DOCTORAT BRETAGNE LOIRE / MATHSTIC

Université de Bretagne Occidentale

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE BRETAGNE OCCIDENTALE

ÉCOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par
Thomas ALVES

Visualisation et Interactions avec une Colonie d'Abeilles Virtuelle

Simulation, complexité et pédagogie

Thèse présentée et soutenue à Brest, le « date »
Unité de recherche : Lab-STICC CNRS UMR 6285
Thèse N° : « si pertinent »

Rapporteurs avant soutenance :

Prénom NOM Fonction et établissement d'exercice
Prénom NOM Fonction et établissement d'exercice
Prénom NOM Fonction et établissement d'exercice

Composition du Jury :

Attention, en cas d'absence d'un des membres du Jury le jour de la soutenance, la composition du jury doit être revue pour s'assurer qu'elle est conforme et devra être répercutée sur la couverture de thèse

Président :	Prénom NOM	Fonction et établissement d'exercice *(à préciser après la soutenance)*
Examinateurs :	Prénom NOM	Fonction et établissement d'exercice
	Prénom NOM	Fonction et établissement d'exercice
	Prénom NOM	Fonction et établissement d'exercice
	Prénom NOM	Fonction et établissement d'exercice
Dir. de thèse :	Vincent RODIN	Professeur d'Université, Université de Bretagne Occidentale, Brest
Co-dir. de thèse :	Thierry DUVAL	Professeur, IMT Atlantique, Plouzané
Encadrant :	Jérémy RIVIERE	Maître de conférences, Université de Bretagne Occidentale, Brest

Invité(s) :

Prénom NOM Fonction et établissement d'exercice

Acknowledgement

Je tiens à remercier
I would like to thank. my parents..
J'adresse également toute ma reconnaissance à ....
....

Table des Matières

TABLE DES FIGURES

INTRODUCTION

Orga Abeille - Visu Interagir avec - Simuler - Modeliser - (pk existant pas bon)proposition Modele motivation (orga++) - Eval Ccl (motivation mieux) - Enfin intéragir et visu

— Colonie d'abeille en tant que système complexe. Beaucoup de recherche sur les butineuses mais moins sur l'intérieur de la ruche.
— Multi agent car concentration sur les individus et leurs interactions
— Comprendre l'auto organisation interne par la modélisation SMA vs Équation différentielles - de l'importance des contacts individuels.
— Transmettre et faciliter l'apprentissage avec l'environnement immersif.

Chapitre 1

État de l'Art : Insectes Sociaux et Simulation Multi-Agents

1.1 Complexité et Colonie d'abeilles, Biologie et Système complexe

Synthèse des connaissances, notamment de notre visite à Avignon et ses 43 degrés à l'ombre.

1.1.1 Complexité

1.1.2 Auto-Organisation de la Colonie

Afin d'assurer le bon fonctionnement et l'épanouissement de la colonie, chaque individu la composant doit effectuer un certain nombre de tâches lorsqu'elle sont nécessaires. Contrairement à nous, ces individus ne disposent d'aucun contrôle central. Personne pour surveiller la température et prévenir une équipe que pendant un certain temps elle serait en charge de la réguler. Pas de chef pour donner des ordres.

Ainsi, c'est chaque individu qui doit en quelque sorte, tout surveiller, et être prêt a adapter son activité en fonction de ses perceptions. Pour reprendre l'exemple de la température, lorsqu'un individu décide qu'il fait un peu trop chaud, il va alors commencer à refroidir ses alentours. Dans le cas des abeilles ce rafraichissement peut être réalisé soit en allant battre des ailes à l'entrée de la ruche afin de créer un courant d'air, soit en vaporisant de l'eau à l'intérieur, pour directement abaisser la température.

Différentes Régulations

— Thermo-Régulation basée sur la perception locale de température et sur les différences entre individus.
— Sélection des meilleures sources de nourritures par les butineuses lors du recrutement.
— Régulation de l'âge du premier butinage en fonction des demandes du couvain.

Un des points clés de cette organisation est que chaque individu a des "tolérances" légèrement différentes. Une diversité qui provient notamment du fait que même si les abeilles ont en général toutes la même mère, elles n'ont pas toute le même père, on parle de "demie sœurs". En effet, lors de son unique vol nuptial, la future reine s'accouple avec des dizaines de mâles (qui meurent et tombent au sol juste après). Cette grande diversité génétique et les seuils de tolérances différents apportent une plus grande stabilité, tant dans les activités des abeilles que dans les paramètres contrôlés. Il a en effet été montré que l'exceptionnelle capacité des abeilles à maintenir la température à 36°C au niveau du couvain [1] est en partie due au fait que certaines abeilles vont commencer à refroidir la ruche assez tôt, alors que d'autres ne le feront que lorsque la température est déjà trop élevée. A l'extrême, certaine abeilles refroidissent la ruche en même temps que d'autres plus frileuses la réchauffent. Cet affrontement qui peut paraitre contre productif permet en réalité à la température d'être extrêmement stable, malgré les variations parfois conséquente de la température extérieure.

Sélection des meilleures sources de nectar

Lors de la collecte du nectar, les butineuses sont très sélectives et préfèrent de très loin les hautes teneurs en sucres. La colonie utilise donc un système de recrutement, presque de recommandation, afin d'allouer ses effectifs de butineuses de manière optimale et dynamique, préférant les sources proches et sucrées aux sources éloignées et peu sucrées.

Ce mécanisme de recrutement se compose de trois étapes. Certaines butineuses plus téméraires vont spontanément quitter la ruche sans destination, dans le seul but de trouver une nouvelle source de nectar. Ces butineuses sont appelées des "scouts", ou des "éclaireurs". Une fois sur une source de nectar, il est temps d'en juger la qualité, c'est la deuxième étape. L'abeille va récolter le nectar, et estimer sa teneur en sucre. Si cette

1. très sensible, une variation d'un degré peut condamner le couvain et mettre la colonie en péril

teneur lui convient, elle va rentrer à la colonie et commencer la troisième étape, le point clé, le recrutement.

Une fois rentrée, l'abeille recruteuse va se mettre à danser la désormais célèbre "Waggle Dance", en forme de 8. Cette danse à un objectif double. Le premier : communiquer la position de la source par rapport au soleil, pour permettre aux autres de la retrouver. Le second est plus indirect : Plus la source est sucrée, plus l'abeille va danser longtemps, et même répéter la danse dans plusieurs endroits de la ruche. Une danse plus longue offre plus de temps à d'autres abeilles de venir la suivre et apprendre la position de cette nouvelle source. Ainsi, plus la source est sucrée, plus la recruteuse va communiquer la position à de nombreuses butineuses.

Une fois recrutée, les nouvelles butineuses vont se rendre a la source et répéter le processus : Collecter, juger, rentrer et recruter. La encore, la diversité est clé, des abeilles moins portées sur la communication vont passer moins de temps à recruter, afin de maximiser le temps de butinage, il y a tout de même des dizaines de milliers de bouches à nourrir ! De plus, certaines butineuses sont moins difficiles que d'autres, elles vont donc communiquer des sources de faible qualité et y maintenir un faible contingent. Ce contingent alors moins utile dans l'immédiat sert de surveillance, car les teneurs en sucres des différents nectars peuvent fortement varier selon les saisons mais aussi pendant les périodes de la journée. Une source de faible qualité peut alors devenir une source extrêmement intéressante en quelques heures. Le groupe alors déjà présent peut observer ce changement et déjà danser dans toute la colonie pour avertir les autres, gagnant ainsi un temps précieux de re-découverte de la source mais aussi le temps d'amorcer une réponse conséquente. Gagner du temps sur une réaction exponentielle est toujours extrêmement précieux.

1.1.3 Pheromones et Physiologie

Afin d'obtenir cette auto-organisation, la colonie s'appuie sur différents mécanismes incluant de nombreuses boucles de rétro-actions. Des perceptions directes des différents individus permettent d'effectuer une partie de cette organisation, comme la température ou la concentration en sucre de leur nourriture. Mais la colonie s'appuie aussi sur des mécanismes indirects, physiologiques, qui prennent place grâce à différentes hormones et phéromones. Nous étudions ici en détail l'importance physiologique des glandes hypopharyngiennes (GH) et de la Corpora Allata. Voici un modèle simplifié des connaissances biologiques que nous avons à l'heure actuelle.

Les GH permettent aux abeilles s'occupant du couvain de transformer le pollen et le

nectar en une substance riche destinée aux larves. Elles permettent aussi aux butineuses de traiter chimiquement le nectar, le rendant utilisable pour les nourrices, transformable en miel et même consommable directement par les autres adultes. Or, ces deux comportements sont incompatibles, les GH subissent une modification physiologique pour effectuer l'une ou l'autre de ces fonctions.

La Corpora Allata permet aux adultes de sécréter une hormone appelée Hormone Juvénile(HJ). Cette hormone est retrouvée en grande quantité chez les butineuses, et en faible quantité chez les nourrices. Une hypothèse répandue est de considérer que la HJ sécrétée par la Corpora Allata permet d'altérer le fonctionnement des GH, dictant ainsi leur utilité pour les nourrices ou les butineuses. Typiquement, la transition de nourrice à butineuse se fait en une vingtaine de jours : La colonie suit ce qu'on appelle le Polyéthisme d'Âge, les adultes ayant le même âge réalisent les mêmes activités. Or, il a été montré que ce polyéthisme est souple, et que dans les bonnes conditions, une abeille peut aller butiner dès ses 5 jours, au lieu de la vingtaine habituelle.

Ce mécanisme a été lié à une phéromone, émise par toute la colonie, l'Ethyle Oléate (EO). Retrouvée majoritairement sur le couvain et la reine, elle est aussi retrouvée chez les butineuses. Lorsque cette phéromone est injectée en grande quantité à des abeilles adultes, il a été montré que celles-ci arrêtent le butinage et voient leur taux d'HJ diminuer.

L'Ethyle Oléate n'est pas une phéromone volatile, elle est majoritairement transmise par contact, principalement lors d'échange de nourriture et de nettoyage mutuel : lorsqu'une abeille en nettoie une autre, ou lorsqu'une nourrice nettoie une larve. Elle serait aussi transmissible sur de courtes distances par évaporation, mais nous avons décidé d'ignorer ce mécanisme pour l'instant.

Dans le cas classique du polyéthisme d'âge, les jeunes abeilles sont nourrices, et les plus âgées sont butineuses. Mais, comme nous venons de le voir, cet âge est souple : Une nourrice peut accélérer sont vieillissement, et une butineuse peut même l'inverser. C'est pour ceci que nous parlerons ici d'âge physiologique, opposé à l'âge réel. Une abeille avec un faible âge physiologique possède les GH nécessaires aux nourrices, et les âgées physiologiques possèdent les GH et les muscles nécessaires au vol et au butinage.

Mécanismes de l'auto organisation Rôles et fonctions physiologiques associées.
Différents rôles de différentes phéromones.

Modélisation estimée hypothèse :
Cas classique de la vie d'une abeille.
Quels changements provoquent quelles réactions dans le métabolisme de l'abeille, et donc dans la répartition du travail.

1.2 Modèles existants de répartition des tâches

Etat de l'art de l'article PAAMS

1.2.1 Foraging For Work

1.2.2 Modèles à Seuils

Ces fonctions sigmoïdes prennent le plus souvent la forme :

$$Score = \frac{x^n}{x^n + \theta^n} \tag{1.1}$$

Avec x le stimulus d'entrée, n le degré de linéarité, placé a $n = 2$ dans la littérature, et θ le seuil, aussi apellé biais, de la sigmoïde. Vous trouverez Figure 1.1 différentes sigmoides mettant en valeur l'impact du paramètre θ. Le seuil sert en quelque sorte de point d'ancrage, lorsque le seuil est strictement équivalent au stimulus d'entrée, alors la valeur du résultat est exactement 0,5.

Conclusion

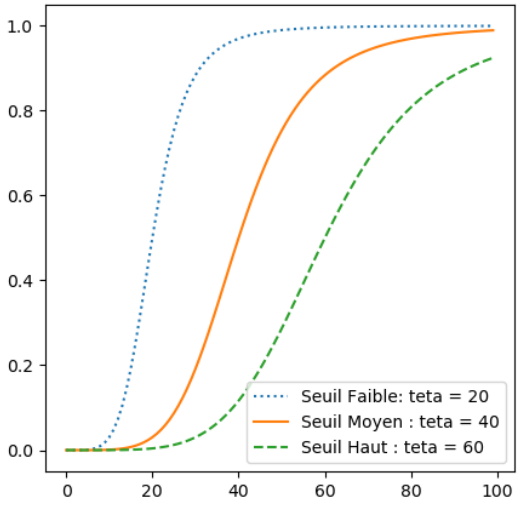


FIGURE 1.1 – L'influence du paramètre téta (θ) sur la forme des sigmoides.

Chapitre 2

Proposition : Prise de Décision et Interruption

Dans ce chapitre et à l'aide de ce que nous venons d'apprendre au sujet de la répartition des tâches, nous allons pouvoir construire notre propre mécanisme générique, que nous utilisons ensuite pour notre cas d'application : la colonie d'abeilles (Chapitre 3). Nous allons voir comment modéliser nos tâches et comment les exécuter. Nous verrons ensuite les mécanismes de sélection, puis d'interruption que nous avons mis en place afin que nos agents effectuent toujours une tâche qui a du sens par rapport à l'état actuel de l'environnement et de l'activité des autres agents. Nous terminerons ce chapitre par un exemple possible d'application de ce modèle dans le cadre de la robotique en essaim, où une population de robot devra se partager deux tâches : collecte de ressources et patrouille.

2.1 Modélisation des tâches : Exécution du comportement

2.1.1 Actions, Activités et Tâches

Afin de modéliser nos tâches, nous allons utiliser 3 concepts : Actions, Activités et Tâches.

Une Action est définie comme une interaction avec l'environnement extérieur, non interruptible et d'une durée déterminée et courte (pas plus de quelques pas de temps, pour ne pas bloquer l'agent). Elle n'est donc pas forcément élémentaire, mais doit s'en approcher. Chaque Action possède une condition d'activation.

Ensuite, une Activité est un ensemble d'Actions et/ou d'autres Activités. Une Activité possède aussi sa propre condition d'activation. Indirectement, tout ce qu'elle contient partage alors sa condition d'activation, ce qui nous permet de factoriser cette condition et d'alléger notre écriture, et ainsi de modéliser des comportements complexes.

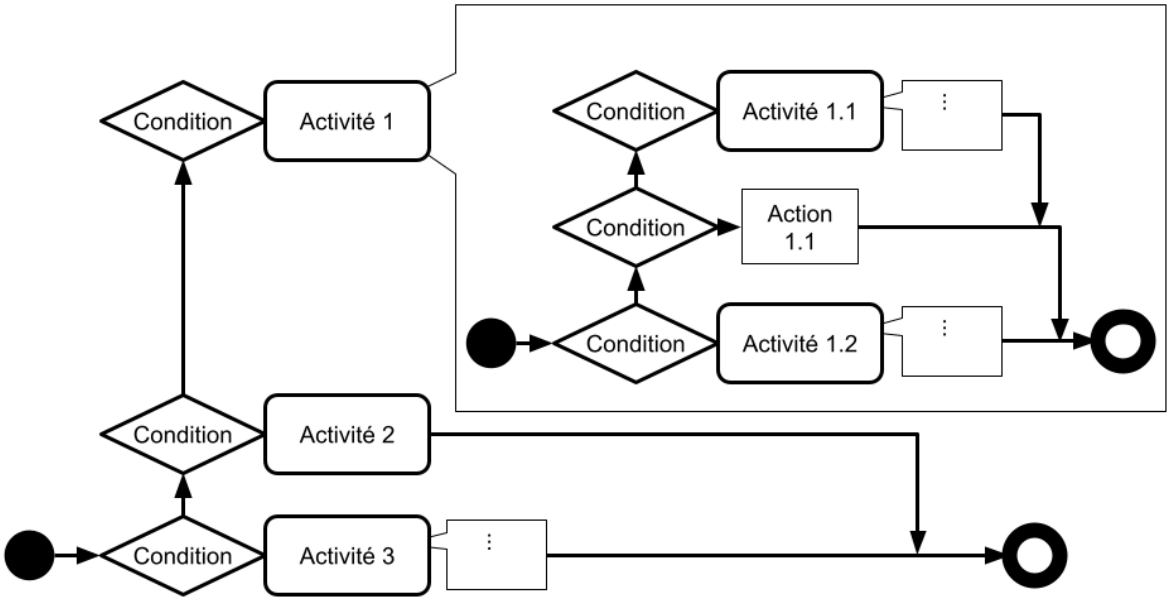


FIGURE 2.1 – Modélisation d'une tâche à l'aide d'une subsomption hiérarchique.

Pour finir, une Tâche est l'ensemble des Activités et Actions. On peut donc voir une Tâche comme l'Activité racine, un peu à la manière d'un système de fichiers : les Activités sont des dossiers et contiennent d'autres dossiers et/ou des fichiers, que sont les Actions.

2.1.2 Subsomption Hiérarchique et Exécution

Une architecture de subsomption permet de hiérarchiser différents comportements entre eux, afin d'obtenir un comportement général cohérent[3]. Dans un ordre défini, la subsomption interroge tour à tour la condition d'activation de chacun de ses différents blocs comportements, et exécute le premier dont la condition est valide. Par exemple, modéliser le comportement d'un mouton peut se faire en trois blocs. Un premier bloc "Chercher à manger", toujours valide. Au dessus de celui-ci, donc avec une priorité plus importante, nous plaçons un autre bloc : "Brouter". Ce-dernier s'active lorsque le mouton a trouvé de quoi manger. Enfin, nous plaçons au sommet le bloc "Fuir", s'activant dès que le mouton perçoit un prédateur, et reste activé le temps de le semer. Ainsi, tant qu'aucun grand méchant loup n'est en vue, le mouton va brouter paisiblement. Dès qu'il en verra un, alors il pourra fuir.

Afin de respecter l'aspect quasi-élémentaire des comportements, le bloc "Fuir" sera

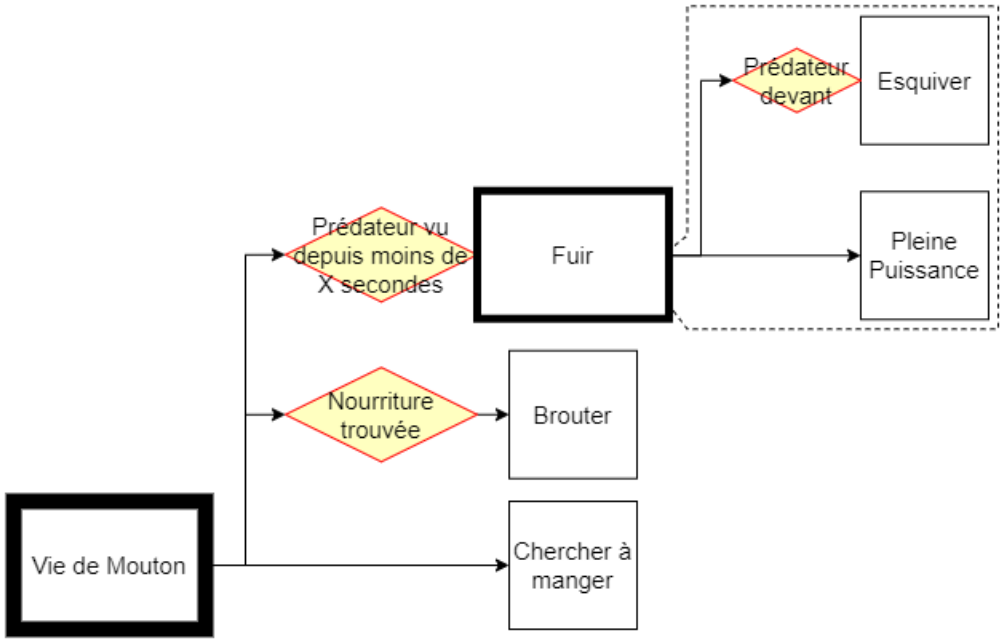


FIGURE 2.2 – Modélisation de la tâche "Vie de Mouton" contenant tout le comportement du mouton de notre exemple.

réalisé une multitude de fois. Ainsi, une seule exécution de Fuir ne fera faire au mouton qu'un pas l'éloignant du prédateur. Il va y faire appel plusieurs fois avant de considérer avoir semé le loup, tant que la condition du bloc "Fuir" sera valide.

Une subsomption hiérarchique ajoute à cette structure simple, le fait que chaque bloc comportement puisse être une autre architecture de subsomption[7]. Cette légère modification apporte une grande modularité dans la conception de ces architectures, et permet de modéliser des comportements plus complexes sans la lourdeur des subsomption classiques.

Ce que nous avons appelé "bloc comportement" des subsomptions correspond à nos actions et activités. Les blocs qui contiennent une autre subsomption sont appelés activités, et ceux qui contiennent du comportement sont des actions. Ensuite, la subsomption en elle même est alors une tâche. La Figure 2.1 présente la notion de subsomption hiérarchique contenant nos concepts définis plus tôt.

Ainsi, pour qu'un agent puisse exécuter une tâche, il interroge l'activité racine puis va récursivement interroger ses composants. Chaque activité ou action interrogée va ainsi vérifier sa condition d'activation. Une activité dont l'activation est valide va alors continuer d'interroger ses composantes. On a donc une recherche en profondeur, par ordre de prio-

rité, qui s'arrête à la première action interrogée avec une condition d'activation valide. Cette action est alors remontée à l'agent, qui pourra alors l'exécuter pendant toute sa durée. Ensuite, une fois l'action terminée, tout ce processus recommence afin de pouvoir déterminer une nouvelle action à exécuter.

Pour reprendre l'exemple du mouton, nous pouvons complexifier son comportement en transformant son Action "Fuir" en une Activité "Fuir" contenant deux Actions. L'Action prioritaire "Esquiver" a pour condition le fait de voir le loup droit devant, et consiste à fuir mais en tournant, afin d'éviter le loup. Ensuite, la deuxième action, "Pleine Puissance" est l'action par défaut, sans condition, qui consiste à courir tout droit tant que ça ne fait pas suffisamment de temps que le loup n'a pas été vu.

La condition d'activation de l'activité "Fuir" définie plus tôt, le fait d'avoir vu un prédateur dans les dernières secondes/minutes, est alors nécessaire à l'activation des deux actions "Esquiver" et "Pleine Puissance" sans que nous ayons à les réécrire explicitement. La Figure 2.2 reprends la structure de la tâche du mouton que nous décrivons dans cette section.

2.2 Motivation : État de l'art

Avant de continuer à construire notre modèle, nous avons besoin d'un mécanisme nous permettant d'améliorer les modèles à seuils dont nous avons discuté dans l'état de l'art. En effet, ceux-ci ne permettent pas de modéliser des tâches n'ayant pas de stimulus déclencheur : par exemple, la faim est un stimulus déclencheur de l'action de manger. En revanche, le stimulus déclencheur de rédiger un état de l'art, ou de ranger sa chambre, est beaucoup plus complexe. Nous proposons donc une solution simple, permettant de prendre en compte ces tâches sans stimulus tout en utilisant un modèle à seuils : en utilisant la motivation interne de l'agent. Avant d'aller plus loin dans notre utilisation de la motivation, voici un rapide état de l'art sur son utilité, et notre positionnement par rapport à celui-ci.

Pour les psychologues, la motivation est la source de l'action et guide son exécution. Deux types de motivations existent : extrinsèque (ou externe), lorsqu'une récompense est offerte par le monde extérieur, et intrinsèque (ou interne), qui n'a à voir qu'avec les croyances ou besoins de l'agent, comme l'amusement ou la curiosité. La théorie du Flow[5],

de son côté, dit que la motivation interne d'un individu est maximale lorsque la difficulté rencontrée lors de la réalisation d'une tâche est suffisante pour susciter l'intérêt (qu'elle n'est pas ennuyeuse) mais suffisamment faible pour ne pas être décourageante (qu'elle n'est pas impossible à réaliser pour l'agent).

On note dès lors que la motivation interne présente dans la littérature peut se diviser en deux catégories : d'un côté la motivation source, comme la faim, qui va provoquer un comportement, et de l'autre côté la motivation guide, motivation au sens d'implication, d'intérêt, qui elle sera plutôt un guide de cette action, comme la curiosité. On retrouve ces notions en éthologie, par exemple chez Lorenz[8], pour qui la motivation interne (guide), couplée à un stimulus (source), va déclencher et entretenir un comportement.

En intelligence artificielle, la motivation intrinsèque *source* est particulièrement utilisée pour les systèmes d'apprentissage [11], *e.g.* pour aider ou guider des agents apprenants [2], notamment en apportant la notion de curiosité à des agents informatiques. Certains travaux [6, 9] s'attachent à sa définition proche de l'éthologie, dans laquelle la motivation intrinsèque peut venir de nombreux stimulus internes différents et plus primitifs, tels que la faim ou la peur.

D'autres travaux se basent sur la théorie du Flow : Un agent qui ne parvient pas à réaliser sa tâche ressent de l'anxiété et cherche une tâche moins difficile [4]. De la même manière, un agent qui accomplit une tâche facile s'ennuie et passe à des tâches plus difficiles. L'idée de compétence apportée par Roohi et al.[10] rejoint ces notions : la compétence est, pour un agent, le sentiment d'être en contrôle et capable d'accomplir sa tâche actuelle. Ainsi, un agent ayant un niveau de compétence qu'il juge trop faible pour la tâche actuelle cherchera une tâche plus facile, plus adaptée.

Nous distinguons donc bien deux catégories dans la motivation interne. La première, que nous allons continuer à appeler la motivation source, proche de l'éthologie, décrit une motivation comme un stimulus interne servant à déclencher des comportements. Pour reprendre l'exemple de notre mouton, si voir un loup ne déclenche pas de réaction physique directe (au final ce ne sont que des pixels plus sombres sur le fond de sa rétine), son cerveau va pourtant reconnaitre le loup et faire augmenter la soudaine motivation de prendre ses jambes à son coup. On peut le voir comme si c'était la peur qui déclenchait la fuite.

La deuxième, la motivation guide, proche de l'idée du *Flow*, permet à l'agent de se situer par rapport à la difficulté de la tâche qu'il exécute, afin de maximiser son appren-

tissage, et donc d'optimiser l'usage de son temps. L'exemple de notre mouton sera ici un peu capilo-tracté, mais nous nous y tiendrons : nous souhaitons apprendre à ce mouton comment résoudre un Rubik's cube. Si le casse-tête lui est donné sans introduction, la tâche est insurmontable, décourageante. Le mouton n'apprend rien, il perd son temps et va donc naturellement se déconcentrer et essayer autre chose (peut être essayera-t-il d'apprendre à jongler avec plusieurs cubes ?). En revanche, si la courbe de difficulté est adaptée, en exercices simples et incrémentaux, le mouton apprendra bien mieux et restera concentré : la difficulté sera alors toujours adaptée à ses compétences.

Avec ces deux notions en tête, motivation source et motivation guide, nous pouvons passer à la suite de la description de notre modèle, dont la sélection et l'interruption des tâches se feront grâce à ces motivations.

2.3 Sélection : Modèle à Seuil

Maintenant que nous avons modélisé le fonctionnement interne de nos tâches, nous allons pouvoir construire le mécanisme permettant à nos agents de sélectionner la plus prioritaire.

Pour cette sélection nous allons utiliser un modèle à seuil, que nous allons légèrement adapter en lui ajoutant un mécanisme d'interruption, décrit dans la section 2.4. Dans un modèle a seuil, chaque tâche possède une fonction lui permettant de calculer son score. Un agent peut ainsi sélectionner la tâche qui possède le plus élevé. Très souvent lié à un stimulus déclencheur, le score de la tâche est calculé à l'aide d'une fonction sigmoïde, paramétrée par un seuil, qui prend en entrée le stimulus (ou une combinaison linéaire de plusieurs stimulus), et nous donne en résultat le score de la tâche, comme nous avons pu le constater dans l'état de l'art, sous-section 1.2.2.

Même si plusieurs agents sont en mesure d'effectuer la même tâche, les seuils de ses tâches lui sont propres. Chaque agent possède une instance différente des tâches, et chaque instance de tâche possède son propre seuil. En modélisant nos tâches, il arrive que certaines n'aient pas de stimulus déclencheur. C'est le cas pour les abeilles butineuses : le butinage de nectar semble être leur comportement par défaut, aucun stimulus global n'a encore été identifié. Dans ce cas, nous appliquons la notion de motivation interne *source*, que nous avons décrit juste avant, pour créer un stimulus artificiel, qui sera inséré dans la sigmoïde afin de calculer le score de la tâche. Ainsi, toutes nos tâches, stimulus artificiel ou non,

sont capables de produire un score cohérent. Nous pouvons alors dire d'une Tâche qu'elle est "motivée" lorsqu'elle utilise un stimulus déclencheur artificiel.

Nous avons désormais à notre disposition deux mécanismes influant sur le score d'une tâche : nous pouvons jouer sur sa motivation source, et modifier son seuil. Afin de garder une approche cohérente, nous proposons de modifier le seuil pour représenter l'état interne de l'agent (caractéristiques physiques, physiologiques, etc.). De la même manière, modifier la motivation source permet de représenter un changement de perception, ou une décision de la part d'un agent.

Afin de toujours réaliser une tâche utile à la communauté, nos agents doivent très régulièrement interroger leurs perceptions, afin de se "mettre à jour". C'est pour ceci que nous avons introduis la notion d'évaluation systématique : un agent réévalue l'ensemble de ses tâches à chaque fois qu'il termine une action. C'est également pour cette raison que les actions doivent avoir une durée courte, de préférence atomique, pour permettre la mise à jour. Ce rafraichissement des perceptions est essentiel pour que le système puisse réagir face à une urgence. Même si notre mouton est paisiblement en train de brouter, il est intuitif de l'autoriser à fuir à la vue d'un loup avant d'avoir parfaitement terminé de brouter. Il doit aussi pouvoir faire une pause lors de la résolution d'un casse tête afin de se nourrir, pour ne pas mourir de faim.

2.4 Interruption : Motivation et Flow

2.4.1 Action Démotivante et Tâche Motivée

La réévaluation systématique nous autorise à ne pas avoir de mécanisme d'interruption : dans le cas général, la fluctuation des stimulus internes et externes suffit à l'agent pour effectuer la bonne tâche. En revanche, la question se pose lorsqu'une tâche possède un stimulus déclencheur artificiel, lié à une motivation source, fixe. Pour décider quand arrêter une telle tâche, nous avons besoin de ce mécanisme d'interruption. Dans ce cas, nos agents vont essayer d'estimer leur apport à la communauté dans leur tâche actuelle. Ainsi, lorsqu'un agent verra qu'il n'arrive pas à réaliser sa tâche, il sera dans l'état de malaise décrit pas le *Flow* et cherchera de plus en plus à changer de tâche. Nous cherchons alors à mesurer l'efficacité de chaque agent dans sa tâche, afin de pouvoir détecter lorsqu'il arrive à la réaliser, mais surtout lorsqu'il n'y arrive pas. Il suffit alors de déterminer, dans

le contenu de la tâche, quelles sont les Actions effectuées représentatives de son échec. Par exemple, un robot ayant pour tâche de récolter des ressources aura dans cet algorithme une séquence de déplacement aléatoire lorsqu'aucune ressource n'est en vue. Répéter en boucle ce déplacement aléatoire, cette Action, est un signe que ce robot n'arrive pas a correctement réaliser sa tâche, il devrait donc essayer de faire autre chose.

Nous proposons alors d'ajouter aux Actions définies plus tôt le fait de pouvoir être "Démotivantes" (notées **M-** dans nos schémas). Lors de l'exécution d'une Action "Démotivante", un agent va baisser sa motivation interne d'un montant défini. Le but est d'augmenter les chances qu'il abandonne cette tâche au profit d'une autre, lors du processus de sélection. L'Action de déplacement aléatoire du robot que nous venons de citer est un bon exemple d'Action démotivante. Une action démotivante doit forcément être dans une tâche motivée : si la motivation interne ne joue aucun rôle dans la sélection de la tâche, il n'y a aucun intérêt à la diminuer.

Le fait de placer une motivation source artificielle brise une des hypothèses de base des modèles à seuil : exécuter une tâche doit faire baisser son stimulus déclencheur. Or ici, ce stimulus déclencheur étant fixe, il n'est pas du tout corrélé à la quantité de fois que son action liée est réalisée. Afin d'intégrer ces motivations au modèle à seuils que nous avons présenté plus tôt, nous modifions légèrement le calcul des scores des tâches motivées : le score d'une tâche motivée est remplacée par la motivation interne de l'agent, si c'est cette tâche que l'agent exécutait au pas de temps précédent. Ainsi, une tâche motivée est bien sélectionnée grâce à son score classique, mais est interrompue par une valeur de motivation plus basse, qui aura tendance à favoriser d'autres tâches. Lorsqu'une nouvelle tâche motivée est sélectionnée (et qu'elle n'était pas la tâche sélectionnée au pas de temps précédent), la motivation interne de l'agent est remontée à sa valeur maximale.

2.5 Définir un Agent

À l'aide de ces définitions nous pouvons désormais décrire nos agents. Un agent est situé dans l'environnement, possède une série de senseurs internes et externes (comme la faim et l'odorat), et contient aussi une liste de tâches qu'il sera peut-être amené à réaliser au court de sa vie. Lors d'une sélection de tâche, l'agent pourra confronter ses perceptions courantes aux différents seuils et conditions de l'algorithme de sélection de tâche, donnés par son état interne ainsi que son environnement direct, afin de savoir

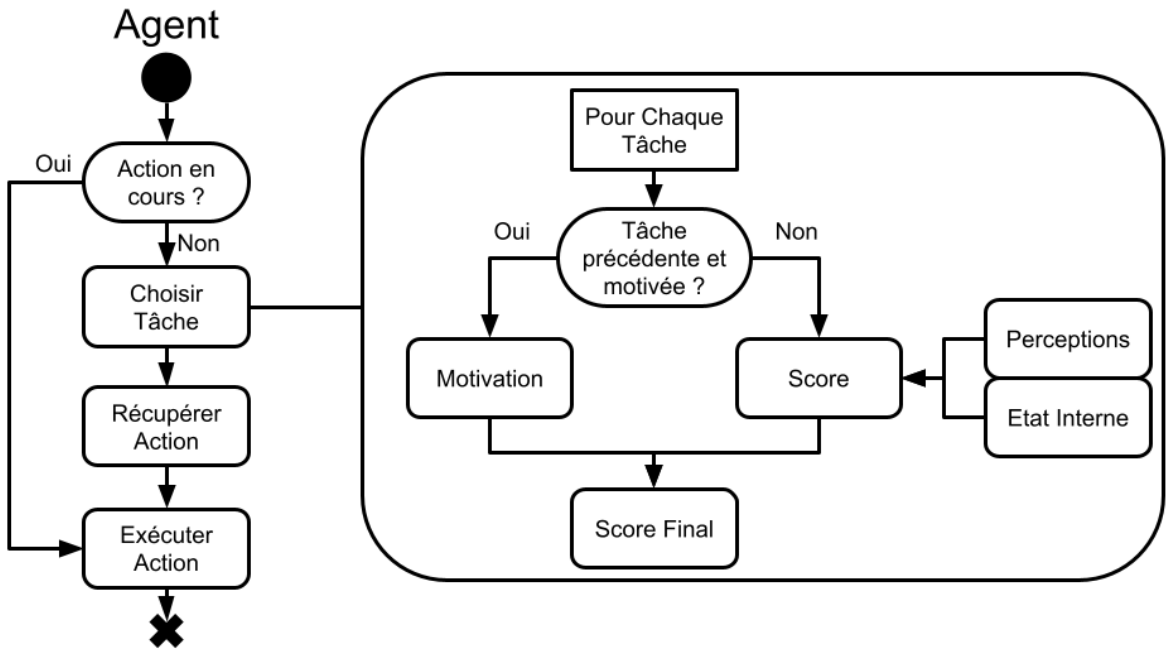


FIGURE 2.3 – Sélection et exécution des tâches par chaque Agent, à chaque pas de temps. Si l'action en cours est terminée, l'agent va sélectionner une nouvelle tâche, en extraire l'action à réaliser puis l'exécuter.

quelles actions effectuer. La Figure 2.3 reprend ces notions et décrit le comportement et la prise de décision d'un agent à chaque pas de temps : si l'agent n'a pas d'Action en cours, il sélectionne une nouvelle Tâche en fonction de son état interne, de sa motivation interne et de ses perceptions, récupère l'Action à réaliser de sa Tâche nouvellement sélectionnée grâce à son architecture de subsomption hiérarchique, puis l'exécute.

Parmi les perceptions internes de l'agent, nous trouvons une variable de motivation interne, servant aussi à la sélection de tâche. Associer une valeur de motivation à chaque tâche aurait aussi été possible et nous pouvons en trouver des équivalences dans d'autres travaux [1], nous avons donc décidé de tenter l'expérience avec une valeur transversale à toutes les tâches, liée à l'agent lui même.

Un agent peut aussi présenter des variations individuelles, un léger *offset* que nous pouvons utiliser pour ajuster légèrement les seuils de ses différentes tâches, en plus des conditions internes et externes, afin de créer une population d'agents plus ou moins homogène. Via ses tâches, un agent possède des seuils variables qui lui sont propres : deux agents avec les mêmes tâches présentent le plus souvent des seuils différents.

2.6 Application possible à la Robotique en Essaim

Les systèmes multi-agents sont souvents utilisés dans la mise en place d'essaims de robots, ce qu'on appelle le domaine des "Swarm Robotics". Ces essaims consistent en une grande quantité (dizaines, voire centaines) de robots simples, amenés à exécuter des tâches complexes en collectivité. L'image de la capacité de fourragement des fourmis est souvent utilisée comme cas d'application, nous avons donc construit notre exemple dans ce contexte. Un ensemble de robots va devoir ramener des ressources à leur base commune. Les ressources sont éparpillées dans l'environnement et doivent être traitées par un robot avant d'être déplacées jusqu'à la base. En plus de cette activité de collecte, les robots vont devoir assurer une surveillance de la base, en patrouillant autour. Ces robots possèdent une mémoire limitée : ils connaissent leur position ainsi que celle de la base, et peuvent se rappeler de la position de gisements de minerai qu'ils ont directement observés.

De plus, nous ajoutons la notion d'outil : un robot devra posséder le bon outil pour exécuter une tâche, par exemple une pioche pour collecter des ressources, et des jumelles pour patrouiller. Les ressources brutes devront être traitées sur place avant de pouvoir être collectées puis amenées à la base.

Nous pouvons dès lors commencer à construire nos tâches :

— **Patrouiller** : le robot effectue des cercles larges autour de la base, observant les alentours. Il se démotive légèrement au fil du temps (noté M- sur la Figure 2.4), et un peu plus lorsqu'il croise un autre robot (noté M– sur la Figure 2.4). Ceci permet aux robots d'éviter de patrouiller si un grand nombre de robots le fait déjà. Le seuil est élevé, sauf si l'agent est équipé des jumelles.
— **Collecter** : le robot parcours l'environnement aléatoirement à la recherche d'un gisement. Une fois trouvé, il traite le gisement pour collecter des ressources, puis les ramène à la base. Il se démotive légèrement à chaque pas de temps où il exécute un déplacement aléatoire. Le seuil est élevé, sauf si l'agent est équipé d'une pioche.
— **Recharger** : le robot se connecte à la base pour recharger ses batteries.
— **Mémoriser** : lorsque le robot voit un gisement qu'il ne connait pas, il l'ajoute à sa mémoire. A l'inverse, lorsqu'il ne voit pas de gisement (ou qu'il voit un gisement épuisé) là où il en avait retenu un, il l'oublie. Ainsi, un robot qui patrouille peut découvrir de nouveaux gisements, tout autant qu'un robot qui en cherche activement un.

Pour *Patrouiller* et *Collecter*, si l'agent active la tâche sans posséder le bon outil, la tâche commencera par le faire retourner à la base pour équiper le bon. Cet outil va alors changer les seuils de ces deux tâches, priorisant celle dont l'outil est le bon. Ainsi, un agent ne changera d'outil que lorsque cela sera nécessaire, les seuils ajoutent ici une notion de coût du changement d'outil. Ensuite, si nous le souhaitons, en ajustant les valeurs des seuils bas (prioritaires) et haut (changement d'outil nécessaire) et/ou la force répulsive des actions démotivantes, nous pouvons ajuster ce coût empiriquement. Les Figures 2.4 et 2.5 présentent respectivement nos modélisations en subsomption hiérarchique des tâches de patrouille et de collecte de ressources.

Ensuite, si *Recharger* a un stimulus déclencheur évident, le niveau courant de batterie, ce n'est pas le cas pour Patrouiller et Collecter. Nous leur appliquons donc une motivation source. Nous pouvons ensuite modifier ces motivations sources avec des perceptions de l'agent, sans qu'elles soient la majorité du stimulus déclencheur : nous pouvons réduire légèrement cette stimulation pour la tâche *Patrouiller* lorsqu'un robot avec des jumelles est dans le champs de vision. De même, nous pouvons réduire la stimulation de *Collecter* lorsqu'un robot avec une pioche est en vue, et l'augmenter lorsqu'un gisement de minerai est en vue.

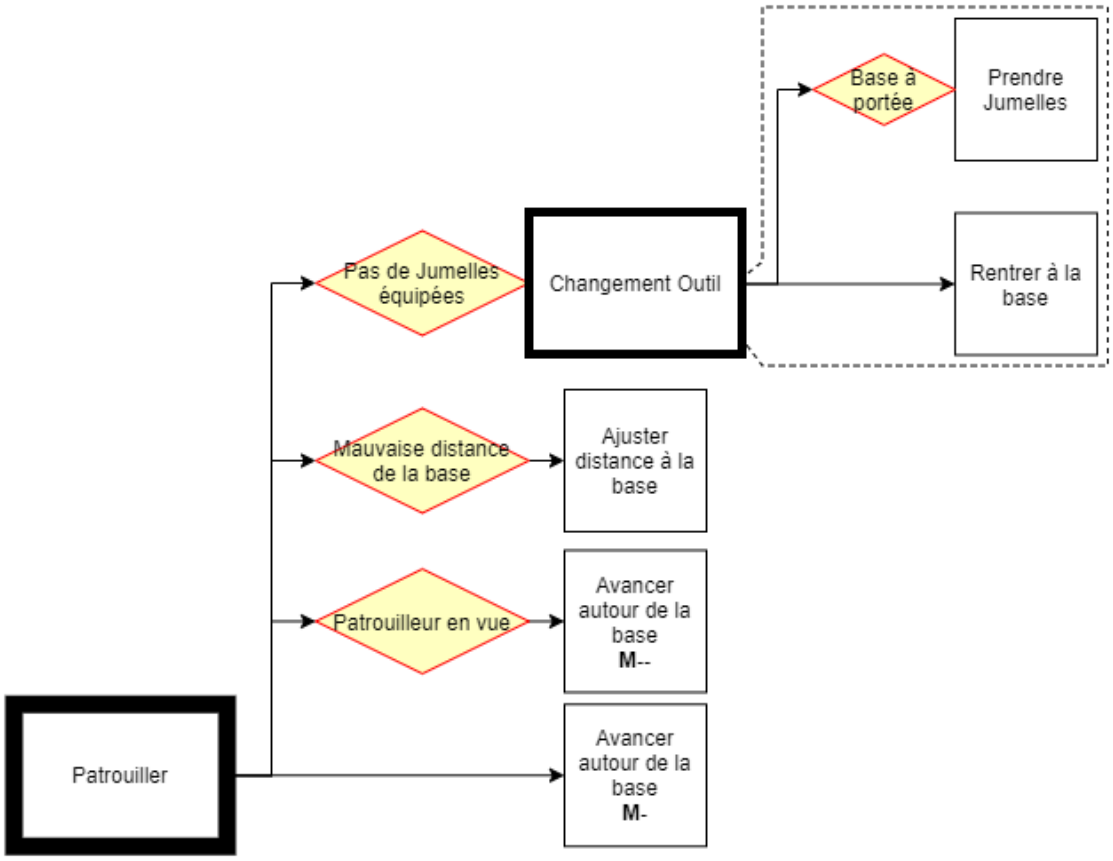


FIGURE 2.4 – Robotique en essaim : modélisation de la tâche de patrouille.

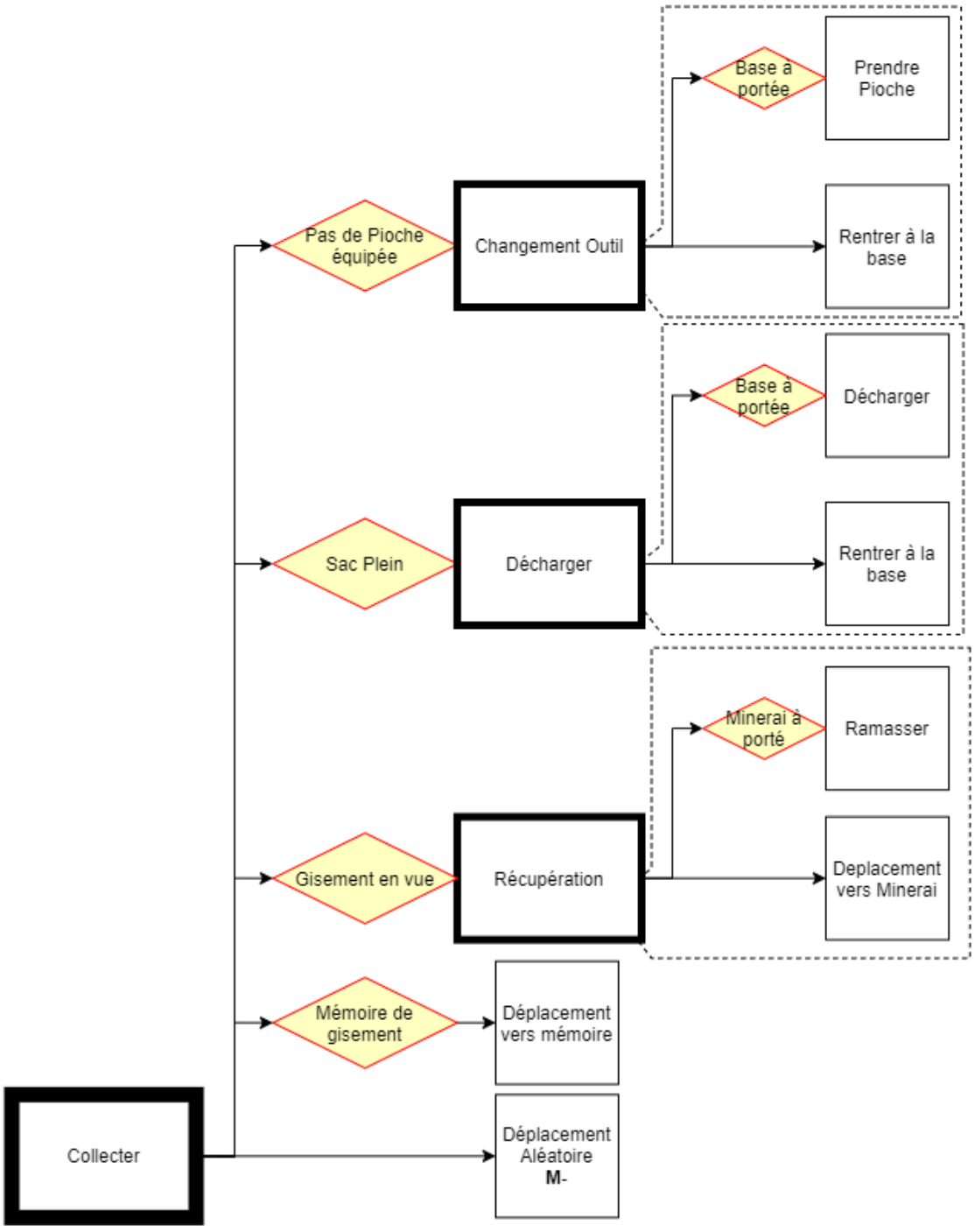


FIGURE 2.5 – Robotique en essaim : Modélisation de la tâche de collecte de ressources.

Conclusion

Dans ce chapitre nous avons décrit notre modélisation des tâches en Activités et Actions, ainsi que l'adaptation du modèle à seuils pour notre sélection de tâches. La sélection de tâche se fait via un système à seuil adapté afin de prendre en compte les motivations source (nous venant plutôt de l'éthologie) et interne (ou guide, nous venant de la théorie du *Flow*), afin d'inclure des tâches ne présentant pas de stimulus déclencheurs définis. Un score est calculé par tâche selon les perceptions et l'état interne de l'agent, et ce dernier sélectionne la tâche qui possède le plus grand score. Lorsque la tâche précédemment réalisée est une tâche motivée, le score est remplacé par la motivation guide, interne à l'agent et transversale à toutes les tâches de cet agent. Une fois la tâche sélectionnée, l'agent utilise notre architecture de tâches en subsomption hiérarchique afin d'obtenir le comportement, l'Action qu'il doit réaliser. Nous avons vu ensemble un exemple possible d'application à un essaim de robot, et nous allons désormais pouvoir aborder l'implémentation de l'application principale de ces travaux, la colonie d'abeilles.

Chapitre 3

Modélisation et Implémentation pour la Colonie d'Abeilles

Nous allons désormais voir ensemble l'implémentation du modèle que nous venons de décrire, appliqué au cas qui nous intéresse ici : La colonie d'abeilles. Pour une première itération, nos abeilles auront pour objectif de correctement se répartir le travail entre deux tâches principales : Nourrir le couvain et Butiner. Nous allons commencer par discuter de notre simulateur et de son architecture logicielle, puis du modèle physiologique de l'abeille adulte et du couvain, tiré de la biologie mais fortement simplifié. Nous verrons ensuite comment nous avons intégré le modèle de répartition des tâches décrit dans le chapitre précédent. Nous finirons par discuter de la calibration de ce système complexe, au plus proche de la biologie, tout en tenant en compte les différentes hypothèses et simplifications de notre modèle.

3.1 Description du Simulateur

3.1.1 Architecture Logicielle

Pour réaliser ce simulateur, nous avions en tête quelques points clés. Cette thèse s'inscrit dans un projet plus grand, l'idée était donc de produire un simulateur propre, qui serai simple d'entretien et facile à améliorer et complexifier par la suite. De plus, notre bonne maitrise du langage Java, ainsi qu'un compromis entre confort d'écriture et performances nous a poussé à utiliser ce langage. Pour nous aider à mettre en place cette notion floue de propreté du code, nous avons utilisé une bonne quantité d'Interfaces Java, permettant de découpler des grande parties du codes, facilitant l'implémentation de modifications. Nous avons, le plus possible, pensé l'architecture en compostants indépendants échangeant le moins d'informations possibles.

Le simulateur, en bon projet orienté objet, est pensé en couches d'abstraction succes-

sives. Au plus haut niveau, nous trouvons le lanceur, le célèbre "*main*", qui se charge des simulations. Présentant deux modes, "Interactions" et "Barrage de simulations", il prépare différent composants logiciels. Dans le mode "Interactions", il prépare le composant réseau qui servira à envoyer et recevoir les informations de l'application interactive, composant qui n'est pas instancié dans le mode "Barage de simulations". Le lanceur instancie ensuite un *MainController*, un contrôleur principal, et lui partage le composant réseau. Le contrôleur principal se charge de gérer une simulation. Désormais nous trouvons deux comportements différents pour les modes : en "Barrage de Simulations", les simulations s'enchainent à pleine vitesse, souvent en variant quelques paramètres. En mode "Interactions", lorsque la simulation est arrêtée, soit le programme s'arrête soit la même est relancée, selon ce qui a été décidé par l'utilisateur.

Le contrôleur principal représente une simulation, une colonie d'abeilles. Il se charge d'instancier correctement les différents composants d'une simulation, et du bon déroulement de l'ensemble. Il compte les pas de temps, ce qui lui permet d'envoyer un signal aux agents, via un autre composant, leur ordonnant de vivre un pas de temps, ou lorsqu'il est temps de "logger". Via une interface Java, il offre un grand nombre de services très haut niveau à différents composants, notamment au *CombManager*, le manager des cadres, qu'il instancie avec les bons paramètres. Le CombManager se charge d'instancier le bon nombre de faces de cadres, qui seront associées par deux en cadres, puis instancie les agents initiaux et les répartis sur ces cadres, via un autre composant, l'*AgentFactory*, qui simplifie la création d'agents.

Chaque face de cadre possède une liste de cellules, qui elles même possèdent quelques attributs principaux : leur numéro et leur position x et y sur le cadre ($numéro = y * largeur + x$), ainsi qu'une place "visite" pour un agent qui serait au dessus de la cellule, et une case "contenu", pour un agent à l'intérieur de la cellule ou de la nourriture, ou même rien, pour une cellule vide. Chaque face de cadre possède une liste contenant tous les agents que contiennent ces cellules. Cette liste agit comme un raccourcis, et permet de ne pas avoir à interroger toutes les cellules à chaque fois qu'on veut accéder aux agents. Le *CombManager* possède aussi une liste d'agents, ceux qui n'appartiennent à aucun cadres, la liste de tous les agents à l'extérieur de la ruche, les butineuses.

Le *CombManager* est aussi responsable de la création, mise à jours, et bonne association des *StimuliManager*, les managers de propagation des différents stimulus dans l'environnement. Chaque cadre a deux faces, et sont placés côte à côte. Ainsi, les faces se faisant face partagent le même *StimuliManager*. Lorsque les cadres sont déplacés, de

nouveau *StimuliManager* temporaires doivent être créés pendant le déplacement, puis lorsqu'ils sont reposés dans la ruche, le *CombManager* se charge de refaire les bonnes associations, et combinaisons avec les temporaires.

Un *StimuliManager* est une grille de la même taille qu'un cadre, dont chaque case possède plusieurs flottant, représentant les intensités des différents stimulus présent sur le cadre. Sa mise à jours utilise une méthode de *double buffering* : la grille est lue et une deuxième est remplie avec les nouvelles valeurs. Une fois la lecture terminée, la deuxième grille prend la place de la première. Afin de gagner du temps, l'évaporation se fait en même temps que le calcul de la propagation. Pour propager les stimulus, nous utilisons une fonction proche d'un flou en traitement d'image : chaque pixel devient une moyenne de lui même et de tous ses voisins. Différents paramètres de stimulus nous permettent d'obtenir des comportements de propagation et d'évaporation différents.

3.1.2 Architecture des Agents

Avec les mêmes objectifs en tête, lisibilité et facilité d'ajouts, nos agents ont été pensés en niveau d'abstraction successifs. Au plus haut niveau nous trouvons la classe abstraite "Agent", qui regroupe l'âge, la position, l'énergie, la faim, une fonction abstraite "live()", ainsi que des fonctions de déplacement simples, comme le mouvement aléatoire. Nous définissons ensuite la classe "EmitterAgent", ou Agent Émetteur, qui hérite d'Agent et ajoute les interactions avec un *StimuliManager*, permettant d'émettre, ressentir des stimulus dans l'environnement. Enfin, héritant d'EmitterAgent, nous trouvons la classe "WorkingAgent", ou Agent Travailleur, qui implémente enfin "live()", avec la sélection de tâches. La fonction "live()" se déroule en quelques étapes. D'abord on vérifie si l'agent est toujours en vie, ensuite nous mettons à jours ses perceptions. L'algorithme de séléction et exécution des tâches est alors appliqué, puis une fonction abstraite est appelée, fonction permettant de faire avancer le métabolisme de l'agent, et qui est implémenté différemment par nos différentes implémentations de WorkingAgent.

Ainsi, dans les plus hautes couches d'abstraction, les différents composants ont des références "Agent", ce qui améliore la simplicité d'ajout du programme. Nous avons 3 classes implémentant WorkingAgent, les classes d'abeille Adulte, de couvain, et une pour la reine. Chacune de ces implémentations ajoute différentes tâches à sa liste de tâches, et implémente la fonction d'avancée du métabolisme.

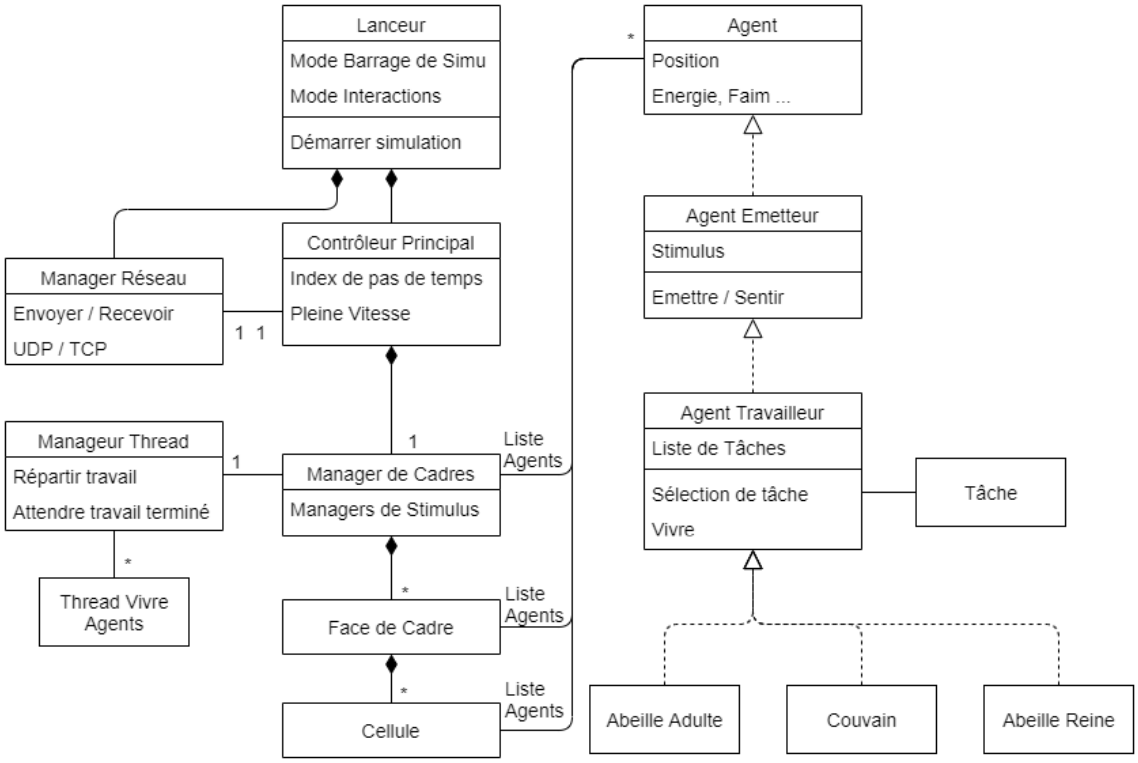


FIGURE 3.1 – Diagramme de classe esquissant l'architecture logicielle du simulateur.

3.1.3 Pas de temps et Multi-Thread

Afin d'accélerer nos simulations pour d'itérer plus confortablement dessus, nous avons mis en place un systême de multi-thread. Chaque duo de faces de cadres se faisant face, ainsi que le groupe des butineuses, vivent en concurrence, car ces différents ensembles n'interagissent pas entre eux. Pour ceci nous avons créé une classe *WorkDispatcher*, qui s'occupe de gérer un groupe de thread, de répartir le travail sur ses différents thread, d'en créer de nouveaux si nécessaire et de surveiller le moment où tous ont terminé leur travail. À chaque pas de temps, sur signal du contrôleur principal, le manager des cadres récupère et combine si nécessaire les différents groupes d'agents à faire vivre ensemble. Il envoi alors ces ensembles au *WorkDispatcher* qui redirige la liste sur un thread disponible. Chaque thread va alors appeler la fonction "live()" de chacun des agents.

Pour le cas particulier des butineuses rentrant dans la ruche, ou en sortant, deux listes synchronisées spéciales ont été ajoutées au niveau du *CombManager*. Une liste contenant toutes les abeilles qui sont sorti de la ruche sur ce pas de temps, et une liste contenant celles qui sont rentré. Les butineuses et les abeilles des cadres ne vivant pas sur les même threads, une abeille rentrante ne peux pas être ajoutée directement au cadre. Ce n'est qu'après avoir fait vivre tout le monde, que le *CombManager* se charge de faire les transferts, déplaçant les abeilles sortantes vers la liste des butineuses, et les abeilles entrante vers un cadre libre aléatoire.

3.1.4 Architecture des Tâches

Nos tâches sont décrites en subsomption hiérarchique dans le modèle, nous les avons donc implémenté de la sorte dans le simulateur. Au sommet de la hiérarchie nous trouvons la classe "Tâche", associant un seuil, un nom de tâche, différents paramètres et une activité racine. Une Tâche est une classe abstraite : toute classe que nous voudrons instancier devra implémenter la fonction qui permet de calculer son score.

La classe "TaskNode", ou Nœud de Tâche, est une interface très simple contenant deux fonctions : "search" (recherche) et "check" (vérifier). Elles nous permettent de mettre en place le fonctionnement de subsomption. La fonction Check représente la condition de chacun des blocs de notre subsomption, condition booléenne que chaque Nœud devra implémenter. Ensuite, la fonction Search, la principale, diffère selon les nœuds. La classe TaskNode est implémentée par deux classes, Activité et Action, respectant notre modèle vu plus tôt. Vous trouverez Figure 3.2 un diagramme de classe de cette architecture. La

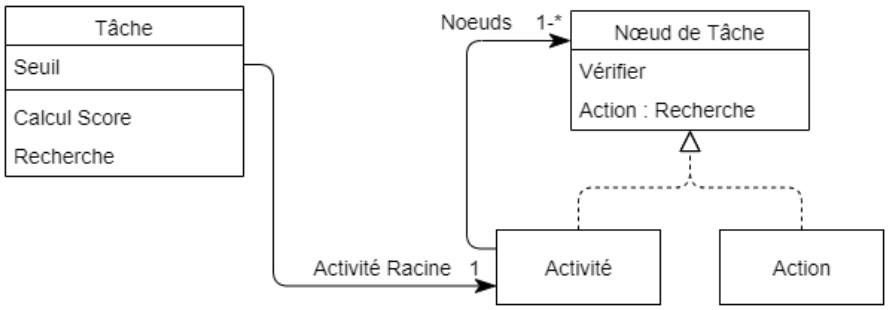


FIGURE 3.2 – Diagramme de classe de l'architecture logicielle de Tâche.

fonction Search renvoi toujours une action, et permet d'interroger récursivement toutes la subsomption hiérarchique. La fonction Search de la classe Action renvoi l'instance d'Action sur laquelle elle est appelée. En revanche, une Activité est un peu plus complexe. Une activité possède une liste de TaskNode, permettant la mise en place de la subsomption hiérarchique. Dans la fonction Search, l'Activité va interroger un par un (dans l'ordre) la fonction Check de tous ses noeuds. Si l'un deux valide son Check, l'activité appelle alors le Search du noeud validé.

Ainsi, afin de récupérer l'action à effectuer d'une Tache sélectionnée, il suffit d'appeler Search sur l'activité racine de la Tâche. Récursivement, on va descendre dans la subsomption dans les blocs aux conditions validés. Dès qu'une action est trouvée, sa fonction Search la renvoi, est elle est remontée dans toutes les fonctions Search appelées précédemment, jusqu'à ressortir au niveau du tout premier Search que nous avons appelé sur l'activité racine de la tâche. L'Action est prête à être exécutée.

Ainsi lors d'un pas de temps, le contrôleur principal demande au manager des cadres de faire vivre tous les agents. Ce-dernier parcours alors toutes ses faces de cadres et récupère tous leurs agents, afin de les envoyer aux gestionnaire de thread pour une exécution parallèle. Une fois tous les agents mis à jours, les agents entrant et sortant de la ruche sont transférés. Si le contrôleur principal a demandé à "log" le tour, alors une nouvelle mécanique s'enclenche. Le manageur des cadres récupèrent à nouveau la liste de tous les agents, et leur demande de décrire en une ligne leur état : ID, tâche, énergie, HJ et EO. À cette ligne est ajoutée au début le numéro du pas de temps transmis par le contrôleur principal. Toutes ces lignes sont alors envoyées au "Logger", qui les transfère sur un autre thread afin d'êtres écrites dans un fichier, permettant une trace de la simulation, pour

utilisation ultérieur en analyse de résultat. Vous trouverez Figure 3.1 un diagramme de classe reprenant ce que nous venons de voir dans cette section.

3.2 Modélisation de la Colonie d'Abeilles

3.2.1 Modélisation de l'Abeille Adulte

Dans notre colonie virtuelle, les adultes tendent toutes à aller butiner rapidement, grâce à l'Hormone Juvénile (HJ) qu'elles sécrètent, mais sont retenues "nourrices" par l'Ethyle Oléate(EO) émise par le couvain et les butineuses, qui "détruit" une part de leur HJ. Cette rétroaction permet de réguler le nombre de nourrice et de butineuses au sein de la colonie : Lorsqu'il y a beaucoup de couvain, très peu d'adultes vont partir butiner, car le couvain va injecter de grande quantité d'EO dans les agents de la colonie, et certaines butineuses peuvent même redevenir des nourrices. Lorsque les butineuses meurent de vieillesse, elles freinent moins le développement des nourrices, et certaines pourront partir butiner. Ces différentes interactions ont été synthétisées Figure 3.3. Dans les colonies réelles, les butineuses échangent très régulièrement des phéromones avec les receveuses, qui sont chargée de délester les butineuses de leur cargaison pour aller les stocker dans un endroit adapté dans la ruche. Nous posons une hypothèse ici, qui est que les receveuses sont le principal vecteur de l'effet rajeunissant des butineuses sur les nourrices. Or, comme nous ne les simulons pas dans cette itération du modèle, nous nous attendons à ne presque pas observer ce mécanisme.

L'Hormone Juvénile (HJ) représente directement la physiologie de l'abeille adulte, ce que nous appelons l'âge physiologique en référence au polyéthisme d'âge que nous observons dans les colonies réelles. Une abeille avec très peu d'HJ ($HJ < 0.5$), sera capable de nourrir le couvain mais incapable butiner. À l'inverse, une abeille avec un fort taux d'HJ ($HJ > 0.5$), sera capable de butiner mais incapable de nourrir le couvain. On parle donc de nourrices lorsque nous considérons les abeilles physiologiquement jeune, et de butineuses pour les abeilles physiologiquement âgées. Nous avons ajouté un paramètre à nos abeilles, leur développement ovarien, toujours nul sauf la reine pour qui il vaut 1. Cette variable n'est pas utilisée dans cette itération de l'implémentation, mais aurais permis de simuler les ouvrières pondeuses que nous retrouvons dans la réalité, lorsque les phéromones de la reine ne parviennent plus a certaines ouvrière en périphérie de la colonie et que ces dernières commencent à pondre.

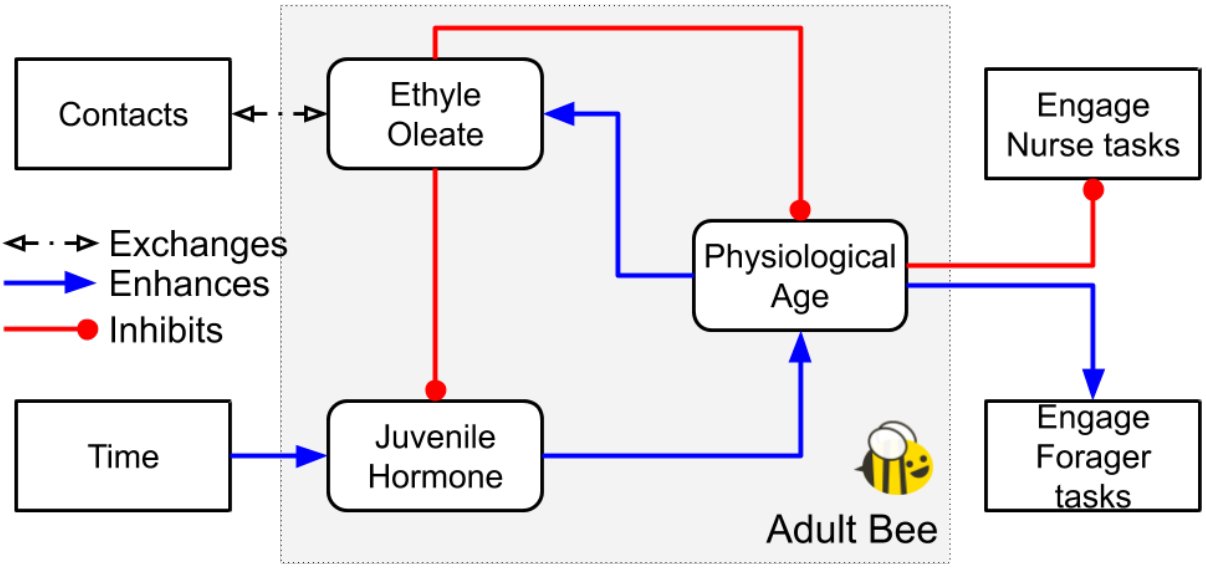


FIGURE 3.3 – Modélisation simplifiée des effets physiologues résponsables de l'auto-organisation.

Dans nos lectures, nous avons pu apprendre que les abeilles d'hiver pouvait vivre jusqu'à une année entière, mais que les butineuses meurent en une trentaine de jours, avec en moyenne les dix derniers jours passés à butiner. Les biologistes pensent que le vol est une activité très épuisante, et qu'il est possible qu'il réduise fortement l'espérance de vie des butineuses. Nous avons donc implémenté ce mécanisme pour les morts de vieillesses de nos agents. Ils meurent en une année, mais subissent une pénalité lorsqu'ils butinent. Ainsi, un agent qui butine voit son âge effectif augmenter trente fois plus vite qu'un agent à l'intérieur de la colonie.

Nos abeilles sont placés sur le cadre, savent que la sortie s'effectue par le bas (et savent aller vers le bas). Le travail de butinage est laissé très simple, le fourragement ne faisant pas partie de nos priorités ici. Les butineuses sortent de la ruche, attendent pendant un nombre donné de pas de temps puis rentrent à nouveau dans la ruche, sur un cadre aléatoire possédant une case disponible au niveau de sa ligne la plus basse.

3.2.2 Modélisation du Couvain

Nous avons modélisé les trois étapes majeures de la vie du couvain : Oeuf, larve et nymphe. Seule la larve requiert de la nourriture, les deux autres étapes n'ont pas besoin de nourrice pour se dérouler correctement. La nymphe n'émet aucune phéromones, la cellule

étant fermée et l'EO que nous avons modélisé étant transmise par contact. La larve émet de l'EO en permanence, nous avons aussi fait le choix de faire émettre ces phéromones par les oeufs. Plusieurs aspects ont motivés ce choix, que nous n'avons pas pu confirmer ou infirmer en biologie : L'oeuf est pondu par la reine qui émet des phéromones très puissante, elle en transmet donc surement à l'oeuf pendant la ponte. De plus, n'ayant pas simulé les receveuses, le mécanisme de rajeunissement des nourrices par les butineuses est minoré, l'oeuf permet donc de compenser légèrement ce biais. L'oeuf ayant une durée de vie très courte, seulement 3 jours sur les 21 totaux, il est tout à fait possible que cette émission n'ai quasiment aucun effets.

3.2.3 Tâches et Auto-Organisation

Le but de notre simulation est de retrouver un équilibre dans le partage des tâches entre ces deux tâches clés : Butiner et Nourrir le couvain. Ces deux tâches présentent pas de stimulus délencheur direct comme nous avons pu en discuter dans le chapitre précédent. Nous avons donc recourt à une motivation source pour ces deux tâches, que nous plaçons arbitrairement à 0.5. Cette motivation source est abaissée à 0 lorsque l'abeille n'est pas physiologiquement apte a réaliser la tâche, et si l'abeille n'as pas suffisamment d'énergie pour survivre le vol aller-retour. Ensuite, nous utilisons la HJ de chaque agent pour ajuster le seuil de chacune de ces deux tâches. Moins une abeille a d'HJ, plus elle aura de chance de sélectionner la tâche de nourrice, et plus une abeille aura d'HJ, plus elle aura de chance de sélectionner la tâche de butineuse. Le débatement des seuils a été ajusté pour empêcher le score de la tâche de dépasser 0.8 : L'intervalle $[0.8; 1]$ est réservé aux tâches prioritaires, que nous allons maintenant décrire. En effet, si nos agents doivent se répartir deux tâches principales, ce ne sont pas les seules, nous avons ajouté des tâches d'entretiens : Se reposer, donner à manger, demander à manger. La tâche de repos est prioritaire, si l'énergie d'un agent devient négative, il meurt [1]. Le couvain possède aussi trois tâches, une pour chacune de ses étapes de développement, et la reine possède une tâche en plus lui permettant de pondre. Vous trouverez Table 3.1 un récapitulatif de toutes les tâches implémentées.

1. Dans cette implémentation, un agent ne peut pas mourir de faim. La nourriture n'étant modélisée que dans sa forme la plus simple, la mort de faim n'a que peu d'intérêts et apporterait beaucoup de contraintes.

TABLE 3.1 – Les différentes tâches que nos agents peuvent exécuter.

Nom de tâche	Calcul du score
Tâches d'entretien	
Se reposer	1-Energie [0 ;1]
Demander à manger	Faim [0 ;0.8]
Donner à manger	Stimulus de demande détecté [0 ;0.8]
Déplacement aléatoire	0.2 (tâche par défaut)
Tâches Principales	
Butiner	0 si $HJ < 0.5$, sinon Sigmoïde seuil : 1-JH déplacé dans [0.3 ;1]
Nourrir le couvain	0 si $HJ > 0.5$, sinon Sigmoïde seuil : JH déplacé dans [0.3 ;1]
Tâches du couvain et de la reine	
Tâche d'oeuf	1 si âge d'oeuf, 0 sinon (le couvain n'a pas besoin de repos)
Tâche de larve	1 si âge de larve, 0 sinon
Tâche de nymphe	1 si âge de nymphe, 0 sinon
Pondre	0.8 si développement ovarien, 0 sinon

3.3 Calibration des Phéromones

La répartition des tâches au sein de la colonie, en particulier le nourrissage et le butinage, dépendent majoritairement de mécanisme d'hormones et de phéromones. Comme nous l'avons vu plus tôt, nous nous intéressons ici au bras de fer entre l'Hormone Juvénile (HJ) et l'Ethyle Oléate (EO), comme décrit Figure 3.3.

Paramétrer cette dynamique a été un processus complexe : Notre grande simplification du modèle, et surtout des différentes phéromones nous détache, pour cette partie, de la réalité biologique. Nous avons donc émis des hypothèse, fixé des paramètre arbitrairement, dans le but de retrouver d'autres "macro-paramètres" émergents. Les deux points clé de cette paramétrisation sont 1) La quantité relative d'EO émise par rapport à l'HJ, et 2) La force des effets de l'HJ et de l'EO. De plus, la paramétrisation s'articule autour de deux points d'équilibre : l'Equilibre en EO (EO_{Eq}), qui représente le moment où un agent a suffisament d'EO pour parfaitement compenser son vieillissement, et donc son émission d'HJ. L'autre point d'équilibre, cette fois a l'HJ (HJ_{Eq}), représente le moment où un agent sécrète la bonne quantité d'HJ pour parfaitement compenser l'évaporation d'EO, et donc maintenir le niveau de cette-dernière.

3.3.1 Hypothèses et décisions arbitraires

Étant loin de la biologie, les quantités absolue des différentes substances ne sont en rien liées à la réalité, nous avons donc pu choisir arbitrairement les quantités et points d'équilibres. Nous avons donc décidé de fixer la quantité d'HJ dans $[0; 1]$, et $HJ_{Eq} = 0.8$. De son côté, la quantité d'EO est simplement comprise dans $[0; \infty[$, et avec $EO_{Eq} = 1$.

Ainsi, un agent abeille adulte avec un taux d'HJ supérieur à HJ_{Eq} va émettre plus d'EO qu'il ne s'en évapore sur lui, et va rajeunir. Un agent avec un taux d'EO inférieur à EO_{Eq} va éliminer moins d'HJ qu'il n'en émet, et va donc vieillir.

Nous faisons ici l'hypothèse que la réduction d'HJ par l'EO se fait seulement avec une fonction prenant en compte la quantité d'EO. De même pour l'émission d'EO en fonction de la quantité d'HJ qui ne se fait que via une fonction dépendante de la quantité d'HJ de l'agent concerné. Nous posons aussi que ces fonctions ont la forme x^n, avec n un paramètre fixé expérimentalement (que nous décrirons plus tard) et x la différence entre le taux courant et la valeur d'équilibre donnée plus tôt.

Les deux effets énoncés précédemment sont imbriqués dans une boucle de rétroaction. Les quantités absolues de ces éléments ne sont donc pas pertinente, en revanche, les écarts entre ces deux produits provoquent la dynamique que nous recherchons. Nous avons ainsi décidé, pour simplifier le paramétrage, d'en fixer un et de chercher le deuxième. Nous avons donc fixé l'émission d'EO en fonction de l'HJ à une fonction linéaire :

$$EO_{em} = (HJ - HJ_{Eq}) * EO_{Evap} \tag{3.1}$$

Avec EO_{Evap} la quantité d'EO qui s'évapore lorsque l'on considère une quantité d'EO égale à EO_{Eq}. Nous retrouvons donc la fonction sous la forme $s = x^n$ décrite plus tôt, avec $n = 1$ et x l'écart d'HJ à l'équilibre facteur de EO_{Evap}.

3.3.2 Intensités des effets

Après avoir fixé ces paramètres, nous devons nous atteler à trouver la bonne combinaison d'intensité des effets des substances, ainsi que la quantité a laquelle on veut les émettre. Pour l'HJ, il suffit de prendre le point de transition, nous avons choisi 0.5, et de le diviser par la quantité de pas de temps minimum qu'il faut pour l'atteindre. Nous pouvons décider de la majorer légèrement afin de prendre en compte l'effet de l'EO, ralentissant

très légèrement le vieillissement en faible quantité. Ce ralentissement est cependant très faible pour un agent isolé, ainsi nous n'avons pas gardé l'option de la majoration.

Nous avons ainsi la quantité d'HJ émise par chaque agent à chaque pas de temps, que nous appellerons désormais HJ_{Incr}. L'EO présente sur chaque agent viendra éliminer une partie de son HJ, combattant ainsi indirectement cet HJ_{Incr}, dont nous pouvons désormais nous servir comme référence. Nous voulons qu'à EO_{Eq}, la réduction d'HJ HJ_{red} soit égale à HJ_{Incr}, pour compenser parfaitement le vieillissement de l'agent. Nous pouvons donc écrire :

$$HJ_{red} = (EO - EO_{Eq})^n * HJ_{Incr} \tag{3.2}$$

Avec n un coefficient dont nous allons nous servir pour modeler l'intensité de l'effet de réduction d'HJ. Avec $n = 1$ (Figure 3.4a), notre fonction est linéaire, nous nous servirons de cette fonction comme base de comparaison. Lorsque $n > 1$, on obtient une fonction quadratique (Figure 3.4c). L'effet rajeunissant de l'EO est diminué avant EO_{Eq} et amplifié après. Nous obtenons donc un vieillissement ainsi qu'un rajeunissement plus rapide. L'effet de l'EO est amplifié lorsque n augmente. Enfin, avec $n < 1$, on obtient une fonction racine (Figure 3.4c). De la même manière, avec une racine, l'intensité des effets de l'EO est réduite.

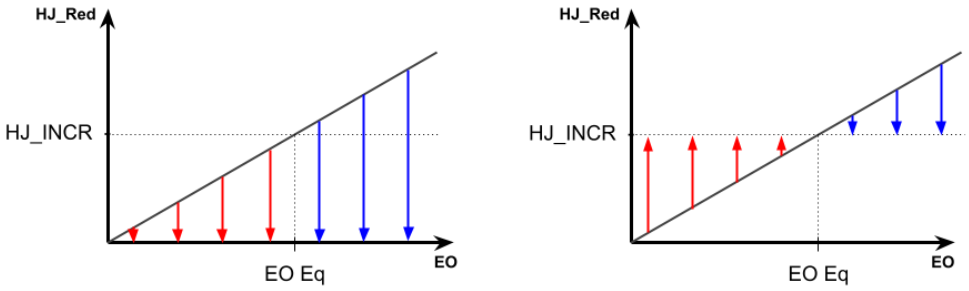
Pour illustrer, nous pouvons aussi imaginer que l'abeille vieilli naturellement rapidement. Mais, cet état d'équilibre est altéré par l'EO, nous pouvons donc visualiser ceci comme l'abeille étant attachée par un ressort à ce point d'équilibre. La rigidité de ce ressort peut alors être interprété comme la puissance de l'EO, proportionnelle à n. Plus n est grand, plus l'abeille sera tirée vers ce point d'équilibre, qu'elle soit avant ou après.

Nous n'avons pas trouvé de moyen de fixer mathématiquement n, il a donc été trouvé expérimentalement, nous y viendrons dans la partie suivante, concernant la calibration.

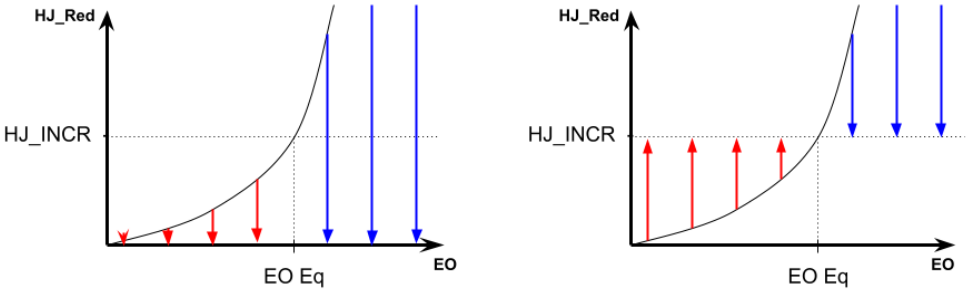
3.3.3 Quantités Émises par les Larves

Un autre point clé de ce modèle est la capacité des larves à influer directement la physionomie des adultes, en contraignant certain à rester physiologiquement jeunes. Il faut donc que ces larves en émettent la bonne quantité : Trop peu et il n'y aura pas assez de nourrices pour s'occuper du couvain, trop et il n'y aura plus assez de butineuses à la récolte de nourriture.

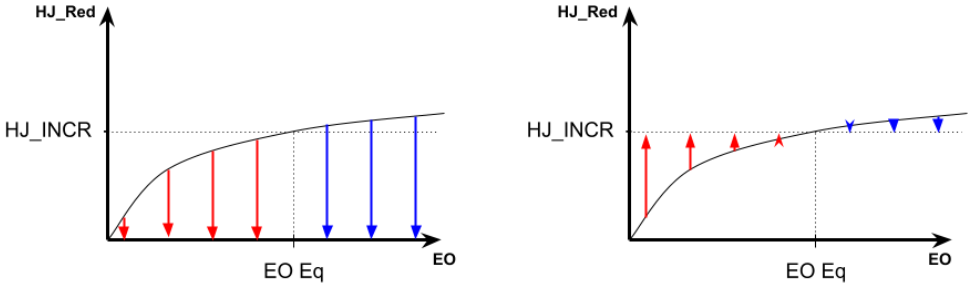
La quantité d'EO émise par chaque larve est donc importante, car elle est censée



(a) Réduction d'HJ **linéaire** (exposant = 1) par l'abeille en fonction de la quantité d'EO qu'elle possède. Lorsque $EO = EO_{Eq}$, la réduction compense parfaitement le vieillissement naturel de l'agent : $HJ_{Red} = HJ_{Incr}$.



(b) Lorsque la fonction de réduction est **quadratique** (exposant > 1) plutôt que linéaire (Fig 3.4a), on observe à gauche que l'EO a moins de puissance avant EO_{Eq}, et beaucoup plus après. A droite, on voit donc que le vieillissement ainsi que le rajeunissement (quand $EO > EO_{Eq}$) sont plus intenses.



(c) Lorsque la fonction de réduction est **une racine** (exposant < 1) plutôt que linéaire (Fig 3.4a), on observe à gauche que l'EO a plus de puissance avant EO_{Eq}, et beaucoup moins après. A droite, on voit donc que le vieillissement ($EO < EO_{Eq}$) ainsi que le rajeunissement ($EO > EO_{Eq}$) sont plus doux.

FIGURE 3.4 – Différents degrés de fonctions pour ajuster l'intensité des effets de l'Ethyle Oléate sur les abeilles adultes. Pour toutes les figures, à gauche les flèches indiquent la force de réduction de l'EO, a droite elles indiquent la vitesse de vieillissement (en rouge lorsque l'agent vieilli, en bleu lorsqu'il rajeuni).

permettre à l'abeille de rester jeune, et même de rajeunir. Nous avons donc commencé par placer cette émission à EO_{Evap} (Voir eq. 3.1). De plus, les contacts étant assez bref, l'émission d'EO des larves doit permettre à une nourrice sollicitée de rester jeune malgré quelques temps de trajets. Ce dernier point porte une incertitude liée a l'émergence de ce comportement, la répartition des larves, les temps de trajets etc. Nous avons donc décidé de placer l'émission des larves à $k * EO_{Evap}$, avec k un coefficient que nous allons trouver expérimentalement, qui sera forcément supérieur à 1.

3.3.4 Objectifs de calibration

Nous saurons que le coefficient k de l'émission d'EO des larves est juste lorsque, peu importe la quantité de larves, la proportion de nourrice par rapport aux butineuses sera globalement constante. Avec une émission trop importante, augmenter le nombre de larve va drastiquement augmenter le ratio de nourrices. Par exemple lors d'un essai, nous obtenions un ratio de 1 butineuse pour 1 nourrice avec 500 larves, mais presque 100% de nourrice pour 1000 larves. A l'inverse, lorsque trop peu de nourrices sont captées par le couvain, ce coefficient doit être augmenté.

Ensuite, il arrive qu'il faille diminuer la quantité d'EO émise par les larves, alors qu'elle est déjà trop faible pour un petit nombre de larves (ou vice versa). C'est ici le signal que l'intensité des effets de l'EO sur les adultes est à ajuster, en variant l'exposant de l'équation donnant HJ_{Red} (Eq 3.2). Ainsi, lorsque 500 larves maintiennent un ratio nourrices/butineuses correct mais faible, mais que 1000 retiennent trop de nourrice, c'est que l'EO a trop d'effet, il faut alors abaisser l'exposant de HJ_{Red}.

Nous avons gardé un exposant entier lorsqu'il est supérieur à 1, et de la forme $1/x$ (avec x entier) pour un exposant inférieur à 1.

Ce mécanisme par échange de phéromones est à la base de ce que nous cherchons à démontrer ici, et est profondément émergent, car dépendant des interactions entre tous nos agents, ce qui explique notre recours ici à un paramétrage expérimental.

Conclusion

Dans ce chapitre nous avons discuté de l'architecture logicielle du simulateur qui fait tourner notre première itération du modèle. Nous agents à l'intérieur de la colonie doivent se répartir deux tâches automatiquement, "Nourrir les larves" et "Butiner". Plusieurs

couches d'abstraction nous offrent une grande modularité dans le développement, et l'utilisation de plusieurs thread d'exécution permet d'accélérer la simulation, nous permettant d'itérer plus confortablement sur le modèle. Ensuite l'implémentation concrète du modèle de sélection et d'interruption de tâche ainsi que celle du modèle physiologiue de l'abeille adultes décrient plus tôt permettent à nous agents de s'organiser. Dans le chapitre suivant nous allons pouvoir nous intéresser à cette auto-organisation, comment la mesurer, est-elle satisfaisante. Nous parlerons aussi de calibration expérimentale, amenée par le caractère émergent de beaucoup de paramètres à retrouver, nous venant d'observation de colonies réelles.

Chapitre 4

Evaluation de l'Implémentation de la Simulation de Colonie d'Abeilles

4.1 Hypothèses et Expérimentations

4.1.1 Calibration rapide - Accélération

Dans sa version PAAMS avec la biologie accélérée Obtenir des résultats qualitatif. Le but était de vérifier la répartition des taches a l'aide du modèle, pas encore de chercher une validité

4.2 Resultats

4.3 Interprétation et Perspectives d'Améliorations

Conclusion

Chapitre 5

Etat de l'art : Simulation Multi-Agents et Environnements Immersifs

5.1 SMA : Recréer et comprendre des systemes complexes existants par l'interaction

Comprendre les mécanismes, créer un modèle puis évaluer l'impact des différents paramètres sur l'évolution du comportement du système.

5.2 Manipuler et observer ces systèmes complexes

Systèmes en général non immersif, on s'arrête à la RA, et [de ce que j'ai vu], sans utilisation d'interacteurs tangibles. Comment mieux comprendre un système complexe : Environnement immersif et interacteurs tangibles. Ruche, cadre et connaissances apicoles.

5.3 DataViz

Comment visualiser de grande quantité de données, les données micro.

5.4 Interaction / visu immersive pour comprendre

Environnement immersif

5.5 Interaction tangible

Interacteurs tangibles

Conclusion

Chapitre 6

Proposition visu interaction

Notre proposition

6.1 Interaction Immersive avec Manettes

Manipulation des cadres avec les manettes

6.2 Interaction Immersice ET tangible

L'apport et les contraintes du tangible (qu'il faut encore définir) pour la manipulation des cadres

6.3 Visualisation : Graph3D sur l'état interne de la colonie

Le graph3D et les information qu'il apporte

6.4 Résultats/Évaluation Visualisation Interactive proposée

Conclusion

Conclusion

Comment l'ensemble se comporte et avis critique sur la totalité. Notamment le modèle multi agent simplifié qui apporte une quantité gigantesque de biais.

Discussions

Perspectives

Tout est possible, pousser le modèle de l'abeille de la simulation, pousser le domaine tangible avec peut être un cadre manette (un cadre avec un ou deux boutons ?)

Conclusion

C'était cool

Rideau

clapclapclapclapclapclapclapclapclap

BIBLIOGRAPHIE

[1] W. Agassounon, A. Martinoli et R. Goodman. « A scalable, distributed algorithm for allocating workers in embedded systems ». In : *2001 IEEE International Conference on Systems, Man and Cybernetics. e-Systems and e-Man for Cybernetics in Cyberspace (Cat.No.01CH37236)*. IEEE International Conference on Systems, Man & Cybernetics. T. 5. Tucson, AZ, USA : IEEE, 2001, p. 3367-3373. isbn : 978-0-7803-7087-6. doi : 10.1109/ICSMC.2001.972039. url : http://ieeexplore.ieee.org/document/972039/ (visité le 02/04/2021).

[2] Gianluca Baldassarre et Marco Mirolli. *Intrinsically Motivated Learning in Natural and Artificial Systems*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2013. isbn : 978-3-642-32374-4 978-3-642-32375-1. doi : 10.1007/978-3-642-32375-1. url : http://link.springer.com/10.1007/978-3-642-32375-1 (visité le 07/11/2019).

[3] R. Brooks. « A robust layered control system for a mobile robot ». In : *IEEE Journal on Robotics and Automation* 2.1 (1986), p. 14-23. issn : 0882-4967. doi : 10.1109/JRA.1986.1087032. url : http://ieeexplore.ieee.org/document/1087032/ (visité le 18/01/2020).

[4] Miquel Cornudella, Paul Van Eecke et Remi van Trijp. « How Intrinsic Motivation can Speed Up Language Emergence ». In : European Conference on Artificial Life 2015. The MIT Press, 20 juill. 2015, p. 571-578. isbn : 978-0-262-33027-5. doi : 10.7551/978-0-262-33027-5-ch100. url : https://www.mitpressjournals.org/doi/abs/10.1162/978-0-262-33027-5-ch100 (visité le 12/11/2019).

[5] Mihaly Csikszentmihalyi. *Finding flow : The psychology of engagement with everyday life*. Basic Books, 1997.

[6] Alexis Drogoul et Jacques Ferber. « Multi-agent simulation as a tool for modeling societies : Application to social differentiation in ant colonies ». In : *Artificial Social Systems*. Sous la dir. de Cristiano Castelfranchi et Eric Werner. Réd. par J. G. Carbonell et al. T. 830. Berlin, Heidelberg : Springer Berlin Heidelberg, 1994, p. 2-23. isbn : 978-3-540-58266-3 978-3-540-48589-6. doi : 10.1007/3-540-

58266-5_1. URL : http://link.springer.com/10.1007/3-540-58266-5_1 (visité le 14/01/2020).

[7] Frederick W. P. Heckel, G. Michael Youngblood et Nikhil S. Ketkar. « Representational complexity of reactive agents ». In : *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*. 2010 IEEE Symposium on Computational Intelligence and Games (CIG). Copenhagen, Denmark : IEEE, août 2010, p. 257-264. ISBN : 978-1-4244-6295-7. DOI : 10.1109/ITW.2010.5593345. URL : http://ieeexplore.ieee.org/document/5593345/ (visité le 18/01/2020).

[8] Konrad Lorenz et Jeanne Etoré. *Les fondements de l'éthologie*. Flammarion Paris, 1984.

[9] Pattie Maes. « The agent network architecture (ANA) ». In : *ACM SIGART Bulletin* 2.4 (1er juill. 1991), p. 115-120. ISSN : 01635719. DOI : 10.1145/122344.122367. URL : http://portal.acm.org/citation.cfm?doid=122344.122367 (visité le 12/11/2019).

[10] Shaghayegh Roohi et al. « Review of Intrinsic Motivation in Simulation-based Game Testing ». In : *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems - CHI '18*. the 2018 CHI Conference. Montreal QC, Canada : ACM Press, 2018, p. 1-13. ISBN : 978-1-4503-5620-6. DOI : 10.1145/3173574.3173921. URL : http://dl.acm.org/citation.cfm?doid=3173574.3173921 (visité le 12/11/2019).

[11] Jürgen Schmidhuber. « Formal Theory of Creativity, Fun, and Intrinsic Motivation (1990–2010) ». In : *IEEE Transactions on Autonomous Mental Development* 2.3 (sept. 2010), p. 230-247. ISSN : 1943-0604, 1943-0612. DOI : 10.1109/TAMD.2010.2056368. URL : http://ieeexplore.ieee.org/document/5508364/ (visité le 15/11/2019).

DOCTORAT BRETAGNE LOIRE / MATHSTIC

Université de Bretagne Occidentale

Titre : titre (en français).....

Mot clés : de 3 à 6 mots clefs

Résumé : Eius populus ab incunabulis primis ad usque pueritiae tempus extremum, quod annis circumcluditur fere trecentis, circummurana pertulit bella, deinde aetatem ingressus adultam post multiplices bellorum aerumnas Alpes transcendit et fretum, in iuvenem erectus et virum ex omni plaga quam orbis ambit inmensus, reportavit laureas et triumphos, iamque vergens in senium et nomine solo aliquotiens vincens ad tranquilliora vitae discessit. Hoc inmaturo interitu ipse quoque sui pertaesus excessit e vita aetatis nono anno atque vicensimo cum quadriennio imperasset. natus apud Tuscos in Massa Veternensi, patre Constantio Constantini fratre imperatoris, matreque Galla. Thalassius vero ea tempestate praefectus praetorio praesens ipse quoque adrogantis ingenii, considerans incitationem eius ad multorum augeri discrimina, non maturitate vel consiliis mitigabat, ut aliquotiens celsae potestates iras principum molliverunt, sed adversando iurgandoque cum parum congrueret, eum ad rabiem potius evibrabat, Augustum actus eius exaggerando creberrime docens, idque, incertum qua mente, ne lateret adfectans. quibus mox Caesar acrius efferatus, velut contumaciae quoddam vexillum altius erigens, sine respectu salutis alienae vel suae ad vertenda opposita instar rapidi fluminis irrevocabili impetu ferebatur. Hae duae provinciae bello quondam piratico catervis mixtae praedonum.

Title: titre (en anglais).....

Keywords: de 3 à 6 mots clefs

Abstract: Eius populus ab incunabulis primis ad usque pueritiae tempus extremum, quod annis circumcluditur fere trecentis, circummurana pertulit bella, deinde aetatem ingressus adultam post multiplices bellorum aerumnas Alpes transcendit et fretum, in iuvenem erectus et virum ex omni plaga quam orbis ambit inmensus, reportavit laureas et triumphos, iamque vergens in senium et nomine solo aliquotiens vincens ad tranquilliora vitae discessit. Hoc inmaturo interitu ipse quoque sui pertaesus excessit e vita aetatis nono anno atque vicensimo cum quadriennio imperasset. natus apud Tuscos in Massa Veternensi, patre Constantio Constantini fratre imperatoris, matreque Galla. Thalassius vero ea tempestate praefectus praetorio praesens ipse quoque adrogantis ingenii, considerans incitationem eius ad multorum augeri discrimina, non maturitate vel consiliis mitigabat, ut aliquotiens celsae potestates iras principum molliverunt, sed adversando iurgandoque cum parum congrueret, eum ad rabiem potius evibrabat, Augustum actus eius exaggerando creberrime docens, idque, incertum qua mente, ne lateret adfectans. quibus mox Caesar acrius efferatus, velut contumaciae quoddam vexillum altius erigens, sine respectu salutis alienae vel suae ad vertenda opposita instar rapidi fluminis irrevocabili impetu ferebatur. Hae duae provinciae bello quondam piratico catervis mixtae praedonum.