université de bretagne occidentale

THÈSE / UNIVERSITÉ DE BRETAGNE OCCIDENTALE
sous le sceau de l'Université européenne de Bretagne

pour obtenir le titre de
DOCTEUR DE L'UNIVERSITÉ DE BRETAGNE OCCIDENTALE
Mention : Informatique
École Doctorale SICMA

présentée par
Sébastien Tripodi
Préparée dans le département d'Informatique
Laboratoire LISyC EA3883

Étude de l'auto-organisation des cellules basées sur le Modèle de Potts Cellulaire

Thèse soutenue le 13 octobre 2011
devant le jury composé de :

Franck DELAPLACE
Professeur des Universités, Université d'Évry / *rapporteur*

Jean-Pierre MÜLLER
Habilité à diriger des recherches, CIRAD / rapporteur

Pascal BALLET
Maître de Conférences, UBO / *examinateur*

Marie BEURTON-AIMAR
Maître de Conférences, Université Bordeaux1 / *examinateur*

Paul BOURGINE
Directeur de Recherche, École Polytechnique / *examinateur*

Vincent RODIN
Professeur des Universités, UBO / *examinateur*

Remerciements

Je tiens à remercier en premier lieu Vincent et Pascal, qui m'ont encadré pendant la durée de cette thèse. Ils ont toujours su me guider et m'épauler tout en me laissant l'autonomie nécessaire pour m'épanouir dans mes recherches.

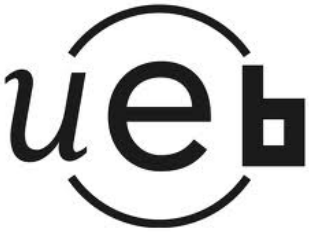
Je remercie également Franck Delaplace et Jean-Pierre Müller d'avoir accepté d'être rapporteurs de ma thèse. Je suis également très reconnaissant à Paul Bourgine et Marie Beurton-Aimar d'avoir accepté d'examiner mon travail.

Je remercie Derek Raine pour son accueil et ses conseils lors de mon séjour à l'Université de Leicester.

Je remercie mes collègues du Laboratoire LISyC et du département d'informatique de l'Université de Bretagne Occidentale qui m'ont accueilli et accompagné durant ces trois années. En particulier, je remercie Ciprian et Amara pour leur soutien et nos échanges passionnants.

Je remercie mes proches de m'avoir soutenu, conseillé et encouragé durant ces trois années.

Ce travail a bénéficié du soutient de la Région Bretagne et du Collège Doctoral International de l'Université Européenne de Bretagne.



Résumé

L'auto-organisation entre les cellules permet d'expliquer la morphogenèse des tissus cellulaires, comme le phénomène de l'embryogenèse. Cependant, il n'existe pas de consensus sur la nature des interactions entre les cellules amenant à cette auto-organisation. La modélisation et la simulation $in\ silico$ offrent un support formel aidant le biologiste dans sa compréhension du phénomène et donnent des arguments en faveur d'une théorie ou d'une autre. La mise en œuvre informatique de processus biologiques permet, en retour, d'améliorer les modèles informatiques existants. Les systèmes multi-agents sont des modèles informatiques qui représentent chaque entité (agent) du système de manière explicite. Les agents sont exécutés de manière autonome et en interaction avec les autres. La mise en œuvre d'un système multi-agents permet de simuler des agents cellules où les interactions reposent sur la consommation et la production de molécules, mais aussi sur l'adhésion et la pression que les cellules exercent les unes sur les autres. Un agent cellule est ici basé sur la cellule définie dans le Modèle de Potts Cellulaire. Ce modèle a été étendu (MorphoPotts) pour permettre aux cellules de cibler une forme générique et dynamique et, de définir un bilan énergétique. La théorie du darwinisme cellulaire, une théorie originale de l'embryogenèse, a été simulée à l'aide du MorphoPotts. Des tissus ont émergé depuis une cellule œuf. Ces tissus sont cohérents car ils se renouvellent en continu et ont une forme reconnaissable. Pour vérifier si les interactions entre les MorphoPotts permettent au système de s'adapter et de s'auto-organiser, les performances des systèmes multi-agents ont dû être améliorées. Nous montrons que la programmation sur les cartes graphiques (GPU) amène à des gains de performance importants. Les simulations faites sur les cartes graphiques montrent comment un darwinisme cellulaire permet aux tissus cellulaires de s'auto-organiser et de s'adapter en réponse aux événements extérieurs.

Mots-clés : Système multi-agents, Modèle de Potts Cellulaire, Morphogenèse, Auto-organisation, Programmation sur cartes graphiques.

Abstract

The self-organization between the cells gives an explanation of the cell tissues morphogenesis, like the phenomenon of embryogenesis. Meanwhile, there is no consensus on the nature of the interactions between the cells leading to this self-organization. On one hand, the $in\ silico$ modelisation and simulation offers a formal support to help the biologist in his understanding of the phenomenon and gives arguments in favour of a theory or of another one. The computer science implementation of biological process allows, on the other hand, to improve the existing computer science models. The multi-agent systems are computer science models which represent each entity (agent) of a system in an explicit way. The agents are executed in an autonomous way and in interaction with the others. The implementation of a multi-agent system allows the simulation of cell agents, where interactions are based on the consumption and production of molecules, but also on the adhesion and pressure the cells exert on the others. A cell agent is based on the cell defined in the Cellular Potts Model. This model has been extended (MorphoPotts) in order to allow the cells to reach a generic and dynamic shape and to define an energy balance. The theory of Darwin at cellular level, an original theory of the embryogenesis, has been simulated via MorphoPotts where tissues emerge from one stem cell. These tissus are coherent because they have a continue renewal and a recognizable shape. To verify if the interactions between the MorphoPotts allow the system to self-organize and self-adapt, the performances of the multi-agent systems were enhanced. We show that the graphics processing unit (GPU) programming leads to considerable performance gains. The simulations done on the GPU show that a cellular darwinism allows the cell tissue to self-organize and self-adapt in reply to exterior events.

Keywords : Multi-agent system, Cellular Potts Model, Morphogenesis, Self-organization, Graphics Processing Unit programming.

Table des matières

« What makes the change, from trivial to interesting, is simply the scale of the events. "Going to equilibrium" is trivial in the simple pendulum, is no more than a single point. But when the system is more complex ; when, say, a country's economy goes back from wartime to normal methods then the stable region is vast, and much interesting activity can occur within it. The computer is heaven-sent in this context, for it enables us to bridge the enormous conceptual gap from the simple and understandable to the complex and interesting. Thus we can gain a considerable insight into the so-called spontaneous generation of life by just seeing how a somewhat simpler version will appear in a computer. »

William Ross Ashby, 1962, dans "Principles of the self-organizing system"

Introduction générale

L'auto-organisation est inhérente aux systèmes biologiques depuis le niveau moléculaire comme le repliement des protéines, au niveau cellulaire comme l'embryogenèse, jusqu'au niveau des populations comme le système proie-prédateur. L'auto-organisation dans ces systèmes émerge des interactions locales entre les entités et non par une coordination centrale. Cependant, les interactions sont difficiles à déterminer car la dynamique est complexe. Cette complexité peut émerger d'interactions locales et simples.

Cette thèse étudie l'auto-organisation en biologie et en particulier entre les cellules. Son objectif est de définir, via l'utilisation des modèles de calcul, les interactions entre les cellules qui permettent au système de s'auto-organiser comme dans le phénomène de la morphogenèse. Nous étendons ces modèles de calcul (apport informatique) pour prendre en compte les données biologiques. Un modèle de calcul va permettre de simuler $in\ silico$ le système biologique et ainsi valider des propriétés et des théories dans ce modèle (apport biologique).

Contexte de la thèse

Cette thèse est à la frontière de plusieurs domaines où différents modèles de calcul sont spécifiés et dont les interactions sont décrites dans la figure 1. Les modèles de calcul sont des modèles mathématiques ou informatiques, ils permettent de définir une classe de fonctions calculables ainsi que les règles permettant de déterminer le résultat d'un calcul. Ce sont ($cf.$ la figure 1) la machine de Turing, le lambda calcul, les automates cellulaires, les systèmes multi-agents, les langages de programmation, ... Nous incluons aussi les méthodes de résolutions numériques telle que celle de Euler ou de Runge-Kutta.

Ces modèles de calcul sont à la base de la mise en œuvre de plusieurs logiciels permettant de simuler des systèmes biologiques comme Netbiodyn, Flame, ComputCell. Certains sont ($cf.$ la figure 1) des abstractions de systèmes biologiques et étudient des phénomènes plus généraux sur le vivant. C'est le cas par exemple du jeu de la vie pour l'auto-organisation et l'auto-réplication. A l'inverse, d'autres modèles ont été spécifiés pour décrire des systèmes biologiques comme le Modèle de Potts Cellulaire et le MorphoBlock. Cette thèse s'inscrit dans ce mouvement en créant les MorphoPotts.

Les modèles de calcul spécifiés pour la biologie prennent en compte des caractéristiques propres aux systèmes biologiques mais leur pouvoir d'expression est réduit. Par exemple, des modèles de calcul sont spécifiés pour modéliser la cellule. Une partie d'entre eux hérite de modèles physiques ($e.g.$ pour modéliser la forme de la cellule), de modèles chimiques ($e.g.$ pour modéliser la diffusion et les réactions de molécules), et d'autres algorithmiques pour représenter les comportements cellulaires comme la division ou la mort cellulaire. Un des enjeux de cette thèse est de synthétiser ces différents modèles de calcul dans un même paradigme.

Positionnement de la thèse

Cette thèse étudie l'auto-organisation entre les cellules au travers des modèles informatiques. Les modèles de cellules doivent être assez riches pour simuler cette auto-organisation. Nous pouvons diviser

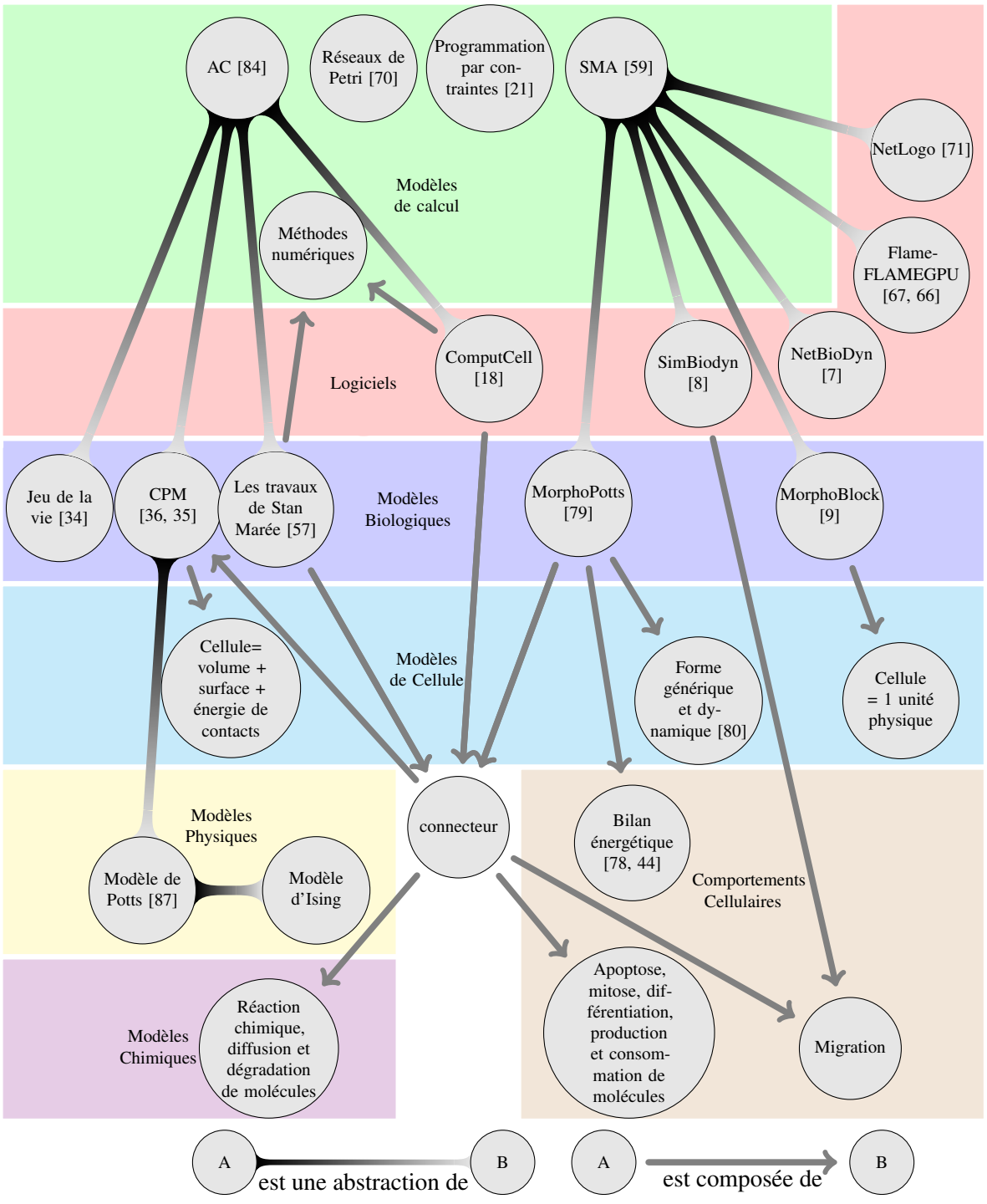


FIGURE 1 – Carte conceptuelle non exhaustive des modèles de calcul appliqués à la biologie. Les modèles de calcul utilisés dans cette thèse sont les Systèmes Multi-Agents (SMAs) et les Automates Cellulaires (ACs). Ils sont spécifiés dans le but de modéliser les systèmes biologiques au niveau cellulaire. Cette thèse présente le MorphoPotts qui est un agent représentant une cellule basée sur la cellule définie dans le Modèle de Potts Cellulaire (CPM) mais se différencie des autres travaux par une forme générique dynamique et un bilan énergétique.

en deux catégories les modèles de cellules : les cellules avec une forme statique ($e.g.$ un point de l'environnement, un carré ou un cercle), les cellules avec une forme dynamique ($e.g.$ définie par un ensemble de ressorts). La forme statique implique que la cellule ne peut pas se déformer, même si cette déformation sur une courte durée lui permet d'atteindre un état plus satisfaisant. Les études sur l'auto-organisation montrent un lien fort entre l'auto-organisation et l'adaptabilité [6]. Les modèles de cellules utilisés dans cette thèse représentent les cellules à l'aide d'une forme dynamique (elle change au cours du temps) pour répondre à ses propres contraintes et à celles de l'environnement. L'auto-organisation entre les cellules nécessite souvent un grand nombre de cellules. La simulation d'une cellule doit se faire dans un temps raisonnable. Dans les modèles existants, le Modèle de Potts Cellulaire est un bon compris entre la prise en compte de la forme dynamique des cellules et le temps d'exécution pour simuler cette forme.

Le Modèle de Potts Cellulaire définit la forme de la cellule en terme de surface et de volume. Dans ce modèle, l'énergie interne à la cellule n'est pas spécifiée. Cette thèse étudie et met en évidence comment l'énergie interne de la cellule dans le Modèle de Potts Cellulaire permet aux cellules de s'auto-organiser dans l'espace en tissus cohérents. Nous montrons aussi comment une forme générique peut être définie dans ce modèle avec 6 degrés de liberté (la forme est définie en 3D et peut subir une rotation et une translation selon les 3 axes) pour permettre l'auto-organisation d'un tissu. Ces deux études utilisent un darwinisme cellulaire de l'embryogenèse et donnent des arguments en sa faveur.

L'auto-organisation entre ces cellules implique souvent qu'un grand nombre d'entités doit être mis en jeu et ces entités doivent être en interaction les unes avec les autres. Nous montrons pourquoi les systèmes multi-agents sont de bons candidats pour modéliser de tels systèmes. Nous présentons aussi le Modèle de Potts Cellulaire (un automate cellulaire) à l'aide d'un système multi-agents. Le but étant ici de faciliter la mise en œuvre des différents comportements cellulaires qui sont nécessaires à l'étude de l'auto-organisation selon les critères développés dans cette thèse.

Un défaut des systèmes multi-agents est que le nombre d'entités à simuler entraîne un temps de simulation excessif. L'amélioration des systèmes multi-agents du point de vue du temps d'exécution est indispensable. Une des pistes possibles est la parallélisation. Aujourd'hui la programmation parallèle sur les processeurs graphiques est un bon compromis entre le prix du matériel, la facilité de la mise en œuvre et les performances. Cette thèse étudie et met en œuvre sur les processeurs graphiques le Modèle de Potts Cellulaire amélioré afin de simuler des auto-organisations complexes.

Organisation du mémoire

L'organisation de ce mémoire s'articule autour de 3 trois parties :

1. Modélisation informatique pour la biologie et choix du paradigme utilisé.
2. Étude, évolution et mise en œuvre du Modèle de Potts Cellulaire vers un Système Multi-Agents : Application à la Morphogenèse.
3. Parallélisation du Système Multi-Agents : Mise en œuvre et comparaison.

La partie 1 a pour but d'extraire un modèle de calcul adapté pour étudier l'auto-organisation entre les cellules. Pour cela une étude des modèles de calcul dans le contexte de la biologie est faite (chapitre 1). A partir du modèle de calcul choisi nous étudions les différentes spécialisations de ce modèle qui prennent en compte les comportements des cellules biologiques (chapitre 2).

Chapitre 1 Modèle de calcul appliqué à la biologie. Les modèles de calcul appliqués à la biologie sont nombreux et apportent des propriétés différentes. Certains d'entre eux permettent de simuler, d'autres de prédire et d'autres d'inférer des dynamiques. Dans ce chapitre nous comparons quelques uns de ces modèles à travers des exemples et nous exhibons un modèle qui nous semble le plus en adéquation avec les propriétés de l'auto-organisation.

Chapitre 2 Choix du modèle utilisé : Le Modèle de Potts Cellulaire. Pour un modèle de calcul donné

plusieurs spécialisations sont possibles pour simuler un phénomène biologique. Par exemple avec les systèmes multi-agents comme modèle de calcul, la littérature fait référence aussi bien à des agents cellules dont la forme est statique [9] qu'à des agents cellules dont la forme est dynamique [8]. Nous comparons les différents systèmes multi-agents et automates cellulaires spécialisés pour modéliser les cellules, et nous choisissons celui qui permet selon nous de simuler au mieux l'auto-organisation.

La partie 2 a pour but d'étudier et de simuler l'auto-organisation à partir du Modèle de Potts Cellulaire. Le phénomène choisi est l'embryogenèse. Premièrement nous formalisons le Modèle de Potts Cellulaire (chapitre 1) puis, nous proposons une mise en œuvre du modèle dans le paradigme des systèmes multi-agents (chapitre 2). Nous donnons des éléments de validation de notre mise en œuvre à travers des résultats connus et démontrons comment l'énergie interne à la cellule et la mise en œuvre d'une forme générique et dynamique permettent une auto-organisation des cellules en tissus complexes (chapitre 3).

Chapitre 1 Présentation formelle du Modèle de Potts Cellulaire. La formalisation du Modèle de Potts Cellulaire permet de définir les notations et d'exprimer formellement les propriétés de ce modèle. Cette formalisation est à la base de la traduction du modèle vers un système multi-agents.

Chapitre 2 Mise en œuvre du Modèle de Potts Cellulaire suivant l'approche multi-agents. La mise en œuvre du Modèle de Potts Cellulaire dans le paradigme multi-agents permet d'ajouter de nombreux comportements de manière plus aisée, notamment le bilan énergétique interne à la cellule et la forme générique. Ces deux comportements étant tout à fait originaux à notre connaissance, par rapport aux travaux existants sur le Modèle de Potts Cellulaire.

Chapitre 3 Applications. Le tri cellulaire est un phénomène fortement étudié dans le Modèle de Potts Cellulaire. Nous nous servons de ces études pour valider notre traduction de ce modèle en système multi-agents. Puis, nous vérifions la mise en œuvre d'une énergie interne à la cellule et la forme générique permettant l'auto-organisation des cellules.

La partie 3 a pour but d'améliorer significativement les performances du système multi-agents mis en place. Pour cela une mise en œuvre parallèle du système sur le GPU (Graphics Processing Unit) a été choisie (chapitre 1). Des éléments de validation de cette mise en œuvre sont donnés et comparés avec les résultats de la mise en œuvre de la partie 2. L'augmentation de la vitesse de simulation nous permet d'explorer des comportements collectifs trop long à simuler avant (chapitre 2).

Chapitre 1 Mise en œuvre parallèle sur GPU. Différents modèles de parallélisation, selon le flux des instructions ou des données, existent. Nous justifions que la parallélisation des données est une bonne solution pour notre système car il ne contient que des agents cellules (homogénéité des agents). La mise en œuvre parallèle sur le GPU respecte le type de parallélisation choisi, et se présente comme un bon compromis. Cette mise en œuvre a pour but d'exécuter chaque agent en parallèle.

Chapitre 2 Éléments de validation et étude des performances. Nous donnons des éléments de validation de notre mise œuvre en comparant les simulations obtenues avec ou sans parallélisation. Nous montrons que le nombre de pas de simulation qui peuvent être effectués dans un temps raisonnable augmente considérablement. Ceci amène à simuler des auto-organisations plus intéressantes. Nous étudions les performances en augmentant progressivement le nombre d'agents et en les simulant sur plusieurs matériels : GPU et CPU (Central Processing Unit). Nous en déduisons les inconvénients et avantages de la programmation sur GPU pour nos systèmes.

Première partie

Modélisation informatique pour la biologie et choix du paradigme utilisé

Chapitre 1

Modèle de calcul appliqué à la biologie

Ce chapitre est destiné à extraire un Modèle de Calcul (MC) depuis la littérature pour modéliser et simuler l'Auto-Organisation (AO) des cellules. Nous étudions les MCs appliqués à la biologie et déterminons les caractéristiques de ces modèles au travers d'exemples (1.1). Puis, nous justifions le choix du MC retenu en le comparant avec les caractéristiques de l'AO (1.2).

1.1 Étude des modèles de calcul appliqués à la biologie

Dans cette section nous justifions l'apport des MCs pour modéliser les systèmes biologiques (1.1.1), puis nous décrivons les propriétés de ces modèles suivant deux exemples : les réseaux de régulation génétiques et le système proie-prédateur. Les modèles étudiés sont les méthodes numériques (1.1.2), la programmation par contraintes (1.1.3), les réseaux de Petri (1.1.4), les automates cellulaires (1.1.5) et les systèmes multi-agents (1.1.6). Nous savons que d'autres MCs existent pour modéliser et simuler les phénomènes biologiques et d'autres exemples auraient pu être choisis, mais les modèles et les exemples retenus permettent de situer notre choix.

1.1.1 Pourquoi utiliser les modèles de calcul

En biologie il est possible de faire des expériences $in\ vivo$ (dans le vivant), $e.g.$ il est possible de manipuler l'ADN pour y insérer des gènes. D'autres sciences, comme l'astrophysique, n'ont pas cette possibilité car leurs objets ne peuvent pas être manipulés ($e.g.$ les planètes, les étoiles, ...). Cependant la biologie computationnelle et la bio-informatique connaissent un essor considérable. Pourquoi les expériences $in\ vivo$ ou $in\ vitro$ (dans le tube) ne sont elles pas suffisantes ? Pour répondre à cette question, prenons deux situations qui peuvent se présenter à un biologiste.

La première situation est la suivante. Supposons un système biologique où une interaction entre deux objets a été mise en évidence. Pour déterminer la nature et les effets de cette interaction, une solution est de l'inhiber. Si le système se comporte différemment, alors on peut déduire les effets de cette interaction. Mais, si le système se comporte de la même façon, aucune déduction ne peut être faite. Cela signifie que le système est redondant, $i.e.$ que d'autres interactions ont les mêmes effets que cette interaction. Une idée est alors d'inhiber un ensemble d'interactions, mais la combinatoire explose, le nombre de tests à effectuer devient trop important et trop cher si l'on souhaite être exhaustif.

La deuxième situation est la suivante. Supposons qu'un biologiste a une théorie. La tester $in\ vivo$ n'est pas évident car du bruit a pu modifier l'influence des interactions du système et ajouter et supprimer des interactions. Si la théorie n'est pas vérifiée le biologiste ne peut être sûr que cela ne provient pas du bruit ou, à l'inverse, si la théorie est vérifiée, le biologiste ne peut savoir si cela est dû au bruit.

Une étape préalable à l'expérience $in\ vivo$ ou $in\ vitro$ pour argumenter en faveur d'une théorie ou pour définir l'effet d'une interaction, peut être l'expérience $in\ silico$ (sur ordinateur). Modéliser les phénomènes biologiques pour les simuler sur ordinateur a des inconvénients et des avantages. Une abstraction du système doit être faite où tout n'est pas pris en compte. C'est là son principal avantage. Se restreindre à certaines interactions permet de délimiter l'ensemble des interactions mises en jeu dans le système, le bruit inhérent au système réel n'est pas présent. Cependant, si on suppose que le bruit fait parti du système lui même, il peut être modélisé. Dans tous les cas l'expérience $in\ silico$ permet d'être sûr que seules les interactions décrites dans le système sont réellement prisent en compte modulo les effets de bords dus à la simulation informatique ($e.g.$ la représentation des nombres flottants). Ceci permet de répondre à la problématique de la seconde situation. La problématique de la première situation trouve une solution directe dans l'expérimentation $in\ silico$ car les expériences peuvent être testées automatiquement, le temps d'une expérience peut être beaucoup plus rapide que l'expérience $in\ vivo$ et elle est assurément moins chère.

L'expérience $in\ silico$ repose sur une modélisation d'un phénomène réel. Les modèles utilisés sont les MCs. Les MCs sont des modèles mathématiques ou informatiques, ils permettent de définir une classe de fonctions calculables ainsi que les règles permettant de déterminer le résultat d'un calcul. Construire un MC permet de répondre aux problèmes cités précédemment. Le modèle permet de fixer les entités du système, les paramètres utilisés et simplifier la complexité. Le choix des paramètres et des interactions est motivé par des objectifs scientifiques. L'objectif scientifique peut être de nature différente : simuler, prédire, inférer, ... Cette thèse met en avant les MCs permettant de représenter des systèmes biologiques complexes, c'est à dire des systèmes constitués d'un grand nombre d'entités en interaction dont la compréhension de leur dynamique n'est pas triviale. Les MCs sont très variés et incluent : la machine de Turing [81], le lambda calcul [10], les réseaux de neurones [58], les langages de programmation, les automates cellulaires [84], les systèmes multi-agents [30], …. La figure 1 cite quelques uns de ces modèles appliqués aux systèmes biologiques. Les sections suivantes décrivent leurs propriétés au travers des deux systèmes suivants :

- les équations de Lotka-Volterra ou modèle proie-prédateur. Ce modèle établi par Alfred James Lotka en 1925 et Vito Volterra en 1926, étudie deux populations d'espèces x et y. L'espèce y a besoin de consommer l'espèce x pour se reproduire, x se reproduit très bien sans y. Ce système étudie donc la relation entre deux espèces.
- les réseaux de régulation génétiques. Ces réseaux décrivent les interactions entre les protéines et les gènes. Une protéine est synthétisée via l'expression d'un gène. Par exemple, supposons le système suivant : une protéine A dont sa concentration x_a a atteint un certain seuil θ peut alors exprimer le gène b qui permet la synthèse de la protéine B. Dans ce système la protéine A permet la production de la protéine B. Ce simple modèle décrit l'évolution de la concentration x_b de la protéine B.

1.1.2 Méthodes numériques

Les méthodes numériques permettent de résoudre, par des calculs et en général par des approximations numériques, des problèmes de mathématique continue. Par exemple, une équation différentielle peut être discrétisée par un ensemble fini de points, même si son domaine est continu. Une résolution numérique d'équation différentielle possible est la méthode d'Euler. La méthode d'Euler permet de connaître un point d'une fonction à partir d'un autre point et de sa dérivée. Les méthodes numériques permettent donc de résoudre des équations mathématiques continues sur ordinateur.

Les équations différentielles sont souvent utilisées pour décrire un système biologique, les méthodes numériques permettent de calculer sa dynamique. Nous voyons, à travers eux, la modélisation de deux systèmes biologiques (le système proie-prédateur et les réseaux de régulation génétiques) et nous extrayons les inconvénients et les avantages de cette méthode.

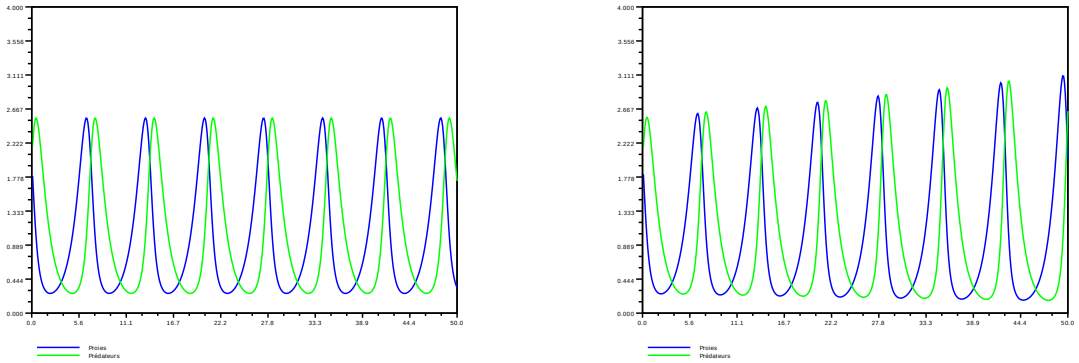
Le système biologique proie-prédateur se traduit par les équations différentielles suivantes :

$$\begin{cases} \frac{dx(t)}{dt} = \alpha x(t) - \beta x(t)y(t) \\ \frac{dy(t)}{dt} = \gamma x(t)y(t) - \delta y(t)) \end{cases} \tag{1.1}$$

où :

- $x(t)$ est le nombre de proies.
- $y(t)$ est le nombre de prédateurs.
- $\frac{dx(t)}{dt}$ et $\frac{dy(t)}{dt}$ représentent les taux de croissance de la population proie et prédateur.
- α et γ sont les taux de reproduction des proies respectivement en absence et en présence des prédateurs.
- β et δ sont les taux de reproduction des prédateurs respectivement en présence et en absence des proies.

Les méthodes numériques vont permettre de calculer la dynamique de ce système, c'est à dire l'évolution du nombre de proies et de prédateurs en fonction du temps t. La méthode d'Euler peut être utilisée pour modéliser ce système d'équations, mais induit de forts artefacts qui peuvent être corrigés par la méthode Runge-Kutta d'ordre quatre (RK4). La méthode Runge-Kutta est basée sur la méthode d'Euler mais avec plus de précision ($cf.$ la figure 1.1).



(a) Runge-Kutta d'ordre quatre (RK4)

(b) Euler

FIGURE 1.1 – Résolutions numériques du système d'équation proie-prédateur avec les paramètres $\alpha = \beta = \gamma = \delta = 1$. La méthode de Runge-Kutta d'ordre quatre (RK4) (1.1(a)) apporte une résolution meilleure que la méthode d'Euler (1.1(b)). Ces simulations sont faites à l'aide du logiciel Scilab [1].

Dans [69] les équations différentielles sont utilisées pour modéliser les réseaux de régulations génétiques. Comme dans l'exemple précédent, une équation différentielle est définie pour chaque population. Ici, la population est la concentration d'une protéine. L'exemple du réseau de régulation donné dans la section précédente peut se traduire par l'équation suivante :

$$x'_b = k_b1 * s^-(x_a, \theta) + k_b2 * s^+(x_a, \theta) - \gamma x_b \tag{1.2}$$

où :

- k_b1 est le taux de production de la protéine B si $x_a < \theta$.
- k_b2 est le taux de production de la protéine B si $x_a \geq \theta$.

1. `http://www.scilab.org/`

– $s^-(x_a, \theta) = 1$ si $x_a < \theta$ sinon 0.
– $s^+(x_a, \theta) = 1$ si $x_a \geq \theta$ sinon 0.
– γ le taux de dégradation de la protéine B.

A l'aide de ces deux exemples des inconvénients et des avantages peuvent être vus. Les inconvénients sont que :

– les entités ne sont pas explicitement décrites. Les équations prennent en paramètre le nombre d'entités. Cela suggère que des données quantitatives existent pour pouvoir simuler ces modèles.
– les valeurs des paramètres doivent être fixées.
– même si des évolutions des équations de Lotka-Volterra existent, il n'est pas trivial d'imposer des contraintes spatiales, comme ajouter des murs ou des niches pour que les proies se réfugient. Il n'est pas non plus trivial de représenter chaque entité dans l'espace. Cela en est de même pour les réseaux de régulation génétiques.
– selon la méthode numérique utilisée, des artefacts peuvent se produire. La figure 1.1 montre que la résolution par la méthode d'Euler amène à une oscillation du nombre d'individus des espèces mais avec une croissance infinie, alors que la méthode de Runge-Kutta d'ordre quatre amène aussi à une oscillation, mais le nombre d'individus des espèces reste entre deux seuils.

Les avantages d'utiliser les méthodes numériques sont que :

– elles résolvent des problèmes mathématiques. Il est possible en amont, durant, et après l'exécution de la méthode numérique d'utiliser les outils mathématiques, par exemple trouver des asymptotes, des points fixes, des états stationnaires et des attracteurs. Il est possible de prouver formellement une convergence du système.
– elles permettent de simuler des problèmes continus. Même si au final tous les points ne sont pas calculés, la précision qu'elles apportent est souvent suffisante.

1.1.3 Programmation par contraintes

La programmation par contraintes [56] permet de résoudre des problèmes combinatoires. Elle consiste en un ensemble de variables et un ensemble de contraintes entre ces variables. Ces contraintes réduisent l'ensemble des valeurs possibles pour une variable. Un solveur de contraintes permet alors de satisfaire ces contraintes si cela est possible.

Dans [21] la programmation par contraintes est utilisée pour modéliser les réseaux de régulation génétiques. Nous modélisons le réseau décrit précédemment selon la méthode définie dans [21] de la façon suivante :

– les variables $Kb1$ et $Kb2$ sont définies. $Kb1$ et $Kb2$ représentent respectivement le taux de production de la protéine B si $x_a < \theta$ et le taux de production de la protéine B si $x_a \geq \theta$.
– $Kb1$ et $Kb2$ sont définies dans $\{0, 1\}$ car deux niveaux de taux de production sont définis.
– l'inégalité suivante est imposée traduisant l'interaction de la protéine A sur la protéine B : $Kb2 > Kb1$.
– une équation focale est associée à chaque état du système (un vecteur dont les composantes sont les concentrations des protéines). Cette fonction focale calcule les états qui sont atteignables depuis un autre état. Elle permet aussi de poser des contraintes pour imposer une dynamique correcte. Par exemple, si la dynamique est synchrone ou pas.
– un chemin indiquant la dynamique voulue, $i.e.$ au minimum imposer sa longueur qui est le nombre d'états successifs voulu.

Comme pour les méthodes numériques, les entités ne sont pas explicitement décrites. Les variables désignent les concentrations des protéines. Les entités ne sont pas représentées, ni les protéines ni les gènes ne sont spatialisés. De plus, ce modèle repose sur la satisfaction de contraintes. Cela peut être un inconvénient si les contraintes sont faibles. C'est à dire ne réduisant que très peu le domaine des variables,

car une énumération des valeurs possibles des variables doit être faite, ce qui coûte cher du point de vue du temps d'exécution. Une solution pour réduire le temps d'exécution est de transformer ces contraintes en formule booléenne et d'utiliser un solveur booléen qui améliore considérablement les résultats [20].

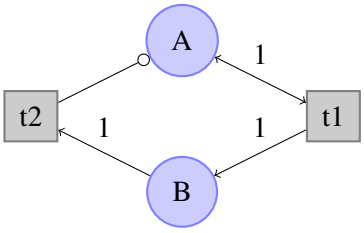
Des avantages peuvent être vus :

- les paramètres ne sont pas fixés.
- elle permet aussi bien de simuler, que d'inférer la dynamique.
- un relâchement de contraintes est possible afin d'éliminer le minimum de contraintes pour trouver une solution dans le cas d'incohérence. Ceci permet d'éliminer le minimum d'observations biologiques ou d'hypothèses pour rendre le système cohérent.
- les propriétés, comme l'existence d'un cycle dans la dynamique ou un état stationnaire, peuvent se traduire en terme de contraintes. Vérifier des propriétés peut être fait de façon formelle.

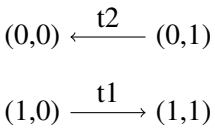
1.1.4 Réseaux de Petri

Les réseaux de Petri ont été créés en 1962, dans la thèse de doctorat de Carl Adam Petri. Un réseau de Petri peut être représenté par un graphe biparti, $i.e.$ il existe deux types de nœuds a et t. Chaque type de nœud est relié par un arc orienté à un nœud d'un type différent. Un nœud t a des nœuds sources a_s, $i.e.$ il existe des arcs de a_s vers t et des nœuds destinations a_d, $i.e.$ des arcs de t vers a_d. Les nœuds a contiennent des jetons avec une capacité limitée. A chaque arc a vers t et t vers a est associé un poids p. L'état du système est le nombre de jetons dans chaque nœud a. Les transitions entre les états sont définies par les nœuds t qui indiquent que les nœuds a_s perdent p_s jetons et les nœuds a_d gagnent p_d jetons durant la transition.

Dans [70, 17], ce modèle est utilisé pour modéliser les réseaux de régulation génétiques. Les nœuds a correspondent aux protéines et le nombre de jetons à leur concentration. Les nœuds t correspondent aux dégradations et synthèses des protéines. Suivant les travaux de [70], le réseau de Petri correspondant au réseau génétique décrit précédemment est donné dans la figure 1.2(a) et la dynamique, $i.e.$ le graphe de transitions, est donnée dans la figure 1.2(b). La transition $t1$ dans le graphe de transitions vérifie que, si la protéine A a une concentration supérieure à un certain seuil ici représenté par 1, alors la protéine B est produite à un taux supérieur à la normale. Si la concentration de la protéine A est en dessous du seuil θ, alors la production de la protéine B reste à un niveau normal ici représenté par 0 avec la transition $t2$. Le poids des nœuds t représente le paramètre cinétique, $i.e.$ le taux de synthèse.



(a) Réseau de Petri.



(b) Graphe de transitions équivalant au réseau de Petri donné.

FIGURE 1.2 – Exemple de réseau de Petri et du graphe d'interactions d'une protéine A qui active l'expression d'un gène b. Les nœuds de type a sont représentés par un cercle, les nœuds de type t par un carré. L'état d'un système est un vecteur (x_a,x_b) où x_i est la concentration de la protéine I, $i.e.$ le nombre de jetons du nœud I.

Comme pour la programmation par contraintes et les méthodes numériques, les entités ne sont pas explicitement décrites. Les variables décrivent les concentrations des protéines. Comme pour les méthodes

numériques, les valeurs des paramètres doivent être fixées.

Les avantages d'utiliser les réseaux de Petri sont que :

- ils permettent d'utiliser les travaux déjà existants, comme la recherche d'états stables, la recherche de cycles et la réduction du réseau.
- des propriétés notamment sur la dynamique peuvent être démontrées formellement.

1.1.5 Automates cellulaires

Un automate cellulaire (AC) est constitué d'une grille de N dimensions. Chaque site p de la grille a un état. L'état S du système correspond à l'ensemble des états s_p des sites p. La dynamique de ces systèmes est donnée par une fonction de transition f. La transition de l'état S au temps t à l'état S' au temps $t+1$ est réalisée en appliquant f à chaque site de la grille. Cette fonction f prend en paramètre l'état s_p d'un site p de la grille à l'instant t et l'ensemble n des états des sites voisins de p. f renvoie l'état du site p au temps $t+1$. Les ACs permettent donc de définir, par des comportements locaux, la dynamique du système. Nous montrons à travers le modèle proie-prédateur la réalisation d'un AC et les inconvénients et les avantages de ce modèle.

Dans le modèle proie-prédateur deux entités sont identifiées : les proies et les prédateurs. Nous considérons qu'une proie, respectivement un prédateur, occupe un site et un seul dans la grille. Un site de la grille a trois états : 0 (contient aucune entité), 1 (une proie est présente sur ce site), 2 (un prédateur est présent sur ce site). L'AC construit définit trois comportements pour les proies et prédateurs : la mort, la reproduction et le déplacement. Pour pouvoir concevoir ces comportements, nous assumons le fait que la fonction de transition f peut modifier aussi l'état des sites voisins. Pour éviter des conflits entre les entités dus à cette extension, une fonction $filtre$ est définie. $filtre$ prend en paramètre un site p et renvoie 1 si $filtre$ appliqué au site voisin de p renvoie 0. Le choix du site p pour lequel la fonction $filtre$ renvoie 1 est fait aléatoirement à chaque transition. La fonction de transition f peut être définie de la manière suivante :

Listing 1.1 – Fonction de transition de l'AC modélisant le système proie-prédateur

```
1  f(< s_p, n =< n_1, ...n_x >>) = deplacement(reproduction(mort(< s_p, n >)))
2  mort(s_p, n) =
3      < 0, n > si aleatoire()%10 = 0
4      < s_p, n > sinon
5
6  reproduction(< s_p, n =< n_1, ...n_x >>) =
7      < s_p, n > si s_p = 0
8      < s_p, n > si s_p = 1 ∧ ∃s_{n_i} = 2
9      < s_p, n' > sinon si s_p = 2∧ aleatoire()%3 = 0∧∃s_{n_i} = 0∧
10                          ∀n_j ≠ n_i(s_{n_j} = s_{n'_j}) ∧ s_{n_i'} = 1
11     < s_p, n' > sinon si s_p = 2 ∧ ∃s_{n_i} = 0 ∧ ∀n_j ≠ n_i(s_{n_j} = s_{n'_j}) ∧ s_{n_i'} = 2
12     < s_p, n > sinon
13
14 deplacement(< s_p, n =< n_1, ...n_x >>) =
15     < s_p, n > si s_p = 0
16     < s_p, n > si s_p = 1 ∧ ∃s_{n_i} = 2
17     < 0, n' > sinon si s_p = 1∧ aleatoire()%2 = 0∧∃s_{n_i} = 0∧
18                          ∀n_j ≠ n_i(s_{n'_j} = s_{n_j}) ∧ s_{n'_i} = 1
19     < 0, n' >sinon si s_p = 2∧ aleatoire()%2 =0 ∧ ∃s_{n_i} = 0∧
20                          ∀n_j ≠ n_i(s_{n'_j} = s_{n_j}) ∧ s_{n'_i} = 2
21     < s_p, n > sinon
```

La fonction f (l.1) est une fonction composée de trois fonctions : $deplacement$, $reproduction$ et $mort$. La fonction $mort$ appliquée à un site p, définit si l'espèce meurt, $i.e.$ la probabilité de mettre à jour l'état du site s_p à 0. Ici la probabilité qu'une espèce disparaisse est de 10% (l.3).
La fonction $reproduction$ appliquée à un site p vérifie que ce site contient une espèce (l.7). Si c'est une proie et qu'à son voisinage il y a un prédateur, la proie ne peut pas se reproduire (l.8). Sinon, la proie se reproduit avec une probabilité de 33%, c'est à dire qu'un site voisin qui était vide est mis à jour avec une proie. Ainsi la reproduction peut se faire s'il existe un site voisin vide (l.9 et l.10). Si le site est occupé par un prédateur, il se reproduit $i.e.$ remplit un site vide par une proie (l.11).
La fonction $deplacement$ appliquée à un site p vérifie que ce site contient une espèce (l.15). Si c'est une proie et qu'à son voisinage il y a un prédateur, la proie ne peut pas se déplacer (l.16). Sinon, la proie se déplace avec une probabilité de 50%, c'est à dire qu'un site voisin qui était vide est mis à jour avec une proie et le site p est remis à 0. Ainsi la reproduction peut se faire s'il existe un site voisin vide (l.17 et l.18). Si le site est occupé par un prédateur, il se reproduit avec une probabilité de 50% (l.19 et l.20).

La dynamique de cet AC est donnée par les figures 1.3(a) et 1.3(c). Nous pouvons observer les mêmes oscillations que dans la simulation faite par les méthodes numériques, à la différence que les oscillations sont moins régulières. Cela est dû entre autre aux probabilités de reproduction, de mort et de déplacement, ainsi que la compétition sur l'espace disponible.

A l'aide de cet exemple des inconvénients et des avantages peuvent être vus, ils sont radicalement opposés à ceux des méthodes numériques, programmation par contraintes et réseaux de Petri. Les inconvénients sont que :

- il est difficile de prouver formellement que le système converge vers une solution. D'une part il n'y a pas une fonction globale, et d'autre part le modèle est probabiliste. Il n'est pas impossible qu'une probabilité ne change totalement la convergence du système. Cependant, si le nombre de pas simulés est important, la convergence du système peut être observée.
- les paramètres doivent être fixés.

Les avantages d'utiliser les ACs sont que :

- les contraintes spatiales sont prises en compte. Les entités sont spatialisées.
- les données sont qualitatives, propres à un individu.
- la construction du modèle est très simple, seules les interactions locales doivent être définies.

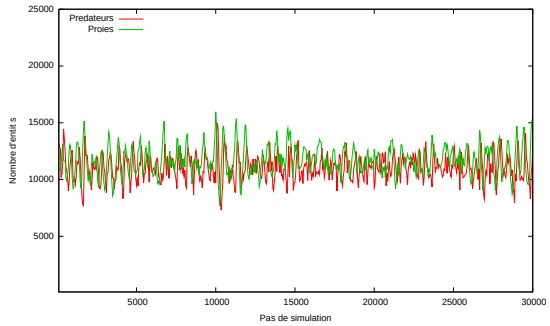
1.1.6 Systèmes Multi-Agents

Les systèmes multi-agents (SMAs), dont une description détaillée est donnée dans le chapitre 2.2, définissent un ensemble d'entités appelées agent, qui sont en interaction dans un environnement. Un agent a ses propres comportements et il est autonome. Chaque agent exécute ses comportements selon un ordonnanceur. Pour illustrer ce modèle de calcul nous modélisons les deux systèmes biologiques vus précédemment.

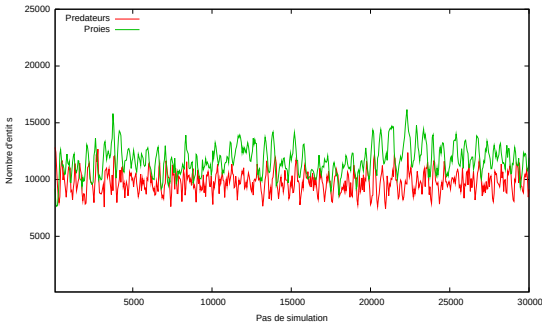
Pour modéliser le système proie-prédateur, nous reprenons les fonctions $mort$, $deplacement$ et $reproduction$ définies dans la section précédente avec les mêmes paramètres. Le SMA est le suivant :

- un environnement est défini. L'environnement correspond à la grille décrite dans la section sur l'AC, dans laquelle un site est, soit rempli par du vide, soit par un agent.
- deux types d'agents sont définis : les proies et les prédateurs.
- les proies et les prédateurs ont trois comportements : la mort, la reproduction et le déplacement. Ces comportements sont exécutés dans cet ordre. Durant un pas de simulation un agent exécute d'abord son comportement de mort, puis de reproduction, puis de déplacement. Pour les mêmes raisons que dans la section sur l'AC une fonction $filtre$ est exécutée par les agents avant chaque

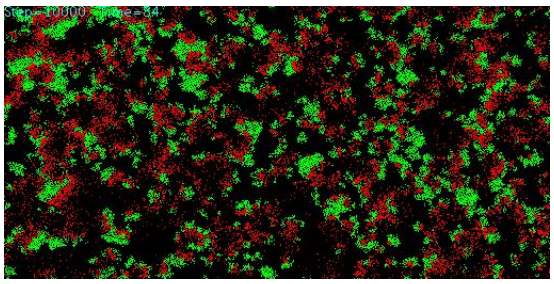
2. http ://www.matrixstudio.org/



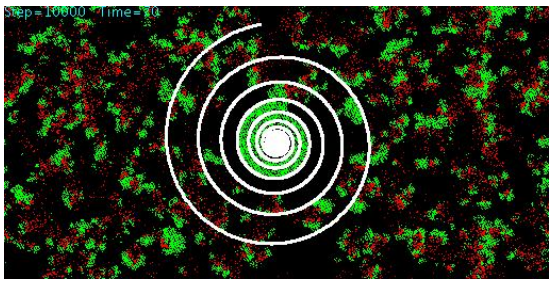
(a) Courbes d'évolution du système proie prédateur simulé par un AC



(b) Courbes d'évolution du système proie prédateur simulé par un AC avec contrainte spatiale



(c) État au temps 10000 du système proie prédateur simulé par un AC



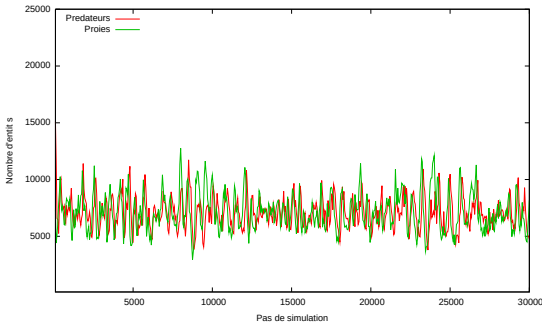
(d) État au temps 10000 du système proie prédateur simulé par un AC avec contrainte spatiale

FIGURE 1.3 – Simulations du système proie-prédateur par un AC avec et sans contraintes spatiales depuis un état aléatoire. Le vide est représenté par la couleur noire, les proies par la couleur verte et les prédateurs par la couleur rouge. La contrainte spatiale est représentée en blanc, elle signifie que ces sites là ne peuvent changer d'état donc que les proies et les prédateurs ne peuvent occuper ces sites. La simulation sans contraintes spatiales donne des oscillations connues du modèle proie-prédateur. Les oscillations ne sont pas régulières car les comportements sont soumis à des probabilités. Le système avec les contraintes spatiales montre comment les proies trouvent refuge. Ces simulations ont été réalisées à l'aide du logiciel MatrixStudio [2]. La vidéo de la simulation sans niche peut être vue à l'adresse suivante : `http://www.youtube.com/watch?v=rQzGTWVlxVI` et celle avec niche à l'adresse à l'adresse suivante : `http://www.youtube.com/watch?v=jcM7aXlT1Rc`

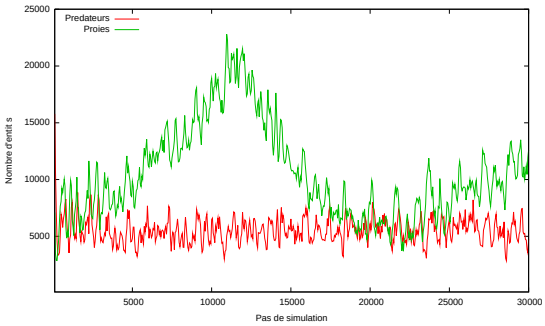
comportement, lui autorisant ou non d'exécuter son comportement. Cela est nécessaire pour éviter les conflits entre les agents, deux agents pouvant se déplacer sur le même site.

– les interactions se font à travers les fonctions de reproduction et de mort, mais aussi à travers la compétition pour occuper un site de la grille.
– un ordonnanceur qui permet d'organiser l'exécution des agents. Chaque agent exécute en parallèle ses comportements modulo la fonction $filtre$ pour éviter les conflits.
  La dynamique de cet SMA est donnée par les figures 1.4(a) et 1.4(b). Nous pouvons observer les mêmes oscillations que dans la simulation faite par les méthodes numériques, à la différence que les oscillations sont moins régulières, comme cela a été montré pour les ACs.

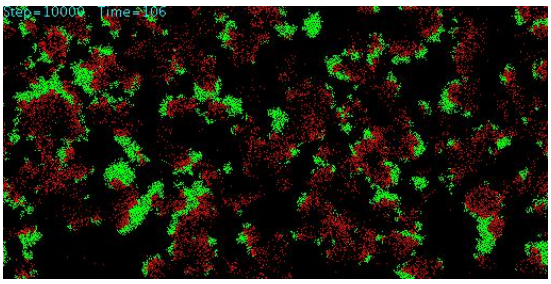
L'autre système étudié est les réseaux génétiques. Dans [82] les SMAs sont utilisés pour modéliser les réseaux de régulation génétiques. Pour modéliser ces réseaux, des agents mobiles sont définis pour représenter les protéines. Ils ont une certaine vélocité, ils ont la capacité de se fixer sur l'ADN, ils ont une durée de vie et ils peuvent former d'autres molécules lors d'une collision. Un agent spécial peut, en fonction d'autres agents, se fixer sur l'ADN, produire d'autres protéines et donc créer d'autres agents.



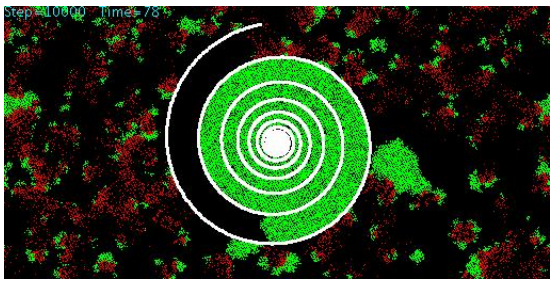
(a) Courbes d'évolution du système proie prédateur simulé par un SMA



(b) Courbes d'évolution du système proie prédateur simulé par un SMA avec contrainte spatiale



(c) État au temps 10000 du système proie prédateur simulé par un SMA



(d) État au temps 10000 du système proie prédateur simulé par un SMA avec contrainte spatiale

FIGURE 1.4 – Simulation du système proie-prédateur par un automate SMA avec et sans contraintes spatiales depuis un état aléatoire. Nous retrouvons les mêmes caractéristiques que dans la figure 1.3, à la différence que les proies trouvent refuge plus facilement dans le SMA. La différence entre le SMA et l'AC est que dans l'AC la fonction $filtre$ est appelée une fois avant que les agents exécutent leurs trois comportements, alors que dans le SMA la fonction $filtre$ est appelée avant chaque comportement. La vidéo de la simulation sans niche peut être vue à l'adresse : `http://www.youtube.com/watch?v=Sv3iriJLnA8` et celle avec niche à l'adresse : `http://www.youtube.com/watch?v=OVgrqwu_Vso&feature=related`.

Comme pour les ACs, les paramètres doivent être fixés. Le problème de convergence est difficile à résoudre car il n'y a pas une fonction globale, le modèle est probabiliste, et de plus différents formalismes peuvent être mélangés.

Les avantages d'utiliser les SMAs, comme pour les ACs, sont que : les contraintes spatiales sont prises en compte, les entités sont spatialisées et les données doivent être qualitatives. Par exemple, pour les réseaux génétiques, la description est très proche du point de vue de la biologie, car le mécanisme opératoire de la synthèse des protéines est mis en place. La construction du modèle est simple, juste les interactions locales doivent être définies. La structure du SMA permet une mise en œuvre plus aisée des comportements, car ils peuvent être définis dans des formalismes différents.

1.2 Propriétés de l'auto-organisation et choix du modèle de calcul

Cette thèse étudie l'Auto-Organisation (AO) des cellules. Dans la section précédente nous avons étudié quelques MCs. Au travers des propriétés de l'AO (1.2.1), nous comparons ces modèles (1.2.2).

1.2.1 Auto-Organisation

L'AO est un phénomène permettant l'émergence d'une structure, d'un motif, d'un arrangement dans un système dynamique. Elle permet l'organisation d'un système sans qu'une contrainte extérieure ou globale au système impose cette organisation. Nous décrivons les différentes études de l'AO et mettons en lumière ses propriétés dans les systèmes biologiques et au niveau cellulaire.

Les travaux sur l'auto-régulation : définition et caractéristiques

Pour définir le terme AO nous donnons quelques exemples dans différents domaines et nous présentons les travaux de William Ross Ashby et de Henri Atlan.

Exemples d'AO L'AO n'est pas propre à certaines sciences. Nous donnons quelques exemples d'AO en chimie, physique, biologie, informatique et mathématique. Le chimiste russe Belousov en 1950 découvrit une réaction oscillante. Il s'est rendu compte qu'en mélangeant cinq composés, on n'atteint pas directement l'état d'équilibre. La solution oscille entre deux états, de façon régulière jusqu'à épuisement d'un des réactifs. Cette réaction est appelée réaction Belousov-Zhabotinsky. Les oscillations sont une AO du système car elles ne sont pas imposées par une contrainte extérieure.
En physique les transitions de phases peuvent être considérées aussi comme une AO. Même si ce changement de phase peut provenir de la variation d'un paramètre extérieur, le système s'auto-organise pour changer de structures (par exemple dans le phénomène d'ébullition et avec la température comme paramètre extérieur).
En biologie, l'embryogenèse est le processus de développement de l'embryon, $i.e.$ le développement des tissus depuis la cellule œuf. Le système est auto-organisant car il amène à une structure organisée et robuste.
Les modèles informatiques et mathématiques peuvent être aussi auto-organisants. Prenons le cas des automates cellulaires, comme les automates auto-reproductibles développés par John von Neumann dans les années 40 [15], le jeu de la vie développé par John Horton Conway en 1970 [34] et les fourmis de Langton [48]. Ces modèles montrent des structures qui peuvent se répliquer dans les automates cellulaires.

Auto-organisation selon William Ross Ashby Le terme AO est apparu dans les travaux de Ashby en 1947 [4, 5]. Mais, cette notion d'AO est plus ancienne, comme dans les travaux de René Descartes sur des lois locales contenues dans les différentes parties qui composent « la nature » et qui permettent, depuis « un chaos le plus confus et le plus embrouillé », l'émergence d'un « Monde très parfait » [14]. Aussi, on retrouve cette notion bien antérierement, avec les travaux des atomistes. Cependant, nous nous intéressons à une définition formelle et réutilisable dans le contexte des modèles de calcul. Les travaux de Ashby y répondent en partie.

Avant de définir le terme AO, définissons le terme organisation. Dans [5] une organisation dans un système ne peut avoir lieu que s'il y a au moins 2 objets A et B dans le système. Si A est en interaction avec B, $i.e.$ la valeur de A est liée à celle de B, alors nous pouvons dire que A et B sont organisés. Ainsi un système est plus ou moins organisé si les objets entre eux sont plus ou moins en interaction et leurs valeurs plus ou moins liées. Dans cette notion on suppose que le « tout », le système, est composé de parties.

Comme l'explique Ashby, l'organisation n'est pas liée à la dynamique. En physique un système est souvent décrit par des variables x_1, x_2 ,..., x_n. Le système est alors traité comme n parties fonctionnelles en interaction. Il est également possible de considérer le système comme un tout à la dynamique pouvant être sophistiquée. Finalement, supposons deux observateurs d'un même système, l'un peut le voir comme

des milliers d'entités en interaction, l'autre seulement comme des trajectoires entre les états. Ainsi, l'organisation dépend du point de vue et du regard que l'on porte sur le système.

Pour essayer de donner une définition formelle de l'organisation, Ashby définit l'organisation dans une machine, $i.e.$ dans un modèle de calcul. Soit la machine suivante : S un ensemble d'états, I un ensemble d'entrées et f une relation entre $S * I$ et S comme décrit dans la figure 1.5. Une organisation dans cette machine est définie par f. Comme l'organisation a été définie, $i.e.$ une interaction entre les parties du système, si f change, alors l'organisation change.

(a) $f(S, I) = (S + I)\%4$ (b) $f(S, I) = (S * I)\%4$

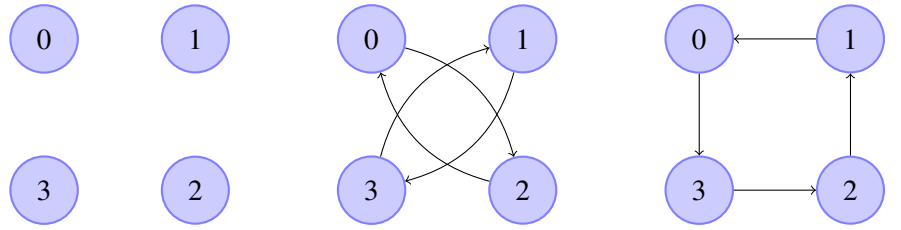
FIGURE 1.5 – Exemple de deux organisations d'un même système selon une relation f différente. Le système défini est le suivant : l'ensemble des états $S = \{0, 1, 2, 3, 4\}$ et des entrées $I = \{2\}$. Les nœuds représentent les états, les flèches les interactions, donc l'organisation entre les états est définie par f.

Finalement, un système où les différentes parties de celui-ci sont en interaction, est dit organisé. Cependant, comment définir qu'une organisation est bonne dépend du point de vue. Cela peut être quantifié, par exemple, par le nombre d'interactions entre les parties du système et beaucoup d'autres critères peuvent être définis qui dépendent de l'observateur.

Ashby distingue deux types d'organisation, dont le premier est un cas particulier du second. Le premier type d'organisation peut s'illustrer par la neurogenèse et, en particulier, la transition entre les neurones formés, mais pas organisés, et la formation des axones et des synapses permettant de former un réseau organisé. Le premier type d'organisation est donc la connexion des parties qui forment le système depuis un système non organisé ($cf.$ les figures 1.6(a) et 1.6(b)). Cette organisation implique une seule relation f car une seule organisation est décrite. Ce type d'organisation est auto-organisé dans le sens qu'aucune contrainte extérieure rentre dans le processus d'organisation.

Le second type d'organisation est plus intéressant : le changement d'organisation. Par exemple, quand un enfant voit le feu il est attiré par lui tant qu'il ne s'est pas brûlé. Le système s'est auto-organisé, mais par un événement extérieur « le feu ». Formellement, dans la machine de Ashby, cela se traduit par le fait que les interactions entre les états S changent ($cf.$ les figures 1.6(b) et 1.6(c)). Ceci implique que f soit modifiée suivant un paramètre α. Cependant seulement et seulement dans le cas où $\alpha \in S$ il peut être affirmé que le système est auto-organisant. Mais comment les états de S peuvent changer f ? Notons que α peut être extérieur au système et correspondre par exemple à la température en physique ou à une fonction aléatoire en mathématique.

Comme Ashby le montre, en général un système dynamique génère sa propre « forme de vie intelligente », c'est dans ce sens que l'on peut dire que le système est auto-organisant. Les systèmes dynamiques vont en général à l'équilibre et, comme tous les états vont vers l'équilibre, le système va d'un ensemble important d'états vers un ensemble plus petit. Le système fait donc une sélection des états possibles pour tendre vers un ensemble d'états qui définissent l'équilibre.
Comme il a été décrit précédemment, l'organisation dépend du point de vue, il faut donc définir un critère permettant de vérifier si l'AO générée par le système est une bonne organisation. Pour cela nous étudions



(a) état non organisé (aucune interaction)

(b) $f(S, I, T) = (S + I + T)\%4$ avec $T = 0$

(c) $f(S, I) = (S + I + T)\%4$ avec $T = 1$

FIGURE 1.6 – Exemple de deux organisations d'un même système selon une fonction f identique, mais avec un paramètre extérieur T. Le système défini est le suivant : l'ensemble des états $S = \{0, 1, 2, 3, 4\}$ et les entrées $I = \{2\}$. Les nœuds représentent les états, les flèches les interactions, donc l'organisation entre les états est définie par une fonction f.

les travaux de Henri Atlan dans la section suivante.

Auto-organisation selon Henri Atlan Les travaux de Henri Atlan [6] expliquent l'AO en travaillant sur la notion d'ordre. Un système est auto-organisant si ce système est capable de s'ordonner. Pour cela ses travaux portent sur la définition de l'ordre en physique. Atlan illustre la notion d'ordre de la façon suivante : le désordre dans un système physique est l'état homogène obtenu après avoir secoué le système, alors que l'ordre est un système particulier et hétérogène.

En physique l'entropie permet de mesurer l'ordre, $i.e.$ le lien qui existe entre ses éléments du point de vue de l'énergie. Plus l'entropie est forte plus la probabilité d'être dans un état plutôt qu'un autre est faible. Ainsi, plus l'entropie est forte, moins le système est ordonné, moins le système est auto-organisant. En physique l'AO est un phénomène de mise en ordre croissant, et allant en sens inverse de l'augmentation de l'entropie. Elle peut être donc définie formellement.

Cette notion d'entropie est tout à fait compatible avec les travaux de Ashby. Dire qu'un état est aussi probable qu'un autre signifie que les parties du système soit n'ont aucune interaction, soit s'annulent. Dans tous les cas les parties du système n'ont pas d'influence les unes sur les autres.
Cependant cette notion de l'AO est difficilement utilisable dans d'autres sciences. Prenons l'exemple d'une goutte d'huile que l'on insère dans de l'eau. Des gouttes d'huile auront tendance à se former, d'un point de vue spatial le système semble s'auto-organiser, mais d'un point de vue physique, on a une augmentation de l'entropie.

Pour résoudre ce problème, Atlan s'est penché sur l'entropie dans la théorie de l'information développée par Shanon. Elle est très proche de celle définie en physique. Soit une information contenant n symboles, l'entropie de Shanon est la plus forte si à la position i de l'information tous les symboles ont la même probabilité, si la dispersion des symboles dans l'information est homogène. A l'inverse, si l'entropie est faible, il y a une relation entre les symboles, l'information n'est plus homogène. Cependant que ce soit l'entropie de Shanon ou l'entropie définie en physique, elle repose sur des probabilités d'apparition d'états et peut donc ne pas avoir de signification dans le système.

Atlan montre aussi que la nature est robuste et très bien adaptée. C'est à dire qu'elle sait changer d'organisation en fonction des perturbations. Atlan rejoint Ashby sur la notion des événements extérieurs (comme le bruit) qui permettent au système de s'auto-organiser. Atlan montre que plus un système est redondant, plus il peut s'auto-organiser et s'adapter en détruisant justement cette redondance. La notion de l'AO est très proche de la notion de l'adaptation.

	MN	PC	RP	AC	SMA
Type de données	-	-	-	+	+
Entités	+	+	+	++	+++
Interactions	+	+	+	++	+++
Décentralisation	+	+	+	++	+++
Bruit	-	-	-	+	+
Robustesse	+	++	+	+	+
Détection	++	+	+	-	- -

TABLE 1.1 – Comparaison des MCs selon les caractéristiques de l'auto-organisation et du type de données au travers des exemples utilisés dans ce chapitre. Les MCs étudiés sont les : Méthodes Numériques (MN), la Programmation par Contraintes (PC), les Réseaux de Petri (RP), les Automates Cellulaires (ACs), les Systèmes Multi-Agents (SMAs)

Définition et propriétés de l'auto-organisation Suite aux travaux de Ashby et Atlan nous définissons l'AO de la manière suivante :
L'AO d'un système est sa capacité à s'organiser par lui-même suite à des perturbations extérieures. Un système s'organise si les interactions entre les parties qui le composent, tendent à vérifier une propriété définie par l'observateur.
L'AO d'un système implique :

- qu'il soit composé de plusieurs parties,
- que les parties qui le composent soient en interaction,
- qu'il n'y ait pas de coordination centrale qui impose une organisation,
- que le système soit redondant, robuste,
- qu'il accepte des perturbations extérieures,
- qu'il vérifie une propriété définie par l'observateur (qui peut être l'ordre d'un système défini par l'entropie mais pas seulement).

1.2.2 Choix du modèle de calcul

Les MCs nous permettent de modéliser et simuler l'AO des phénomènes biologiques. Cependant certains d'entre eux sont plus ou moins adaptés, selon le type d'AO souhaitée. Les systèmes modélisés dans cette thèse mettent en jeu les cellules dont l'AO souhaitée décrit la formation de motifs. Prenant en compte ces contraintes, nous comparons les MCs étudiés au début de ce chapitre au travers :

- des données nécessaires pour permettre de modéliser le système dans ce MC,
- de la prise en compte explicite des entités qui composent le système,
- de la prise en compte des interactions entre les entités,
- de la modélisation décentralisée du système,
- de l'insertion d'événements extérieurs ou du bruit,
- de la robustesse du système,
- de la détection automatique de l'AO.

Le tableau 1.1 montre une comparaison qualitative des MCs. Les méthodes numériques, la programmation par contraintes et les réseaux de Petri impliquent (pour les exemples décrits) des données quantitatives. Cela est motivé par le fait que les entités du système sont des concentrations, des populations, et non des individus. A l'inverse, les SMAs et les ACs représentent chaque entité. Les données sont alors qualitatives. En biologie les données quantitatives font souvent défauts.

Pour les mêmes raisons, les SMAs et les ACs représentent les entités et leurs interactions mieux que les trois autres modèles, car chaque entité est modélisée et simulée. Les SMAs ont une meilleure représentation des entités et des interactions que les ACs car chaque entité est représentée non seulement dans l'environnement, mais aussi de manière explicite. Par exemple, dans le système proie-prédateur, le SMA mis en place permet à chaque agent d'être exécuté directement et de manière autonome, alors que dans l'AC les agents sont exécutés en parcourant l'environnement.

Chacun des MCs présentés modélise de manière décentralisée le système car le système est composé de parties, chaque partie ayant sa propre fonction pour déterminer son évolution. Par exemple, dans le système proie-prédateur, une équation différentielle est définie pour chaque population du système. Cependant les SMAs et les ACs permettent une décentralisation du système plus fine, $i.e.$ que le système peut être découpé dans des parties plus précises (les entités et non des populations d'entités).

Avec les méthodes numériques, la programmation par contraintes et les réseaux de Petri, il n'est pas trivial d'insérer des événements extérieurs. Par exemple, ajouter des probabilités dans les ACs et SMAs se fait de façon directe ($cf.$ les probabilités sur les comportements définis dans le listing 1.1). De même, ajouter des événements extérieurs, comme des contraintes sur l'environnement, se fait de façon plus aisée dans les SMAs et ACs, comme les niches dans le système proie-prédateur.

La robustesse dans les systèmes biologiques peut être vue de plusieurs manières. Dire qu'un système est robuste signifie qu'il accepte un jeu de paramètres valides. La programmation par contraintes est le meilleur choix, car il ne fixe pas les paramètres et fournit un ensemble de valeurs de paramètres qui satisfait les données biologiques. Si la redondance du système biologique est effective par le nombre d'entités mis en jeu, les méthodes numériques, la programmation par contraintes et les réseaux de Petri peuvent modéliser, pour les exemples utilisés, cette robustesse. Ces modèles prennent en paramètre des concentrations, donc un nombre important d'entités est pris en compte (de façon implicite). Il nous semble que les ACs et SMAs sont pertinents pour modéliser et simuler la robustesse des systèmes biologiques.

La détection automatique de l'auto-organisation peut se faire facilement si le formalisme du MC repose sur les mathématiques car les outils existent déjà pour trouver des états stables, des attracteurs dans les modèles continus et discrets. La détection automatique est plus efficace dans les méthodes numériques, que dans les réseaux de Petri et la programmation par contraintes, que dans les ACs. Dans les SMAs cela est difficile car plusieurs formalismes peuvent être utilisés pour définir les comportements des agents.

Les SMAs et les ACs sont de bons candidats pour modéliser et simuler l'auto-organisation des cellules. Les critères qui motivent nos choix sont la représentation des entités et des interactions dans ces MCs. De plus, l'auto-organisation étudiée dans cette thèse est une organisation spatiale des cellules. Comme nous l'avons montré, il est difficile de représenter des propriétés spatiales dans les autres MCs. Ainsi les points forts des autres MCs ne peuvent être réellement utilisés dans l'auto-organisation souhaitée.

Chapitre 2

Choix du modèle utilisé : Le Modèle de Potts Cellulaire

Ce chapitre décrit et formalise (2.1) les Automates Cellulaires (ACs) et les Systèmes Multi-Agents (SMAs), et décrit le Modèle de Potts Cellulaire qui nous semble approprier pour modéliser l'auto-organisation (2.2).

2.1 Formalisation des Systèmes Multi-Agents et Automates Cellulaires

Dans cette section nous définissons et formalisons les ACs et les SMAs. Cette section décrit aussi la notation utilisée dans ce mémoire.

2.1.1 Définition d'un Automate Cellulaire

Les ACs sont des modèles de calculs contemporains. Nous rappelons les travaux à l'origine de leurs conceptions (2.1.1) et nous donnons une définition formelle (2.1.1).

Histoire

Un AC est constitué d'une grille et d'une fonction de transition appliquée à tous les sites de la grille. Chaque site contient une valeur. Cette fonction prend en paramètre la valeur d'un site et la valeur des sites voisins. Ainsi le modèle d'Ising, un modèle physique défini en 1925 par Ising, décrit le phénomène de ferromagnétisme et peut être vu comme un AC. Dans ce modèle, les entités sont des spins qui ont deux orientations possibles. L'orientation d'un spin dépend de l'orientation des spins voisins.

Cependant, traditionnellement les travaux à l'origine des ACs sont attribués à John von Neumann dans les années 40. En effet, von Neumann a étudié et formalisé ce modèle de calcul en concevant les automates auto-reproductibles. Les automates auto-reproductibles sont des ACs avec des structures auto-réplicantes. Ces travaux ont permis à John Horton Conway de développer au début des années 70 le célèbre jeu de la vie. Le jeu de la vie peut être vu comme une analogie à la biologie. Si une cellule est vivante et que le nombre de cellules voisines vivantes est supérieur à 3 la cellule meurt à cause d'un « étouffement ». Un autre cas est quand le nombre de cellules vivantes est inférieur à 2 dû à un « dépérissement ». Une cellule naît si le nombre de cellules voisines vivantes est égal à 2.

Les travaux de Christopher Langton dans les années 80[48, 49] continuent l'étude des automates auto-reproductibles notamment en définissant les fourmis de Langton en 1986. La fourmi de Langton consiste en une « fourmi » qui tourne à gauche et avance d'une case si elle se trouve sur une case blanche

et met cette case en noir, sinon met la case en blanc tourne à droite et avance d'une case. Cette simple description prouve qu'un AC dont les règles parfois simples implique des dynamiques complexes et peut générer une structure qui s'auto-réplique. La fourmi de Langton et le jeu de la vie sont décrits plus en détail dans la section suivante. Une formalisation des ACs est aussi présentée dans la section suivante.

Nous pouvons aussi citer les travaux de Stephen Wolfram[86] dans les années 80 sur les ACs élémentaires c'est à dire un AC à une dimension où les sites peuvent contenir une valeur booléenne et les voisins sont les 2 voisins directs. Il démontra entre autre que ce type d'AC est Turing-complet c'est à dire qu'il peut coder une machine de Turing. En effet de nombreux ACs sont Turing-complet, ce qui montre leur pouvoir d'expression.

Formalisation

Un AC peut être défini par le quadruplet suivant (d, Q, V, F_l) où :

- d est la dimension de la grille de l'AC.
- Q est l'ensemble des valeurs que peut prendre un site de la grille
- V une fonction de voisinage.
  $V : x \rightarrow \{x'\}$ où $x =< x_1, \ldots, x_i, \ldots, x_d >$, $x_i \in \mathbb{Z}$, x représente un site de la grille
- F_l une fonction de transition locale, $i.e$ d'un site.
  $F_l :< s_x^t, \{s_{x'}^t\} >\rightarrow s_x^{t+1}$ où $s_x \in Q$ est la valeur du site x

Un état $S = \{s_x\}$ dans les ACs est l'ensemble des valeurs associées à chaque site de la grille. On définit alors une fonction de transition d'un état à l'instant t à l'état à l'instant $t+1$ par F_t.
Soit s_x^t la valeur du site x à l'instant t, F_t est définie par :

$$F_t(S^t) = \bigcup_{s_x^t \in S^t \wedge n=\{s_{x'}^t\} | V(x)=\{x'\}} F_l(s_x^t, n)$$

Nous illustrons cette formalisation en l'appliquant au jeu de la vie et aux fourmis de Langton. Le jeu de la vie peut être formalisé de la façon suivante :

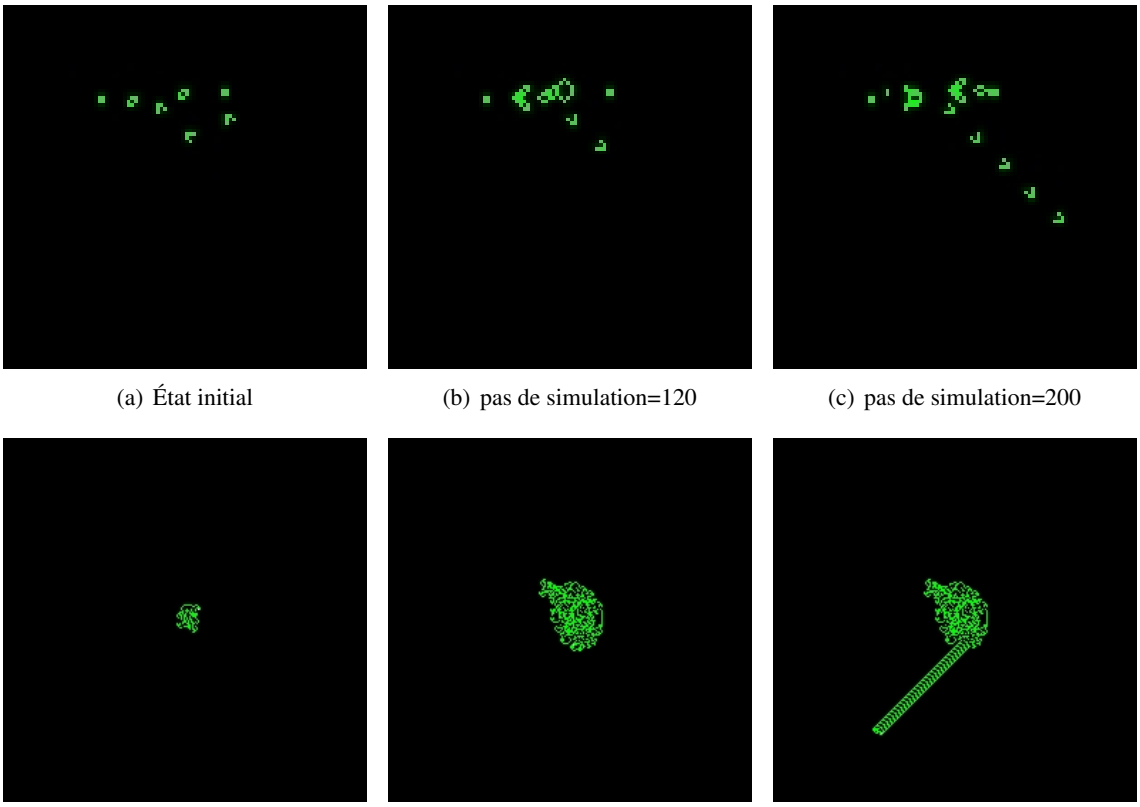
- $d = 2$
- $Q = \{0, 1\}$ (0 pour une cellule morte, 1 pour une cellule vivante)
- $V(< x_1, x_2 >= \{< x_1 + 1, x_2 >, < x_1 + 1, x_2 + 1 >, < x_1 + 1, x_2 - 1 >, < x_1 - 1, x_2 >, < x_1 - 1, x_2 + 1 >, < x_1 - 1, x_2 - 1 >, < x_1, x_2 + 1 >, < x_1 + 1, x_2 - 1 >\}$ (un site a huit voisins)
- $F_l(< s_x^t, n >= \{s_{x_1+1,x_2}, s_{x_1+1,x_2+1}, s_{x_1+1,x_2-1}, s_{x_1-1,x_2}, s_{x_1-1,x_2+1}, s_{x_1-1,x_2-1}, s_{x_1,x_2+1}, s_{x_1+1,x_2-1}\}) =$
  - 1 si $\#\{s | s = 1 \wedge s \in n\} = 3$ (reproduction)
  - s_x^t si $\#\{s | s = 1 \wedge s \in n\} = 2$ (stabilité)
  - 0 sinon (mort)

Les fourmis de Langton peuvent être formalisées de la façon suivante :

- $d = 2$
- $Q =< q_1, q_2, q_3 >$ où $q_1 \in \{0, 1\}$, $q_2 \in \{0, 1\}$, $q_3 \in \mathbb{N}$
  (q_1=1 la fourmi est présente, q_2=1 on est sur une case blanche, q_3 donne la direction de la fourmi)
- $V(< x_1, x_2 >) = \{< x_1 + 1, x_2 >, < x_1 - 1, x_2 >, < x_1, x_2 + 1 >, < x_1, x_2 - 1 >\}$
  (les voisins d'un site sont les quatre voisins directs)

- $F_l(< s_x^t =< q_1, q_2,_q 3 >, n = \quad \{s_{x_1+1,x_2}, s_{x_1-1,x_2}, s_{x_1,x_2+1}, s_{x_1,x_2-1}\}) =$
  $< 1, q_2, q_3' + 90 >$ si $\exists s_{x'} \in ns_{x'} =< 1, 1, q_3' > \wedge rot(x', q_3' + 90) = x$
  $< 1, q_2, q_3' - 90 >$ si $\exists s_{x'} \in ns_{x'} =< 1, 0, q_3' > \wedge rot(x', q_3' - 90) = x$
  $< 0, 1, 0 >$ si $q_1 = 1 \wedge q_2 = 1$
  $< 0, 1, 0 >$ sinon
  $rot(< x_1, x_2 >, \theta) =< x_1', x_2' > \Leftrightarrow \quad \theta = 0 \Leftrightarrow x_1 = x_1' - 1 \wedge x_2 = x_2'$
  $\theta = 90 \Leftrightarrow x_1 = x_1' \wedge x_2 = x_2' - 1$
  $\theta = 180 \Leftrightarrow x_1 = x_1' + 1 \wedge x_2 = x_2'$
  $\theta = 270 \Leftrightarrow x_1 = x_1' \wedge x_2 = x_2' + 1$

La figure 2.1 montre et décrit la dynamique d'une simulation du jeu de la vie et une simulation de la fourmi de Langton.



(a) État initial (b) pas de simulation=120 (c) pas de simulation=200

(d) pas de simulation=1000 (e) pas de simulation=10000 (f) pas de simulation=13000

FIGURE 2.1 – Simulation du jeu de la vie et de la fourmi de Langton. Les images 2.1(a), 2.1(b), 2.1(c) montrent un canon à planeurs. Un planeur est une structure qui se déplace dans le jeu de la vie. Cette simulation montre bien que les interactions locales et simples dans un AC peuvent générer et créer des structures intéressantes qui peuvent être vues aussi comme une auto-organisation. Les images 2.1(d), 2.1(e), 2.1(f) montrent la fourmi de Langton, où les traces laissées par la fourmi sont en vert et la fourmi en blanc. Comme pour le jeu de la vie, la fourmi de Langton présente la possibilité de générer des structures auto-réplicantes. Elle montre en plus comment une dynamique qui semble être aléatoire (2.1(d), 2.1(e)) peut amener soudainement une auto-organisation. Ces simulations ont été réalisées grâce au logiciel MatrixStudio. La simulation du jeu de la vie peut être vue à l'adresse `http://youtu.be/zwC7W-eJ44U` et celle de la fourmi de Langton à l'adresse `http://youtu.be/MT8dgzOoQAU`

2.1.2 Définition d'un Système Multi-Agents

Cette section a pour but de présenter les SMAs. Les SMAs sont des systèmes qui mettent en jeu un ensemble d'agents (entités) en interaction les uns avec les autres et sont exécutés de manières autonomes dans un environnement. Nous présentons tout d'abord l'origine des SMAs, notamment l'Intelligence Artificielle Distribuée (IAD) et quelques travaux historiques. Puis nous décrivons de façon plus précise les SMAs via l'approche voyelle qui permet de définir et concevoir un SMA .

Origine des SMAs.

Les SMAs correspondent à un paradigme à la fois de programmation et à la fois de modélisation. Les SMAs décrivent au minimum un système à partir des entités qui le composent (les agents), des interactions entre ces entités (qu'elles soient directes ou non), et de l'environnement dans lequel les entités évoluent. L'évolution du système est une suite d'états dont le successeur d'un état est la modification de cet état par les entités. Les SMAs ont été élaborés fin des années 70-début des années 80 et ont pour principale origine l'IAD. L'IAD utilise les approches de l'intelligence artificielle classique pour les transposer sur des architectures distribuées.

Nous exhibons deux types de travaux (souvent cités dans la littérature notamment par Jacques Ferber[30]). L'IAD comme le partage de ressource via un tableau noir [53] ou comme l'envoi de message via les acteurs [39].

Dans [53], les tableaux noirs sont utilisés pour la reconnaissance de la parole. Le but est le suivant : supposons un tableau où sont regroupées toutes les données nécessaires à la résolution du problème initial. On définit un ensemble de modules indépendants les uns des autres, appelés KS («Knowledge Source»), qui ne communiquent pas entre eux mais interagissent en partageant les informations via un tableau noir. Par exemple, le KS nommé «syntatic parseur» permet, à l'aide d'une syntaxe donnée, la construction d'une phrase à partir des mots se trouvant sur le tableau noir. Ces mots sont déposés par d'autres KSs, à la suite de nombreux processus dont l'origine est l'analyse sonore. Notons qu'il peut être nécessaire d'avoir un dispositif de contrôle pour gérer les conflits entre les KSs.

Au niveau des échanges de données, nous pouvons citer les travaux sur les «actors» [39]. Un acteur possède une quantité limitée de connaissances à laquelle on lui associe un comportement (un «script»). Il contient une interface permettant de le décrire et a la capacité d'envoyer des messages et de déléguer des tâches. Notons aussi les travaux sur les «beings» (êtres) [50] qui sont les premiers agents autonomes, ils sont capables (à l'aide d'un groupe de spécialistes) de réaliser une tâche par eux-mêmes.

La littérature met en avant trois applications historiques.
La première application est le DVMT («Distributed Vehicle Monitoring Testbed») [52]. Le but de ce système est de suivre les mouvements des véhicules à l'aide de capteurs. Les informations récoltées par les capteurs sont enregistrées sur un tableau noir. La principale difficulté dans ce système est de rendre cohérent l'ensemble des données collectées par les capteurs, afin d'extraire un modèle du trafic routier.

La seconde application est Mace [51] une plateforme générique, où il est décrit explicitement comment mettre en place un système d'IAD. Mace comprend les composants suivants : une collection d'agents, une communauté d'agents systèmes, un ensemble de fonctionnalités qui peuvent être utilisées par les agents, une description de la base de données et une collection de noyaux c'est à dire tout ce qui est nécessaire (comme un ordonnanceur).

La troisième application est Contract Net [73] un protocole gérant la répartition des tâches et repose sur le principe d'appel d'offres. Lors d'une tâche à effectuer, un administrateur envoie une description de la tâche à l'ensemble des agents (les contractants potentiels) ou à un groupe particulier. Les agents soumettent une proposition à l'administrateur qui choisit parmi les soumissions (selon un critère). L'agent qui a reçu le marché devient le contractant et confirme qu'il accepte la tâche.

Les SMAs se caractérisent donc comme l'IAD par le fait qu'il n'y ait pas de contrôle centralisé. Dans l'IAD les entités (ou agents) modifient l'état courant de façon planifiée. Un exemple d'entités dans l'IAD peut être un système expert. Dans les SMAs, les agents modifient l'état courant soit de façon planifiée (agent cognitif), soit de façon non planifiée ou sans but (agent réactif), ou soit de manière mixte. La description d'un système juste à l'aide d'agents réactifs peut impliquer, inférer ou faire émerger une organisation, des propriétés non imposées dans la description. C'est ce dernier point qui caractérise fortement les SMAs. Dans la section suivante nous définissons les SMAs comme paradigme à part entière.

Approche classique des systèmes multi-agents : l'approche voyelle

Pour présenter de façon plus précise les SMAs, nous choisissons de décrire les SMAs suivant l'approche voyelle proposée par Yves Demazeau [22]. Cette approche permet d'étudier et de concevoir un SMA sous 4 aspects : agent, environnement, organisation et interaction. Nous étudions ces 4 aspects dans les sous sections suivantes en faisant référence à la littérature.

Agent La définition d'agents n'étant pas figée, nous en donnons une définition dite «minimaliste» [30]. On appelle agent une entité physique ou virtuelle :

- qui est capable d'agir dans un environnement.
- qui peut communiquer directement avec d'autres agents.
- qui est mue par un ensemble de tendances (sous la forme d'objectifs individuels ou d'une fonction de satisfaction, voire de survie, qu'elle cherche à optimiser).
- qui possède des ressources propres.
- qui est capable de percevoir (mais de manière limitée) son environnement.
- qui ne dispose que d'une représentation partielle de cet environnement (et éventuellement aucune).
- qui possède des compétences et offre des services.
- qui peut éventuellement se reproduire.
- dont le comportement tend à satisfaire ses objectifs, en tenant compte des ressources et des compétences dont elle dispose, et en fonction de sa perception, de ses représentations et des communications qu'elle reçoit.

Il n'y a pas de consensus autour d'une définition d'un agent par le fait de leurs hétérogénéités. Aussi les différents points de cette définition caractérisent plus ou moins les agents en fonction de leurs types. De façon classique nous pouvons répertorier deux types d'agents, les agents réactifs et les agents cognitifs.

Dans [27] une première définition d'un agent réactif peut être établie en reprenant les notions de psychologies comportementales sur le schéma «S-R» (Stimulus-Reaction). Dans ce schéma, un stimuli est un état de l'environnement contenant l'agent et la réaction une séquence d'action basique. Cette définition suppose qu'il n'existe pas de «raisonnement» (ou de planification) entre le stimuli et la réaction. Dans les SMAs ne contenant que des agents réactifs, des propriétés sur la dynamique, sur l'organisation, peuvent être impliquées ou émergées. Ceci est illustré dans le projet MARCH (Multi-Agents Reactive CHess) [26] qui montre que les interactions entre agents réactifs permettent l'émergence d'une stratégie pour jouer aux échecs. Ces propriétés sont aussi retrouvées dans les colonies de fourmis [19]. Notons aussi qu'ils n'ont pas de représentation d'eux mêmes et des autres. Ces systèmes se différencient donc fortement de l'IAD où les entités ont une certaine capacité comme les agents cognitifs.

Les agents cognitifs à l'inverse des agents réactifs ont un but explicite (ou ont un certain «degré d'intelligence»). Un exemple d'agents cognitifs sont les agents BDI («Belief Desire Intention»), des agents dits intentionnels développés au début des années 90. Dans [65] un agent BDI a une certaine représentation mentale de la croyance, du désir, et de l'intention. Ces trois notions représentent, respectivement, l'état de l'agent c'est à dire sa connaissance du monde (il peut notamment avoir une connaissance sur lui-même

Relation au monde **Conduites**	Agents cognitifs	Agents réactifs
Téléonomiques	Agents intentionnels	Agents pulsionnels
Réflexes	Agents «modules»	Agents tropiques

TABLE 2.1 – Les différents types d'agent en fonction de leurs capacités représentationnelles et de leurs modes de conduite [30].

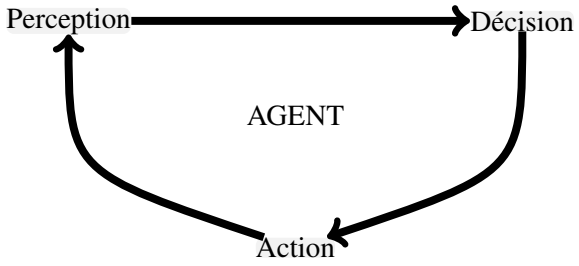


FIGURE 2.2 – Cycle d'exécution d'un agent.

et sur les autres agents), son état motivé (son but) c'est à dire l'état qu'il voudrait atteindre, et son état délibératif, son état futur.

Nous pouvons donc constater que les agents cognitifs et réactifs sont radicalement opposés. L'un est capable de raisonner de négocier d'interagir directement ou non avec les autres agents, l'autre dénué «d'intelligence», ne peut réagir avec les autres agents que par l'intermédiaire de l'environnement.

Cependant il existe d'autres types d'agents, notamment des agents hybrides qui reprennent à la fois les notions d'agents réactifs et les notions d'agents cognitifs.
Quatre types d'agents peuvent être définis [30] :

- des agents intentionnels comme les agents BDI.
- des agents «modules», des agents cognitifs qui peuvent accomplir des tâches, sans que ces tâches soient des buts explicites de l'agent comme des réflexes.
- des agents pulsionnels, des agents réactifs qui peuvent être dirigés par des mécanismes de motivation comme maintenir un niveau d'énergie.
- des agents tropiques, des agents réactifs seulement définis par le schéma Stimuli-Réaction.

Ceci est résumé par le tableau 2.1.

Indépendamment du type d'agents, tous ont un cycle d'exécution qui peut être divisé en trois parties ($cf.$ Fig. 2.2) : Perception, Décision, Action. La perception est la capacité de l'agent à percevoir son environnement. Dans le cas d'agents réactifs sa perception est souvent locale, dans le cas d'agents cognitifs sa perception peut être plus large. La décision est le choix que va faire l'agent pour définir son action. Cela peut être juste un réflexe ou un mécanisme plus élaboré. Enfin, l'action est son influence sur l'environnement et sur les agents, comme ajouter ou supprimer des informations dans l'environnement.

Environnement L'environnement dans un système multi-agents a de nombreuses définitions et dont le rôle n'est pas unique comme cela est décrit dans [85] : l'environnement comme un milieu pour la coordination, l'environnement comme un monde extérieur («external world»), ... Pour donner une définition minimale de l'environnement, nous reprenons la description faite dans [68] ou deux points de vue sont exhibés : celui du système multi-agents et celui de l'agent. Du point de vue du système multi-agents l'environnement est le support des actions des agents. Du point de vue de l'agent, l'environnement

est le support des actions des agents mais aussi des autres agents.

Interaction Les interactions dans les SMAs ont un rôle primordial. Elles sont responsables de la complexité de la dynamique (l'émergence d'une certaine intelligence). Ainsi les interactions entre les agents permettent plus que la somme de leurs actions. D'après [30] ont peut classer les principales situations d'interaction suivant trois critères :

- buts compatibles et incompatibles.
- relations aux ressources.
- capacités des agents par rapport aux tâches.

Chacun de ces critères peut prendre deux états (but : compatible/incompatible, ressources : suffisantes/insuffisantes, capacités : suffisantes/insuffisantes), ce qui nous amène à huit situations possibles (Indépendance, Collaboration simple, Encombrement, Collaboration coordonnée, Compétition individuelle pure, Conflit individuel pour des ressources).

Il est aussi possible de classifier les interactions entre les agents selon qu'elles soient directes ou non. Les interactions indirectes correspondent principalement aux interactions entre agents qui passent par l'environnement, c'est à dire les informations laissées par les agents. Ces informations sont ensuite utilisées par les autres agents. Les interactions directes signifient une communication entre les agents. Ce type d'interaction implique que chaque agent partage un langage commun. Dans cette thèse nous nous intéressons aux interactions indirectes car nous supposons que nos cellules ne communiquent pas entre elles mais sont dépendantes des molécules qui peuvent être produites par d'autres cellules.

Organisation L'organisation permet de structurer l'ensemble des entités. Dans [31] cinq caractéristiques sont exhibées :

1. Une organisation est constituée d'agents qui manifestent un comportement.
2. Une organisation peut être partitionnée (les partions peut se chevaucher). Chaque partition est appelée un groupe.
3. Les comportements des agents sont fonctionnellement liés à l'organisation (notion de rôle).
4. Les agents sont engagés en relation dynamique (également appelé schémas d'activités) qui peut être « typée » au moyen d'une taxonomie des rôles, des tâches ou des protocoles, ce qui décrit une sorte de supra-individualité.
5. Les types de comportements sont liés par les relations entre les rôles, tâches et protocoles.

Cependant cette organisation peut aussi être émergente. Cette thèse s'intéresse à ce type d'organisation. En effet, nous prenons l'hypothèse que les cellules (les agents) n'ont pas conscience des autres, donc qu'il n'y a pas de coordination centrale.

Les auteurs de [13] référencent l'auto-organisation dans quatre types différents de SMAs. Premièrement, dans le cas des agents réactifs, une organisation est possible si elle émerge des interactions et non par un agent qui coordonne les autres. L'exemple typique est la colonie de fourmis [27]. Deuxièmement, dans le cas d'agents gérant l'information de manière coopérative. Par exemple, ces agents sont localisés dans l'environnement et forment alors des connexions à travers lesquelles les messages sont passés. Le troisième cas est la coopération dans les « Adaptive Multi-Agents Systems » (AMAS), c'est à dire que le système s'auto-organise pour s'adapter aux changements venant de son environnement. Trois meta-règles sont définies dans ce contexte : la réception des signaux est perçue sans ambiguïté, l'information reçue est indispensable pour le raisonnement de l'agent et le raisonnement amène à des actions utiles vers les autres agents. Uniquement la capacité de coopération des agents et la vue locale influencent les interactions entre les agents du système. Dernièrement, les holons, sont des structures récursives composées de plusieurs holons, permettant au système de fonctionner comme un tout. Par exemple, dans [13] le comportement

d'un holon est décrit comme ayant quatre rôles : rester seul, être en tête, être une partie ou être multiparties. Un holon va rester dans l'état « seul » tant qu'il est satisfait. Il peut être « en tête » ce qui implique le fait qu'il peut prendre une décision pour tous les autres membres. Si le holon n'est pas en tête cela signifie qu'il a le rôle d'une partie, et il est sous le contrôle du holon « tête ». Les holons ayant le rôle multiparties sont des holons avec le rôle partie, mais dans plusieurs holons.

Où est passé le U ? Nous pouvons rajouter dans l'approche voyelle la lettre U pour utilisateur [77]. C'est à dire le fait que l'utilisateur peut interagir avec le système multi-agents en cours d'exécution. Par exemple dans [23], un système multi-agents est établi pour modéliser une réaction allergique au niveau de l'épiderme. Un objet graphique sonde et un objet graphique seringue, permettent à l'utilisateur de pouvoir connaître la concentration d'espèce dans un espace limité, et de pouvoir injecter une certaine quantité d'espèce chimique dans l'environnement au cours de la simulation.

2.2 Choix du modèle de Modèle de Potts Cellulaire

Cette section a pour but d'extraire le SMA ou l'AC le plus adapté pour modéliser des cellules en interaction. Pour cela, nous décrivons les ACs appliqués à la biologie (2.2.1). Dans une voie similaire nous traitons les SMAs (2.2.2). Finalement, nous montrons pourquoi le Modèle de Potts Cellulaire (CPM) est le meilleur choix.

2.2.1 Automates Cellulaires appliqués à la biologie

Les ACs, malgré leur simplicité, permettent de faire émerger des structures complexes. Le jeu de la vie développé par John Conway [34] ou les fourmis de Langton [48] en sont de parfaits exemples ($cf.$ 2.1.1). Ces automates sont des automates déterministes, cependant d'autres ACs ont été définis comme : les « Lattice gaz » (LGA).

Les LGAs sont des automates cellulaires qui ont été appliqués en premier lieu pour simuler la mécanique de fluides. Comme les ACs traditionnels, ils sont composés d'une grille où chaque site a un certain état. Ce qui les différencie, c'est le fait que les états sont des particules avec des vélocités. Les règles d'évolution sont comme dans les ACs classiques, mais deux problèmes sont à résoudre : la collision et la propagation. En effet, comme les particules sont en mouvement ou se propagent, il est possible qu'il y ait des collisions entre les particules. Ces ACs ont été utilisés pour modéliser les interactions d'adhésion entre les cellules [2], la chimiotaxie et la croissance de tumeur [24]. Des LGAs dans un environnement continu existent aussi comme décrit dans [2]. Ces LGAs ont permis de modéliser l'agrégation et le branchements dans des motifs.

Dans cette thèse, nous ne retenons pas les LGAs, car il faut gérer les problèmes de collision et de propagation. Dans les ACs plus classiques, nous pouvons citer Prebiodyn. Prebiodyn est un automate qui modélise le jeu de la vie avec une force de gravitation et de l'agitation thermique. Dans [38] les ACs sont utilisés pour modéliser le problème « French flag ». Ce problème consiste à réguler la production de deux protéines, sachant que leur synthèse est variable selon les endroits cellulaires : antérieur et postérieur.

Dans les ACs classiques, le Modèle de Potts Cellulaire est un des automates cellulaires le plus utilisé pour modéliser la cellule. Dans cet AC les sites de la grille contiennent des entiers qui permettent d'identifier les cellules. Une cellule est alors représentée, de façon implicite, comme l'ensemble des sites contenant son identifiant. Des énergies de volume, surface et de contact sont définies simplement, permettant d'assurer dans ce modèle une forme cellulaire viable. Plusieurs modèles de morphogénèse ont été simulés depuis le CPM comme le tri cellulaire. Dans les ACs c'est ce modèle qui nous semble le plus

intéressant pour modéliser les interactions cellulaires et pour générer des AOs, de part sa simplicité, et de part les travaux déjà réalisés.

2.2.2 Systèmes Multi-Agents appliqués à la biologie

Dans cette section nous illustrons les SMAs qui modélisent des systèmes biologiques très variés. Puis, nous décrivons plus en détail les SMAs dont les agents sont les cellules.

La biologie, une application typique des SMAs

Une raison d'utiliser les SMAs pour modéliser les systèmes biologiques est le fait qu'ils ont des propriétés communes. Dans [16, 68] les travaux de Hiroaki Kitano, sur ce qui caractéristique un système biologique, sont utilisés. L'utilisation des SMAs est justifiée dans [68] en montrant que les caractéristiques des systèmes biologiques, décrites dans [43] : la robustesse, l'émergence, l'auto-organisation et l'adaptabilité, sont compatibles avec les SMAs. En effet ces caractéristiques sont en partie retrouvées dans la description faite en section 2.1.2. Un cadre de travail pour les SMAs est construit dans [16] en utilisant les concepts développés par Kitano.

Plusieurs SMAs sont utilisés en biologie comme il est décrit dans [59] qui relate le rapport d'activité sur le groupe de travail des agents en bio-informatique (BIOAGENTS). Cette article met en avant divers SMAs appliqués à des domaines très variés de la biologie : des agents pour l'analyse des maladies polygéniques, des agents permettant de construire une ontologie, des agents pour prédire la structure des protéines, ...

Les SMAs peuvent être utilisés pour la biologie autrement que pour la simulation comme dans [33] qui décrit un couple web sémantique-SMA pour la gestion de données en biologie ou comme dans [32] où un système multi-agents est utilisé pour le traitement d'image médicale.

Cependant, nous nous intéressons ici aux SMAs pour modéliser et simuler des systèmes biologiques. Nous donnons une liste non exhaustive de certains de ces SMAs pour :

- modéliser des réactions chimiques : Smoldyn [3] dans un environnement en 3D et continu.
- modéliser l'évolution de population : Bacsim pour l'évolution de la croissance bactérienne [45] dans un environnement en 2D et continu, [55] pour l'essaimage de crustacés dans un environnement avec des attracteurs où chaque agent est représenté par une équation stochastique (mouvement brownien) dans un environnement en 2D et continu,
- modéliser et prédire la structure protéine-protéine : 3DSpi [74] dans un environemment en 3D et continu.
- modéliser la migration cellulaire : SymBiodyn [8] dans un environnement en 3D, aqueux suramorti et continu.

Ces SMAs sont spécialisés dans la description de phénomène biologiques, cependant des plateformes génériques ont rendu possible la simulation de processus biologique comme :

- ARéVi (Atelier de Réalité), un SMA pour modéliser et simuler des entités autonomes dans un environnement 3D. Il a été utilisé pour simuler les réactions allergiques de l'épiderme [23].
- Netlogo, un SMA pour modéliser et simuler des phénomènes naturels et sociaux. Il a été utilisé par exemple pour le «switch» du phage λ [40].

Cette section présente plusieurs SMAs appliqués à la biologie. Pour pouvoir comparer les résultats entre les différents SMAs mais aussi avec les autres approches quelques normes ont été mises en place afin de décrire les systèmes biologiques : SBML (Systems Biology Markup Language) `http://sbml.org/`, CellML `http://www.cellml.org/`, ...

Les agents cellules

Dans ce mémoire, nous nous intéressons aux systèmes multi-agents qui permettent de modéliser les cellules et leurs interactions. Nous pouvons distinguer deux types de modélisation de cellules, les cellules avec une forme rigide, et les cellules avec une forme dynamique. Pour les SMAs modélisant une forme rigide nous pouvons citer les travaux de Agarwal qui a défini le Cell Programming Language (CPL) [1]. Le CPL définit le programme que doit exécuter une cellule, cela peut être considéré comme son génome. De nombreux comportements sont disponibles comme la forme, les concentrations des molécules présentes dans la cellule, la différentiation cellulaire, ... Cependant la forme de la cellule reste rigide, sa forme n'est pas élastique et elle est approximée par un un polygone. Nous pouvons citer aussi le Morphoblock [9], un agent ayant un cœur qui occupe une unité physique et pouvant sécréter et produire des molécules pour modéliser la morphogenèse. De nombreux SMAs existent dans le cas d'agent ayant une forme rigide mais nous nous intéressons ici aux cellules ayant une forme dynamique car nous pensons que la déformation des cellules due à la pression des autres cellules sur elle, peut amener à des AOs. Ces AOs ne seraient peut être pas visibles si cette interaction n'était pas prise en compte. De plus nous pensons aussi que les forces physiques permettent la morphogenèse. Ces forces peuvent émerger des forces emmagasinées dans la forme des cellules lors de la déformation de celle-ci.

Pour les SMAs qui modélisent une forme dynamique nous pouvons citer les travaux de Lionel Dupuy sur la morphogenèse des plantes où la forme de la cellule varie en fonction des flux de molécules. Sa description repose en partie sur des équations différentielles [28]. Nous pouvons citer aussi FlexCell, où la migration cellulaire est modélisée [8]. Un agent cellule est défini. La cellule est représentée par un ensemble de ressorts qui permettent de définir la structure de la cellule. La migration se fait sur un substrat et des agents spécialisés permettent de faire le lien entre le substrat et les ressorts. Les travaux de Doursat [25] définissent des agents cellules où chaque cellule est reliée à ses voisines par des ressorts. La forme de la cellule est déduite depuis ces ressorts. La cellule a de nombreux comportements comme l'intégration d'un réseau génétique. Les SMAs appliqués aux cellules avec une forme dynamique sont beaucoup moins étudiés que ceux avec une forme rigide, et souvent leurs mises en œuvre impliquent des temps de simulation importants comme ceux à base d'ODE ou comme FlexCell où d'autres agents doivent être définis.

2.2.3 Caractéristiques du Modèle de Potts Cellulaire

Les études des ACs et des SMAs appliqués à la modélisation et à la simulation biologique montrent deux caractéristiques. Les modèles de cellules avec une forme déformable sont peu nombreux par rapport à ceux avec une forme rigide. Dans le cas des modèles avec une forme rigide la simulation de ces modèles coûte cher en temps de calcul car ils ont soit des représentations mathématiques comme des équations différentielles soit des structures complexes comme dans[8]. Le critère de temps de simulation d'une cellule est un critère primordial car l'auto-organisation implique la simulation d'un grand nombre de cellules.

Le CPM est celui qui à nos yeux donne le meilleur compromis entre le temps d'exécution et le niveau de modélisation de la cellule. En effet cet AC représente la cellule avec une forme dynamique, la forme n'étant imposée que par un volume cible et une surface cible. Ce volume et surface cible sont le volume et la surface que veut atteindre la cellule, mais elle peut s'éloigner de ces valeurs si les autres cellules exercent une pression sur elle. La pression des cellules entre elles est émergente du fait que chacune veut atteindre son volume et surface cible. Pour déterminer l'évolution du volume et de la surface d'une cellule, une énergie est déterminée comme la différence au carré entre le volume courant (surface courante) et le volume cible (surface cible). Ainsi l'évolution d'un cellule est décrite de manière simple et facilement calculable.

En conclusion le choix du CPM pour étudier l'auto-organisation cellulaire nous semble le plus approprié. Même s'il a été utilisé dans de nombreuses applications nous montrons dans la partie suivante que certains comportements cellulaires doivent être mis en œuvre afin de simuler des théories biologiques non encore étudiées avec ce modèle. Dans la partie suivante nous traduisons le CPM en SMA, notamment pour permettre une évolution de celui-ci de manière plus aisée grâce au fait que les agents sont représentés explicitement.

Deuxième partie

Étude, évolution et mise en œuvre du Modèle de Potts Cellulaire vers un Système Multi-Agents : Application à la Morphogénèse

Chapitre 1

Présentation formelle du Modèle de Potts Cellulaire

Ce chapitre est destiné à présenter le Modèle de Potts Cellulaire. Pour cela nous présentons ses origines (1.1), puis nous donnons une définition formelle en se basant sur le formalisme des automates cellulaires (1.2). Finalement, nous montrons les extensions déjà proposées et nous décrivons certaines qui ne sont pas encore faites mais qui nous semblent importantes pour modéliser l'auto-organisation entre les cellules (1.3).

1.1 Historique

Le Modèle de Potts Cellulaire (CPM) est un automate cellulaire établi par James A. Glazier et Francois Graner en 1992 [36], dans le but de modéliser et simuler des phénomènes qui se produisent lors de l'embryogenèse. Le CPM est une extension du Modèle de Potts développée par Renfrey B. Potts en 1952 qui lui-même est une généralisation du modèle d'Ising développé par Ernst Ising en 1925 comme décrit dans [87]. Ces deux modèles reposent sur des notions de physique statistique. Dans cette section nous décrivons ces notions nécessaires à la compréhension de ce mémoire (1.1.1), et nous décrivons le modèle d'Ising (1.1.2) et de Potts (1.1.3) de manière formelle.

1.1.1 Notions de Physiques

Dans les articles [35, 57] décrivant le CPM et leurs applications nous retrouvons souvent le terme Hamiltonien. C'est une notion qui provient de la mécanique décrite par Hamilton qui correspond à la mécanique lagrangienne au travers de la transformée de Legendre. Lagrange propose en 1788 une façon nouvelle de considérer les problèmes de mécanique. Au lieu de déterminer la position $r(t)$ et la vitesse $v(t)$ d'une particule à l'instant t connaissant son état initial $r(0)$, $v(0)$, Lagrange se pose la question sous l'angle suivant : quelle est la trajectoire effectivement suivie par la particule si, partant de $r(i)$, à l'instant $t(i)$, elle arrive en $r(i+1)$ à $t(i+1)$? [12]. Les phénomènes dans la mécanique de Newton sont décrits en terme de « forces » agissant sur des « systèmes », alors que dans la mécanique décrite par Lagrange la dynamique repose sur le principe de moindre action. Ce principe repose sur le fait qu'il existe une grandeur nommée action qui permet de déduire la dynamique d'une quantité physique comme la position, la vitesse ou l'accélération d'une particule. Maupertuis en 1744 explique que «lorsqu'il arrive quelque changement dans la Nature, la quantité d'action nécessaire pour ce changement est la plus petite qu'il soit possible». L'Hamiltonien (H) d'un système correspond, en règle générale, à l'énergie du système. Ainsi le

principe de moindre action minimise l'énergie (H) employée lors d'un changement d'état. Notons que la mécanique de Lagrange et de Hamilton sont équivalentes à la mécanique classique dite newtonienne.

Parmi les modèles qui ont inspiré la construction du CPM, on retrouve le modèle de Potts et le modèle d'Ising, des modèles de la physique statistique. La physique statistique permet d'expliquer le comportement et la dynamique d'un système physique constitué d'un grand nombre de particules, à partir des propriétés de ses particules. Les entités peuvent être aussi bien des atomes que des ions, ou d'autres éléments. La physique statistique a été utilisée pour expliquer, par exemple, le phénomène de polarité ou de magnétisme. Comme il n'est pas possible de suivre l'évolution de toute les particules du fait de leur nombre, par exemple le nombre de particules dans 22,4 litres de gaz est de $6 * 10^{23}$ (les particules sont ici des molécules), la physique statistique détermine l'évolution du système via des méthodes statistiques.

La physique statique rejoint la notion de la théorie de l'information évoquée dans le chapitre 1. L'énergie d'un état du système en fonction de l'état de chaque particule et la probabilité de passer d'un état à un autre doivent être définies . Comme pour le principe de moindre action, dans la physique statistique ($cf.$ la notion d'entropie mentionnée dans le chapitre 1) la dynamique du système tend vers des états de plus basse énergie. Une approche pour calculer l'évolution d'un système dans la physique statique est la méthode de Monté Carlo. La méthode Monté Carlo désigne la classe des algorithmes qui permettent de calculer le résultat d'un échantillon en se basant sur un processus répétitif et aléatoire. Nous décrivons en particulier l'algorithme de Métropolis. Cet algorithme est utilisé pour construire la dynamique dans le CPM. Cet algorithme est composé de 4 étapes :

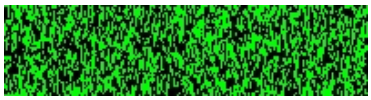
1. Depuis une configuration Sa du système et connaissant son énergie Ea, une nouvelle configuration Sb proche de Sa est obtenue.
2. Eb l'énergie de Sb est calculée.
3. Si $Eb \leq Ea$ alors la transition $Sa \rightarrow Sb$ est acceptée, on tend vers un état de plus basse énergie. Retournez à l'étape 1 avec comme état initial Sb.
4. Sinon si la probabilité $e^{-(Eb-Ea)/kT}$ est acceptée, alors la transition $Sa \rightarrow Sb$ est acceptée. k est la constante de Boltzmann et T la température. On retourne donc à l'étape 1 avec comme état initial Sb. Sinon on reste dans l'état Sa.

Cet algorithme permet donc de calculer des états de plus basses énergies. Si l'état successeur a une énergie inférieure, il est accepté, sinon si l'énergie est supérieure, l'état est accepté selon une probabilité. Cette probabilité dépend de la température et de la constante de Bolzmann. Nous ne discutons pas de ces paramètres car leur sens physique dans le CPM reste mal défini. Par contre nous discutons les effets de leur valeur dans nos applications. Nous pouvons remarquer que dans ces modèles la notion de temps n'est pas prise en compte. Par conséquent, le temps que prend réellement la transition entre deux états ne peut être déterminé ne manière simple. Dans les modèles que nous présentons dans les sections suivantes, la dynamique correspond à la succession des transitions acceptées. Pour utiliser cet algorithme, une fonction de voisinage doit être définie pour calculer un état proche d'un autre ainsi qu'une fonction d'énergie d'un état. Nous définissons des fonctions de voisinages et des fonctions d'énergie selon les exemples et les applications traités dans les sections suivantes.

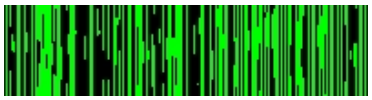
1.1.2 Formalisation du Modèle d'Ising

Le modèle d'Ising développé par Ernst Ising en 1925 est un système d'interaction de spins. Le spin est une propriété quantique intrinsèque associée à chaque particule. Il correspond à une caractéristique de la nature de la particule, au même titre que sa masse et sa charge électrique. Il permet de caractériser le comportement de la particule sous l'effet de la symétrie de rotation dans l'espace.

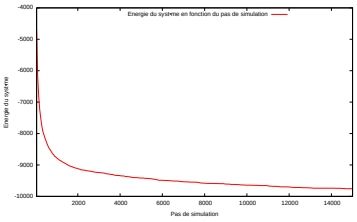
Un objet possède un spin q s'il est invariant sous une rotation d'angle $2\pi/q$. Par exemple un carré a un spin 4. Le modèle d'Ising considère un spin à deux états, deux spins peuvent être parallèles ou antiparallèles.



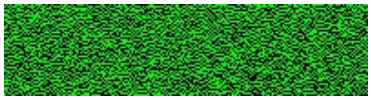
(a) pas de simulation = 50 avec $J(\Theta_{ij}) = -cos(\Theta_{ij})$



(b) pas de simulation = 15000 avec $J(\Theta_{ij}) = -cos(\Theta_{ij})$

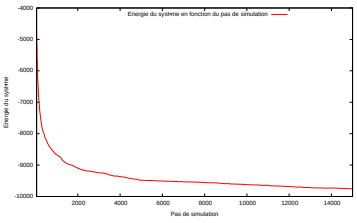


(c) Énergie en fonction du temps avec $J(\Theta_{ij}) = -cos(\Theta_{ij})$



(d) pas de simulation = 50 avec $J(\Theta_{ij}) = cos(\Theta_{ij})$

(e) pas de simulation = 15000 avec $J(\Theta_{ij}) = cos(\Theta_{ij})$



(f) Énergie en fonction du temps avec $J(\Theta_{ij}) = cos(\Theta_{ij})$

FIGURE 1.1 – Simulation du modèle d'Ising. Les figures 1.1(a), 1.1(b) et 1.1(c) montrent la simulation du modèle d'Ising avec $J(\Theta_{ij}) = -cos(\Theta_{ij})$ et les figures 1.1(d), 1.1(e) et 1.1(f) montrent la simulation du modèle d'Ising avec $J(\Theta_{ij}) = cos(\Theta_{ij})$. La couleur verte représente les spins avec l'état 0 et la couleur noire les spins avec l'état π. Les deux simulations partent d'un état aléatoire et un pas de simulation implique le changement d'un seul spin pour chaque colonne de l'environnement. Une colonne pouvant être considérée comme un problème de spin indépendant des autres colonnes car seulement le voisin du haut est pris en compte. Les figures 1.1(a) et 1.1(b) mettent en évidence l'alignement de spins. Les figures 1.1(d) et 1.1(e) montrent que les spins voisins ont des états différents. Les figures 1.1(c) et 1.1(f) montrent l'évolution de l'énergie en fonction du temps. L'énergie décroit en fonction du temps et converge vers une valeur. Ici $T = 0$ ce qui implique qu'à chaque pas de simulation l'énergie reste inchangée ou décroit. La vidéo de la simulation avec $J(\Theta_{ij}) = -cos(\Theta_{ij})$ peut être vue à l'adresse `http://www.youtube.com/watch?v=cfON9DkH2cg` et celle avec $J(\Theta_{ij}) = cos(\Theta_{ij})$ à l'adresse `http://www.youtube.com/watch?v=XQfk5noBUAg&feature=related`.

Dans ce cas, on peut donc caractériser chaque spin par une équation de la forme $\theta_n = 2\pi n/2$.
Considérons un système de spin confiné dans un plan, l'Hamiltonien de ce système est alors :

$$E = -\sum_{i,j} J(\Theta_{ij}) \tag{1.1}$$

où $\Theta_{ij} = \theta_{n_i} - \theta_{n_j}$ est l'angle entre les deux spins au voisinage de i et j, et $J(\Theta_{ij})$ est une fonction 2π périodique (ici $\Theta_{ij} \in \{0; \pi\}$, 0 si les 2 spins sont parallèles π si ils sont antiparallèles).
Comme la dynamique repose sur le principe de moindre action, la transition entre deux états est celle qui coûte le moins d'énergie. Les transitions privilégient de préférence $\Delta H \leq 0$ (on tend vers des états de plus basse énergie).

Ce modèle est utilisé pour modéliser de nombreux phénomènes notamment le magnétisme.
Exemple : si $J(\Theta_{ij}) = cos(\Theta_{ij})$ les spins auront tendance à s'aligner dans des directions opposées, si $J(\Theta_{ij}) = -cos(\Theta_{ij})$ les spins auront tendance à s'aligner dans les mêmes directions. En général, on considère le modèle d'Ising avec interaction entre premiers voisins seulement.

Pour calculer la dynamique du modèle d'Ising, l'algorithme de Métropolis peut être utilisé. Pour trouver un état voisin, nous changeons un spin par sa valeur opposée. Dans ce cas nous pouvons formaliser

le Modèle d'Ising par l'automate cellulaire (d, Q, V, F_l) où :

- $d = 2$ (dans le cas d'un espace à 2 dimensions)
- $Q = \{0, -\pi\}$ (deux états)
- $V(< x_1, x_2 >= \{< x_1 + 1, x_2 >, < x_1, x_2 + 1 >\}$
  (nous prenons en compte seulement 1 voisin)
- $F_l(< s_x^t, n = \{s_{x_1,x_2+1}\} >)) =$
  $< 0, n >$ si $s_x^t = \pi \wedge J(\pi - s_{x_1,x_2+1}) \geq J(0 - s_{x_1,x_2+1})$
  $< \pi, n >$ si $s_x^t = 0 \wedge J(0 - s_{x_1,x_2+1}) \geq J(\pi - s_{x_1,x_2+1})$
  $< 0, n >$ si $s_x^t = \pi \wedge \Delta E = J(0 - s_{x_1,x_2+1}) - J(\pi - s_{x_1,x_2+1}) \wedge \Delta E > 0 \wedge alea() < e^{\Delta E/kT}$
  $< \pi, n >$ si $s_x^t = 0 \wedge \Delta E = J(\pi - s_{x_1,x_2+1}) - J(0 - s_{x_1,x_2+1}) \wedge \Delta E > 0 \wedge alea() < e^{\Delta E/kT}$
  où $alea()$ renvoie un nombre aléatoire dans $[0, 1[$

De façon classique la dynamique d'un automate cellulaire est synchrone, c'est à dire que tous les sites de l'environnement sont modifiés en même temps par la fonction F_l. Pour respecter l'algorithme de Métropolis, à chaque pas de transition, seulement un site est choisi aléatoirement pour être modifié. La figure 1.1 montre deux simulations du modèle d'Ising, l'une où $J(\Theta_{ij}) = -cos(\Theta_{ij})$ (les spins s'alignent), l'autre $J(\Theta_{ij}) = cos(\Theta_{ij})$ (les spins s'opposent). Ces simulations montrent comment le système s'auto-organise. L'organisation ici peut être mesurée par l'énergie d'un état, plus l'énergie est faible plus la probabilité que chaque site ait un état quelconque est faible. Si tous les sites s pouvaient prendre un état quelconque, alors il y aurait autant de chance d'avoir un site voisin à s différent ou identique à s Ainsi l'énergie de l'état du système serait égale à 0. Plus on s'éloigne de 0, plus le système baisse en entropie et plus le système s'auto-organise. Ainsi l'auto-organisation dans le modèle d'Ising peut s'étudier grâce à son entropie.

1.1.3 Formalisation du Modèle de Potts

Le modèle de Potts est une généralisation du modèle d'Ising, il considère des particules à q états ($q \geq 2$). Il a été développé par Potts en 1952. La formalisation de ce modèle est rappelée dans [87]. Les spins de ce système sont de la forme $\theta_n = 2\pi n/q$ où $q \geq 2$ et $n \in [0, q-1]$. L'énergie correspondante à ce système est :

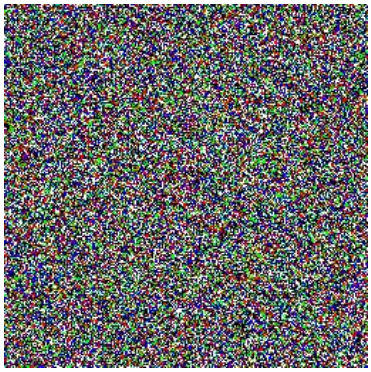
$$E = -\sum_{i,j} J(\Theta_{ij}) \tag{1.2}$$

Un modèle de Potts standard suggéré par Potts lui-même est d'utiliser $J(\Theta_{ij}) = \varepsilon\delta(s_i, s_j)$ où $\delta(s_i, s_j) = 1$ si $s_i \neq s_j$ sinon 0 (s_i le spin s sur le site i). Ce modèle est considéré comme standard et plusieurs noms le désignent : planar Potts model, standard Potts model ou simplement Potts Model.

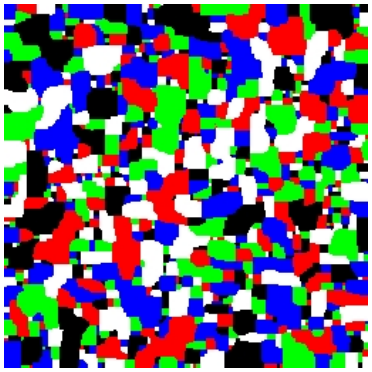
Pour trouver un état voisin, nous suggérons de changer un spin par la valeur de un de ces voisins. Dans ce cas nous pouvons formaliser le modèle de Potts par l'automate cellulaire (d, Q, V, F_l) où :

- $d = 2$ (dans un espace à 2 dimensions)
- $Q = \{\theta\}$ où $\theta_n = 2\pi n/q$ avec $q \geq 2$ et $n \in [0, q-1]$
- $V(< x_1, x_2 >= \{< x_1 + 1, x_2 >, < x_1 - 1, x_2 >, < x_1, x_2 + 1 >, < x_1, x_2 - 1 >\}$
  dans le cas où seulement les 4 voisins directs sont pris en compte :
- $F_l(< s_x^t, n = \{< x_1 + 1, x_2 >, < x_1 - 1, x_2 >, < x_1, x_2 + 1 >, < x_1, x_2 - 1 >\} >) =$
  $< s', n >$ si $s' = alea(n) \wedge J(s' - s_{x_1,x_2+1}) \geq J(s_x^t - s_{x_1,x_2+1})$
  $< s_x^t, n >$ sinon si $s' = alea(n) \wedge \Delta E = J(s' - s_{x_1,x_2+1}) - J(s_x^t - s_{x_1,x_2+1})$
  $\wedge \Delta E < 0 \wedge alea() < e^{\Delta E/kT}$
  $< s_x^t, n >$ sinon
  où $alea(n)$ renvoie un élément au hasard de n

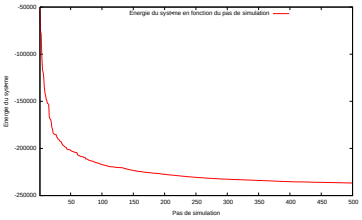
La dynamique de ce système est donnée par la figure 1.2. Comme pour le modèle d'Ising, l'organisation



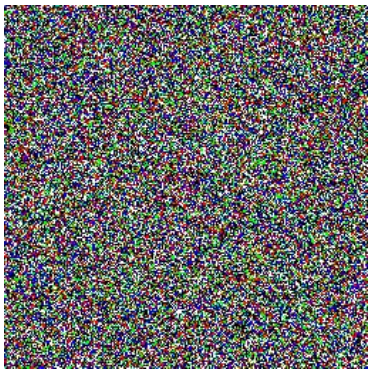
(a) pas de simulation = 1 avec $J(\Theta_{ij}) = -1 * \delta(s_i, s_j)$



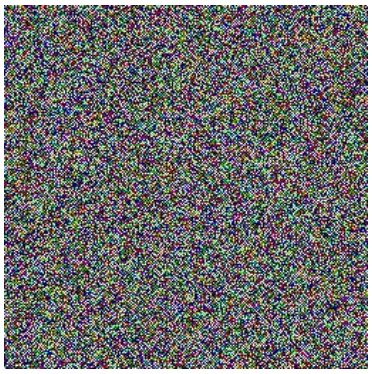
(b) pas de simulation = 500 avec $J(\Theta_{ij}) = -1 * \delta(s_i, s_j)$



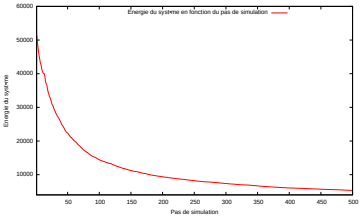
(c) Énergie en fonction du temps avec $J(\Theta_{ij}) = -1 * \delta(s_i, s_j)$



(d) pas de simulation = 1 avec $J(\Theta_{ij}) = 1 * \delta(s_i, s_j)$



(e) pas de simulation = 500 avec $J(\Theta_{ij}) = 1 * \delta(s_i, s_j)$



(f) Énergie en fonction du temps avec $J(\Theta_{ij}) = 1 * \delta(s_i, s_j)$

FIGURE 1.2 – Simulation du modèle de Potts. Les figures 1.2(a), 1.2(b) et 1.2(c) montrent la simulation du modèle de Potts avec $J(\Theta_{ij}) = -1 * \delta(s_i, s_j)$ et les figures 1.2(d), 1.2(e) et 1.2(f) montrent la simulation du modèle de Potts avec $J(\Theta_{ij}) = 1 * \delta(s_i, s_j)$. 5 couleurs (noire, rouge, verte, bleue, blanche) sont définies. Elles correspondent chacune à l'état d'un spin. Les deux simulations partent d'un état aléatoire et un pas de simulation implique que tous les spins ont pu changer d'état. Les figures 1.2(a) et 1.2(b) mettent en évidence l'alignement des spins, signifiant que les spins voisins ont tendance à être dans le même état. Les figures 1.2(d) et 1.2(e) mettent en évidence le fait que les spins voisins tendent à être dans un état différent. Les figures 1.2(c) et 1.2(f) décrivent l'évolution de l'énergie en fonction du nombre de pas de simulation. L'énergie décroit en fonction du temps et converge vers une valeur. Ici $T = 0$ nous considérons donc que si le système tend vers des énergies plus fortes, la transition n'est pas acceptée. La vidéo de la simulation avec $J(\Theta_{ij}) = -1 * \delta(s_i, s_j)$ peut être vue à l'adresse `http://www.youtube.com/watch?v=Q7mdZ2HV3xg` et celle avec $J(\Theta_{ij}) = 1 * \delta(s_i, s_j)$ à l'adresse `http://www.youtube.com/watch?v=qUESlzyQRp8`

dans le modèle de Potts trouve une correspondance directe avec l'entropie. Plus le système s'organise, plus l'entropie baisse. Dans le cas où $J(\Theta_{ij}) = -1 * \delta(s_i, s_j)$ les spins voisins d'un spin auront tendance à être identique, à l'inverse dans le cas où $J(\Theta_{ij}) = 1 * \delta(s_i, s_j)$ les spins voisins auront tendance à se différencier. Dans les deux cas, l'état d'un spin dépend de ses voisins et on a bien une baisse de l'entropie.

Le modèle d'Ising et le modèle de Potts se caractérisent fortement par l'émergence d'une auto-organisation depuis les interactions locales. Nous montrons dans la section suivante comment ces modèles ont inspiré le Modèle de Potts Cellulaire afin de modéliser et simuler l'auto-organisation entre les cellules.

1.2 Formalisation du Modèle de Potts Cellulaire

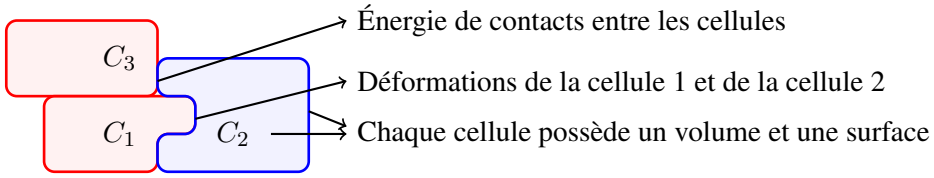


FIGURE 1.3 – Exemple de 3 cellules (C_1, C_2, C_3) destinées à être modélisées. Le modèle de calcul choisi doit permettre de prendre en compte les interactions entre les cellules que ce soit au niveau des énergies de contacts entre les cellules qu'au niveau des déformations qu'elles exercent les unes sur les autres.

Un des objectifs de cette thèse est de construire un agent cellule capable de modéliser et de simuler la morphogenèse. La figure 1.3 montre le type de modélisation des cellules ayant guidé notre réflexion. Une cellule est définie par un volume et une surface qui peuvent se déformer. Les cellules dont les membranes se touchent et peuvent interagir via des énergies de contacts qui favorisent ou non cette adhésion. Nous supposons que ces comportements sont minimums mais nécessaires à la viabilité des cellules. L'auto-organisation que nous souhaitons étudier émerge des interactions entre les cellules qui permettent aux cellules de se déplacer et de changer leur structure (forme) sur une courte durée afin de tendre vers un état plus stable.

Pour cela nous nous sommes intéressés au Modèle de Potts Cellulaire (CPM) car il repose sur un formalise bien fondé, et peut se décrire simplement tout tant simulant des dynamiques complexes comme le tri cellulaire. Comme il a été décrit précédemment, le CPM est un automate développé par Glazier et Graner [36] en 1992 dans le but de modéliser un certain nombre de phénomènes qui se produisent au cours de la morphogenèse [18, 57]. Cet automate est une extension du modèle de Potts développé par Potts en 1952, qui généralise le modèle d'Ising, comme décrit dans [87]. Le modèle d'Ising et de Potts sont des modèles de la physique statistique, autrement dit, ces modèles sont utilisés pour décrire des mécanismes où les interactions locales jouent un rôle important. La dynamique de ces modèles est basée sur la minimisation de l'énergie du système.

La figure 1.2 montre une simulation du modèle de Potts avec 5 états. En fonction des interactions entre les spins, les spins d'un même état peuvent être groupés. Si une cellule est modélisée par l'ensemble des spins d'un même état, aucunes contraintes de volume et de surface ne permettent à la cellule d'avoir une forme viable dans le modèle de Potts. Le CPM résout ce problème en rajoutant dans le modèle de Potts des énergies de volume et de surface. La justification biologique de ces énergies est donnée dans [2].

Dans le cas discret, le CPM est constitué d'une grille où un ensemble d'entités remplit tous les sites de la grille. Les entités du système sont des cellules caractérisées par un volume, une surface et un type. Ils sont en interaction soit de manière directe (via des énergies de contact entre les cellules), soit de manière indirecte (le fait qu'une seule cellule peut être contenue dans un site de la grille et établi donc une concurrence entre les cellules). La figure 1.4 illustre une transition dans le CPM. La suite de cette section présente les notations (1.2.1) utilisées dans ce mémoire et la fonction de transition (1.2.2) afin de formaliser le CPM (1.2.3).

State Sa

$E_v(Sa) = 0 + 1 + 9$

$E_s(Sa) = 0 + 4 + 16$

$E_c(Sa) = 24 + 23 + 13$

$E(Sa) = 90$

$\longrightarrow$

$E(Sa) > E(Sb)$

la transition est acceptée

State Sb

$E_v(Sb) = 0 + 0 + 4$

$E_s(Sb) = 0 + 0 + 4$

$E_c(Sb) = 21 + 19 + 15$

$E(Sb) = 63$

FIGURE 1.4 – Exemple d'une transition dans le CPM. Dans cette figure la grille Sx est de dimension 8x8. A chaque site (i, j) nous associons une valeur $sx_{i,j}$. Nous avons quatre cellules ($\sigma \in [0, 3]$) : 1 cellule pour le milieu extérieur (C_0^m), 2 cellules de types rouges (C_1^r, C_3^r) et une cellule de type bleu (C_2^b). Les cellules de types rouges ont les caractéristiques suivantes : $V_{target} = 7$, $S_{target} = 12$, et les cellules de types bleus les suivantes : $V_{target} = 10$, $S_{target} = 11$. Dans l'état Sa la cellule C_1^r (resp. C_2^b, C_3^r) a un volume de $V1 = 7$ (resp. $V2 = 11$, $V3 = 4$) et une surface de $S1 = 12$ (resp. $S2 = 13$, $S3 = 8$). Dans l'état Sb la cellule C_1^r (resp. C_2^b, C_3^r) a un volume $V1 = 7$ (resp. $V2 = 10$, $V3 = 5$) et une surface $S1 = 12$ (resp. $S2 = 11$, $S3 = 10$). La cellule pour le milieu extérieur n'a pas de contrainte de surface et de volume. La matrice (symétrique) d'énergie de contact (donnée) est définie par : $T_{0,1} = T_{0,3} = 2$, $T_{0,2} = 1$, $T_{1,2} = T_{2,3} = 3$, $T_{1,3} = 0$. Comme la cellule C_1^r et la cellule C_3^r sont de même type $T_{1,*} = T_{*,3}$. $E_v(S)$ (resp. $E_s(S)$, $E_c(S)$) correspond à l'énergie de volume (resp. de surface, de contact) de toutes les cellules dans l'état S.

1.2.1 Notation.

Une grille est notée par Sx et un site de la grille est noté par (i, j). La valeur d'un site (i, j) est notée par $sx_{i,j}$.
Une cellule est notée C_σ^t avec $\sigma \in [1, N]$ où N est le nombre de cellules et t le type de cellule. L'identifiant 0 est utilisé pour représenter pour le milieu extérieur, qui est une cellule sans contrainte de volume et de surface.
Une cellule C_σ^t a une contrainte de volume (resp. de surface) égale à la différence au carré entre le volume cible $V\sigma_t$ et le volume courant $V\sigma$ ($resp.$ surface cible $S\sigma_t$ et la surface courante $S\sigma$).
Les énergies de contact sont contenues dans une matrice T telle que $T_{\sigma,\sigma'}$ ($resp.$ $T_{t,t'}$) est l'énergie de contact entre les cellules C_σ^t et les cellules $C_{\sigma'}^{t'}$ ($resp.$ entre les cellules de type t et t').

Le CPM est composé d'une grille [3] Sx de D dimensions (ici $D = 2$). Chaque site (i, j) est rempli par une partie de la cellule C_σ^t, c'est à dire que la valeur $sx_{i,j}$ du site (i, j) dans l'état Sx est égale à σ. Donc une cellule C_σ^t est égale à $\{(i, j) \in Sx | sx_{i,j} = \sigma\}$ l'ensemble des sites dont la valeur est σ.
Finalement l'état du système est une grille Sx où chaque $sx_{i,j}$ est égale à un entier $\sigma \in [0, N]$.

3. Ici l'environnement est discret mais le cas continu est défini aussi [35].

1.2.2 Fonction de transition

Soit $F_{tr}(Sa, k, t) = Sb$ la fonction de transition du CPM entre l'état Sa et Sb en fonction de k et t deux constantes. Soit Sc l'état Sa où la valeur d'un seul site a été remplacée par la valeur d'un des sites voisins. Si la probabilité de transition P_{tr} entre les états Sa et Sc est acceptée alors $Sb = Sc$ sinon $Sb = Sa$.

$$\begin{aligned} &F_{tr}(Sa, k, t, p) = Sb \Leftrightarrow \exists (i', j') \in neighbour(i, j)(\\ &\quad (sc_{i,j} = sa_{i',j'}) \wedge \\ &\quad (Sc - sc_{i,j} = Sa - sa_{i,j}) \wedge (p = alea() \wedge \\ &\quad (p \leq P_{tr}(Sa, Sc, k, t) \Rightarrow Sb = Sc) \wedge \\ &\quad (p > P_{tr}(Sa, Sc, k, t) \Rightarrow Sb = Sa)) \end{aligned} \tag{1.3}$$

où $alea()$ retourne un élément au hasard de l'ensemble $[0, 1[$, $neighbour(i, j)$ est l'ensemble des sites voisins du site (i, j) et P_{tr} la probabilité de transition.
Nous pouvons observer que seulement un site de la grille peut changer, et plusieurs sites peuvent être candidats pour ce changement. La dynamique est donc asynchrone et non-deterministe.

Probabilité de transition La probabilité de transition utilisée est la probabilité de Monte Carlo suivant une température t. Soit $P_{tr}(Sa, Sb, k, t) = p$ la probabilité de transition entre les états Sa et Sb suivant k et t.

$$\begin{aligned} &P_{tr}(Sa, Sb, k, t) = p \Leftrightarrow \\ &\quad t > 0 \wedge (E(Sb) - E(Sa)) \leq 0 \Rightarrow p = 1 \\ &\quad t > 0 \wedge (E(Sb) - E(Sa)) > 0 \Rightarrow \\ &\quad p = exp((E(Sb) - E(Sa))/kt) \\ &\quad t = 0 \wedge (E(Sb) - E(Sa)) < 0 \Rightarrow p = 1 \\ &\quad t = 0 \wedge (E(Sb) - E(Sa)) = 0 \Rightarrow p = 0.5 \\ &\quad t = 0 \wedge (E(Sb) - E(Sa)) > 0 \Rightarrow p = 0 \end{aligned} \tag{1.4}$$

où $E(S)$ est la fonction d'énergie.
Cette probabilité favorise les transitions qui amènent aux états de plus basse énergie.

Fonction d'énergie Soit $E(S) = e$ la fonction d'énergie de l'état S. Cette fonction caractérise l'état du système. Dans le CPM, une première définition dépend du volume et de la surface de chaque cellule et des énergies de contact entre elles. $E(S)$ peut être définie comme :

$$E(S) = E_c(S) + E_v(S) + E_s(S) \tag{1.5}$$

où :

$$E_c(S) = \sum_{(i,j) \in S} \sum_{(i',j') \in neighbours(i,j)} T_{s_{i,j}, s_{i',j'}} * (1 - \delta_{s_{i,j}, s_{i',j'}}) \tag{1.6}$$

où $T_{s,s'}$ est une matrice d'énergie de contact entre les cellules C_s^t et les cellules $C_{s'}^{t'}$. Si $s = s'$ alors $\delta_{s,s'} = 1$ sinon 0.

$$E_v(S) = \sum_{\sigma \in [1,N]} (V\sigma - V\sigma_t)^2 \tag{1.7}$$

$$E_s(S) = \sum_{\sigma \in [1,N]} (S\sigma - S\sigma_t)^2 \tag{1.8}$$

où $V\sigma_t$ et $S\sigma_t$ sont le volume cible et la surface cible de la cellule C_σ^t, c'est à dire le volume et la surface vers lesquels la cellule tend.

Fonction de voisinage Cette fonction V dépend de l'application. Pour nos applications les voisins d'un site (i, j) sont les sites $(i+1, j), (i, j+1), (i-1, j), (i, j-1)$ (s'ils sont dans la grille) dans le cas à 2 dimensions, et les 6 voisins directs dans le cas à 3 dimensions.

1.2.3 Formalisation comme un Automate Cellulaire

Suivant les fonctions définies dans la section précédente et l'algorithme Métropolis nous pouvons formaliser le CPM par l'automate cellulaire (d, Q, V, F_l) où :

- $d = 2 \vee 3$ (dans un espace à 2 dimensions ou à 3 dimensions)
- $Q = [0, N]$ (où N est le nombre de cellules)
- $V(< x_1, x_2 >= \{< x_1+1, x_2 >, < x_1-1, x_2 >, < x_1, x_2+1 >, < x_1, x_2-1 >\})$
  (les voisins d'un site sont les quatre voisins directs si d=2)
- $F_l(< s_x^t, n = \{< x_1+1, x_2 >, < x_1-1, x_2 >, < x_1, x_2+1 >, < x_1, x_2-1 >\} >) =$
  $< s', n >$ si $s' \in n \wedge alea() \leq P_{tr}(\{s\} \cup n, \{s'\} \cup n, k, t)$
  $< s_x^t, n >$ sinon

Pour assurer une dynamique asynchrone et non-déterministe, la fonction de transition locale est appelée une seule fois sur un site choisi au hasard. Nous pouvons remarquer que pour calculer la différence d'énergie entre deux états, il est possible de calculer seulement la différence d'énergie entre les états voisins. Dans la section suivante nous montrons les évolutions du CPM déjà réalisées. Puis nous proposons une nouvelle mise en œuvre de type multi-agents permettant d'améliorer la modularité du code développé ainsi que la vitesse de calcul. Ceci est important pour la simulation de système multi-cellulaires où la richesse des comportements doit pouvoir être facilement accueillie et où le nombre d'entités en interaction peut s'avérer très élevé ($> 10^4$).

1.3 Différentes évolutions possibles

Nous discutons dans cette section des évolutions déjà réalisées du CPM et nous proposons les évolutions à apporter pour modéliser des comportements plus réalistes.

1.3.1 Évolution du Modèle de Potts Cellulaire déjà existant

Le Modèle de Potts Cellulaire définit de façon simple et réaliste la cellule en terme de volume et de surface, cependant plusieurs extensions peuvent être proposées afin de modéliser des phénomènes plus complexes. Nous pouvons citer les travaux de James A. Glazier qui poursuit et étend le CPM, et les travaux de Stan Marée. Les travaux de Glazier ont permis la mise en œuvre du logiciel CompuCell3D [41, 18]. CompuCell3D permet de modéliser et simuler de nombreux phénomènes dont les suivants :

- le tri cellulaire [36] et l'adhésion cellulaire notamment dans la gastrulation chez le poulet [83].
- la formation de tissu complexe comme la croissance d'une tumeur [75]
- l'angiogenèse (croissance des tissus sanguins) [61]
- le développement des membres notamment dans le cas des vertèbres [37].

Pour pouvoir modéliser ces différents phénomènes, les comportements décrit dans la figure 1.5 ont été mis en œuvre dans CompuCell3D. Nous pouvons noter que certains d'entre eux sont nécessaires pour modéliser la morphogenèse cellulaire comme : la migration vers un ou contre un gradient de molécules, la division cellulaire, une horloge interne à la cellule pour déclencher sa mort. La morphogenèse au niveau cellulaire implique que les cellules meurent et se divisent pour former les tissus. De plus, elle est guidée par des morphogènes, $i.e.$ une protéine dont le but est de fournir un gradient de concentration. Ceci peut permettre de différentier les cellules. Les cellules peuvent ainsi dépendre des molécules secrétées et

Table 1. Selected CompuCell3D plug-ins.

Name	Function
AdvectionDiffusionSolver	Solves an advection-diffusion equation on a cell field.
FlexibleDiffusionSolver	A customizable solver of diffusion equations that also allows secretion, absorption, and diffusion restriction by cell type. Also allows variable space step, time step, diffusion constants, secretion rates, and number of fields.
BoundaryPenalty	Enforces an energy penalty if a cell is close to a boundary, to prevent cells spreading on domain boundaries.
CellBoundaryTracker	Provides locations of cell-boundary voxels.
CellVelocity	Tracks cell speeds.
CenterOfMass	Tracks cell centers of mass.
Chemotaxis	Implements cell chemotaxis to an external chemical field, with forces proportional to chemical gradients.
ExternalPotential	Imposes a directed potential, or force, on cells.
Growth	Implements a cell density-dependent algorithm for domain growth. The lattice maintains its current dimensions until cell density reaches a user-specified threshold, then it grows in positive *z* by a user-specified amount.
LengthConstraint	Implements anisotropic cells.
Mitosis	Implements cell division.
SimpleClock	Implements an internal timer for cells. Provides the ability to start a timer and decrement until it hits zero.
Viscosity	Implements cell viscosity (useful in fluid flow simulations).

FIGURE 1.5 – Les comportements ajoutés au CPM dans le logiciel CompuCell3D. Figure extraite de [18]

diffusées par les autres cellules. A noter que dans compuCell3D, la diffusion des molécules est simulée par un solveur d'équations différentielles. Notons que CompuCell3D utilise plusieurs méthodes de résolution numérique : CPM, équations différentielles, division, ...

Les travaux de Stan Marée montrent [57] un modèle réalisé grâce au CPM de la morphogenèse de *Dictyostelium discoideum* (un amibe, être unicellulaire). Ces amibes ont plusieurs voies d'évolution. Une de ces voies est la sécrétion d'un chimioattractant qui dans le cas de *Dictyostelium discoideum* est l'AMPc. L'agrégation des amibes permet la formation d'une forte source d'AMPc qui attire les amibes voisins. Cette agrégation ressemble à une "limace" pouvant atteindre quelques millimètres, et elle est constituée de milliers d'amibes. Le but de cette agglomération pour ces cellules est de trouver des conditions plus favorables. Dans ce modèle, le CPM a été couplé avec la modélisation d'une voie de signalisation appelé AMPc. La différenciation cellulaire, un modèle de diffusion, la migration cellulaire ont dû en être mis œuvre et ajoutés au CPM. La section suivante décrit quelques comportements qui ne sont pas encore ajoutés au CPM mais qui nous semblent essentiels pour modéliser certains phénomènes cellulaires.

1.3.2 Autres évolutions

Même si de nombreuses extensions ont été déjà proposées au CPM, d'autres extensions doivent être ajoutées. Dans le phénomène de la morphogenèse cellulaire, l'apoptose, la mort des cellules est indispensable [63]. L'apoptose permet par exemple de donner une forme au tissu, comme l'élimination du tissu entre les doigts (*cf.* figure 1.6). L'apoptose peut être déclenchée aussi bien par des hormones, que par des stress ou par des dommages subis par l'ADN. Dans ce mémoire nous proposons de définir l'apoptose comme un bilan énergétique de la cellule. Si la cellule est dans un milieu satisfaisant comme la présence de molécules dont elle à besoin, elle ne meurt pas et peut se diviser, sinon si le bilan devient nul la cellule meurt. Par défaut une cellule consomme son énergie pour sa machinerie interne : comme la synthèse des protéines. Cet notion de bilan énergétique peut être retrouvée dans la "Dynamic Energy

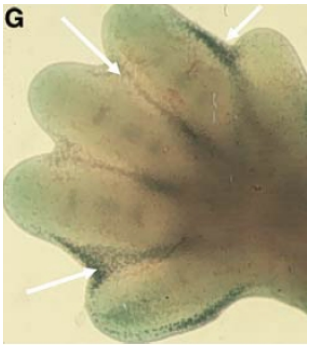


FIGURE 1.6 – Rôle de l'apoptose dans la formation de la main. Figure extraite de [63]. Les cellules colorées en bleu foncé indique les cellules mortes. Elles se trouvent dans l'espace inter-digital. Image d'une pré-main d'un embryon de souris de 13,5 jours. Les flèches indiquent les zones de mort cellulaire les plus soutenues.

Budget theory" (DEB) [44]. La théorie DEB essaye d'unifier les points communs entre les organismes au niveau de l'énergie. Cette théorie présente des règles simples qui décrivent l'absorption et l'utilisation de l'énergie et de nutriments (substrats, de la nourriture, la lumière) et les conséquences pour l'organisation physiologique tout au long du cycle de vie d'un organisme. Dans notre modèle le bilan énergétique peut être vue comme l'absorption de nutriment. Si la cellule ne trouve pas assez de nutriment elle meurt, sinon elle peut se reproduire. Ceci va permettre de tester des théories biologiques comme le darwinisme au niveau cellulaire.

Dans le CPM la forme de la cellule est juste définie en terme de surface et de volume. Nous présentons dans ce mémoire la définition d'une énergie permettant à la cellule de cibler une forme dynamique et générique. D'autres travaux ont permis de donner une forme plus précise à la cellule définie dans le CPM comme [60] pour cibler l'élongation de la cellule mais aucune pour une forme générique. Pourtant la forme de la cellule est une caractéristique importante dans de nombreux phénomènes : la forme sphérique des globules rouges est tout à fait adaptée à leur rôle de transport, ou la forme des cellules musculaires pour la contraction des muscles. La forme des cellules joue aussi un rôle important dans la morphogenèse où la forme du tissu émerge des formes des cellules.

Le CPM étant un automate cellulaire il est difficile d'ajouter des comportements aux cellules puisqu'il n'a qu'une représentation implicite des entités ($cf.$ chapitre 1). Nous proposons dans le chapitre suivant une mise en œuvre Multi-Agents du CPM. La mise en œuvre permet de représenter les entités explicitement dans le système. Chaque cellule est modélisée de manière indépendante les unes des autres et a ses propres comportements. D'un point de vue informatique, chaque cellule a ses propres méthodes et mémoire locale. Le bilan énergétique de la cellule et la forme générique sont implémentés dans ce système dans le chapitre 2. Le chapitre 3 valide ces nouveaux comportements en simulant des phénomènes de morphogenèses.

Chapitre 2

Mise en œuvre du Modèle de Potts Cellulaire suivant l'approche Multi-Agents

Dans le but d'étendre plus facilement le Modèle de Potts Cellulaire (CPM), nous décrivons dans ce chapitre une mise en œuvre du CPM dans le paradigme Multi-Agents. Nous formalisons notre système sous les différents aspects d'un Système Multi-Agents (SMA) (2.1), nous donnons sa mis en œuvre (2.2), puis nous décrivons les extensions du CPM que nous avons ajouté au SMA mis en place (2.3).

2.1 Formalisation du Modèle de Potts Cellulaires suivant l'approche Multi-Agents

Dans cette partie, nous décrivons la mise en œuvre du CPM comme SMA. Nous formalisons cette description selon les parties [22] qui composent un SMA : environnement (2.1.1), agent (2.1.2), interaction (2.1.3), organisation. Dans le SMA défini, aucune organisation n'est imposée. L'organisation est émergente des interactions locales. Nous discutons également de l'ordonnanceur du SMA mis en place (2.1.4).

2.1.1 Environnement

0	0	0	0	0	0
0	0	1	2	2	0
0	1	1	1	2	0
0	0	1	0	0	0
0	0	0	0	0	0

FIGURE 2.1 – Exemple d'une matrice Env de taille 6x5 contenant les agents C_0^m, C_1^v, C_2^b.

L'environnement dans le SMA que nous considérons est défini par une matrice nommée Env. Cette matrice correspond à la grille définie dans le CPM où chaque site de la grille est rempli par un agent C_σ^t où σ est son identifiant et t son type. Aucun site ne peut être vide. La figure 2.1 montre un exemple de matrice Env contenant 3 agents. L'environnement est le support des interactions (directes ou non) qui se produisent entre les agents. L'environnement est toujours initialisé par un agent d'identifiant 0 qui modélise le milieu extérieur. Cela signifie que tous les sites de la grille ont la valeur $s_{i,j} = 0$. La modification de la grille est toujours faite par les agents contenus dans la grille lors de leurs déplacements.

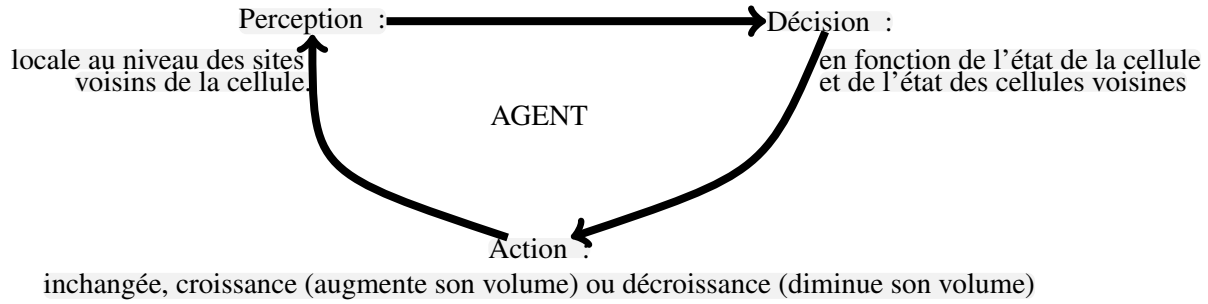


FIGURE 2.2 – Cycle d'exécution d'un agent défini dans le SMA modélisant le CPM.

2.1.2 Agent

Les agents définis dans ce SMA sont les cellules. Les agents peuvent ($cf.$ la figure 2.2), en fonction de leurs perceptions locales de l'environnement et suivant leurs états (décision), modifier l'environnement local c'est à dire la matrice Env ($cf.$ la figure 2.3) lors de leurs déplacements (action). Nous présentons dans cet ordre les différentes caractéristiques de l'agent : sa mémoire, ses comportements, son ordonnanceur.

0	0	0	0	0	0
0	0	1	2	2	0
0	1	1	1	2	0
0	0	1	0	0	0
0	0	0	0	0	0

(a) Matrice Env avant la motilité

0	0	0	0	0	0
0	0	1	2	2	2
0	1	1	1	1	
0	0	1	0	0	0
0	0	0	0	0	0

(b) Matrice Env après la motilité

FIGURE 2.3 – Exemple de motilité des agents. L'agent 2 a supprimé sa présence d'un site et a mis à jour sa présence sur un autre site. L'agent 1 est présent sur un nouveau site.

Mémoire de l'agent Les agents sont les cellules. Un agent C_σ^t est identifié par un entier σ et un type t. Les agents sont caractérisés par un volume, une surface et un type. Tous les agents de type t partagent les mêmes volumes cibles et surfaces cibles ainsi que les énergies de contacts entre les agents. Les volumes et surfaces cibles sont les volumes et surfaces vers lesquels l'agent tend. Les énergies de contacts sont des énergies entre les agents au niveau des sites d'un agent qui sont en contact avec un site voisin contenant une autre agent, $i.e$ au niveau des membranes. Ces énergies de contacts permettent de modéliser l'adhésion entre les cellules.

Chaque cellule C_σ^t «mémorise» :

– ses positions P_σ, $i.e$. l'ensemble des sites de l'environnement où l'agent est présent.

– sa membrane M_σ où :

$$M_\sigma = \{< (i,j), L > | s_{i,j} = \sigma \wedge L = \{(i',j') \in voisins(i,j) | s_{(i',j')} \neq \sigma\} \neq \emptyset\} \quad (2.1)$$

c'est à dire l'ensemble des couples $< (i,j), L >$ où le site (i,j) est sur la membrane et L l'ensemble des sites voisins à l'extérieur de l'agent.

– son volume cible et sa surface cible $(V\sigma_t, S\sigma_t)$, son volume courant et sa surface courante $(V\sigma, S\sigma)$, et ses énergies de volume et de surface (Ev_σ, Es_σ) où :

$$Ev_\sigma = (V\sigma_t - V\sigma)^2 \quad (2.2)$$

$$Es_\sigma = (S\sigma_t - S\sigma)^2 \quad (2.3)$$

– son énergie de contact entre les différents agents Ec_σ où :

$$Ec_\sigma = \sum_{<(i,j),L> \in M_\sigma} \sum_{(i',j') \in L} T_{s_{i,j}, s_{i',j'}} \quad (2.4)$$

Motilité de l'agent. Chaque agent a la capacité de percevoir son environnement local, à savoir l'ensemble de ses voisins. Chaque agent a également la capacité de croître et donc de diminuer le volume d'autres agents (en effet, chaque site de la grille ne peut contenir au minimum et au maximum un agent). Les croissances successives vers une direction et les décroissances successives vers la direction opposée permettent à l'agent de se déplacer vers cette direction. Chaque agent C^t_σ peut donc peut modifier l'environnement local en changeant sa valeur d'un des sites voisins par σ. Soit l'agent C^t_σ qui croît sur un site s où est présent l'agent $C^{t'}_{\sigma'}$. L'agent C^t_σ peut croître sur le site s si la différence d'énergie avant le changement entre l'énergie E_σ de la cellule C^t_σ et l'énergie E'_σ de la cellule $C^{t'}_{\sigma'}$ est supérieure ou égale à la différence après le changement où :

$$E_\sigma = \lambda_c * Ec_\sigma + \lambda_v * Ev_\sigma + \lambda_s * Es_\sigma \quad (2.5)$$

λ_c ($reps.$ λ_v,λ_s) est appelée facteur de contact ($resp.$ de volume, de surface). Si l'énergie est supérieure après à ce qu'elle était avant alors le changement est conditionné par une probabilité de Monte Carlo.

Mise à jour de l'environnement par les agents. Les agents modifient la matrice Env lors de leurs motilités. Comme la dynamique dans le CPM est asynchrone les conflits sont évités. Il n'est pas possible que deux agents modifient la valeur d'un site de la matrice dans le même temps.

L'ordonnanceur des agents. Afin d'assurer leurs mobilités, les agents définis dans ce SMA ont deux comportements : croître et décroître. Comme les agents ont plusieurs comportements il est indispensable qu'il possède un ordonnanceur. Cet ordonnanceur permet de décrire l'ordre d'exécution des agents. L'agent défini dans ce SMA a deux comportements croître ou décroître dans le but de sa motilité. Comme la dynamique du CPM est asynchrone et le choix de la transition est aléatoire, les agents exécutent leurs comportements de croissance et de décroissance de façon asynchrone et chaotique.

2.1.3 Interaction

Les interactions entre les agents sont de deux types ($cf.$ la figure 2.4). Premièrement, des interactions indirectes dues à la concurrence pour occuper l'environnement comme décrit dans [30]. Par exemple, les sites de la matrice Env ne peuvent contenir que un seul agent. Deuxièmement, les cellules sont en interactions directes via les énergies de contacts, les énergies de volume et de surface. Un agent peut croître seulement si le site choisi par cet agent implique un faible coût en terme d'énergie par rapport à l'agent déjà présent sur le site.

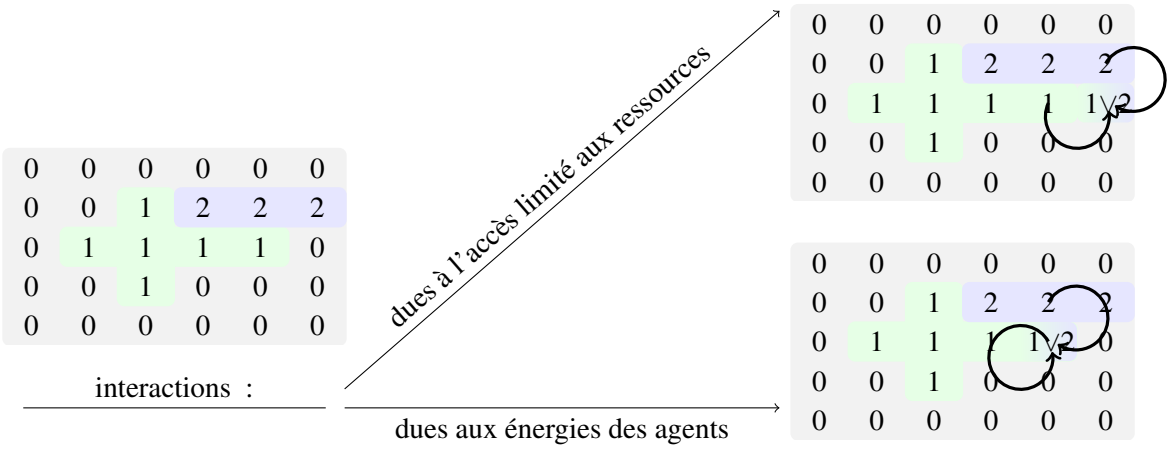


FIGURE 2.4 – Les 2 types d'interaction entre les agents. L'environnement consiste en une matrice Env (à gauche) de taille 5x6. La première interaction est due à l'accès limité aux ressources, dans notre cas l'accès limité à l'environnement. Seulement un agent peut être présent sur un site de la matrice. La seconde interaction est due au fait que un agent (l'agent 2) peut prendre un site à un autre agent (l'agent 1), si son énergie est plus favorable que l'autre agent (agent 1).

2.1.4 L'ordonnanceur du Système Multi-Agents

L'ordonnanceur du SMA indique l'ordre et le mode d'exécution des agents. Comme la dynamique du CPM est asynchrone chaotique, l'ordonnanceur choisi au hasard deux agents voisins. L'un des agents choisis exécute son comportement de croissance l'autre celui de décroissance. Ce choix se fait aussi aléatoirement. Ainsi un pas de simulation selon cet ordonnanceur est le suivant :

1. 2 sites au hasard, (i, j) et un site voisin (i', j'), sont choisis.
2. L'agent $C^t_{s_{i,j}}$ calcule son énergie $E_{s_{i,j}}$. L'agent $C^{t'}_{s_{i',j'}}$ calcule son énergie $E_{s_{i',j'}}$. Soit $E_a = E_{s_{i,j}} + E_{s_{i',j'}}$
3. L'agent $C^t_{s_{i,j}}$ exécute son comportement de croissance sur le site (i', j'). L'agent $C^{t'}_{s_{i',j'}}$ exécute son comportement de décroissance sur le site (i', j').
4. L'agent $C^t_{s_{i,j}}$ calcule son énergie $E_{s_{i,j}}$. L'agent $C^{t'}_{s_{i',j'}}$ calcule son énergie $E_{s_{i',j'}}$. Soit $E_b = E_{s_{i,j}} + E_{s_{i',j'}}$
5. si $(E_b - E_a > 0) \wedge \exp \frac{-(E_b - E_a)}{kT} < alea()$ alors l'agent $C^t_{s_{i,j}}$ exécute son comportement de décroissance sur le site (i', j') et l'agent $C^{t'}_{s_{i',j'}}$ exécuté son comportement de croissance sur le site (i', j'). k et T sont des constantes et $alea()$ renvoie un nombre aléatoire entre $[0, 1[$.

Dans nos applications, les différentes étapes de cet ordonnanceur sont exécutées séquentiellement. Nous pouvons remarquer que cet ordonnanceur n'impose pas que $C^t_{s_{i,j}} \neq C^{t'}_{s_{i',j'}}$. A notre connaissance la littérature ne fait pas référence à ce problème d'optimisation. Ce problème implique qu'un certain nombre de pas de simulations n'amènent pas de changements d'état dans les cas où $s_{i,j} = s_{i',j'}$. Une solution pour résoudre ce problème et de choisir un site au hasard sur la membrane.
Nous apportons la solution suivante. La première étape est remplacée par :

1. Nous choisissons un site au hasard (i, j) depuis $\{(i, j) | < (i, j), L > \in M_\sigma\}$ l'ensemble des sites qui sont sur la membrane des agents et un site au hasard (i', j') depuis L où $< (i, j), L > \in M_\sigma$. La probabilité de choisir un agent est proportionnelle à la taille de sa membrane.

La mise en œuvre de cette solution est décrite dans la section suivante.

2.2 Mise en œuvre du Système Multi-Agents

Nous discutons dans cette section du choix du langage informatique que nous avons retenu (2.2.1), de la structure du programme qui dépend fortement de la description du SMA faite dans la section précédente (2.2.2) et nous décrivons les algorithmes construits dans le but d'optimiser la convergence (2.2.3).

2.2.1 Choix du Langage

Nous voulons que la mise en œuvre du SMA permette à la fois d'exécuter des simulations dans un temps raisonnable, de pourvoir interagir avec le système, et qu'elle soit portable (distribuer le programme sur différentes plate-formes). Il est important aussi que le langage permette un développement rapide. Un langage proche du langage machine permettant une exécution efficace, mais ne permettant pas un prototypage rapide, car ses structures sont éloignées du paradigme multi-agents, ne nous semble pas judicieux. Les langages dont l'approche objet n'est pas définie ne sont donc pas retenus, car ils ne facilitent la conception d'agent.

La réflexivité du langage nous semble un critère important car elle permet à la fois un débogage efficace et une interaction forte avec l'utilisateur. L'interaction dans un outil de simulation permet de faire varier les paramètres, les interactions, en cours de simulation, autrement dit de faire de l'expérience $in - virtuo$ [77]. Le langage Smalltalk vérifie parfaitement ces caractéristiques. Un prototypage rapide du SMA défini dans ce mémoire nous a permis d'étudier ses performances du point de vue du temps de simulation. Les performances étant mauvaises (50x plus long que le la mise œuvre proposée dans cette section) ce langage n'a pas été retenu.

Finalement nous avons opté pour le langage Java qui permet une forte portabilité même si les performances et la réflexivité du langage n'est pas maximum. Pour interagir avec le système, les objets qui ont une interaction avec l'utilisateur, ont une représentation graphique permettant à l'utilisateur de les sélectionner. En utilisant la réflexivité définie en java il est alors possible de lister les attributs de l'objet. Pour chaque attribut nous définissons des accesseurs (méthodes définissant la manière dont l'objet doit être lu et mis à jour) pour pouvoir modifier leurs valeurs en cours de simulation. La figure 2.5 montre l'interface graphique du programme et la représentation graphique de certains objets. Par exemple nous pouvons observer que les objets représentants les agents cellules ont un rendu graphique permettant de voir l'ensemble des sites qu'ils contiennent. En sélectionnant un de ces agents une inspection de leur classe est faite pour lister ses méthodes et attributs. Nous pouvons alors modifier les attributs de l'objet si les accesseurs ont été définis. Les classes définissant ces objets et la structure du programme sont définies dans la section suivante.

2.2.2 Structure du Programme

La structure du programme est définie par le diagramme de classes UML (Unified Modeling Language) donné dans la figure 2.6. Ce diagramme peut être vu comme le méta-modèle de nos modèles de simulation. L'interface Scheduler définie l'ordre dans lesquels sont exécutés les comportements du systèmes, $i.e.$ les comportements des agents mais aussi du rendu graphique. La classe SequentialSheduler réalise cette interface en exécutant les comportements de façon séquentielle. Les comportements définis dans le système héritent tous de la classe Behaviour qui impose une méthode run qui est utilisée par la classe Scheduler pour exécuter les comportements. La classe abstract ImplBehaviour réalise la classe Behaviour. Cette classe est composée des agents et de l'environnement. Les classes Display et CPM hérite de la classe ImplBehaviour car ils définissent des comportements appliqués aux agents représentés dans

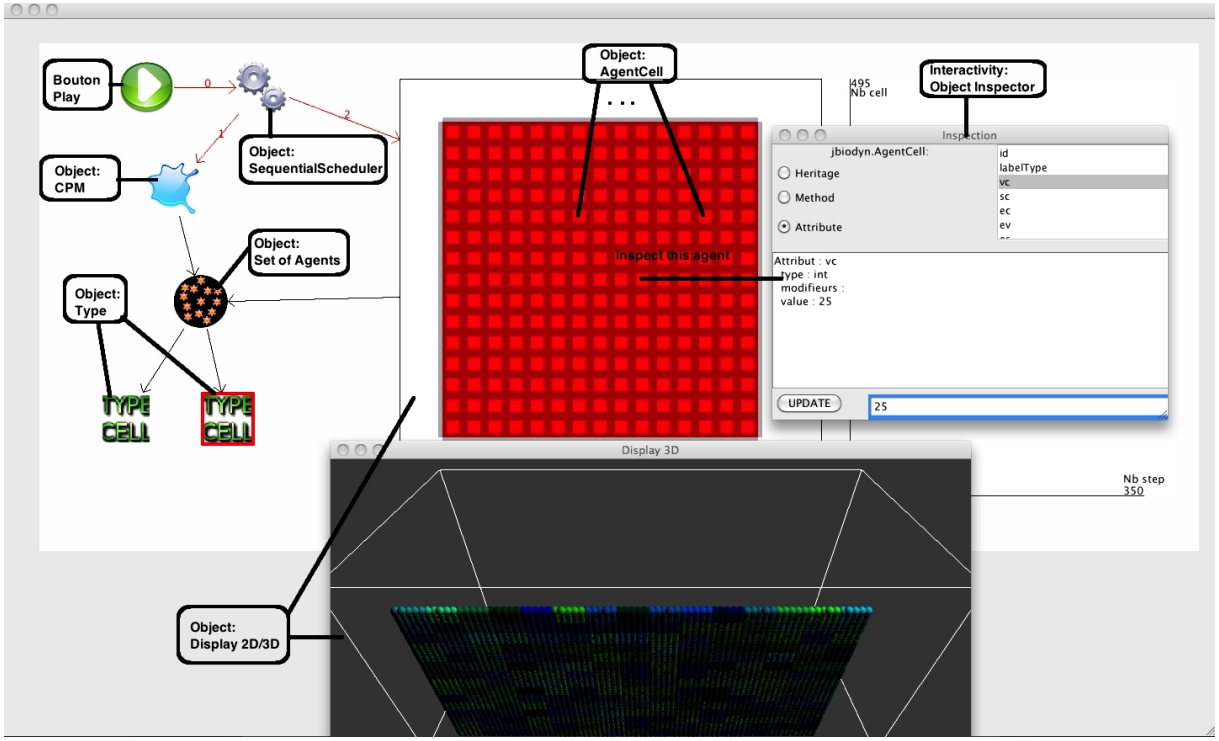


FIGURE 2.5 – Interface graphique du SMA mise en œuvre. Le programme est initialisé avec 225 agents dans un environnement en 2D. Les agents sont représentés en rouge dans la représentation graphique de l'objet Display, et ont un volume de 25 et une surface de 20. En sélectionnant (par clique souris), une inspection est faite de l'objet en utilisant la réflexivité définie en java et les attributs et méthodes peuvent être examinés et les attributs modifiés. Nous pouvons donc vérifier que l'attribut vc de l'agent sélectionné, correspondant au volume, est bien égal à 25. L'interface graphique représente à la fois les interactions entre les différents objets du programme et le rendu. Les objets qui ont une représentation graphique sont : SequentialScheduler, CPM, la liste des agents contenus dans l'objet CPM, Types, Display, AgentCell. Le diagramme de classes de ces objets est donné dans la figure 2.6. Les interactions entre les objets sont représentées par des flèches. Le bouton play permet de démarrer ou de mettre sur pause la simulation. L'environnement peut être en 2D ou 3D, si la 3D est choisie alors le rendu 2D indique une tranche de l'environnement 3D qui est choisie par l'utilisateur. L'inspection des agents ne peut se faire que dans le rendu 2D.

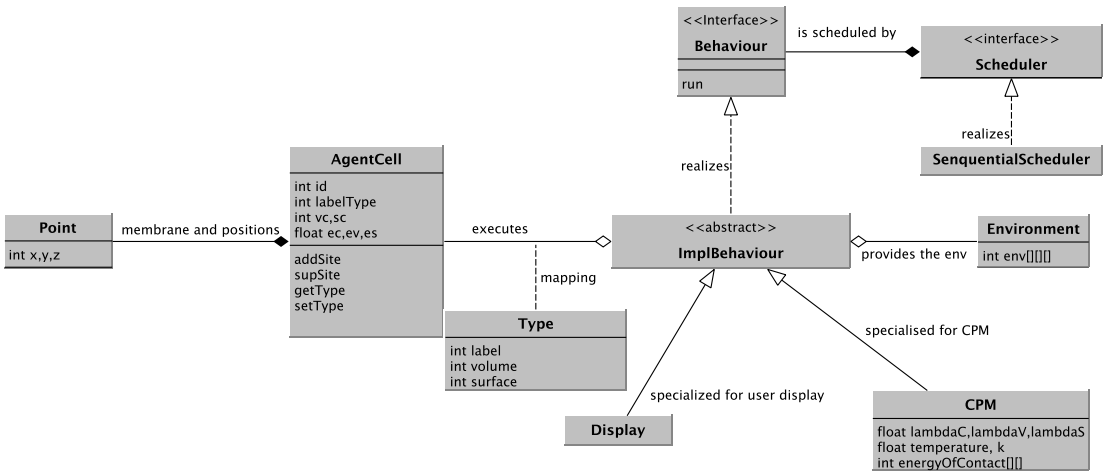


FIGURE 2.6 – Diagramme de classes UML de la mise en œuvre du CPM suivant l'approche Multi-Agents

l'environnement. La classe Display définit le rendu graphique pour l'utilisateur. La classe Environment définit l'environnement, dans notre cas un tableau à 3 dimensions.

La classe CPM définit un pas de simulation du Modèle de Potts Cellulaire ($cf.$ la section 2.1.4). Elle permet aussi d'imposer une dynamique asynchrone chaotique au système multi-agents, respectant ainsi la formalisation du système faite dans la section précédente. Ces attributs représentent la température et la constante k définie dans le Modèle de Potts Cellulaire, la matrice définissant les énergies de contacts entre les différents types cellulaires et les facteurs (lambda) de contact, de volume et de surface.

La classe AgentCell définit un agent, et elle est composée d'un ensemble de points pour représenter les sites où l'agent est présent et ceux qui sont sur la membrane. Elle a comme attributs son identifiant et un label type. Ce label est utilisé par les fonctions getType et setType et à l'aide de la classe Type, elles permettent de faire la correspondance entre un label type et un objet Type. Les attributs vc, sc, ec, ev, es représentent respectivement le volume courant, la surface courante et les énergies de contacts, de volume et de surface. La classe type quant à elle définit le volume cible et la surface cible. Les méthodes addSite et supSite de la classe AgentCell permettent de supprimer un site de l'agent et d'ajouter un site à l'agent. Ainsi, la méthode run de la classe CPM utilise ses deux méthodes. Cette méthode run est définie de la manière suivante : 2 agents voisins par rapport au site (i, j) (C_1 et C_2) sont choisis aléatoirement via les membranes des agents. C_1 exécute sa méthode de croissance addSite(i,j) et C_2 appelle sa méthode décroissance supSite(i,j). Si l'énergie totale entre les deux agents augmentent alors si la probabilité de Monté Carlo n'est pas acceptée, l'agent C_1 exécute sa méthode de décroissance supSite(i,j) et C_2 appelle sa méthode croissance addSite(i,j).

Dans le Modèle de Potts Cellulaire le choix du site pouvant être modifié se fait aléatoirement même s'il n'existe pas un site voisin contenant un agent différent. La classe CPM permet donc d'optimiser ce choix en choisissant un site qui se trouve sur une membrane des agents. Pour cela, la classe AgentCell doit maintenir cette information au cours de l'exécution du programme. Ceci est tout à fait nouveau dans le CPM. Nous donnons donc l'algorithme des méthodes supSite(i,j) et addSite(i,j) dans la section suivante.

2.2.3 Algorithme

Cette section présente les algorithmes des deux méthodes clés de notre mise en œuvre : addSite et supSite. Ces deux méthodes permettent d'ajouter un site et supprimer un site donc de déplacer l'agent. Pour des problèmes d'optimisation (choisir aléatoirement des sites sur les membranes et pour les calculs d'énergie) ces méthodes doivent maintenir à jour les sites auxquels l'agent est présent, les sites se trouvant sur sa membrane et le calcul des énergies de volume surface et contacts. Les algorithmes sont donnés pour des sites représentés en 2D pour des problèmes de lisibilité. Pour des sites représentés en 3D, la traduction de l'algorithme se faite de façon directe en remplaçant (i,j) par (i,j,z). L'algorithme pour la méthode addSite est donné dans le listing 2.1 et pour la méthode supSite dans le listing 2.2.

Listing 2.1 – méthode addSite(i j) de l'agent C_σ

```
1  P_σ ← P_σ + (i,j) /* l'agent ajoute le site dans sa mémoire */
2  s_{i,j} ← σ /*mise à jour de l'environnement avec la cellule qui croît*/
3  Vσ ← Vσ + 1 /*la cellule augmente son volume de 1*/
4  Ev_σ ← (Vσ_t − Vσ)^2 /*met à jour son énergie de volume*/
5  /*V(i,j) retourne les voisins du site (i,j)*/
6  n ← {(i',j') ∈ V(i,j) | s_{i',j'} ≠ σ} /*n les sites voisins avec des agents différents*/
7  Sσ ← Sσ + #n /*la cellule met à jour sa surface*/
8  Es_σ ← (Sσ_t − Sσ)^2 /*met à jour son énergie de surface*/
9  Ec_σ ← Ec_σ + Σ_{(i',j')∈n} T(σ, s_{i',j'}) /*met à jour son énergie de contacts*/
10 si (#n > 0){ /*si l'agent à des voisins*/
11   M_σ ← M_σ + <(i,j), n> /*il met à jour les sites de sa membrane*/
```

```
12 }
13 n ← {(i′, j′) ∈ V(i, j)} /*n les sites voisins quel que soit l'agent*/
14 pour chaque (i′, j′) ∈ n(i, j) { /* Pour chaque agent (i',j') du site modifié (i,j)
15 les énergies et la membrane de l'agent C_{s_{i′,j′}} sont mises à jour*/
16   n_tmp ← {(i″, j″) ∈ V(i′, j′)|s_{i″,j″} ≠ s_{i′,j′}}
17   σ′ ← s_{i′,j′}
18   si (#n_tmp = 0∧ < (i′, j′), L >∈ M_{σ′}){ /*si l'agent voisin n'a plus de voisins
19 au niveau du site (i',j') et si avant le changement il avait un voisin,
20  il supprime les données anciennes*/
21     Sσ′ ← Sσ′ − #L /*met à jour : sa surface*/
22     Es_{σ′} ← (Sσ′_t − Sσ′)² /*son énergie de surface*/
23     M_{σ′} ← M_{σ′} − < (i′, j′), L > /*sa membrane*/
24     Ec_{σ′} ← Ec_{σ′} − ∑_{(i″,j″)∈L} T(σ′, s_{i″,j″})/*énergie de contacts*/
25   }
26   si (#n_tmp > 0){ /*si l'agent voisin à des voisins au niveau
27 du site (i',j')*/
28     si (< (i′, j′), L >∈ M_{σ′}){ /*si avant le changement il avait un voisin,
29 il supprime les données anciennes*/
30       Sσ′ ← Sσ′ − #L/*il met à jour : sa surface*/
31       M_{σ′} ← M_{σ′} − < (i′, j′), L > /*supprime les sites de sa membrane*/
32       Ec_{σ′} ← Ec_{σ′} − ∑_{(i″,j″)∈L} T(σ′, s_{i″,j″})/*met à jour son énergie de contacts*/
33     }
34     Sσ′ ← Sσ′ + #n_tmp /*il met à jour : sa surface en*/
35     Es_{σ′} ← (Sσ′_t − Sσ′)²/*son énergie de surface*/
36     M_{σ′} ← M_{σ′} + < σ′, n_tmp > /*supprime le site de sa membrane*/
37     Ec_{σ′} ← Ec_{σ′} + ∑_{(i″,j″)∈n_tmp} T(σ′, s_{i″,j″}) /*met à jour son énergie de contacts*/
38   }
39 }
```

L'agent C_σ a une méthode addSite qui met à jour l'environnement avec son identifiant le site (i, j) (l.2), son volume (l.3), sa surface (l.7), sa membrane (l.11) et ses énergies de volume (l.4), de surface (l.8) et de contacts (l.9). Comme la valeur du site (i, j) dans l'environnement a été modifiée, les énergies des agents se trouvant au voisinage de (i, j) ont pu être modifiées. Pour chacun des agents voisins $C_{\sigma'}$ au site (i, j) (l.14), soit le site voisin (i', j') sur lequel se trouve $C_{\sigma'}$.

- Si (i', j') n'est pas sur la membrane de $C_{\sigma'}$ et que avant la modification de l'environnent (l.2) ce site était sur la membrane de $C_{\sigma'}$ (l.18 cela est possible seulement si $\sigma = \sigma'$), les énergies de surface (l.22) et de contacts (l.24), la surface (l.21) et la membrane (l.23) de $C_{\sigma'}$ sont mises à jour par rapport à la perte de (i', j') sur sa membrane.

- Si (i', j') est sur la membrane de $C_{\sigma'}$.
  - Si avant la modification de environnent (l.2) ce site était sur la membrane de $C_{\sigma'}$ (l.28), les énergies de contacts (l.32), la surface (l.30) et la membrane (l.31) de $C_{\sigma'}$ sont mises à jour puisque l'agent sur le site (i, j) n'est plus le même. Cette mise à jour supprime le site de la membrane car dans la suite de l'algorithme il est remis à jour avec la prise en compte du nouvel agent sur le site (i, j).

  Les énergies de contacts (l.37), la surface (l.34) et la membrane (l.36) de $C_{\sigma'}$ sont mises à jour par rapport à l'ajout du site (i', j') sur la membrane sachant que l'agent C_σ est sur (i, j).

Listing 2.2 – supSite(i j) de l'agent C_σ

```
1 Vσ ← Vσ − 1 /*décroît son volume de 1 site*/
2 Ev_σ ← (Vσ_t − Vσ)²/*mise à jour de son énergie de volume*/
```

3 $S\sigma \leftarrow S\sigma - \#L$ where $< (i,j), L > \in M_\sigma$ /*mise à jour de sa surface*/
4 $Es_\sigma \leftarrow (S\sigma_t - S\sigma)^2$ /*mise à jour de son énergie de surface*/
5 $Ec_\sigma \leftarrow Ec_\sigma - \sum_{(i',j') \in neighbour(i,j)} T(\sigma, s_{i',j'})$ /*mise à jour de son énergie de contacts*/
6 $M_\sigma \leftarrow M_\sigma - < (i,j), L >$ /*suppressions du site de sa membrane*/
7 $P_\sigma \leftarrow P_\sigma - (i,j)$ /*suppression du site de la mémoire de l'agent*/

La méthode supSite(i,j) supprime l'agent C_σ du site (i,j). Pour cela le volume (l.1), la surface (l.3), la membrane (l.6) et les énergies de volume (l.2), de surface (l.4) et de contacts (l.4) de l'agent C_σ sont mises à jour. Le site (i,j) n'est pas modifié car la méthode run de la classe CPM assure qu'un autre agent a appelé sa méthode addSite permettant ainsi de modifier l'agent se trouvant sur le site (i,j). En effet, nous rappelons qu'un site ne peut être vide. Pour remplir le site avec aucun agent, le programme crée toujours un agent cellule qui modélise le milieu extérieur. Ceci nous permet d'éviter la gestion des cas particuliers où aucun agent ne se trouve au voisinage d'un autre.

Nous pouvons remarquer que ces deux algorithmes bien qu'ils maintiennent à jour les données des agents dynamiquement, sont tout à fait efficaces car ils ne dépendent pas de tous les agents mais seulement des agents sur les sites voisins (dans nos simulations les voisins sont soit les 4 plus proches dans le cas 2D ou les 6 premiers dans le cas 3D). Ainsi le coût de ces algorithmes ne dépend pas du nombre d'agents. Il ne dépend pas non plus du nombre de sites que contient l'agent car les structures qui implémentent les membranes sont des tables de hachage qui assurent un coût moyen constant. Ces deux algorithmes sont donc optimums de ce point de vu. Cependant comme un agent ne peut se déplacer que par un site à la fois, plus l'agent à un volume important plus il faudra de pas de simulation pour qu'il puisse déplacer tous ses sites.

2.3 Extension du modèle

Dans cette section nous proposons et mettons en œuvre une extension du système multi-agents défini précédemment pour étendre le Modèle de Potts Cellulaire. Cette extension porte sur les comportements suivants : production et consommation des molécules de l'environnement par les cellules, la diffusion des molécules, la migration des cellules, la division des cellules, la mort cellulaire, la différentiation cellulaire et le bilan énergétique de la cellule (2.3.1). Nous ajoutons aussi une forme générique et dynamique aux cellules (2.3.2). Ces comportements sont essentiels dans la morphogenèse [18, 57]. Signalons que la forme dynamique et le bilan énergétique sont tout à fait nouveau dans le Modèle de Potts Cellulaire mais jouent un rôle primordial ($cf.$ section 1.3). L'ajout de ces comportements à l'agent cellule présenté dans la section précédente, définit un nouvel agent appelé MorphoPotts, "Morpho" pour la forme et "Potts" pour l'origine du modèle [80, 79, 78]. Le MorphoPotts est proche de l'agent MorphoBlock [9] par rapport à la production, la consommation de molécules et la migration, mais se différencie par une forme dynamique alors que le cœur du MorphoBlock est un point. Nous examinons aussi les évolutions que le MorphoPotts implique dans la structure du programme (2.3.3) mise en place.

2.3.1 Description du MorphoPotts

L'agent MorphoPotts C_σ^t garde les propriétés de la cellule définie dans le CPM mais a la capacité aussi de : produire et consommer des molécules, migrer vers un gradient de molécules, se diviser, mourir, se différentier et maintenir un bilan énergétique. Il peut aussi cibler une forme générique et dynamique ($cf.$ section 2.3.2). Pour prendre en compte ces évolutions : l'environnement, les agents, les interactions et l'ordonnanceur du système multi-agents sont mis à jour.

Division cellulaire	Un MorphoPotts C_σ^t a une méthode $div(\{b, En, cost\}$ qui donne la capacité de se diviser, où $En \in t_{div}$. Si b est vérifiée alors C_σ^t se divise en deux selon un axe (vertical ou horizontal) et un nouveau MorphoPotts $C_{\sigma'}^{t'}$ est créé selon les probabilités de différentiation. b est une condition pour que la cellule se divise et dépend de l'application ($e.g.$ si l'agent a atteint son volume, si son énergie est suffisante, ...). L'énergie B du MorphoPotts $C_{\sigma'}^{t'}$ est alors égale à e' où $(t', e') = En$ et l'énergie B de C_σ^t est alors égal à $E - e' - c$, c le coût énergétique de la division.
Differentiation	Un MorphoPotts C_σ^t a une méthode $diff()$ qui donne le capacité de se différentier. La probabilité que C_σ^t se différencie en $C_\sigma^{t'}$ est égale à : arg où $(arg, t') \in t_{diff}$
Mort cellulaire	Un MorphoPotts C_σ^t a une méthode $mort()$ qui donne la capacité de mourir. La mort cellulaire (l'apoptose) n'est pas définie dans ce mémoire en terme de durée de vie, mais en terme d'énergie qui traduit le fait que l'agent n'est pas dans un milieu propice. Ceci va permettre de simuler des théories biologiques qui portent sur le darwinisme au niveau cellulaire. Si l'agent meurt alors il perd toutes ses capacités et ne génère plus d'énergie dans le CPM.
Maintenance	un MorphoPotts C_σ^t a une méthode $main(arg)$. $main(arg)$ décrémente de arg l'énergie du MorphoPotts. Ceci modélise le coût énergétique de la maintenance de la cellule

FIGURE 2.7 – Les comportements du MorphoPotts sur lui-même : description et formalisation.

Environement L'environnement du système multi-agents mis en place n'est plus seulement composé d'une matrice Env où est enregistrée la position des agents, mais il est composé d'autant de matrices $EnvM$ que de molécules M sont présentes dans le système. Pour chaque site de $EnvM$ se trouve la quantité de la molécule M.

Agents Pour que l'agent ait les nouvelles capacités, la mémoire de l'agent doit être complétée par rapport à l'agent défini dans la section précédente. L'agent "mémorise" en plus :

- (gx, gy) son centre de gravité.
- l'ensemble $t_{prod} = \{arg, Y\}$ qui associe un entier arg à une molécule Y. arg définit le taux de production de Y.
- l'ensemble $t_{cons} = \{arg, Y\}$ qui associe un entier arg à une molécule Y. arg définit le taux de consommation de Y.
- l'ensemble $t_{mol} = \{arg, Y\}$ qui associe un entier arg à une molécule Y. arg définit le nombre de molécules Y consommées.
- l'ensemble $t_{migr} = \{arg, Y\}$ qui associe un entier arg à une molécule Y. arg définit le taux d'adhérence avec la molécule Y comme substrat.
- B l'énergie interne à l'agent. B peut être vue comme le bilan énergétique de l'agent, car elle augmente si des molécules sont consommées et elle diminue pour la maintenance de l'agent.
- $maxB$ l'énergie maximum que peut avoir un agent.
- l'ensemble $t_{ener} = \{arg, Y\}$ qui associe un entier arg à une molécule Y. arg définit l'énergie interne à la cellule obtenue après consommation de la molécule Y.

Production de molécules	Un MorphoPotts C_σ^t a une méthode $prod(arg, Y)$ où $(arg, Y) \in t_{prod}$. $prod(arg, Y)$ est une méthode qui donne la capacité de produire au niveau de son centre de gravité un gradient de molécules Y dans l'environnement. Si le site (i, j) contient n molécules Y dans la matrice $EnvY$ alors, après la production, le site contient un nombre de molécules Y égal à l'entier (arrondi si c'est un nombre réel) : $n + \frac{arg}{\sqrt{(i-gx)^2+(j-gy)^2}}$
Consommation de Molécules	Un MorphoPotts C_σ^t a une méthode $cons(\{maxB, arg\}, Y)$ où $(arg, Y) \in t_{cons}$. $cons(\{max, arg\}, Y)$ est une méthode qui donne la capacité de consommer dans l'environnement des molécules Y si son énergie B est inférieure à $maxB$ (un MorphoPotts peut seulement "manger" si il a "faim"). Consommer cela signifie que si le site (gx, gy) de la matrice $EnvY$ contient n molécules Y et le site (i, j) de la matrice $EnvY$ en contient n', le nombre de molécules Y dans la matrice $EnvY$ au niveau de (i, j) est modifié tel que la nouvelle valeur est 0 si $n' < \frac{min(n,arg)}{\sqrt{(i-gx)^2+(j-gy)^2}}$ autrement la nouvelle valeur est $n' - \frac{min(n,arg)}{\sqrt{(i-gx)^2+(j-gy)^2}}$. Le nombre de molécules Y consommées au centre de gravité (=$min(n', \frac{min(n,arg)}{\sqrt{(gx+gy^2}})$) est ajouté dans t_{mol}
Migration cellulaire	Un MorphoPotts C_σ^t a une méthode $migr(arg, Y)$ où $(arg, Y) \in t_{migr}$. $migr(arg, Y)$ est une méthode qui donne la capacité de migrer vers (si $arg > 0$) ou contre (si $arg < 0$) un gradient de molécules Y. La méthode $migr(arg, Y)$ complète la fonction d'énergie du CPM en ajoutant une nouvelle énergie $E_{migr} = -arg * \sum_{(i,j)\in M_\sigma} nbMolecules((i, j), Y)$ où $nbMolecules((i, j), Y)$ est le nombre de molécules Ysur le site (i, j) de la matrice $EnvY$. arg est aussi appelé facteur de migration λ_{migr}
Transformation des molécules en énergies	Un MorphoPotts C_σ^t a une méthode $ener(arg, Y)$ où $(arg, Y) \in t_{ener}$. $ener(arg, Y)$ donne la capacité de transformer les molécules consommées en énergie : $B = B + \sum_{(arg,Y)\in t_{ener}} arg * Y$

FIGURE 2.8 – Les comportements du MorphoPotts par rapport aux molécules : description et formalisation.

- l'ensemble $t_{main} = \{arg\}$. arg est coût de maintenance de l'agent.
- l'ensemble $t_{diff} = \{arg, Y\}$ qui associe un entier arg à un type d'agent t. arg définit la probabilité que l'agent se différencie dans le type t.
- l'ensemble $t_{div} = \{arg, Y\}$ qui associe un entier arg à un type d'agent t. arg définit l'énergie que l'agent donne au nouvel agent lors d'une division.

Les comportements des MorphoPotts sont résumés et formalisés dans les figures 2.7 et 2.8. Le MorphoPotts a la capacité de percevoir son environnement local mais aussi au delà à travers la production et la consommation des molécules qui se diffusent. La diffusion étant statique, c'est à dire que le gradient est pré-calculé et imposé dans l'environnement sans modéliser chaque molécule. Nous assumons le fait que la cellule produit un gradient car la vitesse des molécules est supérieure à celle de la cellule. De même la consommation de molécules crée un puits de molécules dans l'environnement. Tous les MorphoPotts d'un même type partagent les mêmes comportements.

Interactions L'agent MorphoPotts définit une nouvelle interaction par rapport au système multi-agents décrit dans la précédente section. Les MorphoPotts peuvent produire des molécules dans l'environnement qui sont utilisées par les autres MorphoPotts soit pour migrer soit pour les consommer et les transformer en énergie. Cette énergie est utilisée par exemple lors de la division. Les trois interactions entre les MorphoPotts sont présentées dans la figure 2.9.

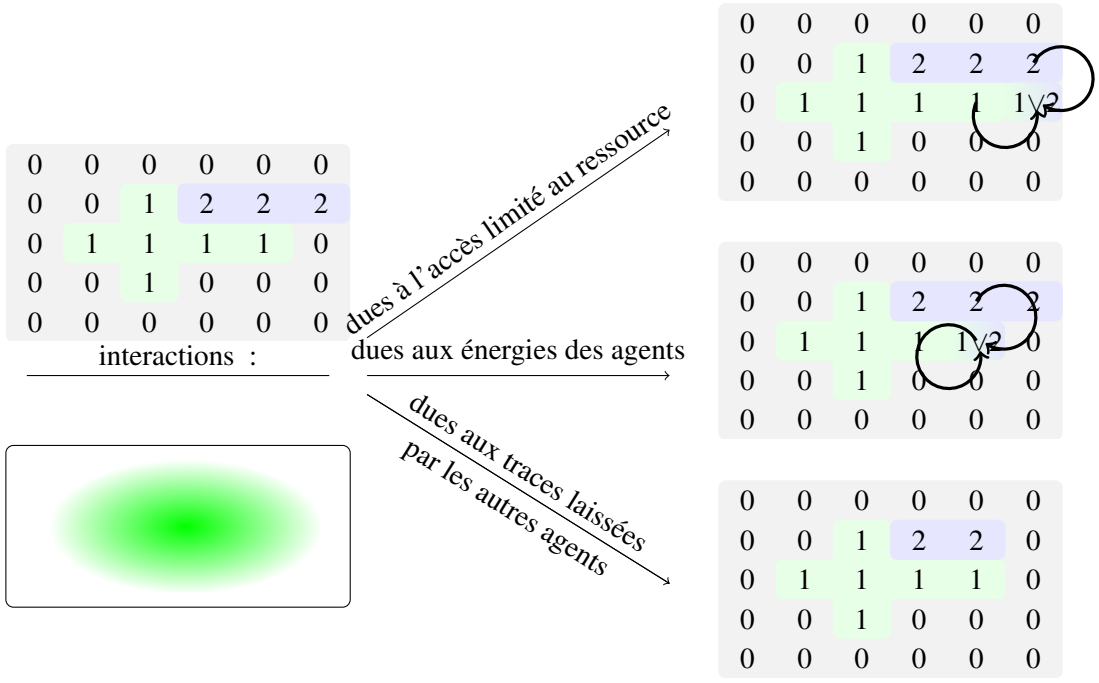


FIGURE 2.9 – Les trois types d'interactions du MorphoPotts. L'environnement consiste en une matrice Env (en haut à gauche) et $EnvY$ (en bas à gauche). L'agent 1 a produit un gradient de molécules Y depuis son centre de gravité. Les deux premières interactions sont définies dans la section 2.1.3. La troisième interaction est due aux molécules laissées par l'agent 1. Dans cet exemple l'agent 2 supprime un de ses sites car il ne trouve pas assez de molécules Y produit par l'agent 1.

Ordonnanceur Dans cette partie nous discutons de l'ordonnanceur qui prend en compte les extensions proposées et du graphe d'interaction qui fixe les paramètres des comportements qui sont ordonnancés. L'ordonnanceur du système multi-agents doit prendre en compte les nouveaux comportements du MorphoPotts. L'ordonnanceur que nous utilisons dans les simulations est le suivant :

1. Soit i égal à 0 et n égal à la taille des membranes de tous les MorphoPotts.
2. tant que i est inférieur de n

   (a) Un pas de l'ordonnanceur (un pas du CPM $cf.$ 2.1.4) est exécuté en ajoutant les nouvelles énergies : énergie de migrations et énergie de formes ($cf.$ la section suivante)
   (b) Pour les deux MorphoPotts choisis dans le pas du CPM, si leur critère de division est vérifié alors les MorphoPotts se divisent
   (c) i est incrémenté de 1
3. Tous les MorphoPotts exécutent leur méthode de maintenance.
4. Tous les MorphoPotts exécutent leur méthode de production.
5. Tous les MorphoPotts (l'ordre dans lequel les agents sont choisis est aléatoire) exécutent leur méthode de consommation.
6. Tous les MorphoPotts exécutent leur méthode de transformation de molécules consommées en énergie.
7. Pour chaque MorphoPotts si leur énergie B est égale à 0, les MorphoPotts meurent.

L'ordonnanceur exécute les différents comportements du MorphoPotts une fois que tous les sites se trouvant sur les membranes du MorphoPotts ont eu une chance d'être changés. Les MorphoPotts appellent les méthodes de production, de consommation, et de mort dans cet ordre. L'ordonnanceur donné ci-dessus est utilisé lors de nos simulations mais la mise œuvre de ce système multi-agents permet à l'utilisateur d'en créer d'autres. Le modèle de notre système multi-agents est donné par le diagramme de classes UML (Unified Modeling Language) présenté dans la figure 2.14. Pour représenter la valeur des paramètres des différents comportements qui sont ordonnancés nous construisons un graphe d'interactions. Ce graphe d'interactions va permettre de décrire nos modèles de simulation. La description du graphe d'interactions est donnée dans la figure 2.10.

2.3.2 Ajout de la forme

Dans la section précédente, nous avons construit un modèle de cellule appelé MorphoPotts. Cependant, la forme des cellules n'est pas fortement définie. Un volume et une surface ne caractérisent pas une forme géométrique unique. L'objectif de cette section est de contraindre la cellule à maintenir une forme générique et dynamique.

Plusieurs propositions ont déjà été faites pour cibler la forme des cellules, comme l'élongation des cellules [60], mais à notre connaissance, aucune pour cibler toutes les formes. L'idée est de donner une forme à l'élastique à la cellule. Pour cela on ajoute un ensemble de ressorts à la cellule comme décrit dans la figure 2.11. Modéliser la forme des cellules par des ressorts a déjà été utilisé et validé d'un point de vue biologique [8] , mais pas dans le formalisme du CPM pour modéliser une forme générique. Dans cette section, nous décrivons le formalisme d'une part et d'autre part la mise en œuvre.

Formalisation de la forme élastique Pour contraindre le MorphoPotts à tendre vers une forme en 3D dans le formalisme du CPM, nous définissons une fonction d'énergie E_{sp} qui est ajoutée à la fonction d'énergie du CPM. Cette énergie est donnée par la somme des énergies fournies par un ensemble de ressorts qui décrivent les formes ($cf.$ figure 2.11). E_{sp} est nulle si les formes sont atteintes par les MorphoPotts. L'énergie E_{sp} et l'énergie $E_{r\sigma}$ d'un ressort $R\sigma$ à la position (p, p') (les deux extrémités du ressort au repos) pour un MorphoPotts C_σ sont définies comme :

$$E_{sp} = \sum_{r\sigma} E_{r\sigma} \tag{2.6}$$

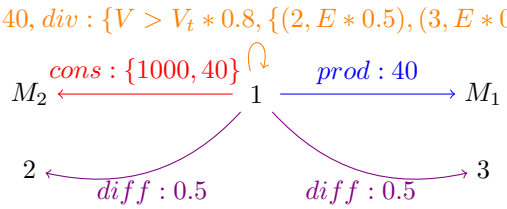
$$E_{r\sigma} = \sum_{a \in P_\sigma} 1/2 * k * S(R\sigma, A) * C(R\sigma, a) * dist(a, R\sigma)^2 \tag{2.7}$$

$$X \xrightarrow{f1 : arg1,..., f2 : argn} Y$$

$$\Leftrightarrow$$

si $X \neq Y$ alors $f1(arg1, Y) \in X_{f1} \wedge ... \wedge fn(argn, Y) \in X_{fn}$
si $X = Y$ alors $f1(arg1) \in X_{f1} \wedge ... \wedge fn(Y) \in X_{fn}$

(a) Sémantique d'une flèche



(b) Exemple d'un graphe d'interactions

FIGURE 2.10 – Graphe d'interactions. Les flèches correspondent aux interactions, les nœuds soit aux types de MorphoPotts soit aux types de molécules. $f(arg, Y) \in X$ signifie que les MorphoPotts de type X ont la capacité donnée par la méthode $f(arg, Y)$. Dans cette figure, le MorphoPotts de type 1 produit un gradient de molécules M_1 dont le nombre de molécules produites à leur centre de gravité est de 40. Si les énergies des MorphoPotts de type 1 sont inférieures à 1000, les MorphoPotts de type 1 peuvent consommer les molécules M_1, soit l'effet inverse de la production, dont la consommation à leur centre de gravité est égale à 40 au maximum. Ce type de MorphoPotts peut également se diviser si son volume actuel est supérieur à 80% du volume cible. Après la division le nouveau MorphoPotts a la même probabilité d'être de type 2 ou de type 3. Le MorphoPotts père donne la moitié de son énergie à son fils. Le coût de maintenance du MorphoPotts 1 est de 40.

où :

- $C(R, a)$ est un critère, renvoie 1 s'il est vérifié sinon 0
- $S(R, a)$ donne le signe de l'énergie, renvoie -1 si R est à la position (p, p') et $a \in [p, p']$ (ressort en compression) sinon 1. En effet l'énergie doit être minimum si la forme est atteinte, $i.e.$ si les sites contenus (en compression) dans la forme appartiennent à l'agent.
- $dist(a, R) = min(|\overrightarrow{ap}|, |\overrightarrow{ap'}|)$
- k est la force au repos du ressort

E_r est égale à la distance au carré de tous les sites qui lui sont associés multipliés par la force au repos. L'ensemble des sites qui lui sont associés, sont les sites a de l'agent qui vérifient le critère $C(R, a)$. La disposition des ressorts dépend de la forme que l'on souhaite imposer. Cette disposition est relative à un MorphoPotts donné. Nous ajoutons donc aux MorphoPotts un système de coordonnées cartésiennes $S_{forme} = (S, Sx, Sy, Sz)$ sur lequel sont disposés les ressorts. Une possibilité est de déposer les ressorts de façon parallèle les uns aux autres comme dans la figure 2.11. Dans cet exemple, un ensemble de ressorts sont disposés perpendiculairement au plan défini par les axes Sx et Sz, $i.e$ les ressorts $R\sigma_p^s$ où $s \in \{+1, -1\}$ dont les deux extrémités sont aux positions (p_x, p_y, p_z) et $(p_x, s * L0_p^s + p_y, p_z)$, $L0_p^s$ étant la longueur au repos du ressort. La distribution des $R\sigma_p^s$ et la longueur $L0_p^s$ dépendent de la forme souhaitée.

Les MorphoPotts ont une énergie $E\sigma_{sp}$ comme la somme des énergies fournies par les ressorts. Dans ce mémoire nous définissons le critère $C(R\sigma_p^s, (i, j, k))$ suivant :
"$R\sigma_p^s$ est le ressort le plus proche de (i, j, k) si un ressort $R\sigma_{p_x,y',p_z}^s$ tel que $dist((i, j, k), R\sigma_p^s) > dist((i, j, k), R\sigma_{p_x,y',p_z}^s)$ n'existe pas"

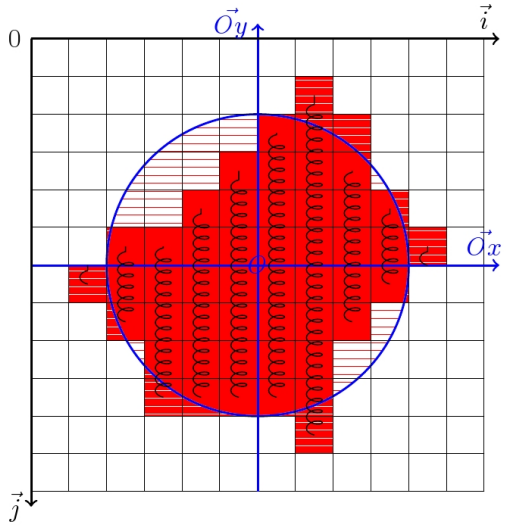


FIGURE 2.11 – Exemple d'une forme élastique. Nous avons un MorphoPotts de type rouge C^r_σ avec une forme élastique. La forme est un cercle représenté en bleu sur la figure. La distribution $sL0^s_p$ des ressorts R^s_p est donnée par la fonction d'un cercle de centre O et de rayon 4, représentée par le cercle bleu. L'énergie de la forme élastique est la somme des distances entre les sites avec les lignes et le cercle bleu. Les sites avec les lignes blanches sont des sites en extension (la longueur du ressort correspondant est supérieure à sa longueur au repos) et ceux avec des lignes rouges sont des sites en compression.

$E\sigma_{sp}$ dans ce mémoire est définie comme :

$$E\sigma_{sp} = \sum_{R\sigma^s_p} E_{r\sigma^s_p} \tag{2.8}$$

Mise en œuvre de la forme élastique Le calcul de la forme élastique est défini par le calcul du critère C entre un site et un ressort, c'est à dire trouver quel est le ressort qui est le plus poche de ce site. Une mise en œuvre naïve serait de parcourir tous les ressorts et de calculer la distance avec tous les sites de l'agent et déterminer quel ressort a la distance minimum pour chaque site. Le problème est que cette mise en œuvre est trop coûteuse d'un point de vu du temps de simulation.

Dans un pas du CPM seulement la valeur d'un site $s_{i,j,k}$ est modifiée, mettant à jour 2 MorphoPotts C_σ C_α. Nous avons donc :

$$\Delta E_{sp} = 1/2 * (E_{r\sigma^s_p} - E_{r\alpha^{s'}_{p'}}) \tag{2.9}$$

où :

- C_σ est le MorphoPotts qui croit, C_α est le MorphoPotts qui décroit
- (i, j, k) le site ajouté ou supprimé.
- $C(R\sigma^s_p, (i, j, k))$ et $C(R\alpha^{s'}_{p'}, (i, j, k))$ sont vérifiés.

Pour utiliser l'énergie E_{sp} dans le CPM, seulement ΔE_{sp} a besoin d'être calculé, c'est à dire finalement vérifié $C(R\sigma^s_p, (i, j, k))$ et $C(R\alpha^{s'}_{p'}, (i, j, k))$. Cela revient à calculer pour deux sites le ressort le plus proche. Une mise en œuvre naïve serait de parcourir tous les ressorts du MorphoPotts C_σ et de déterminer lequel est le plus proche du site (i, j, k), de même avec le MorphoPotts C_α.

Une manière efficace, de calculer ΔE_{sp} est de pré-calculer et d'enregistrer les énergies des ressorts selon les différentes configurations possibles. C'est à dire définir pour un grand nombre de sites dans le repère de la forme S_{forme}, l'énergie correspondante. Ainsi la recherche du ressort le plus près est faite

une fois, avant la simulation. La figure 2.12 montre un exemple d'énergies pré-calculées pour les sites qui entourent la forme. Le nombre de sites où l'énergie est pré-calculée doit être suffisamment grand pour couvrir les sites qui sont ajoutés au MorphoPotts au cours de la simulation. Cette mise en œuvre peut être lourde au niveau de la mémoire nécessaire, si tous les agents ont leurs propres formes. Cependant les modèles de simulation classique que nous faisons, impliquent un grand nombre de MorphoPotts mais avec un nombre de types de MorphoPotts faible. Ainsi les énergies pré-calculées sont enregistrées au niveau des types des MorphoPotts et non pour chaque MorphoPotts. Ceci réduit considérablement la mémoire utilisée. Chaque type de MorphoPotts C_σ^t a une matrice nommée $Forme_t$ qui enregistre ces énergies. Finalement calculer ΔE_{sp}, revient à faire un changement de repère. En effet soit p le site qui est ajouté à C_σ et supprimé à C_α lors d'un pas de simulation du CPM. Calculer l'énergie des ressorts par rapport à p, revient à déterminer la position du point p qui est définie dans le repère de l'environnement, dans le repère de la forme S_{forme} du MorphoPotts C_σ^t et dans le repère de la forme S_{forme} du MorphoPotts $C_\alpha^{t'}$:

$$\begin{aligned}\Delta E_{sp} =& Forme_t(rot(Ro_\sigma, tra(Tr_\sigma, p)) + rot(Ro_\sigma, tra(Tr_\sigma, p)))+ \\ & Forme_{t'}(rot(Ro_\alpha, tra(Tr_\alpha, p)) + rot(Ro_\alpha, tra(Tr_\alpha, p)))\end{aligned} \tag{2.10}$$

où

- $rot(\theta, P)$ renvoie le point défini par la rotation P par le vecteur θ
- $tra(V, P)$ renvoie le point défini par la translation de P par le vecteur V
- Tr_σ est un vecteur défini dans le MorphoPotts C_σ pour indiquer le vecteur translation entre le repère de l'environnement et le repère de la forme
- Ro_σ est un vecteur défini dans le MorphoPotts C_σ pour indiquer le vecteur rotation entre le repère de l'environnement et le repère de la forme. Le changement de repère entre l'environnement et la forme est effectué par une translation de vecteur Tr suivi d'une rotation de vecteur Ro.

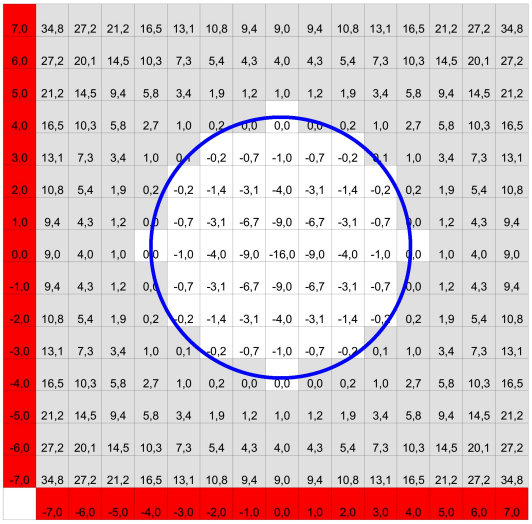


FIGURE 2.12 – Exemple des énergies pré-calculées fournies par les ressorts dans la matrice $Forme$. Les énergies fournies par les ressorts sont pré-calculées avant la simulation afin de gagner en performance. Les énergies sont calculées dans cet exemple pour les valeurs du repère de la forme S_{forme} comprise entre [-7,7] où le centre de la forme est à l'origine. Ici les ressorts sont disposés parallèlement pour cibler le cercle bleu comme décrit dans la figure 2.11. Les sites en compression à l'intérieur de la forme ont une énergie négative, car l'énergie minimum du système doit vérifier que le forme soit atteinte. Pour calculer les énergies le paramètre k du ressort est égal à 1.

Finalement, la mise œuvre présentée ici est particulièrement efficace, car elle n'engendre pas de coût supplémentaire d'un point de vue du temps de simulation. Calculer l'énergie pour cibler une forme dynamique et générique revient à calculer la translation et la rotation de 2 points. Le coût est constant, il est indépendant du nombre d'agents et du volume de l'agent.

Les vecteurs Tr et Ro doivent être mis à jour pour donner aux MorphoPotts la capacité de se déplacer, par exemple pour suivre un gradient de molécules. En effet, Tr et Ro sont initialisés pour positionner les agents à un endroit de l'environnement, si Tr et Ro ne sont pas modifiés en fonction des autres énergies, l'agent reste fixé à l'endroit où la forme a été initialisée. Dans les paragraphes suivant nous discutons de la rotation et de la translation de cette forme pour permettre aux MorphoPotts de se déplacer. Nous montrons comment les sites ajoutés au MorphoPotts permettent de mettre à jour le vecteur rotation et translation.

Rotation de la forme Ce paragraphe décrit comment la forme dynamique et générique donnée à un MorphoPotts peut tourner dans l'environnement. Par exemple, si un MorphoPotts est attiré par un gradient de molécules, le site qui est le plus proche de la source a une probabilité plus forte que les autres sites, d'être ajouté au MorphoPotts. Ce comportement permet au MorphoPotts de tourner dans la direction du gradient. Pour que la forme n'empêche pas ce genre de comportements, le repère où elle est définie doit tourner vers cette direction. La rotation de la forme par rapport à l'environnement pour un MorphoPotts C_σ est donnée par le vecteur rotation Ro_σ.

A chaque site (p_x, p_y, p_z) ajouté ou supprimé à un MorphoPotts C_σ, le vecteur rotation est mis à jour de la façon suivante :

$$\begin{aligned} Ro_\sigma =& Ro_\sigma + \lambda_{ro}/V_\sigma * < arctan2(p_y, p_z), arctan2(p_z, p_x), arctan2(p_x, p_y) > \\ & \text{si le site est ajouté} \\ Ro_\sigma =& Ro_\sigma - \lambda_{ro}/V_\sigma * < arctan2(p_y, p_z), arctan2(p_z, p_x), arctan2(p_x, p_y) > \\ & \text{si le site est supprimé} \end{aligned} \tag{2.11}$$

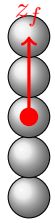
où :

– $arctan2(y, x)$ retourne l'angle $\widehat{AOB}$, où O est l'origine du repère, $A = (x, y)$ et $B = (x, 0)$

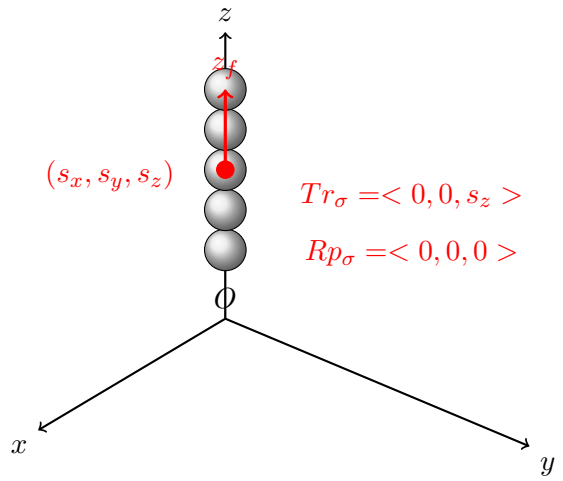
Ainsi après chaque ajout ($resp.$ suppression) d'un site la forme s'oriente dans la direction ($resp.$ dans la direction opposée) de ce site. Cette orientation est normalisée par le volume V_σ de l'agent. En effet, si un MorphoPotts a un volume de 100 et qu'un site est ajouté, l'influence de ce site sera faible. Le paramètre λ_{ro} est appelé facteur de rotation, il permet d'augmenter ou de réduire l'effet de l'ajout d'un site sur la rotation. La figure 2.13 montre un MorphoPotts où une forme en bâtonnet a été donnée et la mise à jour des rotations en fonction des sites ajoutés et supprimés.

Translation de la forme élastique Ce paragraphe décrit comment la forme dynamique et générique donnée à un MorphoPotts peut se déplacer dans l'environnement. Par exemple, si un MorphoPotts est attiré par un gradient de molécules, le site qui est le plus proche de la source a une probabilité plus forte d'être ajouté au MorphoPotts. Ce comportement permet aux MorphoPotts de se déplacer dans la direction du gradient. Pour que la forme n'empêche pas ce genre de comportements, le repère où elle est définie doit aussi être translaté dans cette direction. La translation de la forme par rapport à l'environnement pour un MorphoPotts C_σ est donnée par le vecteur rotation Tr_σ.

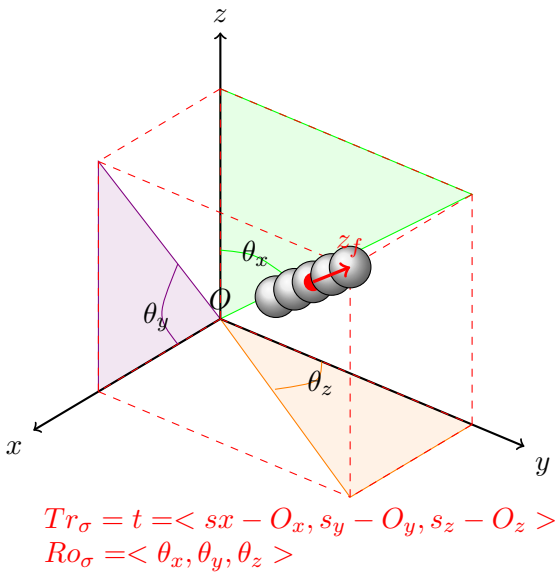
A chaque site (p_x, p_y, p_z) ajouté ou supprimé à un MorphoPotts C_σ, le vecteur translation $Tr_\sigma =$



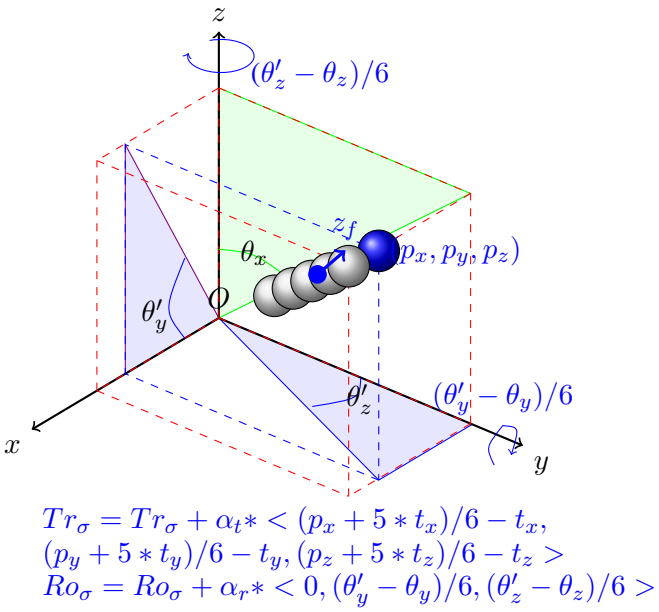
(a) Choix de la forme



(b) Initialisation d'un MorphoPotts C_σ avec cette forme dans l'environnement



(c) Le MorphoPotts a déplacé et a effectué des rotations de sa forme au cours de la simulation et il a atteint sa forme



(d) Translation et rotation de la forme lors de l'ajout d'un site au MorphoPotts

FIGURE 2.13 – Exemple de translation et de rotation de la forme imposée à un MorphoPotts. La figure 2.13(a) présente la forme que cible un MorphoPotts. Cette forme est un bâtonnet composée de 5 sites alignés, l'origine du repère cartésien de la forme est le centre de gravité de la forme, représenté par le cercle rouge. La figure 2.13(b) montre un MorphoPotts C_σ ayant cette forme et placé dans l'environnement. L'origine du repère de la forme est bien au centre de gravité du MorphoPotts, mais l'origine de la forme n'est pas le même que celui de l'environnement. Le MorphoPotts sauvegarde la translation Tr_σ de l'origine du repère où est définie sa forme, dans cet exemple seulement un décalage selon l'axe des z et la rotation Ro_σ de sa forme par rapport au repère de l'environnement. La figure 2.13(c) montre l'état du MorphoPotts après plusieurs pas de simulation. En fonction d'autres énergies le MorphoPotts a pu se déplacer, donc mettre à jour Tr_σ et Ro_σ. Dans cet exemple le MorphoPotts après s'être déplacé et donc déplacer sa forme, a atteint sa forme cible. Aussi la représentation d'un point dans l'environnement a sa représentation dans la forme en subissant la translation Tr_σ puis la rotation Ro_σ. La figure 2.13(d) montre un exemple de mise à jour de Tr_σ et Ro_σ si un site est ajouté au MorphoPotts.

(t_x, t_y, t_z) est mis à jour de la façon suivante :

$$\begin{aligned} Tr_\sigma =& Tr_\sigma + \lambda_{tr} * < \frac{p_x - (t_x * V_\sigma - 1)}{V\sigma}, \frac{p_y - (t_y * V_\sigma - 1)}{V\sigma}, \frac{p_z - (t_z * V_\sigma - 1)}{V\sigma} > \\ & \text{si le site est ajouté} \\ Tr_\sigma =& Tr_\sigma - \lambda_{tr} * < \frac{p_x - (t_x * V_\sigma - 1)}{V\sigma}, \frac{p_y - (t_y * V_\sigma - 1)}{V\sigma}, \frac{p_z - (t_z * V_\sigma - 1)}{V\sigma} > \\ & \text{si le site est supprimer} \end{aligned} \tag{2.12}$$

Ainsi après chaque ajout ($resp.$ suppression) d'un site la forme se déplace dans la direction ($resp.$ dans la direction opposée) de ce site. Ce déplacement est normalisé par le volume V_σ de l'agent. En effet, si un MorphoPotts a un volume de 100 et qu'un site est ajouté, l'influence de ce site sera faible. Le paramètre λ_{tr} est appelé facteur de translation, il permet d'augmenter ou de réduire l'effet de l'ajout ou de la suppression d'un site sur la translation. Si $\lambda_{tr} = 1$ alors la forme tout au long de la simulation aura toujours la même position relative par rapport au centre de gravité du MorphoPotts. La figure 2.13 montre un MorphoPotts où une forme en bâtonnet a été donnée avec la mise à jour des translations en fonction des sites ajoutés et supprimés.

2.3.3 Mise à jour de la structure du programme

La MorphoPotts ajoute de nouveaux comportements par rapport au Modèle de Potts Cellulaire qui sont soit décrits dans le formalisme du MorphoPotts en terme d'énergie (pour la forme et la migration) soit de manière indépendante (division, mort, différentiation, transformation de molécules en énergie, production et consommation de molécules). La section 2.3.1 a montré quelles extensions ont été nécessaires pour définir le MorphoPotts dans le système multi-agents déjà mis en place. Cette section étudie les extensions dans la structure du programme. Le diagramme de classes UML prenant en compte le MorphoPotts est donné dans la figure 2.14.

Les comportements non décrits dans le formalisme de CPM ont tous une classe qui les définit : Maintain, Produce, Consume Divide Differentiate, TransformMoltoEner. Toutes ces classes spécialisent la classe ImplBehaviour car ce sont des comportements portant sur les agents représentés dans l'environnement. La classe EnvMorphoPotts spécialise la classe Env, afin qu'elle soit composée non plus d'une seule matrice pour placer les agents, mais composée d'un ensemble de classes EnvMol pour sauvegarder la quantité de molécules présentes.

Une classe AgentMorphoPotts est définie pour représenter l'Agent MorphoPotts. Elle hérite de la classe AgentCell définie précédemment et qui correspond à la cellule définie dans le Modèle de Potts Cellulaire. La classe AgentMorphoPotts est spécialisée pour ajouter de nouvelles méthodes comme la division, et des attributs comme le centre de gravité. L'attribut labelType de la classe AgentMorphoPotts qui hérité de la classe AgentCell permet d'identifier TypeMorphoPotts. La classe TypeMorphoPotts spécialise la classe Type et donne les paramètres des agents MorphoPotts. La classe TypeMorphoPotts donne entre autre la forme, le taux de production, de consommation des molécules qui sont partagés par les agents d'un même type.

Finalement, la structure du programme prenant en compte le MorphoPotts qui repose fortement sur la description multi-agents, n'a pas impliqué de changement profond juste des spécialisations de classes déjà existantes. L'approche multi-agents nous permet donc de concevoir facilement notre programme de façon qu'il permette l'ajout de nouveaux comportements. Ceci n'aurait pas était possible si par exemple le paradigme des automates cellulaires avait été choisi, car les comportements définis dans le MorphoPotts ne sont pas décrits dans le même formalisme.

Dans le chapitre suivant nous validons :

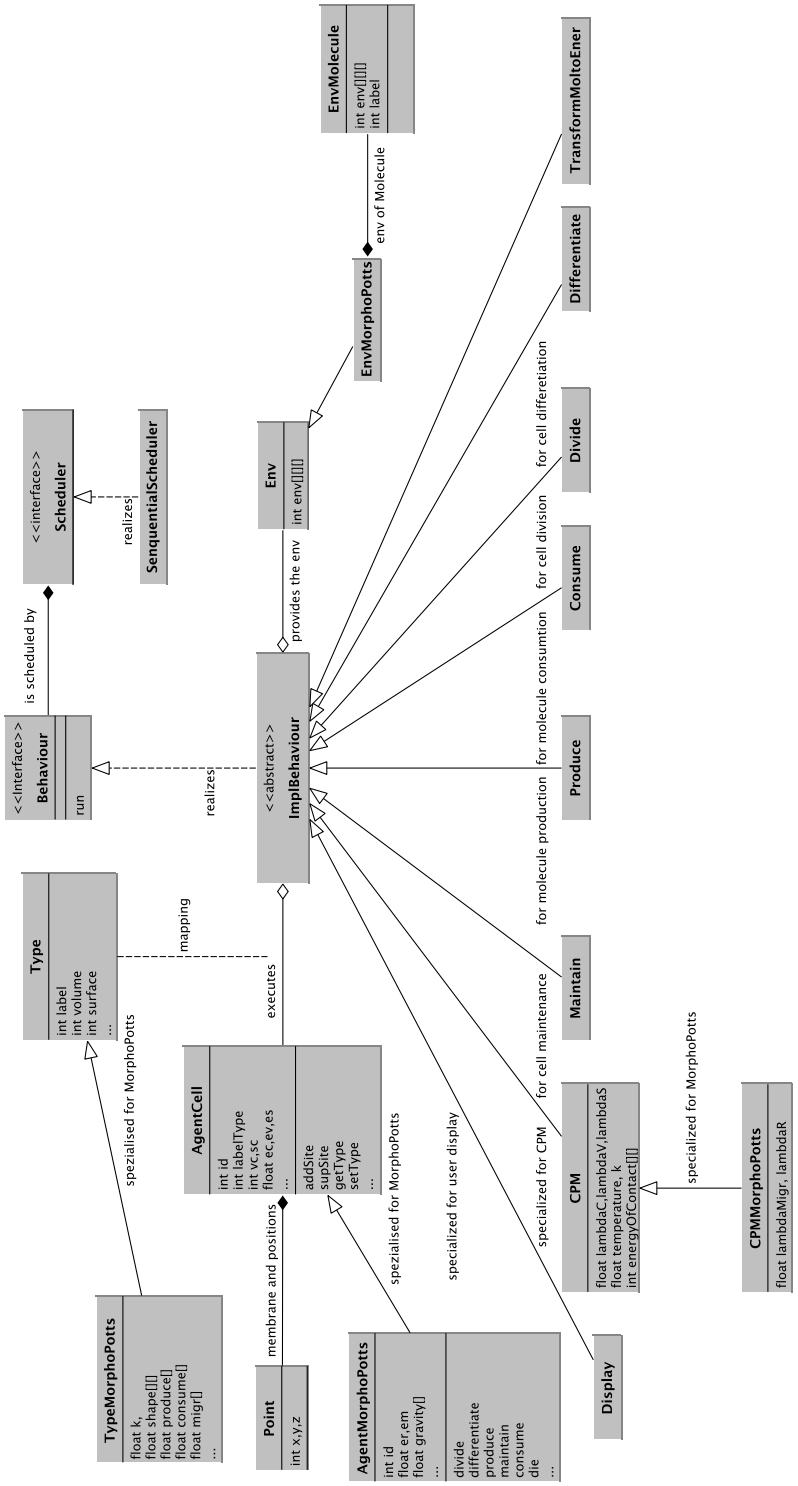


FIGURE 2.14 – Diagramme de classes UML de la mise en œuvre prenant en compte le MorphoPotts

- le système multi-agents sans le MorphoPotts via des modèles de simulation dont les résultats sont connus.
- la forme du MorphoPotts en ciblant une forme imposée.

Le MorphoPotts définit un bilan énergétique, nous montrons dans le chapitre suivant comment ce bilan énergétique permet aux cellules de s'auto-organiser et de construire des tissus cohérents et dynamiques. Nous montrons aussi comment la forme permet de structurer les tissus. Nous essayons de déterminer dans quel cas l'auto-organisation peut être détectée automatiquement ou pas.

Chapitre 3

Application

Ce chapitre valide le système multi-agents (SMA) mis en œuvre et analyse l'auto-organisation entre les cellules à l'aide de ce SMA. Des éléments de validation sont donnés en se basant sur des simulations dont le résultat est connu (3.1). L'auto-organisation étudiée dans ce mémoire s'appuie sur une théorie du darwinisme au niveau cellulaire et sur une théorie du bilan énergétique donnée aux cellules (3.2) . Pour toutes les applications, si cela est possible, nous donnons un critère basé sur l'entropie pour détecter l'auto-organisation. De manière plus techniques, notons que les simulations présentées dans ce chapitre ont été simulées avec un pc Intel Core 2 Quad 2.83 GHz avec 3 Go de RAM sous Linux.

3.1 Éléments de validation

Dans cette section, nous donnons des éléments de validation du SMA défini dans le chapitre précédent. Pour cela, nous vérifions premièrement la traduction du Modèle de Potts Cellulaire (CPM) en SMA à travers un modèle de tri cellulaire (3.1.1), puis nous validons les deux nouvelles énergies que nous avons ajoutées à l'agent MorphoPotts et qui ne sont définies pas dans le formalisme du CPM à travers 2 applications : la migration sur un substrat (3.1.2) et la forme générique et dynamique (3.1.3).

3.1.1 Tri cellulaire

Le tri cellulaire est une application typique du CPM dont la dynamique est connue. En la comparant avec la dynamique de nos simulations nous vérifions la mise œuvre de notre SMA. Nous décrivons dans un premier temps le phénomène du tri cellulaire, puis nous abordons le modèle de simulation et enfin nous discutons des résultats obtenus.

Description

Le tri cellulaire est un phénomène qui se produit durant l'embryogenèse. Il se traduit par le fait que des cellules de types différents placées aléatoirement se regroupent. Un tri entre les cellules s'effectue pour permettre aux cellules d'un même type de se regrouper. Le CPM a été construit notamment dans le but de modéliser et simuler ce phénomène par Glazier et Granner en 1992 [36]. Ce modèle est basé sur l'hypothèse que les cellules peuvent se trier simplement grâce à un différentiel d'énergie d'adhésion selon deux types. Deux types cellulaires sont définis : rouge et vert. Les cellules rouges et vertes ont une énergie de contacts telle que les cellules rouges préfèrent être en contact avec les cellules rouges qu'avec les cellules vertes. Le cellules vertes préfèrent être en contact avec les cellules rouges plutôt qu'avec les cellules vertes. Les cellules vertes et rouges préfèrent être en contact entre elles qu'avec le milieu extérieur. A partir d'un placement aléatoire de ces cellules, il a été montré par Glazier et Granner que les énergies

de contacts permettent aux cellules rouges de se regrouper et aux cellules vertes d'entourer ce groupe. La suite de cette section présente en détail le modèle de simulation et les résultats obtenus.

Modèle de simulations

Le modèle de simulation utilise le SMA présenté dans le précédent chapitre. Trois types d'agents sont définis, un type pour les cellules rouges, un autre pour les cellules vertes, et un dernier pour représenter le milieu extérieur. Le tableau 3.1 donne les valeurs des paramètres utilisés. L'énergie d'un état du système est alors définie par :

$$E = \lambda_s E_s + \lambda_v E_v + \lambda_c E_c \tag{3.1}$$

Les énergies de contacts sont plus faibles entre les agents rouges qu'avec les verts, entre les agents verts et rouges qu'avec les agents verts et le milieu extérieur. Ces paramètres sont cohérents avec la description du phénomène car l'énergie minimale du système est atteinte quand les agents rouges sont regroupés (les énergies de contacts sont seulement de 2) et que les agents verts entourent ce groupe (l'énergie est de 12 par rapport à 14 entre les agents verts et de 16 avec le milieu extérieur). Ces valeurs vérifient les inégalités suivantes qui sont nécessaires au tri spontané des cellules données dans [36] :

$$0 < T_{r,r} < \lfloor \frac{T_{r,r} + T_{v,v}}{2} \rfloor < T_{v,r} < T_{v,v} < T_{v,m} = T_{r,m} \tag{3.2}$$

Intuitivement ces inégalités signifient que, pour que le tri des agents se produisent, il faut que l'énergie entre les types rouges et verts soit plus grande que la moyenne des énergies d'un même type. Sinon les agents rouges et les agents verts formeraient un groupe séparément.
Le paramètre kT correspondant originellement à la température et la constante de Boltzmann. La valeur de cette constante ne trouve pas un sens direct dans le CPM. En effet la constante de Boltzmann est une constante physique permettant de relier la température d'un système à son énergie thermique. kT étant utilisé dans le CPM pour définir une probabilité lorsque le système augmente en énergie, il faut considérer kT comme la probabilité qu'une cellule ne soit pas dans un état favorable et la calibrer en déterminant qu'elle déformation la cellule peut raisonnablement subir. Ici $kT = 10$ c'est à dire que le système a une chance sur deux d'accepter une augmentation d'énergie de 7 (0.5=$e^{\frac{-(7)}{10}}$). Nous assumons donc que si les énergies de contacts et de surfaces sont nulles une cellule est autorisée (une chance sur deux) à se déformer de 4 sites, $i.e.$ s'éloigner de son volume cible de 20%.

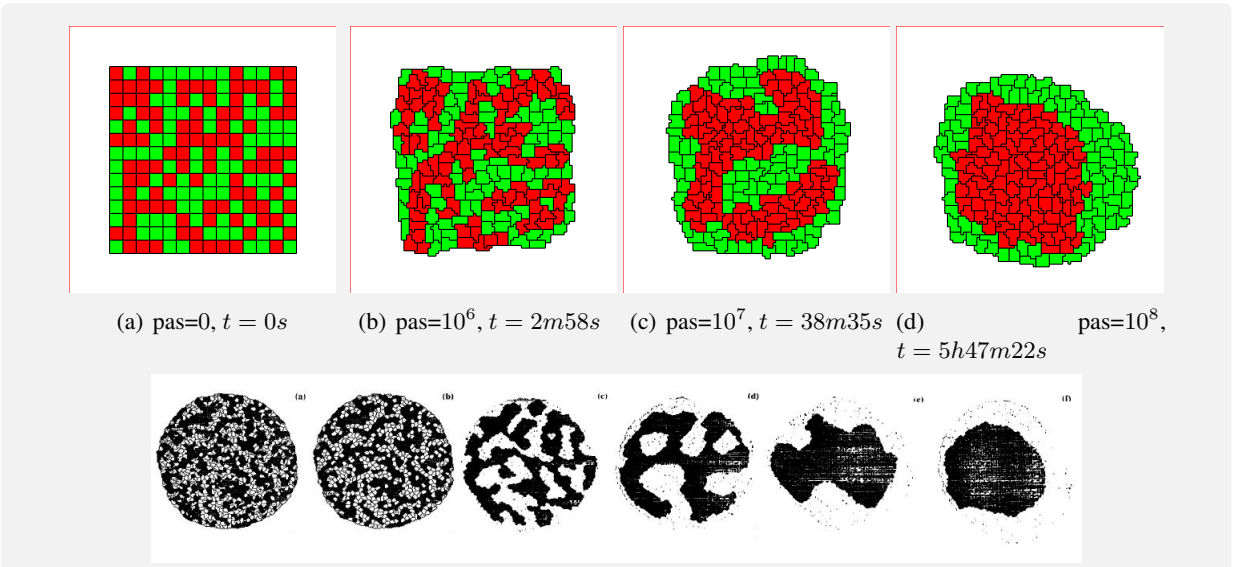
Ce modèle nous sert à tester notre SMA , nous ne discutons pas des effets de la valeur des paramètres sur la dynamique comme cela est fait dans [35]. L'état initial correspond à un environnement de 100x100 où 100 agents rouges et 96 verts sont placés aléatoirement. Le volume de ces agents est de 25 et leur surface de 20.

Résultat

La simulation du modèle est décrite dans la figure 3.1. Les agents rouges se regroupent et forment un disque. Les agents verts entourent ce disque. Nous retrouvons la même dynamique que la simulation faite par Glazier et Graner. Ceci donne des éléments de validation pour notre SMA et montre que notre mise en œuvre garde les propriétés du CPM. L'ordonnancement chaotique asynchrone choisi est donc elle aussi asynchrone. L'énergie du système diminue au fur et à mesure de la simulation. Plus l'énergie baisse, plus l'auto-organisation augmente. Cette auto-organisation se traduit aussi au niveau spatial, un agent rouge a une plus forte probabilité d'avoir un autre agent rouge à coté de lui. Le CPM est un modèle efficace pour étudier l'auto-organisation qui se produit entre les agents. Les sections suivantes valident l'énergie de migration et de forme.

Environnement	Env : 100x100
Agents	3 types d'Agents : rouge r, vert v, milieu m (pour représenter le milieu extérieur)
Type r et v	$V_t = 25, S_t = 20$
Paramètres du CPM	les énergies de contacts sont données dans [36] : $T_{r,r} = 2$, $T_{v,r} = 12$, $T_{r,m} = 16$, $T_{v,v} = 14$, $T_{v,m} = 16$, $kT = 10$, $\lambda_c = \lambda_v = 1, \lambda_s = 0.5$
Fonction de voisinage	Les 4 voisins directs

TABLE 3.1 – Valeur des paramètres utilisés dans le modèle du tri cellulaire de 1992.



(a) pas=0, $t = 0s$ (b) pas=10^6, $t = 2m58s$ (c) pas=10^7, $t = 38m35s$ (d) pas=10^8, $t = 5h47m22s$

(e) simulation réalisée by Glazier and Graner

FIGURE 3.1 – Tri cellulaire. Cette figure montre les résultats de la simulation du tri cellulaire. L'environnement est une matrice 100x100, 100 agents rouges $C^r_{\sigma\in[1,100]}$, 96 agents verts $C^v_{\sigma\in[101,196]}$ et 1 agent C^m_0, sont placés aléatoirement. Le résultat de nos simulations donné dans les figures (a), (b), (c), (d) est similaire à la figure (e). Les agents rouges se regroupent et forment un amas qui est entouré par les agents verts. Le pas est le nombre de de pas de simulation du CPM, $i.e.$ qu'un site a pu changer de valeur. Le temps indiqué est le temps de simulation avec un rafraîchissement de l'image tous les 25000 pas de simulations. Le temps de simulation est raisonnable au vu du nombre de pas de simulation effectués. La vidéo de cette simulation peut être trouvée à l'adresse suivante `http://www.youtube.com/watch?v=NTqWzVpopcA`.

3.1.2 Migration cellulaire sur un substrat

Dans le CPM les agents ne peuvent pas se déplacer. Cette limitation constitue un problème pour modéliser notamment l'embryogenèse où les migrations cellulaires sont importantes. Dans cette section nous mettons en œuvre l'énergie de migration définie dans la partie précédente. Après une description du phénomène en biologie, nous construisons un modèle et nous discutons des résultats de la simulation.

Environnement	Env : 250x250 (matrice des agents), $EnvA$:250x250 (matrice des molécules A)
Agents	2 types d'Agents : rouge r, milieu m (pour représenter le milieu extérieur)
Type r	$V_t = 25, S_t = 20$, migre en présence de molécules A
Paramètres du CPM	$kT = 1, \lambda_c = 0, \lambda_v = 1, \lambda_s = 1, \lambda_{migr} = \{0.1, 0.075\}$
Fonction de voisinage	Les 4 voisins directs

TABLE 3.2 – Exemple de valeurs de paramètres pour notre modèle de migration cellulaire. Deux valeurs sont définies pour montrer l'influence du paramètre du facteur de migration λ_{migr} : un avec $\lambda_{migr} = 0.1$ l'autre avec $\lambda_{migr} = 0.075$.

Description

La chimiotaxie cellulaire est le phénomène dans lequel les cellules d'un organisme mais aussi les bactéries, se déplacent en fonction d'espèces chimiques spécifiques présentes dans l'environnement. Ce phénomène est utilisé par exemple par les bactéries pour trouver du glucose et se diriger vers la source. Il est rendu possible par la présence de récepteurs sur la membrane des cellules spécifiques à une espèce chimique. Dans le cas des bactéries cela permet de donner une direction à leur déplacement aléatoire (sans présence d'un gradient d'une molécule spécifique), dans le cas des eucaryotes cela permet un déplacement des cellules soit dans une direction donnée (chimiotaxie), soit de façon aléatoire (chimiokinèse). Dans la suite de cette section, nous ajoutons la définition de l'énergie de migration E_{migr} permettant de donner la capacité aux cellules de migrer dans une direction donnée (vers ou contre un gradient de molécules).

Modèle de simulation

Pour tester l'énergie E_{migr} définie dans notre extension du CPM, nous proposons deux modèles qui permettent de donner des éléments de validation de la mise en œuvre de E_{migr} mais aussi de décrire les effets du facteur de migration λ_{migr}. Pour l'exemple les modèles proposés sont composés d'un environnement de taille 250x250 où sont placés des agents et des molécules. Les paramètres et leurs valeurs de ces modèles sont donnés dans le tableau 3.2. Deux agents MorphoPotts (C_m, C_r) sont définis, un pour représenter le milieu extérieur (C_m) et l'autre qui modélise la cellule qui migre (C_r). C_r a un volume cible de 25 et une surface cible de 20. L'énergie d'un état du système est égale à l'énergie de volume de surface et de migration telle que :

$$E = \lambda_v E_v + \lambda_s E_s + \lambda_{migr} E_{migr} \tag{3.3}$$

Deux exemples de simulations sont définis, l'un avec $\lambda_v = 1, \lambda_s = 1, \lambda_{migr} = 0.01$ et l'autre avec $\lambda_v = 1, \lambda_s = 1, \lambda_{migr} = 0.075$. L'environnement est initialisé avec C_r à la position $(123, 10)$ ayant un volume cible de 25 et une surface cible de 20. Un gradient de molécules sur lequel C_r peut migrer est imposé dans l'environnement. Le nombre de molécules présent à la position (x, y) est égal à y^2. Les résultats des simulations des deux modèles sont présentés dans la section suivante.

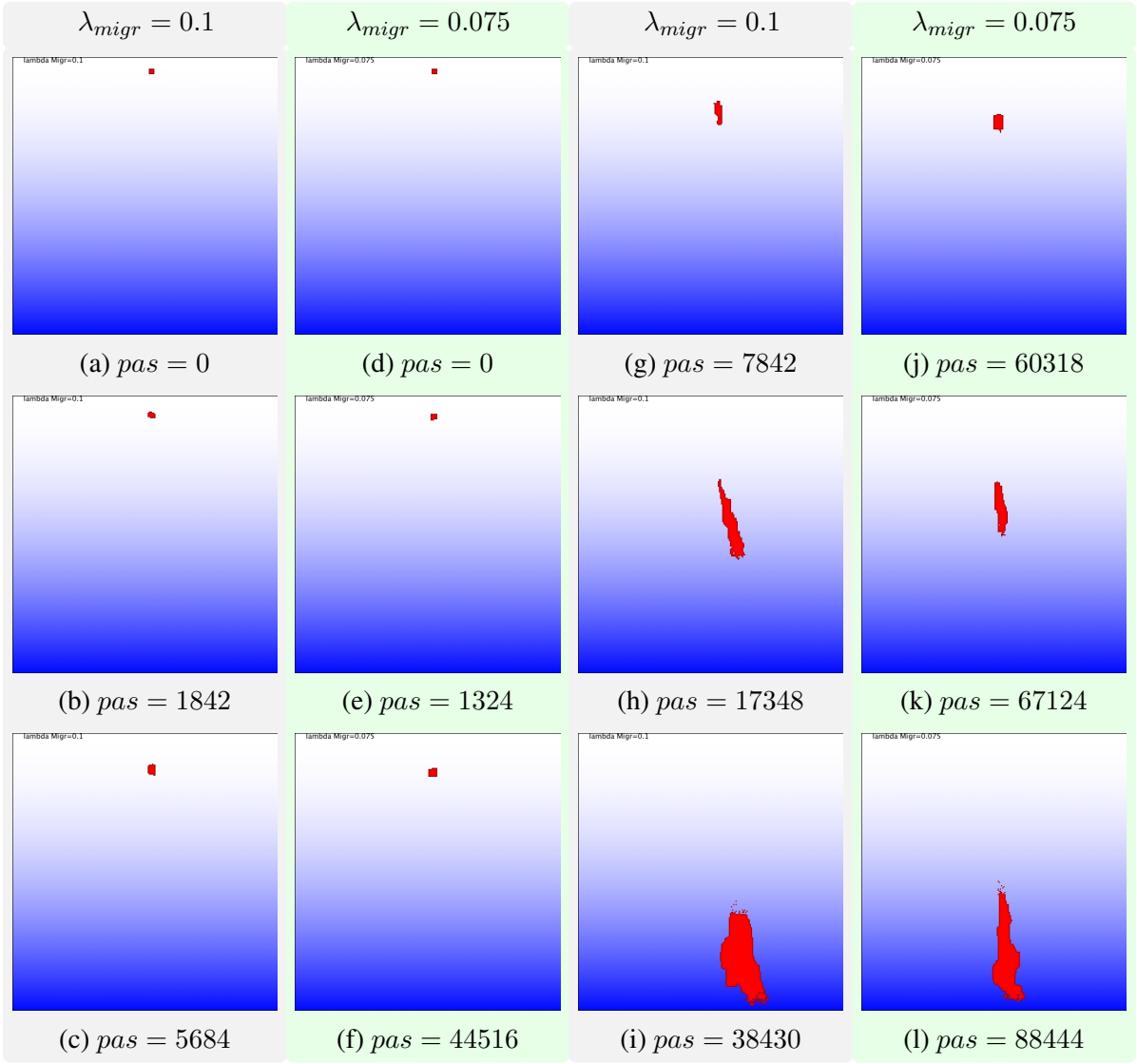


FIGURE 3.2 – Exemple de chimiotaxie. Un MorphoPotts rouge est initialisé dans l'environnement où un gradient de molécules (en bleu) est présent de tel sorte que le nombre de molécules au site (i, j) est égal à $j * j$. Deux simulations sont présentées, une dont le facteur de migration est de 0.1, l'autre dont le facteur est de 0.075. Dans les deux simulations le MorphoPotts migre sur le gradient et se dirige en direction du nombre de molécules le plus important. Cette migration entraîne une déformation du volume et de la surface. La simulation avec $\lambda_{migr} = 0.1$ implique une migration plus rapide que celle avec $\lambda_{migr} = 0.075$. La vidéo de la simulation avec $\lambda_{migr} = 0.1$ peut être vue à l'adresse `http://www.youtube.com/watch?v=2NhaiMqpjEs` et celle avec $\lambda_{migr} = 0.075$ à l'adresse `http://www.youtube.com/watch?v=etd9KrnSVio`

Résultats

Les simulations sont décrites dans la figure 3.2. Dans ces deux simulations, les MorphoPotts se dirigent dans une direction privilégiée. Cette direction est verticale et cible les concentrations de molécules les plus fortes. Ces simulations montrent que l'énergie de migration proposée donne aux MorphoPotts la capacité de chimiotaxie. Ces simulations permettent aussi de calibrer les valeurs du facteur de migration ainsi que le nombre de molécules qui doivent être mises en jeu pour que la cellule ait un volume et une surface

adéquats. Notons que si l'énergie engendrée par la migration est trop forte par rapport à l'énergie de volume et de surface alors la cellule subit une déformation qui ne correspond pas aux données biologiques. Les figures 3.2 (i) et 3.2 (l) montrent une déformation très forte par rapport aux volumes cibles et surfaces cibles décrites dans les figures 3.2 (a) et 3.2 (d).
De manière formelle, $E_{migr} = \lambda_{migr} \sum_{(i,j)\in P} -nbMolecules((i,j),A)$ où P est l'ensemble des sites de l'agent et $nbMolecules((i,j),A)$ est le nombre de molécules A sur le site (i,j) ($cf.$ section 2.3). Plus un site contient de molécules, plus l'énergie nécessaire pour insérer ce site à un MorphoPotts est faible. Formellement les transitions entre les états sont acceptées si :

$$\begin{aligned}
\Delta E \leq 0 \Leftrightarrow \\
\lambda_s \Delta E_s + \lambda_v \Delta E_v + \lambda_{migr} \Delta E_{migr} \leq 0 \Leftrightarrow \\
\lambda_s \Delta E_s + \lambda_v \Delta E_v \leq -\lambda_{migr} \Delta E_{migr} \Leftrightarrow \\
\Delta E_s + \Delta E_v \leq -\lambda_{migr} \Delta E_{migr} \Leftrightarrow \\
\Delta E_s + \Delta E_v \leq -\lambda_{migr} \Delta (-\sum_{(i,j)\in P} nbMolecules((i,j),A)) \\
\Delta E_s + \Delta E_v \leq \lambda_{migr} \Delta (\sum_{(i,j)\in P} nbMolecules((i,j),A))
\end{aligned} \tag{3.4}$$

Comme un seul site (i,j) peut être supprimé ou ajouté à un MorphoPotts durant un pas de simulation et comme dans notre exemple seulement un MorphoPotts C_r est défini (nous rappelons que le MorphoPotts modélisant le milieu extérieur n'as pas de contrainte de volume, de surface et ni de migration) nous avons dans le cas de l'ajout d'un site (i,j) :

$$\begin{aligned}
\Delta E \leq 0 \Rightarrow (\text{car le volume a augmenté de 1} \\
\text{et la surface au maximum de 2)} \\
(S1+2-20)^2 - (S1-20)^2 + (V1+1-25)^2 - (V1-25)^2 \leq \lambda_{migr} nbMolecules((i,j),A) \Leftrightarrow \\
4*(S1-20) + 4 + 2*(V1-25) + 1 \leq \lambda_{migr} * j^2 \Leftrightarrow \\
4*S1 - 80 + 4 + 2*V1 - 50 + 1 \leq \lambda_{migr} * j^2 \Leftrightarrow \\
2*S1 + V1 \leq 1/2 * \lambda_{migr} * j^2 + 62.5
\end{aligned} \tag{3.5}$$

nous avons dans le cas de la suppression d'un site (i,j) :

$$\begin{aligned}
\Delta E \leq 0 \Rightarrow (\text{car le volume a diminué de 1} \\
\text{et la surface a augmentée au maximum de 2)} \\
(S1+2-20)^2 - (S1-20)^2 + (V1-1-25)^2 - (V1-25)^2 \leq -\lambda_{migr} nbMolecules((i,j),A) \Leftrightarrow \\
4*(S1-20) + 4 - 2*(V1-25) + 1 \leq -\lambda_{migr} * j^2 \Leftrightarrow \\
-4*S1 + 80 - 4 + 2*V1 - 50 - 1 \geq \lambda_{migr} * j^2 \Leftrightarrow \\
-2*S1 + V1 \geq 1/2 * \lambda_{migr} * j^2 - 12.5
\end{aligned} \tag{3.6}$$

Dans le cas de l'ajout d'un site, la transition est toujours acceptée si la somme entre le volume courant et la surface courante multipliée par 2 est inférieure à $1/2 * \lambda_{migr} * j^2 + 62.5$. Le MorphoPotts peut donc augmenter son volume et surface de façon affine aux nombres de molécules élevés au carré sur lequel se trouve le site à ajouter. A l'inverse, la suppression d'un site est toujours acceptée si le volume courant est

suffisamment important, $i.e.$ au moins supérieur à $1/2 * \lambda_{migr} * j^2 - 12.5$. Le MorphoPotts supprime donc un site si son volume est important et avec un volume d'autant plus important que le nombre de molécules augmente. C'est pourquoi le MorphoPotts migre dans l'environnement où le nombre de molécules est maximum et son volume et sa surface augmentent en fonction du nombre de molécules où il est présent.

Le facteur de migration λ_{migr} permet à un MorphoPotts de modifier sa vitesse de migration, plus λ_{migr} est important, plus la probabilité qu'un site soit ajouté est importante ($cf.$ l'equation 3.5 et la figure 3.2). Les simulations faites permettent de calibrer nos paramètres, en fonction de la vitesse voulue et de définir le nombre de molécules maximum que l'on peut insérer dans l'environnement pour que la déformation de la cellule en terme de volume et surface soit toujours viable.

3.1.3 Fomes cellulaires génériques et dynamiques

Nous vérifions dans cette section la mise en œuvre de la forme des cellules. Pour cela, une énergie de forme est ajoutée à l'énergie totale d'un état du système dans le CPM. Pour tester cette énergie nous décrivons synthétiquement l'intérêt de la forme des cellules puis nous abordons nos simulations et nos résultats.

Description

La forme des cellules joue un rôle essentiel dans leurs fonctions, comme dans le cas des globules rouges qui ont une forme torique pour le transport de l'oxygène et dans le cas des cellules musculaires lors de leur contraction, cela compte aussi dans la morphogenèse où la forme du tissu émerge des formes des cellules. La forme des cellules est le résultat des propriétés de sa membrane mais aussi de son cytosquelette. Le cytosquelette de la cellule lui fournit l'essentiel de ses propriétés mécaniques. Ces propriétés mécaniques sont indispensables aux déplacements des cellules. Pour vérifier l'énergie de forme qui permet aux MorphoPotts d'avoir une forme cible dynamique, nous utilisons une forme générique mais nous montrons également comment cette énergie permet un déplacement soit aléatoire de la cellule, soit dirigé par un gradient de molécules.

Modèle de simulation

Nous présentons trois simulations pour tester la forme donnée aux MorphoPotts. Premièrement, nous construisons une simulation pour imposer une forme aux MorphoPotts, puis une simulation pour reproduire les propriétés mécaniques du cytosquelette en étudiant les effets de la rotation et de la translation de la forme. Et enfin nous voyons comment ces propriétés mécaniques peuvent être influencées par les molécules se trouvant dans l'environnement. Les paramètres de ces trois simulations sont donnés dans le tableau 3.3.

Modèle de simulation 1 Ce modèle définit 4 MorphoPotts : un pour le milieu extérieur, trois autres de type r avec une forme imposée. La forme imposée est donnée dans la figure 3.1.3. L'énergie d'un état du système est donnée par $E = E_{sp}$. Les énergies de volume, de surface, de contacts et de migration ne sont pas prises en compte car cette simulation a pour but de montrer comment l'énergie de la forme E_{sp} permet d'imposer une forme. Ici les facteurs de rotation et de translation de la forme sont égaux à 0 pour que la forme du MorphoPotts ne se déplace pas dans l'environnement, le MorphoPotts restant fixe dans l'environnement.

Modèle de simulation 2 Ce modèle définit 2 MorphoPotts : un pour le milieu extérieur, un autre avec une forme imposée. La forme imposée est un cercle de rayon 7 où chaque ressort définissant cette forme

Environnement	Env : 100x100x100 et pour le modèle de simulation 3 une autre matrice $EnvA$: 100x100x100 pour l'environnement des molécules A
Agents	2 types d'Agents : r , m (pour représenter le milieu extérieur)
Paramètres du CPM	$kT = 1$, $\lambda_c = 0, \lambda_v = 0, \lambda_s = 0, \lambda_{migr} = 0 \wedge 1$
Paramètres de la fome imposée	forme : pour le premier modèle de simulation voir la figure 3.1.3, pour les autres modèles de simulation un cercle de rayon 7 dont les ressorts ont tous une extrémité ayant l'origine. le paramètre $k = 10^7$
Fonction de voisinage	Les 6 voisins directs

TABLE 3.3 – Valeur des paramètres utilisés dans les modèles de formes données aux MorphoPotts.

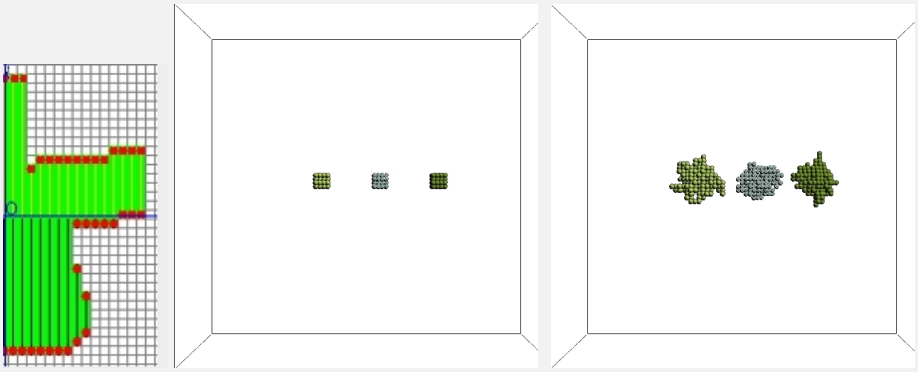
a une extrémité positionnée à l'origine de la forme. A la différence du modèle précédent, le but de ce modèle et d'étudier les facteurs de rotation Ro et de translation Tr. L'énergie d'un état du système est donc donnée par $E = E_{sp}$. Le facteur de Ro permet au MorphoPotts de tourner dans l'environnement. Il permet ainsi de donner une orientation à la cellule. Plus ce facteur est grand, plus le MorphoPotts tourne en fonction de l'ajout ou de la suppression d'un site. Le facteur de translation, quant à lui, permet au MorphoPotts de se déplacer, la forme de la cellule se déplaçant après chaque ajout de site dans la direction de ce site ou à chaque suppression dans la direction opposée à ce site . Ici nous montrons comment les cellules peuvent se déplacer aléatoirement comme c'est le cas pour les bactéries ou pour les cellules eucaryotes dans le phénomène de chimiokinèse (un mouvement aléatoire à la réponse de molécule se trouvant dans l'environnement). Dans cet simulation, nous faisons varier les facteurs de rotation et de translation au cours de la simulation pour étudier leurs effets et ainsi les calibrer.

Modèle de simulation 3 La simulation 2 montre un mouvement aléatoire de MorphoPotts, alors que cette le modèle de simulation 3 montre comment ce mouvement peut être dirigé par les molécules présentes dans l'environnement. Cette simulation étudie les facteurs de translation et de rotation en fonction des molécules présentes dans l'environnement. Ainsi nous combinons l'énergie de migration cellulaire et celle de la forme. L'énergie d'un état du système est donc donnée par $E = E_{sp} + \lambda migr E_{migr}$. Les facteurs de rotation et de translation doivent permettre à la cellule de suivre ce gradient. En effet la mise à jour de ces facteurs est influencée par l'ajout et la suppression des sites du MorphoPotts qui sont influencés par les énergies de migration. Pour construire ce modèle nous nous sommes inspirés du phénomène de phagocytose où les globules blancs poursuivent les bactéries. Notre modèle utilise un gradient de molécules dont la source (la bactérie) bouge aléatoirement et un MorphoPotts (le globule blanc) qui migre vers ce gradient.

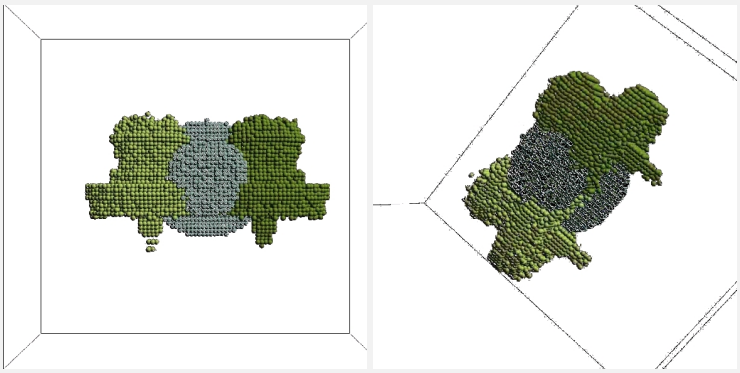
Résultat

Cette section décrit le résultat des simulations des trois modèles précédemment décrits. Les résultats de la simulation 1 vérifie le fait que le MorphoPotts peut tendre vers une forme générique, la seconde

simulation le fait que le MorphoPotts peut se déplacer aléatoirement dans l'environnement et la troisième simulation montre que le MorphoPotts, tout en gardant sa forme et son orientation, migre dans une direction donnée.



(a) forme cible (b) état initial (c) forme en construction



(d) forme construite (e) forme construite et rotation de l'environnement

FIGURE 3.3 – Exemple de forme. Les figures montrent 3 MorphoPotts ciblant une même forme imposée. Initialement le MorphoPotts du milieu est tourné de 90 degrés. La vidéo de cette simulation peut être trouvée à l'adresse suivante `http://www.youtube.com/watch?v=V07-hWux9eA`

Modèle de simulation 1 Cette simulation teste l'énergie de forme pour que le MorphoPotts cible une forme imposée. Les résultats de la simulation sont donnés dans la figure 3.1.3. La figure 3.1.3 (a) montre une coupe de la forme imposée aux MorphoPotts. La figure 3.1.3 (b) décrit l'état initial où 3 MorphoPotts sont posés dans l'environnement. Le MorphoPotts du centre est tourné de 90 degrés par rapport aux autres. Dans la figure 3.1.3 (c) les MorphoPotts sont en train de tendre vers leur forme cible. Les figures 3.1.3(d) et (e) montrent que les MorphoPotts ont atteint leur forme cible. Cette simulation valide donc notre énergie de forme.

Modèle de simulation 2 La simulation 2 montre les effets des facteurs de rotation et de translation de la forme, $i.e.$ le déplacement et la rotation de la forme cible. Le déplacement de forme va permettre le déplacement du MorphoPotts. Dans ce modèle un seul MorphoPotts est présent dans l'environnement. Trois simulations sont faites, premièrement en faisant varier au cours de la simulation le paramètre de

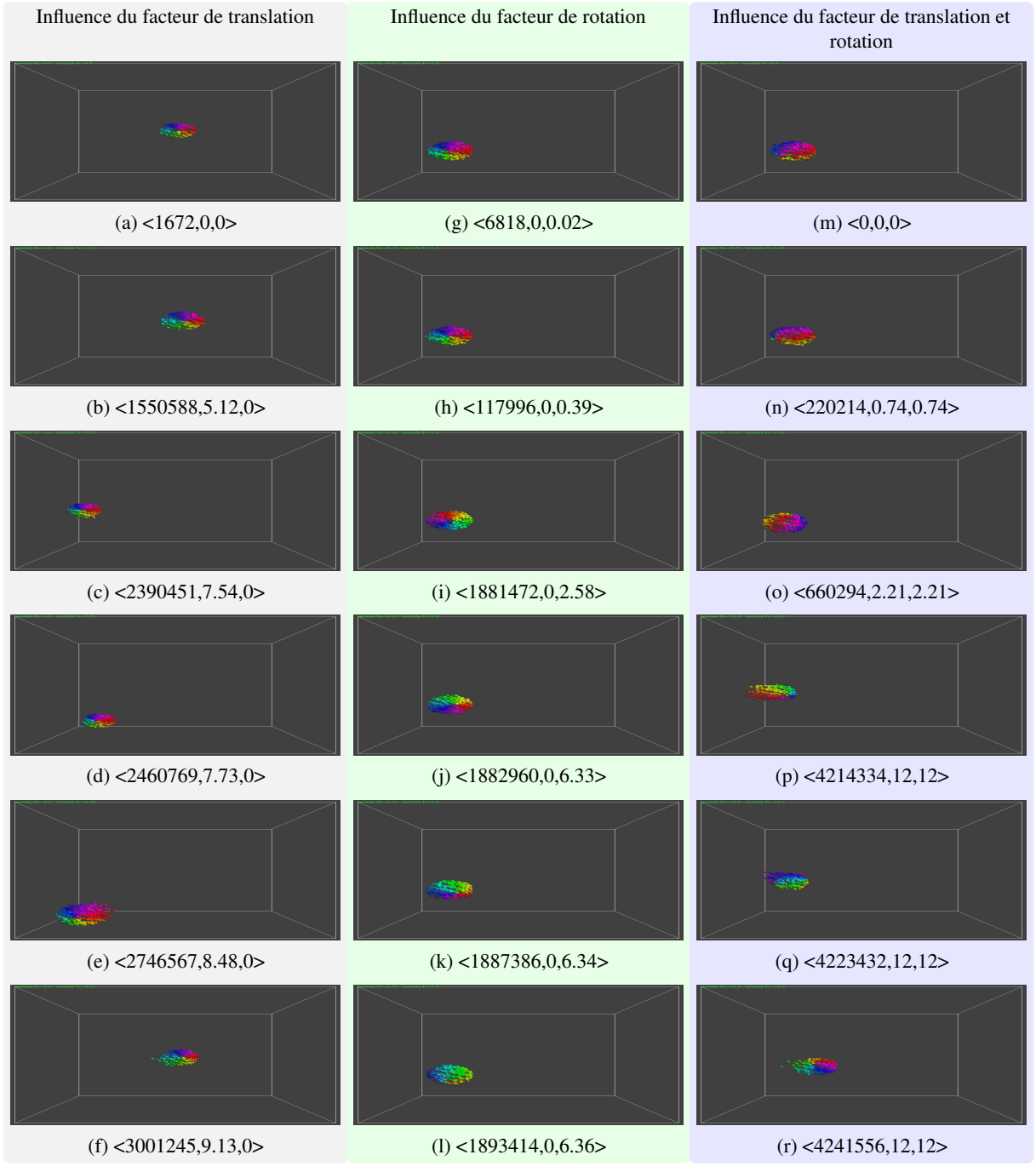


FIGURE 3.4 – Effet des facteurs de rotation et de translation de la forme imposée au MorphoPotts. Le triplet <a,b,c> signifie <nombre de pas du CPM,le facteur de translation,le facteur de rotation>. La première colonne montre l'effet du facteur de translation, la seconde le facteur de rotation, la troisième l'effet des facteurs combinés. La vidéo de ces simulations peut être trouvée à l'adresse suivante `http://www.youtube.com/watch?v=CPciJavb4mg`

translation, secondement en faisant varier le paramètre de rotation et finalement en faisant varier les deux paramètres ensemble. Le résultat de ces simulations est montré dans la figure 3.4. Le rendu du MorphoPotts

est fait de telle manière qu'une couleur correspond à une coupe de sa forme. Ce rendu permet d'étudier la rotation du MorphoPotts. Les figure 3.4(a), (b), (c), (d), (e) et (f) montrent les effets de la translation. La translation de la forme implique un déplacement du MorphoPotts perceptible pour un facteur de translation supérieur à 3. Plus le facteur de translation est important, plus le MorphoPotts se déplace vers les sites ajoutés ou dans le sens inverse d'un site supprimé. Pour un facteur de translation supérieur à 6, le MorphoPotts se déplace aléatoirement. L'effet de la translation étant trop important, chaque ajout d'un site change la direction de la forme de façon radicale, lui permettant ainsi ce déplacement aléatoire. Nous pouvons remarquer que l'orientation de la cellule ne change pas car le rendu graphique du MorphoPotts garde le même dégradé de couleur. Les figures 3.4(g), (h), (i), (j), (k) et (l) montrent l'effet du facteur de rotation. A chaque site ajouté le facteur de rotation permet au MorphoPotts de tourner en direction de ce site. Ceci permet au MorphoPotts de sonder l'environnement et de s'orienter en fonction. La simulation montre que pour un facteur de rotation supérieur à 1, la rotation du MorphoPotts est perceptible. Plus le facteur est important, plus la rotation du MorphoPotts devient aléatoire. Les figures 3.4(m), (n), (o), (p), (q) et (r) montrent l'effet combiné des facteurs de rotation et de translation. Cette simulation garde les propriétés des dernières simulations et permet de modéliser le déplacement aléatoire d'un MorphoPotts et de changer de façon aléatoire son orientation. Le modèle de simulation suivant teste ces deux facteurs pour montrer comment le MorphoPotts peut suivre un champ de molécules tout en gardant une orientation et une forme cohérente.

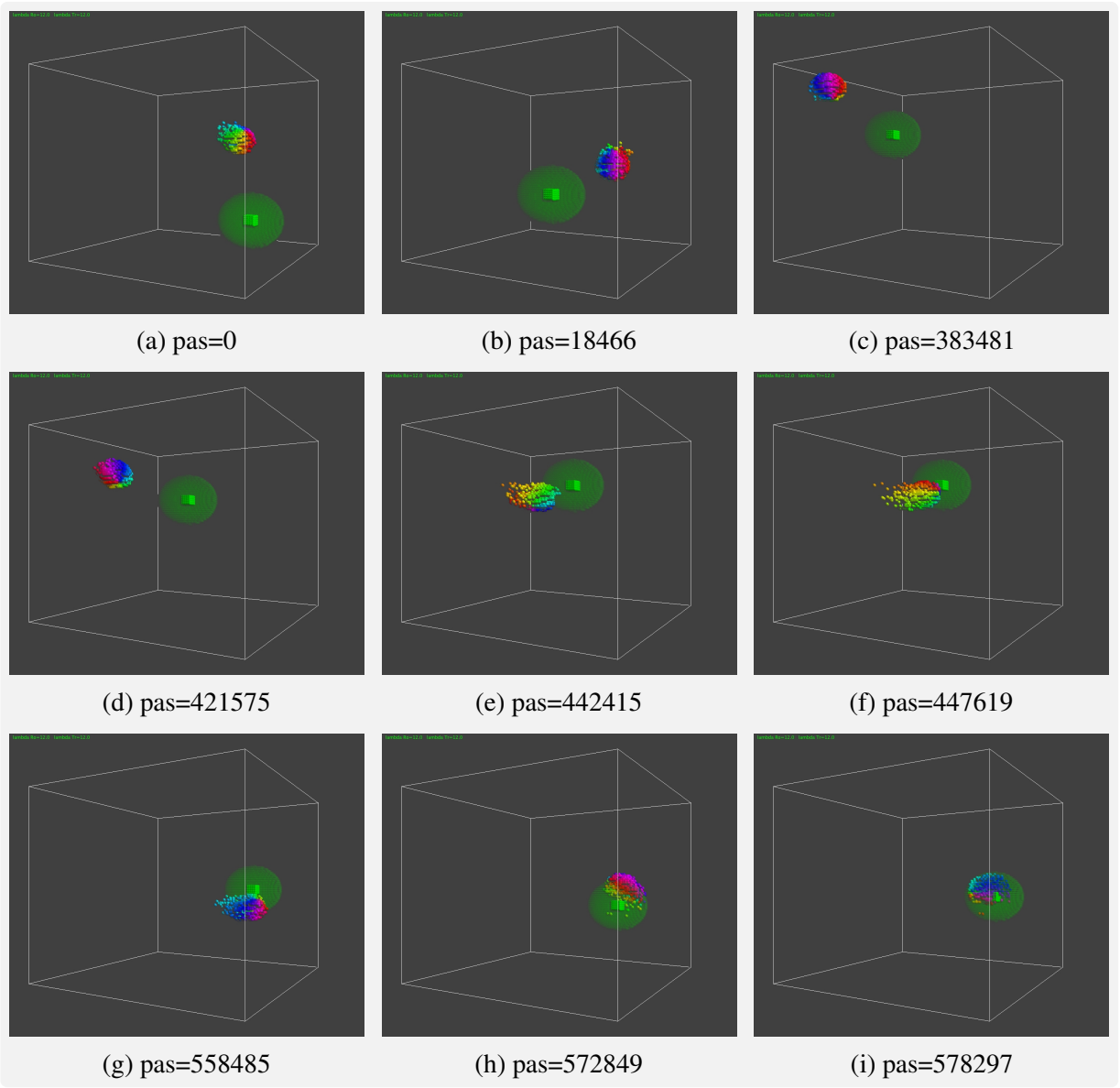
Modèle de simulation 3 La rotation et la translation de la cellule sont influencées par les sites ajoutés et supprimés. Comme cela est décrit dans la section 3.1.2, l'énergie de migration permet de favoriser l'ajout des sites à un MorphoPotts dans la direction de ce gradient. Cette simulation vérifie que la rotation et la translation permet au MorphoPotts de migrer vers ce gradient qui se déplace aléatoirement. La figure 3.5 montre le résultat de la simulation. Le MorphoPotts ne se déplace plus aléatoirement, mais se déplace en fonction du gradient de molécules. Le MorphoPotts « chasse » la source du gradient (la bactérie). Cette simulation valide l'énergie de forme et les facteurs de translation et rotation. Toutes ces simulations ont permis de vérifier notre mise en œuvre des énergies du MorphoPotts définies dans le formalisme du CPM. Dans la section suivante nous appliquons nos MorphoPotts pour déterminer des interactions permettant l'émergence d'un tissu.

3.2 Auto-Organisation dans la Morphogenèse

La morphogenèse entre les cellules notamment dans le phénomène de l'embryogenèse reste encore mal compris. Nous décrivons la théorie biologique que nous utilisons pour comprendre ce phénomène et nous vérifions si elle est modélisable dans notre système (3.2.1). Nous étudions entre autre l'auto-organisation entre les cellules qui permet notamment la formation d'un tissu cellulaire cohérent, $i.e.$ avec une forme reconnaissable et avec un nombre relativement stable de cellules par tissu. Pour cela nous étudions la formation d'un tissu (3.2.2), puis l'auto-organisation entre les cellules de différents types pour former divers tissus (3.2.3).

3.2.1 Description de la théorie utilisée.

Traditionnellement, les étapes initiales de l'embryogenèse sont expliquées par l'hypothèse où la cellule œuf possède un gradient de molécules qui permet lors de sa division successive, la génération de cellules de différents types qui communiquent entre elles via des signaux émanant d'un programme génétique. Ces signaux permettraient de structurer la prolifération cellulaire pour autoriser la formation de l'embryon et la mise en place des différents organes. L'embryogenèse serait donc régie par un processus



(a) pas=0 (b) pas=18466 (c) pas=383481

(d) pas=421575 (e) pas=442415 (f) pas=447619

(g) pas=558485 (h) pas=572849 (i) pas=578297

FIGURE 3.5 – Migration du MorphoPotts en fonction d'un gradient de molécules qui se déplace aléatoirement. Ces figures montrent comment le MorphoPotts suit un gradient (représenté en vert). Cela peut être vu comme une analogie au globule blanc qui « chasse » une bactérie. La vidéo de cette simulation peut être vue à l'adresse suivante : `http://www.youtube.com/watch?v=IPLBTaYAasg&feature=related`

déterministe codé par l'ADN. La théorie que nous modélisons utilise le principe de l'auto-organisation et définit l'embryogenèse comme un procédé stochastique. Cette théorie est le darwinisme cellulaire [46]. Elle est basée sur le fait que la différenciation cellulaire est stochastique et que la sélection naturelle amène à la formation de tissus cohérents. Ceci est possible car les cellules sont inter-dépendantes. Les cellules produisent des molécules dont d'autres cellules ont besoin. Dans nos modèles, Les cellules ne perçoivent pas directement les autres cellules mais sont sensibles aux traces laissées par les autres cellules. Si une cellule est dans un environnement qui lui fournit les molécules dont elle a besoin, elle se divise et prolifère sinon elle meurt.
Les sections suivantes présentent la formation d'un tissu à partir de cette théorie et la formation de

Environnement	Env : 100x100x100 (pour les agents) et $EnvA$: 100x100x100 (pour les molécules)
Agents	3 types d'Agents : 2,1, 0 (pour représenter le milieu extérieur)
Type 2	$V_t = 350, S_t = 350$
Paramètres du CPM	les énergies de contact sont données dans [36] : $T_{0,1} = T_{1,1} = T_{1,2} = 0, T_{0,2} = 100$,$T_{2,2} = -10000$, $\lambda_c = 1\lambda_v = 1, \lambda_s = 1$
Paramètres de la forme	la forme cible est donnée dans la figure 3.7 et le paramètre $k = 10^7$
Fonction de voisinage	Les 6 voisins directs

TABLE 3.4 – Valeur des paramètres utilisés dans le modèle de la formation d'un tissu cellulaire.

plusieurs tissus dont les interactions forment un motif dynamique et complexe.

3.2.2 Organisation des cellules dans un tissu

Cette section décrit le modèle de simulation pour la formation d'un tissu. Nous en discutons ses paramètres et nous en décrivons une simulation.

Description

Pour que les cellules forment un tissu cellulaire, implique que ces cellules aient une affinité entre elles par rapport au milieu extérieur. Nous testons ici l'hypothèse que la forme des cellules et les énergies de contacts peuvent fortement structurer un tissu cellulaire. Selon la théorie biologique définie dans la section précédente, la compétition entre MorphoPotts pour consommer les molécules se trouvant dans l'environnement, permet une croissance finie du tissu cellulaire comme décrit en [47] et la régénération dynamique du tissu. Pour montrer l'intérêt et les propriétés de la rotation et de la translation de la forme mais aussi du bilan énergétique défini dans le MorphoPotts, la section suivante présente un modèle simulant la vie d'un tissu cellulaire.

Modèle de simulation

Les paramètres et les valeurs des paramètres du CPM sont donnés dans le tableau 3.4 et ceux du MorphoPotts dans la figure 3.6.

Ce modèle consiste en trois types de MorphoPotts :

- un type 0 qui modélise le milieu extérieur.
- un type 1 qui produit des molécules dans le milieu.
- un type 2 qui consomme les molécules produites par le second type et se divise. Ce type a une forme qui influence la construction du tissu.

Les MorphoPotts sont :

- en interaction directe. Une énergie de contacts négative (c'est à dire que si les MorphoPotts de type 2 s'agrège, l'énergie du système diminue) est mise entre les MorphoPotts de type 2. Une énergie de

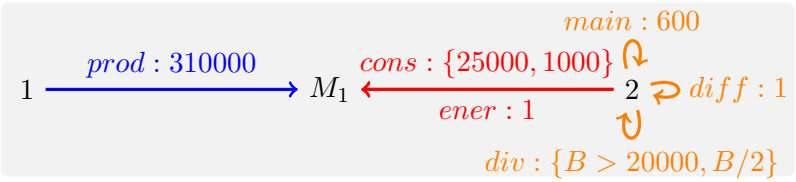


FIGURE 3.6 – Graphe d'interactions de 2 MorphoPotts dans la formation d'un tissu. Ce graphe d'interactions donne aussi les comportements des MorphoPotts. Deux types de MorphoPotts sont définis. L'un (type 1) a la capacité de produire des molécules nécessaires à la survie de l'autre (type 2). Si le MorphoPotts de type 2 a suffisamment d'énergie (ici $2 * 10^4$) B (chaque molécule consommée augmente B de 1), il peut se diviser et partager son énergie avec le nouveau MorphoPotts créé.

contacts positive est mise entre les MorphoPotts de type 2 et 0, c'est à dire que si les MorphoPotts de type 2 se séparent, l'énergie du système augmente : ils ont donc tendance à rester groupés.

– en interaction indirecte. Les MorphoPotts de type 1 fournissent des molécules au MorphoPotts de type 2. Si les MorphoPotts de type 2 ne trouvent pas de molécule ils meurent rapidement.

Les énergies de contacts vérifient que $4 * T_{1,3} + T_{3,3} < 0$. Cette équation implique que lorsque les MorphoPotts de type 2 sont en contact grâce à l'ajout d'un site alors $\Delta E_c < 0$. Cet ajout du site est favorisé par les énergies de contacts.
La concentration des molécules produites par le MorphoPotts de type 1 diminue avec la distance à ce MorphoPotts source du fait de la diffusion cellulaire. Si les MorphoPotts de type 2 sont éloignés (à une distance supérieure à 52) des MorphoPotts de type 1, ils n'ont pas assez de molécules pour survivre car ils dépensent plus d'énergie qu'ils n'en consomment.
Le MorphoPotts de type 2 peut se diviser si son énergie est supérieure à 20000 (valeur calibrée de manière empirique).

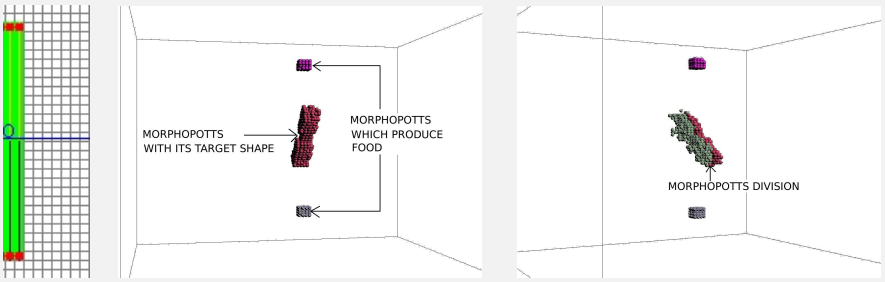
La forme décrite dans la figure 3.7(a) est donnée aux MorphoPotts de type 2. Le volume et la surface de la forme donnée au MorphoPotts sont chacun égaux à 328,64. Donc le volume cible (350) et la surface cible (350) peuvent remplir la forme. 21 sites supplémentaires doivent être ajoutés aux MorphoPotts de type 2 pour vérifier le volume et la surface cible. Les ressorts ($cf.$ la Figure 3.7(a)) sont définis sur un axe horizontal, avec le paramètre $k = 10^7$ pour forcer le MorphoPotts à atteindre la forme voulue : l'énergie de forme est comparable aux autres énergies.

Le facteur λ_c (resp. λ_v, λ_s) est égal à 1 ($resp.$ 10^5, 10^5). Ces valeurs permettent aux MorphoPotts de ne pas dépasser leur volume cible. Les énergies de contacts permettent d'influencer le mouvement des MorphoPotts quand leur forme, volume et surface sont atteints. Le paramètre kt du CPM est égal à 1, donc la probabilité de transition est égale à $e^{-\Delta E}$. Les transitions avec $\Delta E > 0$ ont donc une chance faible d'être acceptées.

Des ressorts de longueur nulle placés selon (Ox) à gauche et à droite de la forme, évitant la croissance de la cellule le long de cet axe. Le facteur Ro (resp. Tr) de la rotation (resp. la translation) est de 10 (resp. de 75). La rotation et la translation ne sont possibles que sur l'axe Oz car on modélise la construction d'un tissu cellulaire selon une seule direction. Les facteurs de rotation et de translation ont été calibrés par dichotomie après plusieurs simulations.

Résultat

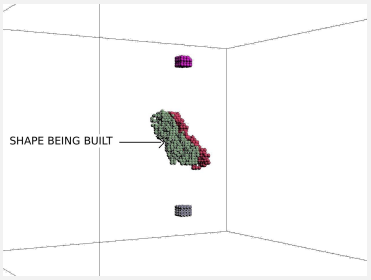
Dans cette section nous présentons une simulation du modèle décrit précédemment pour tester à la fois la translation et la rotation d'une forme, ainsi que la construction d'un tissu cohérent, $i.e.$ une forme globale identifiable et un maintien dynamique du tissu. La figure 3.7 montre les résultats de la simulation.



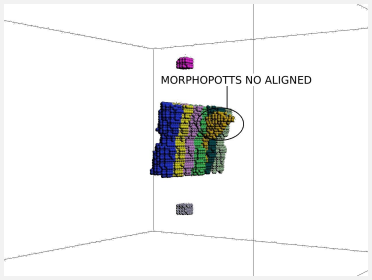
(a) Forme cible

(b) État initial, pas=0, t=0s

(c) Division du MorphoPotts, pas=217058, t=1min30

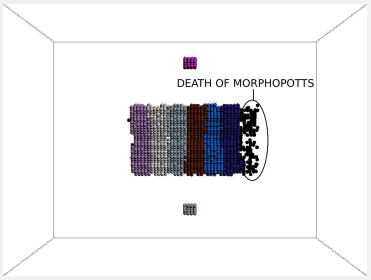


(d) Forme en construction, pas=220474, t=1min32



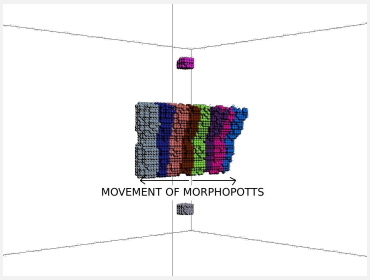
(e) Effet de la rotation (1), pas=4057167, t=20min

(f) Effet de la rotation (2), pas=4626796, t=21min30



(g) Mort du MorphoPotts, pas=7316104, t=35min

(h) Division du MorphoPotts, pas=9178409, t=45min



(i) Déplacement MorphoPotts *via* la translation de la forme après une division, pas=9244846, t=45min20

FIGURE 3.7 – Formation d'un tissu cellulaire simple auto-entretenu. Cette simulation montre comment la forme de la cellule peut structurer et maintenir le tissu cellulaire depuis le commencement de sa croissance et durant sa vie. L'état initial est une cellule œuf qui se divise lorsqu'elle a suffisamment consommé de nourriture. Les cellules filles sont de même type et héritent de la forme de la cellule mère. Les énergies de contacts permettent aux cellules de s'auto-organiser pour faire émerger un tissu où la forme reste constante malgré les morts et divisions cellulaires car les cellules restes alignées. La vidéo de cette simulation peut être vue à l'adresse suivante `http://www.youtube.com/watch?v=GTWG0MxuF8g`

L'état initial (*cf.* la figure 3.7(b)) consiste en un MorphoPotts de type 2 en train de tendre vers sa forme. Les figures 3.7(b) et 3.7(c) montrent le MorphoPotts de type 2 consommer assez de molécules pour permettre sa division. La figure 3.7(d) met en évidence un MorphoPotts de type 2 en train de se construire. Dans le même temps, deux MorphoPotts de type 2 s'auto-alignent grâce aux énergies de contacts. Nous observons dans les figures 3.7(e) et 3.7(f) les effets de la rotation de la forme. Un MorphoPotts n'est pas aligné avec les autres, mais comme les énergies de contacts favorisent l'ajout des sites qui sont en contacts avec d'autres MorphoPotts, le MorphoPotts s'aligne enfin avec les autres. Dans la figure 3.7(g), le

Environnement	Env : 1000x1000
Agents	5 types d'Agents : 5,4,3,2,1 (cellule souche), 0 (pour représenter le milieu extérieur)
Type $1, 2, 3, 4, 5$	$V_t = 49, S_t = 28$
Type 2	$V_t = 350, S_t = 350$
Paramètres du CPM	$T_{x,x} = 0, T_{x,y} = 10, x \neq y \in [0, 5]$ $\lambda_c = 1, \lambda_v = 1, \lambda_s = 1, \lambda_{migr} = 1, kT = 2$
Fonction de voisinage	Les 4 voisins directs

TABLE 3.5 – Valeur des paramètres du CPM dans le modèle de l'embryogenèse. Ces paramètres sont utilisés dans les trois simulations avec kT=2. Les énergies des volumes et surfaces permettent aux MorphoPotts de former un carré de largeur 7. Les énergies de contacts sont telles que deux cellules de même types ont une énergie moindre que deux cellules des types différents.

MorphoPotts du côté droit de la figure est trop loin (une distance supérieure à 52), et son énergie devient donc nulle et il meurt. Ceci permet de garder naturellement une largeur finie pour le tissu cellulaire. Les figures 3.7(e) et 3.7(f) montrent les effets de la translation de la forme. Après une division d'un MorphoPotts dans le centre du tissu, le MorphoPotts est comprimé. Ceci implique une translation des autres MorphoPotts vers l'extérieur du tissu.

La rotation de la forme et l'énergie de contact permettent donc l'auto-alignement du MorphoPotts. La translation de la forme et la compétition entre les MorphoPotts permettent une croissance finie du tissu cellulaire. Pendant la simulation, les MorphoPotts se divisent au centre du tissu, se déplacent vers les extérieurs et meurent aux extrémités du tissu. La forme du tissu émerge aussi des formes des MorphoPotts. Cette simulation montre donc une auto-organisation des MorphoPotts pour former un tissu. Comme dans ce modèle les formes des MorphoPotts ne peuvent subir de rotation ou de translation que selon l'axe Oz, l'auto-organisation du tissu s'explique directement en terme d'énergie : plus les MorphoPotts sont alignés, plus les ils ont une zone de contact large et donc plus les énergies de contacts baissent.

3.2.3 Organisation des cellules dans plusieurs tissus

La simulation précédente montre comment il est possible de construire un tissu cohérent avec les MorphoPotts. Les simulations décrites dans cette section utilisent le comportement de différentiation et montrent comment les tissus entre eux peuvent interagir et donner des motifs différents et dynamiques en fonction des interactions. Nous décrivons nos modèles dans le cadre de la théorie du darwinisme cellulaire. Nous discutons des paramètres et étudions l'auto-organisation observée lors des simulations faites.

Description

Cette section décrit un modèle qui simule une théorie darwinienne au niveau cellulaire [46]. Cette théorie est basée sur le fait que la différenciation cellulaire est stochastique et que la sélection naturelle joue un rôle clé. Le modèle présenté ici est proche de celui défini dans [47] : deux types de cellules sont définies et (chacune représentée par un pixel) produisent et consomment des molécules dans un environnement représenté par une matrice de dimension 50x50. A chaque instant les cellules peuvent se différencier (selon deux types) avec une probabilité qui est dépendante du milieu. Les cellules peuvent se diviser si la quantité de molécules consommées est suffisante. Le premier type produit une molécule qui

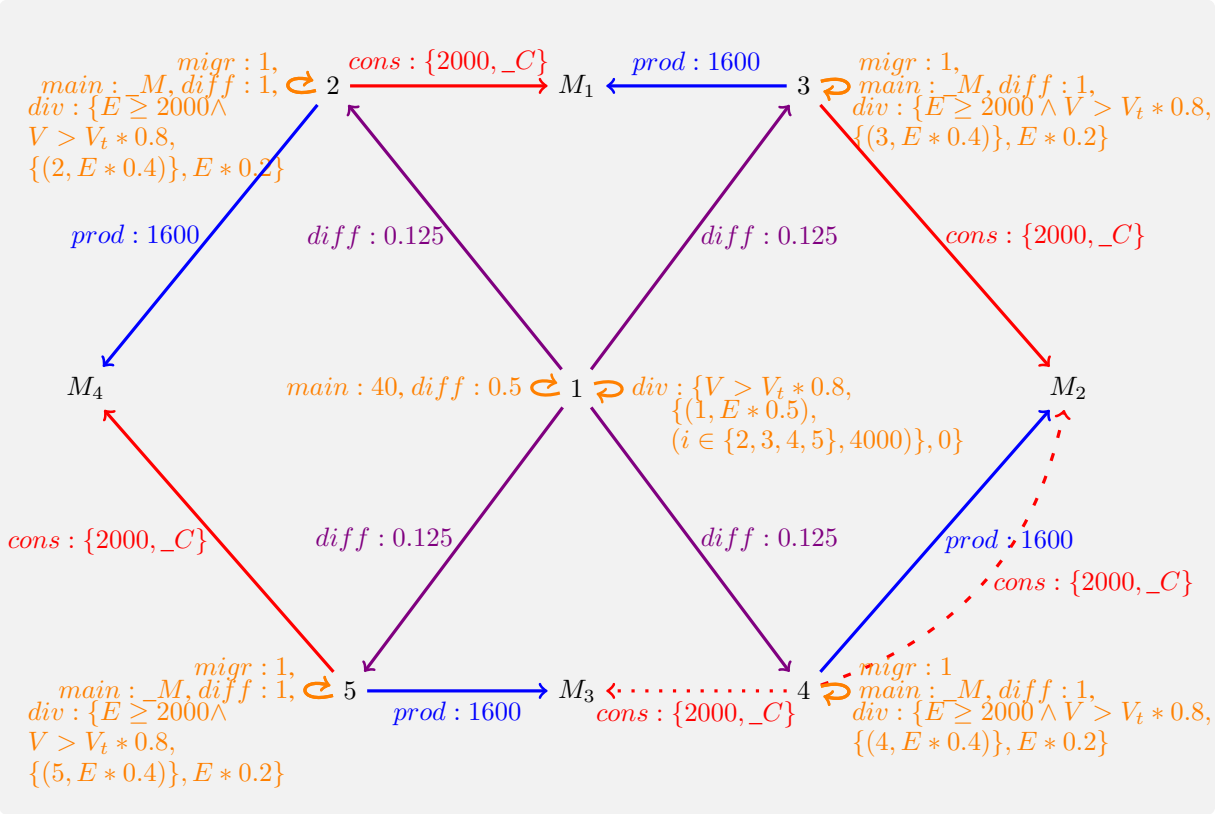


FIGURE 3.8 – Graphe d'interactions utilisé dans les trois modèles de simulations de l'embryogenèse. Dans les simulations 1 et 2 les flèches en pointillés sont effectives, mais pas les flèches hachées. Dans la simulation 3 c'est l'inverse. Dans la simulation 1 nous avons $\_C = 10 \wedge \_M = 400$, et dans les simulations 2 et 3 nous avons $\_C = 30 \wedge \_M = 150$. Dans cette figure, chaque MorphoPotts (excepté le type 1) produit un gradient de molécules dont le nombre de molécules produites au centre est égale à 1600. Si l'énergie de ces MorphoPotts est inférieure à 2000, le MorphoPotts consomme les molécules, $i.e.$ l'effet inverse de la production avec une consommation au centre égale à $\_M$ au maximum. Ces MorphoPotts peuvent aussi se diviser si leur volume courant est supérieur à 80% du volume cible et leur énergie est supérieure à 2000. Après la division, l'énergie du nouveau MorphoPotts et celle de l'ancien MorphoPotts est égale à 40% de l'énergie de l'ancien MorphoPotts. Grâce aux propriétés de migration, les MorphoPotts de même type se groupent pour former un tissu. On peut observer que la dépendance des molécules des MorphoPotts forme un cycle dans les simulations 1 et 2. Ce cycle est brisé dans la simulation 3.

est consommée par le second type, et vice-versa. Cette inter-dépendance, selon les valeurs des paramètres, amène soit à une croissance finie du tissue, soit à une croissance rappelant la prolifération cellulaire lors d'un cancer.

Nos simulations utilisent ces notions (voir figure 3) en modélisant :

– La différentiation cellulaire stochastique par un MorphoPotts de type 1 qui se divise et peut se différencier (avec une probabilité de 0.5) en 4 types de cellules ayant la même probabilité. Ce MorphoPotts ne produit pas et ne consomme pas d'énergie. Par contre, il est initialisé avec assez d'énergie pour se diviser, pour fournir de l'énergie aux nouvelles cellules et pour imposer dans l'environnement une différenciation cellulaire stochastique. Ce MorphoPotts modélise la cellule

œuf qui peut se diviser et qui a suffisamment d'énergie pour amorcer les premières divisions.
– La sélection naturelle est modélisée par la notion de "nourriture". Si un MorphoPotts trouve de la "nourriture", il peut proliférer et se diviser, sinon il meurt.

Pour que la théorie du darwinisme cellulaire soit vérifiée dans ce modèle, une auto-organisation doit être possible entre les cellules pour former à la fois des tissus cohérents et une organisation entre les tissus qui sont robustes, c'est à dire qui soit globalement les mêmes au cours des différentes simulations. L'embryogenèse en conditions normales amène toujours à la croissance finie des tissus et de façon organisée.

Modèle de simulation

Trois modèles sont proposés pour permettre d'étudier les effets des paramètres et des interactions sur les motifs tissulaires obtenus. Les paramètres du CPM sont donnés dans le tableau 3.5 et dans la figure 3.8.

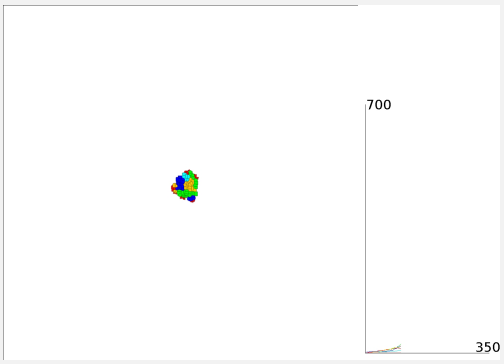
Résultat

Dans cette section nous présentons trois simulations qui partagent les mêmes paramètres du CPM décrit dans le tableau 3.5, mais ils sont différents soit par leurs valeurs, soit par leurs interactions ($cf.$ la figure 3.8). Dans ces simulations, le MorphoPotts du type 1 a la couleur rouge, le MorphoPotts du type 2 a la couleur vert, le MorphoPotts du type 3 a la couleur bleu, le MorphoPotts du type 4 a la couleur jaune, le MorphoPotts du type 5 a la couleur cyan. Une prolifération en spirale émerge dans la première simulation. En changeant la valeur de 2 paramètres, une croissance finie du tissu émerge dans la seconde simulation. Finalement, en modifiant une interaction, une prolifération multi-couches apparaît dans la dernière simulation.

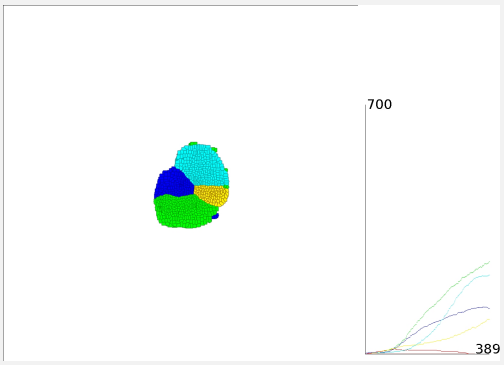
Simulation 1 : prolifération en spirale Dans cette simulation, quatre types de cellules produisent et consomment des molécules. L'inter-dépendance entre les cellules en termes de « nourriture » forme un cycle ($cf.$ la figure 3.8). On considère que la formation du tissu signifie que chaque cellule de même type forme un groupe via les propriétés de migration. L'état initial est une matrice 1000x1000 où un MorphoPotts de type 1 est placé au niveau du site $(500, 500)$ et forme un carré de largeur 7.

Les résultats de la simulation sont donnés dans la figure 3.9. On peut observer 3 étapes dans cette morphogenèse. La première étape ($cf.$ la figure 3.9(a) et 3.9(b)) est la différenciation cellulaire et la sélection naturelle. Le MorphoPotts de type 1 se divise et se différencie aléatoirement dans 4 autres types. Ceci amène à la formation d'un tissu ($cf.$ la figure 3.9(c)). La deuxième étape est le tri ($cf.$ les figures 3.9(c) et 3.9(d)). Le tissu est trié par le simple fait de la mort (les cellules ne trouvent pas de nourriture) et la division (les cellules trouvent de la nourriture). La troisième étape est la prolifération et l'émergence d'un motif ($cf.$ les figures 3.9(d) et 3.9(e)). Ici une prolifération en spirale émerge. Ceci est surprenant dans le sens que ce motif n'est pas imposé dans la description du modèle. Les MorphoPotts de type 2 ($resp.$ 3,4,5) se divisent le long des MorphoPotts de type 3 ($resp.$ 4,5,2). On peut observer que le renouvellement du tissu est continu, $i.e.$ que le tissu n'est pas statique. Sur la largeur de la spirale, la cellule migre de l'intérieur vers l'extérieur ($cf.$ la figure 3.9(f)). Un critère de l'embryogenèse est l'arrêt de la croissance du tissu cellulaire, $i.e.$ que le nombre de cellules est constant. Ce nombre est constant non pas par le fait que les cellules ne se divisent plus, mais par le fait que le ratio de la division cellulaire sur la mort de la cellule est égal à 1. Ici la simulation semble infinie. La division cellulaire surpasse la mort cellulaire. Le but de la simulation suivante est de corriger ce problème.

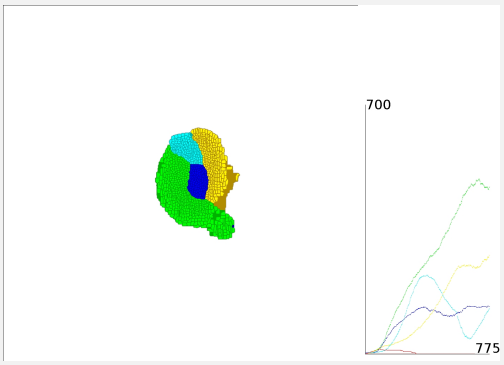
(a) temps de simulation=0, nombre des pas CPM=0



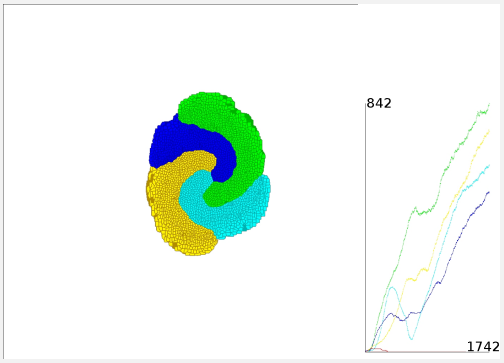
(b) temps de simulation=1min 42sec, nombre des pas CPM=121906



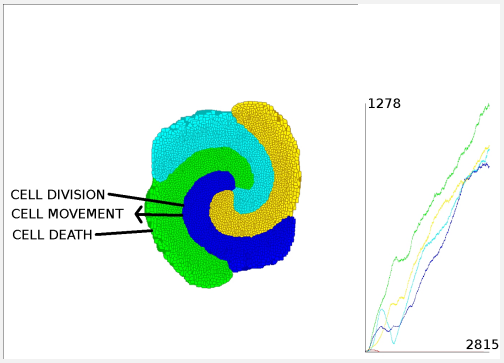
(c) temps de simulation= 36min, nombre des pas CPM=4402498



(d) temps de simulation= 2hours 4min, nombre des pas CPM=16 259 458



(e) temps de simulation= 11hours 10min, nombre des pas CPM=77 660 708



(f) temps de simulation= 30hours 10min, nombre des pas CPM=198 853 366

FIGURE 3.9 – Prolifération en spirale. Cette figure montre les résultats de la simulation 1. La figure 3.9(a) représente l'état initial. La figure 3.9(b) montre la différenciation cellulaire et la sélection naturelle. La figure 3.9(c) montre le tissu formé. La figure 3.9(d) montre le tri cellulaire du tissu. Les figures 3.9(e) et 3.9(f) montrent la prolifération en spirale. Dans ces figures les cellules qui ont la couleur plus foncée meurent. Les courbes de droite représentent le nombre de cellules en fonction du nombre de pas de simulation. Ces courbes suggèrent une prolifération infinie. La vidéo de cette simulation peut être vue à l'adresse suivante `http://www.youtube.com/watch?v=ugP07j7gxPY`

Simulation 2 : la croissance finie du tissu Cette simulation est différente de celle précédente car la division cellulaire est moins favorisée. Le nombre maximum de molécules pouvant être consommés par les cellules est réduit et le coût de maintenance est augmenté ($cf.$ la figure 3.8). Cette simulation a pour but de montrer une croissance finie du tissu qui est un critère de l'embryogenèse. Ceci est possible si le ratio de la division cellulaire sur la mort cellulaire est égal à 1.

Les résultats de la simulations sont donnés dans la figure 3.10. On retrouve les deux premières étapes décrites dans la simulation 1 : la différenciation de la cellule et la sélection naturelle pour former et trier les tissus ($cf.$ les figures 3.10(a), 3.10(b), et 3.10(c)). Après le tri cellulaire, nous observons une croissance finie. Le nombre de cellules varie entre deux seuils ($cf.$ les figures 3.10(e) et 3.10(f)). Donc un équilibre entre la mort cellulaire et la division cellulaire émerge. Nous remarquons aussi l'émergence d'un mouvement induit par les déplacements des tissus, où le motif émergeant tourne sur lui-même ($cf.$ les figures 3.10(d), 3.10(e) et 3.10(f), trois tours complets sont faits). Comme dans la simulation antérieure, le renouvellement du tissu est continu.

Simulation 3 : prolifération multi-couches Dans cette simulation nous remplaçons une interaction par une autre, en supprimant le cycle d'inter-dépendance ($cf.$ la figure 3.8). Les MorphoPotts de type 4 et les MorphoPotts de type 3 sont en inter-dépendance. Cette inter-dépendance est similaire à la simulation définie en [47].

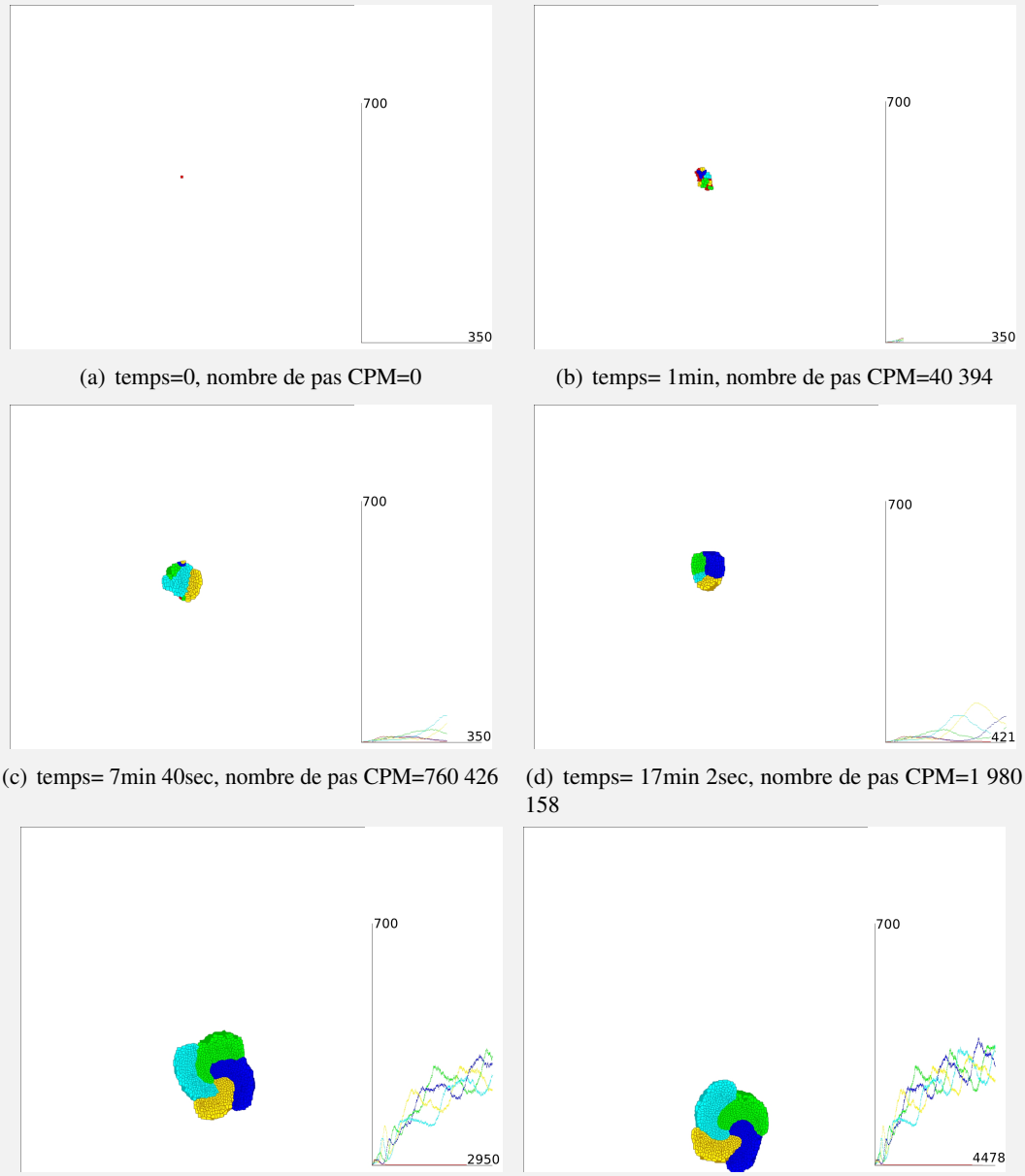
Le résultat de la simulation est donné dans la figure 3.11. Nous retrouvons les deux premières étapes décrites dans la simulation antérieure : la différenciation cellulaire et la sélection naturelle pour former et trier un tissu ($cf.$ les figures 3.11(a), 3.11(b), et 3.11(c)). Après le tri cellulaire nous observons ($cf.$ les figures 3.11(d), 3.11(e) et 3.11(f)) la formation d'une bi-couche comme décrite en [47] entre les MorphoPotts du type 3 et ceux du type 4. Nous observons aussi la formation d'une multi-couche parce que d'une part les MorphoPotts de type 1 ($resp.$ 4) prolifèrent selon le tissu formé composé des MorphoPotts de type 2 ($resp.$ 1) car d'autre part les MorphoPotts de type 2 ($resp.$ 1) produisent la « nourriture » pour les MorphoPotts de type 1 ($resp.$ 4).

On distingue deux comportements des tissus pour les 2 types de MorphoPotts en inter-dépendance :

1. Dans le cas où leurs zones de division cellulaire sont en contact, nous avons une prolifération en bi-couche.
2. Dans le cas où la zone de division cellulaire d'un type de MorphoPotts est sur les extrémités du tissu et la mort cellulaire au centre, nous avons une disjonction ou un branchement du tissu.

Ces simulations montrent le potentiel de la programmation par les MorphoPotts. Le changement d'une interaction ou la valeur d'un paramètre modifie le motif de la prolifération. L'expressivité fournie par les MorphoPotts permet de générer un grand nombre de motifs de prolifération. Nous pouvons remarquer que les motifs obtenus peuvent être retrouvés dans la nature même s'il n'est pas évident que les mécanismes soient les mêmes que ceux observés en réalité. C'est par exemple le cas pour la spirale dans la coquille des escargots, la fleur de tournesol ou la pomme de pin, le branchement des tissus dans les poumons ou le cerveau. La théorie du darwinisme au niveau cellulaire a pu être utilisée avec notre système. La description stochastique qu'elle fait des mécanismes cellulaires permet, dans ces exemples, de structurer les tissus et de faire émerger des motifs robustes. Les différentes simulations réalisées avec les mêmes paramètres amènent presque toujours au mêmes motifs.

Les motifs obtenus étant très différents et si on change les valeurs ou les relations entre MorphoPotts, il n'est pas possible pour le moment de déterminer automatiquement la forme que prend un tissu cellulaire. Le fait que le motif soit dynamique et bouge dans l'espace et que les MorphoPotts ne sont pas seulement définis dans le CPM, l'auto-organisation n'est pas corrélée, à l'inverse de ce que nous avons vu précédemment, avec l'énergie du système. Seul le regard extérieur de l'utilisateur permet ici de détecter l'auto-organisation.



(a) temps=0, nombre de pas CPM=0

(b) temps= 1min, nombre de pas CPM=40 394

(c) temps= 7min 40sec, nombre de pas CPM=760 426

(d) temps= 17min 2sec, nombre de pas CPM=1 980 158

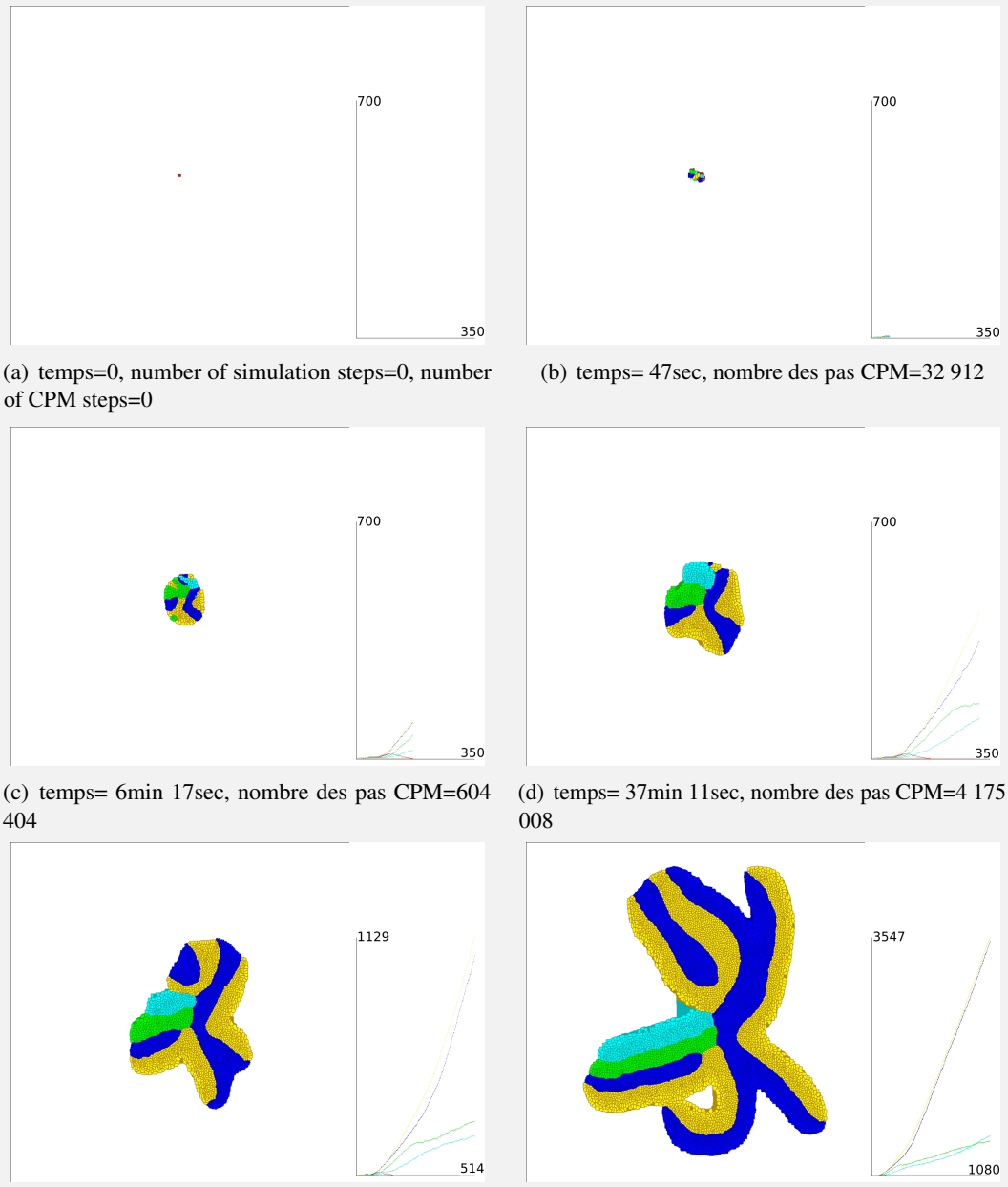
(e) temps= 8heures 43min, nombre de pas CPM=66 218 114

(f) temps= 15heures 23min, nombre de pas CPM=115 787 724

FIGURE 3.10 – Cette figure montre les résultats de la simulation 2. La figure 3.10(b) montre la différenciation cellulaire et la sélection naturelle. La figure 3.10(c) montre le tissu formé. Les figures 3.10(c) et 3.10(d) montrent le tri cellulaire du tissu. Les figures 3.10(d), 3.10(e) et 3.10(f) montrent la croissance finie du tissu et la rotation du motif (trois tours complets sont faits). Dans ces figures les cellules qui ont la couleur plus foncée meurent. Les courbes de droite représentent le nombre de cellules en fonction du nombre de pas de simulation. Ces graphiques montrent une prolifération finie. La vidéo de cette simulation peut être vue à l'adresse suivante `http://www.youtube.com/watch?v=PUTDQSlXJi0`

Le temps de simulation devient trop important si nous voulons augmenter le nombre d'agents ou le nombre de pas de simulations. Comme le MorphoPotts permet de faire émerger des motifs intéressants, il

est indispensable d'améliorer les performances pour pouvoir les énumérer. Les chapitres suivants décrivent la mise en œuvre des MorphoPotts sur les cartes graphiques qui permettent une exécution en parallèle des MorphoPotts.



(a) temps=0, number of simulation steps=0, number of CPM steps=0

(b) temps= 47sec, nombre des pas CPM=32 912

(c) temps= 6min 17sec, nombre des pas CPM=604 404

(d) temps= 37min 11sec, nombre des pas CPM=4 175 008

(e) temps= 2heures 22min, nombre des pas CPM=15 897 324

(f) temps= 20heures 48min, nombre des pas CPM=113 794 950

FIGURE 3.11 – Cette figure montre les résultats de la simulation 3. La figure 3.11(a) représente l'état initial. La figure 3.11(b) montre la différenciation cellulaire et la sélection naturelle. La figure 3.11(c) montre le tissu formé. Les figures 3.11(d), 3.11(e) et 3.11(f) montrent la prolifération du tissu cellulaire et la formation d'une bi-couche entre les MorphoPotts de type 3 et 4 et une multi-couche avec les autres MorphoPotts. Dans ces figures les cellules qui ont la couleur plus foncée meurent. Les courbes de droite représentent le nombre de cellules en fonction du nombre de pas de simulation. Les courbes montrent une prolifération infinie.

Troisième partie

Parallélisation du Système Multi-Agents : Mise en œuvre et Comparaison

Chapitre 1

Mise en œuvre parallèle sur GPU

Ce chapitre est destiné à améliorer considérablement le SMA défini dans la partie précédente en terme de vitesse de calcul. Pour cela une mise œuvre parallèle de ce SMA est une bonne solution. En effet, il peut se paralléliser efficacement car les agents n'ont pas de communications entre eux et que leurs interactions sont locales. Une autre raison est que le matériel pour exécuter des processus en parallèle est devenu performant, notamment les cartes graphiques. Nous décrivons le modèle de parallélisation choisi (1.1) et nous donnons la mise en œuvre du MorphoPotts dans ce formalisme (1.2).

1.1 Modèle de parallélisation choisi

Un défaut dans les systèmes multi-agents est que le temps de simulation peut augmenter trop rapidement quand le nombre d'agents est important. Chaque agent a une représentation explicite et est exécuté de façon indépendante. La simulation sur un seul processeur n'est pas recommandée car le temps de simulation augmente au moins de façon linéaire en fonction du nombre d'agents. A l'heure actuelle plusieurs matériels permettent l'exécution en parallèle : le multi cœur sur le CPU (Central Processing Unit), les grilles de calcul, le GPU (Graphics Processing Unit). Nous décrivons le choix du GPU (1.1.1) et nous donnons les principes de ce modèle de parallélisation (1.1.2) .

1.1.1 Choix de la programmation GPU

Pour choisir le modèle de parallélisation qui convient le mieux à notre SMA, nous étudions la taxonomie de Flynn, une classification des architectures d'ordinateur selon le flux d'instructions et de données. Quatre modèles sont définis : SISD (instruction simple, une seule donnée), SIMD (instruction simple, plusieurs données), MISD (instructions multiples, une seule donnée), MIMD (instructions multiples, plusieurs données).

SISD exécute une instruction sur une donnée. Il correspond aux ordinateurs classiques mono-cœur, il n'utilise pas le parallélisme, tant au niveau des instructions qu'à celui des données. SIMD exécute une seule instruction mais sur plusieurs données, l'instruction est donc exécutée plusieurs fois (éventuellement s'exécute en parallèle) mais sur des données différentes. Il s'agit d'un ordinateur qui utilise le parallélisme au niveau des données, comme les processeurs vectoriels. Le GPU fait partie de ce type. MISD exécute des instructions différentes sur une même donnée. Ce type est très peu mis en œuvre. MIMD exécute plusieurs instructions sur plusieurs données. Les ordinateurs avec un CPU multi-coeur font partie de ce type.

Dans notre système multi-agents chaque agent a un ensemble de comportements qui sont partagés entre eux. Les comportements sont synchronisés, par exemple ils exécutent tous dans le même temps

la production de molécules puis la consommation. Les mêmes comportements sont donc distribués aux agents. SIMD est donc le modèle de parallélisme le mieux approprié. Les instructions sont les comportements, les données mémorisent les attributs des agents. Un même comportement est donc exécuté en parallèle pour chaque agent. Les cartes graphiques reposent sur le modèle SIMD et sont un bon compromis entre le prix du matériel, les performances et la facilité de programmation. Aujourd'hui une carte graphique moyenne gamme permet d'exécuter plus de 100 processus en parallèle, et pour le haut de gamme plus de 1000 processus. Cependant, il est nécessaire d'avoir une API (Application Programming Interface) qui permet d'aider le développement d'application sur ce matériel. Nous pouvons citer deux APIs : Cuda [42] et OpenCL [62]. Nous choisissons OpenCL, car il permet d'exécuter une application sur différents matériels aussi bien sur le GPU et que sur le CPU. A l'inverse CUDA ne fonctionne que sur les cartes NVIDIA.

1.1.2 Description d'OpenCL

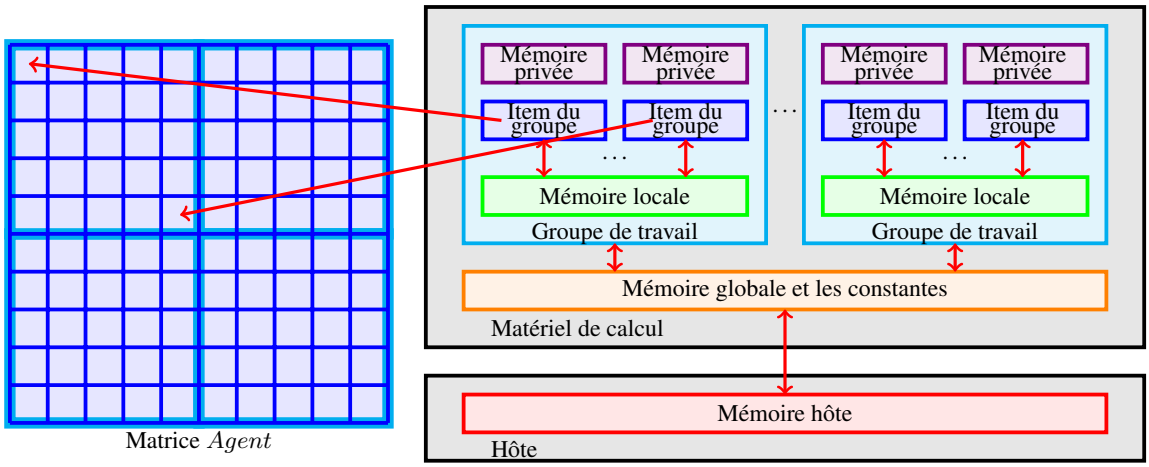


FIGURE 1.1 – Modèle de mémoire défini dans OpenCL. Trois niveaux de mémoires sont définies. Une mémoire globale qui est accessible à tous les processus, une mémoire locale partagée entre certains processus qui sont regroupés dans des groupes de travail, une mémoire privée accessible seulement par un seul processus. OpenCL fournit aussi le transfert des données du matériel hôte (généralement le CPU) vers le matériel de calcul (généralement le GPU). Les processus sont appelés items. OpenCL fournit autant d'item que voulu, ici nous donnons l'exemple où chaque agent (au nombre de 100) se trouve dans une position de la matrice $Agent$ de taille 10x10. 100 items sont donc définis, chacun correspondant à un agent. Tous les items d'un groupe de travail s'exécutent en parallèle par rapport à la mémoire et peuvent être synchronisés. Les items de groupes de travail différents peuvent s'exécuter en parallèle ou de façon séquentielle selon la capacité de la carte. Dans cet exemple, les groupes de travail sont de taille 5x5 donc quatre groupes de travail sont définis.

OpenCL est une interface de programmation et un langage de programmation proche du langage C (mais sans fonction récursive), proposé comme un standard ouvert par Khronos Group. Son but est de permettre le calcul intensif sur un matériel spécifique et l'orchestration de ces calculs par le matériel hôte. Il permet par exemple au programmeur de faciliter la mise en œuvre d'applications parallèles sur GPU (mais pas seulement). OpenCL gère le transfert entre l'hôte et le matériel de calcul, et la programmation sur ce matériel.

OpenCL permet un parallélisme de type SIMD où chaque processus exécute la même instruction sur des données différentes. Le parallélisme se fait sur les données et non sur les instructions. Les différents

niveaux de mémoires définis dans OpenCL sont illustrés dans la figure 1.1. Une mémoire globale partagée entre tous les processus, une mémoire locale partagée par un groupe de processus appelé groupe de travail et une mémoire privée associée à chaque processus. Les processus sont appelés items. L'accès à la mémoire globale par les items est moins rapide que l'accès à la mémoire locale, que l'accès à la mémoire privée. A l'inverse, la taille de la mémoire globale est plus grande que la taille de la mémoire locale qui elle même est supérieure à la taille à la mémoire privée.

OpenCL assure le parallélisme et les synchronisations entre les items d'un même groupe de travail. Les items de groupes de travail différents peuvent s'exécuter de manière parallèle ou séquentielle en fonction de la capacité du matériel. OpenCL indique le nombre de groupes de travail qui s'exécutent en parallèle. Seulement les items d'un même groupe peuvent être synchronisés. Comme le parallélisme est sur les données, si deux items sont exécutés en parallèle cela signifie qu'ils lisent la mémoire dans le même temps, effectuent leur traitement et modifient la mémoire dans le même temps. Cependant l'exécution de leur traitement peut se faire en parallèle ou en séquentielle. OpenCL ne définit pas le nombre de processus qui exécutent leurs instructions en parallèle et cela est dépendant du constructeur. Par exemple chez NVIDIA le nombre de processus qui s'exécutent en parallèle est classiquement de 32. Ainsi, le nombre de processus qui exécutent leurs instructions en parallèle est égale aux nombre de groupes de travail qui s'exécutent en parallèle multiplié par 32. Chez ATI cela dépend fortement de la carte graphique.

Les programmes qui sont lancés sur le matériel de calcul sont appelés noyaux. Pour chaque noyau un nombre d'items et la taille des groupes de travail doivent être définis. Chaque item exécute une instance du noyau appelée tâche. Les noyaux sont écrits en OpenCL, un langage proche du langage C mais sans récursivité et sans allocation dynamique de la mémoire, mais avec une librairie OpenCL pour permettre, par exemple, de connaître l'identifiant (un entier) du groupe de travail ou de l'item. Chaque item exécute donc séquentiellement le noyau. Plusieurs noyaux peuvent être définis et sont orchestrés par le matériel hôte.

OpenCL permet seulement une allocation statique de la mémoire. Toute la mémoire doit être allouée à l'avance. Notre mise en œuvre définit, pour chaque attribut des agents (la membrane, le volume, . . .), un tableau dont la taille est égale au nombre maximum d'agents qui peuvent être exécutés. Ce tableau est défini de tel façon que les attributs d'un agents sont enregistrés à l'indice qui correspond à l'identifiant d'un item. Dans la suite du chapitre nous définissons la mise en œuvre, c'est à dire les différents noyaux et tableaux pour enregistrer les données.

1.2 Mise en œuvre

Cette section décrit l'état de l'art des modèles de cellules mis en œuvre sur GPU (1.2.1) puis décrit l'originalité de notre approche et les problèmes liés à la programmation avec OpenCL (1.2.2).

1.2.1 Travaux connexes

A notre connaissance les travaux les plus proches de notre approche sont : FLAME GPU [67, 66], les travaux faits dans [54] et dans [76]. FLAME GPU est une extension GPU pour le calcul haute performance du cadre de travail FLAME destinée à la conception de modèles de simulation basés sur les agents [72]. FLAME GPU décrit et fournit un code optimisé pour le GPU et gère de manière efficace l'accès à la mémoire. Chaque agent est une "Communicating Stream XMachine" [11]. Chaque agent a un ensemble d'états où chaque transition entre les états correspond aux comportements. Un comportement peut changer la mémoire locale, $i.e.$ la mémoire de l'agent. Un agent peut aussi dépiler des messages laissés par d'autres agents et à son tour peut ajouter des messages qui peuvent être lus par d'autres agents.

Dans [54], un cadre de travail pour des modèles de simulation mega-échelle a été mis en œuvre sur

GPU. Comme dans [66], les agents peuvent se répliquer et mourir. Les agents sont mobiles et contiennent trois tâches de bases : sauvegarder l'état d'un agent, mettre à jour l'état d'un agent, connecter l'agent à l'environnement. L'état d'un agent est encodé dans une texture, $i.e.$ utiliser la valeur RGBA des texels. L'identifiant d'un agent est défini par sa position ce qui permet de le lier à l'environnement. La mise à jour des états des agents est effectuée par les comportements qui sont exécutés aléatoirement, alors que dans les Communicating Stream XMachine, les comportements sont ordonnancés de manière séquentielle.

Dans [76], une mise en œuvre du Modèle de Potts Cellulaire sur le GPU a été réalisée. Elle permet aussi aux agents de se déplacer, de se répliquer et de mourir. A l'inverse des deux précédents travaux, les agents peuvent occuper plusieurs sites dans l'environnement.

Notre mise en œuvre s'éloigne des travaux de [67] et de [54], les agents contiennent plusieurs sites dans l'environnement qui ne sont pas forcément adjacents. Cette propriété impose des contraintes dans la programmation sur GPU notamment sur la représentation de l'agent (pas seulement un pixel) et sur les conflits. Comme dans [67], notre agent a un ensemble de comportements qui sont exécutés dans un ordre donné. Dans [76], les agents aussi ont plusieurs positions, mais leur système est vu comme un automate cellulaire c'est à dire que les agents n'ont pas une représentation d'eux même. Les différentes positions sont seulement codées dans l'environnement. Dans les autres travaux ainsi que le notre, un processus correspond à un agent, alors que dans [76] un processus correspond à un site de l'environnement donc à une partie de l'agent.

1.2.2 Mise en œuvre

Notre mise en œuvre parallèle sur GPU est faite via OpenCL où chaque processus correspondant à un agent exécute les mêmes instructions que tous les autres agents. Comme la mémoire dans OpenCL est allouée statiquement, toutes les données que ce soit l'environnement ou les attributs des agents sont sauvegardées en mémoire globale de telle manière que nous définissions à l'avance le nombre maximum d'agents qui peuvent être exécutés. Chacune des données sont enregistrées dans des matrices. Au minimum une matrice Env correspondant à l'environnement de taille fixe doit être définie. Cette section montre comment notre agent est mis en œuvre et accède à la matrice à travers ses comportements de déplacements, de division, de mort et comment la mémoire est gérée pour permettre de la libérer lorsque les agents meurent.

La mémoire de l'agent

L'agent MorphoPotts connaît entre autre : sa membrane, les sites auxquels il est présent, son volume et sa surface. OpenCL fournit 3 niveaux de mémoire : privée, locale et globale. Même si la mémoire privée est celle dont les accès sont plus performants, le problème est qu'elle est trop petite pour contenir la mémoire d'un agent. La mémoire d'un agent est codée dans la mémoire globale et l'accès est possible via son identifiant. Dans la figure 1.2 nous donnons la représentation de la mémoire de l'agent sous forme d'une matrice, et selon une matrice Env qui représente la position de tous les agents qui se trouvent dans l'environnement. Nous définissons la variable NB_AGENT, qui représente le nombre d'agents possibles. La matrice $Agent$ décrit si un agent est ou n'est pas créé et, s'il est créé, un indice de la division lui est donné. L'indice de la division est un identifiant pour le nouvel agent lorsque l'agent se divise. Un agent peut lire l'ensemble des sites où il se trouve et les sites sur sa membrane, dans les matrices $Sites$ et $Membrane$. L'accès à un agent dans ces matrices se fait en utilisant son identifiant de la forme $identifiant + i * NB\_AGENT$ où $i \in N$. Nous pouvons observer que $Sites[identifiant]$ et $Membrane[identifiant]$ donnent le prochain indice disponible pour insérer le prochain site dans ces matrices pour un agent donné. L'identifiant 0 est utilisé pour indiquer l'espace vide dans l'environnement, donc dans ce cas ni $Membrane$ ni $Sites$ ne sont mises à jour.

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	1	2	2	0
2	0	1	1	1	2	0
3	0	0	1	0	0	0
4	0	0	0	0	0	0

(a) Matrice Env

	0	1	2	3	....	*NB_AGENT*
Mort :	0	0	0	1	....	1
Indice de la division :	0	$2^{log2(4)}$+0	$2^{log2(4)}$+1	0	...	0

(b) Matrice $Agent$

	0	1	2	3	...	*NB_AGENT*
Prochain indice :	1	6	4	1	....	1
site 1 :	0	2+6*1	3+6*1	0	....	0
site 2 :	0	1+6*2	4+6*1	0	....	0
site 3 :	0	2+6*2	4+6*2	0	....	0
site 4 :	0	3+6*2	0	0	....	0
site 5 :	0	2+6*3	0	0	....	0
site 6 :	0	0	0	0	....	0
site .... :	0	0	0	0	....	0
site NB_SITES :	0	0	0	0	....	0

(c) Matrice $Sites$

	0	1	2	3	...	*NB_AGENT*
Prochain indice :	1	5	4	1	....	1
site 1 :	0	2+6*1	3+6*1	0	....	0
site 2 :	0	1+6*2	4+6*1	0	....	0
site 3 :	0	3+6*2	4+6*2	0	....	0
site 4 :	0	2+6*3	0	0	....	0
site 5 :	0	0	0	0	....	0
site 6 :	0	0	0	0	....	0
site .... :	0	0	0	0	....	0
site NB_SITES :	0	0	0	0	....	0

(d) Matrice $Membrane$

FIGURE 1.2 – Exemple de la mémoire de l'agent selon la matrice Env. Nous voyons qu'en $Env[3,1]$ se trouve l'agent 2. Il suffit de prendre cette valeur pour connaître ses propriétés dans les matrices $Agent$, $Sites$ et $Membrane$.

Le mouvement de l'agent

Le mouvement de l'agent est possible *via* deux noyaux[4] : $kAddSite$ ($cf.$ listing 1.1) et $kSupSite$ ($cf.$ listing 1.2). Le noyau est le code qui est exécuté par toutes les items (agents). Les noyaux sont

4. Rappelons qu'un noyau est la terminologie OpenCL pour une fonction C exécutée dans une tâche.

exécutés séquentiellement et sont gérés par un ordonnanceur. Le noyau $kAddSite$ ajoute des sites à un agent C_σ et met à jour sa mémoire et l'environnement. Le noyau $kSupSite$ met à jour la mémoire de l'agent selon l'environnement, car le noyau $kAddSite$ appelé d'un agent C_σ a pu supprimer un site de l'agent C'_σ. Un agent ne peut pas modifier la mémoire d'un autre agent, donc les agents savent si un de leurs sites a été supprimé en comparant leur mémoire avec l'environnement. Cette mise en œuvre évite les points de synchronisation explicites, car chaque agent modifie sa propre mémoire. En contrepartie, lors de l'ajout d'un site par le noyau $kAddSite$ l'agent n'a qu'une approximation de son volume et de sa surface car d'autres agents ont pu supprimer certains de ses sites. Nous supposons que l'erreur est négligeable, car seulement les sites sur la membrane ont pu être supprimés. Lors de calcul des énergies, les agents peuvent lire seulement une sauvegarde de la mémoire des autres agents correspondant à la mémoire avant l'appel du noyau. Les noyaux $kAddSite$ et $kSupSite$ sont connectés, le noyau $kSupSite$ est toujours exécuté après que le noyau $kAddSite$ soit exécuté.

Le noyau $kAddSite$, permet à un agent l'ajout d'un site voisin d'un site qui se trouve dans $Membrane$. Pour cela, pour chaque site s(l.7), l'agent choisit un site voisin aléatoire s' où un autre agent est présent (l.10). Si s' est un site qui n'est pas en conflit $i.e.$ qu'un autre agent ne modifie pas s' (l.17), s' est ajouté en appelant la méthode $addSite$, décrite dans le chapitre 2 partie 2. Pour détecter le conflit, nous créons un filtre, défini par la méthode $scheduler$, qui accepte un site s, si aucun site voisin à s n'est accepté. La position du filtre, $i.e.$ le choix du site s, change aléatoirement. Dans le cas d'un agent a, qui a un site s avec tous ses voisins s' contenant a ou aucun agent, alors le site s ne peut pas être supprimé. En effet, la suppression d'un site est possible que depuis un autre agent. Dans ce cas, où s' ne contient pas d'agent, $i.e.$ contient l'identifiant 0 (l.13), alors selon une probabilité de 50% (l.13) s et s' permutent (l.14) et $idem$ pour les agents a et 0(l.15).

Listing 1.1 – kernel kAddSite

```
1  //get_global_id est une fonction définie dans OpenCL qui donne
2  //l'identifiant de l'item qui est en train d'exécuter ce noyau
3  int agent = get_global_id();
4  if(MatrixAgent[agent]==1)return; //l'agent est mort
5  int size=Membrane[agent];
6  // agent est l'identifiant de l'agent qui correspond à l'item courant
7  for(int i=1;i<size;i++){
8    int s = Membrane[agent+NB_AGENT*i] ;
9    //donne la position relative d'un site voisin
10   int sv = rndNeighboorDiff(s);
11   agent = Env[s];
12   int agentv = Env[sv]; //agent voisin
13   if(agentv == 0 && rndint(Alea)%2 == 0){
14   int save = s; s = sv; sv = save;
15   save = agent; agent = agentv; agentv = save;
16   }
17   if(scheduler(sv){ //filtre pour éviter les conflits
18   /*insérer ici l'appel à la méthode
19   addSite selon le  site s et l'agent*/
20   agent = get_global_id();
21   }
22 }
```

Le noyau $kSupSite$ met à jour sa mémoire si l'agent a été modifié. Pour cela, chaque site s de l'agent contenu dans $Sites$ (l.3), vérifie que le site s dans Env contient l'agent (l.5). Dans ce cas, si un site voisin de s contenant un autre agent existe, alors il ajoute s à $Membrane$ (l.9, l.10, l.11), qui a été réinitialisé (l.2). Dans le cas où s contient un autre agent, s est supprimé du $Sites$ en appelant la méthode $supSite$

définie dans le chapitre 2 partie 2.

Listing 1.2 – kernel kSupSite

```
int agent = get_global_id();
if(MatrixAgent[agent]==1)return; //l'agent est mort
Membrane[agent]=1;
for(int i=1;i<Sites[agent];i++){
 int s = Sites[agent+i*NB_AGENT] ;
 if(Env[s]!=agent){
  /*insérer ici l'appel à la méthode
  supSite selon le  site s*/
 }
 else if(numberOfNeighboorsWithOutType(Env,NB_AGENT,s)>0){
  Membrane[agent+MembraneLocal[agent]*sizeGroup]=s;
  Membrane[agent]++;
 }
}
```

Division de l'agent

Le noyau $kSplit$, décrit dans le listing 1.3, donne la capacité à un agent de se diviser selon un critère $criterionSplit$ (l.2). Pour cela, un nouvel agent est créé grâce à l'indice de division de l'agent (l.4, l.6), qui assure un identifiant unique. Nous mettons à jour l'index de division pour l'agent et pour le nouvel agent crée (l.7, l.9, l.10). Pour chaque site s de $Sites$ où $criterionBorn$ est vérifié (l.13), le site s est transféré au nouvel agent. Finalement, le noyau $kSupSite$ est appelé pour mettre à jour la mémoire de l'agent car il a perdu quelques sites durant la division (l.19). Pour assurer l'indice de la division, nous supposons que le calcul de l'indice de la division se fait de la manière suivante :
Si i' est l'indice de la division de l'agent a d'identifiant i alors, après une division de l'agent a, l'agent a' est créé avec l'identifiant i, l'indice de la division de l'agent a est $exp_2(\lfloor log_2(i') \rfloor) + 1) + (i - 1)$ et l'indice de la division de l'agent a' est $exp_2(\lfloor log_2(i') \rfloor) + 1) + (i' - 1)$.

Listing 1.3 – kernel kSplit

```
int agent = get_global_id();
if(MatrixAgent[agent]==1)return; //l'agent est mort
if(/*insérer ici le critère criterionSplit*/){
 int new_agent = MatrixAgent[agent+NB_AGENT];
 //creation d'un nouvel agent
 MatrixAgent[new_agent]=0;
 int base = floor(log2(new_agent ))+1;
 //calcul du prochain indice de la division
 MatrixAgent[agent+NB_AGENT]=exp2(base)+(agent-1);
 MatrixAgent[new_agent +NB_AGENT]=exp2(base)+(new_agent -1);
 for(int i=1;i<Sites[agent];i++){
  int s = Sites[agent+i*NB_AGENT] ;
  if(/*insérer ici le critère criterionBorn
    selon le site s*/){
   /*insérer ici l'appel à la méthode
   addSite selon le site s et le nouvel agent*/
  }
 }
 /*insérer ici le code du noyau kSupSite*/
}
```

Mort d'un agent

Le noyau $kDeath$, décrit dans listings 1.3, donne à l'agent la capacité de mourir selon un critère $criterionDeath$ (l.3). Pour cela, l'agent met à jour la matrice $Agent$ au champ $Mort$, avec la valeur 1.

Listing 1.4 – kernel kDeath

```
1 if(MatrixAgent[agent]==1)return;
2 if(/*insérer ici le critère criterionDeath*/)
3  MatrixAgent[agent]=1;//mort d'un agent
```

La gestion de la mémoire

Dans OpenCL, le nombre d'items et la taille de la matrice sont statiques. L'agent ne peut pas être créé dynamiquement. Pour résoudre ce problème, nous allouons, selon le nombre maximal d'agents, toute la mémoire nécessaire statiquement. Notre mise en œuvre de la division d'un agent est telle que même s'il y a des identifiants disponibles, la division de l'agent ne peut pas être faite car l'indice du prochain indice de division dépasse le nombre maximal d'agents. Pour résoudre ce problème, la matrice $Agent$ doit être défragmentée avec une certaine fréquence. Ceci est décrit dans la figure 1.3.

Le noyau $kDefrag$, décrit dans le listing 1.5, permet la défragmentation de la matrice $Agent$. Ce noyau n'est pas utilisé par l'agent, mais comme une contrainte externe pour la gestion de la mémoire. Il est exécuté seulement par un item. Le noyau n'est pas exécuté en parallèle, mais ceci n'implique pas une baisse de performance car il est exécuté avec une certaine fréquence (pas à chaque pas de simulation). Le noyau $kDefrag$ met à jour la matrice Env, car l'identifiant de l'agent (sa position dans la matrice Env) a pu être modifié par la défragmentation, ainsi que l'indice de la division ($cf.$ la figure 1.3). Pour chaque agent (l.8) un identifiant inférieur à l'identifiant de l'agent est cherché. Dans le cas où un identifant est trouvé, il devient l'identifiant de l'agent courant et les matrices $Sites$ et $Membrane$ sont mises à jour (l.13, l.14). Sinon l'indice de la division de l'agent est mis à jour (l.21).

Listing 1.5 – kernel kDefrag

```
1 if(time%frequency>0)return; //time est le nombre de pas de simulation
2 //frenquency : la fréquence de la défragmentation
3 int nb=0;//nb est le nombre d'agent mort
4 for(int i=1;i<NB_AGENT;i++){
5   if(Agents[i]==0)nb++;
6 }
7 int base = floor(log2(nb))+1;
8 for(int i=1;i<NB_AGENT;i++){
9  if(Agents[i]==0){//un agent i existe
10  /*insérer le code pour trouver l'indice, nommé ind,
11  qui est disponible entre 1 et i, s'il existe*/
12  /*si il existe faire :*/
13    Agents[ind]=Agents[i];Agents[i]=1;
14    Agents[ind+NB_AGENT]=exp2(base)+ind;Agents[i+NB_AGENT]=0;
15    for(int j=0;j=<NB_SITES;j++){
16      Sites[ind+j]=Sites[i+j];Sites[i+j]=0;
17      Membrane[ind+j]=Membrane[i+j];Membrane[i+j]=0;
18      Sites[i]=1;Membrane[i]=1;
19    }
20  /*sinon faire :*/
21    Agents[i+NB_AGENT]=exp2(base)+i;
22 }
```

```
23
24 }
```

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	1	2	2	0
2	0	1	1	1	2	0
3	0	0	1	5	5	0
4	0	0	0	0	0	0

(a) Matrice Env avant défragmentation

	0	1	2	3	4	5
Mort :	0	0	0	1	1	0
Indice de la division :	0	$2^{log2(4)}$+0	$2^{log2(8)}$+1	0	0	$2^{log2(8)}$+4

(b) Matrice $Agent$ avant défragmentation

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	1	2	2	0
2	0	1	1	1	2	0
3	0	0	1	3	3	0
4	0	0	0	0	0	0

(c) Matrice Env après défragmentation

	0	1	2	3	4	5
Mort :	0	0	0	0	1	1
Indice de la division :	0	$2^{log2(4)}$+0	$2^{log2(4)}$+1	$2^{log2(4)}$+2	0	0

(d) Matrice $Agent$ après défragmentation

FIGURE 1.3 – Exemple de la défragmentation de la matrice $Agent$. L'agent numéro 3 est mort tandis que le 5 est encore vivant. La défragmentation fait passer les données de l'agent 5 à la position 3 et remplace dans Env tous les 5 par des 3.

Ordonnanceur des agents

Nous définissons un ordonnanceur global qui indique l'ordre dans lequel sont exécutés les noyaux. Entre chaque noyaux les agents sont synchronisés. Les agents vont s'exécuter en parallèle (jusqu'à la capacité de la carte graphique) les noyaux. Un ordonnanceur qui est à la base de nos simulations est défini dans la figure 1.4. Pour les noyaux $kAddSite$, $kSupSite$, $kDeath$, $kSplit$, NB_AGENT items vont exécuter ces noyaux, et pour le noyau $kDefrag$ seulement un item. Le noyau $kDefrag$ est appelé à une certaine fréquence pour ne pas baisser les performances. Nous définissons un pas de simulation comme étant une exécution de cet ordonnanceur.

FIGURE 1.4 – Ordonnanceur basique

Application de la morphogenèse

Dans la section précédente, nous avons décrit notre SMA, et la mise en œuvre d'un agent. Dans le chapitre prochain nous appliquons notre mise en œuvre à deux modèles de morphogenèse cellulaire : le tri cellulaire et l'embryogenèse. Nous donnons dans la suite les extensions nécessaires de notre mise en œuvre.

Le tri cellulaire La simulation du tri cellulaire décrit dans le chapitre 3 met en jeu les noyaux $kAddSite$ et $kSupSite$ qui sont exécutés de manière séquentielle. Ce modèle va nous permettre de vérifier que l'approximation du volume et de la surface n'influence pas la dynamique. Les résultats de la simulation et l'étude des performances sont donnés dans le chapitre suivant.

L'embryogenèse Pour simuler le modèle de la formation de plusieurs tissus décrit dans le chapitre 3 partie 2 , nous avons :

- créé le noyau $kProd$ pour donner à l'agent la capacité de produire des molécules dans l'environnement.
- créé le noyau $kCons$ pour donner à l'agent la capacité de consommer des molécules présentes dans l'environnement et aussi pour augmenter son énergie.
- créé le noyau $kMain$ pour décrémenter l'énergie des agents.
- créé le variable $energy$ dans la mémoire de l'agent pour enregistrer son énergie.
- défini le critère $criterionDeath$, qui est vérifié si l'énergie=0.
- créé la variable $gravity$ dans la mémoire de l'agent pour connaître son centre de gravité.
- modifié le $kAddSite$ et le $kSupSite$ pour permettre à l'agent de mettre à jour sa $gravity$.
- modifié le $kAddSite$ pour prendre en compte le gradient de l'énergie dans l'énergie du CPM, pour permettre la migration d'un gradient des molécules.
- modifié le $kSplit$ pour permettre aux nouveaux agents d'avoir un type diffèrent de celui de l'agent père.
- défini le critère $criterionSplit$ qui est vérifié si l'énergie de l'agent est suffisant et son volume effectif est supérieur à 80% du volume cible.
- défini le critère $criterionBorn$ qui est vérifié si le site s concerné est au dessous du centre de gravité. L'agent peut se diviser selon un axe horizontal.
- modifié l'environnement en créant une matrice $MoleculesX$ pour chaque molécule X définie pour connaître quelles molécules sont pressentes dans l'environnement.
- créé les matrices $MoleculesX$ pour enregistrer les molécules dans les matrices dans l'environnement et permettre leur diffusion. Pour cela, nous avons créé le noyau $kMol$ qui permet la diffusion dans la matrice.

Tous ces comportements ont les mêmes définitions que dans le chapitre 2 partie 2, sauf pour la diffusion, production et consommation des molécules. La diffusion des molécules est calculée de telle manière qu'à chaque appel du noyau $kMol$ la quantité des molécules au site s est égale à $1/9$ du nombre de molécules présentes dans ce site, plus $1/9$ du nombre de molécules présentes dans chacun des 8 sites voisins directs. Ceci permet une diffusion plus précise que dans la partie 2 et une mise en œuvre dans le modèle du

parallélisme choisi. La production et la consommation au site s impliquent juste une augmentation et une baisse du nombre des molécules dans ce site. Le chapitre suivant valide et étudie les performances de ce modèle.

Chapitre 2

Éléments de validation et étude des performances

Dans ce chapitre nous vérifions et analysons les performances de la mise en œuvre de notre système multi-agents sur GPU (2.1). Les performances nous permettent alors d'analyser en détails l'auto-organisation des cellules (2.2).

2.1 Résultats et analyse des performances

Dans cette section, nous vérifions que la mise en œuvre de notre SMA, où les agents sont exécutés en parallèle, amène la même dynamique que dans les travaux précédents où les agents étaient exécutés en séquentiel, et nous définissions un protocole pour analyser les performances (2.1.1) via deux simulations. Les deux simulations réalisées se basent sur les modèles décrits dans le chapitre 3 partie 2 : le tri cellulaire (2.1.2) et l'embryogenèse (2.1.3).

2.1.1 Protocole pour analyser le temps de calcul

Pour évaluer la performance de notre système, nous devons prendre en considération deux aspects : le matériel (GPU,CPU) et le nombre d'agents. A chaque pas de simulation les agents sont synchronisés, donc le temps de simulation est la somme de tous les temps de calculs à chaque pas. Dans le cas du CPU comme matériel de calcul, les agents sont exécutés séquentiellement, le temps de simulation augmente linéairement en fonction du nombre d'agents. Dans le cas du GPU, les agents sont exécutés en parallèle jusqu'à ce que le GPU soit surchargé (limitation à l'accès mémoire, branchement dans le code, le nombre maximum d'items et de groupes de travail qui peuvent s'exécuter en parallèle). Le temps de simulation n'augmente pas tant que la surcharge n'est pas atteinte, puis augmente linéairement ou exponentiellement en fonction du nombre d'agents et de la capacité du GPU.

Dans le but de comparer les résultats sur le CPU et sur le GPU, nous exécutons la même simulation avec un nombre d'agents qui augmente exponentiellement. Comme le CPU est bien adapté pour un petit nombre d'agents, le but de l'étude de performance est de connaître le nombre d'agents pour lequel il est avantageux d'utiliser le GPU comme matériel de calcul. Nous testons les modèles de simulation avec un CPU 4 cœurs et deux GPUs : NVIDIA GeForce 9600M GT et ATI FIREPRO V7800. Notre mise en œuvre permet d'utiliser le GPU NVIDIA avec au maximum 512 items dans un groupe de travail, et le nombre de groupes de travail qui peuvent être exécutés en parallèle est de 4. Les items d'un groupe de travail s'exécutent en parallèle sur les données, c'est à dire qu'ils lisent et modifient les données dans le même temps. L'exécution des instructions de tous les items ne se font pas forcément parallèlement.

Chez NVIDIA les instructions sont exécutées en parallèle par groupe de 32 items. Dans nos simulations pour analyser les performances nous imposons des groupes de travail de taille 32 même si cela n'est pas recommandé car plus il y a de groupes de travail plus il y a de changement de contexte. Le GPU NVIDIA peut exécuter 128 items en parallèle sur les instructions. Le GPU ATI peut exécuter au maximum GPU 256 items dans un groupe de travail et le nombre de groupes de travail qui peuvent être exécutés en parallèle est de 18. En prenant en compte que nous imposons la taille des groupes de travail à 32 le GPU ATI exécute 576 items en parallèle. Les sections suivantes analysent les performances avec le modèle de tri cellulaire et avec le modèle de la formation de plusieurs tissus.

2.1.2 Le tri cellulaire

Dans cette partie nous vérifions que la simulation de notre SMA appliqué au tri cellulaire garde la même dynamique que dans des travaux précédents [35]. Nous analysons aussi les performances (2.3) et le gain par la programmation sur GPU.

Validation de la mise en œuvre

La simulation du tri cellulaire est basée sur le modèle décrit dans le chapitre 3 partie 2, et les paramètres utilisés pour l'énergie de contacts sont décrits dans [35]. L'ordonnanceur du système est donné dans la figure 2.1. Chaque noyau est exécuté par 128 items (donc 128 agents) et la taille d'un groupe de travail est fixée à 32.

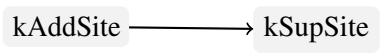


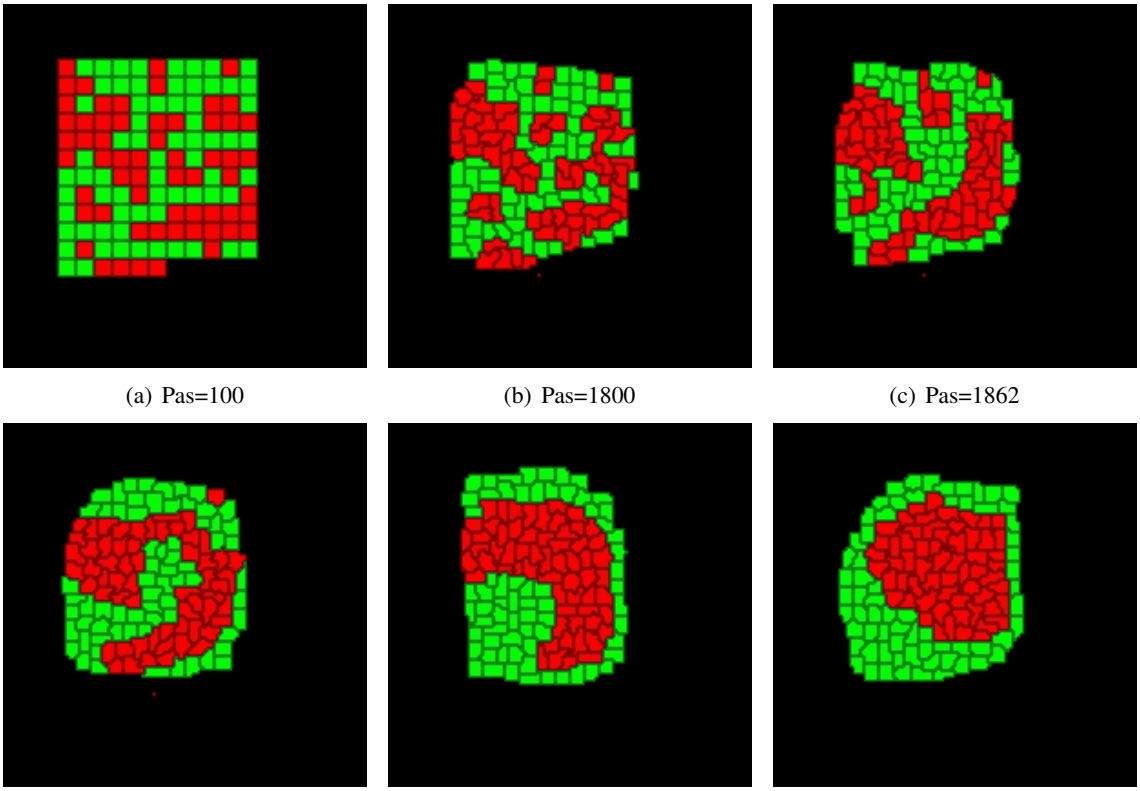
FIGURE 2.1 – Ordonnanceur du tri cellulaire

La figure 2.2 montre une simulation réalisée avec 128 agents. On peut observer la même dynamique que dans le chapitre 3 partie 2. Les agents rouges forment un groupe car l'énergie de contacts favorise cette propriété, et les agents verts entourent ce groupe. Donc la mise en œuvre sur GPU de notre SMA semble validée. Les approximations du volume et de la surface dû à la parallélisation sont négligeables.

Analyse des performances

Nous analysons les performances de la simulation du tri cellulaire selon le nombre d'agents. La simulation a été réalisée sur les différents matériels décrits précédemment. Dans toutes les simulations la taille d'un groupe de travail est de 32. La figure 2.3 montre les performances. Dans le cas du CPU, le temps de simulation augmente linéairement. Ceci n'est pas surprenant, car, comme le CPU a 4 cœurs, seulement 4 agents peuvent être exécutés en parallèle. Pour les deux GPUs, tant que le GPU a un nombre de groupes de travail inférieur au nombre de groupes de travail qui peuvent être exécutés en parallèle, le temps de simulation reste constant. Sinon, le temps de simulation augmente linéairement. Ceci implique que notre mise œuvre est efficace car elle tire pleinement avantage de la parallélisation. Par exemple, avec l'ATI le nombre d'agents augmente seulement de 7% le temps de simulation, entre 32 et 512 agents. Même si ce taux de 7% est négligeable, il est dû aux branchements (les structures conditionnelles) qui se trouvent dans les programmes et aux accès restreints de la mémoire.

Comme le GPU NVIDIA peut exécuter en parallèle 128 agents et comme le GPU ATI peut exécuter 572 agents en parallèle, la surcharge des GPUs est atteinte pour ce nombre d'agents. Pour peu d'agents, le CPU est plus rapide que le GPU, car les noyaux ne sont pas entièrement adaptés au GPU (branches conditionnelles et accès à la mémoire globale) et parce que la fréquence du CPU est plus rapide. Mais, pour un grand nombre d'agents (supérieur à 512 agents), le GPU ATI est 3.6x rapide que le CPU.



(a) Pas=100 (b) Pas=1800 (c) Pas=1862

(d) Pas=22100 (e) Pas=45000 (f) Pas=62143

FIGURE 2.2 – Simulation sur GPU du tri cellulaire avec 128 agents. Le vidéo de la simulation se trouve à l'adresse suivante : `http://www.youtube.com/watch?v=xYQMX7ijh5s`

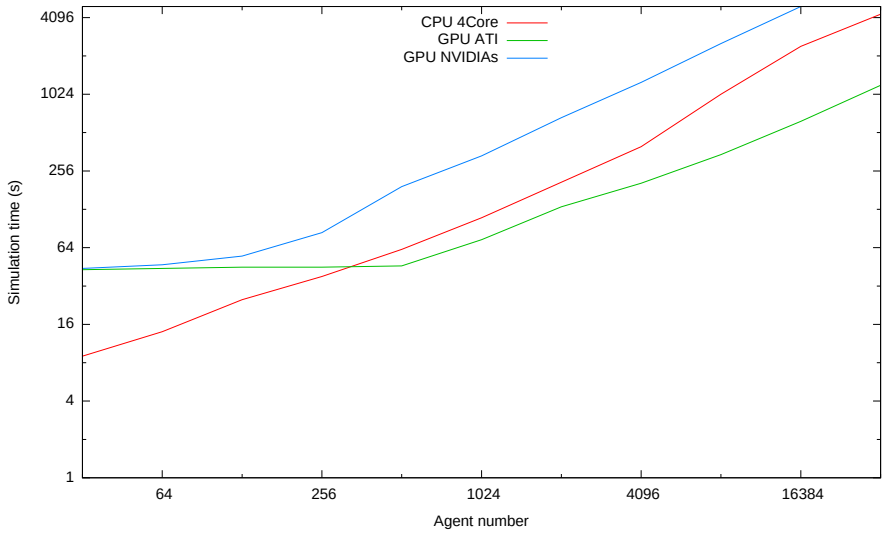


FIGURE 2.3 – Performance du tri cellulaire selon le nombre d'agents sur un CPU 4 cœurs Intel 2,83GHz, GPU NVIDIA GeForce 9600M GT et GPU ATI FIREPRO V7800.

2.1.3 L'embryogenèse

Dans cette partie nous vérifions que la simulation de notre SMA appliquée à l'embryogenèse maintient la même dynamique (2.1.3) que pour la simulation faite dans le chapitre 3 partie 2. Nous analysons aussi

les performances (2.1.3).

Validation de la mise en œuvre

La simulation de l'embryogenèse est basée sur le modèle de la formation de plusieurs tissus, décrit dans le chapitre 2 partie 2. L'ordonnanceur du système est donné dans la figure 2.4. Pour les noyaux $kAddSite$, $kSupSite$, $kDeath$, $kSplit$, $kMain$, $kProd$ et $kCons$ 16384 agents vont être exécutés, pour le noyau $kDefrag$ seulement un agent et pour le noyau $kMol$ 118000 items. En d'autres termes, chaque agent exécute les noyaux $kAddSite$, $kSupSite$, $kDeath$, $kSplit$, $kMain$, $kProd$ et $kCons$ séquentiellement et dans une manière synchrone avec les autres agents. Comme les agents doivent avoir un certain volume et emmagasiner assez d'énergie pour se diviser, le noyau $kSplit$ est exécuté tous les 50 (même fréquence que la division) pas de simulation. Le noyau $kDefrag$ est exécuté à la même fréquence pour éviter les effets de bord. Le noyau $kMol$ est utilisé pour diffuser toutes les molécules de l'environnement, donc chaque item c est associé à un site s de l'environnement. Si les molécules sont présentes sur un site s, alors l'item c distribue ses molécules aux sites voisins. Le noyau $kMol$ est exécuté à chaque pas de simulation.

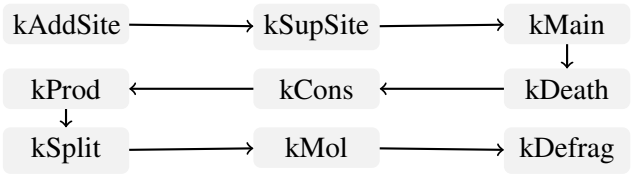
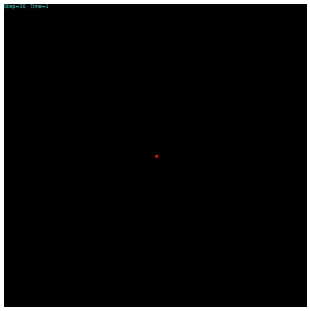


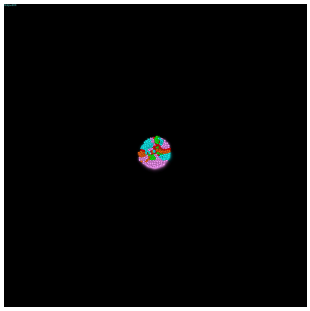
FIGURE 2.4 – Ordonnanceur de l'embryogenèse

La figure 2.5 montre une simulation réalisée avec 16384 agents dans un environnement 768x768. Nous pouvons observer les mêmes étapes que la simulation faite dans le chapitre 3 partie 2. Nous observons 3 étapes dans cette morphogenèse. La première étape ($cf.$ les figures 2.5(a) et 2.5(b)) est la différenciation cellulaire et la sélection naturelle. Le MorphoPotts de type 1 (en rouge) se divise et se différencie aléatoirement dans 4 types. Ceci amène à la formation de tissus. La deuxième étape est le tri des tissus ($cf.$ les figures 2.5(a), 2.5(b), qui sont triés par le simple fait de la mort (les cellules ne trouvent pas de molécules) et de la division (les cellules trouvent des molécules). La troisième étape est la prolifération et l'émergence du motif ($cf.$ les figures 2.5(e), 2.5(f) et2.5(g)). Ici une prolifération en spirale émerge (non imposée dans la description). Nous pouvons observer que le renouvellement du tissu est continu, $i.e.$ le tissu n'est pas statique. Dans le chapitre 3 partie 2, la même spirale est aussi obtenue, mais seulement le début de sa formation, car le temps de simulation est de 30 heures. Nous avons conclu que la prolifération de l'agent semble infinie. La mise en œuvre sur GPU permet de simuler un grand nombre de pas de simulation (pour le même nombre de pas que sur le CPU, le temps de simulation est de 4 minutes). On observe maintenant que la prolifération est finie. La raison est que le centre de la spirale tourne et se dégrade, amenant une dégradation de la spirale à son centre. Un équilibre est trouvé entre la mort et la division des agents.

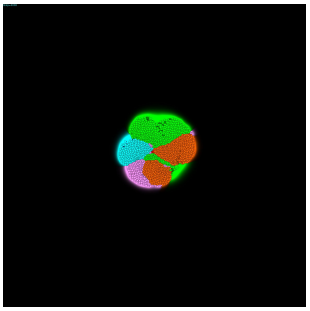
Nous pouvons observer que les courbes représentant le nombre d'agents de même type forment des ondes. Les ondes sont décalées par t (environ 5 000 pas de simulation). Ce décalage est lié à leurs dépendances par rapport aux molécules. Par exemple, l'onde décrite par la courbe 1 atteint son maximum au temps $t1$ si l'onde décrite par la courbe 2 atteint son maximum au temps $t1 - t$. La raison est que les molécules M_2 produites par les agents de type 2 sont consommées par les agents de type 1. La quantité des molécules M_2 est au maximum quand le nombre des agents de type 2 est maximum. La conséquence est une augmentation dans le nombre des agents de type 1. Au temps $t1 - t$ le nombre d'agents de type 2



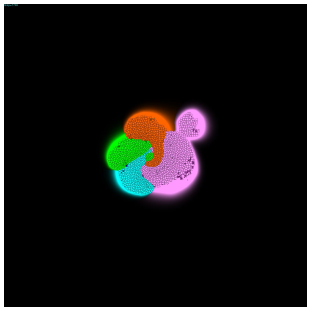
(a) Pas=30



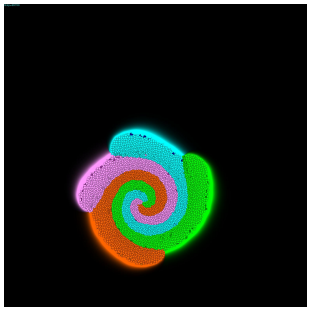
(b) Pas=800



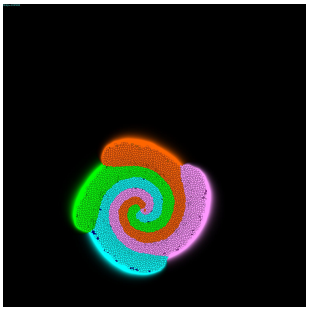
(c) Pas=4000



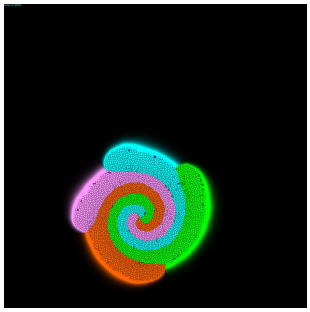
(d) Pas=7700



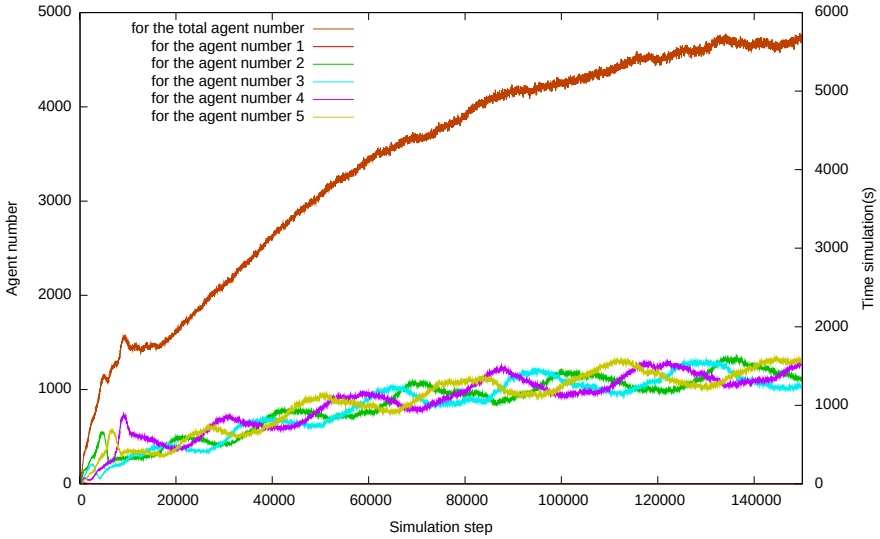
(e) Pas=89700



(f) Pas=104500



(g) Pas=118900



(h) Nombre d'agents en fonction du pas de simulation

FIGURE 2.5 – Simulation sur GPU de l'embryogenèse. Cette simulation montre un motif qui émerge d'une cellule initiale avec la capacité de se diviser, se différencier, produire et consommer des molécules. Les cellules sont dépendantes de molécules spécifiques selon leur type. Le motif obtenu est dynamique car le tissu est continuellement renouvelé mais sa taille reste constante. Le nombre de cellules qui meurent est égal au nombre de cellules qui se divisent. La vidéo de cette simulation se trouve à l'adresse suivante : `http://www.youtube.com/watch?v=v0q8TZeNbPA`

diminue, donc le nombre de molécules M_2 diminue aussi. La dégradation totale de molécules M_2 n'est pas immédiate, donc le nombre d'agents de type 1 durant ce temps continue à augmenter. Ce temps dure t pas de simulation.

La simulation sur GPU nous permet donc de calculer plus rapidement. La stabilisation des simulations

en fonction du nombre d'agents est donc confirmée grâce à la croissance finie de la spirale. Cette auto-organisation ne peut être déterminée automatiquement par le fait que le formalise utilisé est de nature différente (les énergies du Modèle de Potts Cellulaire et les comportements cellulaires comme la division), et par le fait qu'elle est de plusieurs natures (spatiale comme la formation en spirale et numérique par rapport à l'évolution du nombre d'agents).

Analyse des performances

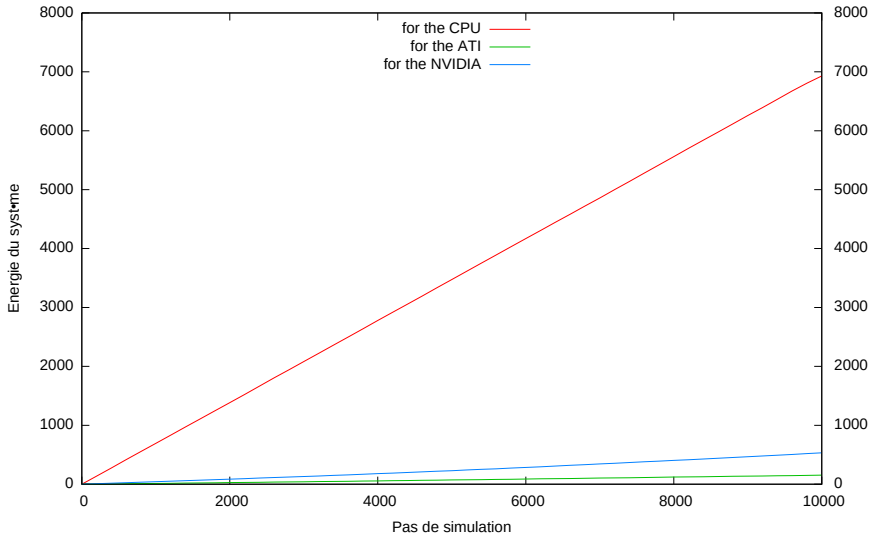
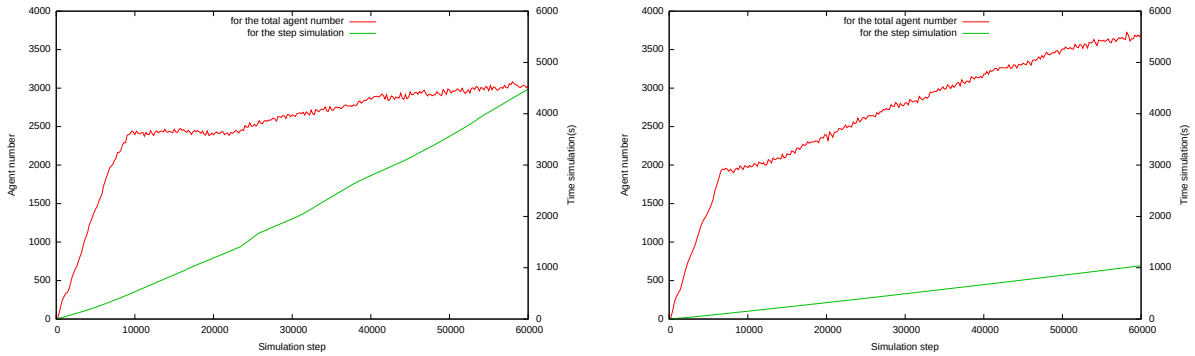


FIGURE 2.6 – Performance pour les premiers 10000 pas de simulation de l'embryogenèse.



(a) NVIDIA (b) ATI

FIGURE 2.7 – Performance pour les premiers 60000 pas de simulation de l'embryogenèse.

Les simulations ont été réalisées avec les matériels décrits précédemment. Dans toutes les simulations la taille d'un groupe de travail est de 32. Les figures 2.6 et 2.7 montrent les performances. Le GPU ATI est 45x plus rapide que le CPU et 4.3x plus rapide que le GPU NVIDIA. Le calcul de la diffusion des molécules est très bien adapté sur le GPU. Il y a peu des transferts de mémoire entre les différents niveaux de la mémoire dans le GPU. De plus, les items qui permettent la diffusion des molécules à chaque site de l'environnement sont regroupés dans des groupes de travail selon leur position. Ceci permet d'optimiser l'accès à la mémoire. Le nombre d'items utilisés pour la diffusion est égal au nombre de

sites de l'environnement, soit 589 824 items. Le temps de simulation dépend fortement du calcul de la diffusion. Ceci est la principale raison du fait que le GPU est beaucoup plus performant que le CPU, dans notre cas. Le GPU ATI est 4.3x plus rapide que la NVIDIA parce que l'ATI peut exécuter 4.5x plus de groupes de travail en parallèle que la NVIDIA.

Dans le CPU, la diffusion des molécules est coûteuse par rapport au temps de simulation. Le GPU peut modéliser et simuler des phénomènes multi-échelles, comme la diffusion des molécules et l'exécution de comportement cellulaire. Cette simulation montre l'intérêt d'utiliser le GPU. Nous pouvons noter que le GPU est 500x plus rapide que la simulation faite dans le chapitre 3 partie 2, où la mise en œuvre est en séquentiel, en JAVA. La section suivante étudie les différents motifs qui peuvent être obtenus à partir de notre SMA. Ceci n'est rendu possible que grâce aux performances maintenant obtenues via l'utilisation du GPU.

2.2 Étude avancée de l'auto-organisation

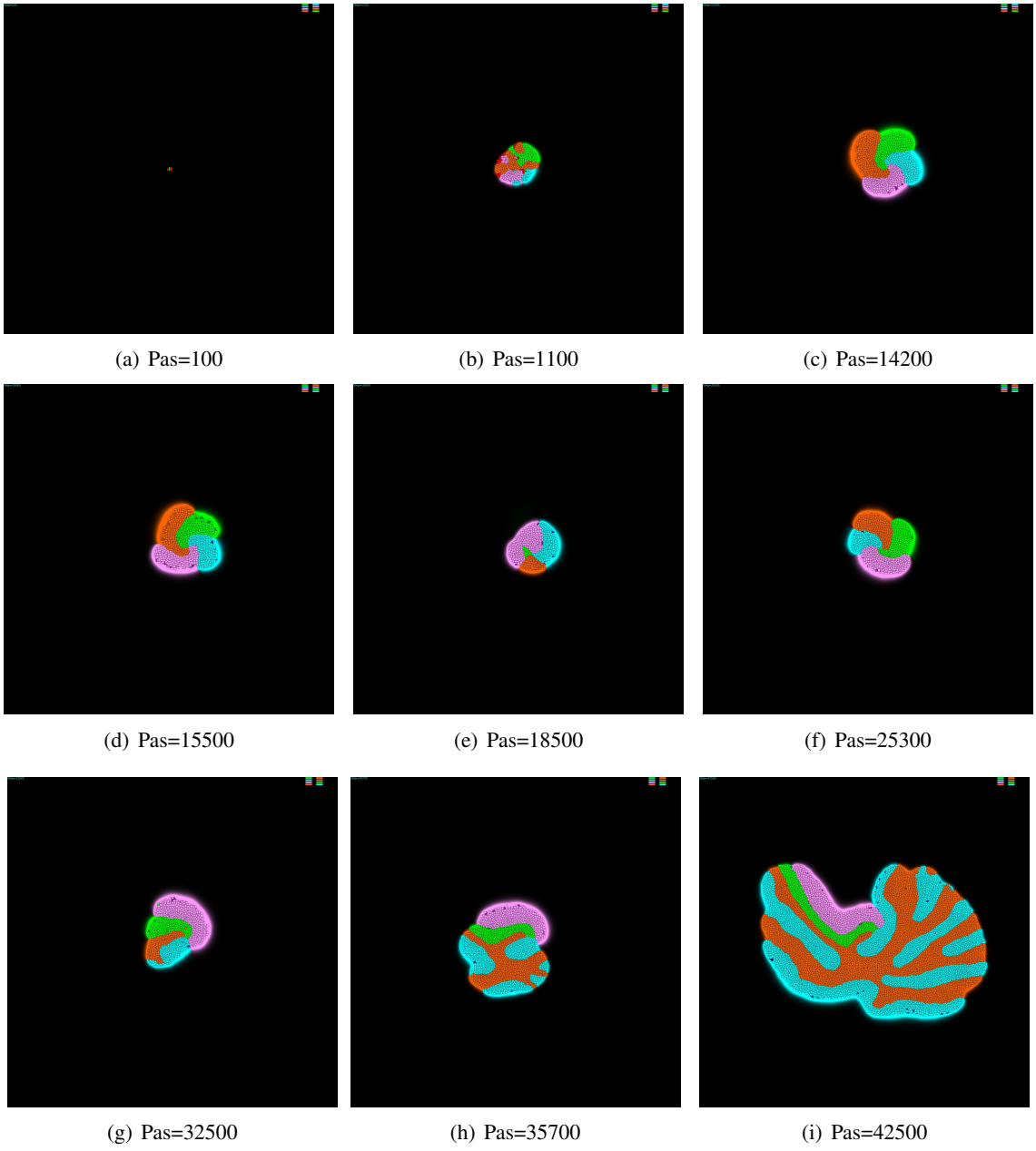
La programmation sur GPU nous permet d'être plus ambitieux pour les simulations que nous pouvons exécuter dans un temps raisonnable. Il est donc possible de tester des propriétés intéressantes comme la robustesse ou l'adaptation qui mettent en jeu un grand nombre de cellules. La robustesse dans les systèmes biologiques et l'adaptation sont des caractéristiques importantes. Nous montrons comment la théorie du darwinisme cellulaire permet de vérifier ces propriétés. Nous construisons deux modèles pour déterminer la robustesse de l'auto-organisation et l'adaptation des cellules : selon le changement des interactions dynamiques (2.2.1) et selon les valeurs des paramètres (2.2.2). Les modèles de simulation présentés jusqu'à présent vérifient le premier type d'auto-organisation défini par William Ross Ashby ($cf.$ chapitre 1 partie 1) : la formation d'un système auto-organisé. Le changement d'interaction permet de simuler le second type d'auto-organisation : le changement d'auto-organisation.

Les simulations présentées jusqu'à présent dans ce mémoire n'imposent pas d'auto-organisation car elle n'est pas connue. Les modèles de simulations sont théoriques. Dans ce qui suit, nous construisons un modèle de simulation où la dynamique et l'auto-organisation sont connues, afin de les reproduire (2.2.3). Ce modèle nous permet aussi de tester les énergies de forme cellulaire dans la mise en œuvre sur GPU et montre l'étendue des phénomènes qui peuvent être modélisés par notre système multi-agents.

2.2.1 Réponse aux changements d'interactions

Les systèmes biologiques sont robustes et s'adaptent à l'environnement dans lequel ils se trouvent. Cela implique que les interactions entre les parties du système changent. Nous modélisons la robustesse et l'adaptation de notre modèle de l'embryogenèse par le fait que les cellules changent leur dépendance par rapport aux molécules consommées et produites. Si le système est robuste et a un pouvoir d'adaptation, des motifs doivent émerger à chaque changement d'interactions, $i.e.$ une nouvelle organisation doit apparaître.

Les figures 2.8 et 2.9 montrent la simulation d'un modèle de 4 types de cellules où un changement des interactions (le type de molécules consommées par un type de cellule) est effectué dynamiquement tous les 15000 pas de simulation. Initialement, les interactions forment un cycle de dépendance entre les cellules. Nous retrouvons le motif déjà expliqué précédemment. Au pas de simulation 15000 les interactions changent pour donner un autre cycle de dépendance. Les cellules meurent et prolifèrent vers leur nouveau type cellulaire qui fournit les molécules dont elles ont besoin. La prolifération s'adapte donc à ce nouveau type d'interactions. Après la phase d'adaptation, le motif en spirale émerge. Tous les types cellulaires sont toujours présents. Au pas de simulation 30000, les interactions sont modifiées de telle sorte que le cycle de dépendances est brisé. Les cellules oranges et cyans sont en inter-dépendance. Les cellules oranges et cyans prolifèrent ensemble. Les cellules vertes et roses n'ont que très peu d'influence sur l'organisation



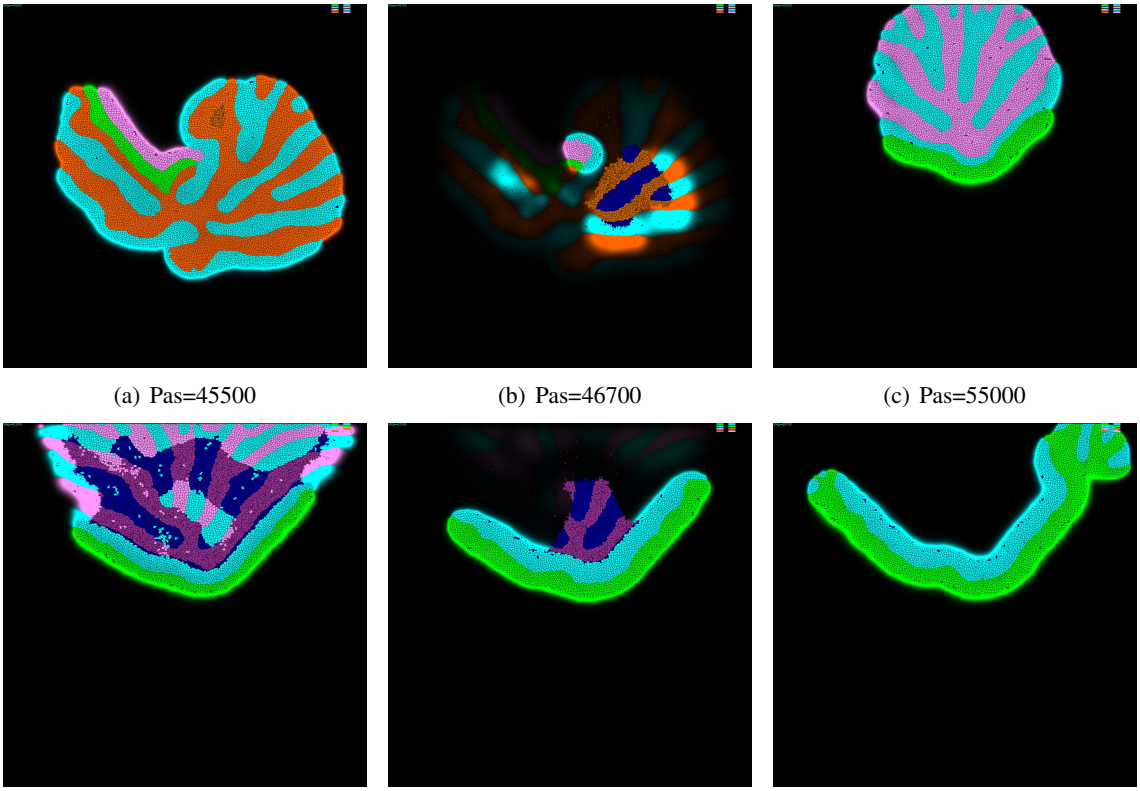
(a) Pas=100 (b) Pas=1100 (c) Pas=14200

(d) Pas=15500 (e) Pas=18500 (f) Pas=25300

(g) Pas=32500 (h) Pas=35700 (i) Pas=42500

FIGURE 2.8 – Simulation sur GPU de l'auto-organisation et adaptation des tissus cellulaires selon le changement d'interactions du pas 0 à 42500. Le changement des interactions (le type de molécules consommées par un type de cellule) est effectué dynamiquement tous les 15000 pas de simulation. Les figures 2.8(a), 2.8(b) et 2.8(c) montrent la simulation d'une dépendance cyclique des cellules par rapport aux molécules consommées et produites. Les figures 2.8(d), 2.8(e) et 2.8(f) montrent la simulation avec une autre dépendance cyclique des cellules. Les figures 2.8(g), 2.8(h) et2.8(i) montrent la simulation où le cycle de dépendance est cassé, les cellules oranges et cyans sont en interdépendance. La vidéo de cette simulation peut être vue à l'adresse suivante : `http://www.youtube.com/watch?v=Kccmds9nF4o`

du système. La prolifération entre les cellules oranges et cyans amène à des bifurcations de leur tissus. Le système s'est donc une nouvelle fois auto-organisé. Au pas de simulation 45000 l'inter-dépendance est entre les cellules roses et cyans. Les cellules oranges meurent car les cellules cyans ne sont plus en inter-dépendance avec elles. Les cellules cyans et roses prolifèrent ensemble. Au pas de simulation 60000



(a) Pas=45500 (b) Pas=46700 (c) Pas=55000

(d) Pas=61000 (e) Pas=62300 (f) Pas=65700

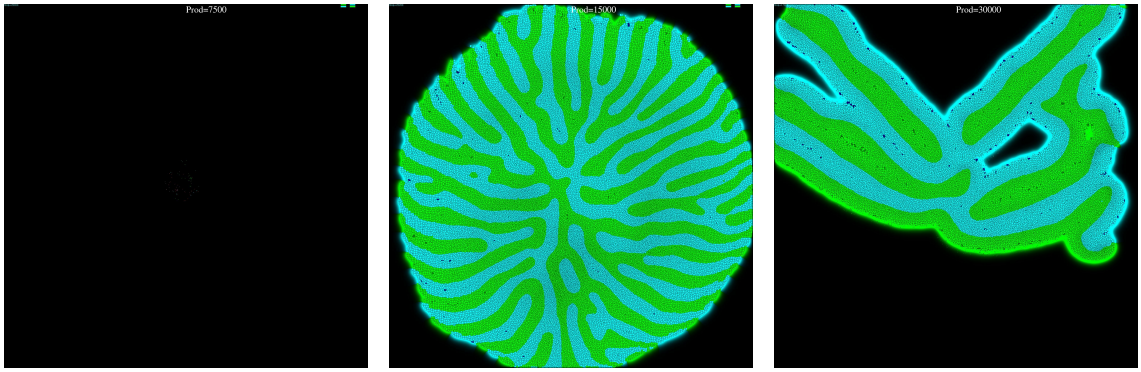
FIGURE 2.9 – Simulation sur GPU de l'auto-organisation et adaptation des tissus cellulaires selon le changement d'interactions du pas 45500 à 65700. Les figures 2.9(a), 2.9(b) et 2.9(c) montrent la simulation où les cellules roses et cyans sont en interdépendance, les cellules vertes dépendent des molécules produites par les cellules cyans. Les figures 2.9(d), 2.9(e) et 2.9(f) montrent la simulation où les cellules cyans et vertes sont en interdépendance. La vidéo de cette simulation peut être vue à l'adresse suivante : `http://www.youtube.com/watch?v=Kccmds9nF4o`

les cellules roses meurent car les cellules cyans ne sont plus en inter-dépendance avec elles, mais avec les cellules vertes.

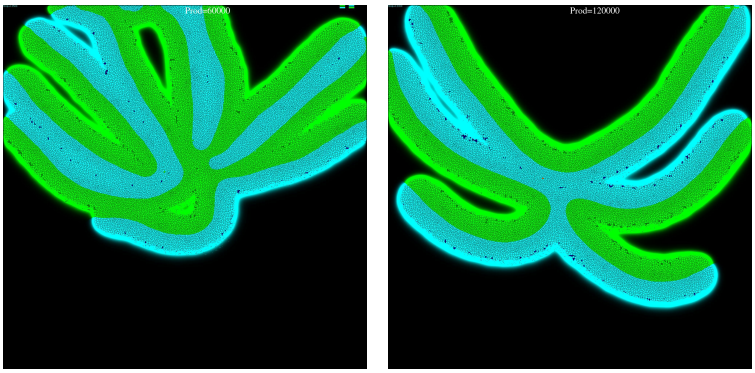
La théorie du darwinisme au niveau cellulaire permet aux tissus cellulaires de s'auto-organiser et de s'adapter. Différents motifs émergent en fonction des interactions. Cependant, une suite consécutive de changements amène à la destruction de certains tissus. Cela peut être vu comme une phase de maturité où l'organisation des cellules gagne en complexité en fonction des changement d'interactions et une phase de vieillissement où les cellules ne peuvent plus répondre aux perturbations, amenant à la mort des tissus. Ces 2 phases sont décrites par Henri Atlan dans sa définition de l'auto-organisation et des propriétés de la redondance ($cf.$ chapitre 1 partie 1). La section suivante décrit les effets de la production des molécules sur l'auto-organisation dans le cas d'une inter-dépendance entre deux cellules.

2.2.2 Réponse aux changements de valeurs des paramètres

La simulation présentée dans la section précédente montre que deux cellules en inter-dépendance prolifèrent ensemble et amènent à une disjonction de leur tissus. Ce motif est récurrent dans les systèmes biologiques (les tissus pulmonaires ou le cerveau en sont des exemples). Le darwinisme cellulaire permet donc d'expliquer la formation de ce motif. Nous étudions la robustesse de ce motif selon la valeur du paramètre de production des molécules par les cellules.



(a) Pas=28000, Prod=7500 (b) Pas=20200, Prod=15000 (c) Pas=17800, Prod=30000



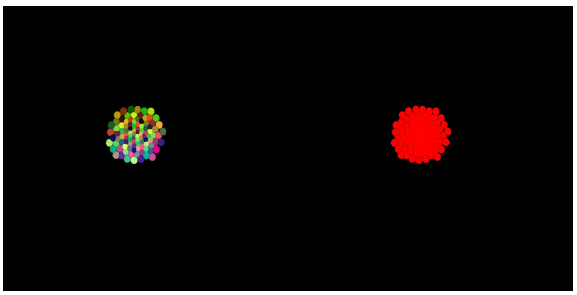
(d) Pas=13500, Prod=60000 (e) Pas=12300, Prod=120000

FIGURE 2.10 – Simulations sur GPU d'une inter-dépendance entre deux cellules selon le taux de production des molécules par les cellules. La figure 2.10(a) (*resp.* 2.10(b), 2.10(c), 2.10(d) et 2.10(e)) correspond à la formation d'un tissu depuis une cellule œuf avec une production de molécules de 7500 (*resp.* 15000, 30000, 60000 et 120000). La vidéo de cette simulation peut être vue à l'adresse suivante : `http://www.youtube.com/watch?v=54yYssjdq_0`

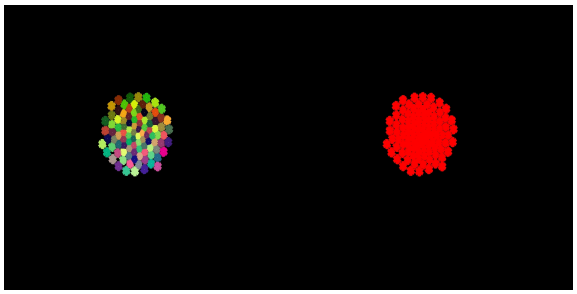
La figure 2.10 montre l'effet du paramètre de production de molécules sur le motif de branchement des tissus cellulaires. Pour un taux de production de 7500, les molécules ne sont pas en quantité suffisante pour nourrir les cellules qui en ont besoin. La formation du tissu ne pouvant se faire, les cellules des types en inter-dépendance meurent ($cf.$ la figure 2.10(a)). La simulation avec un taux de production de 15000 permet, à partir d'une cellule œuf, la prolifération de deux types de cellules qui sont en inter-dépendance. Les branchements des tissus sont retrouvés, et le tout forme un amas de cellules ($cf.$ la figure 2.10(b)). La largeur d'un tissu est relativement faible. La simulation avec le taux de production de 30000 amène à la prolifération de deux types cellulaires et au branchement des tissus ($cf.$ la figure 2.10(c)). La largeur d'un tissu est plus importante que dans la simulation avec le taux de production à 15000, car les cellules ont plus de molécules à consommer. Ceci amène à une mort cellulaire à l'intérieur du tissu car les cellules du milieu se trouvent trop éloignées des sources de molécules. Alors que dans la simulation avec le taux de production à 15000 l'espace entre les branchements d'un tissu est comblé par les cellules de l'autre type, ici cet espace n'est pas comblé. Les simulations avec les taux de production de 60000 ($cf.$ la figure 2.10(d)) et 120000 ($cf.$ la figure 2.10(e)) amènent à la formation de la même auto-organisation mais avec des largeurs de tissus de plus en plus importantes. Le darwinisme cellulaire permet d'être robuste aux paramètres, car les tissus cellulaires s'auto-organisent en fonction des valeurs de ces paramètres. Ces simulations donnent donc des arguments forts en faveur du darwinisme cellulaire pour expliquer la robustesse et l'adaptation des tissus cellulaires face à l'environnement.

2.2.3 La forme des cellules pour l'émergence de motif complexe

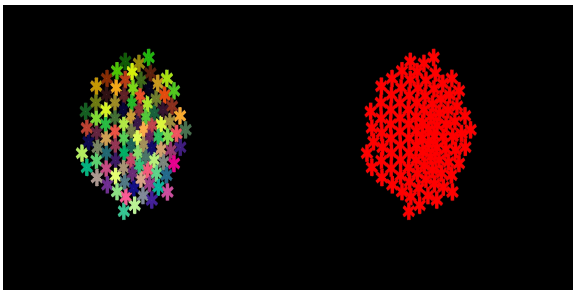
(a) Pas=50



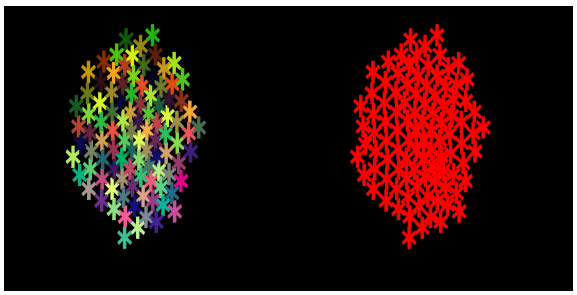
(b) Pas=1000



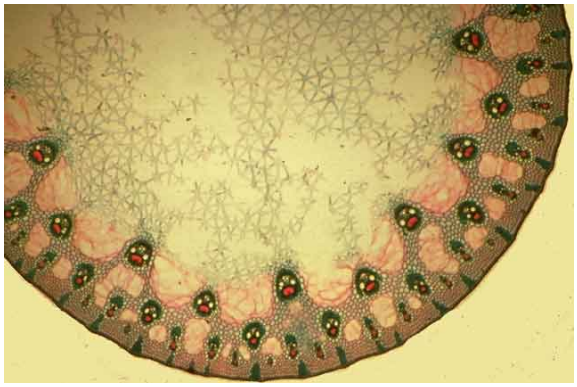
(c) Pas=2000



(d) Pas=5000



(e) Pas=7000



(f) Coupe transversale d'une tige de Jonc [29]

FIGURE 2.11 – Simulation sur GPU de la croissance des cellules se trouvant dans la tige de Jonc. La figure 2.11(a) montre l'état initial. Un ensemble de cellules sont en contact et leur forme est un hexagone. Durant la croissance, les parois des cellules s'allongent pour former un réseau étoilé, comme décrit dans la figure 2.11(f) extraite de [29]. Les figures 2.11(b), 2.11(c), 2.11(d) et 2.11(e) montrent la simulation de ce phénomène. La vidéo de cette simulation peut être vue à l'adresse suivante : `http://www.youtube.com/watch?v=IOeJU5YyRyM`

Les simulations précédentes montrent comment notre SMA peut simuler l'auto-organisation à l'aide de la théorie du darwinisme cellulaire. Nous construisons un modèle comme une perspective pour monter l'étendue des simulations que notre SMA peut exécuter. Nous nous sommes inspirés de la croissance des cellules dans la tige de Jonc où une organisation étoilée entre les cellules est réalisée. Ce réseau étoilé impose un espace gazeux dans le tissu pour permettre la circulation du gaz à travers la tige.

La figure 2.11 montre une simulation de ce phénomène, le modèle de simulation utilisé impose aux cellules une forme hexagonale (à l'aide de l'énergie de forme définie dans le chapitre 2 partie 2). Au cours de la simulation la longueur des ressorts qui construisent la forme donnée aux cellules,

s'allonge pour permettre le réseau étoilé. Des énergies de contacts favorisent l'adhésion entre les parois des cellules. La simulation montre que, depuis des cellules hexagonales, l'extension de leurs parois permet une auto-organisation complexe. La figure 2.11(e) montre qu'un réseau étoilé émerge.

Cette simulation valide les énergies de forme mises en œuvre sur le GPU et montre que d'autres auto-organisations peuvent être modélisées à l'aide de notre système. L'énergie de forme définie dans cette thèse peut servir de modèle dans de nombreux phénomènes biologiques. La suite de ce mémoire présente les conclusions, les discussions et les perspectives du SMA mis en œuvre dans cette thèse.

Discussion & Conclusion

Dans ce mémoire nous avons étudié l'auto-organisation entre les cellules à travers les modèles de calculs et plus particulièrement les automates cellulaires et les systèmes multi-agents. L'auto-organisation des cellules permet d'expliquer les phénomènes de morphogenèse notamment l'embryogenèse. L'embryogenèse se traduit par le fait qu'à partir d'une cellule œuf et de sa division, sa différentiation et les interactions entre des cellules, une organisation des tissus se produit. Cette organisation peut être détectée par les motifs qui émergent. Nous avons montré que les systèmes multi-agents sont de bons candidats pour modéliser de tels systèmes où chaque entité est autonome et permet une représentation explicite de ces entités.

Plusieurs théories de l'embryogenèse existent, certaines expliquent ce phénomène par le fait que la cellule œuf a un gradient de concentration de molécules et ses différentes phases de division amènent donc à des cellules différenciées qui communiquent entre elles. Ces communications étant déterminées $a\ priori$ par le programme génétique. Cette théorie repose donc sur un procédé déterministe. Une autre théorie est le darwinisme cellulaire qui explique l'embryogenèse par des procédés stochastiques, où les cellules s'auto-organisent car si elles trouvent des molécules spécifiques, elles survivent, sinon elles meurent. Ces molécules sont produites par d'autres cellules. Même si la différentiation cellulaire est aléatoire, l'auto-organisation entre les cellules est possible par la sélection naturelle. La mise en œuvre du système multi-agents défini dans ce mémoire et les simulations faites donnent des arguments en faveur de cette seconde théorie.
Nous montrons comment :

- les tissus cellulaires peuvent être renouvelés continuellement.
- le tri des tissus est possible.
- cette théorie amène à une prolifération structurée des cellules.

Différents motifs, que ce soient des spirales ou des branchements (des motifs qui sont trouvées dans de nombreux tissus cellulaires), émergent des simulations faites. Nous avons vérifié aussi que cette théorie permet au tissu cellulaire de s'auto-organiser et s'auto-adapter selon les interactions et les valeurs des paramètres. Ces propriétés sont des caractéristiques importantes des phénomènes biologiques qui peuvent être donc expliquées par le darwinisme cellulaire.

Pour construire notre système multi-agents, nous avons étudié et étendu les modèles de cellules existants. Le modèle de cellule qui nous a semblé le plus approprié pour modéliser l'auto-organisation cellulaire est le Modèle de Potts Cellulaire. Ce modèle permet de représenter à la fois les cellules d'une manière réaliste (leur forme est contrainte par un volume et une surface) et l'exécution de ce modèle se fait dans un temps raisonnable. Le Modèle de Potts Cellulaire repose sur des modèles de physiques statistiques qui définissent une probabilité de passer d'un état à un autre. Cette probabilité dépend de la différence d'énergie entre les états. Ceci nous permet dans certains cas de quantifier l'auto-organisation. Dans d'autres cas, c'est le regard de l'observateur qui permet de le faire. Le Modèle de Potts Cellulaire ne couvre pas tous les comportements cellulaires essentiels, comme la production et la consommation de molécules ou la forme des cellules.
Pour modéliser la théorie du darwinisme cellulaire, ou plus généralement les phénomènes d'auto-

organisation cellulaire, nous avons mis en œuvre :

- un bilan énergétique de la cellule pour modéliser la sélection naturelle.
- la possibilité que la cellule cible une forme générique et dynamique.

Ces deux comportements ont permis la modélisation de la formation des tissus. La forme d'un tissu émerge des formes des cellules et de la sélection naturelle au niveau cellulaire.

D'un point vue informatique la thèse a apporté des améliorations dans la mise en œuvre des systèmes multi-agents. L'auto-organisation cellulaire implique qu'un grand nombre de cellules soit mis en jeu et donc le temps de simulation devient critique. Le temps de simulation augmente au moins de façon linéaire lorsque le système multi-agents est exécuté de façon séquentielle (les agents sont exécutés les uns après les autres). Nous avons montré dans ce mémoire que la programmation parallèle des agents sur le GPU (Graphics Proccessing Unit) permet d'exécuter des agents sur chaque cœur de la carte graphique ($e.g.$ 128 pour une carte moyenne gamme). Les performances augmentent considérablement jusqu'à 45x comparé à une mise en œuvre mais sur le CPU (Central Processing Unit). La programmation sur GPU pose des contraintes fortes notamment sur la gestion de la mémoire. La mémoire ne peut être allouée dynamiquement. Nous avons proposé des solutions pour créer dynamiquement des agents sur le GPU tout en libérant la mémoire lorsque les agents ne sont plus disponibles. Cette thèse a donc amélioré la mise en œuvre des systèmes multi-agents en étudiant des problématiques biologiques.

Perspectives

Nous distinguons trois types de perspectives des travaux présentés dans ce mémoire :

1. Premièrement sur le Modèle de Potts Cellulaire. Nous avons testé des théories biologiques qui portent sur le darwinisme cellulaire, mais nous n'avons pas pris en compte les théories de l'embryogenèse où un gradient de molécules est contenu dans la cellule. Il serait intéressant de faire évoluer le modèle de cellules défini dans cette thèse (le MorphoPotts) pour qu'il prenne en compte ce gradient. Le Modèle de Potts Cellulaire définit une température qui peut être vue comme l'énergie apportée à la cellule pour se déformer. Dans le MorphoPotts, un bilan énergétique est défini. Une amélioration du Modèle de Potts Cellulaire serait de remplacer la température par ce bilan. Ainsi, plus une cellule a pu emmagasiner de l'énergie, plus elle peut utiliser cette énergie pour atteindre un état plus satisfaisant. De cette manière, la probabilité qu'une cellule se déforme n'est plus la même pour chaque cellule, mais elle dépend de l'environnement dans lequel se trouve la cellule.
2. Deuxièmement sur les applications biologiques. Nous avons étudié l'auto-organisation notamment dans le phénomène de l'embryogenèse. D'autres phénomènes pourraient être étudiés avec notre système, comme le cancer et l'origine de la vie. L'étude du cancer avec le Modèle de Potts Cellulaire [75] et l'étude de la prolifération d'un tissu dans un autre tissu en fonction des énergies de contacts [83] ne sont pas nouveaux. Cependant, d'autres théories expliquent la prolifération de tumeur avec le darwinisme cellulaire [47]. Notre système pourrait simuler cette théorie dans une voie plus réaliste. Le modèle de cellules défini dans cette thèse est assez complexe, car il est défini par un volume, une surface et une forme générique. Des théories de l'origine de la vie caractérisent le type des proto-cellules (cellules primitives) en fonction des molécules présentes sur leur membrane [64]. Les réactions chimiques impliquent que les proto-cellules changent de type. Cette étude met en évidence l'évolution des proto-cellules. Dans ces travaux la division des proto-cellules est une probabilité définie en fonction des molécules qui sont contenues dans les proto-cellules. Ceci est une abstraction du fait que les molécules permettent la déformation des cellules et peuvent amener à sa division. Nous sommes en train de reprendre le modèle décrit dans [64] en définissant une nouvelle énergie pour prendre en compte la déformation des membranes par les molécules. Le problème principal de cette mise en œuvre est la détection automatique de la division des proto-cellules dans un temps raisonnable.
3. Troisièmement sur la programmation sur GPU. Nous avons proposé dans cette thèse une mise en œuvre sur GPU de notre système multi-agents. Nous avons géré le problème d'allocation mémoire. Cependant, l'accès à la mémoire pourrait être amélioré en étudiant l'impact du coût de transfert entre les différents niveaux de mémoire et l'accès à ces niveaux de mémoire. En fonction du matériel et des algorithmes, il peut être plus efficace de transférer des mémoires du niveau global au niveau local, car l'accès mémoire au niveau local est plus rapide qu'au niveau global. Ceci dépend fortement du matériel car la taille de la mémoire locale est différente. Nous avons proposé un algorithme pour défragmenter la mémoire. Cet algorithme est naïf et il serait intéressant de le paralléliser sur le GPU. Finalement, ces problèmes de mise en œuvre sur GPU ne devraient pas être

visibles au programmeur. Il serait utile, comme cela est fait dans FlameGPU [66], de construire un meta-modèle de notre mise en œuvre pour abstraire la programmation sur GPU dans notre contexte (Domain Specific Languages).

Bibliographie

[1] P. Agarwal. The cell programming language. *Artificial Life*, 2(1) :37–77, 1994.

[2] A.R.A. Anderson, MAJ Chaplain, and K.A. Rejniak. *Single-Cell-Based Models in Biology and Medicine*. Birkhauser, 2007.

[3] S. Andrews. Smoldyn, A spatial stochastic simulator (Documentation). `http://www.smoldyn.org`, 2008.

[4] W.R. Ashby. Principles of the self-organizing dynamic system. *The Journal of general psychology*, 37(2) :125, 1947.

[5] W.R. Ashby. Principles of the self-organizing system. *Principles of Self-organization*, pages 255–278, 1962.

[6] H. Atlan. *Entre le cristal et la fumée : essai sur l'organisation du vivant*. Éditions du Seuil, 1979.

[7] P. Ballet, P. Redou, S. Tripodi, and V. Rodin. Lambda phage switch : a spatialised and stochastic approach. In *Tutorials of the Spring School on Modelling Complex Biological Systems in the Context of Genomics*, March 2009.

[8] P. Ballet and P. Tracqui. Migration de cellules virtuelles déformables- modélisation biomécanique multiagent de la migration cellulaire. *RSTI série TSI (Numero Spécial Modélisation et simulation pour la post-génomique, Edition Lavoisier - Hermes Sciences - volume 26*, February 2007.

[9] P. Ballet, S. Tripodi, and V. Rodin. Morphoblock programming : a way to model and simulate morphogenesis of multicellular organisms. *Journal of Biological Physics and Chemistry ISSN 1512-0856*, pages 37–44, March 2009.

[10] Hendrik Barendregt. *The Lambda Calculus*. North-Holland, Amsterdam, 1984.

[11] J. Barnard, J. Whitworth, and M. Woodward. Communicating X-machines. *Information and Software Technology*, 38(6) :401–407, 1996.

[12] Jean-Louis Basdevant. *Principes variationnels & Dynamique*. Vuibert, 2005.

[13] C. Bernon, V. Chevrier, V. Hilaire, and P. Marrow. Applications of Self-Organising Multi-Agent Systems : An Initial Framework for Comparison. *INFORMATICA-LJUBLJANA-*, 30(1) :73, 2006.

[14] A. Bitbol-Hespériès and J.P. Verdet. *René Descartes-Le monde*. Seuil, 1996.

[15] A.W. Burks, A. Walter, and J. Von Neumann. *Theory of self-reproducing automata*. University of Illinois Press, 1966.

[16] N. Cannata, F. Corradini, E. Merelli, A. Omicini, and A. Ricci. An Agent-Oriented Conceptual Framework for Systems Biology. *LNCS*, 3737 :105, 2005.

[17] C. Chaouiya, E. Remy, P. Ruet, and D. Thieffry. Qualitative modelling of genetic networks : From logical regulatory graphs to standard petri nets. *Applications and Theory of Petri Nets 2004*, pages 137–156, 2004.

[18] T. Cickovski, K. Aras, M. Swat, R.M.H. Merks, T. Glimm, H.G.E. Hentschel, M.S. Alber, J.A. Glazier, S.A. Newman, and J.A. Izaguirre. From Genes to Organisms Via the Cell : A Problem-Solving Environment for Multicellular Development. *Computing in Science & Engineering*, 9(4) :50–60, 2007.

[19] B. Corbara, A. Drogoul, D. Fresneau, and S. Lalande. Simulating the sociogenesis process in ant colonies with manta. In *Toward A Practice of Autonomous Systems : Proceedings of the First European Conference on Artificial Life*, pages 224–235, 1993.

[20] F. Corblin, L. Bordeaux, E. Fanchon, Y. Hamadi, and L. Trilling. Connections and integration with SAT solvers : A survey and a case study in computational biology. *Hybrid Optimization*, pages 425–461, 2011.

[21] F. Corblin, S. Tripodi, E. Fanchon, D. Ropers, and L. Trilling. A declarative constraint-based method for analyzing discrete genetic regulatory networks. *Biosystems*, 98(2) :91–104, 2009.

[22] Y. Demazeau. From interactions to collective behaviour in agent-based systems. In *Proceedings of the First European conference on cognitive science. Saint-Malo,France*, pages 117–132, 1995.

[23] . Desmeulles. *Réification des interactions pour l'expérience in virtuo de systèmes biologiques multi-modèles.* PhD thesis, Université de Bretagne Occidentale, December 2006.

[24] A. Deutsch and S. Dormann. *Cellular automaton modeling of biological pattern formation : characterization, applications, and analysis*. Birkhauser, 2005.

[25] R. Doursat. Organically grown architectures : Creating decentralized, autonomous systems by embryomorphic engineering. *Organic Computing*, pages 167–199, 2008.

[26] A. Drogoul. When Ants Play Chess (Or Can Strategies Emerge From Tactical Behaviors ?). In *From Reaction to Cognition : 5th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW'93, Neuchatel, Switzerland, August 25-27, 1993. Selected Papers*. Springer, 1995.

[27] A. Drogoul and J. Ferber. Multi-agent simulation as a tool for modeling societies : Application to social differentiation in ant colonies. In *MAAMAW*, pages 3–23, 1992.

[28] L. Dupuy, J. Mackenzie, T. Rudge, and J. Haseloff. A system for modelling cell–cell interactions during plant morphogenesis. *Annals of botany*, 101(8) :1255, 2008.

[29] Biologie et Multimédia Université Pierre et Marie Curie UFR de Biologie. L'espace gazeux des végétaux. `http://shop.maxonmotor.com/ishop/article/article/148866.xml`, page consultée le 06/06/2011, 2011.

[30] J. Ferber. *Les Systèmes multi-agents : vers une intelligence collective*. InterEditions, 1995.

[31] J. Ferber, F. Michel, and J. Baez. ACRE : Integrating Environments with Organizations. In *Environments for Multi-Agent Systems : First International Workshop, E4MAS 2004, New York, NY, July 19, 2004 : Revised Selected Papers*. Springer, 2005.

[32] J. Fleureau, M. Garreau, AI Hernandez, A. Simon, and D. Boulmier. Multi-Object and NDSegmentation of Cardiac MSCT Data Using SVM Classifiers and a Connectivity Algorithm. *Computers in Cardiology 2006, Proceedings*, 33 :817–820, 2006.

[33] F. García-Sánchez, J. Tomás T. Fernández-Breis, R. Valencia-García, J.M. Gómez, and R. Martínez-Béjar. Combining semantic web technologies with multi-agent systems for integrated access to biological resources. *Journal of biomedical informatics*, 41(5) :848–859, May 2008.

[34] M. Gardner. Mathematical games : The fantastic combinations of John Conway's new solitaire game 'Life'. *Scientific American*, 223(4) :120–123, 1970.

[35] J.A. Glazier and F. Graner. Simulation of the differential adhesion driven rearrangement of biological cells. *Physical Review E*, 47(3) :2128–2154, 1993.

[36] F. Graner and J.A. Glazier. Simulation of biological cell sorting using a two-dimensional extended Potts model. *Physical Review Letters*, 69(13) :2013–2016, 1992.

[37] H.G..E Hentschel, T. Glimm, J.A. Glazier, and S.A. Newman. Dynamical mechanisms for skeletal pattern formation in the vertebrate limb. *Proceedings-Royal Society of London. Biological sciences*, 271(1549) :1713–1722, 2004.

[38] G.T. Herman and W.H. Liu. The daughter of celia, the french flag, and the firing squad. *Simulation*, 21(2) :33, 1973.

[39] C. Hewitt, P. Bishop, and R. Steiger. A universal modular actor formalism for artificial intelligence. In *3rd International Joint Conference on Artificial Intelligence*, pages 235–245, 1973.

[40] G. Hutzler. Cellular Automata and Agent-Based approaches for the modelling and simulation of biological systems : application to the lambda phage, 2006.

[41] J.A. Izaguirre, R. Chaturvedi, C. Huang, T. Cickovski, J. Coffland, G. Thomas, G. Forgacs, M. Alber, G. Hentschel, SA Newman, and J. A. Glazier. Compucell, a multi-model framework for simulation of morphogenesis. *Bioinformatics*, 20(7) :1129–1137, 2004.

[42] D.B. Kirk and W.H. Wen-mei. *Programming massively parallel processors : A Hands-on approach*. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 2010.

[43] H. Kitano. *Foundations of Systems Biology*. The MIT Press, October 2001.

[44] S. Kooijman. *Dynamic energy and mass budgets in biological systems*. Cambridge Univ Pr, 2000.

[45] J.U. Kreft, G. Booth, and J.W. Wimpenny. BacSim, a simulator for individual-based modelling of bacterial colony growth. *Microbiology*, 144(12) :3275–3287, 1998.

[46] J.J Kupiec. A Darwinian theory for the origin of cellular differentiation. *Molecular and General Genetics MGG*, 255(2) :201–208, 1997.

[47] B. Laforge, D. Guez, M. Martinez, and J.J. Kupiec. Modeling embryogenesis and cancer : an approach based on an equilibrium between the autostabilization of stochastic gene expression and the interdependence of cells for proliferation. *Progress in biophysics and molecular biology*, 89(1) :93–120, 2005.

[48] C.G. Langton. Self-reproduction in cellular automata. *Physica D : Nonlinear Phenomena*, 10(1-2) :135–144, 1984.

[49] C.G. Langton. Studying artificial life with cellular automata* 1. *Physica D : Nonlinear Phenomena*, 22(1-3) :120–149, 1986.

[50] D.B. Lenat. Beings : Knowledge as interacting experts. In *IJCAI*, pages 126–133, 1975.

[51] C.B. Les Gasser and N. Herman. MACE : A Flexible Testbed for Distributed AI Research. *Distributed Artificial Intelligence*, 1987.

[52] V.R. Lesser and D.D Corkill. The Distributed Vehicle Monitoring Testbed : A Tool for Investigating Distributed Problem Solving Networks. *AI Magazine*, 4(3) :15–33, 1983.

[53] V.R. Lesser, R.D. Fennell, L.D. Erman, and D.R Reddy. Organization of the Hearsay-II Speech Understanding System. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-23(1) :11–24, 1975.

[54] M. M. Lysenko and R.M. DÕSouza. A framework for megascale agent based model simulations on graphics processing units. *Journal of Artificial Societies and Social Simulation*, 11(4) :10, 2008.

[55] R. Mach and F. Schweitzer. Modeling Vortex Swarming In Daphnia. *Bulletin of Mathematical Biology*, 69(2) :539–562, 2007.

[56] A.K. Mackworth. Consistency in networks of relations. *Artificial intelligence*, 8(1) :99–118, 1977.

[57] S. Marée. *From Pattern Formation to Morphogenesis*. PhD thesis, Utrecht University, 2000.

[58] W.S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology*, 5(4) :115–133, 1943.

[59] E. Merelli, G. Armano, N. Cannata, F. Corradini, M. d'Inverno, A. Doms, P. Lord, A. Martin, L. Milanesi, S. Moller, et al. Agents in bioinformatics, computational and systems biology. *Briefings in Bioinformatics*, 8(1) :45, 2007.

[60] R.M.H. Merks, S.V. Brodsky, M.S. Goligorksy, S.A. Newman, and J.A. Glazier. Cell elongation is key to in silico replication of in vitro vasculogenesis and subsequent remodeling. *Developmental biology*, 289(1) :44–54, 2006.

[61] R.M.H. Merks and J.A. Glazier. Dynamic mechanisms of blood vessel growth. *Nonlinearity*, 19 :C1–C10, 2006.

[62] A. Munshi. OpenCL. *Parallel Computing on the GPU and CPU, SIGGRAPH*, 2008.

[63] C. Penaloza, L. Lin, R.A. Lockshin, and Z. Zakeri. Cell death in development : shaping the embryo. *Histochemistry and cell biology*, 126(2) :149–158, 2006.

[64] D. Raine and Symons S. Emergence of pregenetic coding in a prebiotic ecology. *Journal of Biological Physics and Chemistry*, 9(3) :93–99, 2010.

[65] A.S. Rao and M.P. Georgeff. BDI Agents : From Theory to Practice. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 312–319. San Francisco, CA, 1995.

[66] P. Richmond, S. Coakley, and D. Romano. Cellular level agent based modelling on the graphics processing unit. In *2009 International Workshop on High Performance Computational Systems Biology*, pages 43–50. IEEE, 2009.

[67] P. Richmond, S. Coakley, and D.M. Romano. A high performance agent based modelling framework on graphics card hardware with CUDA. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 1125–1126. International Foundation for Autonomous Agents and Multiagent Systems, 2009.

[68] V. Rodin. Contribution à l'utilisation de l'informatique en biologie. mémoire d'habilitation à diriger des recherches, université de rennes i. Technical report, December 2004.

[69] D. Ropers, H. De Jong, M. Page, D. Schneider, and J. Geiselmann. Qualitative simulation of the carbon starvation response in Escherichia coli. *BioSystems*, 84(2) :124–152, 2006.

[70] E. Simao, E. Remy, D. Thieffry, and C. Chaouiya. Qualitative modelling of regulated metabolic pathways : application to the tryptophan biosynthesis in E. Coli. *Bioinformatics*, 21(suppl 2) :ii190, 2005.

[71] E. Sklar. NetLogo, a multi-agent simulation environment. *Artificial Life*, 13(3) :303–311, 2007.

[72] R.H. Smallwood, W.M.L. Holcombe, and D.C. Walker. Development and validation of computational models of cellular interaction. *Journal of Molecular Histology*, 35(7) :659–665, 2004.

[73] R.G. Smith. The contract net protocol : High-level communication and control in a distributed problem solver. *IEEE Trans. Computers*, 29(12) :1104–1113, 1980.

[74] H. Soula, C. Robardet, F. Perrin, S. Gripon, G. Beslon, and O. Gandrillon. Modeling the emergence of multi-protein dynamic structures by principles of self-organization through the use of 3DSpi, a multi-agent-based software. *BMC Bioinformatics*, 6(1) :228, 2005.

[75] E.L. Stott, N.F. Britton, J.A. Glazier, and M. Zajac. Stochastic simulation of benign avascular tumour growth using the potts model. *Mathematical and computer modelling*, 30(5-6) :183–198, 1999.

[76] J.J. Tapia and R.M. D'Souza. Parallelizing the Cellular Potts Model on graphics processing units. *Computer Physics Communications*, pages 857–865, 2010.

[77] Jacques Tisseau. Réalité virtuelle : autonomie in virtuo. Technical report, Habilitation à Diriger des Recherches - Université de Rennes I, December 2001.

[78] S. Tripodi, P. Ballet, and V. Rodin. *Computational energetic model of morphogenesis based on multi-agent Cellular Potts Model. Book chapter in Advances in Computational Biology*, pages 685–692. Advances in Experimental Medicine and Biology. Research book series Springer, 2010.

[79] S. Tripodi, P. Ballet, and V. Rodin. Multilayer cell proliferation with an adaptation of a cellular potts model. *Journal of Biological Physics and Chemistry*, 10 :33–38, March 2010.

[80] S. Tripodi, P. Ballet, and V. Rodin. Self-organization of a virtual multicellular organism by adding a shape model in the cellular potts model. In *Twelfth International Conference on the Synthesis and Simulation of Living Systems - ALIFE XII*, pages 249–256, August 2010.

[81] A.M. Tueixg. On computable numbers, with an application to the Entscheidungsproblem. *J. of Math*, 58 :345–363, 1936.

[82] V. Vallurupalli and C. Purdy. Agent based modeling and simulation of biomolecular reactions. *Scalable Computing : Practice and Experience*, 8(2), 2007.

[83] B. Vasiev, A. Balter, M. Chaplain, J.A. Glazier, and C.J. Weijer. Modeling gastrulation in the chick embryo : formation of the primitive streak. *PloS one*, 5(5) :e10571, 2010.

[84] J. Von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Champaign, IL, USA, 1966.

[85] D. Weyns, A. Omicini, and J. Odell. Environment as a first class abstraction in multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 14(1) :5–30, 2007.

[86] S. Wolfram. Universality and complexity in cellular automata. *Physica D : Nonlinear Phenomena*, 10(1-2) :1–35, 1984.

[87] F. Y. Wu. The potts model. *Rev. Mod. Phys.*, 54(1) :235–268, Jan 1982.

Résumé

L'auto-organisation entre les cellules permet d'expliquer la morphogenèse des tissus cellulaires, comme le phénomène de l'embryogenèse. Cependant, il n'existe pas de consensus sur la nature des interactions entre les cellules amenant à cette auto-organisation. La modélisation et la simulation $in\ silico$ offrent un support formel aidant le biologiste dans sa compréhension du phénomène et donnent des arguments en faveur d'une théorie ou d'une autre. La mise en œuvre informatique de processus biologiques permet, en retour, d'améliorer les modèles informatiques existants. Les systèmes multi-agents sont des modèles informatiques qui représentent chaque entité (agent) du système de manière explicite. Les agents sont exécutés de manière autonome et en interaction avec les autres. La mise en œuvre d'un système multi-agents permet de simuler des agents cellules où les interactions reposent sur la consommation et la production de molécules, mais aussi sur l'adhésion et la pression que les cellules exercent les unes sur les autres. Un agent cellule est ici basé sur la cellule définie dans le Modèle de Potts Cellulaire. Ce modèle a été étendu (MorphoPotts) pour permettre aux cellules de cibler une forme générique et dynamique et, de définir un bilan énergétique. La théorie du darwinisme cellulaire, une théorie originale de l'embryogenèse, a été simulée à l'aide du MorphoPotts. Des tissus ont émergé depuis une cellule œuf. Ces tissus sont cohérents car ils se renouvellent en continu et ont une forme reconnaissable. Pour vérifier si les interactions entre les MorphoPotts permettent au système de s'adapter et de s'auto-organiser, les performances des systèmes multi-agents ont dû être améliorées. Nous montrons que la programmation sur les cartes graphiques (GPU) amène à des gains de performance importants. Les simulations faites sur les cartes graphiques montrent comment un darwinisme cellulaire permet aux tissus cellulaires de s'auto-organiser et de s'adapter en réponse aux événements extérieurs.

Mots-clés : Système multi-agents, Modèle de Potts Cellulaire, Morphogenèse, Auto-organisation, Programmation sur cartes graphiques.

Abstract

The self-organization between the cells gives an explanation of the cell tissues morphogenesis, like the phenomenon of embryogenesis. Meanwhile, there is no consensus on the nature of the interactions between the cells leading to this self-organization. On one hand, the $in\ silico$ modelisation and simulation offers a formal support to help the biologist in his understanding of the phenomenon and gives arguments in favour of a theory or of another one. The computer science implementation of biological process allows, on the other hand, to improve the existing computer science models. The multi-agent systems are computer science models which represent each entity (agent) of a system in an explicit way. The agents are executed in an autonomous way and in interaction with the others. The implementation of a multi-agent system allows the simulation of cell agents, where interactions are based on the consumption and production of molecules, but also on the adhesion and pressure the cells exert on the others. A cell agent is based on the cell defined in the Cellular Potts Model. This model has been extended (MorphoPotts) in order to allow the cells to reach a generic and dynamic shape and to define an energy balance. The theory of Darwin at cellular level, an original theory of the embryogenesis, has been simulated via MorphoPotts where tissues emerge from one stem cell. These tissus are coherent because they have a continue renewal and a recognizable shape. To verify if the interactions between the MorphoPotts allow the system to self-organize and self-adapt, the performances of the multi-agent systems were enhanced. We show that the graphics processing unit (GPU) programming leads to considerable performance gains. The simulations done on the GPU show that a cellular darwinism allows the cell tissue to self-organize and self-adapt in reply to exterior events.

Keywords : Multi-agent system, Cellular Potts Model, Morphogenesis, Self-organization, Graphics Processing Unit programming.