0.25	weight : 5 threshold : 0.92 interruption_factor : 0.15 action : Eat duration_cycle : 1 gain : 88 move_coast : 0.18	Exécutées : 36 Solution : 36	Best : 19.62 Wrose : 72.03	16 :39 :27

5.4.2.5 Scénario 4 : Moitié des composants sont manquants

5.4.2.5.1 Présentation du problème

La proposition de ce scénario est orientée beaucoup plus dans la logique de l'évaluation des performances de ANIMETA. En effet, dans les scénarios 1, 2 et 3, où nous sommes montés progressivement en complexité, nous avons remarqué que les performances, principalement le temps de calcul, se dégradent proportionnellement. Ainsi, il serait intéressant de tester aussi progressivement les limites de ANIMETA. Pour cela, avant de nous intéresser dans la section 5.4.2.6 à un scénario où tous les composants sont manquants, ce scénario se place comme une évaluation à mi-chemin, puisque sur les 6 unités de comportements du modèle à compléter, il en comptera 3 qui sont manquantes. Nous aurons donc à trouver dans ce scénario les valeurs optimales pour 3 composants *Desire* (configuration, les actions de la tâche associée de même que leur paramètre). La représentation d'un tel modèle partiel est présentée à la figure 5.34.

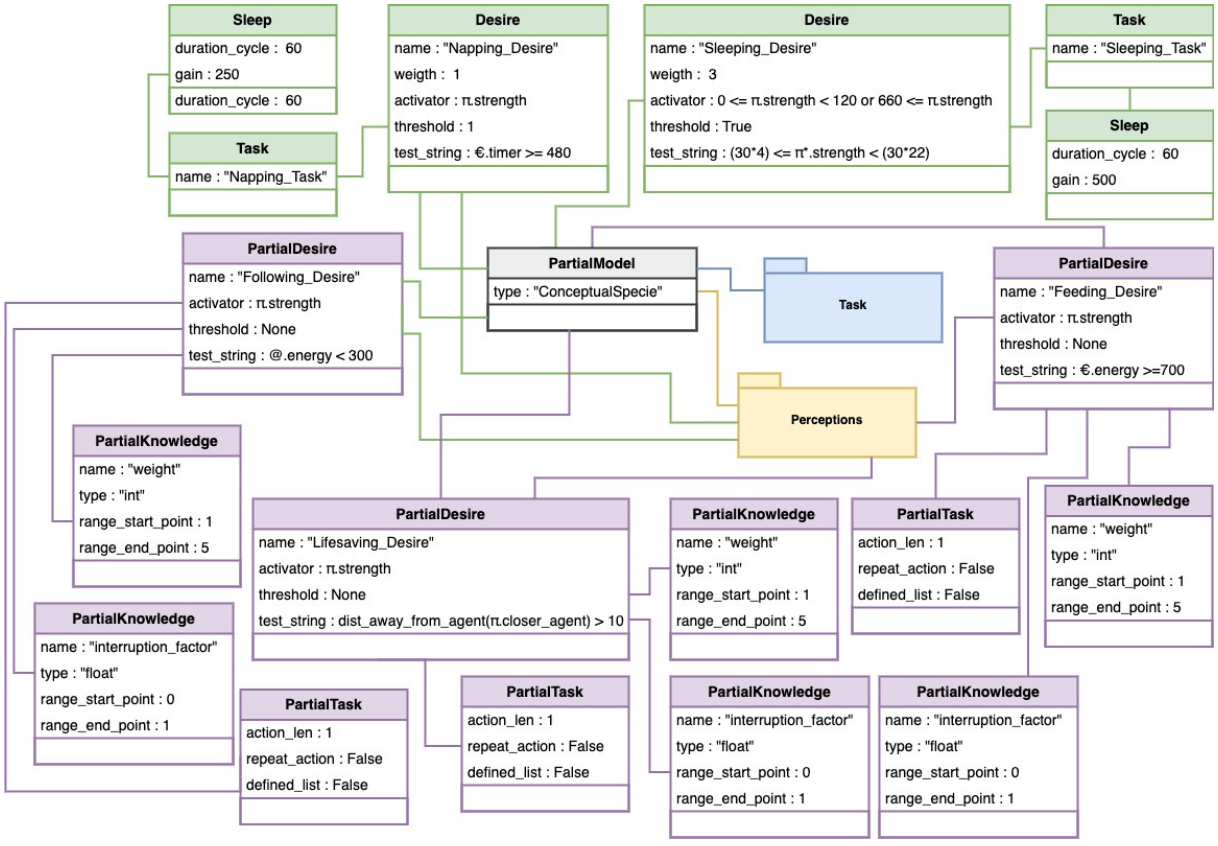


FIGURE 5.34 – Diagramme d'objet du modèle partiel (scénario 4)

5.4.2.5.2 Condition d'exécutions et résultats obtenus

Le tableau 5.24 présente les résultats obtenus pour une première optimisation du scénario 4. Comme on pouvait s'y attendre, les performances se sont considérablement dégradées. Les scores sont moins bons que ceux obtenus pour les précédents scénarios et, pour certains algorithmes, la solution optimale n'est pas trouvée et la durée d'optimisation a grandement évolué (cas de l'algorithme de recherche harmonique). L'inefficacité du processus d'optimisation peut d'ailleurs être constatée sur la courbe de la figure 5.35 montrant l'évolution du score de la meilleure au fil des itérations pour l'algorithme de la recherche harmonique. Bien que l'optimisation ait été exécutée sur les 100 itérations, nous remarquons que les solutions relativement acceptables sont apparues à la 30$^{\text{ième}}$ itération et la solution finale a été trouvée à l'itération 91 avec une très faible variation. On aura donc perdu 60 itérations pour une différence peu significative.

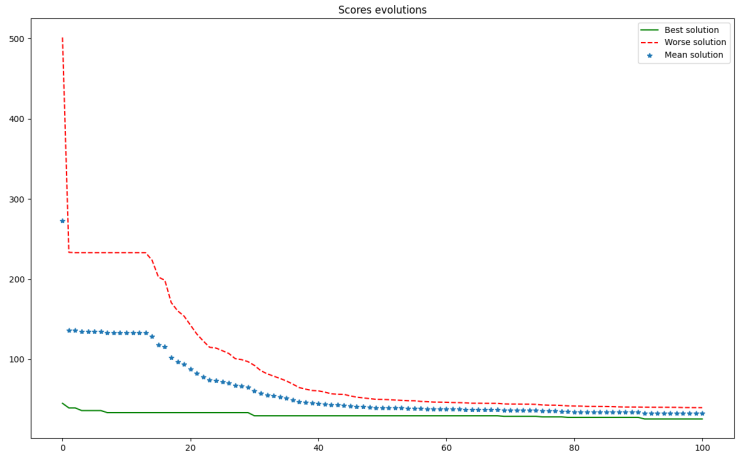


FIGURE 5.35 – Évolution des scores au fil des itérations pour l'algorithme Immunitaire

Néanmoins, ce qui est intéressant dans ces premiers résultats, c'est que nous remarquons que pour toutes les propositions de solution, certaines séquences sont récurrentes. Tel est le cas pour FeedingDesire FollowingDesire avec les actions *Eat* et *Follow*, ainsi que les paramètres de configuration des composants *Desire*. Ce constat pourrait être une probable piste de recherche pour obtenir le modèle optimal recherché. ANIMETA étant conçu sur la base d'un processus incrémental, un nouvel incrément pourrait être envisagé en se servant des séquences récurrentes pour mettre à jour les composants du modèle partiel d'origine. D'où une fois encore, l'intérêt d'inclure l'expert dans le processus, puisqu'une telle analyse ne peut qu'être idéalement faite par le biologiste.

TABLE 5.24 – Conditions et résultats de l'optimisation à l'a première optimisation au scénario 4

Algorithme	Paramétrage	Solution attendue	Solution Obtenue	Itérations	Évaluations	Durée
Génétique *(Détails sur les paramètres au tableau 4.2)*	nb. individus (n) : 100 nb. croisements (n_c) : 10 nb. nouveaux (n^+) : 10 nb. itération (n_{it}) : 100 Proba. mutation (p_m) : 0.20 Score limite (f_{opt}) : 20	**FeedingDesire** weight : 4 threshold : 0 interruption_factor : 0.5 Action :Eat duration_cycle : 1 gain : 100 move_coast : 0.25 **FollowingDesire** weight : 2 threshold : 2 interruption_factor : 0.8 Action :Follow duration_cycle : 1 spacing : 2 move_coast : 1 **LifesavingDesire** weight : 5 threshold : 0 interruption_factor : 0.1 Action :RunAway duration_cycle : 1 move_coast : 1.5	**FeedingDesire** weight : 3 threshold : 3 interruption_factor : 0 Action :Eat duration_cycle : 1 gain : 88 move_coast : 0.76 **FollowingDesire** weight : 2 threshold : 4 interruption_factor : 0.96 Action :Eat duration_cycle : 3 gain : 75 move_coast : 5.6 **LifesavingDesire** weight : 5 threshold : 3 interruption_factor : 0 Action :RunAway duration_cycle : 9 move_coast : 1.11	Exécutées : 47 Solution : 47	Best : 18.99 Wrose : 33.58	09 :08 :57

Harmonique *(Détails sur les paramètres au tableau 4.6)*	nb. harmoniques (n) : 100 Proba. improvisation (sp_s) : 0.8 Proba. d'ajustement (p_{aj}) : 0.8 Taux d'ajustement (t_{aj}) : 0.2 Proba. technique (p_{choix}) : 0.5 nb. itération (n_{it}) : 100 Score limite (f_{opt}) : 20	**FeedingDesire** weight : 4 threshold : 0 interruption_factor : 0.5 Action :Eat duration_cycle : 1 gain : 100 move_coast : 0.25 **FollowingDesire** weight : 2 threshold : 2 interruption_factor : 0.8 Action :Follow duration_cycle : 1 spacing : 2 move_coast : 1 **LifesavingDesire** weight : 5 threshold : 0 interruption_factor : 0.1 Action :RunAway duration_cycle : 1 move_coast : 1.5	**FeedingDesire** weight : 4 threshold : 4 interruption_factor : 0.4 Action :Eat duration_cycle : 2 gain : 84 move_coast : 0.05 **FollowingDesire** weight : 2 threshold : 3 interruption_factor : 0.9 Action :Follow duration_cycle : 1 spacing : 3 move_coast : 3.6 **LifesavingDesire** weight : 5 threshold : 0 interruption_factor : 0.1 Action :Eat duration_cycle : 4 gain : 78 move_coast : 0.5	Exécutées : 100 Solution : 91	Best : 25.57 Wrose : 39.1	17 :17 :20

Immunitaire *(Détails sur les paramètres au tableau 4.5)*	nb. anticorps (n) : 100 nb. nouveaux (n^+) : 10 nb. tentative ($n_{attempt}$) : 4 nb. itération (n_{it}) : 100 Score limite (f_{opt}) : 20	**FeedingDesire** weight : 4 threshold : 0 interruption_factor : 0.5 Action :Eat duration_cycle : 1 gain : 100 move_coast : 0.25 **FollowingDesire** weight : 2 threshold : 2 interruption_factor : 0.8 Action :Follow duration_cycle : 1 spacing : 2 move_coast : 1 **LifesavingDesire** weight : 5 threshold : 0 interruption_factor : 0.1 Action :RunAway duration_cycle : 1 move_coast : 1.5	**FeedingDesire** weight : 5 threshold : 4.7 interruption_factor : 0.6 Action :Eat duration_cycle : 0 gain : 96 move_coast : 0.68 **FollowingDesire** weight : 0 threshold : 3 interruption_factor : 0.17 Action :Sleep duration_cycle : 10 gain : 11.87 **LifesavingDesire** weight : 5 threshold : 4.3 interruption_factor : 0.9 Action :Rest duration_cycle : 1	Exécutées : 02 Solution : 02	Best : 16.41 Wrose : 46.69	07 :47 :57

L'exploitation donc des résultats du tableau 5.24, met en évidence la possibilité de nouvelles connaissances qui peuvent être formulées comme suit :

— composant *Desire* intitulé FeedingDesire :
  — le composant *Task* est constitué de l'action élémentaire *Eat*. Ses paramètres *duration_cycle, move_coast* et gain sont respectivement dans les intervalles [1 ; 10], [0 ; 1] et [80 ; 100] ;
— composant *Desire* intitulé FollowingDesire :
  — le paramètre threshold est un entier dans l'intervalle [1 ; 5]
— composant *Desire* intitulé LifesavingDesire :
  — l'action du composant *Task* est l'action élémentaire *RunAway*

En mettant à jour le modèle partiel du début avec ces connaissances puis en procédant à une nouvelle optimisation, nous obtenons un meilleur résultat au bout d'une durée drastiquement baissée comme le montre le tableau 5.25. Cette performance est palpable sur la figure 5.36 qui montre la similarité des données du modèle généré avec les données de références.

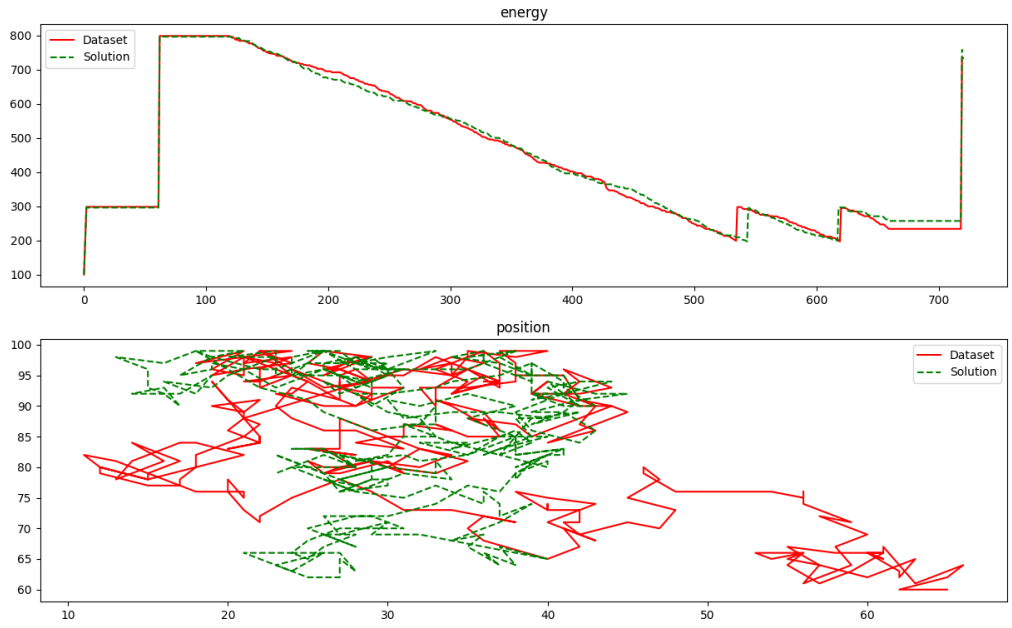


FIGURE 5.36 – Comparaison des données du modèle généré au scénario 4

Avec ces premiers résultats, il apparaît donc que procéder de façon incrémentale à la génération des modèles est l'option idéale lorsqu'on a de plus en plus de données manquantes dans le modèle partiel de début.

TABLE 5.25 – Conditions et résultats de l'optimisation la seconde optimisation au scénario 4

Algorithme	Paramétrage	Solution attendue	Solution Obtenue	Itérations	Évaluations	Durée
Génétique *(Détails sur les paramètres au tableau 4.2)*	nb. individus (n) : 100 nb. croisements (n_c) : 10 nb. nouveaux (n^+) : 10 nb. itération (n_{it}) : 100 Proba. mutation (p_m) : 0.20 Score limite (f_{opt}) : 20	**FeedingDesire** weight : 4 threshold : 0 interruption_factor : 0.5 Action :Eat duration_cycle : 1 gain : 100 move_coast : 0.25 **FollowingDesire** weight : 2 threshold : 2 interruption_factor : 0.8 Action :Follow duration_cycle : 1 spacing : 2 move_coast : 1 **LifesavingDesire** weight : 5 threshold : 0 interruption_factor : 0.1 Action :RunAway duration_cycle : 1 move_coast : 1.5	**FeedingDesire** weight : 3 threshold : 3 interruption_factor : 0 Action :Eat duration_cycle : 1 gain : 93 move_coast : 0.2 **FollowingDesire** weight : 6 threshold : 5 interruption_factor : 0 Action :Get$_C$*lose* *duration_cycle* : 1 *max_attempt* : 2 **LifesavingDesire** weight : 5 threshold : 1 interruption_factor : 0 Action :RunAway duration_cycle : 1 move_coast : 0.5	Exécutées : 21 Solution : 21	Best : 18.73 Wrose : 51.76	04 :33 :56

5.4.2.6 Scénario 5 : Générer entièrement un modèle optimal

5.4.2.6.1 Présentation du problème

Les quatre premiers scénarios de validation ont principalement été axés sur la complétion de modèles partiels. Dans ce cinquième scénario, nous évaluons la capacité d'ANIMETA à générer intégralement un modèle à partir des données fournies. Cette situation pourrait correspondre au cas d'utilisation où le biologiste ne possède aucune connaissance précise sur les conditions de déclenchement de l'ensemble des unités de comportement de l'animal ni sur les actions réalisées et leurs paramètres associés. Le modèle partiel soumis à l'optimisation dans ANIMETA est illustré dans la figure 5.37. Ce scénario simule une situation réaliste dans laquelle les données initiales sont incomplètes, situation à laquelle les biologistes sont confrontées lorsqu'ils tentent de modéliser le comportement d'une espèce sans disposer d'informations exhaustives.

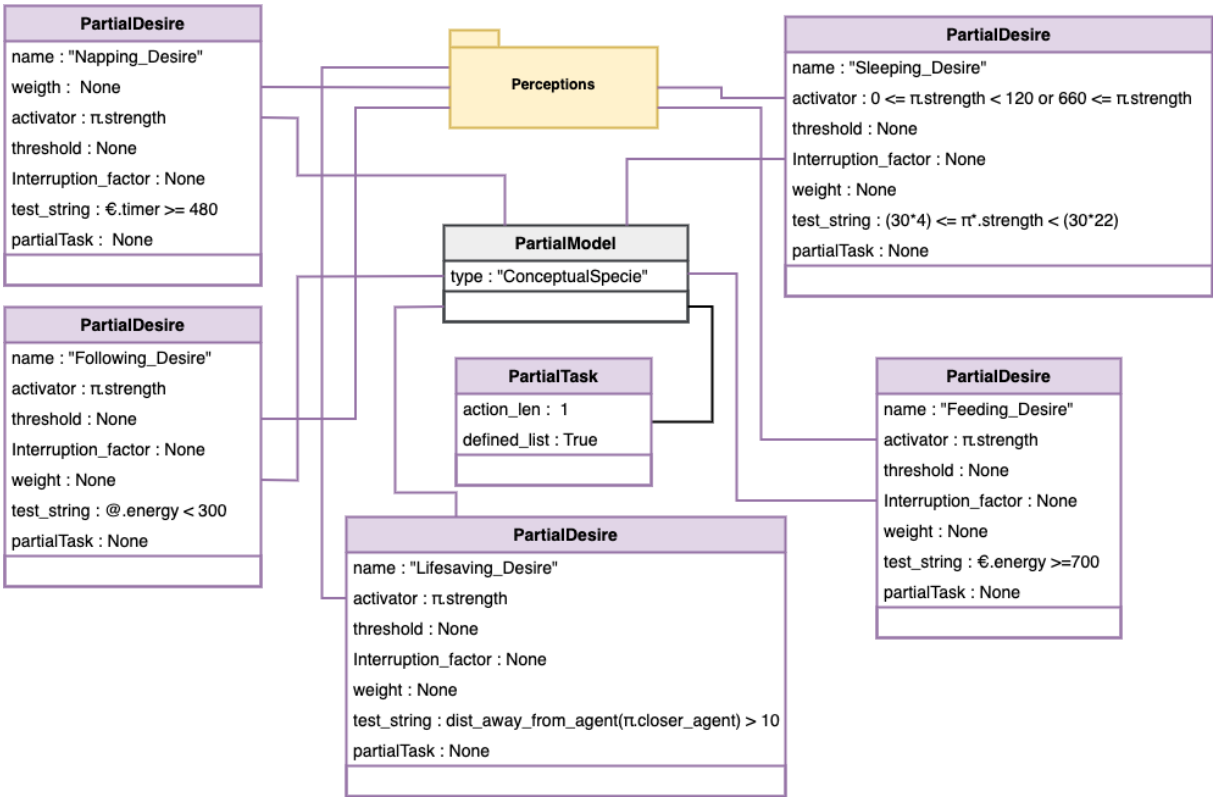


FIGURE 5.37 – Diagramme d'objet du modèle partiel (scénario 5 - version 1)

5.4.2.6.2 Condition d'exécutions et résultats obtenus

La complexité de ce problème requiert des ajustements dans la configuration de l'optimiseur. Les modifications apportées ont porté sur :

- Le nombre de solutions générées par itération dépend de chaque algorithme et est influencé par des paramètres propres à chaque méthode. Par exemple, dans le cas de l'algorithme génétique, le nombre de croisements, ou pour l'algorithme de la recherche harmonique, le nombre d'harmoniques, sont des paramètres qui ont un impact direct sur la quantité de solutions produites à chaque itération.
- Le score de la solution optimale est ajusté en prenant en considération les résultats du scénario 4, en particulier la durée de l'optimisation et la stabilité de l'amélioration des scores sur plusieurs itérations. Ainsi, nous avons décidé d'augmenter initialement le score de la solution optimale pour les premiers incréments, puis de le réduire progressivement. Cette modification vise, d'une part, à éviter l'exécution de toutes les itérations (pour économiser du temps) et, d'autre part, à obtenir rapidement des propositions de séquences récurrentes pour améliorer le modèle partiel d'origine, tout comme dans le scénario 4.
- La fonction d'évaluation sélectionnée est la « n-step prediction error » en raison de l'ampleur de l'espace de recherche, susceptible d'engendrer d'importantes variations dans l'évaluation des solutions. De plus, la problématique des déplacements aléatoires pourrait également surgir. Le choix de cette fonction d'évaluation est motivé par sa pertinence potentielle pour ce cas d'utilisation (voir la section 4.3.1).

En tenant compte de ces divers ajustements, et après quatre incréments tout en identifiant les séquences récurrentes de la même manière qu'au scénario 4, nous avons réussi à obtenir un modèle partiel plus précis. Celui-ci est illustré par le diagramme d'objet figurant à la 5.38.

Ainsi, à partir de ce modèle partiel, le cinquième incrément a conduit à l'obtention des résultats présentés dans le tableau 5.26.

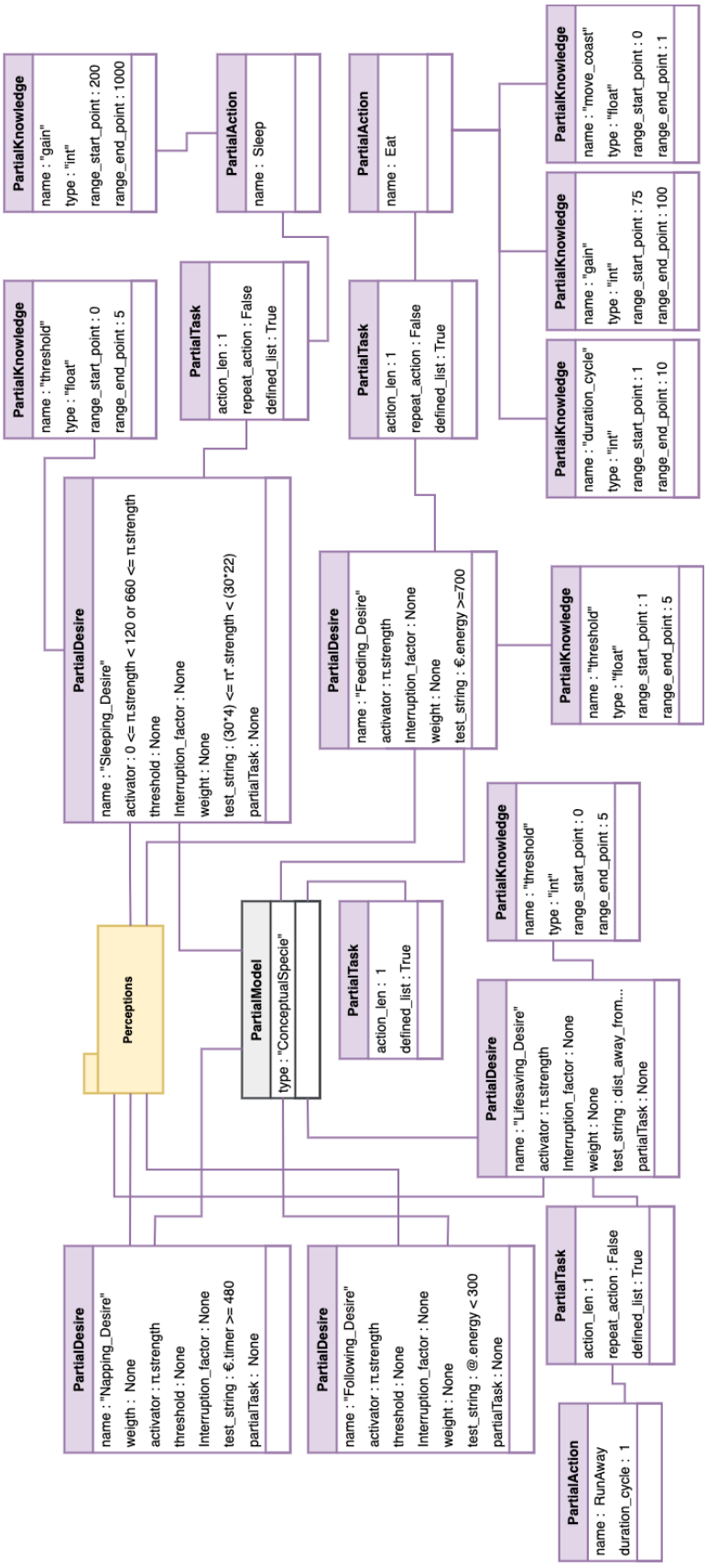


FIGURE 5.38 – Diagramme d'objet du modèle partiel (scénario 5 - version 2

TABLE 5.26 – Conditions et résultats de l'optimisation au scénario 5

Algorithme	Paramétrage	Solution attendue	Solution Obtenue	Itérations	Évaluations	Durée
Génétique *(Détails sur les paramètres au tableau 4.2)*	nb. individus (n) : 100 nb. croisements (n_c) : 20 nb. nouveaux (n^+) : 10 nb. itération (n_{it}) : 100 Proba. mutation (p_m) : 0.20 Score limite (f_{opt}) : 20	**FeedingDesire** weight : 4 threshold : 0 interruption_factor : 0.5 Action :Eat duration_cycle : 1 gain : 100 move_coast : 0.25 **FollowingDesire** weight : 2 threshold : 2 interruption_factor : 0.8 Action :Follow duration_cycle : 1 spacing : 2 move_coast : 1 **LifesavingDesire** weight : 5 threshold : 0 interruption_factor : 0.1 Action :RunAway duration_cycle : 1 move_coast : 1.5 **NappingDesire** weight : 1 threshold : 1 interruption_factor : 0.1 Action :Sleep duration_cycle : 60 gain : 250 **SleepingDesire** weight : 3 threshold : True (1) interruption_factor : 0.1 Action :Sleep duration_cycle : 60 gain : 500	**FeedingDesire** weight : 5 threshold : 3 interruption_factor : 0.5 Action :Eat duration_cycle : 1 gain : 96 move_coast : 0.11 **FollowingDesire** weight : 2 threshold : 5 interruption_factor : 0.2 Action :Sleep duration_cycle : 5 gain : 22 **LifesavingDesire** weight : 5 threshold : 5 interruption_factor : 0 Action :RunAway duration_cycle : 3 move_coast : 1 **NappingDesire** weight : 4 threshold : 4.02 interruption_factor : 0.27 Action :Sleep duration_cycle : 82 gain : 600 **SleepingDesire** weight : 3 threshold : 0.8 interruption_factor : 0.18 Action :Sleep duration_cycle : 60 gain : 574	Exécutées : 97 Solution : 97	Best : 19.81 Wrose : 33.82	18 :17 :29

Lors de la comparaison entre les données de simulation du modèle obtenu et les données de référence, comme nous l'illustrons à la figure 5.39, nous observons effectivement une similitude de tendance dans les comportements. Cette observation démontre la capacité de ANIMETA à partir d'un jeu de données de référence initial, puis, à travers plusieurs itérations, à générer un modèle dont les résultats de simulation convergent avec les comportements attendus. Cette concordance conforte la validité et l'efficacité de notre approche dans la modélisation du comportement animal en simulation.

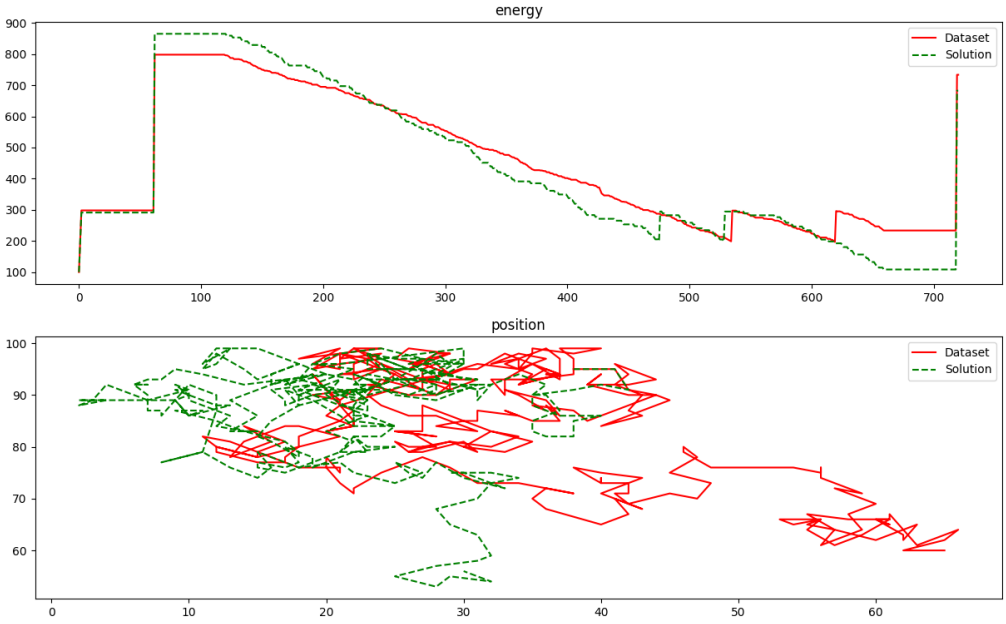


FIGURE 5.39 – Comparaison des données du modèle généré au scénario 5

Le modèle obtenu est entièrement interprétable et explicable, puisque nous disposons des éléments nécessaires pour expliquer le fonctionnement du modèle et les mécanismes de prise de décisions. Nous savons quelles actions sont réalisées, pourquoi elles le sont, et comment elles le sont.

Conclusion

Après avoir présenté ANIMETA dans les chapitres précédents, ce chapitre a été dédié à la vérification et à la validation des différents concepts intégrés dans notre approche. Cette confirmation a été établie à travers une série de tests mettant en œuvre divers types de modèles générés.

La première série de tests a été axée sur la vérification de l'implémentation de l'approche. Après l'avoir intégrée dans un environnement informatique soigneusement choisi pour répondre aux exigences de notre cadre de recherche, nous avons pu confirmer que l'implémentation a été réalisée conformément à notre conception initiale. Cette étape nous a garanti que le système ainsi disponible est fonctionnel et prêt à passer à la phase suivante, où nous évaluons la capacité de notre approche à produire les résultats escomptés, c'est-à-dire nous assurer que notre approche est en mesure de générer des modèles de comportement animal fiables.

La seconde série de tests a donc été spécifiquement dédiée à la validation de la performance de notre approche. Nous avons soumis ANIMETA à des modèles d'espèces réelles ainsi qu'à des modèles d'espèces conceptuelles, en explorant une variété de scénarios. Ces tests ont révélé deux aspects importants : la capacité de ANIMETA à générer un modèle de réponse de fuite du Sciaena umbra, d'une part, et d'autre part, à créer des modèles à partir d'informations incomplètes.

Néanmoins, certaines situations de contre-performance ont été observées au cours de nos tests.

— La durée de l'optimisation s'est révélée parfois très longue, ce qui peut constituer un point de friction dans certaines applications où le temps est un facteur critique. Cette observation souligne la nécessité d'explorer des avenues d'optimisation ou d'ajustement des paramètres pour améliorer l'efficacité du processus.

— nous avons identifié que certains algorithmes utilisés ne sont pas parfaitement adaptés au type de problème résolu par ANIMETA. Ainsi, nous avons établi un classement selon les performances des différents algorithmes au tableau 5.27.

L'identification de ces situations de contre-performance nous offre des pistes qui nous permettront d'affiner continuellement ANIMETA pour en faire un outil d'aide robuste et efficace dans la modélisation du comportement animal.

TABLE 5.27 – Classement des métaheuristiques testés selon les performances

Classement	Algorithme	Justificatif
N°1	Algorithme génétique	Il a résolu de manière efficace la totalité des problèmes qui lui ont été soumis, en parvenant systématiquement à trouver des solutions optimales dans des délais acceptables par rapport à l'ensemble des algorithmes disponibles. Cette efficacité découle de son processus rapide d'identification des solutions dites létales et d'une gestion contrôlée du nombre de solutions générées à chaque itération. De plus, la probabilité élevée d'obtenir, à chaque étape, des solutions différentes lui confère la capacité de s'extraire rapidement des optimaux locaux.
N°2	Algorithme de recherche harmonique	Il a répondu efficacement aux problèmes. Cependant, en le comparant à l'algorithme génétique, on constate un temps d'optimisation relativement élevé. Cette contre-performance peut être expliquée par la nature même de l'algorithme. D'une part, il a tendance à maintenir une population stable, ce qui augmente le risque de rester bloqué dans un optimum local. D'autre part, il est susceptible de générer, au cours d'une même itération, des solutions similaires, contribuant ainsi à stabiliser la population et accroissant le risque de blocage dans un optimum local. Néanmoins, cette limitation peut être surmontée en ajustant la probabilité d'intégrer de nouvelles solutions dans la population. Opter pour une valeur élevée de cette probabilité permet de dépasser ces contraintes. Il est toutefois important de souligner que le choix de cette valeur doit être raisonnable, car une probabilité trop élevée pourrait paradoxalement prolonger le temps d'optimisation en introduisant une diversité excessive et en rendant la convergence plus difficile à atteindre.
N°3	Algorithme immunitaire	Il s'est distingué par des temps d'optimisation particulièrement élevés, principalement à cause de la grande quantité de nouvelles solutions qu'il peut générer à chaque itération. En prenant l'exemple d'une population initiale de 100 solutions, chacune devant être clonée 5 fois, nous nous retrouvons avec un total de 500 solutions à évaluer à chaque étape du processus. De plus, si les solutions dans la population sont de meilleure qualité, le nombre de clones par solution augmente également. Ainsi, avec un nombre important de clones, la probabilité d'obtenir des solutions similaires augmente, entraînant une convergence plus lente du processus d'optimisation.
N°4	Algorithme de colonie des fourmis	Il s'est avéré être l'algorithme affichant les résultats les moins satisfaisants. Sa mise en œuvre a été rapidement interrompue, car dans aucun des cas il n'a réussi à trouver une solution optimale. Cette performance décevante est attribuable à l'espace de recherche, qui demeure à la fois restreint et fixe tout au long du processus d'optimisation. L'incapacité de l'algorithme à explorer un espace de recherche plus vaste et dynamique a entravé sa capacité à identifier des solutions optimales.

Conclusion générale

Bilan des Travaux

Nos travaux ont porté sur la proposition d'une nouvelle approche de modélisation du comportement des espèces animales, guidée par les métaheuristiques. Cette initiative découle du besoin de l'homme, depuis les temps préhistoriques jusqu'à nos jours, de comprendre le comportement des animaux pour des raisons écologiques, économiques, voire de survie.

Dans un écosystème de plus en plus fragile, où les interventions humaines sont limitées, la modélisation et la simulation des comportements des animaux revêtent un enjeu majeur. Ces outils permettent de mener des études concrètes sur les animaux en se libérant de nombreuses contraintes d'ordre géographique, temporel, logistique et bien d'autres.

Bien qu'il existe des travaux déjà entrepris dans cette direction, présentant des approches de modélisation et de simulation du comportement animal, leur mise en œuvre demeure relativement fastidieuse. Dans certains cas, notamment pour les approches d'apprentissage automatique de type «Deep learning», des problèmes d'interprétabilité et d'explicabilité se posent, limitant parfois leur applicabilité. D'autre part, ces méthodes peuvent parfois sacrifier la précision au profit de la complexité inhérente à leur structure.

Dans le cadre de nos travaux, nous avons élaboré ANIMETA, une nouvelle approche qui cherche à contourner ces contraintes. Notre méthodologie se veut plus accessible et efficace, visant à surmonter les difficultés rencontrées par les approches existantes. ANIMETA se place comme une approche qui apporte à cette thématique, des solutions aux problématiques d'implémentation fastidieuse, d'interprétabilité et d'explicabilité propres à certaines méthodes existantes.

L'approche ANIMETA repose sur l'hypothèse selon laquelle il serait possible d'utiliser des connaissances sur le comportement d'une espèce donnée pour modéliser le comportement d'une tout autre espèce. Concrètement, il s'agissait d'aborder la modélisation du comportement animal comme un problème d'optimisation en constituant un ensemble d'actions élémentaires préalablement connues. Dans cet ensemble, l'objectif serait de trouver si possible, lors de la génération de modèles, les actions et les paramètres les plus appropriés reprodui-

sant au mieux le comportement décrit dans le jeu de données de référence collecté à partir d'observations effectuées sur l'espèce animale à modéliser. Ainsi, en exploitant les similarités dans les comportements élémentaires entre les espèces, ANIMETA les utilisent pour proposer des modèles qui sont ainsi explicables, car constitués d'actions connues dont le mécanisme de déclenchement est aussi connu.
Pour cela, nous avons :

- Proposé ANIMETA-MOD, un modèle générique de comportement conçu pour fonctionner avec une série d'actions paramétrables. Pour les besoins de la simulation, ce modèle générique a été intégré dans un système multi-agents (ANIMETA-SMA) développé entièrement à cet effet.
- Proposé pour le choix d'actions et le paramétrage associé, une approche qui utilise les métaheuristiques (ANIMETA-ENGINE). En raison des contraintes qu'implique cette optimisation, qui est à la fois combinatoire et continue, et de plus avec des variations de taille des solutions pour un même problème, des ajustements spécifiques à chaque métaheuristique sont nécessaires. Ainsi, quatre métaheuristiques parmi les plus reconnues, à savoir l'algorithme génétique, l'algorithme de recherche harmonique, l'algorithme immunitaire par clonage et l'algorithme de la colonie de fourmis, ont été choisis et adaptés à cette approche.
- Proposé un ensemble d'outils simples, accessibles et intuitifs pour permettre aux biologistes de créer et/ou de générer des modèles de comportement animal à partir des données qu'ils entrent dans le système (ANIMETA-HIM et ANIMETA-API). Cette approche donne au biologiste un rôle central dans le processus de construction du modèle. À chaque étape de l'optimisation, le système lui soumet des modèles, lui offrant ainsi la possibilité d'évaluer la pertinence des actions et des paramètres sélectionnés. De manière incrémentale et transparente, le biologiste contribue à la conception d'un modèle conçu avec lui et pour lui.

Toutes ces propositions ont fait l'objet d'une vérification minutieuse et d'une validation à travers diverses applications, telles que les animaux terrestres, les poissons, les animaux domestiques, et les modèles expérimentaux d'espèces.

La première série de validations visait à évaluer la similarité entre les résultats de simulation d'un modèle ANIMETA-MOD et ceux d'autres types de modèles. À cette fin, deux modèles, l'un représentant un cochon et l'autre expérimental, ont été implantés avec ANIMETA-MOD ainsi que sur deux autres plates-formes de modélisation.

Cette étape a démontré que les résultats d'ANIMETA-MOD concordaient avec ceux des autres plates-formes de modélisation. De plus, elle a mis en évidence le gain de temps possible en optant pour ANIMETA-MOD.

La deuxième série de tests s'est concentrée sur la validation de la génération de modèles. Deux types d'applications ont été utilisés ici également. La première concernait une espèce réelle de poisson, le Sciaena umbra ou Corb, pour laquelle nous avons généré un modèle de réponse de fuite à partir de données collectées dans des vidéos. La seconde application portait sur un autre modèle expérimental utilisé dans cinq scénarios différents pour illustrer différents niveaux de complexité lors de la génération de modèles,

permettant ainsi d'évaluer la capacité de notre approche à y faire face.

De ces tests variés, nous avons conclu que l'algorithme de colonie de fourmis n'était pas adapté à notre approche en raison de son principe qui restreint l'espace de recherche dans notre contexte. Comme l'on pouvait s'y attendre, à mesure que la complexité augmente, l'efficacité de l'approche diminue, soit par le maintien d'un score élevé, soit par une durée d'optimisation considérable. Heureusement, l'inclusion du biologiste dans le processus et le caractère incrémental de l'approche permettent de remédier à cette perte d'efficacité. En effet, pour des modélisations présentant un grand niveau de complexité, plutôt que d'espérer obtenir directement un modèle optimal lors de la première incrémentation, il est recommandé de réviser légèrement à la hausse le score et d'extraire les informations potentielles du modèle proposé en effectuant une analyse généralement simple. Ces informations peuvent ensuite être utilisées pour renforcer le modèle partiel de base. Cette proposition a été testée et validée dans nos séries de tests, où nous avons pu effectivement constater une réduction de la durée d'optimisation.

Bien que nous ayons connu quelques situations de contre-performance, les résultats que nous avons obtenus au terme de ces travaux sont très encourageants et ouvrent la voie à des perspectives très intéressantes. Les points qui pourraient être améliorés sont présentés dans la section suivante :

Perspectives

- La conception d'une nouvelle version de l'algorithme de colonie de fourmis :

  L'algorithme de colonie de fourmis a donné les résultats les moins satisfaisants. La cause de cette inefficacité a été identifiée comme étant le fait que l'espace de recherche demeure statique tout au long du processus d'optimisation. Afin d'améliorer les performances de cet algorithme, il serait envisageable de concevoir une nouvelle version d'adaptation qui permettrait d'introduire des mécanismes de diversification dans son espace de recherche.

  En modifiant l'algorithme de manière à rendre son espace de recherche plus dynamique, on pourrait potentiellement augmenter sa capacité à explorer un plus grand ensemble de solutions et ainsi améliorer ses performances d'optimisation. Cette approche pourrait impliquer l'ajout de mécanismes de perturbation périodiques, la modification des règles de mise à jour des phéromones, ou d'autres ajustements visant à favoriser une exploration plus diversifiée.

- La réduction de la durée d'optimisation :

  Une des contraintes les plus significatives a été la durée des opérations d'optimisation. À titre d'illustration, la phase d'optimisation la plus rapide a duré une trentaine de minutes, tandis que la plus longue a demandé plus de 42 heures. En examinant de manière approfondie le déroulement de chaque algorithme, nous avons identifié le principal goulot d'étranglement : l'évaluation des solutions à chaque itération.

  En effet, pour des algorithmes tels que celui de l'algorithme immunitaire, générant parfois jusqu'à 500 solutions par itération, leur évaluation est considérablement prolongée.

Dans un tel contexte, des solutions de parallélisation et/ou de distribution du calcul peuvent être envisagées.

Une autre approche envisageable consisterait à modifier le processus d'exécution des métaheuristiques, de manière à évaluer les solutions au fur et à mesure de leur génération. Actuellement, les solutions sont évaluées l'une après l'autre, après avoir toutes été générées. Ainsi, supposons que nous ayons 100 solutions à évaluer ; même si la solution n°1 est la meilleure, toutes les autres sont évaluées, engendrant une perte de temps considérable.

Enfin, intégrer de la possibilité d'interrompre le processus à n'importe quel moment et de récupérer la meilleure solution disponible à ce moment précis serait une fonctionnalité qui contribuerait à la flexibilité d'utilisation de ANIMETA. En permettant aux utilisateurs d'interrompre le processus à tout moment, même avant d'atteindre une solution optimale, ils peuvent accéder rapidement aux résultats générés jusqu'à ce point. Cela faciliterait une analyse des séquences récurrentes et donc une prise de décisions basées sur les résultats disponibles.

- L'élargissement de la base de données d'actions élémentaires :

  La capacité d'ANIMETA à générer des modèles optimaux, notamment en présence de très peu d'informations initiales, repose sur la complétude de sa base de données d'actions élémentaires. Actuellement, cette base est constituée de 16 actions élémentaires recueillies au cours de nos recherches. Pour améliorer la précision des modèles générés, il serait bénéfique d'élargir cette base de données. À cette fin, l'expansion pourrait inclure des comportements spécifiques à certaines classes d'espèces. Par exemple, plutôt que de se limiter à évoquer un déplacement aléatoire pour les poissons, nous pourrions détailler l'action précise de la nage, avec toutes ses caractéristiques.

  Dans cette perspective, la création d'une plateforme contributive est envisagée. Cette plateforme permettrait à la communauté des biologistes et des experts en comportement animal de soumettre des propositions d'actions afin d'obtenir une représentation plus étendue et précise des comportements.

- La définition de fonctions d'évaluation dédiées au comportement animal :

  La question de l'évaluation d'un modèle reproduisant le comportement animal a constitué une problématique majeure dans nos travaux. Cette complexité découle notamment de la présence d'une composante significative d'aléatoire dans toutes nos applications. Dans ce contexte, les méthodes classiques de vérifications point-à-point semblent peu adaptées. C'est pourquoi, en étroite collaboration avec des biologistes et des experts en comportement animal, nous envisageons la possibilité de créer des fonctions d'évaluation spécifiquement dédiées à ce domaine. Notre objectif principal consiste à définir des métriques qui captent de manière exhaustive la diversité des comportements.

Bibliographie

1. Abar S, Theodoropoulos GK, Lemarinier P et O'Hare GMP. Agent Based Modelling and Simulation tools : A review of the state-of-art software. Computer Science Review 2017 ;24 :13-33. ISSN : 1574-0137.
2. Abdullah TAA, Zahid MSM et Ali W. A Review of Interpretable ML in Healthcare : Taxonomy, Applications, Challenges, and Future Directions. Symmetry 2021 ;13. Number : 12 Publisher : Multidisciplinary Digital Publishing Institute :2439. ISSN : 2073-8994.
3. Agushaka JO, Ezugwu AE, Abualigah L, Alharbi SK et Khalifa HAEW. Efficient Initialization Methods for Population-Based Metaheuristic Algorithms : A Comparative Study. Archives of Computational Methods in Engineering 2023 ;30 :1727-87. ISSN : 1886-1784.
4. Akashah FW, Ouache R, Zhang J et Delichatsios M. A model for quantitative fire risk assessment integrating agent-based model with automatic event tree analysis. In : *Handbook of Probabilistic Models*. Sous la dir. de Samui P, Tien Bui D, Chakraborty S et Deo RC. Butterworth-Heinemann, 2020 :107-29. ISBN : 978-0-12-816514-0. DOI : 10.1016/B978-0-12-816514-0.00004-7.
5. Alcazar JA. A Simple Approach to Multi-Predator Multi-Prey Pursuit Domain. In : *Unifying Themes in Complex Systems*. Sous la dir. de Minai AA, Braha D et Bar-Yam Y. Berlin, Heidelberg : Springer, 2011 :2-9. ISBN : 978-3-642-17635-7. DOI : 10.1007/978-3-642-17635-7_1.
6. Allan R. Survey of Agent Based Modelling and Simulation Tools. 2009.
7. Amouroux E, Chu TQ, Boucher A et Drogoul A. GAMA : An Environment for Implementing and Running Spatially Explicit Multi-agent Simulations. In : *Agent Computing and Multi-Agent Systems*. Sous la dir. de Ghose A, Governatori G et Sadananda R. Lecture Notes in Computer Science. Berlin, Heidelberg : Springer, 2009 :359-71. ISBN : 978-3-642-01639-4. DOI : 10.1007/978-3-642-01639-4_32.
8. Andradóttir S. Simulation Optimization. In : *Handbook of Simulation*. Section : 9 _eprint : https ://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470172445.ch9. John Wiley & Sons, Ltd, 1998 :307-33. ISBN : 978-0-470-17244-5. DOI : 10.1002/9780470172445.ch9.
9. Anthony P, Soon G, On C, Alfred R et Lukose D. Agent Architecture : An Overview. TRANSACTIONS ON SCIENCE AND TECHNOLOGY 2014 :18-35.
10. Apsemidis A et Psarakis S. Support Vector Machines : A Review and Applications in Statistical Process Monitoring. In : *Data Analysis and Applications 3*. Section : 7 _eprint : https ://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119721871.ch7. John Wiley & Sons, Ltd, 2020 :123-44. ISBN : 978-1-119-72187-1. DOI : 10.1002/9781119721871.ch7.

11. Ashwood Z, Jha A et Pillow JW. Dynamic Inverse Reinforcement Learning for Characterizing Animal Behavior. In : Advances in Neural Information Processing Systems. 2022.
12. Axelrod R. Advancing the Art of Simulation in the Social Sciences. In : *Simulating Social Phenomena*. Sous la dir. de Conte R, Hegselmann R et Terna P. Lecture Notes in Economics and Mathematical Systems. Berlin, Heidelberg : Springer, 1997 :21-40. isbn : 978-3-662-03366-1. doi : 10.1007/978-3-662-03366-1_2.
13. Baccouche A. Incertitude et flexibilité dans l'optimisation via simulation ; application aux systèmes de production. These de doctorat. Clermont-Ferrand 2, 2012.
14. Bachmann PGH. Die analytische Zahlentheorie / dargestellt von Paul Bachmann. 2005.
15. Baerends GP. The functional organization of behaviour. Animal Behaviour 1976 ;24 :726-38. issn : 0003-3472.
16. Bakeman R et Quera V. Sequential Analysis and Observational Methods for the Behavioral Sciences. Cambridge : Cambridge University Press, 2011. isbn : 978-1-107-00124-4. doi : 10.1017/CBO9781139017343.
17. Balaji PG et Srinivasan D. An Introduction to Multi-Agent Systems. In : *Innovations in Multi-Agent Systems and Applications - 1*. Sous la dir. de Srinivasan D et Jain LC. Studies in Computational Intelligence. Berlin, Heidelberg : Springer, 2010 :1-27. isbn : 978-3-642-14435-6. doi : 10.1007/978-3-642-14435-6_1.
18. Barbati M, Bruno G et Genovese A. Applications of agent-based models for optimization problems : A literature review. Expert Systems with Applications 2012 ;39 :6020-8. issn : 0957-4174.
19. Barnhart C, Johnson EL, Nemhauser GL, Savelsbergh MWP et Vance PH. Branch-and-Price : Column Generation for Solving Huge Integer Programs. Operations Research 1998 ;46. Publisher : INFORMS :316-29. issn : 0030-364X.
20. Bateson P et Laland KN. Tinbergen's four questions : an appreciation and an update. Trends in Ecology & Evolution 2013 ;28 :712-8. issn : 0169-5347.
21. Battiti R et Tecchiolli G. The Reactive Tabu Search. ORSA Journal on Computing 1994 ;6. Publisher : ORSA :126-40. issn : 0899-1499.
22. Behera N. Chapter 8 - Analysis of microarray gene expression data using information theory and stochastic algorithm. In : *Handbook of Statistics*. Sous la dir. de Srinivasa Rao ASR et Rao CR. T. 43. Principles and Methods for Data Science. Elsevier, 2020 :349-78. doi : 10.1016/bs.host.2020.02.002.
23. Beisel T, Wiersema T, Plessl C et Brinkmann A. Cooperative multitasking for heterogeneous accelerators in the Linux Completely Fair Scheduler. In : *ASAP 2011 - 22nd IEEE International Conference on Application-specific Systems, Architectures and Processors*. ASAP 2011 - 22nd IEEE International Conference on Application-specific Systems, Architectures and Processors. ISSN : 2160-052X. 2011 :223-6. doi : 10.1109/ASAP.2011.6043273.
24. Birta LG et Arbez G. Modelling and Simulation Fundamentals. In : *Modelling and Simulation : Exploring Dynamic System Behaviour*. Sous la dir. de Birta LG et Arbez G. Simulation Foundations, Methods and Applications. Cham : Springer International Publishing, 2019 :19-53. isbn : 978-3-030-18869-6. doi : 10.1007/978-3-030-18869-6_2.
25. Bittencourt LF, Goldman A, Madeira ERM, Fonseca NLS da et Sakellariou R. Scheduling in distributed systems : A cloud computing perspective. Computer Science Review 2018 ;30 :31-54. issn : 1574-0137.
26. Blum C et Roli A. Metaheuristics in Combinatorial Optimization : Overview and Conceptual Comparison. ACM Comput. Surv. 2001 ;35 :268-308.
27. Bohnslav JP, Wimalasena NK, Clausing KJ et al. DeepEthogram, a machine learning pipeline for supervised behavior classification from raw pixels. eLife 2021 ;10 :e63377. issn : 2050-084X.
28. Boisson JC. Modélisation et résolution par métaheuristiques coopératives : de l'atome à la séquence protéique. These de doctorat. Lille 1, 2008.

30. Boumans I. Simulating pigs : Understanding their motivations, behaviour, welfare and productivity. Thèse de doct. 2017.
31. Boumans IJMM, Hofstede GJ, Bolhuis JE, Boer IJM de et Bokkers EAM. Agent-based modelling in applied ethology : An exploratory case study of behavioural dynamics in tail biting in pigs. Applied Animal Behaviour Science 2016 ;183 :10-18. ISSN : 0168-1591.
32. Boussaïd I, Lepagnot J et Siarry P. A survey on optimization metaheuristics. Information Sciences. Prediction, Control and Diagnosis using Advanced Neural Computations 2013 ;237 :82-117. ISSN : 0020-0255.
33. Bratman ME, Israel DJ et Pollack ME. Plans and resource-bounded practical reasoning. Computational intelligence 1988 ;4. Publisher : Wiley Online Library :349-55.
34. Brazier F, Dunin-keplicz B et Jennings N. Formal Specification of Multi-Agent Systems : a Real-World Case. 1996.
35. Brazier FMT, Jonker CM, Treur J et Wijngaards NJE. On the use of shared task models in knowledge acquistion, strategic user interaction and clarification agents. International Journal of Human-Computer Studies 2000 ;52 :77-110. ISSN : 1071-5819.
36. Brooks RA. Solving the find-path problem by good representation of free space. IEEE Transactions on Systems, Man, and Cybernetics 1983 ;SMC-13. Conference Name : IEEE Transactions on Systems, Man, and Cybernetics :190-7. ISSN : 2168-2909.
37. Browning E, Bolton M, Owen E, Shoji A, Guilford T et Freeman R. Predicting animal behaviour using deep learning : GPS data alone accurately predict diving in seabirds. Methods in Ecology and Evolution 2018 ;9. _eprint : https ://onlinelibrary.wiley.com/doi/pdf/10.1111/2041-210X.12926 :681-92. ISSN : 2041-210X.
38. Burrell J. How the machine 'thinks' : Understanding opacity in machine learning algorithms. Big Data & Society 2016 ;3. Publisher : SAGE Publications Ltd :2053951715622512. ISSN : 2053-9517.
39. Butts DJ, Thompson NE, Christensen SA, Williams DM et Murillo MS. Data-driven agent-based model building for animal movement through Exploratory Data Analysis. Ecological Modelling 2022 ;470 :110001. ISSN : 0304-3800.
40. Caillou P, Gaudou B, Grignard A, TRUONG Chi Q et Taillandier P. A simple-to-use BDI architecture for agent-based modeling and simulation. 2017.
41. Cardoso R, Kourtis G, Dennis L et al. A Review of Verification and Validation for Space Autonomous Systems. Current Robotics Reports 2021 ;2.
42. Carroll G, Slip D, Jonsen I et Harcourt R. Supervised accelerometry analysis can identify prey capture by penguins at sea. Journal of Experimental Biology 2014 ;217. Publisher : The Company of Biologists :4295-302. ISSN : 0022-0949.
43. Systems and Models. In : *Introduction to Discrete Event Systems*. Sous la dir. de Cassandras CG et Lafortune S. Boston, MA : Springer US, 2008 :1-51. ISBN : 978-0-387-68612-7. DOI : 10.1007/978-0-387-68612-7_1.
44. Ceballos G, Ehrlich PR, Barnosky AD, García A, Pringle RM et Palmer TM. Accelerated modern human–induced species losses : Entering the sixth mass extinction. Science Advances 2015 ;1. Publisher : American Association for the Advancement of Science :e1400253.
45. Cha SH et Srihari SN. On measuring the distance between histograms. Pattern Recognition 2002 ;35 :1355-70. ISSN : 0031-3203.
46. Chaachoua H et Saglam A. Modelling by differential equations. Teaching Mathematics and Its Applications : International Journal of the IMA 2006 ;25. Conference Name : Teaching Mathematics and Its Applications : International Journal of the IMA :15-22. ISSN : 1471-6976.
47. Chen X, Liu Q, Mallet F, Li Q, Cai S et Jin Z. Formally verifying consistency of sequence diagrams for safety critical systems. Science of Computer Programming 2022 ;216 :102777. ISSN : 0167-6423.

48. Cheyssac J. Etude comportementale et resocialisation des chimpanzés captifs : approche méthodologique et applications. 2015 :151.
49. Chong HQ, Tan AH et Ng GW. Integrated cognitive architectures : A survey. Artificial Intelligence Review 2007 ;28 :103-30.
50. Çimen ME, Garip Z et Boz AF. Comparison of metaheuristic optimization algorithms for numerical solutions of optimal control problems. Concurrency and Computation : Practice and Experience 2023 ;35. _eprint : https ://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.7663 :e7663. issn : 1532-0634.
51. Collette Y et Siarry P. Optimisation multiobjectif : Algorithmes. Editions Eyrolles, 2011. 294 p. isbn : 978-2-212-16752-8.
52. Cook SA. An overview of computational complexity. In : *ACM Turing award lectures*. New York, NY, USA : Association for Computing Machinery, 2007 :1982. isbn : 978-1-4503-1049-9.
53. Coquelle L. Simulation de comportements individuels instinctifs d'animaux dans leur environnement : De la description éthologique à l'exécution de comportements réactifs. Université de Bretagne Occidentale 2005 :165.
54. Coquillard P et Hill D. Modélisation et simulation d'écosystèmes : des modèles déterministes aux simulations à événements discrets. Paris : Masson, 1997. 273 p. isbn : 978-2-225-85363-0.
55. Costantini S et Tocchio A. About Declarative Semantics of Logic-Based Agent Languages. In : *Declarative Agent Languages and Technologies III*. Sous la dir. de Baldoni M, Endriss U, Omicini A et Torroni P. Lecture Notes in Computer Science. Berlin, Heidelberg : Springer, 2006 :106-23. isbn : 978-3-540-33107-0. doi : 10.1007/11691792_7.
56. Darwin C. De l'origine des espèces ou des lois du progrès chez les êtres organisés. 1859.
57. Darwin C. The Expression of the Emotions in Man and Animals. 1872.
58. De Castro L et Von Zuben F. The Clonal Selection Algorithm with Engineering Applications. Artificial Immune Systems 2001 ;8.
59. Dehghani M, Trojovská E et Zuščák T. A new human-inspired metaheuristic algorithm for solving optimization problems based on mimicking sewing training. Scientific Reports 2022 ;12. Number : 1 Publisher : Nature Publishing Group :17387. issn : 2045-2322.
60. Deng L et Yu D. Deep Learning : Methods and Applications. Foundations and Trends® in Signal Processing 2014 ;7. Publisher : Now Publishers, Inc. :197-387. issn : 1932-8346, 1932-8354.
61. Deris AM, Zain AM et Sallehuddin R. Overview of Support Vector Machine in Modeling Machining Performances. Procedia Engineering. International Conference on Advances in Engineering 2011 2011 ;24 :308-12. issn : 1877-7058.
62. Descombe B. Liste rouge de l'UICN : Les activités humaines dévastent les espèces marines, des mammifères aux coraux. UICN France. 2022.
63. Dokeroglu T, Sevinc E, Kucukyilmaz T et Cosar A. A survey on new generation metaheuristic algorithms. Computers & Industrial Engineering 2019 ;137 :106040. issn : 0360-8352.
64. Domínguez A, Juan A et Kizys R. A Survey on Financial Applications of Metaheuristics. ACM Computing Surveys 2017 ;50 :1-23.
65. Dong Y, Chbat NW, Gupta A, Hadzikadic M et Gajic O. Systems modeling and simulation applications for critical care medicine. Annals of Intensive Care 2012 ;2 :18. issn : 2110-5820.
66. Dorigo M, Maniezzo V et Colorni A. Ant system : optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 1996 ;26. Conference Name : IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) :29-41. issn : 1941-0492.
67. Doshi-Velez F et Kim B. Towards A Rigorous Science of Interpretable Machine Learning. arXiv : Machine Learning 2017.

68. Dréo J, Pétrowski A, Siarry P et Taillard É. Métaheuristiques pour l'optimisation difficile : recuit simulé, recherche avec tabous, algorithmes évolutionnaires et algorithmes génétiques, colonies de fourmis... Algorithmes. Paris : Eyrolles, 2003. ISBN : 978-2-212-11368-6.

69. Drogoul A. De la simulation multi-agents a la resolution collective de problemes : une etude de l'emergence de structures d'organisation dans les systemes multi-agents. These de doctorat. Paris 6, 1993.

70. Ducos S. Restauration de population du denti Dentex dentex et du corb Sciaena umbra en Méditerranée : évaluation des performances individuelles, suivi individuel et populationnel des juvéniles. These de doctorat. Corte, 2023.

71. Ebert C, Cain J, Antoniol G, Counsell S et Laplante P. Cyclomatic Complexity. IEEE Software 2016 ;33. Conference Name : IEEE Software :27-9. ISSN : 1937-4194.

72. The Simplex Method. In : *Linear Programming and its Applications*. Sous la dir. d'Eiselt HA et Sandblom C. Berlin, Heidelberg : Springer, 2007 :129-65. ISBN : 978-3-540-73671-4. DOI : 10.1007/978-3-540-73671-4_5.

73. Elshaer R et Awad H. A taxonomic review of metaheuristic algorithms for solving the vehicle routing problem and its variants. Computers & Industrial Engineering 2020 ;140 :106242. ISSN : 0360-8352.

74. Ericsson AC, Crim MJ et Franklin CL. A Brief History of Animal Modeling. Missouri Medicine 2013 ;110 :201-5. ISSN : 0026-6620.

75. Ezugwu AE, Shukla AK, Nath R et al. Metaheuristics : a comprehensive overview and classification along with bibliometric analysis. Artificial Intelligence Review 2021 ;54 :4237-316. ISSN : 1573-7462.

76. Farmer JD, Packard NH et Perelson AS. The immune system, adaptation, and machine learning. Physica D 1986 ;2 :187-204. ISSN : 0167-2789.

77. Ferreira D, França PM, Kimms A, Morabito R, Rangel S et Toledo CFM. Heuristics and meta-heuristics for lot sizing and scheduling in the soft drinks industry : a comparison study. In : *Metaheuristics for Scheduling in Industrial and Manufacturing Applications*. Sous la dir. de Xhafa F et Abraham A. Studies in Computational Intelligence. Berlin, Heidelberg : Springer, 2008 :169-210. ISBN : 978-3-540-78985-7. DOI : 10.1007/978-3-540-78985-7_8.

78. Fielding RT. Architectural Styles and the Design of Network-based Software Architectures. In : 2000.

79. Filippi JB, Bosseur F, Pialat X, Santoni PA, Strada S et Mari C. Simulation of Coupled Fire/Atmosphere Interaction with the MesoNH-ForeFire Models. Journal of Combustion 2011 ;540390.

80. Foead D, Ghifari A, Kusuma MB, Hanafiah N et Gunawan E. A Systematic Literature Review of A* Pathfinding. Procedia Computer Science. 5th International Conference on Computer Science and Computational Intelligence 2020 2021 ;179 :507-14. ISSN : 1877-0509.

81. Foramitti J. AgentPy : A package for agent-based modeling in Python. Journal of Open Source Software 2021 ;6 :3065. ISSN : 2475-9066.

82. Franceschini RR. Approche formelle pour la modélisation et la simulation à évènements discrets de systèmes multi-agents. Thèse de doct. Université Pascal Paoli, 2017.

83. Frantz FK. A taxonomy of model abstraction techniques. In : *Proceedings of the 27th conference on Winter simulation*. WSC '95. USA : IEEE Computer Society, 1995 :1413-20. ISBN : 978-0-7803-3018-4. DOI : 10.1145/224401.224834.

84. Gadagkar R. More Fun Than Fun : The Joys and Burdens of Our Heroes – The Wire Science. 2021.

85. Geem ZW. Improved Harmony Search from Ensemble of Music Players. In : *Knowledge-Based Intelligent Information and Engineering Systems*. Sous la dir. de Gabrys B, Howlett RJ et Jain LC. Lecture Notes in Computer Science. Berlin, Heidelberg : Springer, 2006 :86-93. ISBN : 978-3-540-46536-2. DOI : 10.1007/11892960_11.

86. Geem ZW, Kim JH et Loganathan G. A New Heuristic Optimization Algorithm : Harmony Search. SIMULATION 2001 ;76. Publisher : SAGE Publications Ltd STM :60-8. ISSN : 0037-5497.

87. Gendreau M et Potvin JY, éd. Handbook of Metaheuristics. T. 146. International Series in Operations Research & Management Science. Boston, MA : Springer US, 2010. ISBN : 978-1-4419-1663-1 978-1-4419-1665-5. DOI : 10.1007/978-1-4419-1665-5.

88. Genero M, Fernández-Saez AM, Nelson HJ, Poels G et Piattini M. Research Review : A Systematic Literature Review on the Quality of UML Models. Journal of Database Management (JDM) 2011 ;22. Publisher : IGI Global :46-70. ISSN : 1063-8016.

89. Genesereth MR et Ketchpel SP. Software agents. Communications of the ACM 1994 ;37 :48-ff. ISSN : 0001-0782.

90. Georgeff MP et Ingrand F. Decision-Making in an Embedded Reasoning System. In : International Joint Conference on Artificial Intelligence. 1989.

91. Gilpin LH, Bau D, Yuan BZ, Bajwa A, Specter M et Kagal L. Explaining Explanations : An Overview of Interpretability of Machine Learning. In : *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*. 2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA). 2018 :80-9. DOI : 10.1109/DSAA.2018.00018.

92. Giraldeau LA et Dubois F. Le comportement animal. 2015.

93. Glover F. Future paths for integer programming and links to artificial intelligence. Computers & Operations Research. Applications of Integer Programming 1986 ;13 :533-49. ISSN : 0305-0548.

94. Gogna A et Tayal A. Metaheuristics : review and application. Journal of Experimental & Theoretical Artificial Intelligence 2013 ;25. Publisher : Taylor & Francis _eprint : https ://doi.org/10.1080/0952813X.2013.782347 :50 26. ISSN : 0952-813X.

95. Goldberg DE. Genetic Algorithms in Search, Optimization and Machine Learning. 1st. USA : Addison-Wesley Longman Publishing Co., Inc., 1989. 372 p. ISBN : 978-0-201-15767-3.

96. Gong W et Zhou X. A survey of SAT solver. AIP Conference Proceedings 2017 ;1836 :020059. ISSN : 0094-243X.

97. Gopalakrishnan K. Particle Swarm Optimization in Civil Infrastructure Systems : State-of-the-Art Review. In : *Metaheuristic Applications in Structures and Infrastructures*. Sous la dir. de Gandomi AH, Yang XS, Talatahari S et Alavi AH. Oxford : Elsevier, 2013 :49-76. ISBN : 978-0-12-398364-0. DOI : https://doi.org/10.1016/B978-0-12-398364-0.00003-6.

98. Grimm V. Ecological Models : Individual-Based Models. In : *Encyclopedia of Ecology (Second Edition)*. Sous la dir. de Fath B. Oxford : Elsevier, 2019 :65-73. ISBN : 978-0-444-64130-4. DOI : 10.1016/B978-0-12-409548-9.11144-3.

99. Grünewälder S, Broekhuis F, Macdonald DW et al. Movement Activity Based Classification of Animal Behaviour with an Application to Data from Cheetah (Acinonyx jubatus). PLOS ONE 2012 ;7. Publisher : Public Library of Science :e49120. ISSN : 1932-6203.

100. Guerra-Hernández A, El Fallah-Seghrouchni A et Soldano H. Learning in BDI Multi-agent Systems. In : *Computational Logic in Multi-Agent Systems*. Sous la dir. de Dix J et Leite J. Lecture Notes in Computer Science. Berlin, Heidelberg : Springer, 2005 :218-33. ISBN : 978-3-540-30200-1. DOI : 10.1007/978-3-540-30200-1_12.

101. Gunantara N. A review of multi-objective optimization : Methods and its applications. Cogent Engineering 2018 ;5. Sous la dir. d'Ai Q. Publisher : Cogent OA _eprint : https ://doi.org/10.1080/23311916.2018.1502242 :15 ISSN : null.

102. Han Y, Misra S et Jin Y. Multifrequency electromagnetic data acquisition and interpretation in the laboratory and in the subsurface : A comprehensive review. In : *Multifrequency Electromagnetic Data Interpretation for Subsurface Characterization*. Sous la dir. de Misra S, Han Y, Jin Y et Tathed P. Elsevier, 2021 :3-70. ISBN : 978-0-12-821439-8. DOI : 10.1016/B978-0-12-821439-8.00005-7.

103. Handcock RN, Swain DL, Bishop-Hurley GJ et al. Monitoring Animal Behaviour and Environmental Interactions Using Wireless Sensor Networks, GPS Collars and Satellite Remote Sensing. Sensors 2009 ;9. Number : 5 Publisher : Molecular Diversity Preservation International :3586-603. ISSN : 1424-8220.

104. Helbing D. Agent-Based Modeling. In : *Social Self-Organization : Agent-Based Simulations and Experiments to Study Emergent Social Behavior*. Sous la dir. d'Helbing D. Understanding Complex Systems. Berlin, Heidelberg : Springer, 2012 :25-70. ISBN : 978-3-642-24004-1. DOI : 10.1007/978-3-642-24004-1_2.

105. Hilaire JBS et Aristotle. Histoire des Animaux D'Aristote Traduite en Français. Google-Books-ID : znkhAQAAIAAJ. Hachette et cie., 1883. 608 p.

106. Holland JH. Adaptation in Natural and Artificial Systems : An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. 1992. DOI : 10.7551/mitpress/1090.001.0001.

107. Hopcroft JE, Motwani R et Ullman JD. Introduction to automata theory, languages, and computation, 2nd edition. ACM SIGACT News 2001 ;32 :60-5. ISSN : 0163-5700.

108. Huang C, Li Y et Yao X. A Survey of Automatic Parameter Tuning Methods for Metaheuristics. IEEE Transactions on Evolutionary Computation 2020 ;24. Conference Name : IEEE Transactions on Evolutionary Computation :201-16. ISSN : 1941-0026.

109. Hyndman RJ et Koehler AB. Another look at measures of forecast accuracy. International Journal of Forecasting 2006 ;22 :679-88. ISSN : 0169-2070.

110. Iba H. Multi-agent simulation based on swarm. In : *Agent-Based Modeling and Simulation with Swarm*. Google-Books-ID : xVPSBQAAQBAJ. CRC Press, 2013 :45-73. ISBN : 978-1-4665-6240-0.

111. Im DJ, Kwak I et Branson K. Evaluation metrics for behaviour modeling. arXiv :2007.12298 [cs, stat] 2020.

112. Ingalls GR. Introduction to simulation. In : 33nd conference on Winter simulation. 2001 :7-16.

114. Janiesch C, Zschech P et Heinrich K. Machine learning and deep learning. Electronic Markets 2021 ;31 :685-95. ISSN : 1422-8890.

115. Jaxa-Rozen M et Kwakkel JH. PyNetLogo : Linking NetLogo with Python. Journal of Artificial Societies and Social Simulation 2018 ;21 :4. ISSN : 1460-7425.

116. Jonker CM et Treur J. Agent-Based Simulation of Animal Behaviour. Applied Intelligence 2001 ;15 :83-115. ISSN : 1573-7497.

117. Joshi SK et Bansal JC. Parameter tuning for meta-heuristics. Knowledge-Based Systems 2020 ;189 :105094. ISSN : 0950-7051.

118. Jourdan L, Basseur M et Talbi EG. Hybridizing Exact methods and Metaheuristics : A taxonomy. European Journal of Operational Research 2009 ;199 :620-9.

119. Juan AA, Faulin J, Grasman SE, Rabe M et Figueira G. A review of simheuristics : Extending metaheuristics to deal with stochastic combinatorial optimization problems. Operations Research Perspectives 2015 ;2 :62-72. ISSN : 2214-7160.

120. Karunanithy K et Velusamy B. An efficient data collection using wireless sensor networks and internet of things to monitor the wild animals in the reserved area. Peer-to-Peer Networking and Applications 2022 ;15 :1105-25. ISSN : 1936-6450.

121. Kasereka S, Kasoro N, Kyamakya K, Doungmo Goufo EF, Chokki AP et Yengo MV. Agent-Based Modelling and Simulation for evacuation of people from a building in case of fire. Procedia Computer Science. The 9th International Conference on Ambient Systems, Networks and Technologies (ANT 2018) / The 8th International Conference on Sustainable Energy Information Technology (SEIT-2018) / Affiliated Workshops 2018 ;130 :10-17. ISSN : 1877-0509.

122. Katoch S, Chauhan SS et Kumar V. A review on genetic algorithm : past, present, and future. Multimedia Tools and Applications 2021 ;80 :8091-126. issn : 1573-7721.

123. Kaur S, Kumar Y, Koul A et Kumar Kamboj S. A Systematic Review on Metaheuristic Optimization Techniques for Feature Selections in Disease Diagnosis : Open Issues and Challenges. Archives of Computational Methods in Engineering 2023 ;30 :1863-95. issn : 1134-3060.

124. Kazil J, Masad D et Crooks A. Utilizing Python for Agent-Based Modeling : The Mesa Framework. 2020. isbn : 978-3-030-61254-2. doi : 10.1007/978-3-030-61255-9_30.

125. Kirkpatrick S, Gelatt CD et Vecchi MP. Optimization by Simulated Annealing. Science 1983 ;220. Publisher : American Association for the Advancement of Science :671-80.

126. Korn GA. Desire software for interactive modelling of environmental systems. Systems Analysis Modelling Simulation 1996 ;25 :179-89. issn : 0232-9298.

127. Korte B, Vygen J, Fonlupt J et Skoda A. Programmation linéaire. In : *Optimisation combinatoire : Théorie et algorithmes*. Sous la dir. de Korte B, Vygen J, Fonlupt J et Skoda A. Collection IRIS. Paris : Springer, 2010 :51-72. isbn : 978-2-287-99037-3. doi : 10.1007/978-2-287-99037-3_3.

128. Kotsiantis SB. Decision trees : a recent overview. Artificial Intelligence Review 2013 ;39 :261-83. issn : 1573-7462.

129. Kozen D. On parallelism in turing machines. In : *17th Annual Symposium on Foundations of Computer Science (sfcs 1976)*. 17th Annual Symposium on Foundations of Computer Science (sfcs 1976). ISSN : 0272-5428. 1976 :89-97. doi : 10.1109/SFCS.1976.20.

130. Kraibi K, Ben Ayed R, Collart-Dutilleul S, Bon P et PEIT D. Analysis and Formal Modeling of Systems Behavior Using UML/Event-B. Journal of Communications 2019 :980-6.

131. Ladyman J, Lambert J et Wiesner K. What is a complex system ? European Journal for Philosophy of Science 2013 ;3 :33-67. issn : 1879-4920.

132. Lambora A, Gupta K et Chopra K. Genetic Algorithm- A Literature Review. In : *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*. 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon). 2019 :380-4. doi : 10.1109/COMITCon.2019.8862255.

133. Landau E. Handbuch der Lehre von der Verteilung der Primzahlen. Von dr. Edmund Landau. 2005.

134. Laporte G. The traveling salesman problem : An overview of exact and approximate algorithms. European Journal of Operational Research 1992 ;59 :231-47. issn : 0377-2217.

135. Laubenbacher R, Hinkelmann F et Oremland M. Agent-Based Models and Optimal Control in Biology : A Discrete Approach. In : *Mathematical Concepts and Methods in Modern Biology*. Sous la dir. de Robeva R et Hodge TL. Boston : Academic Press, 2013 :143-78. isbn : 978-0-12-415780-4. doi : 10.1016/B978-0-12-415780-4.00005-3.

136. Leach MC, Coulter CA, Richardson CA et Flecknell PA. Are We Looking in the Wrong Place ? Implications for Behavioural-Based Pain Assessment in Rabbits (Oryctolagus cuniculi) and Beyond ? PLOS ONE 2011 ;6. Publisher : Public Library of Science :e13347. issn : 1932-6203.

137. Lebar Bajec I, Zimic N et Mraz M. The computational beauty of flocking : Boids revisited. Mathematical and Computer Modelling of Dynamical Systems - MATH COMPUT MODEL DYNAM SYST 2007 ;13 :331-47.

138. Legrain A, Bouarab H et Lahrichi N. The Nurse Scheduling Problem in Real-Life. Journal of Medical Systems 2014 ;39 :160. issn : 1573-689X.

139. Leimeister JM. Collective Intelligence. Business & Information Systems Engineering 2010 ;2 :245-8. issn : 1867-0202.

140. Li Q. Interpretable machine learning for malware detection and adversarial defense. Publisher : McGill University. Thèse de doct. Montreal : McGill University, 2022. 141 p.

141. Linardatos P, Papastefanopoulos V et Kotsiantis S. Explainable AI : A Review of Machine Learning Interpretability Methods. Entropy 2021 ;23. Number : 1 Publisher : Multidisciplinary Digital Publishing Institute :18. ISSN : 1099-4300.

142. Lorenz KZ. The comparative method in studying innate behavior patterns. In : *Physiological mechanisms in animal behavior. (Society's Symposium IV.)* Oxford, England : Academic Press, 1950 :221-68.

143. Luke S, Cioffi-Revilla C, Panait L, Sullivan K et Balan G. MASON : A Multiagent Simulation Environment. SIMULATION 2005 ;81. Publisher : SAGE Publications Ltd STM :517-27. ISSN : 0037-5497.

144. Luncean L et Becheru A. Communication and interaction in a multi-agent system devised for transport brokering. In : 2015.

145. Ma Y, Gu X et Wang Y. Histogram similarity measure using variable bin size distance. Computer Vision and Image Understanding 2010 ;114 :981-9. ISSN : 1077-3142.

146. Manicacci FM, Mourier J, Babatounde C et al. A Wireless Autonomous Real-Time Underwater Acoustic Positioning System. Sensors 2022 ;22. Number : 21 Publisher : Multidisciplinary Digital Publishing Institute :8208. ISSN : 1424-8220.

147. Manolis E, Osman TE, Herold R et al. Role of modeling and simulation in pediatric investigation plans. Pediatric Anesthesia 2011 ;21. _eprint : https ://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1460-9592.2011.03523.x :214-21. ISSN : 1460-9592.

148. Martiskainen P, Järvinen M, Skön JP, Tiirikainen J, Kolehmainen M et Mononen J. Cow behaviour pattern recognition using a three-dimensional accelerometer and support vector machines. Applied Animal Behaviour Science 2009 ;119 :32-8. ISSN : 0168-1591.

149. Mathis A, Mamidanna P, Cury KM et al. DeepLabCut : markerless pose estimation of user-defined body parts with deep learning. Nature Neuroscience 2018 ;21. Number : 9 Publisher : Nature Publishing Group :1281-9. ISSN : 1546-1726.

150. Matthews RB, Gilbert NG, Roach A, Polhill JG et Gotts NM. Agent-based land-use models : a review of applications. Landscape Ecology 2007 ;22 :1447-59. ISSN : 1572-9761.

151. McCabe T. Cyclomatic complexity and the year 2000. IEEE Software 1996 ;13. Conference Name : IEEE Software :115-7. ISSN : 1937-4194.

152. McDougall W. An Outline of Abnormal Psychology. London : Routledge, 1926. 590 p. ISBN : 978-1-315-67366-0. DOI : 10.4324/9781315673660.

153. Mehdizadeh M, Nordfjaern T et Klöckner CA. A systematic review of the agent-based modelling/simulation paradigm in mobility transition. Technological Forecasting and Social Change 2022 ;184 :122011. ISSN : 0040-1625.

154. Michel F. Formalisme, outils et éléments méthodologiques pour la modélisation et la simulation multi-agents. Thèse de doct. Montpellier II, 2004.

155. Miller T. Explanation in artificial intelligence : Insights from the social sciences. Artificial Intelligence 2019 ;267 :1-38. ISSN : 0004-3702.

156. Misik S, Cela A et Bradac Z. Distributed Systems - A brief review of theory and practice. IFAC-PapersOnLine. 14th IFAC Conference on Programmable Devices and Embedded Systems PDES 2016 2016 ;49 :318-23. ISSN : 2405-8963.

157. Mittelstadt B, Russell C et Wachter S. Explaining Explanations in AI. In : *Proceedings of the Conference on Fairness, Accountability, and Transparency*. FAT* '19. New York, NY, USA : Association for Computing Machinery, 2019 :279-88. ISBN : 978-1-4503-6125-5. DOI : 10.1145/3287560.3287574.

158. Mladenović N et Hansen P. Variable neighborhood search. Computers & Operations Research 1997 ;24 :1097-100. ISSN : 0305-0548.

159. Mor B, Garhwal S et Kumar A. A Systematic Review of Hidden Markov Models and Their Applications. Archives of Computational Methods in Engineering 2021 ;28 :1429-48. issn : 1886-1784.

160. Morrison DR, Jacobson SH, Sauppe JJ et Sewell EC. Branch-and-bound algorithms : A survey of recent advances in searching, branching, and pruning. Discrete Optimization 2016 ;19 :79-102. issn : 1572-5286.

161. Nesmachnow S. An overview of metaheuristics : accurate and efficient methods for optimisation. International Journal of Metaheuristics 2015. Publisher : Inderscience Publishers (IEL).

162. Nichols JA, Herbert Chan HW et Baker MAB. Machine learning : applications of artificial intelligence to imaging and diagnosis. Biophysical Reviews 2018 ;11 :111-8. issn : 1867-2450.

163. North M, Collier N et Vos J. Experiences Creating Three Implementations of the Repast Agent Modeling Toolkit. ACM Trans. Model. Comput. Simul. 2006 ;16 :1-25.

164. Nowicki S et Searcy WA. Song and Mate Choice in Birds : How the Development of Behavior Helps us Understand Function. The Auk 2005 ;122. Place : US Publisher : Ornithological Society of North America :1-14. issn : 0004-8038(Print).

165. Oberkampf WL et Roy CJ. Verification and Validation in Scientific Computing. Cambridge : Cambridge University Press, 2010. isbn : 978-0-521-11360-1. doi : 10.1017/CBO9780511760396.

166. Omidvar MN, Li X et Yao X. A Review of Population-Based Metaheuristics for Large-Scale Black-Box Global Optimization—Part I. IEEE Transactions on Evolutionary Computation 2022 ;26. Conference Name : IEEE Transactions on Evolutionary Computation :802-22. issn : 1941-0026.

167. Ord T, Martins E, Thakur S, Mane K et Borner K. Trends in animal behaviour research (1968-2002) : Ethoinformatics and the mining of library databases. Animal Behaviour 2005 ;69 :1399-413.

168. Özgün O et Barlas Y. Discrete vs. Continuous Simulation : When Does It Matter ? In : 27th International Conference of The System Dynamics Society. 2009.

169. Papadimitriou CH. Computational complexity. In : *Encyclopedia of Computer Science*. GBR : John Wiley et Sons Ltd., 2003 :260-5. isbn : 978-0-470-86412-8.

170. Phung T, Winikoff M et Padgham L. Learning Within the BDI Framework : An Empirical Analysis. In : *Knowledge-Based Intelligent Information and Engineering Systems*. Sous la dir. de Khosla R, Howlett RJ et Jain LC. Lecture Notes in Computer Science. Berlin, Heidelberg : Springer, 2005 :282-8. isbn : 978-3-540-31990-0. doi : 10.1007/11553939_41.

171. Pnueli A. Specification and Development of Reactive Systems (Invited Paper). In : *Information Processing 86, Proceedings of the IFIP 10th World Computer Congress, Dublin, Ireland, September 1-5, 1986*. Sous la dir. de Kugler HJ. North-Holland/IFIP, 1986 :845-58.

172. Poggi B. Développement de concepts et outils d'aide à la décision pour l'optimisation via simulation. Theses. SPE UMR CNRS 6134 ; Université de Corse, 2014.

173. Poggi B, Babatounde C, Vittori E et Antoine-Santoni T. Efficient WSN Node Placement by Coupling KNN Machine Learning for Signal Estimations and I-HBIA Metaheuristic Algorithm for Node Position Optimization. Sensors 2022 ;22. Number : 24 Publisher : Multidisciplinary Digital Publishing Institute :9927. issn : 1424-8220.

174. Pop CB, Cioara T, Anghel I et al. Review of bio-inspired optimization applications in renewable-powered smart grids : Emerging population-based metaheuristics. Energy Reports 2022 ;8 :11769-98. issn : 2352-4847.

175. Quinlan J. Decision trees and decision-making. IEEE Transactions on Systems, Man, and Cybernetics 1990 ;20. Conference Name : IEEE Transactions on Systems, Man, and Cybernetics :339-46. issn : 2168-2909.

176. Rabbinge R et Wit CTd. Systems, models and simulation. Simulation and systems management in crop protection 1989. Publisher : Pudoc :3-15.

177. Rabbouch B, Rabbouch H, Saâdaoui F et Mraihi R. Foundations of combinatorial optimization, heuristics, and metaheuristics. In : *Comprehensive Metaheuristics*. Sous la dir. de Mirjalili S et Gandomi AH. Academic Press, 2023 :407-38. isbn : 978-0-323-91781-0. doi : 10.1016/B978-0-323-91781-0.00022-3.

178. Rai R, Das A, Ray S et Dhal KG. Human-Inspired Optimization Algorithms : Theoretical Foundations, Algorithms, Open-Research Issues and Application for Multi-Level Thresholding. Archives of Computational Methods in Engineering 2022 ;29 :5313-52. issn : 1886-1784.

179. Rao AS et Georgeff MP. An abstract architecture for rational agents. In : *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*. KR'92. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1992 :439-49. isbn : 978-1-55860-262-5.

180. Rasmussen RV et Trick MA. Round robin scheduling – a survey. European Journal of Operational Research 2008 ;188 :617-36. issn : 0377-2217.

181. Reynolds CW. Flocks, herds and schools : A distributed behavioral model. In : 14th annual conference on Computer graphics and interactive techniques. 1987.

182. Romanes G. Animal Intelligence. Kegan Paul, Trench, & Co. London, 1882.

183. Roni MHK, Rana MS, Pota HR, Hasan MM et Hussain MS. Recent trends in bio-inspired metaheuristic optimization techniques in control applications for electrical systems : a review. International Journal of Dynamics and Control 2022 ;10 :999-1011. issn : 2195-2698.

184. Russell S et Norvig P. Artificial Intelligence : A Modern Approach. Prentice Hall. 2nd Edition. 2002.

185. Sahin O et Akay B. Comparisons of metaheuristic algorithms and fitness functions on software test data generation. Applied Soft Computing 2016 ;49 :1202-14. issn : 1568-4946.

186. Sarang P. Regression Analysis. In : *Thinking Data Science : A Data Science Practitioner's Guide*. Sous la dir. de Sarang P. The Springer Series in Applied Machine Learning. Cham : Springer International Publishing, 2023 :55-73. isbn : 978-3-031-02363-7. doi : 10.1007/978-3-031-02363-7_3.

187. Schank J, Joshi S, May C, Tran JT et Bish R. A Multi-Modeling Approach to the Study of Animal Behavior. In : *Unifying Themes in Complex Systems*. Sous la dir. de Minai AA, Braha D et Bar-Yam Y. Berlin, Heidelberg : Springer, 2011 :304-12. isbn : 978-3-642-17635-7. doi : 10.1007/978-3-642-17635-7_37.

188. Shannon R. Introduction to the art and science of simulation. In : *1998 Winter Simulation Conference. Proceedings (Cat. No.98CH36274)*. 1998 Winter Simulation Conference. Proceedings (Cat. No.98CH36274). T. 1. 1998 :7-14 vol.1. doi : 10.1109/WSC.1998.744892.

189. Al-Sharawneh J et Williams MA. ABMS : Agent-Based Modeling and Simulation in Web Service Selection. In : International Conference on Management and Service Science, MASS. Journal Abbreviation : Proceedings - International Conference on Management and Service Science, MASS 2009 Publication Title : Proceedings - International Conference on Management and Service Science, MASS 2009. Beijing, China, 2009 :6. doi : 10.1109/ICMSS.2009.5301897.

190. Shawki KM, Kilani K et Gomaa MA. Analysis of earth-moving systems using discrete-event simulation. Alexandria Engineering Journal 2015 ;54 :533-40. issn : 1110-0168.

191. Simsir F et Ekmekci D. A metaheuristic solution approach to capacitied vehicle routing and network optimization. Engineering Science and Technology, an International Journal 2019 ;22 :727-35. issn : 2215-0986.

192. Singh MP. Agent Communication Languages : Rethinking the Principles. In : *Communication in Multiagent Systems : Agent Communication Languages and Conversation Policies*. Sous la dir. d'Huget MP. Lecture Notes in Computer Science. Berlin, Heidelberg : Springer, 2003 :37-50. isbn : 978-3-540-44972-0. doi : 10.1007/978-3-540-44972-0_2.

193. Sober E. Morgan's canon. In : *The evolution of mind*. New York, NY, US : Oxford University Press, 1998 :224-42. isbn : 0-19-511053-6 (Hardcover).

194. Steels L. Cooperation between distributed agents through self-orcamsation. In : *proceedings of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World, Cambridge, England*. Citeseer, 1990.

195. Taillandier P, Vo DA, Amouroux E et Drogoul A. GAMA : A Simulation Platform That Integrates Geographical Information Data, Agent-Based Modeling and Multi-scale Control. In : *Principles and Practice of Multi-Agent Systems*. Sous la dir. de Desai N, Liu A et Winikoff M. Lecture Notes in Computer Science. Berlin, Heidelberg : Springer, 2012 :242-58. isbn : 978-3-642-25920-3. doi : 10.1007/978-3-642-25920-3_17.

196. Talbi EG. A Unified Taxonomy of Hybrid Metaheuristics with Mathematical Programming, Constraint Programming and Machine Learning. Studies in Computational Intelligence 2013 ;434 :3-76. issn : 978-3-642-30670-9.

197. Tang W et Bennett DA. Agent-based Modeling of Animal Movement : A Review. Geography Compass 2010 ;4. _eprint : https ://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1749-8198.2010.00337.x :682-700. issn : 1749-8198.

198. Tarjan R. Depth-First Search and Linear Graph Algorithms. SIAM Journal on Computing 1972 ;1. Publisher : Society for Industrial and Applied Mathematics :146-60. issn : 0097-5397.

199. Tawarmalani M et Sahinidis NV. A polyhedral branch-and-cut approach to global optimization. Mathematical Programming 2005 ;103 :225-49. issn : 1436-4646.

200. Taylor C, Koyuk H, Coyle J, Waggoner R et Newman K. An Agent-Based Model of Predator-Prey Relationships Between Transient Killer Whales and Other Marine Mammals. 2007.

201. Thomas RK. Thinking Clearly About Concepts in Behavioral Science. 2001.

202. Tinbergen N. On aims and methods of Ethology. Zeitschrift für Tierpsychologie 1963 :410-33.

203. Tisue S et Wilensky U. NetLogo : A simple environment for modeling complexity. In : International Conference on Complex Systems. 2004.

204. Tsibulsky VL et Norman AB. Mathematical models of behavior of individual animals. Current Pharmaceutical Design 2007 ;13 :1571-95. issn : 1873-4286.

205. Turing A. On computable numbers, with an application to the entscheidungsproblem. J. of Math 1936 ;58 :345-63.

206. Valletta JJ, Torney C, Kings M, Thornton A et Madden J. Applications of machine learning in animal behaviour studies. Animal Behaviour 2017 ;124 :203-20. issn : 0003-3472.

207. Van Den Bergh J. Pre-Darwinism, Darwinism and Neo-Darwinism. Human Evolution beyond Biology and Culture, Evolutionary Social, Environmental and Policy Sciences 2018 :45-85.

208. Von Bertalanffy L. General System Theory. General System Theory. 1956.

209. Wang B, Hess V et Crooks A. Mesa-Geo : A GIS Extension for the Mesa Agent-Based Modeling Framework in Python. 2022. doi : 10.1145/3557989.3566157.

210. Wang GG, Deb S et Coelho LdS. Elephant Herding Optimization. In : *2015 3rd International Symposium on Computational and Business Intelligence (ISCBI)*. 2015 3rd International Symposium on Computational and Business Intelligence (ISCBI). 2015 :1-5. doi : 10.1109/ISCBI.2015.8.

211. Weise T, Zapf M, Chiong R et Nebro A. Why Is Optimization Difficult ? In : *Studies in Computational Intelligence*. T. 193. Journal Abbreviation : Studies in Computational Intelligence. 2009 :1-50. isbn : 978-3-642-00266-3. doi : 10.1007/978-3-642-00267-0_1.

212. Weyns D, Van Dyke Parunak H, Michel F, Holvoet T et Ferber J. Environments for Multiagent Systems State-of-the-Art and Research Challenges. In : *Environments for Multi-Agent Systems*. Sous la dir. de Weyns D, Van Dyke Parunak H et Michel F. Lecture Notes in Computer Science. Berlin, Heidelberg : Springer, 2005 :1-47. isbn : 978-3-540-32259-7. doi : 10.1007/978-3-540-32259-7_1.

213. Wijeyakulasuriya DA, Eisenhauer EW, Shaby BA et Hanks EM. Machine learning for modeling animal movement. PLOS ONE 2020 ;15. Publisher : Public Library of Science :e0235750. ISSN : 1932-6203.

214. Wilensky U. Netlogo Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. 1999.

215. Wilson EO et Peter FM, éd. Biodiversity. Washington (DC) : National Academies Press (US), 1988. ISBN : 978-0-309-03783-9 978-0-309-03739-6.

216. Wilson SW. Knowledge Growth in an Artificial Animal. In : *Adaptive and Learning Systems : Theory and Applications*. Sous la dir. de Narendra KS. Boston, MA : Springer US, 1986 :255-64. ISBN : 978-1-4757-1895-9. DOI : 10.1007/978-1-4757-1895-9_18.

217. Wong W et Ming C. A Review on Metaheuristic Algorithms : Recent Trends, Benchmarking and Applications. Pages : 5. 2019. 1 p. DOI : 10.1109/ICSCC.2019.8843624.

218. Wooldridge M et Jennings NR. Intelligent agents : theory and practice. The Knowledge Engineering Review 1995 ;10. Publisher : Cambridge University Press :115-52. ISSN : 1469-8005, 0269-8889.

219. Yang HC et Chao A. Modeling Animals' Behavioral Response by Markov Chain Models for Capture-Recapture Experiments. Biometrics 2005 ;61. Publisher : [Wiley, International Biometric Society] :1010-7. ISSN : 0006-341X.

220. Yang XS. Metaheuristic Optimization. Scholarpedia 2011 ;6 :11472. ISSN : 1941-6016.

221. Yang XS et Koziel S. Computational Optimization : An Overview. In : *Computational Optimization, Methods and Algorithms*. Sous la dir. de Koziel S et Yang XS. Studies in Computational Intelligence. Berlin, Heidelberg : Springer, 2011 :1-11. ISBN : 978-3-642-20859-1. DOI : 10.1007/978-3-642-20859-1_1.

222. Younis A, Bakhit A, Onsa M et Hashim M. A comprehensive and critical review of bio-inspired metaheuristic frameworks for extracting parameters of solar cell single and double diode models. Energy Reports 2022 ;8 :7085-106. ISSN : 2352-4847.

223. Zang H, Zhang S et Hapeshi K. A Review of Nature-Inspired Algorithms. Journal of Bionic Engineering 2010 ;7 :S232-S237. ISSN : 2543-2141.

224. Zeigler B, Prähofer H et Kim TG. Theory of Modeling and Simulation : Integrating Discrete Event and Continuous Complex Dynamic Systems. 2000 ;2.

Table des figures

Liste des tableaux

Liste des Algorithmes

Liste des sigles et acronymes

API	*Application Programming Interface*
BDI	*Beliefs-Desire-Intentions*
CNRS	*Centre National de la Recherche Scientifique*
EOA	*External Observable Antecedent*
EOC	*External Observable Consequent*
FAP	*Fixed Action Pattern*
GPS	*Global Positioning System*
HMI	*Human Machine Interface*
IOA	*Internat Observable Antecedent*
IOC	*Internal Observable Consequent*
IRM	*Innate Releasing Mechanism*
JSON	*Java Script Object Notation*
POO	*Programmation Orientée Objet*
PyPI	*Python Package Index*
SMA	*Système Multi Agent*
UICN	*Union Internationale pour la Conservation de la Nature*
UL	*Unité de Longueur*
UML	*Unified Modeling Language*
URL	*Uniform Resource Locator*
UT	*Unité de Temps*
VPS	*Virtual Private Server*

Annexes

A1. Configuration de l'agent «ConceptualSpecie» du modèle C

L'agent est modélisé avec 5 composants *Desire*, 5 composants *Desire* et une tâche par défaut définie par un composant *Task*.

A1.1. Les composants *Perception*

— ***feeding_perception*** : Détecte les agents de type «Plant» pour le comportement d'alimentation.
— ***napping_perception*** : Détecte le moment pour faire la sieste.
— ***lifesaving_perception*** : Détecte les prédateurs pour le comportement d'évitement.
— ***sleeping_perception*** : Détecte le moment pour faire un sommeil prolongé.
— ***following_perception*** : Détecte les «ConceptualSpecie» pour la formation d'un troupeau.

TABLE 28 – Configuration des composant *Perception feeding_perception*

Attribut	**Propriétés**	
name	feeding_perception	
Sense 1	*target*	@
	boolean_expression	True
	angle	Null
	radius	Null
	strength_expression	@.*energy* < 200
Sense 2	*target*	Plant
	boolean_expression	False
	angle	360
	radius	10
	strength_expression	$.*len*
strength_expression	$2	

TABLE 29 – Configuration des composant *Perception napping_perception*

Attribut	**Propriétés**	
name	napping_perception	
Sense 1	*target*	€
	boolean_expression	True
	angle	Null
	radius	Null
	strength_expression	$(14 * 30) <= €.\text{timer} < (16 * 30)$
Sense 2	*target*	@
	boolean_expression	True
	angle	Null
	radius	Null
	strength_expression	$@.energy >= 300$
Sense 3	*target*	ConceptualSpecie
	boolean_expression	False
	angle	360
	radius	10
	strength_expression	$\$.len$
strength_expression	$3	

TABLE 30 – Configuration des composant *Perception lifesaving_perception*

Attribut	**Propriétés**	
name	lifesaving_perception	
Sense 1	*target*	Predator
	boolean_expression	False
	angle	360
	radius	5
	strength_expression	$\$.len$
strength_expression	$1	

TABLE 31 – Configuration des composant *Perception sleeping_perception*

Attribut	**Propriétés**	
name	feeding_perception	
Sense 1	*target*	€
	boolean_expression	False
	angle	Null
	radius	Null
	strength_expression	€.timer
strength_expression	$1	

TABLE 32 – Configuration des composant *Perception following_perception*

Attribut	**Propriétés**	
name	feeding_perception	
Sense 1	*target*	@
	boolean_expression	True
	angle	Null
	radius	Null
	strength_expression	@.*energy* > 300
Sense 2	*target*	ConceptualSpecie
	boolean_expression	False
	angle	360
	radius	10
	strength_expression	$.*len*
strength_expression	$2	

A1.2. Les composants *Desire*

— ***feeding_desire*** : comportement d'alimentation.
— ***follow_desire*** : comportement d'évitement.
— ***lifesaving_desire*** : Comportement de sommeil prolongé.
— ***napping_desire*** : Comportement de sieste.
— ***sleeping_desire*** : Comportement de formation d'un troupeau.

TABLE 33 – Configuration du composant *Desire* : *feeding_desire*

Attribut	**Propriétés**
name	feeding_desire
test_string	@.*energy* >= 700
activator	π.*strength*
perceptions	[feeding_perception]
weight	4
threshold	0
interruption_factor	1
priority	1
task	[feeding_desire_task]

TABLE 34 – Configuration du composant *Task* de *feeding_desire*

Attribut	**Propriétés**		
name	feeding_desire_task		
Actions	Eat	*duration_cycle*	1
		gain	100
		move_coast	0.25

TABLE 35 – Configuration du composant *Desire* : *following_desire*

Attribut	**Propriétés**
name	following_desire
test_string	@.*energy* < 300
activator	π.*strength*
perceptions	[following_perception]
weight	2
threshold	2
interruption_factor	1
priority	1
task	[following_desire_task]

TABLE 36 – Configuration du composant *Task* de *following_desire*

Attribut	**Propriétés**		
name	following_desire_task		
Actions	Follow	*duration_cycle*	1
		spacing	2
		move_coast	1

TABLE 37 – Configuration du composant *Desire* : *lifesaving_desire*

Attribut	**Propriétés**
name	lifesaving_desire
test_string	$@.energy < 300$
activator	$dist\_away\_from\_agent(\pi.closer\_agent) > 10$
perceptions	[lifesaving_perception]
weight	5
threshold	0
interruption_factor	1
priority	1
task	[lifesaving_desire_task]

TABLE 38 – Configuration du composant *Task* de *lifesaving_desire*

Attribut	**Propriétés**		
name	lifesaving_desire_task		
Actions	RunAway	*duration_cycle*	1
		move_coast	1.5

TABLE 39 – Configuration du composant *Desire* : *napping_desire*

Attribut	**Propriétés**
name	napping_desire
test_string	€.timer >= (480)
activator	$\pi.strength$
perceptions	[napping_perception]
weight	1
threshold	1
interruption_factor	1
priority	1
task	[napping_desire_task]

TABLE 40 – Configuration du composant *Task* de *napping_desire*

Attribut	**Propriétés**		
name	napping_desire_task		
Actions	Sleep	*duration_cycle*	60
		gain	250

TABLE 41 – Configuration du composant *Desire* : *sleeping_desire*

Attribut	**Propriétés**
name	sleeping_desire
test_string	$120 <= \pi.strength < 660$
activator	$0 <= \pi.strength < 120$ or $660 <= \pi.strength < 720$
perceptions	[sleeping_perception]
weight	1
threshold	0
interruption_factor	1
priority	1
task	[sleeping_desire_task]

TABLE 42 – Configuration du composant *Task* de *sleeping_desire*

Attribut	**Propriétés**		
name	sleeping_desire_task		
Actions	Sleep	*duration_cycle*	60
		gain	500

A1.3. Le composant *Task* de la tâche par défaut

TABLE 43 – Configuration du composant *Task* de la tâche par défaut

Attribut	**Propriétés**		
name	DefaultTask		
Actions	RandomMove	*duration_cycle*	60
		max_velocity	(3, 3)
		move_probability	0.5
		lost	1
		max_attempt	4

A2. Figures supplémentaires

time	data		
	Type	parameters	values
18	*plant*	*alive*	99
		energy	9900
		positions	...
	herbivorous	*alive*	50
		energy	5189.660156531173
		positions	...
	carnivorous	*alive*	30
		energy	4619.490535124818
		positions	...
19	*plant*	*alive*	98
		energy	9800
		positions	...
	herbivorous	*alive*	50
		energy	5029.266552548883
		positions	...
	carnivorous	*alive*	30
		energy	4556.236483216594
		positions	...
20	*plant*	*alive*	96
		energy	9600
		positions	...
	herbivorous	*alive*	50
		energy	4894.542324380653
		positions	...
	carnivorous	*alive*	30
		energy	4463.156185100751
		positions	...

FIGURE 40 – Exemples de données collectées pour les modèles expérimenteaux

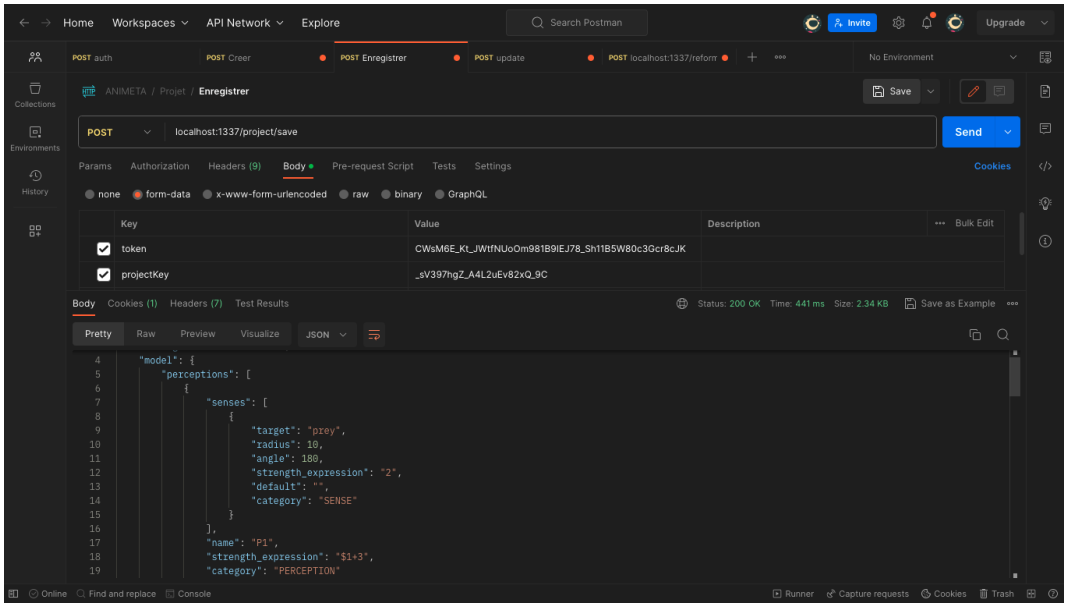


FIGURE 41 – Exemple de requête de création de modèle via ANIMETA-API

WWW.UNIVERSITA.CORSICA
10 novembre 2024