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 :

Yves Duthen	Professeur d'Université, Institut de Recherche en Informatique de Toulouse
Sébastien Picault	Chargé de Recherche, Institut National de Recherche pour l'Agriculture, l'Alimentation et l'Environnement

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

Remerciements

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

Observés depuis de nombreuses années, les insectes sociaux fascinent par leur organisation. Que ce soit les autoroutes de fourmis, les danses des abeilles où les imposantes structure des termites, tout ceci est possible malgré l'absence de hiérarchie, de contrôle central. Nous nous intéressons ici principalement aux abeilles domestiques *Apis Mellifera*, l'abeille à miel européenne, largement utilisée en apiculture à travers le monde. En effet, ces travaux s'inscrivent dans le projet *SIMBACA* [1], visant à produire une simulation fidèle de ces abeilles à miel, dans un objectif double. Un objectif scientifique se concentrant sur la compréhension des mécanismes complexes régissant la colonie, ainsi que sur les possibilités de tirer de ces simulations des concepts utilisables dans d'autres domaines, ou encore de fournir aux biologistes une plateforme de simulation permettant d'évaluer l'impact de différents stress, parasites, climat, pesticide, sur la colonie, ainsi que les solutions proposées. Le deuxième objectif est pédagogique, se concentrant sur la transmission de connaissances sur la complexité de la colonie et de ses mécanismes, ainsi que l'assistance à la formation de nouveaux apiculteurs, en offrant une simulation fidèle permettant d'itérer plus rapidement et avec moins de risques pour les abeilles sur les différentes procédures apicoles. En effet, là où une action apicole peut nécessiter des mois avant que ses conséquences soient visibles, une simulation permettant d'en observer les conséquences en quelques minutes permettrai de raccourcir les cycles d'apprentissage, permettant aux apprenants d'augmenter le nombre d'essai, mais aussi de plus facilement faire le lien entre ce leurs actions et leurs conséquences (le tout sans mettre en danger de colonie réelle, alors que le secteur apicole dans le monde entier est en crise[27]). Dans le cadre du projet SIMBACA nous travaillons en collaboration avec des biologistes de l'unité "Abeilles et Environnement" de l'*Institut National de Recherche pour l'Agriculture, l'Alimentation et l'Environnement* (INRAE), ainsi qu'avec le *Groupement de Défense Sanitaire Apicole du Finistère* (GDSA29).

Le premier objectif de ces travaux est de proposer une simulation de colonie d'abeilles se focalisant sur les différentes tâches à réaliser à l'intérieur de la ruche, et les mécanismes permettant à chaque individu de savoir quelle tâche réaliser : la répartition des tâches (en

1. SIMBACA : https ://siia.univ-brest.fr/simbaca/

effet, là où beaucoup de travaux se concentrent sur les mécanismes de butinages, moins ont modélisés l'intérieur de la colonie). Nous utiliserons pour ceci une simulation à base d'agents, nous permettant de nous focaliser sur chaque individu et ses interactions avec les autres individus, et de retrouver les propriétés générales des colonies d'abeilles sans avoir à les préciser : en effet, le comportement simple de chaque individu doit faire émerger un comportement complexe à l'échelle de la colonie.

Ensuite, le second objectif est d'offrir des moyens de visualisations et d'interactions avec cette simulation, offrant à un utilisateur le moyen d'interagir naturellement avec la simulation, d'altérer son court et d'en observer les différentes propriétés, autant à l'échelle de l'individu que du système complet. Nous désirons ainsi mettre en place une visualisation claire des différents mécanismes permettant la répartition des tâches à l'intérieur de la colonie.

Dans ce document, nous commençons par un chapitre de contexte détaillant les notions de systèmes complexes et celles en découlant, ainsi qu'un état de l'art simplifié sur les connaissances biologiques actuelles des abeilles, de leur physiologie et comportements. Ensuite, le *Chapitre I* nous permet de réaliser un état de l'art sur les simulations multi-agents, les différents modèles permettant la répartition des tâches, et un tour d'horizon sur des travaux s'intéressant de près ou de loin aux insectes sociaux et leur auto-organisation présents dans la littérature. Nous aborderons ensuite notre proposition de modèle de sélection de tâche, basé sur les Modèles à seuils détaillé dans l'état de l'art et modifié afin d'inclure un plus grand nombre de type de tâches. Nous y développons aussi notre modèle de tâche, divisées en Activité et Actions afin de permettre une grande modularité. Le *Chapitre III* décrit l'implémentation de notre simulation à base d'agents de la colonie d'abeilles. Nous y décrivons notre architecture logicielle, la gestion des différents *threads* ainsi que notre modèle simplifié de la physiologie d'une abeille adulte. Le chapitre suivant décrit les différentes hypothèses et expériences mise en place afin de valider ces hypothèses, afin de valider ou non le modèle de sélection de tâche et la simulation en découlant. Nous y indiquons nos résultats puis les discutons, avant de présenter quelques perspectives à la suite de ces travaux. Nous abordons ensuite dans le *Chapitre V* un nouvel état de l'art concernant la visualisation et les interactions avec des systèmes multi-agents, que ça soit en environnement immersif ou non, afin de servir de base pour le chapitre suivant. Dans ce dernier, nous décrivons nos propositions de visualisation et d'interactions dans un environnement virtuel et immersif, respectivement à l'aide d'un graphique en trois

dimensions, et en utilisant des interacteurs tangibles : manipuler de réelle cadres (vides) traqués par la simulation qui fera les modifications en conséquences. Nous décrivons dans le *Chapitre VII* notre évaluation de ces propositions. Pour ceci un atelier a été mis en place en collaboration avec le GDSA29, où quinze apiculteurs ont pu venir réaliser une procédure apicole en réalité virtuelle, et en observer les conséquences simulées. Nous discutons des résultats et présentons quelques perspectives, pour les expérimentations comme pour nos propositions. Enfin, nous récapitulerons nos propositions, résultats et perspectives dans le chapitre du conclusion, fermant ce document.

Contexte : Colonie d'abeilles, Biologie et Complexité

Les insectes sociaux sont depuis longtemps étudiés pour leur capacité à se répartir dynamiquement le travail, sans l'aide d'un contrôle central. Ce chapitre présente tout d'abord les principales notions de complexité et détaille ensuite plusieurs exemples de phénomènes complexes que l'on peut retrouver dans la vie d'une colonie d'abeilles. Nous verrons dans le chapitre suivant quelques modèles Multi-Agents présents dans la littérature servant à modéliser ces systèmes.

Systèmes complexes

Il n'existe à ce jour pas de définition précise de ce qu'est un système complexe [26], du fait de l'étendue vertigineuse des domaines et cas d'applications touchés. Ils sont composés d'une multitude de composants hétérogènes ne communiquant que très peu entre eux, formant un tout par leurs interactions mutuelles. Un système complexe est un bon exemple de l'adage "Le tout vaut plus que la somme des parties" d'Aristote [17]. Les notions de chaos, d'interactions locales et d'imprévisibilité sont souvent citées dans la littérature. Un système complexe ne peut pas être anticipé par le calcul, le seul moyen d'en connaitre l'état futur est de l'observer. Ceci est notamment lié à leur propriétés émergentes : nous parlons d'émergence lorsque des comportements apparaissent grâce à des interactions entre différents composants, et que ces comportements sont absents lorsque ces mêmes composants sont étudiés séparément. Les propriétés émergentes sont le plus souvent imprévisibles, donnant ainsi cette propriété aux systèmes complexes. Toutes ces émergences de comportements et boucles de rétroactions génèrent un système qui peut être dit chaotique. Suivant les théories du chaos, ou "l'effet papillon", ces systèmes sont alors hautement imprévisibles et de légères variations de paramétrage ou de conditions initiales peuvent avoir de grandes conséquences sur le comportement global du système : "petit changement, grandes conséquences". De cette manière, nous pouvons imaginer un système peuplé de nombreuses entités aux caractéristiques bien différentes, mais qui interagissent massivement entre elles. C'est ainsi que l'ordre semble émerger du chaos. Lorsque

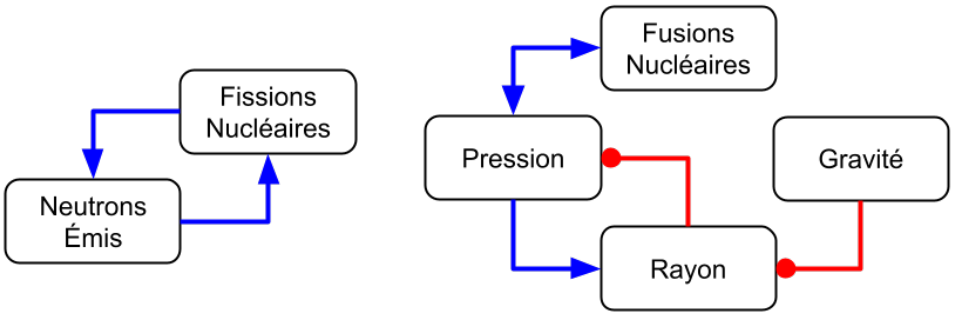


FIGURE 1 – Boucles de rétroactions simplifiées du fonctionnement d'une bombe nucléaire à fission, ou bombe A (à gauche) et de l'équilibre hydrostatique du soleil (à droite).
Un lien bleu se terminant par une flèche indique un renforcement, par exemple, la Pression au centre du Soleil vient augmenter son Rayon. Un lien rouge se terminant par un cercle indique une réduction : la Gravité à tendance à réduire le Rayon du soleil, en appliquant une force constante de toute part.

le système complexe s'organise indépendamment, sans intervention extérieure, et obtient ordre et structure, nous parlons d'auto-organisation.

Clés dans cette organisation, les interactions multiples entre les différentes composantes du système font le plus souvent émerger des boucles de rétroactions. Une boucle de rétroaction est un processus qui lie un effet à sa propre cause. En électronique, cela consiste à brancher la sortie d'un composant à l'une de ses entrées. Les boucles de rétroaction se présentent sous deux formes, les rétroactions positives et négatives. Les boucles de rétroactions positives consistent en des phénomènes de renforcement, d'accélération. Nous parlons alors dans le langage courant de "cercle vicieux ou vertueux". Une bombe nucléaire à fission fonctionne avec ce principe, une fission va en déclencher plusieurs autres, qui vont à leurs tours en déclencher bien d'autres, sous une forme d'escalade de la violence exponentielle. La Figure 1 illustre cet exemple, montrant l'amplification constante de la réaction, où chaque fission va augmenter le nombre de fissions réalisées par la bombe, par le biais d'émission de neutrons. Sans autre mécanisme pour les réguler, ce sont des phénomènes très transitoires. Les autoroutes de fourmis, de leur côté, utilisent un mécanisme de renforcement pour marquer les pistes avec des phéromones. Cette boucle de rétroaction positive est régulée par l'évaporation de ces phéromones, qui viennent contrer un potentiel emballement du système.

Les boucles de rétroaction négatives consistent en mécanismes régulateurs, stabilisant un système vers un équilibre. Nous pouvons en trouver de toutes sortes, sous bien des formes. Le soleil par exemple, voit sa forme maintenue par une boucle de rétroaction

négative, à ce qui est appelé un équilibre hydrostatique [24] : lorsque son cœur réalise la fusion nucléaire, il crée une immense force en son centre, le faisant s'étendre. Le soleil gonfle alors et fait ainsi baisser la pression en son centre, le nombre de fusion est alors légèrement réduit. La force poussant le soleil à s'étendre est alors contrée par la gravité, le poussant de toute part vers son centre. Mais du même temps, la pression en son centre remonte, les fusions reprennent de plus belle et il gonfle à nouveau : le soleil danse légèrement autour de son point d'équilibre. La Figure 1 illustre ces interactions, et montre la rétroaction entre la Pression et le Rayon du soleil : augmenter la Pression augmente le Rayon du soleil, noté par une flèche bleu. Or, augmenter le Rayon réduit la Pression, noté par un lien rouge se terminant par un cercle.

Pour récapituler, un système complexe est composé d'une multitude de composants interagissant mutuellement avec des communications limitées. Ils sont imprévisibles et voient la plupart de leur propriété émerger des interactions entre composants, sous la forme de boucle de rétroactions.

Un bel exemple de tels systèmes auxquels nous sommes soumis tous les jours est le climat. De faibles interactions qui peuvent paraitre insignifiantes ont de grosses répercussions sur le système dans son ensemble. Nous augmentons d'un demi degré la température des océans et ce sont les courants marins, les vents et tempêtes, les chaines alimentaires et bien d'autres qui se dérèglent. Augmenter de quelques points le pourcentage de gaz à effet de serre dans l'atmosphère et le circuit s'emballe et réchauffe le tout, augmentant la fréquence d'événements violents etc [2]. Une grande quantité d'entités, jusqu'à l'échelle moléculaire, vont interagir entre elles, faisant varier leurs températures, leurs pressions, leurs absorptions et réflexions de la lumière, altérant par la même occasion le climat sur une échelle planétaire, affectant jusqu'à notre mode de vie. Le climat est en bonne partie imprévisible, des modèles de plus en plus complexes sont mis en places et constamment améliorés par les météorologues afin d'affiner leurs prédictions.

Nous allons maintenant constater en quoi une colonie d'abeilles [2] est un bon représentant de la grande famille des systèmes complexes : aucun contrôle central et des dizaines de milliers d'individus hétérogènes dotés de moyens de communication limités interagissant massivement entre eux, créant des boucles de rétroactions comportementales, chimiques mais aussi physiologiques.

2. La ruche désigne l'objet, souvent une boite pour les abeilles domestiques, où les abeilles vivent. La colonie désigne l'ensemble des abeilles vivant ensemble, dans la ruche.

Abeilles et Systèmes complexes

Une colonie classique d'abeilles domestiques *Apis Melifera* héberge en moyenne quelques dizaines de milliers d'individus, 50 000 est un nombre qui revient souvent. Tous ces individus ont des besoins en nourritures et en eau, mais l'attention au couvain, regroupant tous les stades de vie de l'abeille avant sa phase adulte, demande un tout autre niveau d'attention : nourriture spéciale, température précise, cellules de cires propres etc. Tous ces besoins créent une chaine logistique impressionnante, que des apiculteurs observent depuis des milliers d'années [38]. En effet, les abeilles (et bien d'autres insectes sociaux) arrivent à survivre et même à s'épanouir sans aucun contrôle central, aucun individus spéciaux chargés du management ou de la surveillance des stocks. Des rôles ont été créés pour classer les différents maillons de cette chaîne, dont nous allons décrire les principaux [55, 56, 48].

Emblématique de la colonie d'abeilles et présentant des différences physiques notables, la reine, plus grande que les autres abeilles, est chargée de pondre à une vitesse ahurissante : près d'un œuf par minute en moyenne sur toute sa vie. Elle est en permanence entourée d'une "cour royale", d'autres abeilles qui viennent la lécher de tous côtés, attirées par ses phéromones puissantes. Elles vont ensuite répandre les phéromones royales dans toute la ruche, permettant ainsi à l'ensemble de la colonie de sentir la présence de la reine. Souvent ces abeilles sont des nourrices, elles sont chargées de nourrir le couvain. De vraies cuisinières, elles parcourent la ruche à la recherche de miel et de pollen afin de concocter un liquide calorifique dont les larves se nourrissent. Parfois, une recette alternative encore plus riche, la gelée royale, est donnée à des larves spécialement sélectionnées pour devenir de nouvelles reines.

Les larves sont des systèmes digestifs autonomes. Elles ingurgitent de grandes quantités de nourriture par rapport à leur poids, pour assurer leur croissance rapide, et leur permettre de devenir par la suite de solides ouvrières. La reine pond des œufs, qu'elle dépose au fond de cellules. Ces cellules doivent être bâties par des ouvrières cirières, et nettoyées, la reine ne pond jamais dans une cellule sale. Une fois pondu, un œuf d'ouvrière va mettre trois jours pour évoluer en larve, puis consommer son mélange de miel et de pollen pendant six jours, avant de devenir une nymphe, étape importante où la larve va se métamorphoser progressivement en sa forme finale, une abeille adulte, 21 jours après la ponte. La Figure 2 tiré des travaux de Winston [55] illustre les différents temps de croissance ainsi que les différents stades pour les différents types d'abeilles. En effet, les reines, les ouvrières et les mâles ne partagent pas les même temps de croissance, une reine

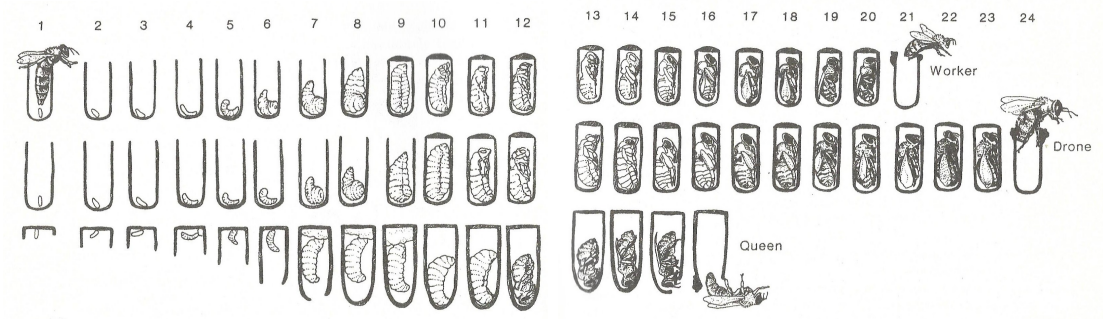


FIGURE 2 – Tiré de Winston, 1991 [55]. Croissance de l'oeuf à l'émergence, pour les ouvrières, les mâles et les reines. La croissance se passe en trois étapes, oeuf, larve, nymphe, dont les durées varient selon l'abeille. Pour les ouvrières ces étapes sont en moyenne de 3 jours en tant qu'œuf, puis de 6 jours en tant que larve et enfin de 12 jours de nymphe.

émerge 16 jours après que son œuf ai été pondu, alors que cette durée est de 24 jours pour les mâles. Dès qu'une larve passe au stade de nymphe, elle est repérée par une ouvrière cirière qui va méthodiquement operculer la cellule : créer une sorte de toit afin de celer la nymphe qui devra ouvrir sa cellule elle même, dans ses premières heures d'adulte, à l'aide de ses nouvelles mandibules.

Pour que toute cette croissance se passe bien, la température du couvain doit être exactement de 35°C. Un écart de l'ordre du degré peut engendrer des pertes colossales, et mettre la colonie en péril. Certaines abeilles effectuent alors des tâches de thermo-régulation, lorsqu'il fait trop chaud ou trop froid, afin de toujours garder une température optimale dans la ruche.

Les abeilles à miel sont surtout célèbres pour leur butinage, rôle tenu par les "butineuses". Celles-ci sélectionnent les meilleurs sites de ressources sur une surface dépassant souvent les 100km² [47] et ramènent nectar, pollen et eau à la colonie. Certaines abeilles 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". Elles repèrent les fleurs grâce à leurs immenses yeux sensibles aux ultra-violets, que les fleurs ont appris à bien réfléchir dans une co-évolution avec les pollinisateurs [52]. Une fois rentrée, la butineuse cherche une "receveuse" : elle lui donne une partie de sa charge de nectar. Les butineuses donnent ainsi leur récolte à trois ou quatre receveuses. Ensuite, elles viennent déposer leurs ballots de pollen dans des cellules proches du couvain, puis repartent butiner ou vont se reposer.

Dans le même temps, les receveuses vont déposer le nectar reçu dans des cellules, toujours très hautes sur le cadre. Certaines iront aussi tasser les ballots de pollen dans leurs

cellules, afin de ne pas perdre de place. Le nectar est alors prêt à subir ses transformations pour devenir du miel. Certaines abeilles vont alors venir "cracher" sur le nectar, afin d'y déposer leurs enzymes qui feront le travail de transformation. Elles y déposent du même coup une substance antiseptique, s'assurant ainsi que le miel ne sera pas contaminé par de mauvaises bactéries ou virus (c'est en partie ce qui lui confère ses propriétés médicales). Ensuite, pendant ce travail d'enzymes, des "ventileuses" viendront se placer au dessus du miel et battre frénétiquement des ailes. Elles ajustent ainsi l'hygrométrie du miel, l'amenant très précisément à 15% d'humidité, parfait pour la conservation.

Toutes ces ouvrières sont sœurs ou demi-sœurs. Elles ont la plupart du temps toutes la même mère, mais pas forcément le même père. Au moment de la ponte, la reine "choisi" de féconder ou non son œuf. Les cellules où sont pondus les mâles sont plus grandes que celles des femelles, l'hypothèse la plus répandue est que les cellules plus petites des femelles viennent comprimer l'abdomen de la reine, provoquant ainsi la fécondation de l'œuf ainsi déposé, sans réelle décision au moment de la ponte. Un œuf fécondé donnera une femelle, un œuf non fécondé donnera un mâle d'après un mécanisme appelé "Parthénogenèse". Les mâles ont une vie très particulière : ils sont incapables de quoi que ce soit, ni de se laver, ni de se nourrir seuls, et ne font rien dans la colonie. Dès que la situation se gâte et que les ressources se font rares (souvent au début de l'hiver), ce sont les premiers à être chassés de la colonie. Reconnaissable à leurs immenses yeux, ils ne sont bon qu'à une chose, s'envoler au bon moment vers un point de rencontre défini, localiser une reine et la féconder. Ils meurent instantanément après l'acte. Une reine est ainsi fécondée par une douzaine de mâles et va garder leur semence dans sa spermathèque et s'en servir tout au long de sa vie.

Nous allons désormais nous intéresser plus en détails à ces nombreux mécanismes de contrôle et à l'auto-organisation de la répartition des tâches qu'ils induisent au sein de la colonie.

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'elles sont nécessaires, et sans aucun contrôle central. Par exemple, la colonie ne compte pas d'individus dont la seule tâche est de surveiller la température et, au besoin, assigner une équipe à sa régulation pendant un certain temps. Chaque individu utilise ses perceptions locales, internes et/ou externes, pour savoir quel travail réaliser, ainsi nous parlons de l'auto-organisation de

l'allocation du travail. Nous nous intéressons ici en particulier aux phénomènes complexes suivants et à leurs mécanismes d'auto-organisation (dont leurs boucles de rétroaction) :

- La thermorégulation : la température est maintenue précisément à 35°C au niveau du couvain.
- La sélection des meilleures sources de nourritures par les butineuses par le biais de leur mécanisme de recrutement.
- La régulation de l'âge du premier butinage en fonction des demandes du couvain.

Thermorégulation

Lorsqu'il fait légèrement trop chaud dans la ruche, nous trouvons des abeilles thermorégulatrices : elles battent des ailes sur le couvain et/ou au niveau de la sortie de la ruche pour créer un courant d'air et rafraichir la ruche. Certaines vaporisent même de l'eau dans la ruche pour aider à faire chuter la température. Lorsqu'il fait froid, les abeilles se resserrent progressivement au dessus du couvain, afin de créer ce qui est appelé la "grappe" et tenir le couvain au chaud grâce à leurs corps. Si leur simple présence ne suffit plus, elles peuvent actionner certains de leurs muscles afin de générer un peu plus de chaleur, ce qui arrive pendant l'hiver, phase critique pour la colonie. Si la grappe est trop petite pour recouvrir l'ensemble de couvain alors une bonne partie de ce dernier, en périphérie, n'aura pas été protégé par les adultes et va mourir, on l'appelle alors le "couvain refroidi".

Un des points clés de cette capacité est que chaque individu a des "tolérances" légèrement différentes. Une diversité qui provient notamment de la présence des "demi-sœurs", dont nous avons parlé ci-dessus. J. Jones et.al. [28] ont montré que la capacité des abeilles à réguler précisément la température est liée à cette diversité génétique. En effet, pour que la colonie régule correctement, il est important d'éviter les réactions trop importantes à l'échelle de la colonie, afin de ne pas osciller entre trop chaud et trop froid. La diversité génétique modifie légèrement les seuils de tolérances des abeilles, qui vont alors progressivement réguler la température. Lorsqu'elle sera légèrement trop haute, seules certaines abeilles ventileront, les autres, de part leurs tolérances plus élevées, ne vont pas participer. Ainsi, si l'action des premières est suffisante, la température revient doucement au bon niveau. Si la température continue de monter, alors de plus en plus d'abeilles la verront atteindre leur seuil de tolérance, et se joindront à l'effort.

De cette manière, chaque individu mesure et juge la température actuelle de la colonie, et prends alors la décision de réchauffer, refroidir, ou réaliser d'autres devoirs. Sa position dans la ruche et sa sensibilité personnelle vont entrer en compte dans cette décision. Ainsi,

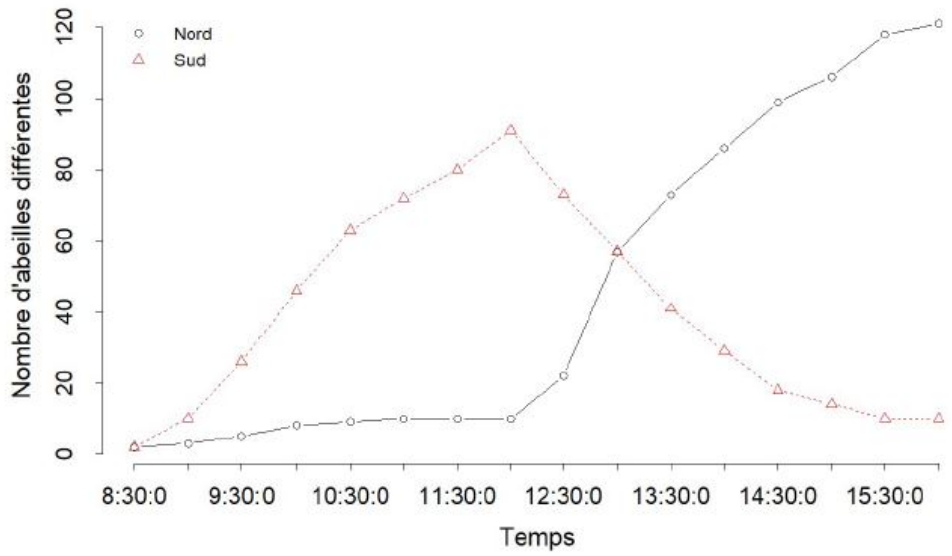


FIGURE 3 – Résultats redessinés d'une expérience de T.D.Seeley [50] montrant les capacités des butineuses à sélectionner les meilleurs sources. La source Sud commence à 2.5mol/L de sucre, et la source Nord à 1.0mol/L. La source Sud est donc la plus profitable. À midi, la source Sud passe à 0.75mol/L et la source Nord est augmentée à 2.5mol/L. La source Nord devient alors la plus profitable. Cette cassure est visible sur les courbes, où la source Sud, alors bien exploitée, se voit abandonnée en masse, tandis que l'affluence à la source Nord augmente subitement, montrant l'auto-organisation rapide des butineuses.

par l'action autonome de chacun des individus, la température du couvain ne s'écarte jamais plus d'un degré de 35°C. La colonie a toujours un nombre optimal d'individus régulant la température, permettant de répondre parfaitement au besoin sans délaisser les autres tâches.

Sélection des meilleures sources de nectar

Lors de la collecte du nectar, les butineuses sont très sélectives et préfèrent les nectars aux hautes teneurs en sucres sur des fleurs situées le plus proche possible de la ruche. La colonie utilise donc un système de recrutement afin d'allouer ses effectifs de butineuses de manière optimale et dynamique, préférant les sources proches et nutritives [39].

Avant de décoller, l'abeille recruteuse va se mettre à danser la *"Waggle Dance"*, en forme de 8. la Figure 4, tirée des travaux de T.D.Seeley [47], illustre rapidement le fonctionnement de cette danse, réalisée sur une zone bien définie de la ruche, proche de l'entrée, appelée la "piste de danse". La *Waggle Dance* a un objectif double. Le premier consiste à communiquer la position de la source de nectar, en indiquant sa direction par rapport au

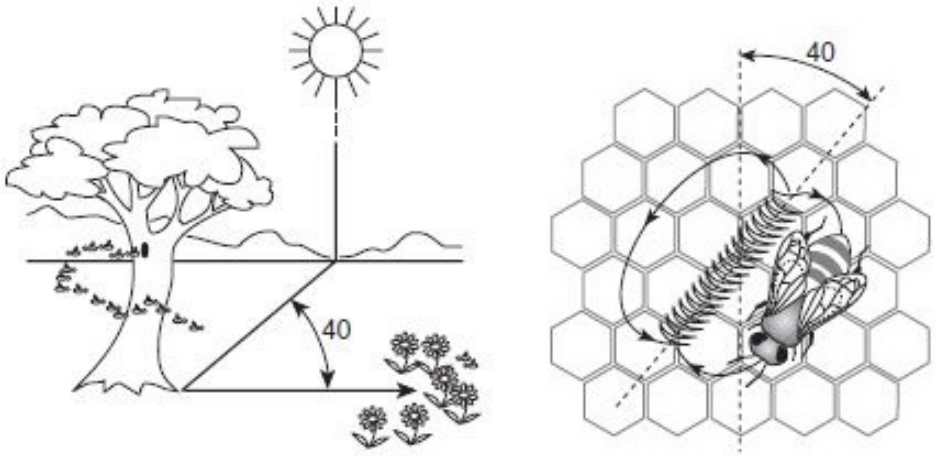


FIGURE 4 – Tiré des travaux de T.D.Seeley [47] sur la Waggle Dance.
L'abeille effectuant la Waggle Dance indique non seulemement la distance de la source à la ruche, mais aussi sa direction, en prenant le soleil comme référence. Danser vers le haut, avec un angle de 0°, indique ainsi d'aller directement vers le soleil.

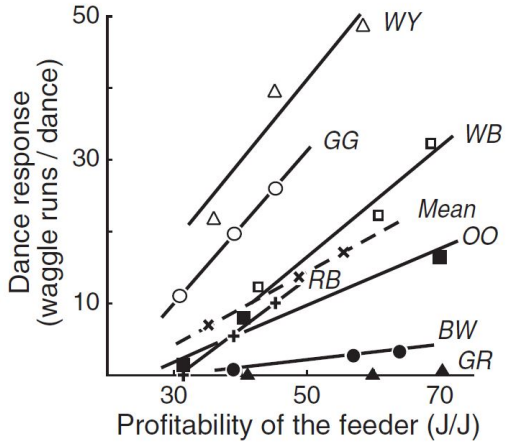


FIGURE 5 – Tiré des travaux de T.D.Seeley [47]. En abscisses la profitabilité de la source, et en ordonnées le nombre de danses répétées par chaque abeilles après être rentrée à la ruche. 7 butineuses ont été observées après avoir récolté du nectar sur une source contrôlée, leurs danses ont été enregistrées. Ils ont ensuite montré que le nombre de danse et leurs durées sont directement liés à la teneur en sucre de la source.

soleil ainsi que sa distance à la ruche, pour permettre aux autres de la retrouver. Le second est plus indirect : plus la source est sucrée et proche, et donc profitable, 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 profitable, plus la recruteuse va communiquer la position à de nombreuses butineuses. La Figure 5 est tirée des travaux de T.D.Seeley [47] et illustre empiriquement que les abeilles dansent de manière plus intense pour des sources plus sucrées.

Une fois recrutées, les nouvelles butineuses se rendent à la source et répètent le processus : collecter, juger, rentrer et parfois recruter. Là encore, la diversité est clé : différents profils génétiques poussent certaines abeilles à danser pour des sources moins profitables, d'autres à très peu danser (maximisant ainsi le butinage), et même certaines à tout simplement abandonner la source. Communiquer des sources de faible qualité va permettre à la colonie d'y maintenir un faible contingent. Ce contingent alors moins utile dans l'immédiat sert de surveillance, car les teneurs en sucres des nectars de différentes fleurs peuvent fortement varier selon les saisons mais aussi pendant la journée. Une source de faible qualité peut devenir une source extrêmement intéressante en quelques heures. Le groupe alors déjà présent peut observer ce changement et déjà danser, 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. C'est en effet une boucle de rétroactions positive, plus il y a d'abeilles à apprécier la source plus il y aura de recruteuses, augmentant ainsi encore le nombre de recruteuses par la suite.

T.D.Seeley et son équipe ont réalisé une expérience mettant en valeur cette aptitude, en plaçant deux sources de nectar aux teneurs en sucre contrôlables aux alentours d'une ruche [50]. Ils ont pu obtenir un grand nombre d'information en observant à la fois le comportement des abeilles dans la ruche, mais aussi sur chaque source artificielle. Une source, au nord de la ruche, était remplie avec un nectar à 1.0 mol/L de sucre, et l'autre au sud était quant à elle remplie d'un nectar sucré à 2.5 mol/L. La source sud est alors largement privilégiée par les butineuses, tandis qu'une faible population de butineuses exploite la source nord. À midi, la source sud est abaissé à 0.75 mol/L de sucre, et la source nord est montée à 2.5 mol/L. Alors, la source sud est abandonné en quelques heures au profit de la source nord. La Figure 3 illustre les résultats de cette expérience, où les auteurs ont observés le nombre de butineuses sur chaque source pendant la journée.

Ils ont ainsi l'adaptation de la colonie à un changement rapide d'environnement dans un temps relativement court, et que le gros de ses effectifs a bien été redirigé vers la source la plus profitable.

La diversité génétique fait ici effet de régulation, et permet d'ajuster le butinage en un savant équilibre entre exploration et exploitation. Il est certes important de ramener d'énorme quantité de nourriture à la colonie, mais avoir la totalité de ses effectifs sur une même source présente des risques. Les abeilles "scouts", ainsi que les abeilles dansant pour des sources moins profitables vont alors permettre de maintenir l'exploration et la découverte de nouvelles ressources à un niveau sécurisant, multipliant les opportunités.

Phéromones et Physiologie

L'auto-organisation de la colonie ne se joue pas que dans les perceptions externes, la colonie s'appuie aussi sur des mécanismes indirects, long-termes, 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, dont nous avons pu longuement parler avec nos partenaires biologistes. 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 [32]. 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.

De plus, les abeilles (ainsi que d'autres insectes sociaux) emploient différentes phéromones pour parvenir à survivre, et agrandir leurs colonies. Ces phéromones peuvent être séparées en deux catégories : les phéromones *modificatrices*, et les phéromones *incitatrices*. Les phéromones modificatrices, comme leur nom l'indique, viendront modifier la physiologie des individus : ce sont elles qui sont notamment responsables de la modification des propriétés des GH que nous venons d'aborder. Les phéromones incitatrices quant à elles viendront déclencher des comportements, sans altération physique sur les agents. Ainsi, nous pouvons voir les phéromones incitatrices comme ayant un effet très court terme, et les phéromones modificatrices un effet très long terme. Par exemple, lors d'une attaque, les premières abeilles témoins émettent une phéromone d'alarme très volatile, qui a pour

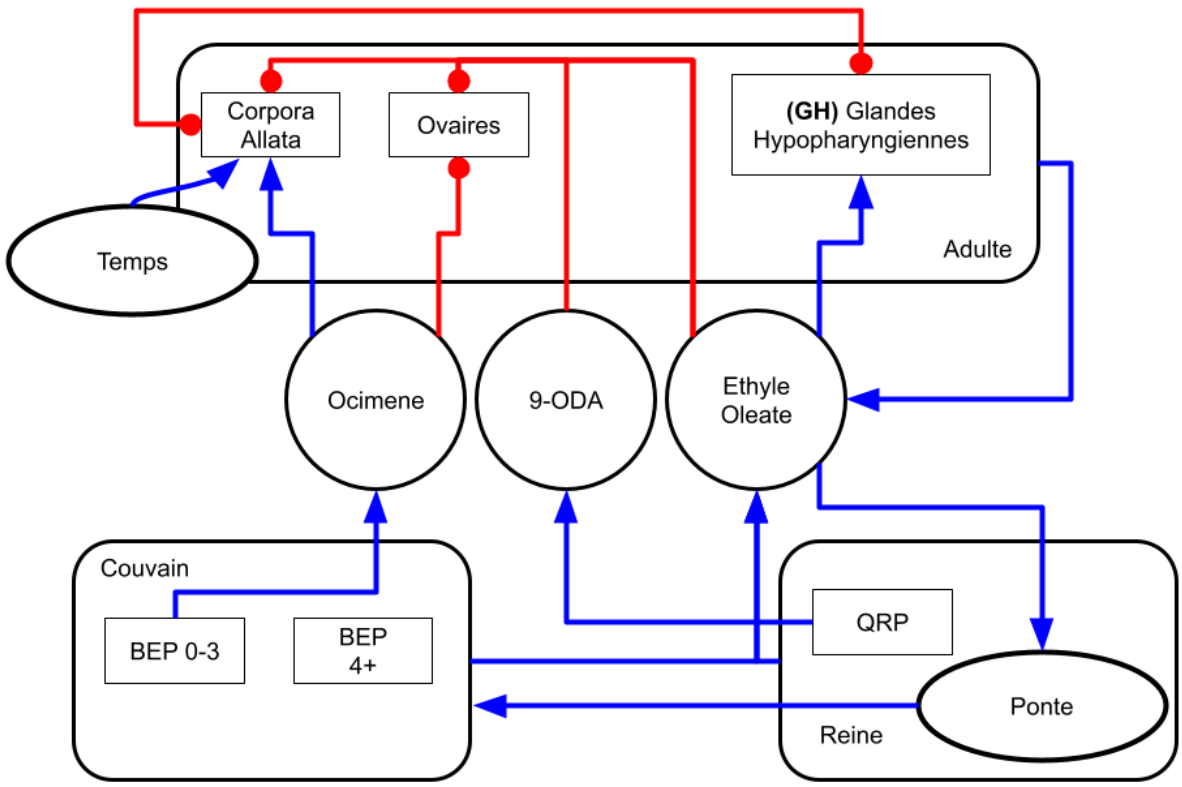


FIGURE 6 – Synthèse des connaissances simplifiées et schématisées des phéromones modificatrices au sein de la colonie. Un lien bleu se terminant par une flèche indique un renforcement, par exemple, l'Ocimene vient augmenter la concentration d'Hormone Juvénile des adultes en renforcant le fonctionnement de la Copora Allata. Un lien rouge se terminant par un cercle indique une réduction : le 9-ODA vient réduire, ou inhiber les Ovaires des ouvrières.

effet de faire sortir une grande quantité d'abeilles de la ruche pour servir de renfort. Cette phéromone incitatrice augmente aussi fortement l'agressivité des abeilles [3].

La Figure 6 schématise les interactions entre phéromones modificatrices et glandes, ainsi qu'entre différents individus de la colonie, que nous allons décrire dans cette partie. Nous avons pu construire ce modèle simplifié des phéromones modificatrices avec nos partenaires biologistes au sein du projet SIMBACA [4]. Cette Figure présente les trois principales phéromones modificatrices connues à ce jour. À gauche, les phéromones E-β-Ocimene (que nous appellerons désormais Ocimene pour plus de clarté) sont majori-

3. Fait étonnant, d'après certains apiculteurs, cette phéromone aurait parfois une odeur proche de la banane.

4. SIMBACA : https ://siia.univ-brest.fr/simbaca/

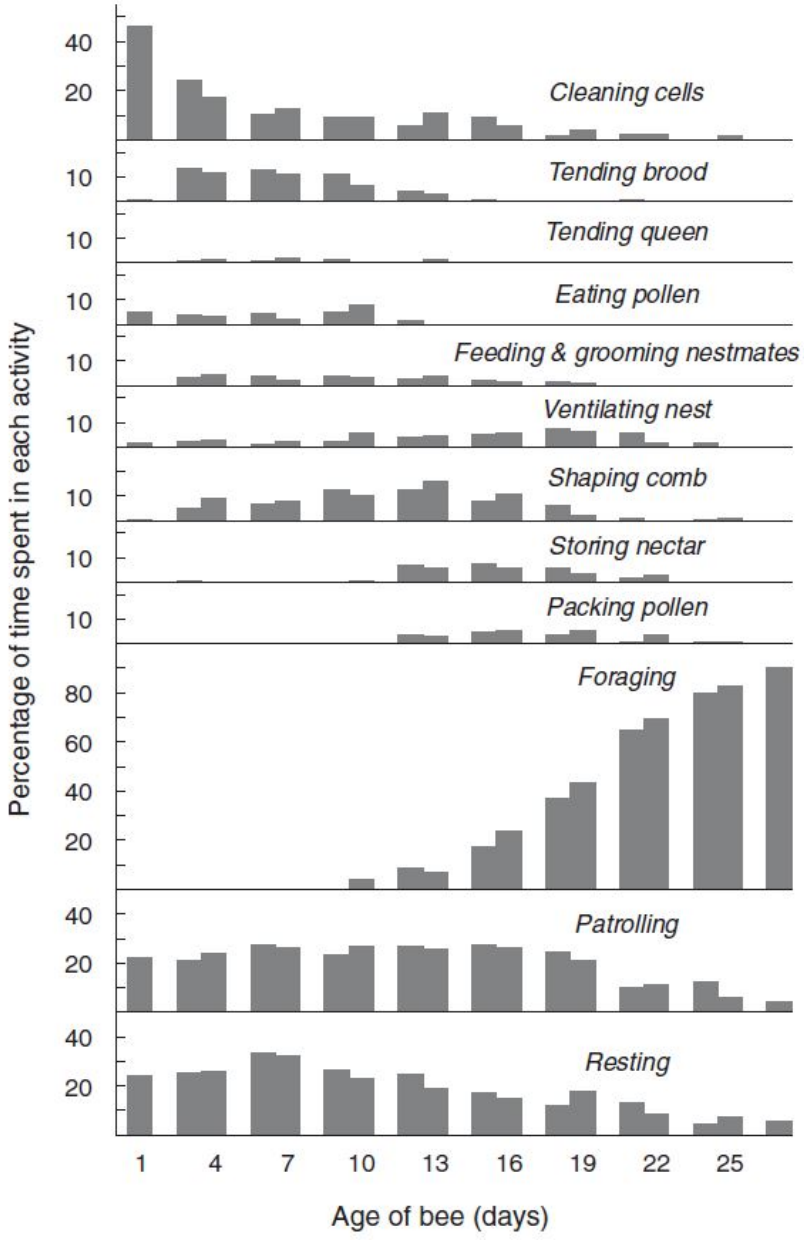


FIGURE 7 – Tiré des travaux de T.D.Seeley [47]. Occurences de tâches pour chaque abeille observée selon son âge, en jour, visible en abscisses.

tairement émises par de très jeunes larves (moins de 3 jours) [36]. Ces phéromones sont modificatrices, elles réduisent le développement ovarien des ouvrières (représenté par un lien rouge sur la figure), mais aussi incitatrices, elles déclenchent une forte hausse de butinage de pollen de la part des butineuses, qui se concentrent habituellement sur le nectar [36]. Ensuite, le 9-ODA, un des composants des puissantes phéromones de reine, ralenti lui aussi le développement ovarien des butineuses mais réduit aussi les concentrations en Hormone Juvénile (HJ) chez les ouvrières. Une hypothèse répandue est de considérer que l'HJ sécrétée par la Corpora Allata permet d'altérer le fonctionnement des GH, dictant ainsi leur utilité pour les nourrices ou les butineuses [40]. La colonie suit ce qu'on appelle le polyéthisme d'âge : les adultes ayant le même âge réalisent les mêmes activités [48]. Typiquement, la transition de nourrice à butineuse se fait en une vingtaine de jours. La Figure 7, tirée des travaux de T.D.Seeley [47], illustre les tâches réalisées par quelques abeilles observées, selon leur âge. Il a été montré que ce polyéthisme est souple, et que dans certaines conditions, une abeille adulte peut aller butiner entres 6 et 10 jours [23], au lieu de la vingtaine habituelle. Dans la même idée, en cas d'abondance de couvain, certaines abeilles retardent leur premier butinage et restent nourrices plus longtemps. Ce mécanisme est lié (du moins en partie [5]) à une phéromone, émise par toute la colonie, l'Ethyle Oléate (EO) [32]. La réduction d'HJ provoquée par l'Ocimene et le 9-ODA permet donc de maintenir chez ces ouvrières une physiologie de nourrices. Giray et. al. [23] ont aussi montré que ces différences de vitesse de développement physiologique sont aussi dues à des prédispositions génétiques.

L'Ethyle Oleate est 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. On observe sur la Figure 6 que l'EO favorise le développement des GH de nourrices, qui elles même réduisent et son altérées par l'HJ. Nous y observons aussi que l'EO ralenti le développement ovarien des ouvrières. Dans le cas classique du polyéthisme d'âge, les jeunes abeilles effectuent un travail de nourrices, et les plus âgées butinent. Mais, comme nous venons de le voir, une

5. Le couvain émets un bouquet d'une dizaine de phéromones, dont 20% EO, qui ont ensemble l'effet de retarder l'âge du premier butinage.

nourrice peut accélérer sont vieillissement, et une butineuse peut même l'inverser, afin de s'adapter aux besoins changeant de la colonie. C'est pour ceci que nous parlerons ici d'*Age Physiologique*, opposé à l'âge réel. Une abeille avec un faible âge physiologique possède les GH nécessaires aux devoirs de nourrices, et les plus âgées physiologiquement possèdent les GH et les muscles nécessaires au vol, au butinage et surtout au traitement du nectar.

Conclusion

Dans ce chapitre nous avons présenté le concept de système complexe. Nous avons ensuite pu décrire quelques parties du fonctionnement d'une colonie d'abeilles, et constater qu'il s'agissait bien d'un système complexe : régulation de la température, adaptation de la physiologie en fonction des besoins, allocation de travailleurs sur différentes sources de nourritures en fonction de leur profitabilité. Beaucoup restent encore à être découvert. Nous avons du même temps réalisé un rapide état de l'art sur les connaissances biologiques de ces insectes sociaux. Toute cette auto-organisation, sans aucun contrôle central, représente un grand intérêt de recherche, pour comprendre leurs méthodes, les reproduire et essayer d'en extraire les principes pour des applications scientifiques et technologiques. Le chapitre suivant aborde les différents modèles informatiques utilisés pour simuler de tels systèmes, et présente quelques exemples d'application de ces-derniers pour des simulation.

Chapitre 1

État de l'Art : Simulation Multi-Agents et Répartition des Tâches

Afin de comprendre et reproduire les capacités complexes des insectes sociaux, ceux-ci sont étudiés depuis une cinquantaine d'années dans le domaine des Simulations Multi-Agents (SMA). Plusieurs travaux ont ainsi pu reproduire et approcher leurs capacités d'auto-organisation dynamique par la simulation, notamment Swarm Intelligence [5] et bien d'autres [15, 42, 13]. Ce chapitre présente quelques modèles théoriques présents dans la littérature servant à modéliser ces systèmes, ainsi que quelques applications de ces derniers, notamment dans le cadre de la simulation d'insectes sociaux.

Modèles Mathématiques

Bien que nous parlions ici de simulations multi-agents, il est tout de même intéressant d'en observer leurs principales alternatives, les simulations mathématiques. Souvent à base d'équations différentielles, ces modèles permettent de rendre compte d'évolution de populations par rapport à des paramètres de haut niveau. Un des premiers et des plus cités est le système d'équations "Proies-Prédateurs" de V.Volterra [53]. Celui-ci reproduit les fluctuations de deux populations, l'une se nourrissant de l'autre, au travers de deux équations différentielles inter-connectées prenant la forme :

$$\begin{cases} \frac{dN_1}{dt} = (\epsilon_1 - \gamma_1 N_2) N_1 \\ \frac{dN_2}{dt} = (-\epsilon_2 + \gamma_2 N_1) N_2 \end{cases} \tag{1.1}$$

avec N_1 et N_2 les populations des deux espèces en nombre d'individus, respectivement les proies et les prédateurs. ϵ_1 représente le taux de croissance des proies en l'absence de prédateur, et ϵ_2 le taux de décroissance des prédateurs en l'absence de proie. Ensuite, γ_1

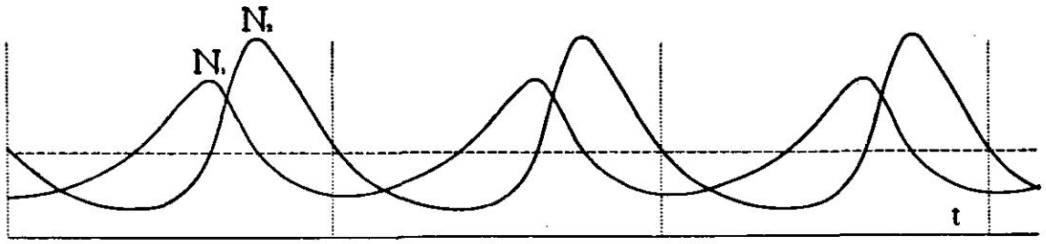


FIGURE 1.1 – [V.Volterra, 1928] [53] Évolution des populations du système mathématique "Proies-Prédateurs", avec N_2 écrit en haut et N_1 dessous.

représente l'efficacité de chasse des prédateurs ainsi que l'efficacité de fuite des proies, là où γ_2 peut représenter l'efficacité des prédateurs à convertir les proies en descendance, ce qui peut revenir au même. $\frac{dN_1}{dt}$ représente les variations de la population des proies au cours du temps, et $\frac{dN_2}{dt}$ celles des prédateurs. la Figure 1.1 est tirées des travaux de V.Volterra et illustre ce système d'équations en fonction du temps, pour des paramètres choisi par l'auteur. Nous allons décrire ces équations, en prenant l'exemple classique d'une population de loups N_2 confrontée à une population de moutons N_1. Plus les loups sont nombreux ($N2$ est grand) plus les moutons se font chasser : $\gamma_1 N_2$ est élevé, la croissance de $N1$ est alors diminuée car ϵ_1 devient de plus en plus faible par rapport à $\gamma_1 N_2$. La population de moutons commence même à diminuer lorsque $\gamma_1 N_2$ devient supérieur à ϵ_1, car $N1$ est alors multiplié par un nombre négatif (puisque $\epsilon_1 < \gamma_1 N_2$). La variation de population de moutons au cours du temps $\frac{dN_1}{dt}$ est alors négative, ce qui veut mathématiquement dire que la population de moutons diminue. Arrivé à un certain point, il n'y a plus assez de moutons pour la population de loups qui commence alors à s'effondrer : $-\epsilon_2 > \gamma_2 N_1$, donc $\frac{dN_2}{dt}$ est négative. Peu après, la faible population de loups transforme l'environnement en un paradis pour moutons qui voient alors leur population grimper, $\gamma_1 N_2$ étant très faible, le coefficient de N_1 est proche de ϵ_1, la variation de population de moutons est alors proche de son maximum. Cela transforme peu après l'environnement en un paradis pour loups rempli de moutons, $\gamma_2 N_1$ est élevé, $-\epsilon_2 + \gamma_2 N_1$ et du même coup $\frac{dN_2}{dt}$ sont alors proches de leur maximum, la population de loup augmente à toute vitesse. Ensuite, tout recommence de manière cyclique, ce que nous retrouvons dans la Figure 1.1.

Certains de ces modèles ont été créés pour simuler les évolutions de populations de colonies d'abeilles [45]. Bien qu'efficace, ce genre de simulation présente quelques lacunes [14]. Se focalisant sur les populations, ils ne rendent pas compte des décisions prises par chaque individu, ni de leurs stratégies ou comportements. Il est de ce fait impossible

pour ces simulations de prendre en compte l'importance de différents stimulus pour la réalisation de certaines actions. De la même manière, les contacts et petites interactions entre agents ne sont pas simulées. Les populations n'altèrent pas leur environnement, leurs actions sont seulement vues d'une manière probabiliste, et seuls leurs résultats sont pris en compte. De plus, les modélisations mathématiques font apparaitre des paramètres n'ayant pas vraiment de sens du point de vue du système réel, un peu à la manière des γ_1 et γ_2 de Volterra, dans son exemple pourtant très simple. Ces points nous font nous tourner vers la simulation multi-agents afin de représenter le système complexe qu'est la colonie d'abeilles, fondamentalement dépendant des interactions et contacts entre individus.

1.1 Modèles existants de répartition des tâches

La division du travail se produit lorsque plusieurs agents doivent décider quelle tâche exécuter dans un environnement partagé. Les sociétés d'individus doivent trouver des moyens de répartir efficacement leur main-d'œuvre entre les tâches nécessaires pour survivre et s'étendre. En informatique, le contrôle décentralisé inspiré par les insectes sociaux a été étudié pendant des années et s'est avéré efficace dans de nombreuses applications. Dans cette section, nous allons passer en revue ce qui a été fait dans le domaine des modèles de répartition des tâches.

1.1.1 Foraging For Work [21]

Dans ce modèle, les différentes tâches que les agents doivent accomplir sont spatialement dispersées en zones. Les agents, en recherche active de travail, tentent d'exécuter la tâche associée à leur zone. Les zones offrent une quantité fixe de travail : lorsque suffisamment d'agents y travaillent, aucun agent supplémentaire ne pourra effectuer cette tâche. Ceux-ci vont alors exécuter un déplacement aléatoire. Ainsi, les zones surpeuplées "poussent" indirectement les agents vers les zones voisines, ce qui entraîne une division du travail. Lorsque de nouveaux agents apparaissent dans une zone spécifique et que les agents meurent de vieillesse, ce modèle assez simple recrée le polyéthisme d'âges : les agents du même âge effectuent globalement les mêmes tâches, que nous retrouvons dans les colonies d'abeilles [48]. En effet, la zone voyant des agents naitre va contenir plus d'agents qu'elle n'offre de travail, là où une zone voyant des agents mourir aura plus de travail à offrir qu'elle ne contient d'agents. Ainsi, nous obtenons une zone de naissance qui

aura tendance à repousser les agents, que la zone de "mort" va alors capter, ayant besoin de main d'œuvre. Nous obtenons une répartition spatiale liée à l'âge des individus, et ainsi une forme de polyéthisme d'âge car les zones sont associées à des tâches. Ce modèle nous intéresse particulièrement pour cette capacité. En effet chez les abeilles domestiques, une hypothèse est que la migration des nourrices vers le rôle de butineuse est provoquée par l'émergence de nouvelles nourrices au centre de zones de couvain [48]. Ainsi, les jeunes nourrices repoussent les plus âgées vers d'autres activités, ce que le *Foraging For Work* est tout à fait à même de recréer.

Pour résumer, ce modèle repose sur deux hypothèses fortes :

1. Chaque tâche est associé à un niveau de priorité, connu de tous les agents.
2. Les tâches sont dispersées en zones géographiques définies.

1.1.2 Modèles à Seuils

FTM : "Fixed Threshold Model" [5]

Avec ce modèle, chaque tâche a un score, représentant sa priorité. La probabilité qu'un agent engage une tâche est directement proportionnelle à son score. Le FTM est basé sur quelques hypothèses fondatrices, dont voici la première : chaque tâche est associée à un stimulus. Le score de chaque tâche est calculé à partir de l'intensité du stimulus associé perçu par l'agent, généralement à l'aide d'une fonction sigmoïde. Soit T la tâche évaluée par l'agent, $P(T)$ le score de la tâche T et S_T son stimulus associé (simple ou complexe) perçu par l'agent, ces fonctions prennent alors la forme :

$$P(T) = \frac{S_T^n}{S_T^n + \Theta_T^n} \quad ou \quad P(T) = 1 - \frac{S_T^n}{S_T^n + \Theta_T^n} \tag{1.2}$$

avec n un entier pour la non-linéarité de la fonction (généralement $n = 2$ [42, 22]) et Θ_T le seuil de la tâche, aussi appelé biais, de la sigmoïde. La Figure 1.2 présente différentes sigmoïdes mettant en valeur l'impact du paramètre Θ_T, propre à chaque agent. 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 $\frac{1}{2}$. Ce biais est utilisé pour modifier la perception des agents : avec un biais très faible, les agents sont très sensibles au stimulus associé et s'engagent dans la tâche plus tôt que les agents avec un biais plus élevé [13].

Largement utilisés pour modéliser et piloter des simulations d'insectes sociaux, les modèles à seuils reposent fortement sur l'association entre tâches et stimulus. Les modèles

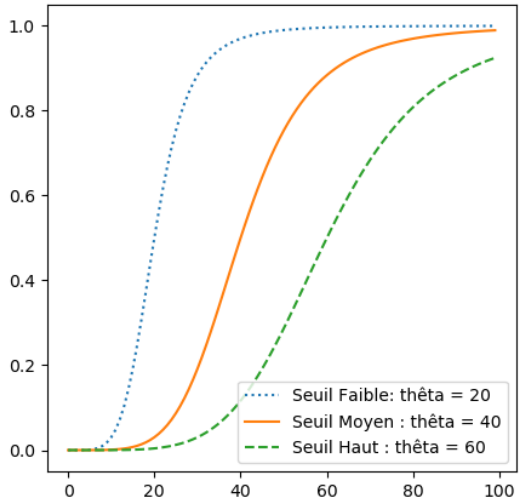


FIGURE 1.2 – L'influence du paramètre thêta (Θ) sur la forme des sigmoides.

à seuils supposent également que l'exécution d'une tâche diminue le stimulus qui lui est associé, et que ne pas exécuter une tâche augmente son stimulus associé. Le stimulus doit être une représentation de la priorité de la tâche qui lui est associée, à tout moment.

Par exemple, Bonabeau et. al. [6] utilisèrent un FTM pour modéliser la répartition du travail au sein d'une espèce de fourmi contenant deux types d'individu aux caractéristiques physiques très différentes [54]. Appelés "Majors" et "Minors" dans leurs travaux, ces castes correspondent respectivement à ce que nous pourrions voir comme de grands soldats et de petites ouvrières. Dans la nature, il a été observé que les ouvrières travaillent en permanence, alors que les soldats travaillent seulement lorsque la demande est trop forte par rapport au nombre d'ouvrières présentes. Ces deux castes ont alors été modélisées, chacune avec un seuil différent pour une même tâche abstraite. Les ouvrières ont reçu un seuil très faible, elles s'engagent donc dans la tâche même lorsque le stimulus déclencheur est relativement faible. À l'inverse, les soldats ont un seuil élevé, elles nécessitent donc un stimulus déclencheur très intense pour engager la tâche. Ainsi, lorsque le nombre d'ouvrières est suffisant pour maintenir le stimulus à un niveau faible (elles sont suffisamment nombreuse par rapport à la demande), les soldats ne travaillent pas car le stimulus n'atteindra jamais une valeur suffisamment élevée. Lorsque des ouvrières sont retirées de la simulation (ou de la colonie), le reste des ouvrières ne parvient plus à maintenir le stimulus bas : la demande dépasse l'offre. Le stimulus grimpe donc régulièrement,

jusqu'à atteindre le seuil déclencheur des soldats, qui se mettent alors au travail. Lors de la réintroduction des ouvrières, les soldats arrêterons rapidement de travailler, le stimulus déclencheur redevenant trop faible du fait du travail des nouvelles ouvrières. On obtient donc un exemple d'auto-organisation sans aucun contrôle central, sur une seule tâche et avec deux populations aux seuils fixes, mais différents.

Dans ce modèle, chaque tâche a une probabilité d'interruption aléatoire évaluée à chaque pas de temps [22]. Dès lors, l'agent recherche une nouvelle tâche en utilisant les scores de chaque tâche. Cette interruption aléatoire repose sur les hypothèses fondatrice des modèles à seuils : le score représente directement la priorité de la tâche. Ainsi, en interrompant régulièrement l'agent, on le force à observer ces niveaux de priorité, nous assurant ainsi qu'il exécute toujours la tâche la plus prioritaire.

RTM : "Response Threshold Model", Renforcement du biais

Sur la base du FTM et de l'équation 1.2, différents travaux des années 90 [51, 16, 22] ont proposé de mettre en place des mécanismes de renforcement de la valeur Θ. Modifier ainsi la sensibilité des agents pendant l'exécution offre la possibilité de former des spécialistes. Cette mise à niveau du FTM est plus généralement appelée "Response Threshold Model".

Cicirello et Smith [10] utilisèrent un modèle à seuils pour résoudre un problème d'allocation de ressource : une ligne d'assemblage de *General Motors* qui doit peindre de différentes couleurs des camions tout juste assemblés. Chaque compartiment de peinture est alors vu comme un agent qui possède une tâche par couleur de camion. Ainsi, une tâche consiste à peindre un camion d'une couleur, et changer de couleur signifie changer de tâche. Chaque tâche possède un seuil variable, permettant d'exprimer à la fois la spécialisation d'un compartiment pour une couleur, mais aussi indirectement d'exprimer le coût du changement de couleur. En effet, lors d'un changement de couleur, beaucoup de temps est perdu car il faut purger tout le système du compartiment, gâchant du même coup une bonne quantité de peinture. L'idée est donc de minimiser les coûts en peinture ainsi que le temps pour peindre une grande série de camions de couleurs différentes et inconnues *a priori*. Une file de camions à peindre arrive en entrée et les compartiments doivent en accepter certains pour les peindre. Un compartiment ajuste les seuils de ses tâches à chaque pas de temps. Ainsi, il diminue celui de sa tâche correspondant à sa couleur actuelle, augmentant ses chances d'accepter de peindre un camion de sa couleur. À l'inverse, il augmente les seuils de toutes ses autres tâches. Lorsqu'un compartiment n'a aucun camion à peindre, il diminue alors tous ses seuils de manière exponentielle.

De cette manière, Cicirello et Smith arrivent à grandement limiter le nombre de changement de couleurs nécessaires, tout en conservant un rendement proche des méthodes traditionnellement utilisées pour cette classe de problème.

1.2 Applications de Modèles de Répartitions des Tâches

Nous nous intéressons ici à quelques applications pratiques de modèles théoriques, notamment ceux que nous venons de décrire.

Drogoul et.al. [14] ont réalisé une simulation multi-agents de colonie de fourmis, ainsi qu'un modèle de tâche et de sélection de tâches. Ils ne parlaient à l'époque (1992) pas encore de systèmes à seuils, mais en étaient déjà très proche. Dans leur modèle, chaque tâche est liée à un stimulus déclencheur, et possède un poids. Pour être sélectionnée, la tâche doit multiplier son poids et son stimulus déclencheur, et la compare à sa valeur de **seuil**. Lorsque la valeur dépasse le seuil et le niveau d'activation de la tâche courante (le produit poids - stimulus au moment de sa sélection), la tâche devient *activable*. Parmi toutes ses tâches activables, l'agent vient ensuite sélectionner celle dont le *niveau d'activation* (le produit poids - stimulus) est le plus élevé (aléatoire en cas d'égalité). Une fois choisie, la tâche passe en tâche courante, et son produit poids-stimulus devient son niveau d'activation. Si aucune tâche n'est activable, le niveau d'activation de la tâche courante est réduit légèrement. Les seuils varient en fonction des activités de l'agent : ils arrivent ainsi à créer des spécialistes, des agents qui s'engagent principalement dans les même tâches. À chaque sélection d'une tâche, le seuil de celle ci est légèrement abaissé, et ceux des autres tâches sont augmentés.

Nous allons désormais décrire la simulation de colonie d'abeilles de Schmickl et Crailsheim [43]. Leur simulation se concentre principalement sur les flux de nourritures (synthétisés en nectar) au sein de la colonie. Quatre tâches sont modélisées, dont une tâche spéciale "Sans Tâche" servant de transition entre les trois autres : "Butiner", "Stocker" et "Nourrir le couvain". Chacune de ces trois tâches possède une probabilité de sélection et d'interruption. Les probabilités d'interruption sont fixes et les probabilités de sélection proviennent d'un modèle à seuils. Un score est calculé via une fonction sigmoïde paramétrée par le seuil de la tâche. Un agent doit absolument interrompre sa tâche via l'interruption aléatoire, puis passer un pas de temps à effectuer la tâche "Sans Tâche"

avant de pouvoir en engager une autre.

Le butinage est simplifié, les butineuses sortent de la ruche et y retournent un peu plus tard, avec leur réserve de nourriture remplie. Une fois rentrée à la colonie, elles attendent une receveuse pour leur transmettre le nectar. Les butineuses vont ensuite se reposer pour un nombre de pas de temps fixes puis repartent butiner. Avant cela, selon le temps d'attente T_{search} en pas de temps, chaque butineuse décide de danser ou non. Si $T_{search} <= 20$ la butineuse effectue une "Waggle Dance", afin de recruter plus de butineuses. Si $T_{search} >= 50$, la butineuse va effectuer une "Tremble Dance", afin de recruter plus de receveuses. Dans les autres cas elle ne danse pas. Les *Waggle* et *Tremble Dances* sont respectivement les stimulus déclencheurs des tâches de butinage et de stockage (retrouvée dans les colonies réelles [55]), permettant à des abeilles sans emplois de choisir une activité. Les receveuses ayant reçu du nectar vont alors aller le stocker dans le haut de la ruche, où il sera récupéré par tous les agents ayant besoin de manger, ainsi que par les nourrices qui s'en serviront pour nourrir les larves. Ces-dernières émettent un stimulus selon leur niveau de faim, servant de stimulus déclencheur à la tâche de nourrice. Dans leurs simulations, les larves ne deviennent pas des adultes et ces-derniers ne meurent pas de vieillesse. En revanche, tous les agents peuvent mourir de faim, le point principal de cette simulation étant la distribution de nectar. Ils ont ensuite pu faire des simulations en altérant l'environnement afin d'observer la réponse de la colonie, dans son organisation. Ils ont par exemple retiré des agents d'une certaine tâche au milieu de la simulation, ou rajouté/enlevé du couvain. Ils ont pu montrer par la suite que leur système s'ajuste à ces changements, faisant apparaitre son auto-organisation. Ils concluent en disant que le stimulus déclencheur de la tâche de butinage est surement complexe, et ne peut dépendre que de stimulus externes.

Nous nous éloignons légèrement des insectes sociaux pour nous diriger vers des essaims de robots. Brutschy et. al. [8] proposent un modèle permettant à des robots ne communiquant pas entre eux de se répartir efficacement le travail, entre deux tâches inter connectées. En effet, la première de ces tâches est d'aller récolter une ressource. La deuxième est d'attendre qu'un robot amène une ressource dans un endroit défini, pour aller ranger cette même ressource dans la base. Nous voyons bien ici que ces tâches sont inter-dépendantes (pour utiliser leur vocabulaire) : sans robot effectuant l'une d'elle, les robots de l'autre tâche sont coincés. Afin de correctement décider quelle tâche effectuer, les robots estiment, d'après leurs perceptions, les besoins de chaque tâches. Pour ceci, une

fois rendu à "l'interface" entre les deux tâches, chaque robot va maintenir une mémoire de la moyenne des temps d'attente auxquels il est confronté. Si ce temps d'attente dépasse un certain **seuil**, alors le robot va sélectionner l'autre tâche que celle qu'il effectue. Ainsi, au long de la simulation, chaque robot va affiner sa moyenne des temps d'attentes pour les deux tâches. Même s'ils ne parlent pas directement de modèles à seuils pour décrire leur proposition, il est intéressant de constater qu'ils utilisent seuils et sigmoïdes, avec toutefois une emphase sur les calculs probabilistes. Dans leur modèle, les transitions entre les tâches sont explicites : un agent ne peut sélectionner une des tâches que s'il exécute déjà l'autre. Les modèles à seuils proposent quant à eux des transitions implicites, où chaque tâche est en quelque sorte en concurrence pour être sélectionnée par l'agent. L'utilisation de perception locale afin d'estimer la pertinence de l'exécution d'une tâche par un agent est toutefois très intéressante.

Betti et.al. [4] proposent une simulation complexe d'une colonie d'abeilles. Avec une emphase certaine sur l'influence de l'environnement sur la vie de la colonie, notamment les pesticides et quelques parasites comme le Varroa dont nous avons parlé au Chapitre précédent, leur répartition des tâches est simplifiée. En effet, les agents ont connaissances de la répartition courante des tâches, ainsi que des réserves de nourriture de la colonie, et se servent de ces connaissances globales pour prendre leurs décisions. Les agents apparaissent en tant que larve, fécondée ou non, devenant par la suite un mâle (ne servant à rien) ou une femelle. Les ouvrières sont alors "Juvéniles" jusqu'à ce qu'elles atteignent 3 jours, et deviennent alors nourrices. Si la colonie en a besoin, une ouvrière peut changer de rôle entre nourrice et "maintenance", qui consiste en du nettoyage. Ces ouvrières peuvent aussi devenir butineuses, elles ne reviendront alors plus en arrière. La simulation du butinage est assez poussée, prenant en compte l'exploration, les danses de recrutement, ainsi que l'effet des pesticides sur la précision de vol des butineuses. Il serait particulièrement intéressant de combiner leur simulation poussée du butinage et des effets des pesticides (ainsi que l'effet des autres parasites sur les larves) avec un modèle de répartition des tâches plus lié aux perceptions locales des agents.

Enfin, nous allons nous intéresser aux travaux Schmickl et Crailsheim [44] sur le butinage et le processus de prise de décision des agents butineurs. Afin de simuler les différentes danses effectuées par les abeilles rentrant dans la ruche, ils ont mis en place un système à seuils comprenant différentes fonctions de seuils, afin de s'approcher d'observations de

colonies réelles réalisées par T.D.Seeley [49]. Différentes sigmoïdes sont utilisés, avec des paramètres très hétérogènes. Ces travaux nous prouvent l'efficacité des modèles à seuils à simuler des comportements différents, les faisant cohabiter de manière cohérente. Leurs abeilles peuvent alors décider de danser pour recruter d'autres butineuses, en leur communiquant la position de la source dont elles reviennent (avec de légères erreurs sur la communication). Mais elles peuvent aussi danser pour attirer et recruter des receveuses, ou même décider de ne pas danser du tout. Certaines partent après avoir suivi une danse, mais d'autres peuvent décider de partir en tant que "scout" explorer l'environnement, afin de peut-être prendre connaissance d'une nouvelle source de nourriture, pour en communiquer ensuite la position à ses collègues.

Conclusion

Le modèle "Foraging For Work" nous intéresse car il permet de simuler un cas intéressant de la colonie, la migration des nourrices vers le butinage. En revanche, il ne nous permettra pas de simuler l'ensemble de la colonie. C'est pour ceci que nous nous tournons vers les modèles à seuils : les "RTM". Ils présentent trois hypothèses fondatrices pour fonctionner : chaque tâche doit être associée à un stimulus déclencheur, l'exécution de la tâche vient réduire l'intensité de son stimulus associé, qui augmente lorsque la tâche n'est pas (ou pas assez) exécutée par les agents. D'après nos connaissances sur les abeilles, nous trouvons des tâches qu'elles réalisent qui ne respectent pas ces trois conditions : pas de stimulus directs pour pousser au butinage (sauf quelques cas précis) ou à l'alimentation du couvain. Nous nous intéressons donc ici aux situations dans lesquelles ces hypothèses ne sont pas vraies. Aussi, si certaines tâches, notamment celles liées au couvain, peuvent être séparées en zone définie dans la ruche, ce n'est pas le cas de la majorité : le modèle FFW ne pourra donc pas nous aider à modéliser l'ensemble de la colonie.

De plus, les abeilles montrent très peu de spécialisations individuelles, nous n'aurons donc pas besoin d'intégrer ce mécanisme à nos RTM [29].

Nous décrivons dans le chapitre suivant notre modèle fondé sur un RTM et agrémenté d'un mécanisme supplémentaire basé sur la motivation interne pour gérer ces tâches ne respectant pas les hypothèses. Nous allons aussi devoir nous passer de l'interruption aléatoire proposée par ces modèles, qui ne fait sens que lorsque les trois hypothèses sont valides.

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 utiliserons ensuite pour notre cas d'application : la colonie d'abeilles (Chapitre 3). Nous allons voir comment modéliser nos tâches et comment les faire exécuter par nos agents. 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 à 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 robots 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. Plusieurs définitions différentes existent dans la littérature. Un point qui revient souvent est que le très souvent cité "rôle" est un concept abstrait [20, 57, 9], raison pour laquelle il n'apparait pas à ce niveau de notre modèle, malgré son omniprésence dans le domaine Multi-Agents. Ici et dans la littérature, un rôle peut être utilisé pour qualifier des agents qui réalisent certaines tâches particulières. Un rôle peut ainsi englober plusieurs tâche, une seule, ou même ne concerner qu'une sous-partie d'une tâche. Le rôle n'a pas d'intérêt pour le système, en revanche, il est utile à l'observateur pour mieux analyser, communiquer et décrire ce qu'il voit. Tâches et Activités sont souvent utilisées de manière

très différentes, et parfois interchangeable, mais nous nous rapprochons de la vision de Krieger et.al. [30], disant qu'une Tâche correspond à ce qui doit être fait, là où une Activité définit plutôt ce qui est en train d'être fait. Ainsi nos Tâches représentent un travail à réaliser dans sa globalité, là où l'Activité est plus proche de ce que va réaliser un agent, plus concret. Nous ajoutons à ceci le concept d'Action, qui se glisse dans cette définition décrivant le travail élémentaire réalisé par un agent à un instant t. Voici donc une définition précise de ce que nous entendons dans ce manuscrit par ces trois termes, Action, Activité 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 de simulation, 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 élaborés.

Pour finir, une Tâche contient l'ensemble des Activités et Actions concernant un même comportement général. 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. Une Tâche est alors le dossier racine, le *root*, de cet ensemble de fichier. La Tâche assure aussi le lien entre les Activités et Actions et le modèle de sélection de tâche définit dans la section suivante, à base de stimulus déclencheur et de système à seuils.

Notre concept d'Action peut se rapprocher du concept de *primitive* avancé par Drogoul et.al.[14], et nos Tâches et Activités proche de leur concept de *tâche*. En effet, leurs *primitives* sont des comportements bas niveau élémentaires, que l'agent n'exécute qu'a travers l'éxécution de *tâches*, qui regroupent un ensemble de *primitives*. De plus, leurs *tâches* permettent aussi de faire le lien entre *primitives* et stimulus déclencheur. Ils ne parlaient pas encore de modèles à seuils, mais leur modèle de sélection en est relativement proche.

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[7]. 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 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[25]. Cette légère modification apporte une grande modularité dans la conception de ces architectures, et permet de modéliser des comportements plus élaborés sans la lourdeur des subsomptions 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. La Tâche y est représentée en un rectangle aux traits épais, les Activités ont leurs traits moyens et les Actions ont les traits les plus fins. Une bulle permet de représenter la subsomption présente à l'intérieur de l'Activité 1. Les Conditions de chaque bloc sont représentée par des losanges placés juste à côté de leur bloc correspondant. Un bloc ne sera valide que lorsque sa Condition le sera aussi.

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 priorité, qui s'arrête à la première Action interrogée avec une condition d'activation valide. Cette Action est alors remontée à l'agent, qui pourra l'exécuter pendant toute sa

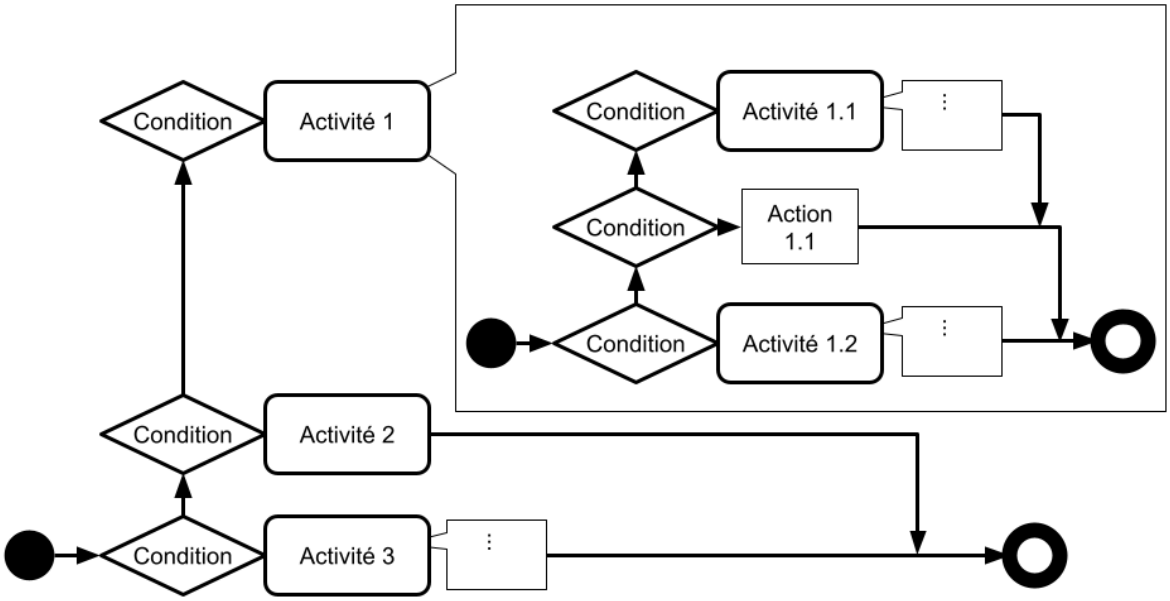


FIGURE 2.1 – Modélisation d'une tâche à l'aide d'une subsomption hiérarchique.

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, qui est validée lorsqu'un prédateur a été vu 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 reprend la structure de la tâche du mouton que nous décrivons dans cette section. Nous y observons une tâche représentée par un rectangle au trait épais, et nommée "Vie de Mouton". Elle contient une Activité "Fuir", en haut donc la plus prioritaire et représentée par un rectangle au trait moyennement épais. La Tâche contient aussi deux Actions, rectangles aux traits fins, par ordre de priorité "Brouter" et "Chercher à Manger". L'Action la moins prioritaire ne possède pas de condition, qui sont ailleurs représentées par des losanges, c'est une

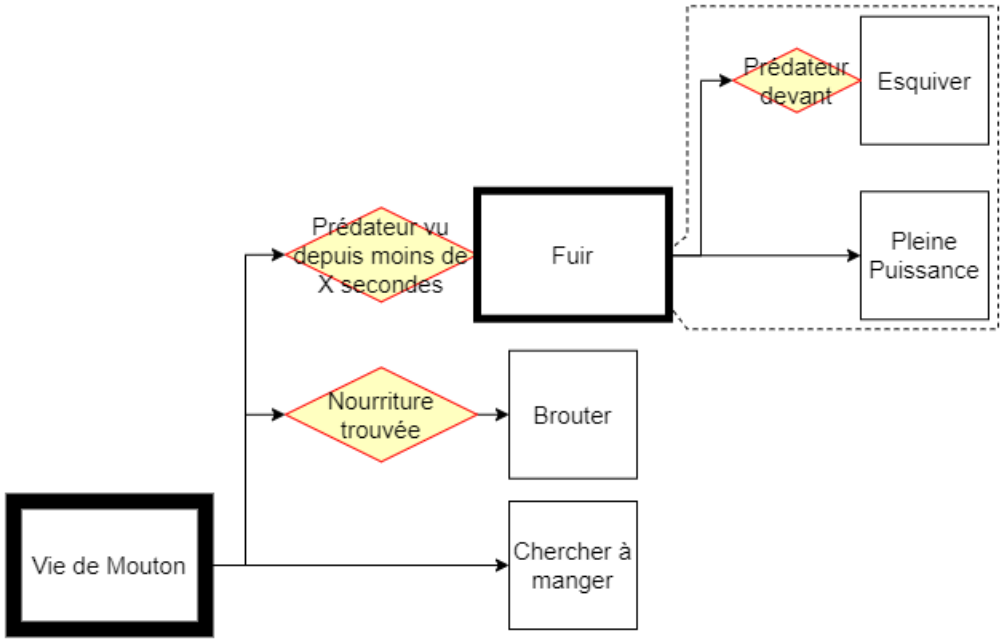


FIGURE 2.2 – Modélisation de la tâche "Vie de Mouton" contenant tout le comportement du mouton de notre exemple.

Action par défaut : si aucun autre bloc plus prioritaire n'a sa condition validée, cette dernière Action est toujours valide. Ensuite, l'Activité "Fuir" contient elle-même deux actions, "Esquiver" et "Pleine puissance" avec "Pleine Puissance" en Action par défaut ne contenant aucune condition et "Esquiver", prioritaire par rapport à "Pleine Puissance". Nous observons donc dans cette Figure l'architecture en subsomption hiérarchique, grâce à l'Activité "Fuir" qui contient une nouvelle subsomption.

2.2 Sélection : Modèle à Seuils

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.

2.2.1 Sélection avec un RTM Classique

Pour cette sélection nous allons utiliser un modèle à seuils, que nous allons légèrement adapter en lui ajoutant un mécanisme d'interruption décrit dans la Section 2.2.3. Dans un modèle à seuils, chaque tâche possède une fonction lui permettant de calculer son score. Un agent peut ainsi sélectionner la tâche qui possède le plus élevé. 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 1.1.2.

Même si plusieurs agents sont en mesure d'effectuer la même tâche, les seuils de ses tâches leurs sont propres. Chaque agent possède une instance différente des tâches, et chaque instance de tâche possède son propre seuil.

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 introduit 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 (nous en reparlerons) afin de se nourrir, pour ne pas mourir de faim.

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. Pour 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 "résoudre un casse tête", "ranger sa chambre" ou encore "rédiger un état de l'art", 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 la description de notre utilisation de la motivation, voici un rapide état de l'art sur son usage dans la littérature, et notre positionnement par rapport à celui-ci.

2.2.2 Motivation : Point sur la Littérature

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[12], 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[33], 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 [46], *e.g.* pour aider ou guider des agents apprenants

[3], notamment en apportant la notion de curiosité à des agents informatiques. Certains travaux [16, 35] 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 [11]. 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.[41] 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 trouvons notamment Agassounon et.al. [1] qui utilise une mesure locale de l'efficacité de robots pour distribuer des tâches.

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 cou. On peut le voir comme si c'était la peur qui déclenchait la fuite, la peur est donc une Motivation Source.

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 apprentissage, et donc d'optimiser l'usage de son temps. L'exemple de notre mouton sera ici quelque peu improbable, mais nous nous y tiendrons : nous souhaitons apprendre à ce mouton comment résoudre un *Rubik's Cube*. Si le célèbre 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 Guide, nous pouvons passer à la suite de la description de notre modèle, dont la sélection et l'interruption des tâches se feront dorénavant grâce à ces motivations.

2.2.3 Action Démotivante et Tâche Motivée

Stimulus Artificiel

Dans le cas de Tâche dont aucun stimulus déclencheur ne peut être trouvé, nous appliquons la Motivation Source et proposons de créer un stimulus déclencheur artificiel, d'une valeur donnée, qui peut être fixe mais que nous pourrions faire varier au besoin. Ce stimulus artificiel viendra agir exactement comme si l'agent percevait ce stimulus, et sera alors traité par la fonction de calcul de score de la tâche comme un stimulus classique. Ainsi, toutes nos tâches, ayant recours à un stimulus artificiel ou non, sont capables de produire un score cohérent et sont comparables entre elles. 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 peut permettre de représenter un changement de perception, ou une décision de la part d'un agent, si besoin. La motivation source est conservée constante dans la totalité de nos implémentations.

En revanche, le score d'une Tâche Motivée ne représente alors plus sa priorité. Afin de résoudre ce débordement des hypothèses fondatrices des modèles à seuils, nous proposons un mécanisme d'interruption basé sur la motivation guide.

Motivation comme Mécanisme d'Interruption

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. 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 à correctement réaliser sa tâche, il devrait donc essayer de faire autre chose.

Nous proposons 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 (représentante de la Motivation Guide de l'agent) 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. Rien d'aléatoire, mais plus une tâche a un score élevé, moins il y a de chances qu'une autre tâche soit sélectionnée à sa place. En effet, lors du calcul du score d'une tâche, celui d'une Tâche Motivée est remplacé par la motivation interne de l'agent, seulement si c'est cette tâche que l'agent exécutait au pas de temps précédent. Ainsi, une Tâche Motivée est sélectionnée grâce à son score provenant du stimulus artificiel, mais est interrompue par une valeur de motivation interne 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.

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. Nous avons aussi fait le choix de ne pas intégrer de Tâches Motivantes dans le modèle, qui augmenterai la motivation interne de l'agent. Ce choix tient plus de positionnement : afin de garder un modèle simple nous désirions en effet voir si ne remonter la motivation interne qu'aux changements de tâches suffirait. La Section 4.4 aborde notamment nos discussions et perspectives sur ce point.

Dans la littérature nous trouvons les travaux de W. Agassounon [1], qui sont assez proches de ce que nous proposons ici. Ils proposent de mesurer le temps depuis lequel un agent n'a pas réussi à être productif. Il mesure l'efficacité de chaque robot dans la tâche de collecte de ressource en regardant depuis combien de temps chaque agent n'a pas

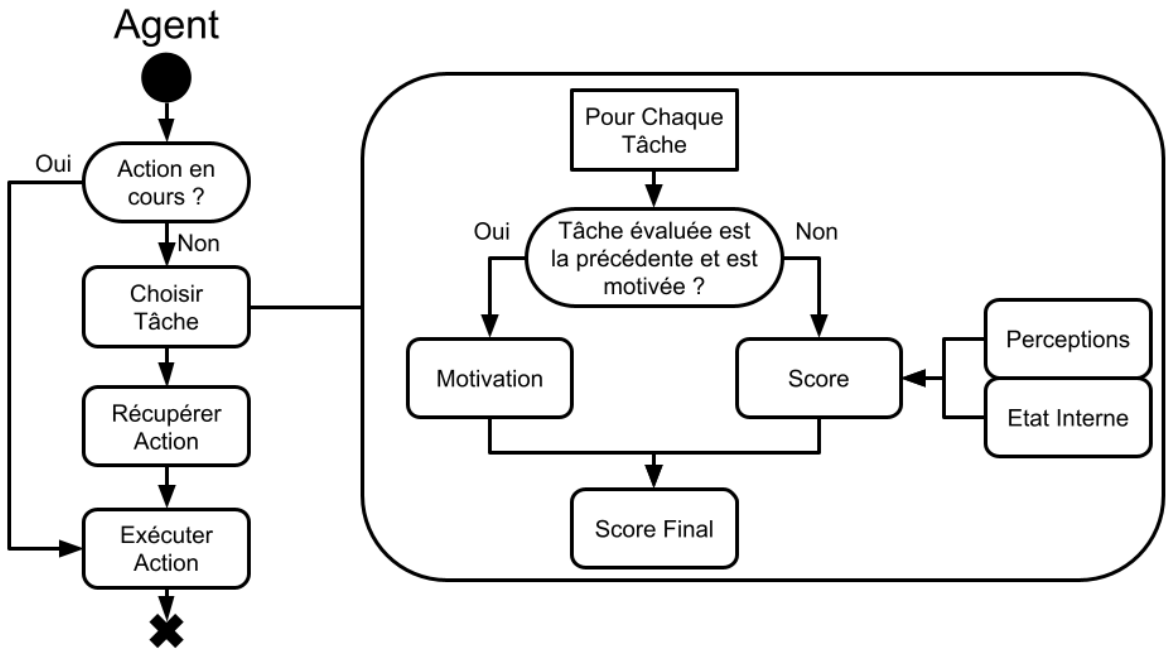


FIGURE 2.3 – Sélection et exécution des tâches par chaque Agent, à chaque pas de temps. Si l'Action en cours est terminée, l'agent va sélectionner une nouvelle tâche, en extraire l'Action à réaliser puis l'exécuter.

attrapé ou déposé une ressource. Lorsque ce temps dépasse un seuil, l'agent arrête alors ce qu'il fait et déclenche sa tâche de repos. Lorsque le temps de repos passe à son tour au delà d'un certain seuil, l'agent reprends alors sa tâche de collecte de ressource. Ces transitions sont fixes et fonctionnent en couple, sa tâche de repos n'est sélectionnée que lorsque des agents ne parviennent pas à correctement réaliser leur tâche de collecte. Ainsi, notre modèle fonctionne sur le même principe mais étend la sélection et l'interruption aux modèles à seuils, permettant à de nombreuses tâches hétérogènes de cohabiter sans liens explicites.

2.3 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 quelle Action 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 cités précédemment, nous avons donc décidé de tenter l'expérience avec une valeur transversale à toutes les tâches, liée à l'agent lui même, permettant de limiter le nombre de paramètres. La Section 4.4 aborde nos discussions sur ce point.

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.

2.4 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. Ils ne peuvent pas communiquer entre eux. Ils possèdent en revanche des capteurs leurs permettant de se voir entre eux.

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. L'outil porté par un agent est visible des autres agents l'observant.

Nous pouvons dès lors commencer à construire nos tâches :

- **Patrouiller**, Tâche Motivée : 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**, Tâche Motivée : le robot parcourt 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. Chaque robot pourra ensuite, dans sa tâche de collecte, utiliser cette mémoire pour se rendre directement au gisement.

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,

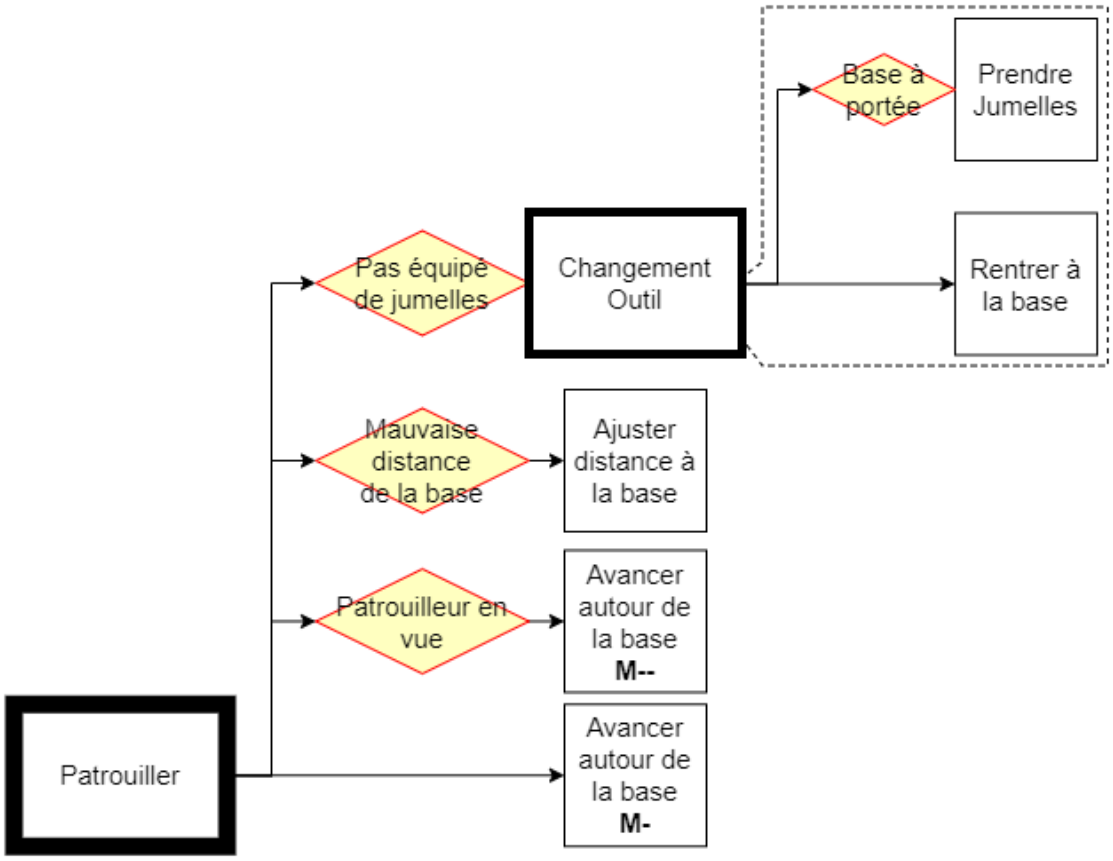


FIGURE 2.4 – Robotique en essaim : modélisation de la tâche de patrouille.

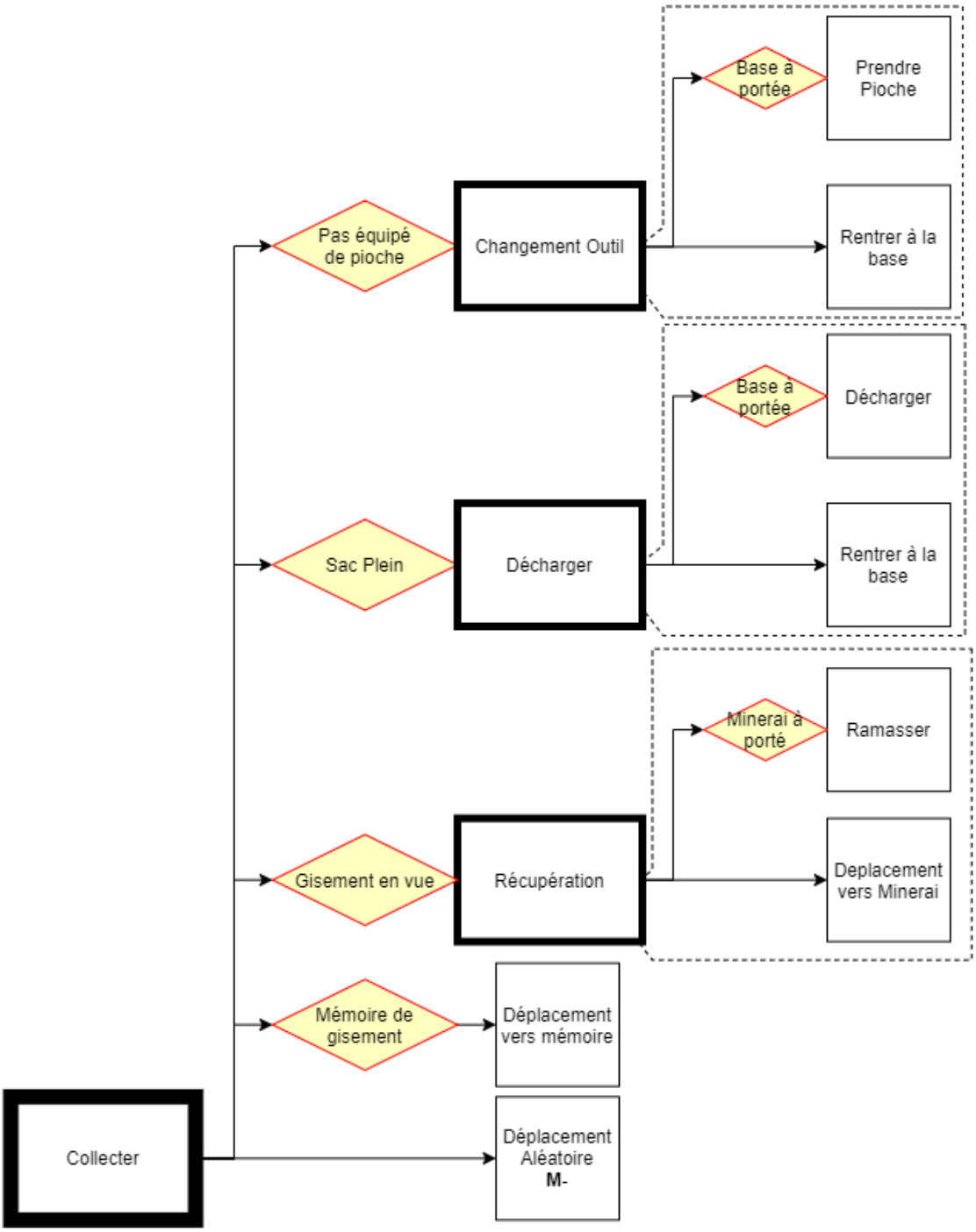


FIGURE 2.5 – Robotique en essaim : Modélisation de la tâche de collecte de ressources.

ce n'est pas le cas pour *Patrouiller* et *Collecter*. Nous leur appliquons donc un stimulus artificiel. Nous pouvons ensuite modifier légèrement ces stimulus artificiels avec des perceptions de l'agent : 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.

Cet exemple d'implémentation du modèle de répartition des tâches à fait l'objet d'un projet étudiant de Master 1.

Conclusion

Dans ce chapitre nous avons décrit notre modélisation des Tâches en Activités et Actions, ainsi que la modification des modèles à seuils pour notre sélection de tâches. Cette sélection se fait via un système à seuils adapté afin de prendre en compte les Motivations Source (nous venant plutôt de l'éthologie) et Guide (ou interne, nous venant de la théorie du *Flow*), afin d'inclure des tâches ne présentant pas de stimulus déclencheur défini. 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 élevé. Lorsque la tâche précédemment réalisée est une Tâche Motivée, son score est remplacé par la Motivation Guide, interne à l'agent et transversale à toutes ses tâches. 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. Certaines Actions peuvent être Démotivantes : un agent qui exécute une Action Démotivante va réduire sa Motivation Guide, augmentant ainsi ses chances de changer de tâche au prochain pas de temps. On parle alors de mécanisme d'interruption. Nous avons décrit un exemple possible d'application à un essaim de robots, et nous allons désormais pouvoir aborder l'implémentation de l'application principale de ces travaux, la colonie d'abeilles et son simulateur.

Chapitre 3

Modélisation et Implémentation pour la Colonie d'Abeilles

Nous allons désormais aborder dans ce chapitre 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 prenant 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 : l'idée est de produire un simulateur propre, qui serait simple d'entretien et facile à améliorer et complexifier par la suite, au delà des travaux de thèse présentés ici. Un compromis entre confort d'écriture et performances qu'offre Java, ainsi que notre bonne maitrise du langage nous a poussé l'utiliser. Pour nous aider à mettre en place ces notions de propreté du code, nous avons utilisé des Interfaces Java, permettant de découpler des grande parties du programme, facilitant l'implémentation de modifications. Nous avons, le plus possible, pensé l'architecture en composants indépendants échangeant le moins d'informations possibles. Nous allons désormais décrire les principaux composants de cette architecture, la Figure 3.2 résume le tout graphiquement.

Le simulateur est pensé en couches d'abstraction successives. Au plus haut niveau, nous trouvons le lanceur, la classe *BeekeeperLauncher*, le célèbre "*main*", qui se charge des simulations. Il présente deux modes :

— Interactions : Ce mode permet de lancer une simulation et prends en compte les interactions de l'utilisateur (via le réseau) pour altérer le comportement de la simulation, et renvoie une importante quantité de donnée par ce même réseau.

— Barrage de Simulations : mode permettant de lancer un grand nombre de simulations à la suite, sans interactions possibles. Ce mode sert principalement pour calibrer certains paramètres.

Dans le mode "Interactions", le lanceur prépare le composant réseau *NetworkManager* qui servira à envoyer et recevoir les informations de l'application interactive, composant qui n'est pas instancié dans le mode "Barrage de Simulations". Le lanceur instancie ensuite le *MainController*, le contrôleur principal, et lui partage le composant réseau si besoin. Le *MainController* 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 une autre est relancée avec les mêmes paramètres, selon ce qui a été décidé par l'utilisateur.

Le *MainController* gère 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. Via une interface Java, il offre un grand nombre de services 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 pour former les cadres. Il instancie ensuite les agents initiaux et les réparti sur ces cadres, via un autre composant, l'*AgentFactory*, qui simplifie et centralise la création d'agents. La Figure 3.1 illustre les différentes interactions entre composants logiciels lors de l'initialisation d'une simulation.

Chaque face de cadre possède une liste de cellules, des *Cell*, qui elles-mêmes possèdent quelques attributs : 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 place "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

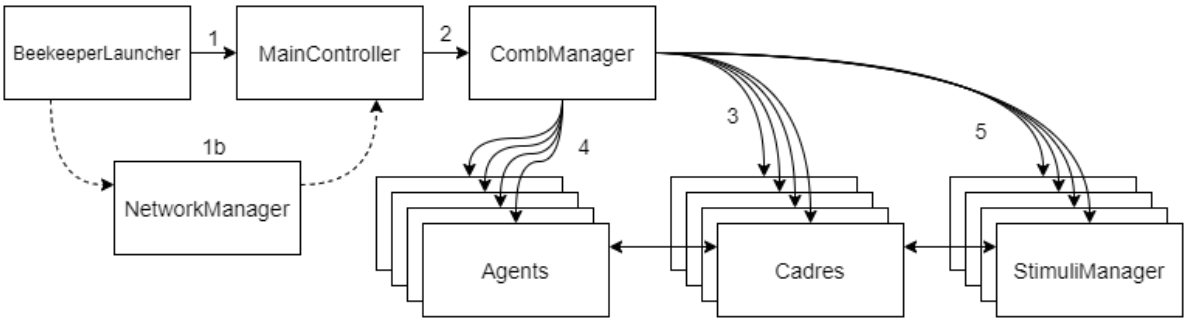


FIGURE 3.1 – Sequencement de l'initialisation d'une simulation.
1- Le lanceur *BeekeeperLauncher* instancie et initialise le *MainController* qui se chargera d'une simulation. 1b- Dans le cas où le lanceur est dans le mode Interactions, il instancie un *NetworkManager* et en donne la référence au MainController (Lors d'un changement de simulation, le *MainController* est remplacé mais le *NetworkManager* reste le même). 2- Le *MainController* se charge désormais d'instancier un *CombManager* avec les bons paramètres. 3- Le *CombManager* commence par initialiser les *Cadres*, il instancie en réalité des faces de cadres qu'il va associer par deux pour former les cadres de la simulation. 4- Une fois créés, les *Cadres* sont alors peuplés par le *CombManager* avec des *Agents*. 5- Ensuite, des *StimuliManager* sont instanciés par le *CombManager* et associés à des faces de cadres.

les agents que contiennent ces cellules. Cette liste agit comme un raccourci, et permet de ne pas avoir à interroger toutes les cellules à chaque fois que nous voulons accéder aux agents. Le *CombManager* possède aussi une liste d'agents, ceux qui n'appartiennent à aucun cadre : la liste de tous les agents à l'extérieur de la ruche, les butineuses.

Le *CombManager* est aussi responsable de la création, mise à jour, et bonne association des *StimuliManager*, les managers de propagation des différents stimulus dans l'environnement. Les cadres sont placés les uns à côtés dans autres, les faces se faisant face partagent le même *StimuliManager*.

Un *StimuliManager* est une grille 2D de la même taille qu'un cadre, dont chaque case possède plusieurs flottants, représentant les intensités des différents stimulus présents sur le cadre. Sa mise à jour 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. La propagation se fait à chaque pas de temps pour chaque stimulus et, 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 pondérée de lui-même et de tous ses voisins. Pour nous, chaque cellule voit sont intensité devenir la moyenne

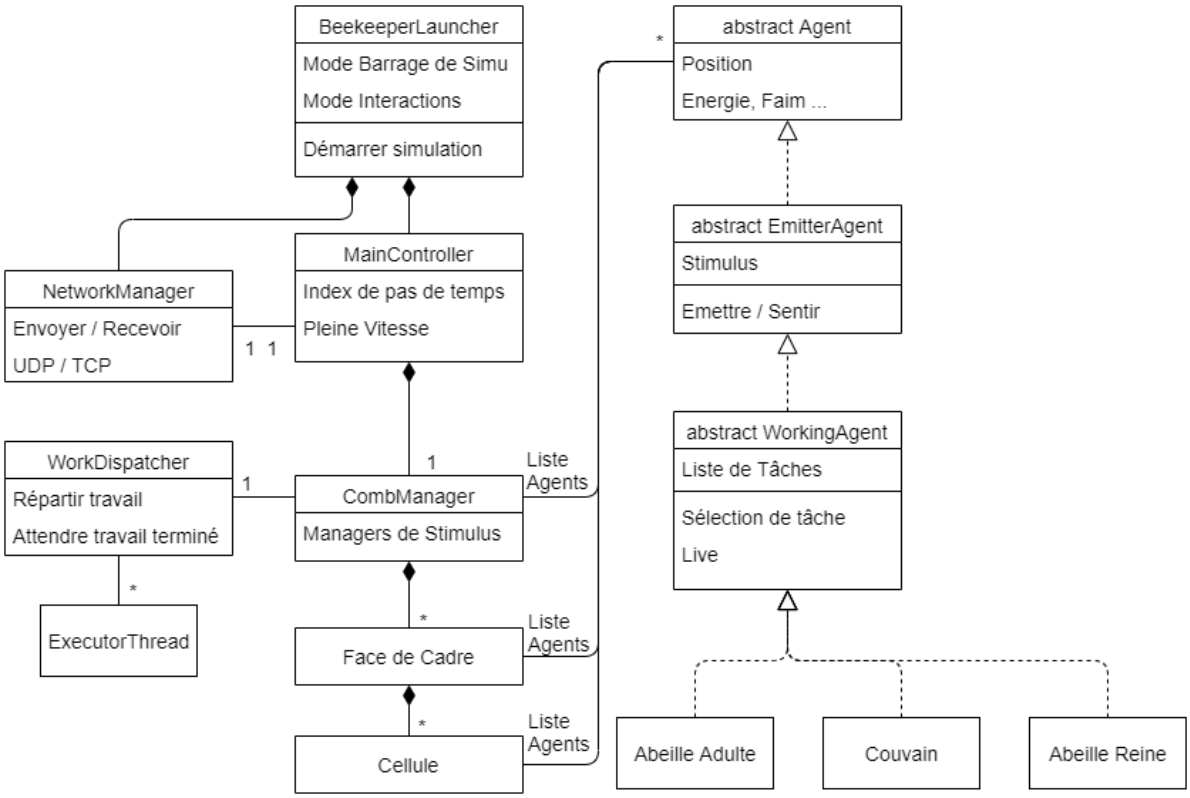


FIGURE 3.2 – Diagramme de classe esquissant l'architecture logicielle du simulateur.

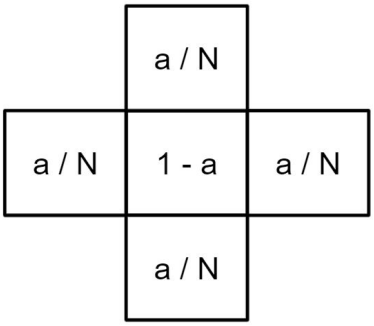


FIGURE 3.3 – Filtre utilisé pour connaitre la nouvelle valeur de l'intensité du stimulus évalué. Avec a la volatilité du stimulus évalué, et N le nombre de voisin, ici $N = 4$. Chaque pixel prend alors comme valeur la combinaison linéaire de sa valeur et de celles de ses voisins avec ce filtre pour coefficient.

pondérée de sa propre intensité et de celles de ses voisines, en suivant cette équation :

$$V0_{t+1} = p * ((1 - a) * V0_t + \sum_{n=1}^{N} Vn_t * \frac{a}{N}) \tag{3.1}$$

avec $V0_t$ la cellule évaluée à l'instant t, Vn ses voisines et N le nombre total de voisines. Nous utilisons aussi a et p deux paramètres de stimulus : la propagation dans l'espace a (volatilité) et dans le temps p (évaporation/amortissement). Ces paramètres nous permettent d'obtenir des comportements variés partageant les mêmes mécanismes. La figure 3.3 illustre cette équation sous la forme d'un filtre. Les valeurs d'évaporations sont alors utilisées sous la forme "par pas de temps". Pour mieux les contrôler, nous utilisons l'équation suivante pour les exprimer en demie-vie (durée après laquelle le stimulus aura perdu la moitié de son intensité), notion plus courante en biologie, puis les convertir en valeurs "par pas de temps" utilisables par l'algorithme :

$$k = \exp(\frac{-ln(2) * C}{\lambda}) \tag{3.2}$$

avec λ la demie vie en seconde, C le coefficient appliqué pour convertir des secondes en pas de temps, et k le coefficient à appliquer à chaque pas de temps pour obtenir une demie-vie de λ.

3.1.2 Architecture des Agents

Les mêmes objectifs en tête, lisibilité et facilité d'ajouts, nos agents ont été pensés en niveaux d'abstraction successifs. Au plus haut niveau nous trouvons la classe abs-

traite *Agent*, qui regroupe l'âge, la position, l'énergie, la faim, une fonction abstraite *live* (vivre), 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 l'algorithme de sélection de tâches présenté Chapitre 2 et détaillé section 3.1.4. La Figure 3.2 présente aussi l'architecture des agents. 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 sélection 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 implémentée différemment par nos différentes implémentations de *WorkingAgent*. Nous avons pour l'instant 3 classes implémentant *WorkingAgent*, les classes d'abeille adulte *AdultBee*, de couvain *BroodBee*, et de reine *QueenBee*. Chacune de ces implémentations ajoute différentes tâches à sa liste de tâches (que nous allons décrire Section 3.2), et implémente la fonction d'avancée du métabolisme.

Ainsi, dans les plus hautes couches d'abstraction, les différents composants ont des références *Agent*, ce qui améliore la modularité du programme, facilitant l'ajout de nouvelles implémentations de *WorkingAgent* à l'avenir.

3.1.3 Pas de temps et Multi-Thread

Afin d'accélérer nos simulations pour itérer plus confortablement dessus, nous avons mis en place un système de parallélisation, aussi appelé *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 d' *ExecutorThread*, de leur répartir le travail, 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 *MainController*, le *CombManager* récupère et combine si nécessaire les différents groupes d'agents à faire vivre ensemble. Il envoie alors ces ensembles au *WorkDispatcher* qui redirige la liste sur un *ExecutorThread* disponible. Chaque *ExecutorThread* va alors appeler la fonction *live* de chacun des agents de sa liste. La Figure 3.4 illustre ces échanges.

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

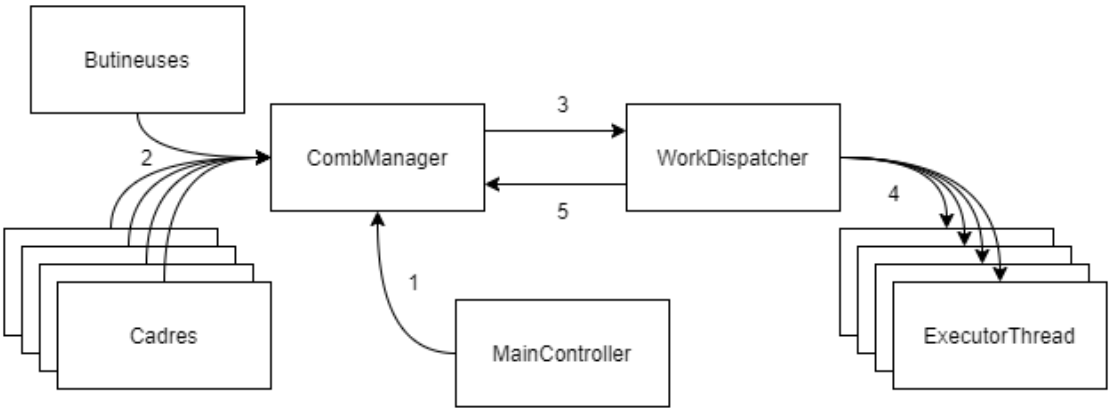


FIGURE 3.4 – Sequencement de la partie d'un pas de temps concernant les agents. 1- Le *MainController* envoie le signal au *CombManager* de faire vivre les agents d'un pas de temps. 2- Celui-ci va alors récupérer les listes d'agents présents sur les cadres ainsi que celle des butineuses. 3- Il réarrange ensuite ces listes en rassemblant les agents ne pouvant pas vivre en concurrence, puis envoie ces listes aux *WorkDispatcher*. 4- Ce-dernier va ensuite envoyer ces listes à des *ExecutorThread* et en instancier de nouveaux s'il le faut. Ils vont alors, chacun en concurrence, parcourir leur liste d'agents et les faire vivre chacun leur tour. 5- Enfin, lorsque le *WorkDispatcher* détecte que tous les threads ont terminé de faire vivre leurs agents, il le signal au *CombManager* qui rend alors la main au *MainController*.

toutes les abeilles qui sont sorties de la ruche sur ce pas de temps, et une liste contenant celles qui sont rentrées. 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.

Ensuite, lorsque tous les agents ont vécus, et que les déplacements entre cadres ont été réalisés, le *CombManager* rend la main au *MainController* qui se charge alors de terminer le pas de temps. Dans le mode "Barrage de Simulations", le pas de temps suivant est exécuté juste après, de la même manière que le précédent. En revanche, en mode "Interactions", le *MainController* va attendre une seconde (dans le cas où un pas de temps correspond à une seconde, ce qui est notre cas), en en déduisant le temps de calcul du pas de temps courant. Ainsi, nous respectons notre simulation en temps réel où un pas de temps représente une seconde réelle, et l'assurant précisément tant que la durée de calcul d'un pas de temps ne dépasse pas la seconde. Dans ce cas théorique le *MainController* enchainerait directement avec le pas de temps suivant, sa rapprochant ainsi autant que possible de notre temps réel voulu. Ce temps réel est nécessaire pour permettre à l'utilisateur d'interagir avec les agents sans créer de biais dans la simulation. Lorsque l'utilisateur décide d'accélérer le temps pour observer l'état futur de la colonie, le *MainController* cesse de faire ces pauses entre les pas de temps, comme dans le mode "Barrage de Simulations", ce qui permet de grandement accélérer les calculs, mais interdisant toutes interactions (ainsi, simuler 80 jours, au lieu de prendre quelques mois, ne prends que quelques minutes).

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, et peupler l'Activité racine d'Actions et/ou d'Activités.

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

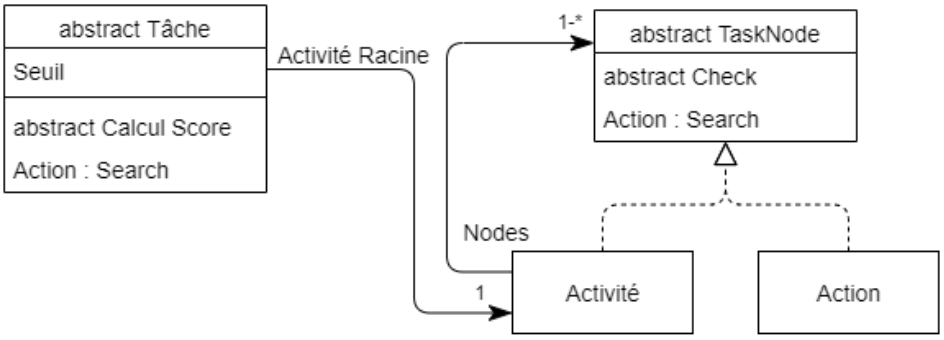


FIGURE 3.5 – Diagramme de classe de l'architecture logicielle de Tâche.

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 3.5 un diagramme de classe de cette architecture. La fonction *search* renvoi toujours une *Action*, et permet d'interroger récursivement toute la subsomption hiérarchique. La fonction *search* de la classe Action renvoi l'instance d'*Action* sur laquelle elle est appelée. En revanche, une *Activité* possède une liste de *TaskNode*, permettant la mise en place de la subsomption hiérarchique. Dans la fonction *search*, l'*Activité* va interroger une par une (dans l'ordre de priorité de la subsomption) la fonction *check* de tous ses nœuds. Si l'un d'eux valide son *check*, l'*Activité* appelle alors le *search* du nœud validé, continuant la recherche en profondeur si c'est appelé sur une Activité, ou mettant ainsi fin à la recherche lorsqu'appelé sur une Action.

Ainsi, pour qu'un agent puisse récupérer l'*Action* à effectuer de sa *Tâche* sélectionnée, il lui suffit d'appeler *search* sur l'*Activité* racine de la *Tâche*. Récursivement, la recherche va descendre dans la subsomption dans les blocs aux conditions validées. Dès qu'une *Action* est trouvée, sa fonction *search* la renvoie et 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 par l'agent.

Pour conclure, lors d'un pas de temps, le contrôleur principal demande au manager des cadres de faire vivre tous les agents. Ce dernier parcourt alors toutes ses faces de cadres et récupère tous leurs agents, afin de les envoyer au gestionnaire de *threads* pour une exécution parallèle. Une fois tous les agents mis à jour, les agents entrant et sortant de la

ruche sont transférés. Si le contrôleur principal a demandé d'enregistrer le tour, alors une nouvelle mécanique s'enclenche. Le manageur des cadres récupère à nouveau la liste de tous les agents, et leur demande de décrire en une ligne leur état : numéro d'identifiant, 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'être écrites dans un fichier, permettant une trace de la simulation, pour utilisations ultérieures en analyse de résultat.

3.2 Modélisation de la Colonie d'Abeilles

3.2.1 Modélisation des Agents "Abeille Adulte"

Dans notre colonie virtuelle, les adultes (qui sont désormais des agents) 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. Les détails de la biologie de l'abeille, des phéromones en jeu et leurs effets sur l'auto-organisation sont décrits dans le chapitre de contexte biologique, au début de ce manuscrit. Cette rétroaction permet de réguler le nombre de nourrices 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 3.6. Dans les colonies réelles, les butineuses échangent très régulièrement des phéromones avec les receveuses, qui sont chargées 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 un vecteur majeur de l'effet rajeunissant des butineuses sur les nourrices. Or, comme nous ne simulons pas de tâches de receveuses dans cette itération du modèle, nous nous attendons à ne presque pas observer ce mécanisme.

L'Hormone Juvénile (HJ) représente directement la physiologie de nos agents abeilles adultes, ce que nous appelons l'âge physiologique en référence au polyéthisme d'âge que nous observons dans les colonies réelles. Un agent avec très peu d'HJ ($HJ < 0.5$), sera capable de nourrir le couvain mais incapable de butiner. À l'inverse, un agent avec un fort taux d'HJ ($HJ > 0.5$), sera capable de butiner mais incapable de nourrir le couvain

(ces mécanismes sont décrits le chapitre de contexte). On parle donc de nourrices lorsque nous considérons les agents physiologiquement jeunes, et de butineuses pour les agents physiologiquement âgés. Nous avons ajouté un paramètre à nos agents, leur développement ovarien qui est toujours nul sauf la reine pour qui il vaut 1. Cette variable n'est pas utilisée dans cette itération de l'implémentation, mais permettra de simuler les ouvrières pondeuses que nous retrouvons dans la réalité, lorsque les phéromones de la reine ne parviennent plus à certaines ouvrières en périphérie de la colonie et que ces dernières commencent à pondre.

Dans nos lectures, nous avons pu apprendre que les abeilles d'hiver pouvaient 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 (celui qui est utilisé pour déterminer la mort de vieillesse) augmenter trente fois plus vite qu'un agent à l'intérieur de la colonie.

Nos agents sont placés sur le cadre et savent que la sortie de la ruche 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. En effet, ce mécanisme passionnant fait déjà l'objet de grandes quantités de recherches, et nous nous concentrons ici sur l'intérieur de la ruche. À l'avenir, une simulation plus poussée des mécanismes de fourragement, sélection des sources de nectar, recrutement etc. pourraient être ajoutés au modèle (nous avons travaillé sur un modèle de butinage [39] en parallèle de ces travaux, qui pourra tout à fait être intégré par la suite). Pour l'instant, les butineuses sortent de la ruche, attendent simplement 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. La durée de ce butinage est pour l'instant la même systématiquement, pour tous les agents, et correspond à une valeur moyenne des temps de butinages observés pour des butineuses réelles.

3.2.2 Modélisation du Couvain

Nous avons modélisé les trois étapes majeures de la vie du couvain : œuf, 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 phéromone, la

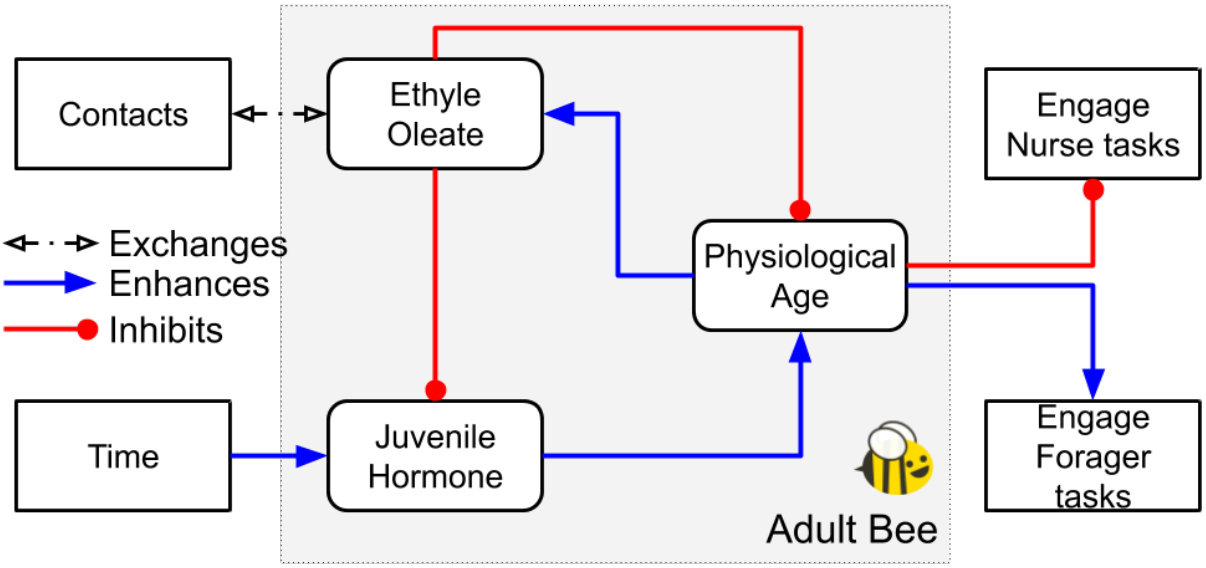


FIGURE 3.6 – Modélisation simplifiée des effets physiologiques responsables de l'auto-organisation.

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

Les phéromones émisent par les agents du couvain le long de leur vie vont altérer la physiologie des agents adultes. Elles sont échangées lorsqu'un agent adulte passe sur la case d'un agent couvain, et en plus grande quantité lorsqu'un agent effectuant la tâche de nourrice vient déposer de la nourriture à un agent larve, du fait du contact prolongé avec celle-ci.

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 ne présentent

pas de stimulus déclencheur direct comme nous avons pu en discuter dans le chapitre précédent. Nous avons donc recours 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, ou, pour le butinage, si l'agent n'a pas suffisamment d'énergie pour survivre aux vols aller et retour. Ensuite, nous utilisons l'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 (La description des modèles à seuils se trouve Section 1.1.2). Le débattement 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. Ces tâches ne sont pas motivées, elles possèdent des stimulus déclencheurs : la faim et l'énergie. La tâche de repos est prioritaire, son score est donc autorisé à dépasser 0.8. En effet, si l'énergie d'un agent devient négative, il meurt [1]. Le couvain possède aussi trois tâches très simple, une pour chacune de ses étapes de développement. Ces différentes tâches altèrent seulement leurs émissions de phéromones (en effet, le seul rôle du couvain est de se nourrir). La reine possède une tâche lui permettant de pondre. La Table 3.1 présente un récapitulatif de toutes les tâches implémentées.

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 3.6.

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 hypothèses, fixé des paramètres arbitrairement,

1. Dans cette version du modèle nous avons fait le choix de ne représenter la nourriture qu'au plus simple. Ainsi, la mort par sous alimentation ne fait pas partie de cette itération. L'importance des butineuses en est donc fortement réduite : même si 100% de la colonie s'occupe du couvain, il y aura toujours suffisamment de nourriture. Ajouter la nourriture, détailler ses mécanismes de distribution et introduire son mécanisme de collecte est une des perspectives privilégiées pour la suite.

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 (et motivées)	
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

dans le but de retrouver d'autres "macro-paramètres" émergents. Les deux points clés 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. Le premier, l'équilibre en EO (EO_{Eq}), qui représente le moment où un agent a suffisamment d'EO pour parfaitement compenser son vieillissement, et donc son émission d'HJ. L'autre point d'équilibre, cette fois en 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

Pour l'instant, les quantités absolues des différentes substances ne sont en rien liées à la réalité, nous avons donc pu choisir arbitrairement les quantités et points d'équilibres. Il sera peut être intéressant par la suite de correspondre aux quantités relevées sur les abeilles réelles, mais cette approche ne présentait pas d'intérêt, par rapports à la difficulté apportée, pour cette première version. 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, permettant à l'EO de s'accumuler, le faisant potentiellement

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 pertinentes, 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 avec la fonction linéaire :

$$EO_{em} = (HJ - HJ_{Eq}) * EO_{Evap} \tag{3.3}$$

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 à 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 pourrions 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 :

$$HJ_{red} = (EO - EO_{Eq})^n * HJ_{Incr} \tag{3.4}$$

avec n un coefficient dont nous allons nous servir pour modeler l'intensité de l'effet de réduction d'HJ. Avec $n = 1$ (Figure 3.7a), notre fonction est linéaire, nous nous servirons de cette fonction comme base de comparaison. Lorsque $n > 1$, on obtient une fonction quadratique (Figure 3.7b). L'effet rajeunissant de l'EO est diminué sous EO_{Eq} et amplifié au delà. 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 3.7c). De la même manière, avec une racine, l'intensité des effets de l'EO est réduite.

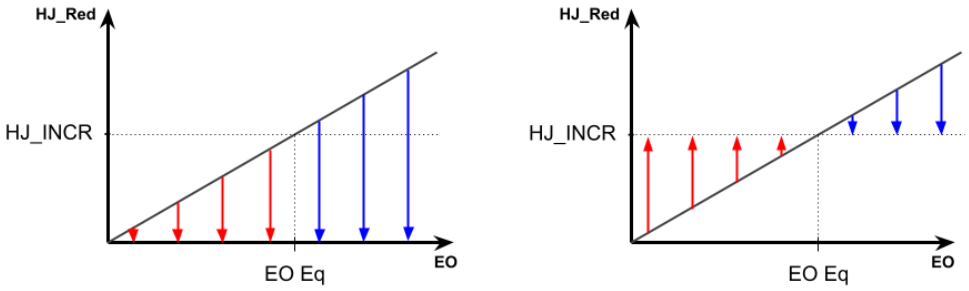
Pour illustrer, nous pouvons aussi imaginer que l'abeille vieillit naturellement rapidement. Mais, cet état d'équilibre est altéré par l'EO, nous pouvons donc visualiser ceci comme l'abeille étant attachée par un ressort à ce point d'équilibre. La rigidité de ce ressort peut alors être interprétée comme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

Nous n'avons pas trouvé de moyen de fixer mathématiquement n, ni dans la littérature, ni même après mûres réflexions : il a donc été trouvé expérimentalement, nous y viendrons dans la partie suivante, concernant la calibration.

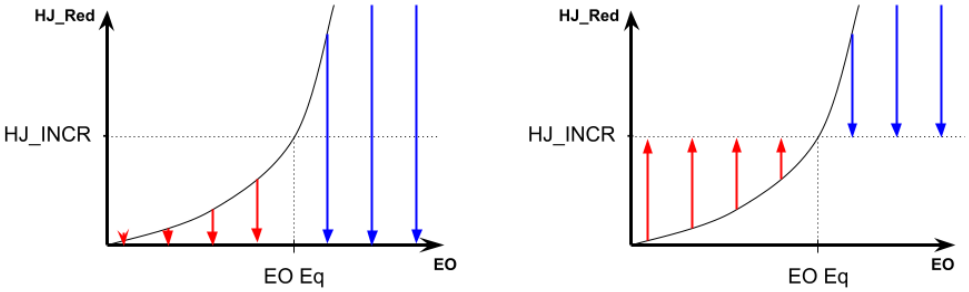
3.3.3 Quantités Émises par les Agents Larves

Un autre point clé de ce modèle est la capacité des agents larves à influer directement sur 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.

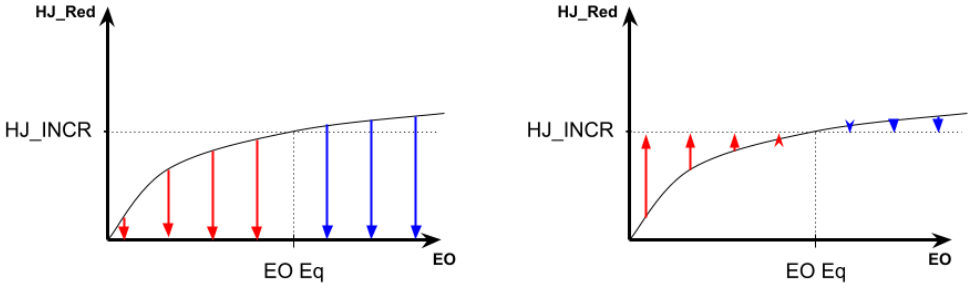
La quantité d'EO émise par chaque larve est donc importante, car elle est censée permettre à l'abeille de rester jeune, et même de rajeunir. Nous avons donc commencé par placer cette émission à EO_{Evap} (Voir eq. 3.3). De plus, les contacts étant assez bref, l'émission d'EO des larves doit permettre à une nourrice sollicitée de rester jeune malgré quelques temps de trajets. Ce dernier point porte une incertitude liée à l'émergence de ce comportement : la répartition des larves sur le cadre, les temps de trajets des nourrices



(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 3.7a), on observe à gauche que l'EO a moins de puissance sous EO_{Eq}, et beaucoup plus au dessus. A droite, on voit donc que le vieillissement (lorsque $EO < EO_{Eq}$) ainsi que le rajeunissement ($EO > EO_{Eq}$) sont plus intenses.



(c) Lorsque la fonction de réduction est **une racine** (exposant < 1) plutôt que linéaire (Fig 3.7a), on observe à gauche que l'EO a plus de puissance sous EO_{Eq}, et beaucoup moins au delà. A droite, on voit donc que le vieillissement ($EO < EO_{Eq}$) ainsi que le rajeunissement ($EO > EO_{Eq}$) sont plus doux.

FIGURE 3.7 – 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, à droite elles indiquent la vitesse de vieillissement (en rouge lorsque l'agent vieillit, en bleu lorsqu'il rajeunit).

entre les larves qui accepteront de la nourriture, l'aléatoire dans le trajet même que va choisir la nourrice 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. En effet, nous savons qu'en dessous les larves n'auront pas le pouvoir d'empêcher les nourrices de vieillir et de partir butiner par la suite. La question est donc de savoir à quel point il doit être 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 nourrices par rapport aux butineuses sera globalement constante. Avec une émission trop importante, augmenter le nombre de larves va drastiquement augmenter le ratio de nourrices. Par exemple, lors d'un essai nous obtenions 50% de nourrice dans la population d'adultes pour 500 larves, mais presque 100% de nourrice pour 1000 larves. À 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 3.4). Ainsi, lorsque 500 larves maintiennent un ratio nourrices/butineuses correct mais faible, mais que 1000 retiennent trop de nourrices, 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. La méthodologie et les résultats de la calibration seront présentés et discutés dans le chapitre suivant.

Conclusion

Dans ce chapitre nous avons discuté de l'architecture logicielle du simulateur qui fait tourner notre première itération du modèle. Nos 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. Nous avons ensuite décrit l'implémentation concrète du modèle de sélection et d'interruption de tâche, à partir de ce que nous avons pu construire au chapitre précédent : les seuils pour la physiologie de l'agent, modéliser les tâches en Activité ainsi qu'en Action, parfois démotivante etc. Nous avons aussi décrit l'implémentation du modèle physiologique simplifié de l'abeille adulte, ses glandes, hormones, phéromones et leurs influences mutuelles ainsi que sur le comportement de nos agents. Après calibration de certains paramètres émergents difficile à pré-calculer ou à estimer, nos agents devraient être capables de se répartir le travail sans contrôle central, et de manière dynamique, en s'adaptant aux changements d'environnement et de populations. 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'observations de colonies réelles.

Chapitre 4

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

Après avoir modélisé puis construit notre modèle multi-agents et son simulateur, il est désormais temps d'en observer, interpréter puis valider (ou non) les résultats qu'il nous donne. Nous allons donc définir ce que nous attendons de ce modèle et de sa calibration. Nous discuterons ensuite des résultats obtenus puis les interpréterons, afin d'en sortir notamment des perspectives d'améliorations.

4.1 Hypothèses et Calibration

4.1.1 Hypothèses de Validation

Nous avons mis en place deux grandes familles de vérifications, afin de cerner le problème sous plusieurs approches. La première approche vise à vérifier les capacités d'auto-organisation de notre modèle, elle consiste donc en plusieurs simulations aux paramètres identiques mais dont nous avons fait varier la situation initiale. La seconde approche vise à vérifier si notre modèle de la colonie arrive à reproduire le comportement de colonies réelles. Afin de mieux préciser cette deuxième vérification, pour l'instant très large, nous nous sommes concentrés sur un cas particulier de la vie d'une colonie : l'adaptation de la colonie suite à une division. une division est une opération apicole qui consiste à séparer une colonie dont la population a dépassé un certain seuil, afin de créer deux colonies. L'apiculteur doit alors faire de son mieux pour bien répartir les différentes populations sur les deux colonies. Les deux nouvelles colonies ainsi crées doivent alors faire face à un changement rapide d'environnement, et donc s'auto-organiser, afin de survivre à cette épreuve.

Nous avons donc deux hypothèses principales :

— **H1** : Notre modèle de colonie d'abeilles virtuelle est capable d'auto-organisation.

— **H2** : Notre modèle est capable d'approcher les dynamiques de populations observées dans les colonies d'abeilles réelles.

Afin de valider **H1** nous avons simplifié le modèle de la colonie, afin d'obtenir un environnement plus stable, mettant mieux en valeur l'adaptation de nos agents. Ainsi, le cycle de vie n'était pas simulé : aucune naissance (et donc, pas de reine pour pondre), aucune émergence (larves devenant adultes), aucune mort de vieillesse. En revanche, les larves manquant de nourritures meurent de faim. Nous pouvions ainsi conserver un nombre d'ouvrières et de larves constant, faisant de ce fait mieux ressortir les différents ratio de population. Dans cette validation, la proximité avec la biologie de l'abeille n'est pas nécessaire, nous avons ainsi largement accéléré les phénomènes physiologiques de nos agents, ainsi que réduit le nombre d'agents pendant les simulations. Ceci nous a permis d'itérer plus confortablement. Valider **H1** consiste alors à observer un point d'équilibre dans les populations d'ouvrières, entre le nombre de butineuses et de nourrices. Il faut aussi que ce point d'équilibre dépende du nombre de larves présentes dans la colonie. Nous parlons alors de "ratio d'ouvrières par larve". Nous avons donc mis en place 5 scénario :

— **Scénario 1.1** : Répartition initiale aléatoire des âges avec 150 abeilles adultes et 150 larves.

— **Scénario 1.2** : 150 adultes et 150 larves mais toutes les abeilles adultes commencent très jeunes.

— **Scénario 1.3** : 150 adultes et 150 larves, mais toutes les abeilles adultes commencent âgées.

— **Scénario 1.4** : Répartition initiale aléatoire des âges avec 150 adultes et 50 larves.

— **Scénario 1.5** : Répartition initiale aléatoire des âges avec 150 adultes et 300 larves.

Afin de valider **H2**, nous avons mis en place une expérimentation dont nous parlerons bien plus tard dans la Section 7.1, dont une partie est dédiée à cette hypothèse. Nous avons montré des graphiques de populations de nos simulations à des apiculteurs de différents niveaux. Ils ont ensuite été invités à noter les évolutions de populations de 1 à 5, si elles étaient ou non, selon eux, cohérentes avec ce qu'ils auraient attendu d'une colonie réelle. Notre modèle est alors dans sa dernière version, avec cycle de vie et calibration au plus proche de la biologie. Comme énoncé plus tôt, les scénario présentés aux apiculteurs correspondaient à des colonies sortant tout juste de division, sans reine pondeuse [1] mais

1. En réalité la reine est présente dans la simulation, mais n'est autorisée à pondre qu'après avoir

avec réserve de couvain. Nous avons ensuite pu faire varier la répartition des âges de la population de la nouvelle colonie. Nous en avons ainsi fait 4 scénario :

- **Scénario 2.1** : Répartition initiale avec seulement des abeilles adultes très jeunes.
- **Scénario 2.2** : Répartition initiale avec 50% d'abeilles adultes jeunes, et 50% d'abeilles adultes âgées.
- **Scénario 2.3** : Répartition initiale avec 100% d'abeilles adultes âgées.
- **Scénario 2.4** : Répartition initiale avec 50% d'abeilles adultes jeunes, et 50% d'abeilles adulte âgées. La reine commence à pondre 30 jours après la division, au lieu de 21.

Tous ces scénario démarrent avec 500 adultes et 500 agents couvains, répartis uniformément en œuf, larves, et nymphes. Le délai avant la ponte de la nouvelle reine a été choisi pour être dans la tranche basse des âges observés en réalité, afin de rester réaliste tout en proposant un cas favorable, pour aider la survie de la colonie. Pour des raisons de temps de passage seulement les deux premiers ont été intégrés dans l'expérimentation. En effet, ces deux scénario présentent le plus d'intérêts : le troisième est très court et conduit à la perte de la colonie, et le quatrième n'est qu'une légère modification du second mais entrainant lui aussi la perte de la colonie. L'auto-organisation est donc beaucoup moins visible.

De plus, il est classique dans la réalité d'obtenir une colonie avec une immense majorité de très jeunes adultes. En effet, si la nouvelle colonie est déposée proche ($<$ 1km) de son ancien placement, alors les butineuses retourneront à leur emplacement initial. Nous nous retrouvons ainsi avec une colonie uniquement composée de nourrices, n'ayant pas encore réalisé leur vol d'orientation. Si la nouvelle ruche est placée loin de sa position initiale avant division, alors les butineuses resteront dans cette nouvelle ruche.

4.1.2 Calibration

La calibration du modèle pour les expériences visant à valider **H1** a été plus rapide et plus simple que la calibration du modèle complet, car nous ne cherchions pas à obtenir de résultat proche de la biologie, mais seulement retrouver l'auto-organisation. La méthode a tout de même été plus ou moins la même. En bon système complexe, nous avons à notre disposition une grande quantité de paramètres s'influant mutuellement de manières émergentes. Les deux principaux points de la calibration concernaient les phéromones émises

attendu un certain temps, correspondant au temps avant ponte d'une reine réelle après une division.

par nos agents, ainsi que leurs effets sur ces mêmes agents. Notre objectif de calibration, comme détaillé plus tôt section 3.3.4, est de retrouver un équilibre des proportions entre les populations de couvain et de nourrices, avec le reste de la population se plaçant en butineuses. On remarque alors que l'objectif de la calibration est déjà, au final, de valider **H1**, le modèle est alors calibré pour répondre à l'hypothèse initiale. La question est en réalité légèrement différente : nous essayons de voir si l'espace offert par l'ensemble des paramètres du modèle nous permet d'atteindre une calibration pour laquelle le modèle réponds à l'hypothèse. Ce qui pose d'ailleurs d'autres soucis d'ordres pratiques : pendant la calibration, si nous n'arrivons pas a obtenir les résultats que nous souhaitons, est-ce parce que le modèle ne permet pas de les atteindre ou n'est-il seulement pas correctement calibré ?

Après une bonne quantité d'itérations sur le modèle complet, nous avons fini par positionner le coefficient n de l'intensité des phéromones à $n = \frac{1}{3}$ (équation 3.4), et le coefficient k de la quantité de phéromones émises par les agents du couvain à $k = 2$ (section 3.3.3). Ces paramètres ont été ajustés pour une simulation démarrant avec 500 larves, 500 abeilles et une reine, sur deux faces de cadres. Nous avons réalisés quelques essais nous faisant penser que la calibration reste valide en changeant le nombre d'agents en présence ainsi que leur répartition spatiale, mais la question est vaste et pas si facilement répondue, plus de travaux devront donc être réalisés dans ce sens.

Nous allons pouvoir désormais aborder nos résultats, sous la forme de courbes générées à partir de fichiers eux-mêmes générés pendant chaque simulation.

4.2 Résultats

4.2.1 Modèle à environnement constant

Le modèle à environnement constant, réalisé principalement pour vérifier la validité d'**H1**, comporte cinq scénario détaillés plus tôt. Les résultats de ces scénario sont décrits Figure 4.1, chacun ayant été joué cinq fois, ainsi les écarts-types sont aussi présent sur la figure, on y observe :

— **Scénario 1.1 (S1)** : Répartition initiale uniforme des physiologies et ratio adultes/larves de 1. Au lancement de la simulation, 50% des adultes de la colonie sont des nourrices, du fait de notre initialisation. Nous observons ici une convergence en quelques milliers de pas de temps vers 60% d'agents effectuant un travail de nourrice, et donc

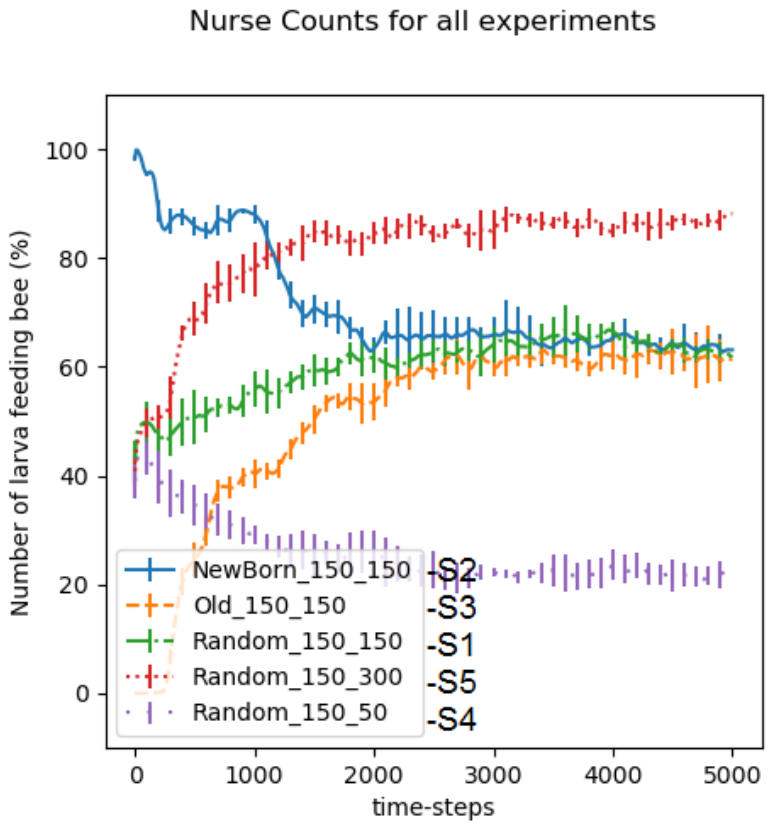


FIGURE 4.1 – Proportions de nourrices pour les 5 scénario visant à valider **H1**. Chaque scénario a été simulé 5 fois, les écarts-types sont visibles en barres verticales sur chacune des courbes.

Scénario	Population Initiale	Ratio Adultes par Larve	Équilibre
1.1	uniforme	1 pour 1	60%
1.2	jeune	1 pour 1	60%
1.3	âgée	1 pour 1	60%
1.4	uniforme	1 pour 3	20%
1.5	uniforme	2 pour 1	85%

TABLE 4.1 – Récapitulatif des différents scénario, leurs ratio adultes par larves ainsi que leur valeur finale d'équilibre, en proportion de nourrices dans la population d'adultes de la colonie virtuelle.

environ 40% de butineuses.

— **Scénario 1.2 (S2)** : Répartition initiale des physiologies avec uniquement de jeunes adultes et ratio adultes/larves de 1. Au lancement de la simulation, 100% des adultes de la colonie sont des nourrices, du fait de notre initialisation. Nous observons ici la même convergence en quelques milliers de pas de temps vers ce même équilibre, 60% de nourrices.

— **Scénario 1.3 (S3)** : Répartition initiale uniforme des physiologies et ratio adultes/larves de 1. Au lancement de la simulation, 0% des adultes de la colonie sont des nourrices, toutes sont butineuses, du fait de notre initialisation. Nous observons à nouveau un équilibre à 60% de nourrices en un peu plus de 2000 pas de temps.

— **Scénario 1.4 (S4)** : Répartition initiale uniforme des physiologies et ratio adultes/larves de 1 pour 3. Au lancement de la simulation, 50% des adultes de la colonie sont des nourrices, du fait de notre initialisation. Nous observons cette fois-ci un équilibre en un peu plus de 2000 pas de temps, mais à environ 20% de nourrices.

— **Scénario 1.5 (S5)** : Répartition initiale uniforme des physiologies et ratio adultes/larves de 2 pour 1. Au lancement de la simulation, 50% des adultes de la colonie sont des nourrices, du fait de notre initialisation. L'équilibre est atteint un peu avant 2000 pas de temps, mais se situe cette fois aux alentours de 85% de nourrices.

Le Tableau 4.1 reprend ces différents résultats de manière plus concise. Vous y trouverez, pour chaque scénario, le ratio adultes/larves ainsi que l'équilibre de répartition de travail atteint, donné en proportion de nourrices dans la population d'adultes de la colonie virtuelle.

Avant d'interpréter ces résultats, nous allons consulter ceux du modèle complet vis à vis d'**H1**, mais aussi de **H2**.

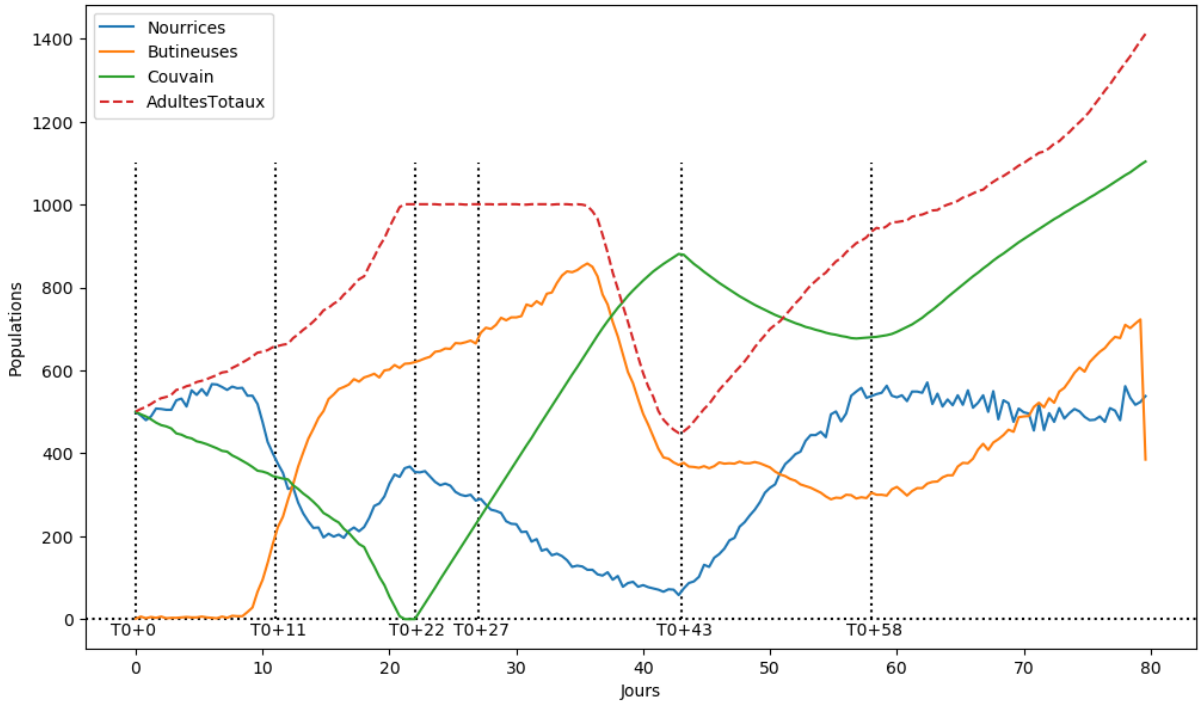


FIGURE 4.2 – Les différentes populations de la colonie après la division du Scénario 2.1.

4.2.2 Modèle Complet, Division et Cycle de vie

Les Figures 4.2 et 4.3 présentent graphiquement respectivement les résultats des scénario 2.1 et 2.2. Sous la forme de différentes courbes de populations au court du temps, ces figures nous renseignent sur la vie de nos colonies virtuelles juste après une division. Quelques indicateurs de temps ont été placés afin d'en faciliter la lecture : T0 indique le début de la simulation, le moment ou la division vient d'être réalisée. Par exemple, T0+11 indique le 11ème jour après la division. Nous y observons les populations de nourrices, de butineuses, mais aussi de couvain ainsi que la population totale des adultes de la colonie, soit la somme des populations de nourrices et de butineuses. Dans ces simulations, des agents naissent et meurent régulièrement, ce qui rend la validation de **H1** moins triviale : le ratio adultes par larves n'est pas constant. Voici nos résultats :

Scénario 2.1 (Figure 4.2) : Répartition initiale des physiologies avec uniquement de très jeunes abeilles adultes, départ avec 500 adultes et 500 agents de couvain (1/3 d'œufs, 1/3 de larves et 1/3 de nymphes). Par ordre chronologique, nous observons à T0+11 un important changement dans la répartition du travail, où le nombre de nourrices chute drastiquement au profit du nombre de butineuses. La quantité de couvain n'a de cesse

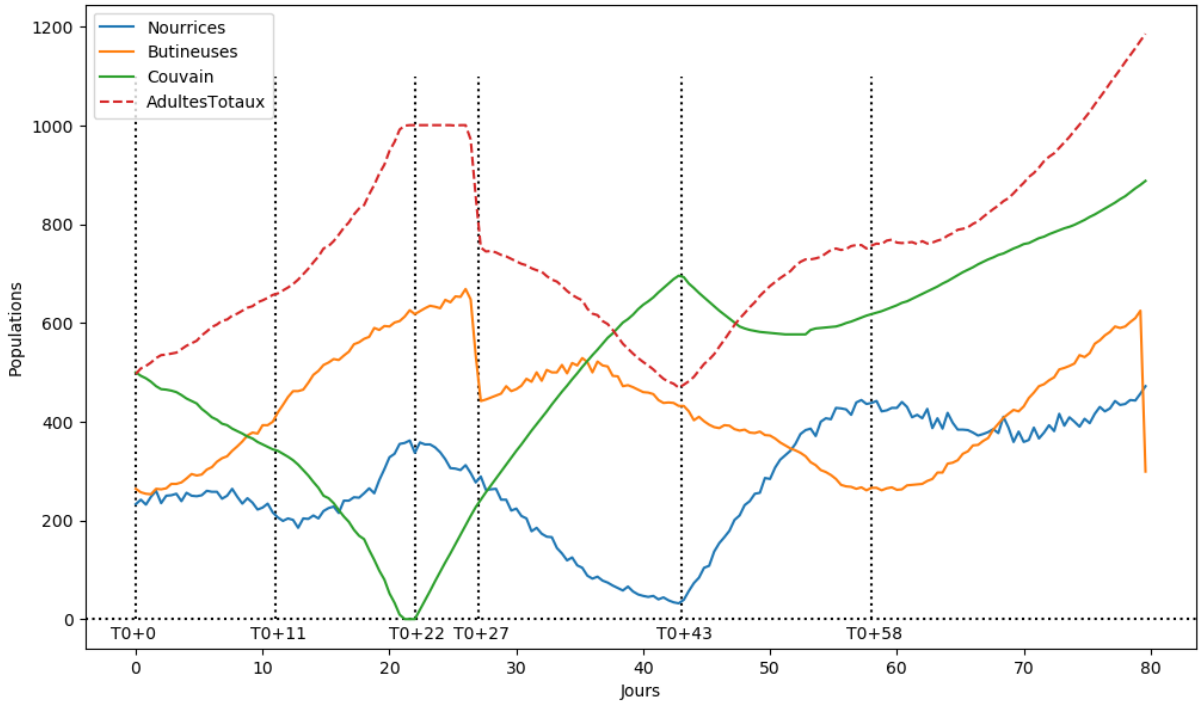


FIGURE 4.3 – Les différentes populations de la colonie après la division du Scénario 2.2.

de chuter jusqu'a T0+21 où il atteint zéro. La population totale cesse alors de croitre. À T0+22, la reine commence sa ponte, le couvain repart à la hausse, mais le nombre de nourrices continue de chuter. Nous observons ensuite une chute de la population totale : les butineuses commencent à mourrir de vieillesse. En effet, la reine vient seulement de recommencer à pondre. La population repart à la hausse à T0+43, lorsque les premières pontes de la reine émergent enfin, venant renforcer le faible contingent de nourrices. Enfin, à T0+58, nous observons à nouveau un transfert de répartition du travail : la population de nourrices qui croit alors très rapidement, va voir nombre de ses agents partir butiner. La population de nourrices se stabilise ensuite jusqu'à la fin de la simulation, 80 jours après la division, alors que le couvain augmente en permanence.

Scénario 2.2 (Figure 4.3) : Répartition initiale des physiologies uniforme, départ avec 500 adultes et 500 agents de couvain (1/3 d'œufs, 1/3 de larves et 1/3 de nymphes). Nous observons une population constante de nourrices, aidée par les émergences du couvain, et une population de butineuses croissante. La population de nourrices décrit alors un pic à T0+22, au moment où les dernières nymphes émergent et alors que la reine commence à pondre. À T0+27, les butineuses déjà présentes au début de la simulation meurent déjà de vieillesse. La progression suit ensuite de très près la chronologie du scénario précédent,

Scénario	Moyennes	Ecarts-types	Notes Individuelles (1-5)														
2.1	3.93	0.77	4	3	3	4	4	2	5	4	4	4	4	5	4	5	4
2.2	3.93	1.06	4	4	4	4	5	4	1	4	5	2	4	5	4	5	4

TABLE 4.2 – Résultats de l'expérimentation auprès de quelques apiculteurs concernant la cohérence des variations de populations de nos simulations par rapports aux colonies réelles.

avec une population globale légèrement plus faible.

Nous avons ensuite pu présenter ces deux courbes à des apiculteurs au cours d'une journée d'expérimentations organisée entre l'UBO, l'IMT-Atlantique et le GDSA29. Le Tableau 4.2 récapitule les réponses qu'ont donnés les participants à la question *" L'évolution de la population de la colonie vous semble-t-elle cohérente avec la façon dont la division a été réalisée ? "*, tour à tour pour les Figures 4.2 et 4.3. "La façon dont la division a été réalisée" concerne les quantités de populations présentes dans la colonie après la division, et est un exercice auxquels les apiculteurs sont très sensibilisés : il en va de la survie de la colonie. Les deux scénario obtiennent alors la même note moyenne de 3.93 ainsi que la même médiane à 4, mais les écarts-types diffèrent, respectivement 0.77 et 1.06.

4.3 Interprétations des Résultats

Nous allons désormais comparer nos résultats à nos hypothèses, évaluer les écarts ainsi que les biais, et parler d'éventuelles perspectives d'améliorations.

Nous nous intéressons ici à notre première hypothèse, sobrement appelée **H1**, que nous rappelons :

H1 : Notre modèle de colonie d'abeilles virtuelle est capable d'auto-organisation.

Une première version du modèle, appelée version "à environnement constant" (car elle conserve les niveaux de populations sur toute la durée de la simulation) nous sert principalement à vérifier cette hypothèse. Ces cinq scénario, manipulant les conditions initiales ainsi que les rapports de populations, permettent de faire ressortir les capacités d'adaptation de notre modèle. Ainsi, comme récapitulé dans le Tableau 4.1, on note que la répartition initiale de la physiologie de nos agents n'influe pas sur l'équilibre de la répartition des tâches entre eux. Le ratio adultes / larves en revanche semble être le facteur directeur de l'équilibre. Plus il y a de larves par agents adultes, plus ces derniers

seront nombreux à effectuer un travail de nourrice. Ainsi, nous pouvons dire que notre modèle valide **H1** : nos agents se répartissent correctement les tâches selon les besoins de l'environnement, ici plus ou moins de larves.

Nous pouvons nous intéresser à ce qu'apporte les scénario 2.1 et 2.2 en ce qui concerne cette première hypothèse. En effet, la calibration "temps réel", sans les accélérations physiologiques présentent dans le modèle environnement constant, ainsi que le cycle de vie apportent des nouveautés. La répartition du travail est toujours visible, mais le cycle de décision des agents est plus long. En effet, une abeille nourrice décidant de devenir butineuse va devoir attendre quelques jours que ses glandes soient en état (attendre que son niveau d'HJ soit assez élevé). Ce délai influe sur la réactivité des agents, mais représente plus fidèlement la réalité. Les scénario 2.1 et 2.2 dans leurs figures respectives 4.2 et 4.3, présentent tout deux à T0+22 des répartitions de populations identiques, avec environ 350 nourrices et 600 butineuses, alors que les conditions initiales sont différentes. Ceci est un pas vers la vérification d'**H1** sur le modèle complet. Les suites de ces deux scénario diffèrent car la reine ajuste sa vitesse de ponte en fonction de la population de la colonie, et que dans le scénario 2.2, les butineuses meurent beaucoup plus tôt, ralentissant plus rapidement la vitesse de ponte.

Il est toutefois étonnant de constater que les nourrices continuent de devenir butineuses entre T0+22 et T0+43, alors que la population du couvain augmente. Les premières larves ont besoin de soins aux alentours de T0+25 (après 3 jours en tant qu'œuf), il est donc très étonnant que la population de nourrices ne se stabilise pas. Une des raisons à laquelle nous pensons, et que nous avons déjà évoquée à plusieurs reprises dans ce manuscrit, est l'absence des receveuses. Les receveuses agissent comme un intermédiaire systématique entre les butineuses et les nourrices, permettant ainsi aux butineuses, plus âgées, de mieux ralentir la production d'HJ de leurs jeunes compatriotes. Sans cet effet, la proportion de nourrice par rapport aux butineuses peut être très faible, alors qu'en réalité, il n'est jamais observé de colonie avec si peu de nourrices.

Nous allons désormais pouvoir nous intéresser à notre deuxième hypothèse, **H2**, qui énonce :

H2 : Notre modèle est capable d'approcher les dynamiques de populations observés dans les colonies d'abeilles réelles.

D'après les retours donnés par les apiculteurs, avec une médiane à 4 sur 5 lorsque nous leur avons demandé de juger de la cohérence des évolutions de populations comparés à

leurs connaissances pratiques, nous pouvons dire qu'**H2** est validée. Pas parfaite, pour les quelques raisons que nous avons évoquées juste au dessus. Il nous a aussi été pointé que si les dynamiques semblent bonnes, les quantités ne sont pas du tout respectées. En effet, les colonies d'abeilles comptent régulièrement jusqu'à 50 000 individus, ce qui est bien loin de nos 1000 individus. Il sera donc intéressant, à l'avenir et maintenant que la dynamique est validée, d'essayer d'augmenter drastiquement le nombre d'individus, afin d'observer si la calibration ainsi que ses propriétés sont toujours valides ou si elles ne passent pas à l'échelle. La calibration ne fonctionne peut être qu'avec un nombre réduit d'agents. En effet, la répartition spatiale peut avoir un impact très important sur l'émergence des propriétés d'auto-organisation.

Lors de nos premiers essais préliminaires sur de très grand nombre d'agents, nous observions dans nos simulations un "embouteillage" d'abeilles, où les agents se bloquent tellement entre eux que leur navigation devient impossible. Il est peut être possible de régler ce soucis en gérant différemment les déplacements, mais il sera peut être nécessaire de changer d'approche quant à la spatialisation des agents dans la ruche et sur les cadres.

4.4 Perspectives d'Améliorations

4.4.1 Perspectives du Modèle de prise de décisions

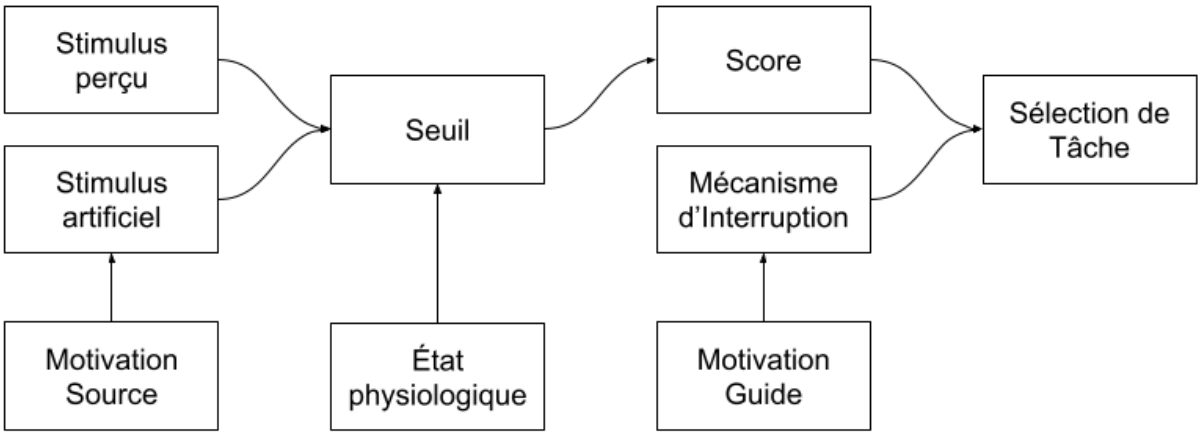
Après ces travaux sur le modèle de prise de décision présenté Chapitre 2, nous avons en tête quelques améliorations pour d'éventuelles nouvelles implémentations de celui-ci. La Figure 4.4a illustre le modèle de prise de décision tel qu'il est décrit dans ce manuscrit : un stimulus, qu'il soit perçu ou artificiel (représentant la Motivation Source), sert à calculer un score pour sa Tâche par le biais d'une fonction sigmoïde et de son seuil. Ce dernier représente alors l'état physiologique/physique de l'agent. Enfin, la sélection se fait soit en prenant en compte le score de la tâche, soit, pour une Tâche Motivée précédemment exécutée, c'est le mécanisme d'interruption, représentant la Motivation Guide transversale à l'agent, qui est pris en compte. Ainsi la Tâche ayant le score final, score ou motivation, le plus élevé sera sélectionnée.

Ce que nous proposons pour une version améliorée est de se passer de ce mécanisme d'interruption tel qu'il est présenté, en fusionnant les deux types de motivations au niveau du stimulus artificiel. La Figure 4.4b décrit ce nouveau fonctionnement : le calcul du score se fait soit via le stimulus perçu, soit, dans le cas de Tâche Motivée, via le

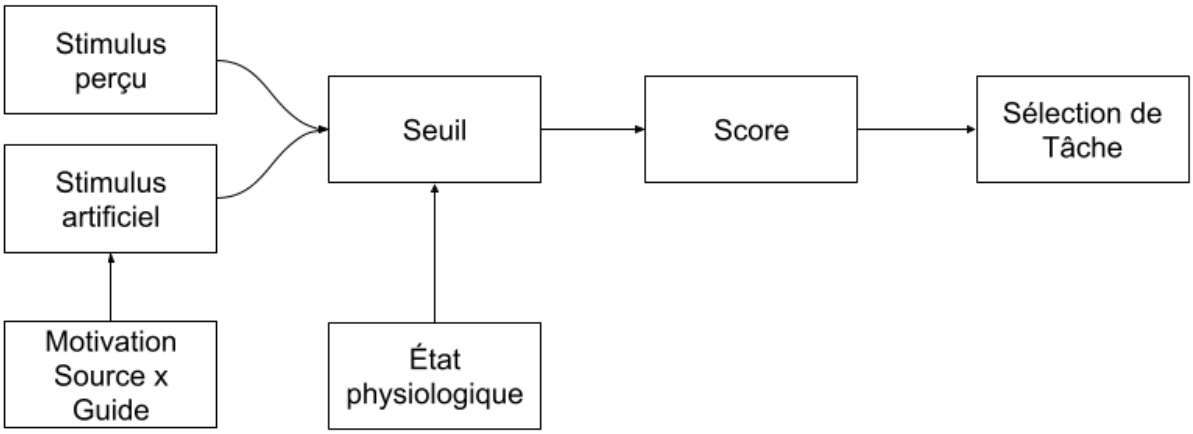
produit des Motivations Source et Guide de la Tâche. L'agent perd alors sa motivation interne transversale à toutes les tâches, et voit ses Motivations Guides être liées à chaque Tâche Motivée. Ainsi, le score d'une tâche représente les deux motivations, et contient déjà l'information que notre mécanisme d'interruption apportait, car le stimulus est alors une image des deux motivations que nous avons décrites. Il est alors nécessaire d'intégrer le concept d'Actions Motivantes, qui, à l'inverse d'Actions Démotivantes, remontent la Motivation Guide d'une tâche à chaque exécution. Agissant un peu à la manière d'une récompense, les Actions Motivantes seront sélectionnées afin de concerner les Actions qu'un agent réalise lorsqu'il réussi sa Tâche. La Motivation Guide devra aussi remonter lentement au fil du temps même si la Tâche n'est pas sélectionnée, propriété que nous retrouvons classiquement dans les modèles à seuils, à propos de seuils s'ajustant au fil de la simulation. Il sera alors intéressant d'observer l'importance des paramètres de baisses et de hausses de la motivation dans la répartition du travail, et d'en modifier les dynamiques en utilisant par exemple des incréments constants ou pourquoi pas une fonction exponentielle accélérant la baisse ou la hausse selon la fréquence d'appel de ces changements. Nous pensons qu'avec la Motivation Guide liée à chaque tâche, chaque agent aura un comportement plus cohérent, en évitant d'osciller entre deux tâches, ce qui devrait par la même occasion augmenter son efficacité. Cette nouvelle itération du modèle permet une intégration plus naturelle des concepts de motivations dans les modèles à seuils : en ne jouant que sur un stimulus artificiel nous n'avons pas besoin de modifier le fonctionnement du modèle à seuils, là où la version actuelle nous demande d'y intégrer un nouveau mécanisme.

4.4.2 Perspectives de la Simulation Multi-Agents

Il est possible d'enrichir le modèle de la colonie d'abeilles de beaucoup de manières différentes, afin de rapprocher nos résultats d'observations faites sur de réelles colonies. L'amélioration la plus prioritaire selon nous serait d'enrichir la simulation de la nourriture dans le modèle. La nourriture est pour l'instant ramenée par les butineuses, mais est aussi présente à l'intérieur de la colonie, dans quelques cellules de cadres agissant comme des sources infinies. Ainsi, nous avons pu nous intéresser pour l'instant à l'importance de sa distribution, plutôt qu'à sa collecte. Pour ceci, il sera nécessaire d'échelonner plus en détail les quantités de nourritures consommées par les larves et les adultes, mais aussi les quantités rapportées par les butineuses. Il sera aussi nécessaire d'ajouter les tâches de "receveuses" que nous avons décrit dans le Chapitre de Contexte, au début de ce



(a) Notre modèle de prise de décision tel qu'implémenté. La Motivation Source nous permet de créer un stimulus artificiel pour les Tâches n'ayant pas de stimulus déclencheurs évidents. La Motivation Guide permet d'interrompre la réalisation de cette tâche lorsque l'agent n'arrive pas à la réaliser correctement, via les Actions Démotivantes.



(b) Proposition pour une amélioration du modèle de prise de décision. Le mécanisme d'interruption disparait, et le stimulus artificiel pour les Tâches Motivées devient le produit des Motivations Source et Guide. Ainsi, le score devient l'image de la combinaison de ces deux motivations sans avoir recours à un mécanisme d'interruption tout en produisant les mêmes effets.

FIGURE 4.4 – Schéma du modèle de prise de décision tel que décrit dans ce manuscrit (a), sous lequel nous présentons une version améliorée (b).

manuscrit. De plus, ces tâches de receveuses recoupent une hypothèse que nous avons déjà énoncé dans ce manuscrit, qui est qu'elles sont les principales responsables de l'impact rajeunissant des abeilles plus âgées sur les jeunes adultes. Elles jouent en effet un rôle très important dans la propagation de phéromones dans la colonie, en faisant le pont entre les âges. De la même manière il serait intéressant d'intégrer un module gérant le butinage de manière plus poussée que la version minimaliste de cette itération. Nous pourrions alors observer les impacts qu'ont les changements de l'environnement extérieur de la ruche sur la répartition du travail à l'intérieur de celle-ci.

Il serait aussi possible d'intégrer la gestion de la température, au fil des journées mais aussi des saisons, qui est un point clé de la colonie, dont nous avons aussi parlé dans le chapitre de Contexte. Nous pourrions alors intégrer la récolte d'eau par les butineuses, qui sert à la régulation de la température. De plus, les abeilles d'hiver présentent des caractéristiques très intéressantes qu'il serait intéressant de retrouver par la simulation.

Autre point qu'il serait intéressant d'aborder est l'ajout du parasite Varroa Destructor dans la simulation. Varroa est un acarien parasite qui s'attaque aux larves et nymphes et provoquent des malformations lorsque ces abeilles deviennent adultes, mettant ainsi en danger toute la colonie [31]. Il serait donc particulièrement intéressant d'utiliser la simulation pour essayer de reproduire les attaques Varroa, pour éventuellement essayer de détecter des points clés dans les procédés, et en sortir des propositions pour aider les apiculteurs et biologistes dans leur lutte contre ce parasite particulièrement vorace.

Couplé à un butinage plus poussé, nous pourrions de la même manière et pour les mêmes raisons ajouter dans l'environnement extérieur des facteurs de stress : variations brutales de température limitant le butinage, monocultures limitant la disponibilité de nectar et pollen, pesticides et bien d'autres.

Dans tous les cas, étoffer la simulation pourra nous permettre de simuler plus de scénario différents, nous permettant de réitérer l'expérimentation où nous demandons l'avis d'apiculteurs sur des scénario plus variés, amenant donc à des résultats plus précis. Nous pourrions de la même manière proposer ces interprétations de résultats de simulation à des biologistes de l'abeille, dont l'attention se portera naturellement sur d'autres aspects de la colonie, apportant un regard complémentaire à celui des apiculteurs.

Conclusion

Dans ce chapitre nous avons abordé les enjeux et méthodes de calibration des paramètres du modèle multi-agents. Nous avons ensuite observé les résultats qu'il a pu produire dans différentes conditions, et différentes itérations du modèle. Nous avons alors pu valider nos deux hypothèses : notre modèle multi-agent est capable d'auto-organisation et il produit des résultats cohérents avec les observations de colonies d'abeilles réelles. Forts de ces résultats, nous avons pu clore en énonçant quelques perspectives pour la suite de ces travaux, tant au niveau du modèle multi-agents qu'au niveau du modèle de prise de décision. Voici qui conclut la première partie de ce manuscrit, orienté vers la simulation multi-agents, nous allons désormais aborder la partie concernant la visualisation, les interactions et les environnements immersifs. De cette manière le chapitre suivant réalise un état de l'art de ces domaines, mais toujours à travers le prisme des systèmes complexes.

Chapitre 5

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

5.1 Visualiser et Interagir avec une Simulation Multi-Agents

5.1.1 Simuler un Système pour Améliorer notre Compréhension

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

Jacques Tisseau

5.1.2 Visualisation de Systèmes Multi-Agents

Les travaux connexes sur la visualisation des colonies d'abeilles visent principalement à aider les apiculteurs à prendre des décisions, en leur fournissant des informations sur les populations d'abeilles [18, 19, 37]. Par exemple, Engelke et. al. [18] utilisent la réalité augmentée pour afficher des données provenant de plusieurs capteurs dans une série de ruches réelles et permettre à l'utilisateur de parcourir toutes les données de manière intuitive et immersive. Ces données sont collectées au niveau de la ruche (macro), comme la température et le poids. Ils sont ainsi capables de savoir dans quelle ruche se trouvent certaines abeilles et d'étudier la "dérive des abeilles" : lorsque les abeilles d'une colonie partent et rejoignent une autre colonie. Nous, en revanche, travaillons sur une colonie d'abeilles simulée et nous concentrons sur le "micro" monde, au niveau de l'abeille. Notre objectif n'est pas d'aider (directement) les apiculteurs à prendre des décisions, mais de leur permettre de visualiser et de comprendre les effets de leurs actions sur l'organisation

de la colonie (virtuelle), à l'intérieur de la ruche.

Louloudi et. al. [34] ont proposé dans leurs travaux de séparer la partie simulation multi-agents de la partie visualisation de leur programme. Séparation offre de pouvoir faire tourner le modèle pleine balle sans la visu quand pas besoin, et visu quand nécessaire. CIls proposent un composant logiciel "relai" entre simu et visu agissant comme un convertisseur universel. Or, les granularité de temps et d'espaces peuvent varier fortement d'un modèle multi agent à un autre, ce qui, couplé à un besoin de transmettre des informations depuis l'application de visualisation vers le simulateur rendent cette notion de relai universel bien difficile à mettre en place.

5.1.3 Interactions avec des Systèmes Multi-Agents

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.

5.2 Visualisation en Environnements Immersifs

5.3 Interactions en Environnements Immersifs

Conclusion

Chapitre 6

Propositions : Visualisation et Interactions

Nous souhaitons permettre à un utilisateur de pouvoir interagir avec notre simulation multi-agents de la colonie d'abeilles. De manière générale, les apiculteurs manipulent leurs ruches en ne manipulant que les cadres. Ils effectuent parfois certaines actions directement sur les reines, comme la placer dans une petite cage pour limiter ses mouvements, ou tout simplement la retirer et/ou en insérer une nouvelle. Nous ne nous sommes pas pour l'instant intéressés à ces cas particuliers, et avons décidés de permettre à l'utilisateur de manipuler les cadres de la ruche. En effet, déplacer les cadres ou en remplacer par d'autres permet d'altérer fortement l'environnement de la colonie. Les individus la constituant devront alors faire preuve d'auto-organisation afin de s'adapter à ces nouvelle conditions. En plus de provoquer cette réaction d'auto-organisation, nous souhaitons l'observer, et observer en détail les mécanismes qui la provoque.

Nous nous sommes ainsi principalement intéressés au cas de la division. La division est un geste apicole séparant une colonie, dont la population est trop importante, en deux colonies. Une colonie trop populeuse laissée en autonomie va "Essaimer" : les ouvrières vont élever une nouvelle reine, puis une bonne partie vont la suivre lors de son envol pour former une nouvelle colonie, ailleurs. La division permet à l'apiculteur de prévenir un éventuel essaimage et de garder le contrôle sur les deux nouvelles colonies, au lieu d'en perdre une dans la nature. Pour réaliser une division, les apiculteurs collectent différents cadres et les abeilles présentent dessus, et essayent de diviser en mieux en deux la colonie. Une nouvelle ruche est alors remplie de quelques cadres, en essayant d'y intégrer des abeilles de tout âge, du couvain de tout âge ainsi que différentes ressources. Les proportions doivent être à peu près les même pour la colonie source et la nouvelle colonie, afin de ne pas en condamner une. Une fois séparée, les cadres manquants de chaque ruche sont remplacés par des cadres vides.

6.1 Interactions avec la Simulation

L'utilisateur devra alors pouvoir interagir avec les cadres, les déplacer ou les retirer de la ruche. Il aura aussi besoin de manipuler la simulation en elle-même à travers ses interactions. Ainsi un utilisateur de notre application sera en mesure d'effectuer plusieurs actions :

- Attraper un cadre
- Frapper le cadre manipulé : cette action permet d'en faire tomber les abeilles, afin de pouvoir observer le contenu du cadre. Cette action est presque systématique pour l'apiculteur.
- Lâcher le cadre manipulé : soit dans la ruche sur un emplacement de cadre vide, soit à l'extérieur, permettant ainsi de laisser un emplacement vide. Un cadre peut aussi être lâché dans une deuxième ruche, permettant de simuler la division.
- Valider la division : cette action importante isole l'une des deux ruches (au choix de l'utilisateur). Les cadres et abeilles alors présents dans l'autre ruche sont retirés de la simulation.
- Accélérer le temps : l'utilisateur demande au simulateur de ne plus respecter le temps réel pour un temps donné. Le simulateur alors débridé calcule plus rapidement. Une fois la durée donnée par l'utilisateur atteinte, le simulateur repasse en mode temps réel. L'utilisateur indique un temps en seconde, c'est l'application interactive qui se charge de faire la conversion en pas de temps.
- Redémarrer la simulation
- Arrêter la simulation, fermer l'application.

6.2 Architecture Logicielle

6.2.1 Communications

Nous proposons de séparer la simulation et sa visualisation, suivant les conseils de Louloudi et. al. [34], en utilisant un serveur Java pour exécuter la simulation, et un client *Unity3D* pour la visualisation et les interactions. La Figure 6.1 illustre notre proposition. Nous n'utilisons toutefois pas de composant "relai" universel permettant le lien entre les deux parties, comme ils l'ont proposé. Ce rôle est tenu à la fois par un protocole de communication UDP/TCP que nous avons élaboré, et par un composant dédié aux communications dans les deux parties du système. Le simulateur et l'application interactive

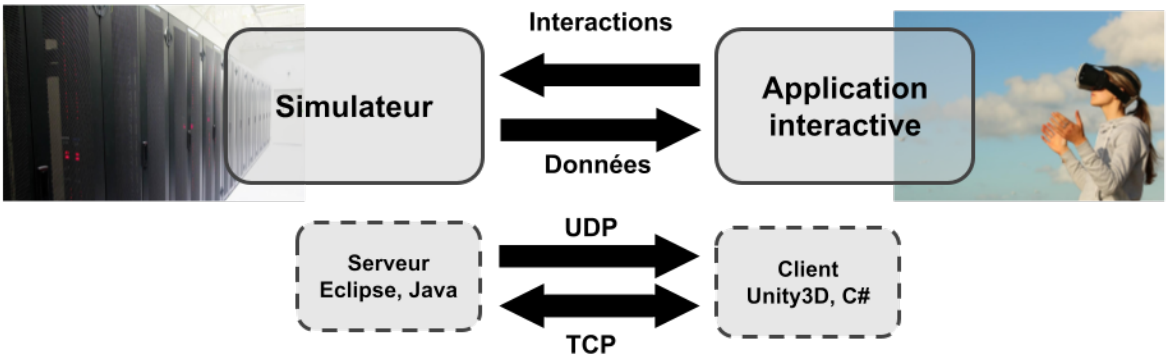


FIGURE 6.1 – Architecture du simulateur et de l'application interactive. Les deux composants sont totalements séparés et communiquent via le réseau, en utilisant des requêtes UDP ou TCP selon les besoins.

possèdent chacun une section réseau indépendante, permettant d'envoyer et recevoir des requêtes réseau.

En plus des bénéfices cités par Louloudi et. al. [34] et énumérés dans le chapitre prédécent, cette séparation permet de limiter l'impact des contraintes d'un support sur l'autre : contrainte graphique dans la simulation, ou contrainte de simulation dans l'environnement graphique.

Les communications du serveur vers le client se font majoritairement via UDP. Ce protocole simple permet l'envoi d'une grande quantité de donnée, mais ne permet pas d'assurer de la bonne réception de ces dernières. Ainsi, nous nous en servons pour transmettre sous la forme d'un flux les différente données de la simulation. En cas de mauvaise réception, il suffit d'attendre la prochaine itération du flux afin de corriger le tir. Ainsi, les positions des agents, leurs âges etc. passent par un flux UDP, formaté sous la forme :

[Mot Clé] [Paramètre] {Données}

Cette forme unique nous permet de faciliter le décodage de ces informations à la réception. Un mot clé permet d'orienter rapidement l'application client vers la bonne manière de traiter la suite des données. Ensuite, un paramètre permet de transmettre une information importante, qui n'a pas sa place dans le gros des données. Lorsqu'il n'est pas utilisé, ce paramètre est placé à -1, afin d'éviter que la première donnée soit comptée comme le paramètre. Ensuite les données consistent en une suite de caractères, séparés par des espaces. Le serveur utilise le protocole TCP pour communiquer des accusés de réception au client. Dans deux cas la réponse TCP du serveur est un peu plus complexe. Le premier cas, lors du démarrage ou du redémarrage d'une simulation, le serveur envoie via TCP les

Fréq	**Mot Clé**	**Param**	**Donnée(s)**	**Effets**
TCP				
-	STARTUDP	-	{combId}	La liste des identifiants des cadres de la simulation, en réponse aux STARTUDP et RESTART du client.
-	DEATHS	nb	{agentId}	La liste des identifiants des agents morts depuis la dernière requête "DEATHS", liste de longueur "nb".
UDP				
1/2	AGENTS	combId	{agentId, X, Y}	La liste des agents présents sur le cadré numéroté "combId". Ensuite, pour chaque agent, son numéro d'identification "agentId" ainsi que sa position "(X,Y)" sur le cadre.
1/4	FORAGERS	-	{agentId}	La liste des identifiants "agentId" de tous les agents à l'extérieur de la ruche.
1/20	CONTENT	combId	{Nb, contentId, fill}	Le contenu du cadre numéroté "combId". Les données contiennent ensuite pour chaque cellule du cadre, son numéro "Nb" (lié à sa position), un identifiant "contentId" indiquant le type de contenu et son remplissage "fill".
1/4	STATES	timeStep	{agentId, age, HJ, phEx}	Les états courant de tous les agents adultes. À la suite et pour chaque agent la liste contient son identifiant "agentId", son age, son taux d'HJ ainsi que la quantité de phéromones "phEx" que l'agent a échangé.

TABLE 6.1 – Tableau récapitulant les messages du serveur vers le client, leurs types et leurs sens.

identifiants ordonnés des cadres présents, permettant au client d'en connaitre le nombre et leurs positions les uns par rapport aux autres. Le second concerne la mort des agents. C'est en effet une information importante, nous ne voulons pas que le client la rate, car elle ne sera pas renvoyée. Ainsi, les identifiants des agents morts sont communiqués au client via TCP, sur une demande du client. La Table 6.1 récapitule les principales communications du serveur vers le client, en omettant les accusés de réceptions simples. Le tableau nous informe aussi de la fréquence des envoi UDP. En effet, afin de ne pas surcharger le client et le réseau, ces informations ne sont pas envoyées systématiquement. Le serveur fonctionne de manière autonome par rapport à la simulation, et possède ses propres pas de temps définis. Les fréquences sont données dans la Table en pas de temps réseau. Toutefois, le serveur est bridé afin de ne pas pouvoir être plus rapide que la simulation. Si le pas de temps de la simulation est plus lent que le serveur, alors celui-ci sera bloqué jusqu'au pas de temps suivant de la simulation. Ceci évite d'envoyer des données inutiles tout en permettant de ne pas envoyer de données à chaque pas de temps du simulateur, ce qui peut poser problème lorsque celui-ci est débridé.

Le client n'utilise que le protocole TCP pour communiquer avec le serveur. En effet, ne communiquant que très peu, et pour des informations importantes qui ne seront pas répétées, l'UDP n'est pas adapté. La Table 6.2 récapitule les formats et effets des différentes requêtes du client vers le serveur. Ces requêtes permettent de contrôler la simulation ainsi que d'influer sur le modèle multi-agents en fonction des actions de l'utilisateur.

Lors d'une avance rapide, le serveur cesse d'envoyer les positions des agents, les agents en butinage ainsi que le contenu des cadres. Ne sont envoyés que les activités des agents, la commande "*STATES*" du tableau 6.1. En effet, en avance rapide, ces informations concernant chaque abeille changent extrêmement rapidement et n'intéressent alors pas l'utilisateur, il n'est donc pas nécessaire de surcharger les communications.

6.2.2 Traitement

Le serveur gère ses envois grâce à un composant logiciel réseau communiquant avec le contrôleur principal de la simulation. Le composant réseau récupère ainsi régulièrement les différents états de la simulation, et les envoie à son rythme. Lors de la réception d'une requête client, la réponse est construite et immédiatement envoyée.

L'application interactive possède de son côté un autre composant réseau, divisé en plusieurs étapes, permettant d'interpréter les commandes et les rediriger vers les composants logiciels concernés. Tous ces échanges réseaux amènent une grande part d'asynchronicité,

Mot Clé	**Donnée(s)**	**Effets**
STARTUDP	-	Demande au serveur de démarrer le flux de données.
RESTART	-	Demande au serveur de redémarrer la simulation.
CLOSE	-	Demande au serveur d'interrompre le flux de données, ainsi que la connexion.
FFWD	x	Demande à débrider le serveur pour "x" pas de temps.
FrUP	Id	Signale au serveur que l'utilisateur vient de saisir le cadre numéroté "Id".
FrDOWN	Id, pos, sens	Signale au serveur que l'utilisateur vient de replacer dans la ruche le cadre numéroté "Id", à la position "pos". La commande indique aussi si le cadre a été retourné, avec le booléen "sens".
FrHIT	Id	Signale au serveur que l'utilisateur vient de frapper le cadre numéroté "Id".
REBASE	target, combIds	Demande au serveur d'effectuer la division, en gardant la colonie source ou destination selon "target", ainsi que la liste des identifiants des cadres "combId" à conserver.
Deaths	-	Demande au serveur la liste des agents morts depuis la dernière requète "Deaths".

TABLE 6.2 – Tableau récapitulant les messages TCP du client vers le serveur, et leurs effets.

nous imposant de travailler sur différent *threads*. Les messages sont reçu sur un *thread* dédié, ce qui permet de ne pas bloquer l'ensemble de l'application interactive dans l'attente d'une commande. Ce *thread* se charge ensuite d'interpréter le mot clé ainsi que le paramètre, afin de mettre en forme les données, qui sont jusque là des chaines de caractères. Une fois remise en forme, elles sont redirigées vers le *thread* principal en utilisant une série de liste synchronisée, une part type de commande. Ces listes sont remplies par le *thread* réseau, mais sont lues par le *thread* principal, à intervalle régulier. Ces files sont alors vidées par le *thread* principal. Une système de verrou a été mis en place afin d'éviter que le *thread* principal n'accède à une liste alors qu'elle est en train d'être mise à jours. Dans ce cas le *thread* principal n'a pas accès à la liste, elle sera lu lors du prochain passage.

Ainsi nos messages asynchrones sont traités, mis en forme, et transférés au *thread* principal, afin d'être traités de manière synchrone. Les messages envoyés par le client vers le serveur le sont sur le *thread* principal.

6.2.3 Modèle et Visualisation de la Colonie

Un composant logiciel de l'application interactive, appelé "Ruche", collecte et rassemble toutes les informations provenant du simulateur. Les agents sont rangés dans une liste et possèdent quelques attributs, comme leur dernière tâche en cours ou leur âge. Ils sont bien moins complexes que leurs homologues du simulateur. La ruche possède un autre composant logiciel, un manageur de cadre, permettant l'affichage des abeilles, via la gestion des différents cadres. Chaque cadre est ainsi représenté par un modèle 3D, et possède un nuage de point représentant les abeilles qu'il héberge. Lorsqu'une abeille change de cadre dans la simulation, elle doit être retirée du nuage de point du cadre d'où elle vient dans la visualisation, et ajoutée au nouveau. Ceci est possible grâce à l'utilisation de dictionnaire "clé-valeur", où chaque abeille est rangée en tant que valeur et a pour clé son numéro d'identifiant. Chaque nuage de points possède d'autres identifiants pour référencer leurs différents points, c'est pourquoi nous avons du créer un dictionnaire de transition entre les identifiants d'abeilles et les identifiants de points. Ainsi nous savons quel point supprimer afin de supprimer une abeille précise.

Nous obtenons ainsi autant de nuage de points que nous avons de cadres, et autant de points qu'il y a d'abeilles. Lors de la réception des positions des abeilles dans la simulation, une nouvelle position est générée, conservant les coordonnées X et Y, mais plaçant l'identifiant du cadre à l'axe z. Ainsi nous pouvons retrouver où afficher une abeille, et détecter un changement de cadre lorsque la coordonnée en z change d'une mise

à jours à l'autre. Les butineuses sont placées de la même manière, avec un identifiant de cadre égal à -1, tout comme leurs coordonnées X et Y. Les abeilles sur le cadre -1 sont alors affichées sur un nouveau nuage de points, lié à aucun cadre, virevoltant aléatoirement autour de la ruche. De la même manière, lorsqu'un changement de cadre est détecté, les points sont ajoutés et supprimés des nuages correspondant. Afin de lisser le déplacement des abeilles, la position de chaque point est interpolée entre sa position précédente et la nouvelle position reçue sur le réseau. Ainsi la visualisation, du fait de l'interpolation, est légèrement en retard par rapport à la simulation : une abeille venant d'atteindre la case X, Y dans la simulation, ne fera que commencer son trajet vers cette case X, Y dans la visualisation. La durée de l'interpolation est calibrée sur les pas de temps du serveur.

Ces nuages de points sont des polygones difformes possédant un sommet à chaque emplacement d'abeille. Le nuage de points est rendu possible par l'utilisation d'un "*Shader*" spécial, indiquant au moteur graphique de ne pas l'afficher comme une polygone normal. Au lieu de chercher à dessiner des triangles entre ces différents sommets, nous indiquons via ce nouveau *Shader* au moteur de créer de nouvelle forme autour de chacun de ces sommets. Ainsi, pour chaque sommet, nous dessinons un cube dont le centre correspond au sommet en question. Il est en plus possible de commander la couleur de ces cubes, pour chacun des points, ce dont nous nous servons pour un premier niveau de visualisation augmentée. Afin d'aider l'utilisateur à y voir un peu plus clair dans tous ces points répartis dans différents nuages, les abeilles sont colorés selon leur taux d'Hormone Juvénile (HJ). Une abeille avec un faible taux d'HJ (une nourrice) est dessinée en rouge, et une abeille avec un fort taux d'HJ (une butineuse) sera dessinée en jaune. La reine quant à elle est dessinée en blanc. La reine n'est pas détectée par la simulation tant qu'elle n'a pas pondu. Dès qu'une abeille, dans une mise à jours de statut, est noté comme étant en train de pondre, elle est classée en tant que reine et sera dessinée en blanc pour le reste de l'exécution de l'application interactive. Nous n'avons en effet que la reine qui peut pondre dans cette itération du simulateur, cette méthode sera à revoir lorsque les ouvrières seront autorisées à pondre aussi, ce qu'elles font dans de rares cas dans la réalité.

Les cadres sont chacun représentés par un modèle 3D, mais les cellules sont plaquées sur leurs faces sous la forme d'une texture. Cette texture est modifiée régulièrement, seulement aux endroits qui ont été modifiés, afin de suivre l'état de la simulation. Un nuage de point était utilisé auparavant, mais ces derniers n'étaient pas encore optimisé et il a donc été

décidé de changer de méthode. Les nuages de points on par la suite été optimisé, nous ne savons donc pas si la méthode de la texture apporte un avantage par rapport à cette nouvelle version des nuages de points, utilisés pour les abeilles. La texture est dessinée en fonction de l'identifiant de contenu reçu par le réseau. Un 0 signifie que la cellule est remplie de nourriture, un 1 indique un œuf, une larve ou une nymphe, et -1 indique une cellule vide. De cette manière différents contenus peuvent être ajoutés par la suite. Le code reçu donne sa couleur à la cellule, et son opacité dépend de la quantité présente dans la cellule. Assez explicite pour de la nourriture, cette quantité nous permet aussi de transmettre l'age de l'abeille infantile. La texture est ensuite dessinée en peignant des cercles de ces couleurs au centre des cellules. Nous n'avons donc pas d'hexagones, pourtant emblématiques de la ruche, mais la quantité de cercles présents permet d'obtenir un effet similaire pour un coup moindre.

6.3 Interaction Clavier Souris

La première étape d'interactions avec la ruche a été réalisé avec les interfaces classiques, clavier, souris et écran. Même si ces interfaces ne sont pas le cas d'utilisation final que nous souhaitons, nous y avons tout de même prêté du soin. En effet, une version clavier/souris de l'application interactive permettrai de toucher un publique plus large qu'en nécessitant un équipement plus avancé.

Nous avons donc opté pour une interaction classique que nous retrouvons régulièrement, surtout dans les jeux vidéo. L'utilisateur contrôle un avatar à l'aide de son clavier, qu'il déplace dans l'environnement virtuel. L'avatar ne possède pas de modèle 3D, l'utilisateur voit comme "à la première personne", à travers les yeux virtuels de son avatar. Un viseur se trouve au centre de l'écran, et permet d'interagir avec l'environnement au clic de la souris. Lorsque l'utilisateur effectue un clic gauche, le moteur 3D trace alors une ligne partant de la caméra et passant par ce viseur afin de retrouver avec quoi l'utilisateur souhaite interagir. Lorsque rien n'est trouvé, rien ne se passe. Quand un object permettant l'interaction est détecté, alors l'interaction est lancée. Ici, ce sont les cadres de la ruche avec lesquels nous pouvons interagir.

Lorsque l'utilisateur interagit avec un cadre, la requête "*FrUP*" est envoyée au serveur (que nous retrouvons dans la Table 6.2) et le cadre est détaché de la ruche et placé dans "les mains" de l'utilisateur. Ce cadre va désormais suivre tous ses mouvements. Une touche est associé avec la commande pour "frapper" le cadre, permettant d'en faire tomber les

abeilles. Dans la réalité il est important de réaliser cette action avec le cadre au dessus de la ruche, sinon les abeilles ne sachant par encore voler son perdues. Nous n'avons pour l'instant pas mis en place ce mécanisme. Les abeilles qui chutent sont replacées aléatoirement sur les autres cadres présents dans la ruche. L'utilisateur peut éloigner ou rapprocher le cadre de sa caméra grâce à la molette de sa souris. Il peut aussi le faire pivoter sur ses 3 axes à l'aide de 6 touches de clavier, deux touches par axe, une négative et une positive. Par exemple, "a" permet de faire pivoter le cadre autour de son axe vertical vers la gauche, et "e" sur le même axe mais vers la droite. De cette manière l'utilisateur peut observer avec attention le contenu du cadre.

Pour replacer le cadre dans la ruche, l'utilisateur doit l'approcher de l'emplacement auquel il souhaite le mettre, puis faire un clic gauche. Une fois assez proche, l'application placera un cadre "fantôme" rouge, permettant à l'utilisateur de savoir qu'en lâchant le cadre, ce-dernier ira prendre la place du cadre fantôme. En effet, il est assez compliqué de replacer le cadre correctement une fois l'avoir fait pivoter dans les 3 directions, c'est pourquoi nous avons mis en place cette aide. Il est ainsi possible de replacer un cadre précisément à l'emplacement que l'on souhaite. Lâcher le cadre alors que le cadre fantôme n'est pas apparu le laissera flotter là où il a été lâché. La commande signalant au serveur la pose du cadre ne se fait que lorsque celui-ci est replacé dans la ruche.

6.4 Interaction Immersive

Nous avons ensuite travaillé sur un autre mode d'interactions, prenant place dans un environnement immersif. L'utilisateur utilise alors un casque de réalité virtuelle et les manettes associées, afin de visualiser et d'interagir avec l'environnement 3D.

L'utilisateur se déplace dans l'application interactive en se déplaçant physiquement. L'interaction avec les cadres se fait via les manettes. L'utilisateur va placer ses mains (et donc ses manettes), de tel manière à ce que la représentation de ses manettes dans l'environnement se superposent à la partie supérieure du cadre. Il peut alors maintenir appuyé un bouton par manette afin de saisir le cadre, qui suivra désormais les mouvements de ses mains. Replacer un cadre dans la ruche est assez proche de la version Clavier Souris. L'utilisateur doit placer le cadre suffisamment proche de l'emplacement où il souhaite l'insérer, et le lâcher. Le cadre ira alors se placer automatiquement à l'emplacement libre le plus proche. Les autres fonctionnalités d'interactions sont obtenues via différents boutons sur les manettes.

Les communications réseaux de cette version ont été coupée, l'application lit directement des *logs* de simulation déjà générés. Il n'est donc pas possible d'interagir directement avec la simulation. La manipulation des cadres se fait à des fins purement pédagogiques et la lecture des *logs* permet la visualisation de données de la simulation. Cette version de l'application interactive a été développée dans le cadre d'un stage de M2 dans le but de servir de support à l'évaluation, dont nous parlerons dans le *Chapitre VII*.

6.5 Interaction Naturelle et Immersive

Ce mode d'interaction est une des contributions principales de ces travaux. Afin de proposer des moyens d'interactions naturels avec la simulation multi-agents, nous proposons ici d'utiliser des interacteurs tangibles. Ainsi, l'utilisateur portera un casque de réalité virtuelle, l'immergeant dans l'environnement 3D.

6.5.1 Le Cadre et la Ruche Traqués

Nous plaçons dans l'environnement physique de l'utilisateur une ruche bien réelle, contenant un unique cadre spécial, servant de manette. Ce cadre et la ruche sont "traqués" : via quelques cibles infra rouges placés dessus (et une série de caméra réparties dans la salle), leurs positions et rotations sont en permanence connues de l'ordinateur. Nous pouvons ainsi superposer la ruche virtuelle présente dans l'application interactive à la ruche physique, après de rapides calibrations. De la même manière, le cadre traqué possède une représentation virtuelle superposée, ce qui permet à l'utilisateur de saisir le cadre physique en ne voyant que le cadre virtuel. Ce cadre "manette" est facilement différentiable des cadres virtuels de la simulation : il est creux et blanc, là où les autres présentent des alvéoles et sont couleur bois. Dans cette première itération, l'utilisateur à tout de même besoin d'une manette pour réaliser ses interactions.

6.5.2 La Manette

Afin de ne pas trop gêner l'utilisateur lors de ses manipulations, les différentes interactions se trouvent toutes sur une seule manette, permettant de tenir le cadre physique de l'autre main. Afin de faire avec le nombre limité de boutons que propose la manette, nous avons mis en place un système de mode, permettant de varier les effets de mêmes boutons. Cette méthode permettra d'ajouter de nouveaux modes par la suite au besoin.

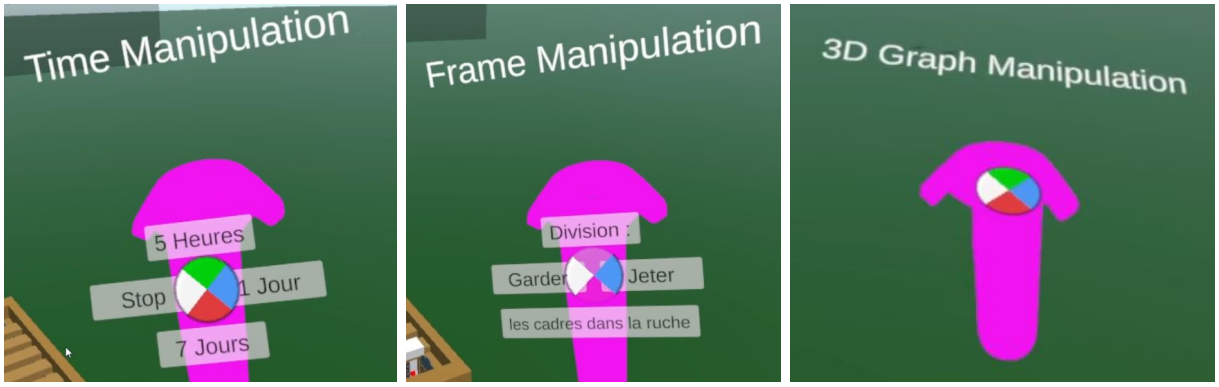


FIGURE 6.2 – Manettes vue par l'utilisateur, respectivement en modes Temps, Cadre et Visualisation. On observe le bouton principal divisé en quatre cadrans, transformant ce bouton unique en quatre boutons "virtuels".

Le bouton principal de la manette que nous utilisons, la manette du *HTC Vive*, est aussi une surface tactile. Ainsi, nous pouvons détecter la position du doigt de l'utilisateur sur le bouton lors de l'appui. Ceci nous permet de diviser ce bouton en différents cadrans (ou sections), et ainsi plusieurs boutons virtuels. Nous pouvons ainsi utiliser un nombre très limité de boutons de la manette, permettant aux utilisateurs de se familiariser rapidement avec la manette et ses fonctionnalités.

Trois modes ont été implémenté pour cette itération :

1. Mode Cadre : le bouton principal permet de réaliser une division. Le bouton secondaire permet de saisir et de lâcher les cadres virtuels à l'aide du cadre physique.
2. Mode Temps : le bouton principal divisé en plusieurs cadrans permet d'accélérer la simulation pour différents temps définis.
3. Mode Visualisation : nous parlerons de ce mode dans la Section 6.6.2.

La Figure 6.2 illustre la vision de la manette par l'utilisateur dans chacun de ces trois modes. On y observe notamment le bouton principal divisé en quatre cadrans, différenciés par des couleurs tranchées. Un autre bouton de la manette, au dessus du bouton principal, permet de faire défiler les modes les uns à la suite des autres. Un texte flottant au dessus de la manette permet à l'utilisateur de rapidement identifier le mode courant. De plus, des textes et couleurs sont plaquées sur la manette, s'adaptant aux différents modes afin de toujours refléter la fonction du bouton. Le bouton principal ainsi que le bouton de mode sont utilisés avec le pouce, le bouton secondaire est une gâchette généralement actionnée avec l'index.

6.5.3 Manipulation des Cadres

Afin de saisir un cadre, l'utilisateur vient superposer le cadre physique avec un des cadres virtuels présent dans la ruche. Ensuite, appuyer sur le bouton secondaire va venir plaquer le cadre virtuel ainsi sélectionné sur le cadre réel. L'utilisateur manipule désormais directement le cadre virtuel tiré de la simulation via son cadre physique. Il peut donc l'orienter facilement et l'observer de toute part de manière naturelle, comme nous le ferions avec n'importe quel objet bien réel. Un nouvel appui du bouton secondaire fera lâcher le cadre virtuel qui restera à flotter dans l'air. Si le cadre virtuel est lâché alors que le cadre manette est superposé à un emplacement vide dans la ruche, alors le cadre virtuel y est inséré et le serveur est prévenu.

Pour réaliser une division, l'utilisateur a deux possibilités, matérialisées sur deux des quatre cadrans du bouton principal. La division consiste à séparer la ruche en deux, et se concentrer ensuite sur l'une de ces deux nouvelles colonies. Une fois que l'utilisateur à sorti une certaine quantité de cadres, il peut choisir de réaliser la division. Les cadres à l'intérieur de la ruche formeront une nouvelle colonie, ainsi que tous ceux qui sont à l'extérieur. L'utilisateur doit donc choisir quelle ruche il désire suivre, l'ancienne ou la nouvelle. La manette propose ainsi de "Jeter" les cadres présents dans la ruche initiale, ou de les "Garder" et de jeter les autres, comme illustré dans la Figure 6.2.

Pour une itération future de ce moyen d'interaction, nous avons pensé équiper le cadre physique d'un bouton, afin de pouvoir saisir et lâcher les cadres virtuels sans avoir besoin de la manette. Ce bouton remplacerait alors le bouton secondaire de la manette, la gâchette. Ceci permet de manipuler les cadres à deux mains, comme le font les apiculteurs. En effet, si les cadres vides et sans cire sont légers, les cadres bâtis (remplis d'alvéoles de cire) et plein de miel peuvent facilement peser quelques kilogrammes.

Une autre possibilité d'amélioration pour cette interaction est de diminuer la taille des cibles infra-rouge. En effet, leur taille nous empêche de traquer plusieurs cadres sans que ceux-ci ne se gênent pour ensuite rentrer dans la ruche. De plus petits capteurs permettraient de traquer l'ensemble des cadres de la ruche, rendant la manipulation encore plus naturelle. Il sera alors nécessaire de détecter lorsque l'utilisateur saisi et lâche un cadre via les mouvements et positions des cadres.

Quelques prototypes ont été réalisés de ces deux propositions, mais beaucoup de travail reste à faire.

Frapper un cadre se fait par l'appui d'un autre bouton sur la manette. En effet, nos

premiers essais de détection de frappe sur le cadre via le *tracking* (changements brusques de vitesses) nous donnait des comportements trop imprévisibles. En effet, les données sont bruités et le coup sec que les apiculteurs donnent au cadre pour en faire tomber les abeilles est difficile à isoler. Nous pensons intégrer une centrale inertielle au cadre afin de ne pas détecter le choc via les caméra, mais directement depuis le cadre. Il serait peut être possible d'utiliser les données du *tracking* en y appliquant un traitement adapté.

6.5.4 Manipulation du Temps

Pour manipuler le temps, le bouton principal est divisé en quatre cadrans. Trois de ceux-ci présentent différents temps d'accélération prédéfinis, 7 jours, 1 jour et 5 heures, visibles sur la Figure 6.2 à gauche. L'intervalle de 7 jours correspond en général à la fréquence des visites de ruches de la part d'apiculteurs, les autres ont été choisi arbitrairement. Le troisième cadran "Stop" permet à l'utilisateur d'interrompre l'avance rapide, demandant au serveur de repasser en temps réel sans attendre d'avoir fini la précédente requête d'accélération. Envoyer une nouvelle requête efface la précédente, les durées d'accélération ne s'additionnent pas. Par exemple, le bouton Stop demande en réalité au serveur une avance rapide de zéro pas de temps, ce qui a pour effet de stopper toute avance rapide alors engagée.

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

Dans le but d'offrir une visualisation mettant en valeur les mécanismes de l'auto-organisation au sein de la colonie d'abeilles, nous proposons d'utiliser un graphique en 3 dimensions (sur 3 axes), que nous avons pour l'instant simplement surnommé le "*Graph3D*".

6.6.1 Fonctionnement du Graph3D

Pour reprendre simplement ce que nous avons vu dans le *Chapitre Contexte* ainsi que dans le *Chapitre III*, les abeilles, via leurs phéromones, viennent altérer leur âge physiologique. Cette altération devient visible car cet âge physiologique se décorrèle de leur âge réel. Nous proposons donc de placer chacune de ses données sur un axe, et de positionner ensuite chaque abeille de la simulation en un point du volume ainsi formé. La

Figure 6.3 illustre le Graph3D dans le cas d'une colonie classique respectant le polyéthisme d'âge, et dans le cas théorique d'une colonie peu après une division. Dans le premier cas, Figure 6.3a, l'âge physiologique et l'âge réel sont proches, les abeilles âgées butinent et les jeunes s'occupent du couvain. Dans le cas de la division, l'interruption de la ponte lié à la perte de la reine provoque un grand déséquilibre dans les répartitions des âges de la colonie. Nous nous attendons ainsi à observer de jeunes abeilles partir butiner plus tôt, c'est pour ceci que nous observons sur la Figure 6.3b des cubes jaunes (physiologie de butineuse) dans la moitié gauche du Graph3D, contenant les jeunes abeilles.

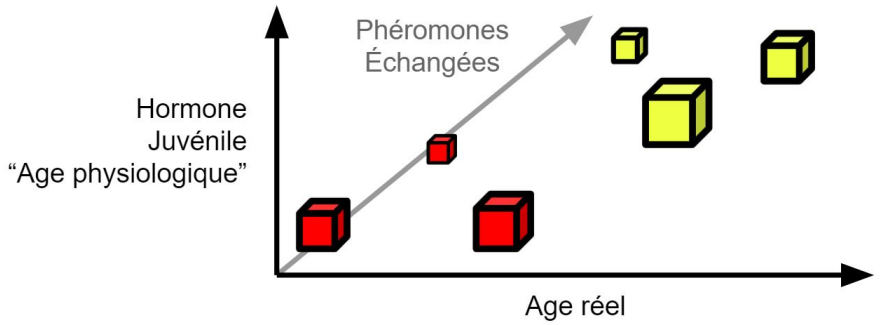
Le troisième axe, celui dédié aux échanges de phéromones, permet comparativement de comprendre pourquoi certaines abeilles partent butiner plus tôt que d'autres. Les abeilles partant butiner plus tôt sont celles qui ont le moins échangé de phéromones, censées les faire rester physiologiquement jeunes. Ainsi, plus une abeilles est loin sur l'axe des phéromones échangées, plus elle aura un âge physiologique faible. Cette donnée est obtenu en additionnant l'ensemble des échanges de chaque abeille avec les autres, en valeur absolue. Ainsi, donner une quantité de phéromone, ou recevoir cette même quantité aura le même impact sur cette valeur. Pour mieux représenter la réalité, cette somme est multipliée à chaque pas de temps par un coefficient légèrement inférieur à 1, afin de minimiser progressivement l'impact d'échanges lointains.

Les abeilles sont positionnées dans le Graph3D en utilisant un nuage de points semblable à ceux utilisés pour les placer sur les cadres. Nous y retrouvons, pour faciliter la lecture, la redondance de l'information de l'âge physiologique sous la forme de la couleur des abeilles. Toutes les abeilles au delà de $\frac{1}{2}$ sur l'axe de l'âge physiologique sera dessinée jaune, les autres en rouge.

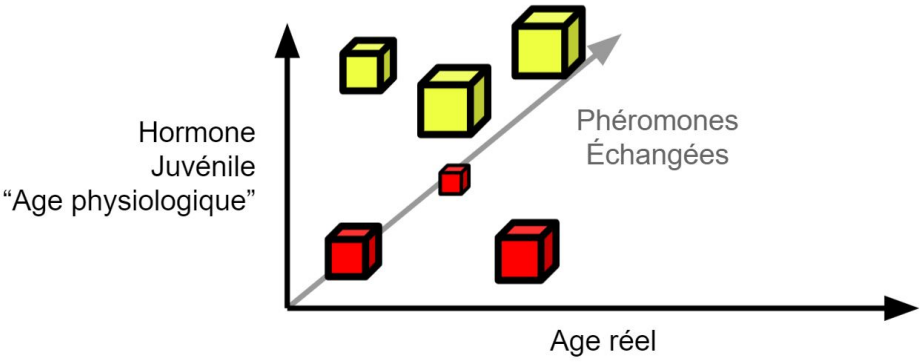
6.6.2 Manipulation du Graph3D

Clavier Souris

Dans la version Clavier Souris, le graph 3D est manipulé en utilisant trois positions prédéfinies. Les touches "1", "2" et "3" sont chacune associée à une rotation du Graph3D, qui viendra à l'appui se placer devant l'utilisateur dans la rotation souhaité. Ensuite l'utilisateur peut naviguer autour du graphique afin de mieux en saisir la profondeur. En effet, sur un écran le Graph3D est relativement difficile à lire, du fait de la projection 2D. Le Graph3D a été pensé pour fonctionner en environnement immersif.



(a)



(b)

FIGURE 6.3 – Notre *Graph3D* et ses trois axes. Chaque cube représente une abeille. La couleur du cube est une redondance de l'âge physiologique, rouge lorsqu'il est inférieur à $\frac{1}{2}$, jaune sinon. Une cube plus petit représente un cube plus loin sur l'axe de la profondeur, donc ici une abeille qui aurai échangé plus de phéromones. Dans le cas d'une colonie standard, nous nous attendons à trouver le Graph3D sous la forme (a), dans le cas d'une division nous nous attendons plutôt à la forme (b).

Environnement Immersif et Manette

Équipé d'une casque de réalité virtuelle, l'utilisateur peut manipuler le graphique à l'aide d'une manette. De la même manière que la version Clavier Souris, le Graph3D possède des ancres de rotation, et appuyer sur un des quatre cadrans du bouton principal sélectionne l'une de ses rotations. Lors de la sélection, le Graph3D vient se placer juste au dessus de la manette, dans la rotation souhaitée. L'utilisateur peut aussi appuyer sur le bouton secondaire de la manette afin de "saisir" le graphique. Une fois saisi, le graphique est "fixé" sur la manette, ainsi les mouvements de poignets de l'utilisateur déplaceront le graphique, en position comme en rotation. Cette méthode de manipulation, aidée par le casque de réalité virtuelle, permet de mieux ressentir les trois axes du Graph3D. De plus, cette dernière méthode de manipulation permet une interaction plus naturelle avec l'objet de visualisation, en effet, manipuler le graphique se fait comme nous manipulons des objets réels.

Conclusion

Dans ce chapitre nous avons présenté notre architecture logicielle. La séparation complète du simulateur et de l'application de visualisation et d'interaction nous offre souplesse et modularité. Nous avons décrit les échanges réseaux de ces deux parties, ainsi que les traitements qu'ils déclenchent. Nous avons passé en revue les différentes modélisations et représentations 3D de l'application, dont l'utilisation de nuages de points pour représenter les abeilles. Nous avons ensuite décrit les différents moyens d'interactions avec la simulation, depuis l'application interactive, dans différentes versions, pour différents supports, dont notre proposition de l'utilisation d'interacteurs tangibles pour des interactions naturelles. Nous avons ensuite présenté notre proposition de visualisation des mécanismes régissant la répartition des tâches au sein de la colonie, sous la forme d'un graphique en trois dimension, présentant l'état interne de chaque abeille de la simulation, dans un nuage de points.

Nous allons dans le chapitre suivant évaluer certaines de ces propositions, en décrivant l'expérimentation que nous avons mis en place, puis discuter des résultats obtenus et des perspectives qu'ils offrent.

Chapitre 7

Évaluation : Visualisation et Interactions

7.1 Expérimentation Visualisation Interactive

7.2 Résultats

7.3 Perspectives

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] M.R. Allen et al. *2018 : Framing and Context. In : Global Warming of 1.5°C. An IPCC Special Report on the impacts of global warming of 1.5°C above pre-industrial levels and related global greenhouse gas emission pathways, in the context of strengthening the global response to the threat of climate change, sustainable development, and efforts to eradicate poverty*. 2018. url : https://www.ipcc.ch/site/assets/uploads/sites/2/2019/05/SR15_Chapter1_Low_Res.pdf (visité le 22/07/2021).

[3] 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).

[4] Matthew Betti et al. « Bee++ : An Object-Oriented, Agent-Based Simulator for Honey Bee Colonies ». In : *Insects* 8.1 (10 mars 2017), p. 31. issn : 2075-4450. doi : 10.3390/insects8010031. url : http://www.mdpi.com/2075-4450/8/1/31 (visité le 30/07/2021).

[5] Eric Bonabeau, Marco Dorigo et Guy Theraulaz. *From Natural to Artificial Swarm Intelligence*. New York, NY, USA : Oxford University Press, Inc., 1999. isbn : 978-0-19-513158-1.

[6] Eric Bonabeau, Guy Theraulaz et Jean-Louis Deneubourg. « Quantitative Study of the Fixed Threshold Model for the Regulation of Division of Labour in

Insect Societies ». In : *Proceedings : Biological Sciences* 263.1376 (1996), p. 1565-1569. URL : http://www.jstor.org/stable/50403.

[7] 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).

[8] Arne Brutschy et al. « Self-organized task allocation to sequentially interdependent tasks in swarm robotics ». In : *Autonomous Agents and Multi-Agent Systems* 28.1 (jan. 2014), p. 101-125. ISSN : 1387-2532, 1573-7454. DOI : 10.1007/s10458-012-9212-y. URL : http://link.springer.com/10.1007/s10458-012-9212-y (visité le 28/07/2021).

[9] Adam Campbell et Annie S. Wu. « Multi-agent role allocation : issues, approaches, and multiple perspectives ». In : *Autonomous Agents and Multi-Agent Systems* 22.2 (2011), p. 317-355. ISSN : 1387-2532, 1573-7454. DOI : 10.1007/s10458-010-9127-4. URL : http://link.springer.com/10.1007/s10458-010-9127-4 (visité le 18/09/2019).

[10] Vincent A Cicirello et Stephen F Smith. « Wasp-like Agents for Distributed Factory Coordination ». In : (2004), p. 30.

[11] 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).

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

[13] Anna Dornhaus et al. « Task Selection in Honeybees - Experiments Using Multi-Agent Simulation ». In : (1998), p. 13.

[14] A Drogoul, Jacques Ferber et Christophe Cambier. « Multi-agent simulation as a tool for analysing emergent processes in societies ». In : (1992), p. 14.

[15] Alexis Drogoul. « 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 ». thesis. Paris 6, 1er jan. 1993. URL : http://www.theses.fr/1993PA066544 (visité le 29/05/2019).

[16] 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).

[17] Bruce Edmonds. « What is Complexity ? - The philosophy of complexity per se with application to some examples in evolution. » In : *Centre of Policy Modelling, Manchester Metropolitan University* (1999).

[18] Ulrich Engelke et al. « A Visual Analytics Framework to Study Honey Bee Behaviour ». In : *2016 IEEE 18th International Conference on High Performance Computing and Communications ; IEEE 14th International Conference on Smart City ; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. 2016 IEEE 18th International Conference on High Performance Computing and Communications ; IEEE 14th International Conference on Smart City ; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS). Sydney, Australia : IEEE, déc. 2016, p. 1504-1511. ISBN : 978-1-5090-4297-5. DOI : 10.1109/HPCC-SmartCity-DSS.2016.0214. URL : http://ieeexplore.ieee.org/document/7828555/ (visité le 11/03/2020).

[19] Ulrich Engelke et al. « MelissAR : Towards Augmented Visual Analytics of Honey Bee Behaviour ». In : *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. CHI'16 : CHI Conference on Human Factors in Computing Systems. San Jose California USA : ACM, 7 mai 2016, p. 2057-2063. ISBN : 978-1-4503-4082-3. DOI : 10.1145/2851581.2892367. URL : https://dl.acm.org/doi/10.1145/2851581.2892367 (visité le 10/08/2021).

[20] Jacques Ferber et Olivier Gutknecht. « eta-model for the analysis and design of Organizations in multi-agent systems ». In : (1998), p. 8.

[21] Nigel R Franks et Chris Tofts. « Foraging for work : how tasks allocate workers ». In : *Animal Behaviour* 48.2 (1994), p. 470-472.

[22] Jacques Gautrais et al. « Emergent Polyethism as a Consequence of Increased Colony Size in Insect Societies ». In : *Journal of Theoretical Biology* 215.3 (avr. 2002), p. 363-373. issn : 00225193. doi : 10.1006/jtbi.2001.2506. url : https://linkinghub.elsevier.com/retrieve/pii/S0022519301925068 (visité le 31/07/2021).

[23] Tugrul Giray et Gene E. Robinson. « Effects of intracolony variability in behavioral development on plasticity of division of labor in honey bee colonies ». In : *Behavioral Ecology and Sociobiology* 35.1 (juill. 1994), p. 13-20. issn : 0340-5443, 1432-0762. doi : 10.1007/BF00167054. url : http://link.springer.com/10.1007/BF00167054 (visité le 10/08/2021).

[24] H. J. Haubold et A. M. Mathai. « Analytic stellar structure ». In : *Astrophysics and Space Science* 197.1 (1992), p. 153-161. issn : 0004-640X, 1572-946X. doi : 10.1007/BF00645079. url : http://link.springer.com/10.1007/BF00645079 (visité le 04/08/2021).

[25] 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).

[26] Francis Heylighen. « Complexity and Self-organization ». In : *Free University of Brussels, Krijgskundestr* (2008), p. 20.

[27] Renée Johnson. « Honey Bee Colony Collapse Disorder ». In : (2010), p. 20.

[28] J. C. Jones. « Honey Bee Nest Thermoregulation : Diversity Promotes Stability ». In : *Science* 305.5682 (2004), p. 402-404. issn : 0036-8075, 1095-9203. doi : 10.1126/science.1096340. url : https://www.sciencemag.org/lookup/doi/10.1126/science.1096340 (visité le 16/07/2021).

[29] Steven A. Kolmes. « A Quantitative Study of the Division of Labour among Worker Honey Bees ». In : *Zeitschrift für Tierpsychologie* 68.4 (1984), p. 287-302. issn : 00443573. doi : 10.1111/j.1439-0310.1985.tb00130.x. url : https://

onlinelibrary.wiley.com/doi/10.1111/j.1439-0310.1985.tb00130.x (visité le 04/08/2021).

[30] Michael J.B. Krieger et Jean-Bernard Billeter. « The call of duty : Self-organised task allocation in a population of up to twelve mobile robots ». In : *Robotics and Autonomous Systems* 30.1 (2000), p. 65-84. issn : 09218890. doi : 10.1016/S0921-8890(99)00065-2. url : https://linkinghub.elsevier.com/retrieve/pii/S0921889099000652 (visité le 21/07/2021).

[31] Yves Le Conte, Marion Ellis et Wolfgang Ritter. « Varroa mites and honey bee health : can Varroa explain part of the colony losses ? » In : *Apidologie* 41.3 (mai 2010), p. 353-363. issn : 0044-8435, 1297-9678. doi : 10.1051/apido/2010017. url : http://link.springer.com/10.1051/apido/2010017 (visité le 23/07/2021).

[32] Yves Le Conte, Arezki Mohammedi et Gene E. Robinson. « Primer effects of a brood pheromone on honeybee behavioural development ». In : *Proceedings of the Royal Society of London. Series B : Biological Sciences* 268.1463 (22 jan. 2001), p. 163-168. issn : 1471-2954. doi : 10.1098/rspb.2000.1345. url : http://www.royalsocietypublishing.org/doi/10.1098/rspb.2000.1345 (visité le 05/06/2019).

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

[34] Athanasia Louloudi et Franziska Klugl. « A NEW FRAMEWORK FOR COUPLING AGENT-BASED SIMULATION AND IMMERSIVE VISUALISATION ». In : (2012), p. 7.

[35] 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).

[36] Alban Maisonnasse et al. « E-B-Ocimene, a Volatile Brood Pheromone Involved in Social Regulation in the Honey Bee Colony (Apis mellifera) ». In : *PLoS ONE* 5.10 (21 oct. 2010). Sous la dir. de Martin Giurfa, e13531. issn : 1932-6203. doi : 10.1371/journal.pone.0013531. url : https://dx.plos.org/10.1371/journal.pone.0013531 (visité le 05/06/2019).

[37] Huyen NGUYEN et al. « Augmented Reality Based Bee Drift Analysis : A User Study ». In : *2017 International Symposium on Big Data Visual Analytics (BDVA)*. 2017 International Symposium on Big Data Visual Analytics (BDVA). Adelaide, SA : IEEE, nov. 2017, p. 1-8. ISBN : 978-1-5386-0781-7. DOI : 10.1109/BDVA.2017.8114581. URL : http://ieeexplore.ieee.org/document/8114581/ (visité le 13/05/2020).

[38] Benjamin P. OLDROYD. « Domestication of honey bees was associated with expansion of genetic diversity : News and Views : Perspective ». In : *Molecular Ecology* 21.18 (2012), p. 4409-4411. ISSN : 09621083. DOI : 10.1111/j.1365-294X.2012.05641.x. URL : https://onlinelibrary.wiley.com/doi/10.1111/j.1365-294X.2012.05641.x (visité le 04/08/2021).

[39] Jérémy RIVIÈRE et al. « Modèle multi-agent d'auto-organisation pour le butinage au sein d'une colonie d'abeilles ». In : (2021), p. 27.

[40] Gene E. ROBINSON et Zhi-Yong HUANG. « Colony integration in honey bees : genetic, endocrine and social control of division of labor ». In : *Apidologie* 29.1 (1998), p. 159-170. ISSN : 0044-8435. DOI : 10.1051/apido:19980109. URL : http://www.apidologie.org/10.1051/apido:19980109 (visité le 28/07/2021).

[41] 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).

[42] T SCHMICKL et K CRAILSHEIM. « TaskSelSim : a model of the self-organization of the division of labour in honeybees ». In : *Mathematical and Computer Modelling of Dynamical Systems* 14.2 (2008), p. 101-125.

[43] Thomas SCHMICKL et Karl CRAILSHEIM. « Analysing honeybees' division of labour in broodcare by a multi-agent model ». In : (2008), p. 9.

[44] Thomas SCHMICKL et Karl CRAILSHEIM. « Costs of Environmental Fluctuations and Benefits of Dynamic Decentralized Foraging Decisions in Honey Bees ». In : *Adaptive Behavior* 12.3 (déc. 2004), p. 263-277. ISSN : 1059-7123, 1741-2633. DOI : 10.1177/105971230401200311. URL : http://journals.sagepub.com/doi/10.1177/105971230401200311 (visité le 04/06/2019).

[45] Thomas Schmickl et Karl Crailsheim. « HoPoMo : A model of honeybee intracolonial population dynamics and resource management ». In : *Ecological Modelling* 204.1 (mai 2007), p. 219-245. ISSN : 03043800. DOI : 10.1016/j.ecolmodel.2007.01.001. URL : https://linkinghub.elsevier.com/retrieve/pii/S0304380007000075 (visité le 30/07/2021).

[46] 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).

[47] Thomas D. Seeley. *The wisdom of the hive : the social physiology of honey bee colonies*. Cambridge, Mass : Harvard University Press, 1995. 295 p. ISBN : 978-0-674-95376-5.

[48] Thomas D. Seeley et Steven A. Kolmes. « Age Polyethism for Hive Duties in Honey Bees - Illusion or Reality ? » In : *Ethology* 87.3 (1991), p. 284-297. ISSN : 01791613, 14390310. DOI : 10.1111/j.1439-0310.1991.tb00253.x. URL : http://doi.wiley.com/10.1111/j.1439-0310.1991.tb00253.x (visité le 30/01/2020).

[49] ThomasD. Seeley. « The tremble dance of the honey bee : message and meanings ». In : *Behavioral Ecology and Sociobiology* 31.6 (déc. 1992). ISSN : 0340-5443, 1432-0762. DOI : 10.1007/BF00170604. URL : http://link.springer.com/10.1007/BF00170604 (visité le 04/06/2019).

[50] ThomasD. Seeley, Scott Camazine et James Sneyd. « Collective decision-making in honey bees : how colonies choose among nectar sources ». In : *Behavioral Ecology and Sociobiology* 28.4 (avr. 1991). ISSN : 0340-5443, 1432-0762. DOI : 10.1007/BF00175101. URL : http://link.springer.com/10.1007/BF00175101 (visité le 06/08/2021).

[51] G. Theraulaz, E. Bonabeau et J.-L. Deneubourg. « Response threshold reinforcements and division of labour in insect societies ». In : *Proceedings of the Royal Society B : Biological Sciences* 265.1393 (22 fév. 1998), p. 327-332. ISSN : 0962-8452. DOI : 10.1098/rspb.1998.0299. URL : https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1688885/ (visité le 02/10/2019).

[52] John N. Thompson. « Concepts of coevolution ». In : *Trends in Ecology & Evolution* 4.6 (juin 1989), p. 179-183. ISSN : 01695347. DOI : 10.1016/0169-5347(89)90125-0. URL : https://linkinghub.elsevier.com/retrieve/pii/0169534789901250 (visité le 04/08/2021).

[53] V. Volterra. « Variations and Fluctuations of the Number of Individuals in Animal Species living together ». In : *ICES Journal of Marine Science* 3.1 (1er avr. 1928), p. 3-51. ISSN : 1054-3139, 1095-9289. DOI : 10.1093/icesjms/3.1.3. URL : https://academic.oup.com/icesjms/article-lookup/doi/10.1093/icesjms/3.1.3 (visité le 30/07/2021).

[54] Edward O. Wilson. « The relation between caste ratios and division of labor in the ant genus Pheidole (Hymenoptera : Formicidae) ». In : *Behavioral Ecology and Sociobiology* 16.1 (nov. 1984), p. 89-98. ISSN : 0340-5443, 1432-0762. DOI : 10.1007/BF00293108. URL : http://link.springer.com/10.1007/BF00293108 (visité le 07/07/2021).

[55] Mark L. Winston. *The Biology of the Honey Bee*. Harvard University Press, 1991. 300 p. ISBN : 978-0-674-07409-5.

[56] Mark L. Winston et al. « The role of queen mandibular pheromone and colony congestion in honey bee (Apis mellifera L.) reproductive swarming (Hymenoptera : Apidae) ». In : *Journal of Insect Behavior* 4.5 (sept. 1991), p. 649-660. ISSN : 0892-7553, 1572-8889. DOI : 10.1007/BF01048076. URL : http://link.springer.com/10.1007/BF01048076 (visité le 28/05/2019).

[57] Michael Wooldridge, Nicholas R. Jennings et David Kinny. « A methodology for agent-oriented analysis and design ». In : *Proceedings of the third annual conference on Autonomous Agents - AGENTS '99*. the third annual conference. Seattle, Washington, United States : ACM Press, 1999, p. 69-76. ISBN : 978-1-58113-066-9. DOI : 10.1145/301136.301165. URL : http://portal.acm.org/citation.cfm?doid=301136.301165 (visité le 21/07/2021).

DOCTORAT BRETAGNE LOIRE / MATHSTIC

Université de Bretagne Occidentale

Titre : Visualisation et Interactions avec une colonie d'abeilles virtuelle : simulation, complexité et pédagogie.

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 quadriennised 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: Visualisation and Interactions with a virtual honey bee colony : simulation, complexity and pedagogy

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, int, 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.