

Travail d'étude et d'implémentation

Simulation d'un écosystème aquatique
à l'aide de MOBIDYC et d'autres

Auteur Robert Fisch

Date 19.11.2003

Avant propos

Étant donné que la formulation de ce sujet laisse place à beaucoup de choix et d'interprétations personnelles, j'avais du mal à trouver un début. J'ai donc décidé de commencer à jeter un coup d'œil aux simulations d'individus en générales, de plonger ensuite dans MOBIDYC avant de faire l'analyse et une implémentation possible de l'écosystème aquatique proposé.

Pour toutes ces étapes, j'ai essayé de noter dans tous les détails mon travail et de mettre sur papier les idées y relatives que j'ai eues. J'essaye de motiver le plus que possible les différents choix que j'ai faits et d'expliquer comment j'ai abouti à quel résultat.

Les simulations d'individus présentent un domaine qui m'intéresse beaucoup. En effet, j'avais gagné en 1998 le Concours Informatique Luxembourgeois [CTE03]¹ lors duquel le logiciel à réaliser pendant la finale consistait en une simulation d'araignées mécaniques avec différentes contraintes et relations (cf. «Spiders 98» en annexe).

¹ <http://cil.cte.lu>

Table des matières

1. Simulations monoexétron	5
1.1. Le terrain	5
1.2. Simulation incrémentale simple [SIS]	6
1.3. Simulation incrémentale avec transitions [SIT]	8
1.4. Simulation ordonnancée avec transactions [SOT]	9
1.5. Problème restant	10
1.6. Conclusion	10
2. Simulations multiexétrons	11
2.1. Le terrain	11
2.2. Remarque préalable	11
2.3. Blocage de ressources communes.....	11
2.3. Simulation à blocage absolu [SBA].....	11
2.4. Simulation à blocage cellulaire [SBC].....	12
2.5. Conclusion	12
3. Simulations parallèles	13
3.1. Base	13
3.2. Mise en parallèle	13
3.3. Améliorations	14
4. Analyse de Mobidyc	15
4.1. Introduction.....	15
4.2. Son principe.....	16
4.3. Fonctionnement	16
4.4. Conclusions et améliorations possibles	18
5. Analyse de l'écosystème aquatique	19
5.1. Analyse	19
5.2. Approche I.....	19
5.3. Approche II	20
5.4. Relations.....	21
6. Implémentations	23
6.1. MOBIDYC	23
6.2. SimPAMu 1	28
6.3. SimPAMu 2	30
7. Simulation	33
7.1. Motivation	33
7.2. Influence de jours et nuits	33
8. Conclusions.....	35
8.1. MOBIDYC	35
8.2. L'écosystème aquatique	35
8.3. SimPAMu	35
9. Bibliographie	36
10. Annexes	37
10.1. Spiders 98	37
10.2. Guide utilisateur SimPAMu 2	39

Table des figures

Figure 1: terrain à cellules quadratiques	5
Figure 2: terrain sous la forme d'un torus	5
Figure 3: balayage cellule par cellule	6
Figure 4: erreur de calcul lors du balayage	7
Figure 5: balayage individu par individu	7
Figure 6: transaction interne	8
Figure 7: problème de l'occupation d'une cellule	10
Figure 8: cycle d'une SOT	13
Figure 9: cycle d'une SOT parallèle	13
Figure 10: Synchronisation de MOBIDYC [HBS02]	17
Figure 11: approche I d'un écosystème aquatique	19
Figure 12: approche II d'un écosystème aquatique	20
Figure 13: MOBIDYC, représentation des espèces	23
Figure 14: MOBIDYC, erreur de définition	24
Figure 15: MOBIDYC, erreur de non conformité du code	24
Figure 16: MOBIDYC, erreur de division par zéro	25
Figure 17: MOBIDYC, visualisation des cellules et agents.....	25
Figure 18: MOBIDYC, code couleur des cellules	26
Figure 19: Erreur d'accès à la mémoire	28
Figure 20: simulation à blocage absolu	29
Figure 21: simulation ordonnancée avec transitions	30
Figure 22: évolution du nombre d'individus	31
Figure 23: contrôle de SimPAMu 2.....	32
Figure 24: éditeur de valeurs de SimPAMU 2	32
Figure 25: Influence de jour et de nuit	34
Figure 26: Influence de jour et de nuit (suite).....	34

Avant d'analyser MOBIDYC et avant d'implémenter un écosystème aquatique, j'ai fait quelques réflexions sur les simulations en générales, notamment sur les différents types et leurs avantages ou désavantages éventuels.

1. Simulations monoexétron

1.1. Le terrain

Avant de passer à travers les différents modèles de conception, il est important de fixer les idées sur le terrain sur lequel les individus simulés vont se balader. La version la plus simple consiste à scier le terrain en différentes cellules quadratiques, comme indiqué sur la figure suivante:

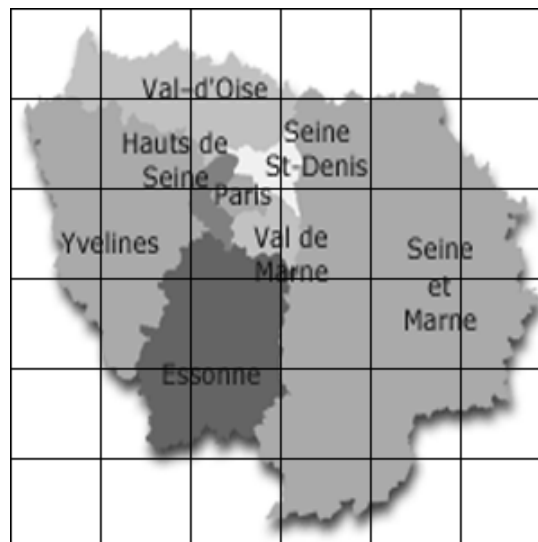


Figure 1: terrain à cellules quadratiques

En supposons que le bas soit connecté avec le haut et la gauche avec la droite, on obtient un torus. De cette manière chaque cellule du terrain possède donc huit cellules voisines.

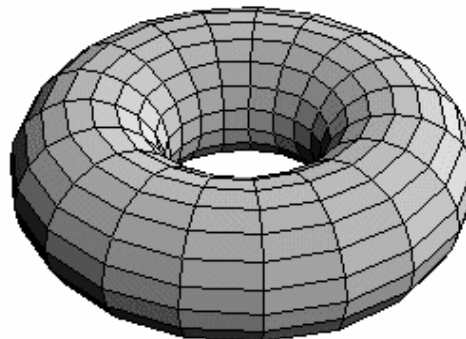


Figure 2: terrain sous la forme d'un torus

Afin de me concentrer sur la méthode de simulation, ce qui est à mon avis l'essentiel, je vais garder ce modèle de terrain tout au long des pages suivantes.

1.2. Simulation incrémentale simple [SIS]

Avec cette méthode chaque individu se trouve sous une forme ou une autre dans la mémoire. Les individus sont passifs, c'est à dire que l'application ne possède qu'un seul exétron qui est chargé d'appeler à tour de rôle les individus. Cela peut se faire de deux manières différentes.

Dans la première le terrain est balayé cellule par cellule et à chaque fois qu'un individu est trouvé, il est exécuté.

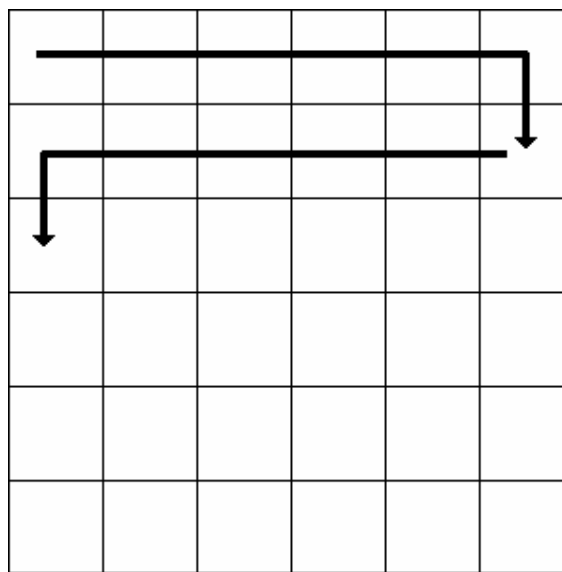


Figure 3: balayage cellule par cellule

Par «exécuté» je veux dire qu'il "vit" une unité de temps. En effet, le temps est discret. En plus, le temps pendant lequel chaque individu s'exécute, dépend de comment il "vit", c'est à dire, s'il a beaucoup de calculs à faire ou non.

Évidemment cette méthode entraîne une certaine erreur de calcul. Comme indiqué sur la figure suivante, supposons qu'il y ait trois l'individus A, B et C. (Par abus de langage je ne vais plus répéter le mot «individu» dans la suite, mais référencer les individus uniquement par leur symbole associé.) Donc, si A, B et C sont dépendantes les uns des autres, il n'y aura pas de problème pour A, par contre pour B et C. En effet lorsque c'est le tour de C, l'état de A a déjà changé. Pire pour B, car là A et C ont déjà été modifiés.

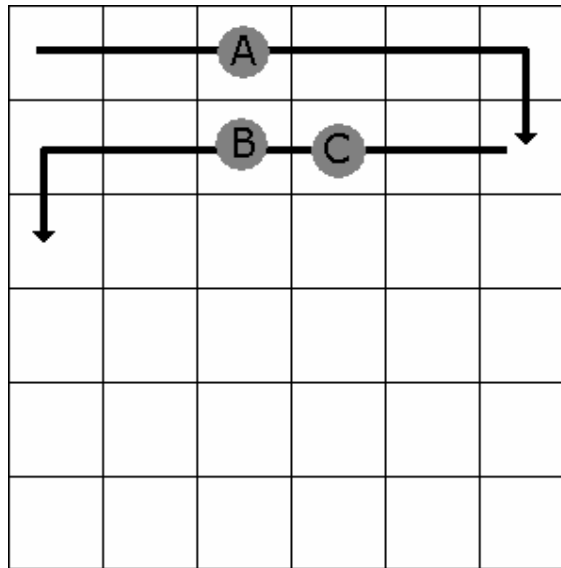


Figure 4: erreur de calcul lors du balayage

La deuxième méthode consiste à faire exécuter les individus d'après leur ordre de naissance dans le système, donc en parcourant par exemple le vecteur système contenant toutes les références.

Il faut noter que cette méthode entraîne la même erreur que la précédente en ce qui concerne le changement d'état des individus. En effet le changement d'état d'un individu dépendant d'un autre peut être différent selon si celui dont il dépend est exécuter avant ou après lui-même.

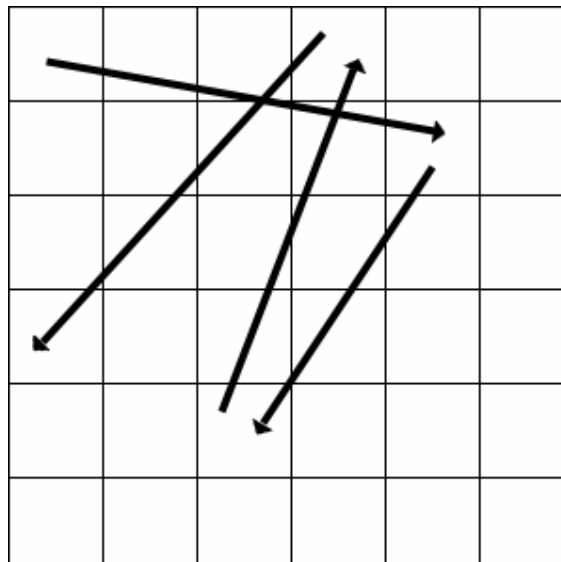


Figure 5: balayage individu par individu

1.3. Simulation incrémentale avec transitions [SIT]

Un des problèmes majeurs de la «simulation incrémentale simple» est l'introduction d'une erreur pour les individus qui sont interdépendants des états des autres.

Peu importe la méthode de balayage, supposons qu'une unité de temps soit le temps mis pour exécuter tous les individus une seule fois. Remarquons que ceci donne toujours lieu à des problèmes temporels.

Afin de ne plus introduire l'erreur d'auparavant, il faut donc que l'état des individus ne change qu'après que le dernier serait exécuté. Pour obtenir cela, il faut que les individus possèdent un mécanisme transactionnel qui permette de garder l'état actuel en lecture, de le changer en interne tout en gardant l'ancien en externe et de rendre définitif le changement qu'après validation.

Un cycle de vie consiste donc en deux balayages: un premier pendant lequel l'état interne des individus est changé et un deuxième lors duquel les changements sont validés et appliqués.

Est-ce que ces transactions sont-elles bien ACID²? L'atomicité est garantie par le fait que les événements se déroulent séquentiellement. Il en est de même pour la consistance et l'isolation. Puisque c'est lors du deuxième cycle que toutes les transactions sont validées et que cette opération ne consiste qu'à mettre à jours les variables de lecture, la transaction est aussi durable.

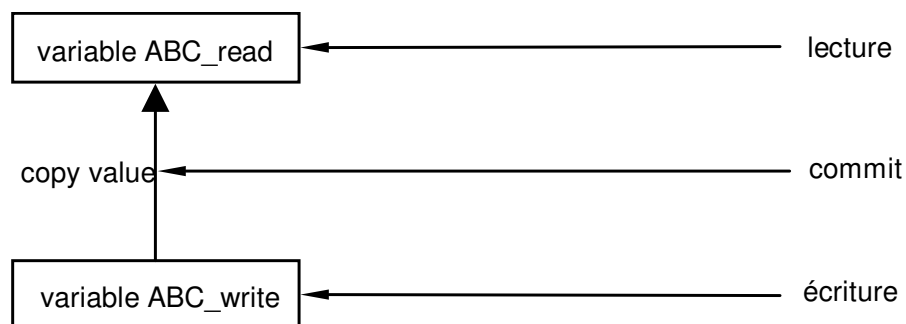


Figure 6: transaction interne

² A = Atomic – Transactions are atomic (all or none), C = Consistency – A consistent state of the database may be expected at all times, I = Isolation – Transactions are isolated from another; race conditions ensure that multiple transaction instances attempts do not collide, D = Durable – Once a transaction commits, it's updates survive even, if the system goes down.

1.4. Simulation ordonnancée avec transactions [SOT]

La «simulation incrémentale avec transactions» propose déjà un mécanisme plus amélioré et correcte que la «simulation incrémentale simple». Par contre elle ne résout pas encore les problèmes temporels tel que par exemple le fait que différents individus se propagent plus vite à travers le terrain que d'autres. Il faudrait donc ordonnancer les exécutions des différents individus.

Une manière possible d'atteindre cela, est de garder toujours une des deux méthodes de balayage, par contre, chaque individu ne sera pas exécuté à chaque balayage.

Supposons que $L \in \mathbb{N}$ représente le nombre de vie d'un individu, avec la propriété que si pour deux individus A et B, avec $L_A > L_B$, alors A sera exécuté plus de fois que B.

Si C représente le nombre de cycles nécessaires afin que chaque individu soit exécuté au moins une fois, alors $C = \text{ppcm}(L_1, \dots, L_n)$ avec n le nombre total d'individus. Un cycle équivaut toujours à deux balayages, un pour les calculs et un pour les validations. Un individu X sera alors exécuté à chaque $\left(\frac{C}{L_X}\right)^{\text{ième}}$ cycle.

Une seconde méthode consiste à laisser gérer le temps par les individus eux-mêmes. Pour cela, supposons que $L \in \mathbb{N}$ représente le nombre de vie d'un individu, avec la propriété que si pour un individu A donné, avec un nombre de vie L_A qui lui est associé, alors A sera exécuté à chaque cycle, mais uniquement à chaque $(L_A)^{\text{ième}}$ cycle il réaliserait des calculs.

La dernière présente l'avantage que l'individu est plus indépendant du système. Cela lui permet, par exemple, d'accélérer ou de freiner sa vitesse en fonction de ce qui se passe autour de lui ou bien de réagir plus vite à un événement qu'à un autre.

1.5. Problème restant

Bien qu'avec ce système on puisse réaliser des simulations qui s'approchent jusqu'à un certain niveau de la réalité, il reste pourtant certains problèmes:

Ce modèle ne permet, par exemple, pas qu'une seule cellule soit occupée par un seul individu à un instant donné. Supposons donc trois cellules adjacentes contenant deux individus de telle manière que la cellule du milieu soit libre. Alors, si dans le prochain cycle les deux individus désirent se déplacer vers la cellule du milieu, rien ne leur indique que l'autre a les mêmes intentions. En effet, ils ne «voient» que l'état du cycle précédent de l'autre.

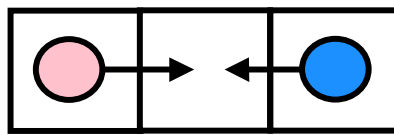


Figure 7: problème de l'occupation d'une cellule

Afin de résoudre aussi ce problème, il faudrait que chaque individu calcule aussi les intentions de ses voisins ce qui d'une part augmenterait considérablement les calculs et donc aussi les temps de réponse et d'autre part cela n'est pas du tout cohérent avec la notion d'individu.

1.6. Conclusion

En résumé, ce système permet de tenir compte du fait que différents individus vivent en parallèle à un instant donné et qu'ils s'influencent mutuellement. Il respecte aussi le fait que certains sont plus ou moins lents.

Il est en outre clair que ce type de simulation ne permet pas des simulations en temps réel lorsque le nombre d'individus est très important. En plus, le temps de calcul sera trop élevé, du moins pour un seul processeur.

C'est pourquoi je vais essayer de voir de plus près des modèles de simulations multiexétons³ et ensuite des modèles de simulations distribuées.

³ en anglais «multi-thread»

2. Simulations multiexétrons

2.1. Le terrain

Afin de garder la même simplicité durant l'analyse des simulations multiexétrons, je garde le même terrain que pour les simulations monoexétron, à savoir un torus aplati.

2.2. Remarque préalable

Concernant les multiexétrons, il ne faut pas oublier qu'il s'agit d'un parallélisme simulé et qu'au fond des choses les différents exétrons sont traités de manière sérielle. Par contre, ils permettent, jusqu'à un certain nombre de ressources, de manipuler plus facilement des simulations «temps réel». Ceci est dû au fait que certains exétrons puissent s'endormir pendant un temps déterminé donné.

2.3. Blocage de ressources communes

Dans les modèles suivants, chaque individu de la simulation possède son propre exétron, c'est à dire qu'il ne sera plus exécuté par l'exétron central, mais qu'il fonctionnera autonome. Ils sont dénommés dans la suite somme agents.

Ayant donc plusieurs exétrons tournant sous le même processus et étant donné que les agents s'influencent mutuellement, il devient nécessaire de gérer les accès concurrents aux ressources partagées.

Exemple:

Lors que, par exemple, un prédateur est en train de "manger" sa proie, et que les deux s'exécutent indépendamment, il est évident qu'il ne sera pas possible que la proie continue à s'exécuter pendant qu'elle est supprimée.

2.3. Simulation à blocage absolu [SBA]

Lors de ce type de simulation, tout est bloqué afin qu'un agent puisse accéder à une ressource commune. Par tout j'entends aussi bien le blocage d'un agent lorsqu'il est examiné ou modifié par un autre que l'environnement dans lequel tous les agents se trouvent et via lequel ils ont la possibilité de s'apercevoir les uns les autres.

Il est très important de tout bloquer, car, imaginons qu'un agent ait lu le contenu d'une cellule, et qu'il détienne désormais une liste d'agents qui s'y trouvent, alors, si son exécution est arrêtée après avoir obtenu la liste et que pendant son repos partiel un des agents de cette liste meurt, une erreur se produira nécessairement lors de la reprise de l'exécution de l'agent.

Le fait que même l'environnement est entièrement bloqué lors d'une partie de l'exécution d'un agent entraîne nécessairement un ralentissement énorme de toute la simulation. Afin d'améliorer les performances, il est donc nécessaire de faire les blocages plus intelligemment.

2.4. Simulation à blocage cellulaire [SBC]

La différence entre la «simulation à blocage absolu» et la «simulation à blocage cellulaire» est que dans cette dernière, seul les cellules, avec les agents qui s'y trouvent, auxquelles un agent accède, sont bloquées. Ceci diminue le temps d'attente des autres agents et augmente donc à nouveau les performances globales de la simulation.

2.5. Conclusion

Bien que les modèles à base d'agents semblent, à première vue, être mieux adaptés à créer des simulations proches de la réalité, ils ne le sont pas en les regardons de plus près.

Par exemple, ils ne respectent pas à 100% le fait que différents individus s'influencent. En effet, c'est le système d'opération qui décide quand un exétron a le droit de s'exécuter et quand il doit s'arrêter. C'est donc le système d'exploitation qui s'occupe de l'ordonnancement des différents exétrons et cet ordonnancement n'est ni prévisible ni nécessairement le meilleur.

Or cela conduit à mon avis en une erreur non négligeable lors de la simulation. En plus, je suis persuadé que pour des simulations sur un seul processeur, laisser l'ordonnancement au système d'exploitation revient au même que de le réaliser par l'exétron primaire. Dans ce dernier cas, on a au moins la possibilité de se débrouiller de telle manière à ce les interactions se déroulent correctement.

3. Simulations parallèles

3.1. Base

Pour les simulations parallèles je me base essentiellement sur le modèle de «simulation ordonnancée avec transitions». Le terrain considéré sera donc aussi le même.

3.2. Mise en parallèle

Lors de la simulation ordonnancée avec transitions un cycle était constitué d'un parcours de calcul suivi d'un parcours de mise à jour des attributs.

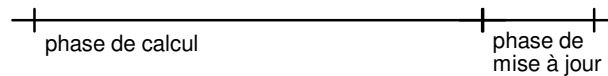


Figure 8: cycle d'une SOT

Supposons maintenant que la simulation soit exécutée sur plusieurs processeurs, alors, vu l'utilisation des transactions, il est évident que pendant la phase de calcul d'une part et la phase de mise à jour d'autre part, il n'y a jamais d'accès concurrent en écriture à des données communes. Ceci n'est pas vrai pour les accès en lecture.

Les calculs des différents agents peuvent se réaliser donc indépendamment les uns des autres. Il en est de même pour les mises à jours. Ce qui reste notamment important est la synchronisation de ces deux phases. En effet une phase ne doit commencer qu'après que chaque processeur ait fini la précédente.

Comment procéder si chaque processeur possède sa propre mémoire? Dans ce cas-là il faut partager aussi les données d'une certaine manière. La première idée qu'on pourrait avoir c'est que soit l'environnement soit partagé, ce qui revient à partager toute la simulation, soit que chaque processeur posséderait sa propre copie de l'environnement, y inclus toutes les entités. Dans le premier cas, on retombe presque sur une simulation monoprocesseur vue que l'environnement devrait être bloqué, donc à écarter immédiatement à mon avis. Pour ce qui est de la deuxième, elle demande beaucoup de travail de synchronisation, car il faut non seulement synchroniser l'environnement mais aussi toutes les entités.

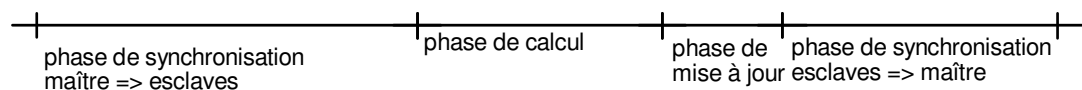


Figure 9: cycle d'une SOT parallèle

Comme le montre la figure précédente, j'imagine donc qu'il existerait une machine maîtresse avec laquelle toutes les autres, les esclaves, vont être synchronisées.

Une nouvelle situation se présente dès lors. En effet, on n'a plus besoin que les différents processeurs se synchronisent après chaque étape, vu que les calculs se font sur des données redondantes propres à chaque processeur, mais uniquement en début et en fin de cycle.

Esclave

Ayant reçu de son maître sa synchro, un processeur donné peut commencer immédiatement à calculer toutes les entités demandées par le maître. Ensuite, il les met à jour et communique les résultats, donc uniquement ceux des entités à calculer, au maître. Après cela, il doit attendre à ce que le maître lui envoie à nouveau une synchro.

Maître

Le maître de son côté doit faire la découpe de la simulation, puis envoyer à ses esclaves l'ensemble des données dont ils ont besoin pour réaliser leurs calculs. Ensuite, il attend à ce que ses esclaves lui transmettent leurs résultats. Si jamais il y a un esclave qui tombe, le maître doit redistribuer aux autres la portion de travail restante. Seulement après avoir reçu toutes les parties calculées, le maître fait de son côté une mise à jour des attributs des entités. C'est ensuite qu'il peut lancer le prochain tour de calcul.

3.3. Améliorations

Afin de réduire les temps de synchronisation, il ne faut pas que le maître envoie toute la simulation aux esclaves, mais uniquement la portion minimale nécessaire.

Exemple

Supposons une simulation avec des lions qui mangent des gazelles et qu'un esclave ait la charge de faire les calculs pour un lion donné, alors la portion minimale nécessaire à faire les calculs correspondait à la portion de terrain, y inclus ce qui se trouve dessus, de l'entourage du lion dans lequel il peut agir ou concevoir d'autres entités.

4. Analyse de Mobidyc

4.1. Introduction

MOBIDYC est l'acronyme pour «MOdélisation Basée sur les Individus pour la Dynamique des Communautés». Il s'agit d'un système multi-agents capable de réaliser des simulations d'individus. Bien que homophone avec «Moby Dick», cette application n'a rien de particulière à faire avec l'oeuvre de Herman Melville [HME51], à part qu'on puisse simuler à l'aide de MOBIDYC une baleine chassée par un pêcheur.

Ce logiciel est développé et maintenu par une équipe de chercheurs du Centre d'Avignon de l'Institut National de La Recherche Agronomique (INRA), dirigée par Vincent Ginot⁴. Actuellement MOBIDYC est disponible dans la version 2.0. Il est écrit en VisualWorks [CCS03]⁵ Smalltalk [PWL03]⁶ ce qui explique entre autre sa portabilité.

D'après son site Internet [VGL03]⁷, MOBIDYC cible surtout les chercheurs non informaticiens. En effet, ses développeurs prétendent qu'on n'a ni besoin de connaître un langage de programmation ni d'écrire du code afin d'établir une simulation et de s'en servir. Voilà pourquoi ils mettent à disposition un certain nombre d'outils de construction et d'analyse.

C'est en effet via l'interface graphique que l'utilisateur entre toutes ces données, à l'aide desquelles la partie constructrice de la simulation de MOBIDYC écrit soi-même son propre code Smalltalk correspondant, qui sera exécuté lors du lancement de la simulation. Ceci est possible parce que Smalltalk est un langage interprété capable de se modifier en cours d'exécution.

Dépendant des performances de la machine sur laquelle MOBIDYC est exploité et de la complexité du modèle implémenté, MOBIDYC présente quand même un certain nombre de contraintes. En effet l'espace dans lequel les individus vivent ainsi que le nombre d'agents actifs dans le système sont limités. Un excès entraîne des calculs très lents.

⁴ ginot@avignon.inra.fr

⁵ <http://www.cincom.com/scripts/smalltalk.dll/index.ssp>

⁶ <http://www.smalltalk.org>

⁷ http://www.avignon.inra.fr/internet/unites/biometrie/mobidyc_projet/version_index.html

4.2. Son principe

Comme décrit par [GPS02], MOBIDYC repose sur un modèle de décomposition des tâches en primitives, c'est à dire en opérations unitaires. Le nombre de ces dernières est limité et ce afin de garder l'utilisation du logiciel le plus simple que possible. Les agents ne sont donc pas décrits globalement ou au niveau de leurs rôles, mais plutôt par leurs actions (les tâches). Ces dernières se décomposent en primitives.

Une primitive peut être une structure de contrôle, par exemple du type «Si ... Alors» ou une opération de base, comme par exemple «ModifierAttribut».

Toujours dans le soucis de simplifications, MOBIDYC exploite un modèle «tout agent» qui réduit considérablement la complexité des primitives. D'après [HBS02] tout dans ce modèle est agentifié et le système se compose de trois types d'agents:

- Les «animats» (terminologie introduite par [WIL87]), qui représente des individus actifs ou passifs plongés dans l'environnement.
- Les agents de d'espace, qui représentent les différentes cellules de l'environnement. Ceci est nécessaire à cause du fait que tout est modelé comme étant des agents.
- Les agents non situés, qui sont en fait des observateurs. Ce type d'agent n'est pas plongé dans l'environnement. Le plus visible est le «simulateur» qui porte les caractéristiques de la simulation comme par exemple le pas de temps et la durée total.

4.3. Fonctionnement

D'après les spécifications techniques de MOBIDYC, il s'agit bien d'une plateforme multi-agents. En plus, d'après la définition de Ferber [BFS91], un agent est autonome et il possède le contrôle sur son comportement. On en conclut souvent qu'il doit donc s'agir au moins d'un objet actif qui doit posséder son propre exétron⁸. Or, en faisant tourner une des simulations exemples fournies avec MOBIDYC et, en contrôlant, à l'aide des outils fournis avec le système d'exploitation, le nombre total d'exétrons présents dans le système on ne constate aucune variation. Dans un premier temps j'ai conclu que soit MOBIDYC n'utilise pas des exétrons pour la simulation des individus, soit que les exétrons d'un programme Smalltalk ne sont pas visibles au système d'exploitation, ce qui me semble quand même étrange.

En effet, après l'analyse des bases du logiciel, j'ai découvert que les tâches sont gérées par le programme lui-même. Comme illustre la figure suivante et décrit par [HBS02], MOBIDYC propose deux modes différents de traitement des tâches: un mode séquentiel et un mode synchrone.

⁸ en anglais «thread»

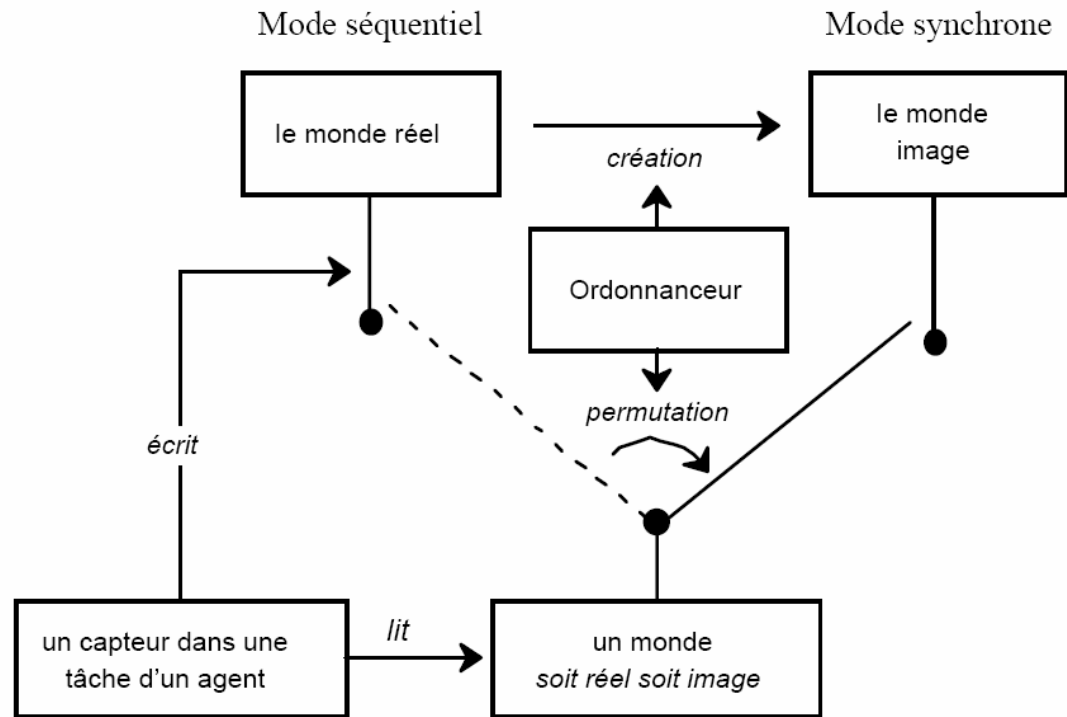


Figure 10: Synchronisation de MOBIDYC [HBS02]

«En mode séquentiel, les primitives lisent et écrivent dans le monde réel. En mode parallèle, l'ordonnanceur crée une image du monde et la bascule vers les tâches afin que tous les agents concernés voient la même chose.» - [HBS02, page 8]

Le comportement de l'ordonnanceur est au choix de l'utilisateur. Ce dernier possède même la possibilité d'influencer la démarche.

Le mode séquentiel de MOBIDYC correspond donc à ce que j'ai appelé tantôt «simulation incrémentale simple» (cf. chapitre 1.2), tandis que son mode synchrone revient à ma «simulation incrémentale avec transactions» (cf. chapitre 1.3).

4.4. Conclusions et améliorations possibles

MOBIDYC se dit simple à utiliser. Or, à mon avis, l'interface graphique n'est pas assez intuitive. Toute personne qui aimerait l'utiliser est sensée de lire la documentation, ce qui est d'ailleurs la procédure standard pour arriver à comprendre le fonctionnement d'un logiciel. Or, un grand nombre de gens se passent de cette étape préliminaire et, du coup, abandonnent le programme sans même l'avoir testé.

Voilà pourquoi je suis d'avis que le succès d'un logiciel dépend toujours fortement de son interface graphique. Comme le montre [URS98] et bien d'autres, c'est ce qui est perçu qui emporte sur tout le reste. Cela est une des raisons pourquoi certaines gens ont du mal à comprendre ce qu'est en train de faire un programme quand ils ne le voient pas à l'écran. Il en résulte la popularité des éditeurs WYSIWYG⁹.

Je me demande donc s'il ne serait pas possible d'ajouter une couche graphique plus orientée WYSIWYG et peut-être des «wizards» interactifs qui proposeraient de réaliser certains modèles. Cela permettrait peut-être d'améliorer la popularité de MOBIDYC et de gagner plus d'utilisateurs.

Je suis persuadé que pour toutes les primitives, il existe une bonne représentation graphique. À l'aide de ces représentations on pourrait concevoir un éditeur «drag & drop». Comme le déroulement d'une tâche est séquentiel, on pourrait adopter une vue semblable à celle des éditeurs de vidéo numérique. Ces derniers semblent être très intuitifs et assez facile à comprendre puisqu'un grand nombre de personnes, dont même des non spécialistes dans le domaine, savent les manipuler.

⁹ What You See Is What You Get

5. Analyse de l'écosystème aquatique

5.1. Analyse

Avant d'implémenter l'écosystème aquatique [DOD03]¹⁰, j'ai essayé de rassembler un certain nombre d'informations dans un modèle.

5.2. Approche I

Dans un premier modèle, j'ai considéré tout, sauf la lumière, comme étant des agents. Ces agents avaient différentes propriétés, comme par exemple le sel, qui était vu comme un agent complètement inactif. J'en ai dégagé le schéma suivant:

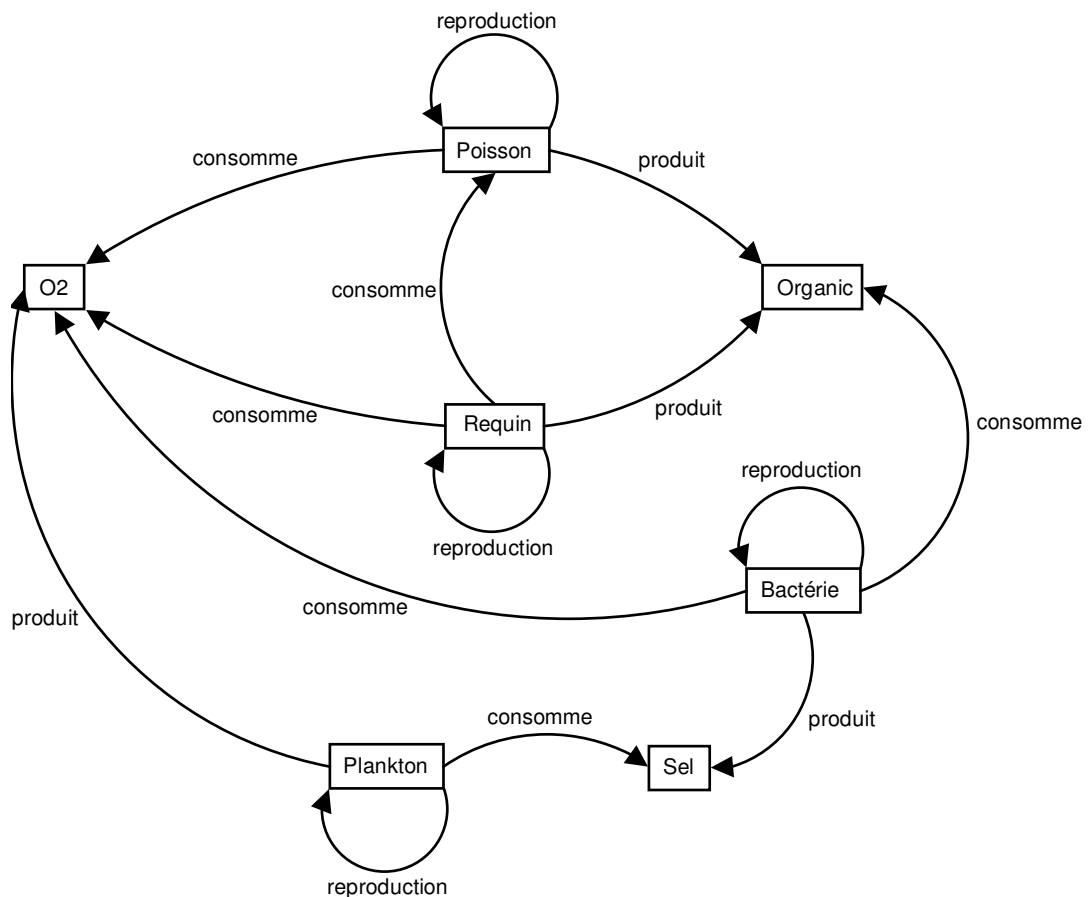


Figure 11: approche I d'un écosystème aquatique

Comme à mon avis l'oxygène, qui est une molécule assez rapide, était en mouvement, je me suis dit qu'il faudrait qu'elle aussi soit considérée comme étant un agent. De là, j'ai fait de même pour les sels et les déchets

¹⁰ <http://scott.univ-lehavre.fr/~olivier/Enseignement/DEA/ecosysteme.html>

organiques. La lumière par contre, je l'ai vue comme étant un attribut de l'environnement.

5.3. Approche II

Après plusieurs réflexions, et ce afin d'optimiser mon modèle, j'ai abouti à la conclusion que l'oxygène, ainsi que les sels et les déchets organiques, pourraient aussi faire partie de l'environnement.

Par exemple, une cellule de l'environnement pourrait très bien avoir un attribut contenant le nombre de molécules d'oxygène, de sels ou de déchets organiques présents.

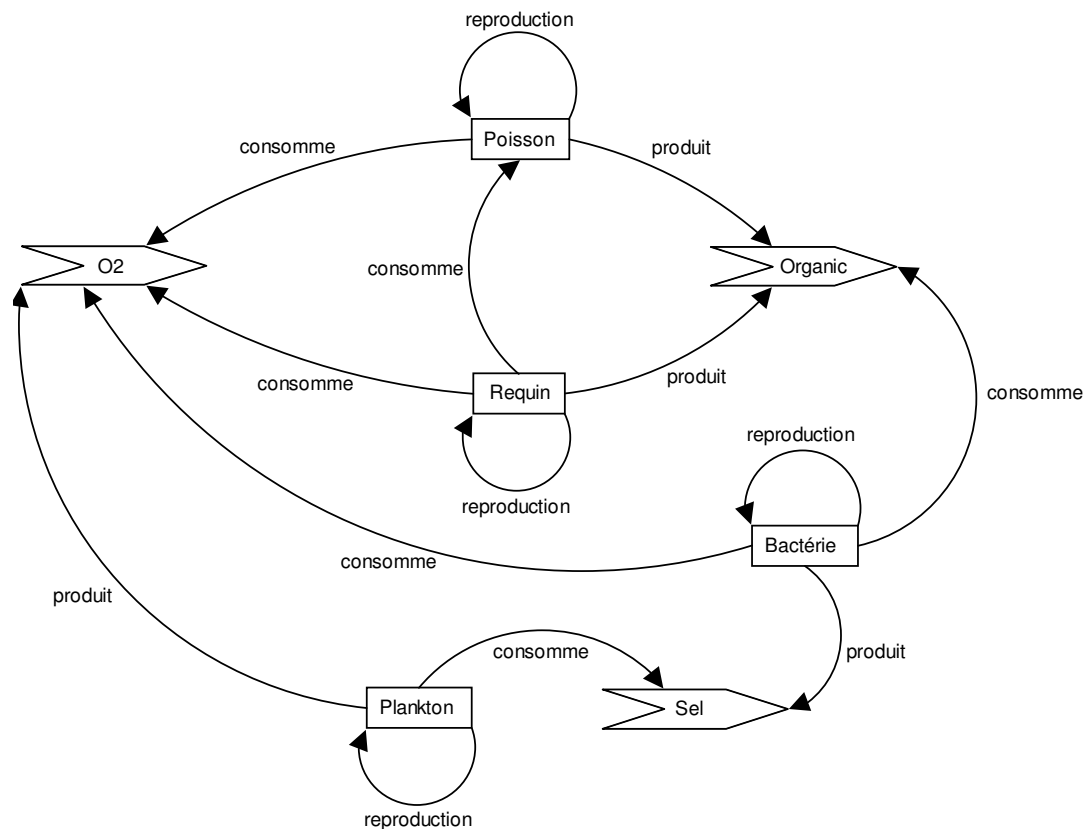




Figure 12: approche II d'un écosystème aquatique

De cette manière, il me reste 4 agents actifs. Les taux d'oxygène, de sels minéraux, de déchets organiques ainsi que l'intensité lumineuse du soleil sont représentés par des attributs des cellules respectives de l'environnement



5.4. Relations

Ensuite, j'ai mis au point différentes relations entre les entités du modèle. Ces relations restent bien entendu assez vagues et il faudrait y mettre des valeurs numériques. Pour s'approcher le plus possible de la réalité, il serait préférable de travailler ensemble avec un biologiste qui se connaît mieux dans le domaine.

Plancton



- Le plancton peut collectionner des sels minéraux. Le nombre de sels minéraux qu'il peut emmener avec soi est limité.
- Il possède un indice de maturité. Dès que cet indice dépasse une valeur seuil, il se reproduit.
- Il a un certain âge, qui est incrémenté avec les différents cycles de vie.
- À l'aide de la lumière présente dans le système, il transforme par photosynthèse un sel en oxygène. En même temps son indice de maturité est incrémenté. Bien entendu il ne sait pas faire de la photosynthèse durant la nuit. Lorsqu'il ne sait plus produire, il est en quelque sort à bout de force, il meurt.
- À un certain âge, le plancton meurt.
- Représentation lors de mes simulations:  ou 

Bactérie



- La bactérie peut collectionner des déchets organiques. Le nombre de déchets qu'elle peut emmener avec soi est limité.
- Elle peut aussi stocker de l'oxygène. Ce stockage est aussi limité. Si elle ne trouve plus d'oxygène, elle meurt.
- Elle possède un indice de maturité et se reproduit en fonction de celui-ci.
- Elle a un certain âge qui est incrémenté avec les différents cycles de vie.
- Elle produit des sels minéraux des déchets organiques. A chaque production son indice de maturité est incrémenté. La bactérie meurt si elle ne peut plus transformer des déchets organiques en sels minéraux.
- À un certain âge, la bactérie meurt.
- Représentation lors de mes simulations:  ou 

Poisson

- Le poisson possède un indice de faim. Si cet indice tombe à zéro, le poisson meurt.
- Il peut manger du plancton. Son estomac ne peut en supporter qu'un nombre limité. S'il mange un plancton, son indice de faim est majoré.
- Il a besoin d'oxygène pour vivre. Il peut emmener une petite réserve, mais s'il n'y en a plus, il meurt.
- Il possède un indice de maturité en fonction duquel il se reproduit.

- Il a un certain âge qui est incrémenté avec les différents cycles de vie.
- A sa mort il produit un certain nombre de déchets organiques. A chaque cycle il produit aussi une unité de déchet organique.
- Lorsque l'intensité lumineuse de l'environnement baisse en dessous d'une certaine limite, le poisson dort, c'est à dire qu'il ne bouge plus et qu'il ne mange plus.
- À un certain âge, le poisson meurt.
- Représentation lors de mes simulations:  ou 

Requin

- Le requin possède un indice de faim. Si cet indice tombe à zéro, le requin meurt.
- Il peut manger du plancton. Son estomac ne peut en supporter qu'un nombre limité. S'il mange un plancton, son indice de faim est majoré.
- Il mange aussi des poissons. Le nombre de poissons qu'il peut digérer en même temps est limité. A chaque fois qu'il mange un poisson, son indice de faim également est augmenté.
- Il a besoin d'oxygène pour vivre. Il peut emmener une petite réserve, mais s'il n'y en a plus, il meurt.
- Il possède un indice de maturité en fonction duquel il se reproduit.
- Il a un certain âge qui est incrémenté avec les différents cycles de vie.
- A sa mort il produit un certain nombre de déchets organiques. A chaque cycle il produit aussi une unité de déchet organique.
- Lorsque l'intensité lumineuse de l'environnement baisse en dessous d'une certaine limite, le requin dort, c'est à dire qu'il ne bouge plus et qu'il ne mange plus.
- À un certain âge, le requin meurt.
- Représentation lors de mes simulations:  ou 

Pour toutes ces relations, il faut encore bien entendu y mettre des valeurs numériques. Le déroulement de la simulation dépendra évidemment du bon choix de ces valeurs. Afin d'obtenir des résultats signifiants, il faudrait donc au moins connaître à peu près les bons rapports entre les différentes constantes.

6. Implémentations

6.1. MOBIDYC

Pour la réalisation du système aquatique dans MOBIDYC, j'ai commencé à créer une grille de cellules ayant les quatre attributs définis dans mon approche II.

Ensuite j'ai mis en place trois classes d'agents (BPlankton, BBacterie et BFish) avec quatre instaurations, à savoir une pour chaque agent du modèle. À chaque agent est ensuite affecté une représentation graphique.



Figure 13: MOBIDYC, représentation des espèces

Ensuite, chaque agent a reçu ses propres attributs, comme par exemple la réserve d'oxygène, la faim ou encore l'indice de maturité pour le poisson. Dans une prochaine étape, j'ai déduit, à partir des relations établies dans le chapitre précédent, les différentes tâches que chaque agent devrait avoir. Toute tâche a ensuite été divisée en primitives.

J'ai décrit de cette manière les agents représentant les bactéries et les planctons. Pour les deux autres par contre, donc les poissons et les requins, c'était plus délicat. En effet, tous les deux se nourrissent de planctons. Jusque là tout va bien, puisqu'il existe une tâche «TuerSurNom». Or, lorsqu'ils mangent un plancton, il faudrait que leur faim soit diminuée et que leur indice de maturité soit augmenté. Malheureusement la tâche «TuerSurNom» ne le permet pas. Il s'y rajoute qu'ils ne mangent que pendant la journée et uniquement s'ils ont faim, ce qui nécessite donc une condition supplémentaire.

Il faut donc nécessairement créer une nouvelle tâche plus adaptée aux besoins. Cette nouvelle tâche, je vais la dénommer «Manger», devrait donc commencer par une primitive «Si ... Alors» suivie de toutes les primitives qui seront à exécuter.

En gros la tâche «Manger» devrait être définie comme suit:

```
1: If ... then
2: My cell
3: From cells to agents
4: To unroll a list
5: To kill
6: Modify Attributes
7: End - To unroll a list
```

Cela correspondait donc à:

1. Si c'est le jour et que le poisson a faim, alors continuer
2. Sélectionner la cellule du poisson
3. et obtenir la liste de tout les agents qui y vivent
4. parcourir la liste en sélectionnant les planctons
5. manger le plancton
6. augmenter l'indice de faim du poisson
7. fin du parcours de la liste

En mettant au point ces tâches, il faut bien être conscient du fait qu'il faut penser linéairement, du genre "Quick Basic" ou "Turbo Pascal" avec des «break» ou «goto».

Arrivé à ce stade, MOBIDYC a commencé à me montrer son côté négatif:

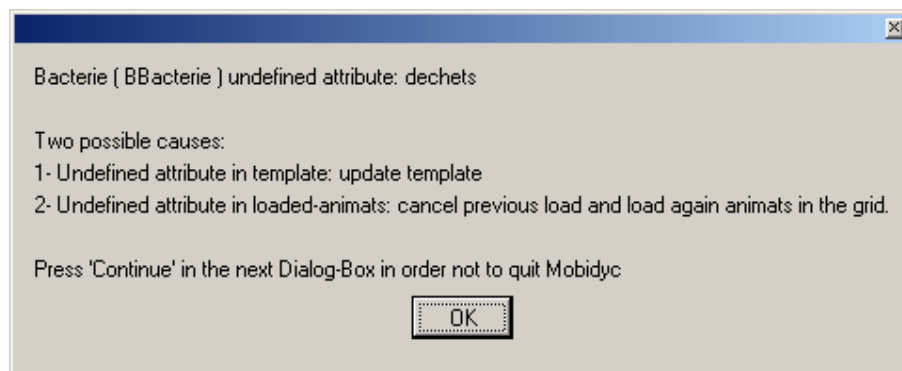


Figure 14: MOBIDYC, erreur de définition

Celle-ci est encore très compréhensible, indiquant qu'il y a un attribut de la bactérie qui n'est pas défini. Ce qui reste drôle, voir même étrange, c'est que le dit attribut est bien défini dans l'animat.

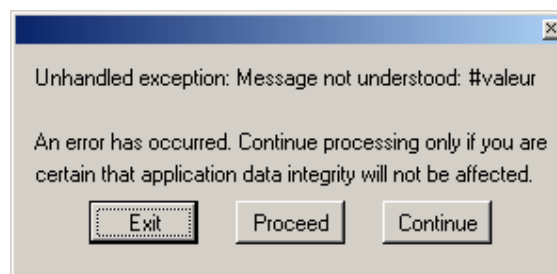


Figure 15: MOBIDYC, erreur de non conformité du code

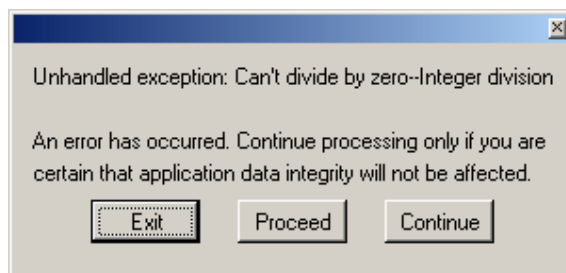


Figure 16: MOBIDYC, erreur de division par zéro

Il y en avait encore d'autres comme celles ci-dessus. J'ignore où se situe la faute. Remarquons que le bouton «Exit» est le seul qui fonctionne à tous les coups.

Déterminé d'obtenir avec MOBIDYC à un résultat plus ou moins satisfaisant, j'ai recommencé à zéro. Il m'a fallu plusieurs essais avant de remarquer que dépendant du bon choix des primitives et des noms qu'on affecte aux tâches, MOBIDYC génère une erreur. Certaines ne sont pas graves tandis que d'autres nécessitent dl recommencement de certaines opérations. Manque de connaissance du logiciel, la solution la plus simple consistait donc à recommencer à zéro à chaque fois et de faire beaucoup de sauvegardes de secours.

De cette manière, j'ai enfin réussi à obtenir une simulation tournante et respectant mon modèle.

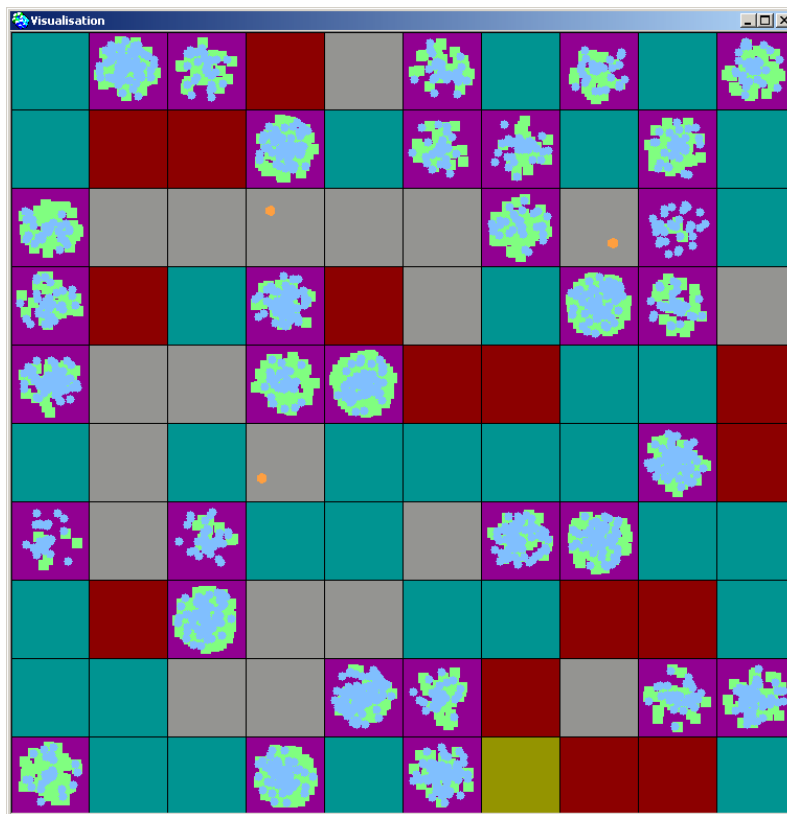


Figure 17: MOBIDYC, visualisation des cellules et agents

Sur l'image ci-dessus la couleur des cellules donne une indication sur les attributs oxygène, sels minéraux et déchets organiques. Le code couleur est le suivant:

■	Cell: oxygene > 0 AND salt = 0 AND organic = 0
■	Cell: oxygene = 0 AND salt > 0 AND organic = 0
■	Cell: oxygene = 0 AND salt = 0 AND organic > 0
■	Cell: oxygene > 0 AND salt > 0 AND organic = 0
■	Cell: oxygene > 0 AND salt = 0 AND organic > 0
■	Cell: oxygene = 0 AND salt > 0 AND organic > 0
■	Cell: oxygene > 0 AND salt > 0 AND organic > 0

Figure 18: MOBIDYC, code couleur des cellules

Afin que les différents agents puissent se déplacer conformément au modèle établi, j'ai dû définir une tâche «DéplacerConditionnel» qui est définie comme suit:

```

1: If ... then
2: My neighborhood
3: To unroll a list
4: To move to (or fly to) one cell
5: End - To unroll a list

```

Pour la bactérie, cela correspond donc à:

1. Si ma cellule ne contient plus d'oxygène ou de déchets, alors continuer
2. Sélectionner les cellules autour de moi dans un certain rayon
3. parcourir la liste des cellules obtenues et choisir celles qui contiennent encore de l'oxygène et des déchets organiques
4. se placer dans la cellule choisie
5. fin du parcours

Voilà ce qui fonctionne bien pour les bactéries et les planctons, mais pas très bien pour les deux types de poissons. En effet, la condition de déplacement pour les poissons et les requins ne dépend pas uniquement de la valeur des attributs des cellules destinataires mais aussi du nombre de planctons respectivement de poissons qui y vivent. Voilà donc un nouvel obstacle. Il faudrait en effet une tâche qui puisse déterminer le nombre d'un certain type d'espèces dans une cellule.

Certes, il existe une primitive «To count», mais il est impossible de l'incorporer dans une tâche agent de telle manière à ce qu'elle corresponde aux besoins exprimés. Voilà pourquoi j'ai essayé de diviser cette tâche en deux: D'une part j'ai ajouté aux cellules deux attributs, à savoir «nbsPlanctons» et «nbrFishs», ainsi que deux tâches du type «CompterAgents» pour compter les agents d'un type défini et de mettre la valeur obtenue dans l'attribut respectif.

J'ai défini la tâche «CompterAgents» comme suit:

```
1: Me
2: From Cells to Animats
3: Select following the name
4: To count
```

Cela correspond donc à:

```
1: Sélectionner moi-même
2: Déterminer la liste de tous les animats qui se trouvent chez moi
3: Sélectionner parmi eux tous les ( planctons | poissons )
4: Compter combien il y en a et mettre la valeur dans l'attribut ( nbsPlanctons | nbrFishs )
```

Le principe est donc que les cellules comptent eux-mêmes le nombre d'agents qui se trouvent chez elles, et que les agents puissent accéder à cette valeur dans la suite. Cela ne pose pas de problèmes, vu que dans l'ordonnancement de la simulation, les cellules sont exécutées avant les animats.

Il reste à mettre en place une tâche qui tient compte des attributs de la cellule pour déterminer vers où un poisson doit se déplacer. Cette tâche, que je vais désigner comme «DéplacerConditionnelDouble», devrait être définie comme ceci:

```
1: If ... Then
2: If ... Then
3: My neighborhood
4: To unroll a list
5: To move to (or fly to) one cell
6: End - To unroll a list
```

En effet, elle est presque identique à la tâche «DéplacerConditionnel», avec la seule différence qu'il y a une condition de plus au début. Elle est nécessaire à cause du fait qu'une primitive du type "If ... Then" ne peut réunir des conditions exclusivement par l'opérateur logique ET ou exclusivement par l'opérateur logique OU. Or pour le cas précis des poissons, et aussi pour les requins, la condition de départ peut se formuler:

```
(s'il fait jour)
ET
(
    (si ma cellule ne contient plus d'oxygène)
    OU
    (si ma cellule ne contient plus de planctons)
)
```

Tout cela semble logique et tient bien le coup en théorie. Malheureusement en pratique il semble y avoir encore un problème de définition, vu que MOBIDYC refuse catégoriquement l'implémentation des deux tâches, soit à cause d'un attribut pas défini, soit à cause d'une clé non trouvée.

6.2. SimPAMu 1

Il s'agit d'un simulateur du type SBA (cf. chapitre 2.3) que j'ai développé dans Delphi 6 PE, utilisant un exétron par agent simulé. «SimPAMu» est l'acronyme de «Simulateur Pour Agents Multiples». La simulation est basée sur l'approche I (cf. chapitre 5.2) décrite dans le chapitre précédent. C'est en effet qu'après avoir fait les différentes expériences avec ce simulateur que j'ai retravaillé mon modèle de simulation.

Dans un premier temps, j'avais fait tourner la simulation sans blocage des ressources communes, ce qui se terminait, comme prévu, par des fautes de mémoires.

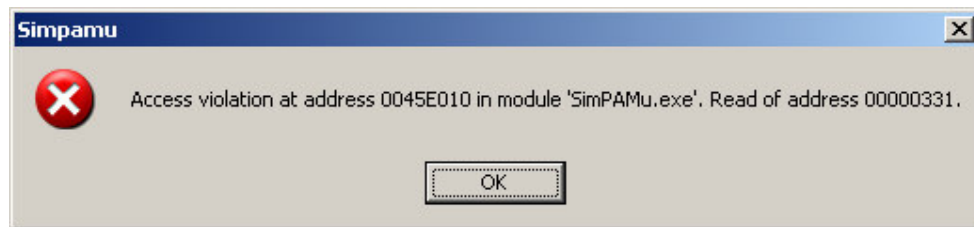


Figure 19: Erreur d'accès à la mémoire

Le fait de procéder de cette manière peut sembler peu intelligent, mais le but n'était pas d'arriver à des erreurs, mais de montrer qu'au fur et à mesure que des blocages sont introduits, les performances de la simulation décroissent rapidement.

Dans une deuxième étape, j'ai donc ajouté des blocages sur chaque agent, de telle manière, que si un agent consultait un autre, il lui fallait d'abord le bloquer, y lire ou apporter des changements, puis le débloquent à nouveau. De même, chaque agent se bloque soi-même avant son exécution et se débloquent après.

Bien entendu, cela n'a pas encore résolu tous les problèmes de blocage, mais j'ai déjà pu constater un certain ralentissement lors de l'exécution. Il reste à noter que pas toutes les exécutions se terminaient par une faute de mémoire, mais c'était plutôt très rare.

J'ai ensuite ajouté un blocage absolu au niveau de l'environnement, ce qui a finalement résolu tous les problèmes d'accès concurrents et de fautes de mémoire. Cela dit, les performances chutaient encore une fois.

J'en conclus que ce type de simulation est très limité, lent et pas évolutif du tout. Voilà aussi pourquoi je n'ai plus implémenté un blocage par cellule, qui n'apporterait à mon avis pas d'énormes avantages.

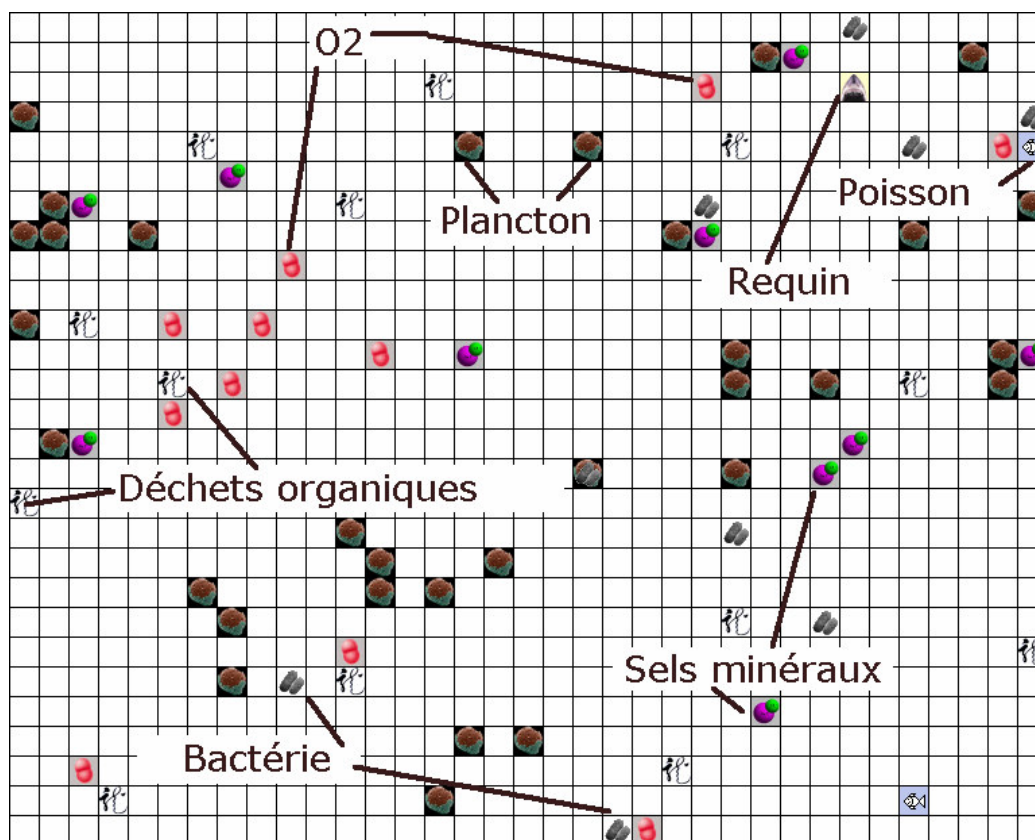


Figure 20: simulation à blocage absolu

Ayant à ma disposition une machine biprocesseur, j'ai lancé un dernier test sur ce type de simulation. J'ai donc lancé la simulation simultanément d'une part sur une machine biprocesseur PIII 1GHz et d'autre part sur une machine monoprocesseur PIV 1,8 GHz (les deux tournant sous le même système d'exploitation) et j'ai constaté que la machine biprocesseur était plus rapide. J'ai répété l'expérience afin d'avoir une certitude que le résultat reste constant.

Je m'explique le résultat obtenu par le fait que le système d'exploitation a bien réussi à affecter un exétron à un processeur unique. Donc il y avait à chaque moment au moins deux exétrons non bloqués qui ont pu faire utilisation de la CPU, ce qui explique la différence de performance par rapport à la machine monoprocesseur.

Remarque:

Lors de la désactivation de la sortie graphique de la simulation, les performances peuvent être considérablement améliorées, par contre le contrôle et la surveillance des événements seront d'autant plus difficiles.

6.3. SimPAMu 2

Pour ce simulateur, qui est du type SOT (cf. chapitre 1.4), j'ai aussi utilisé Delphi 6 PE. La mise en oeuvre de l'écosystème aquatique est basée sur l'approche II (cf. chapitre 5.3) que j'ai décrite dans le chapitre précédent.

Vu certains défauts de SimPAMu 1, j'avais décidé d'implémenter une deuxième version et ceci espérant de pouvoir m'approcher un peu plus de la réalité. Cela m'a permis aussi d'acquérir des connaissances sur la différence de performance des deux modèles testés.

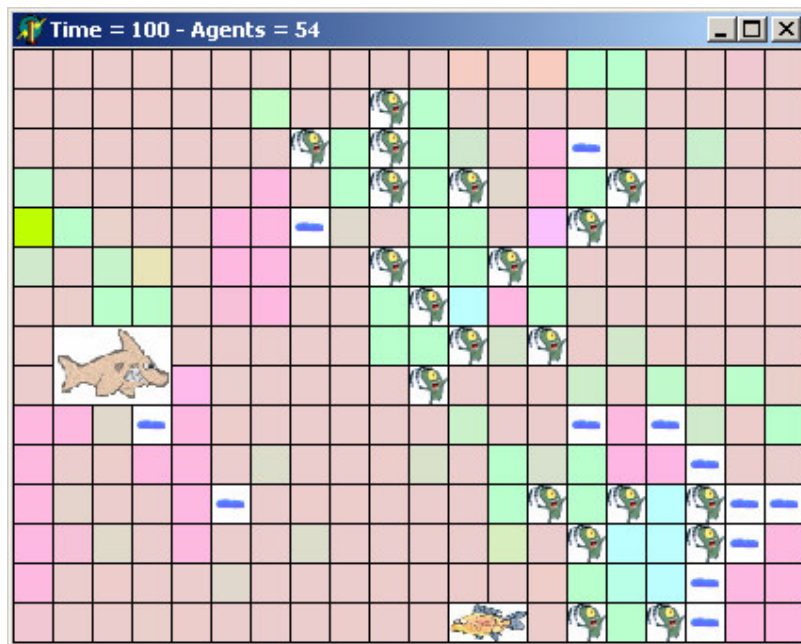


Figure 21: simulation ordonnancée avec transitions

Comme décrit dans le modèle, les cellules possèdent quatre attributs, dont un est l'indice de luminosité. La couleur des cellules change en fonction des trois autres, de la manière à ce qu'elles deviennent bleues quand il y a beaucoup d'oxygène, roses quand il y a beaucoup de sels minéraux et vertes quand il y a plus de déchets organiques.

Afin de mieux pouvoir observer le comportement des populations, j'ai ajouté un module observateur, qui compte le nombre d'individus de chaque espèce après chaque cycle.

Sur la figure suivante on peut par exemple observer que le nombre de poissons est en augmentation en même temps que le nombre de plancton est en baisse. C'est normal puisque les poissons mangent du plancton. Ensuite, dès que le nombre de poissons diminue à nouveau (les poissons meurent soit parce qu'ils sont trop âgés, soit par faim, soit par manque d'oxygène), les planctons se multiplient rapidement jusqu'à ce que tous les sels minéraux initialement dans les cellules soient transformés. Les bactéries suivent un chemin semblable mais avec des variations pas aussi extrêmes.



Figure 22: évolution du nombre d'individus

Comme ce simulateur me semble le plus adapté, je l'ai rendu plus dynamique en rajoutant par exemple un éditeur de valeurs, ce qui me permet de sauvegarder et de restituer différentes configurations sans avoir besoin de recompiler le simulateur à chaque fois.

En plus, j'ai la possibilité de déconnecter à la volé le système de sortie graphique des calculs internes, ce qui rend la simulation beaucoup plus rapide. Ceci est surtout avantageux lorsque la quantité d'agents dans le système dépasse un certain nombre.

Un autre atout de cette version, qui pèse peut-être moins mais je le trouve quand même important, est qu'elle permet que les agents soient de tailles différentes. Comme on peut le voir sur la capture d'écran plus haut, le poisson occupe deux cases et le requin même six.

J'ai également refait la même expérience avec les deux machines que pour la simulation précédente. Cette fois-ci j'ai d'abord constaté que ce n'était plus toujours la machine biprocesseurs qui emportait sur le monoprocesseur. En plus, j'ai remarqué que lors de l'exécution de la simulation, qui tournait en mode sans graphisme, aucun des deux processeurs n'était jamais exploité à 100%, ce qui par contre était bien le cas pour la simulation multiexétron. Cela s'explique probablement par le fait que la simulation tourne dans un seul exétron, et que de là, le système d'exploitation n'a pas su comment diviser et répartir l'application. Dû à cela, les ressources du système ne sont pas exploitées entièrement, ce qui explique pourquoi la machine monoprocesseur était plus rapide lors de ces tests.

Tools

Environment Generator

Cell: Width 20, Height 20

Field: Width 40, Height 30

Generate

Simulation

☐ real time ☐ cells ☒ stats

Show cells

Show agents

Step

Start

Agents

Type: Bacterie, Fish, Hai, Plankton

Count: 10

Insert

Standards

Bacterie: 10

Plankton: 50

Fish: 2

Hai: 0

Insert

Figure 23: contrôle de SimPAMu 2

Agents

Key	Value
PLANKTON_SALT_MAX	5
PLANKTON_REPRODUCTION	50
PLANKTON_DIE	100
PLANKTON_VIEW_SALT	5
BACTERIE_OXYGENE_MAX	5
BACTERIE_ORGANIC_MAX	5
BACTERIE_REPRODUCTION	80
BACTERIE_DIE	150
BACTERIE_VIEW_OXYGENE	5
BACTERIE_VIEW_ORGANIC	5
FISH_OXYGENE_MAX	40
FISH_FOOT_MAX	40
FISH_REPRODUCTION	5
FISH_FISH_ORGANIC	10
FISH_PLANKTON_FOOT	19
FISH_DIE	300
FISH_VIEW_ENNEMY	10
FISH_VIEW_PREY	10
FISH_VIEW_OXYGENE	10
HAI_OXYGENE_MAX	150
HAI_FOOT_MAX	200
HAI_REPRODUCTION	30
HAI_DIE	750
HAI_FISH_FOOT	40
HAI_PLANKTON_FOOT	19
HAI_HAI_ORGANIC	10
HAI_VIEW_PREY	10
HAI_VIEW_OXYGENE	10
CELL_INIT_OXYGENE	20
CELL_INIT_ORGANIC	50
CELL_INIT_SALT	50

Save as Save Load

Figure 24: éditeur de valeurs de SimPAMu 2

Pour l'implémentation de SimPAMu 2 il faut encore ajouter que je me suis inspiré un peu du concept de MOBIDYC, surtout pour la partie de la programmation des agents. Cela se voit assez bien dans le code des agents, où on peut distinguer facilement les différentes tâches et même les différentes primitives. Voici un extrait de l'agent "Plankton":


```
// collect salt
if ((cell.salt>0) and (self.salt<values.get('PLANKTON_SALT_MAX'))) then
begin
  cell.salt:=cell.salt-1;
  self.salt:=self.salt+1;
end;

// produce
if ((SystemEnvironment.light>0) and (self.salt>0)) then
begin
  self.salt:=self.salt-1;
  cell.oxygene:=cell.oxygene+1;
  self.maturity:=self.maturity+1;
end;
```

On remarque que ces deux cas exemples se ramènent très bien à la tâche prédéfinie de MOBIDYC qui s'appelle «ModifierContitionnel».

7. Simulation

7.1. Motivation

Étant donné que mon modèle dans MOBIDYC ne fonctionnait pas comme souhaité, j'ai commencé à analyser les simulations faites avec SimPAMu 2. Cela présente aussi l'avantage que je puisse modifier à volonté le code, notamment définir les agents de manière intrinsèque. De cette manière, j'ai plus de contrôle.

7.2. Influence de jours et nuits

Première chose à remarquer est l'influence du jour et de la nuit. Étant donné que les poissons ne bougent et ne mangent pas pendant la nuit, les planctons ont en quelque sorte une phase de repos pendant laquelle ils peuvent se reproduire tranquillement. Cela se fait remarquer assez bien sur le graphique suivant (en rouge en haut les planctons, les poissons, bruns, tout petit en bas):

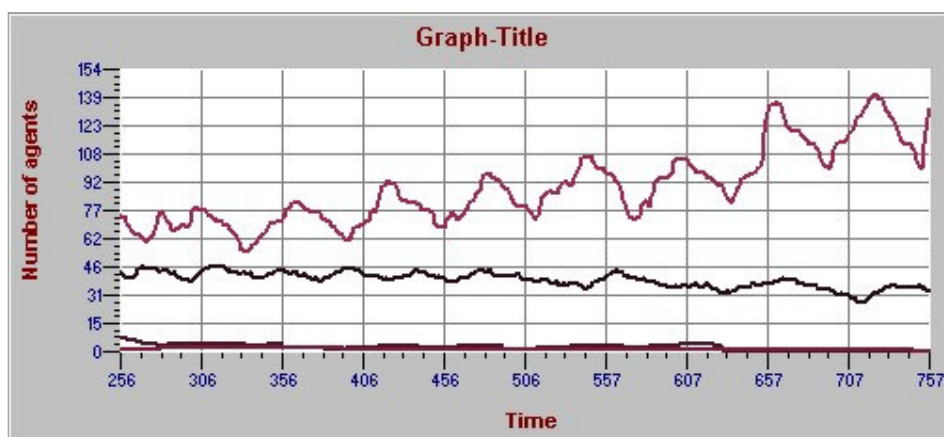


Figure 25: Influence de jour et de nuit

A un certain moment, le nombre de poissons est très réduit et la population des planctons commence à augmenter plus rapidement, et cela jusqu'au moment où il n'y a plus assez de déchets minéraux. Donc les bactéries ne peuvent plus produire des sels et donc les planctons ne se reproduisent plus et sont condamnés à mort suite à un manque de sels.

En effet, tout dépend encore un peu de la quantité initiale de matière dans les cellules. Si, par exemple, le taux initial d'oxygène est trop bas, les poissons meurent tout de suite à cause du manque d'oxygène, ce qui entraîne que les planctons puissent se reproduire plus vite mais vont également mourir plus vite vu qu'il n'y aura plus de déchets organiques ou bactérie pour produire du sel.

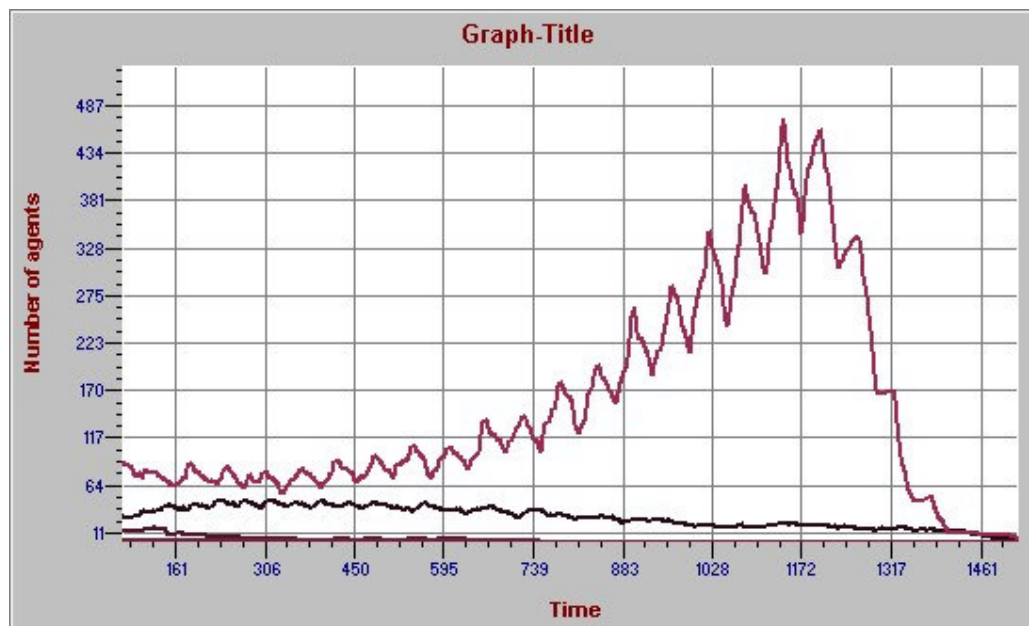


Figure 26: Influence de jour et de nuit (suite)

8. Conclusions

8.1. MOBIDYC

Le concept de base de MOBIDYC est très fort et simple à comprendre. Dans l'état actuel, il reste malheureusement un peu lourd à manipuler, surtout au niveau de l'interface graphique.

J'aurais aussi préféré d'avoir un langage de description unifié pour la description des agents. Une sorte de «Unified Agent Modeling Language» (UAML) ou «eXtensible Agent Description Language» (XADL). Cela dit, remarquons qu'un groupe de chercheurs de l'Université de Hambourg travaille à une mise en place d'un tel langage UAML [RVU03]¹¹. Leur site Internet [RVG03]¹² date du 12 mai 2003, il s'agit donc d'un mouvement encore assez récent. Le site est malheureusement toujours sous construction et ne contient pas vraiment d'informations directes sur UAML.

8.2. L'écosystème aquatique

Il est très intéressant de voir le développement des populations et d'observer leur comportement en fonction des différents paramètres entrés. Ce qui est clair, est que même ce tout petit écosystème aquatique, qui à mon avis ne prend pas énormément de choses en compte, est d'une grande complexité, et cela d'autant plus qu'on essaie de s'approcher du monde réel. J'aimerais encore dire dans ce contexte que ce n'était pas compliqué de mettre en œuvre les différentes relations entre les individus, mais que c'était le grand nombre de relations et possibilités auxquels est due cette complexité.

Je trouve aussi un peu dommage qu'il y ait un assez grand manque de réalité dans ces simulations. Voilà pourquoi il aurait été plus intéressant de travailler ensemble avec un biologiste, ceci non seulement afin d'avoir eu plus de détails sur le vrai comportement des espèces, mais aussi pour pouvoir réaliser une simulation dans un contexte signifiant et non pour faire de l'art pour l'art.

8.3. SimPAMu

Comme je l'ai déjà mentionné au début, le domaine des simulations m'intéresse beaucoup. C'est pourquoi je n'ai pas pu m'empêcher d'implémenter moi-même un simulateur. Vu que le nombre d'agents simulable sur une seule machine est quand même assez restreint, je me suis même fait des idées sur une version distribuée...

Une autre remarque que je voulais faire à ce propos, c'est que j'ai mis beaucoup moins de temps à implémenter un simulateur opérationnel moi-même que d'acquérir les connaissances nécessaires à exploiter MOBIDYC, voir d'y implémenter mon modèle. Je suis tout à fait conscient du problème et aussi du fait qu'il ne faut pas essayer de réinventer la roue à chaque fois (d'ailleurs cela me fut déjà reproché à maintes reprises), mais je suis d'avis que, si on met plus de temps à utiliser quelque chose d'existant et de le transformer selon ses besoins que de l'implémenter soi-même, c'est qu'il y a un problème avec ce qui se dit être réutilisable.

¹¹ <http://www.uaml.de>

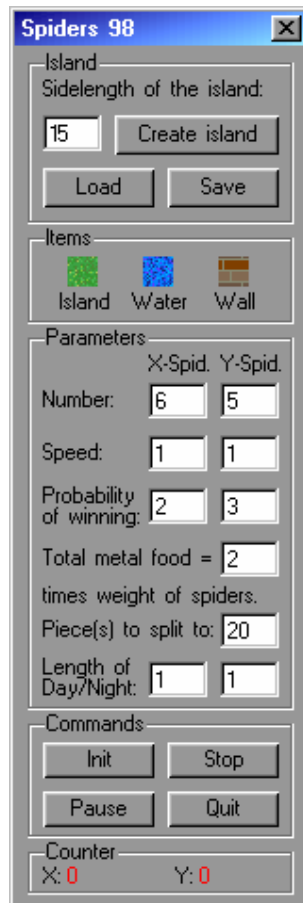
¹² <http://www.informatik.uni-hamburg.de/TGI/agenttechnology>

9. Bibliographie

- [BFS91] T. Bourbon, J. Ferber, and F. Samuel. Mages: A multi-agent testbed for heterogeneous agents. In Y. Demazeau and J.P. Muller, editors, *Distributed Artificial Intelligence Volume II*, pages 195-216. Elsevier Science Publishers B.V., Amsterdam, 1991.
- [CCS03] Cincom, Cincom Smalltalk, (page visitée le 26 novembre 2003), <http://www.cincom.com/scripts/smalltalk.dll/index.ssp>
- [CTE03] Centre de Technologie de l'Education, Concours Informatique Luxembourgeois, 24 octobre 2003 (page visité le 4 décembre 2003), <http://cil.cte.lu>
- [DOD03] Damien Olivier, Description d'un écosystème aquatique, (page visitée le 17 novembre 2003), <http://scott.univ-lehavre.fr/~olivier/Enseignement/DEA/ecosysteme.html>
- [GPS02] V. Ginot, C. Le Page, S. Souissi. A multi-agents architecture to enhance end-user individualbased modelling. *Ecological Modelling* 157 (2002) 23-41. Elsevier.
- [HBS02] D. Houssin, S. Bornhofen, S. Souissi et V. Ginot. Entre programmation par composants et langages d'experts: Rendre la modélisation individu-centrée plus accessible à l'utilisateur. *RSTI - TSI* 21/2002. Systèmes multi-agents pages 525 à 548.
- [HME51] H. Melville. *Moby Dick or, The Whale*. Editor Luther S. Mansfield Howard P. Vincent. Hendricks House, New York, 1952
- [PWL03] Peter William Lount, Smalltalk, (page visitée le 26 novembre 2003), <http://www.smalltalk.org>
- [RVG03] Prof. Dr. Rüdiger Valk & Group Theoretical Foundations of Computer Science, UAML, (page visitée le 5 décembre 2003), <http://www.uaml.de>
- [RVU03] Prof. Dr. Rüdiger Valk & Group Theoretical Foundations of Computer Science, AgentTechnology, 12 mai 2003 (page visitée le 5 décembre 2003), <http://www.uaml.de>
- [VGL03] Vincent Ginot, Le projet Mobidyc, 3 juin 2003 (page visitée le 19 novembre), http://www.avignon.inra.fr/internet/unites/biometrie/mobidyc_projet/version_index_html
- [WIL87] Wilson S.W., Classifier Systems and the Animat Problem. *Machine Learning*, 2: 199-228
- [URS98] De quelques structures sémiotiques des médias électroniques, *Cahiers pédagogiques*, À l'heure d'Internet, mars, 362, 26-28, 1998 sous le titre «Une révolution sémiotique».

10. Annexes

10.1. Spiders 98



Spiders '98

Now first of all, I want to show you what the user interface of **Spiders '98** is like. The reason why I called it **Spider '98** is that it looks like *Windows '98*, even if there is a little difference. Do you see it?

I took all procedures and functions out of a unit I called *Win98*. I wrote it for the *ULB '98*. I there had to write a program called **Othello**. Unfortunately I missed the date to send it in, so I didn't participate, but you may download a copy of **Othello '98** from my homepage at <http://surf.to/ivans>. [broken link]

Now I took this unit, added some features and created some new objects. Notice that **Spiders '98** is a *DOS* based program, which uses *BGI* graphic interface, so don't try to make screenshots of it.

For any further comments please mail me at ivan@netlimit.com. [broken link]

The Island-Panel:

Use the edit field to enter the sidelight of the island you want to create. Click on it to edit it. The entered number must be a variable of type integer between 2 and 60.

I could have increased this, but because I was limited to a screen resolution of 640x480 and because I wanted to have clear graphical interface, I didn't. Notice that the size of each item will change automatically depending on the side length.

Click next on the "Create Island"-button to make your island visible. You could save it, by giving it a name, which shouldn't be longer than 8 characters. Load it by using the "Load"-button.

I added this features because, while writing and testing the program, I didn't always want to recreate my test maps. Please try the "patience" map I included!

The Items-Panel:

Click with the left mouse button on one of the icons to select it. Click now on your map, to place it there. Notice that a right mouse click will always put an icon of the type "Island" on the map

Other Items:

an X-spider.



an Y-spider.



a dead spider. It appears if an attacking spider loses a fight.

This is just another added extra.



metal food.

Parameters

	X-Spid.	Y-Spid.
Number:	6	5
Speed:	1	1
Probability of winning:	2	3
Total metal food =	2	
times weight of spiders.		
Piece(s) to split to:	20	
Length of Day/Night:	1	1

The Parameters-Panel:

This panel is used to enter all parameters for the game. Number will give the number of each sort of spiders. Speed is how many moves the spiders should do during one second. Probability of winning if for X-spiders the probability they win a fight during the day and for Y-spiders the probability they win a fight during the night. Total metal food is the metal food given relatively to the total weight of all the spiders. Pieces to split to is the number of metal pieces that should be placed on the island. Length of Day/Night is the length of day and night expressed in seconds. Notice that every entry should be a variable of the type *integer*, expect the Total metal food, which could be a *real*; for example 3.25.

X-spider and Y-spider number is restricted to 1999 spiders of each sort. Notice that this number depends of the system memory and could be less on older machines. The number of metal pieces is limited to 1000. Unfortunately I was unable to test my memory management procedures, so I don't know whether they are correct or not. If there are too many spiders on the map, length of day and night isn't real ok, so seconds become a bit longer...

Commands

Init	Stop
Pause	Quit

The Commands-Panel:

When clicking on the button *Init* simulation starts. *Stop* actually stops it. *Pause* is obvious too. If you click on this button, simulation pauses and a dialog will appear which tells you how many spiders are on the island. *Quit* quits the program without asking if you want to stay longer or whether you're sure to quit.

The count method, when clicking on the pause button is different than the one used by the counter, so this represents fine way of checking whether everything's alright.

Counter

X: 0	Y: 0
------	------

The Counter-Panel:

It continuously shows how many spiders of each sort are on the island.

Information:

This program has been written by Fisch Bob with Borland™ Turbo Pascal on a Pentium II 300 with 64 RAM for CIL '98 and tested as well on my old 486 DX 2/66 with 20 RAM.

There was only a minimal difference between Pentium II and 486, which is so small that it's rather unnecessary to mention it.

10.2. Guide utilisateur SimPAMu 2

SimPAMu 2 est constitué de différentes fenêtres. Une des plus importantes est celle qui permet d'éditer les constantes relatives au comportement des individus.

Elle permet de mettre en place des configurations bien déterminées, de les sauvegarder et de les charger ultérieurement.

Il faut noter que les valeurs peuvent également être changées lorsqu'une simulation est en cours. Les nouvelles valeurs sont immédiatement prises en compte par le système, sauf pour tout ce qui est valeur initiale.

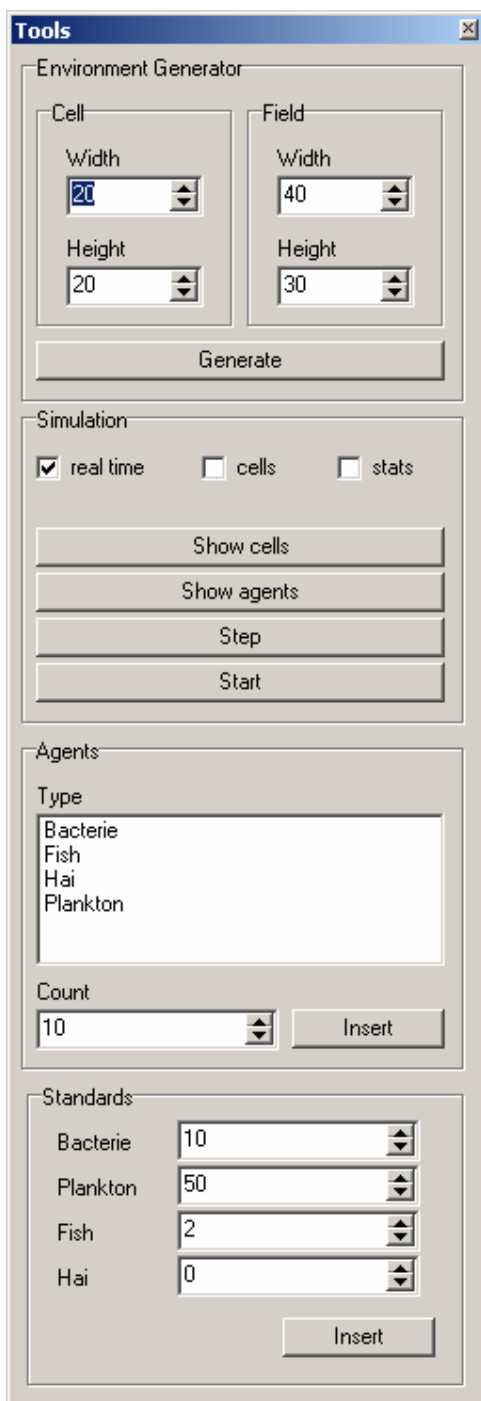
La deuxième fenêtre qui joue un rôle très important est celle qui permet de contrôler toute la simulation.

Dans le cadre supérieur on a la possibilité de définir la taille de la grille ainsi que la taille des cellules. En cliquant sur le bouton «Generate», une grille de cellules est générée. Chaque cellule contient les valeurs initiales spécifiées dans l'éditeur des valeurs. Si jamais, il y avait encore une simulation qui était en cours, elle est stoppée. De même, tous les agents restant sont supprimés.

Le prochain cadre permet de contrôler le déroulement propre de la simulation. Le bouton «Start/Stop» permet de démarrer ou d'arrêter la simulation. Le bouton «Step» effectue le calcul d'une unité de temps. Le bouton «Show agents» affiche tous les agents et le bouton «Show cells» met à jour la couleur des cellules. Ces deux boutons sont le plus utile si on fait tourner une simulation en mode affichage déconnecté.

Key	Value
PLANKTON_SALT_MAX	5
PLANKTON_REPRODUCTION	50
PLANKTON_DIE	100
PLANKTON_VIEW_SALT	5
BACTERIE_OXYGENE_MAX	5
BACTERIE_ORGANIC_MAX	5
BACTERIE_REPRODUCTION	80
BACTERIE_DIE	150
BACTERIE_VIEW_OXYGENE	5
BACTERIE_VIEW_ORGANIC	5
FISH_OXYGENE_MAX	40
FISH_FOOT_MAX	40
FISH_REPRODUCTION	5
FISH_FISH_ORGANIC	10
FISH_PLANKTON_FOOT	19
FISH_DIE	300
FISH_VIEW_ENNEMY	10
FISH_VIEW_PREY	10
FISH_VIEW_OXYGENE	10
HAI_OXYGENE_MAX	150
HAI_FOOT_MAX	200
HAI_REPRODUCTION	30
HAI_DIE	750
HAI_FISH_FOOT	40
HAI_PLANKTON_FOOT	19
HAI_HAI_ORGANIC	10
HAI_VIEW_PREY	10
HAI_VIEW_OXYGENE	10
CELL_INIT_OXYGENE	20
CELL_INIT_ORGANIC	50
CELL_INIT_SALT	50

Save as Save Load



est représenté par une image donnée. La couleur de chaque cellule indique son état. Si elle est plutôt bleue, elle contient un surplus d'oxygène, rose indique un surplus de sels minéraux et vert un surplus de déchets organiques.

Il existe en haut trois cases à cocher pour faire cela. La première indique si les agents doivent être affichés en temps réel, c'est à dire en même temps que les calculs se font. La deuxième permet de définir si les cellules doivent aussi être affichées en temps réel. Le mode d'affichage déconnecté est actif lorsque ces deux cases sont déconnectées. La troisième «stats» indique si oui ou non le système doit produire une courbe indiquant le nombre d'individus par espèce par unité de temps.

Ces trois cases peuvent également être cochées ou décochées, même si une simulation est en cours. Étant donné le fait que l'affichage des agents est très gourmand en ressources, cela offre donc la possibilité d'accélérer la simulation fortement, et même de déconnecter l'affichage si un certain nombre d'agents présents dans le système est atteint.

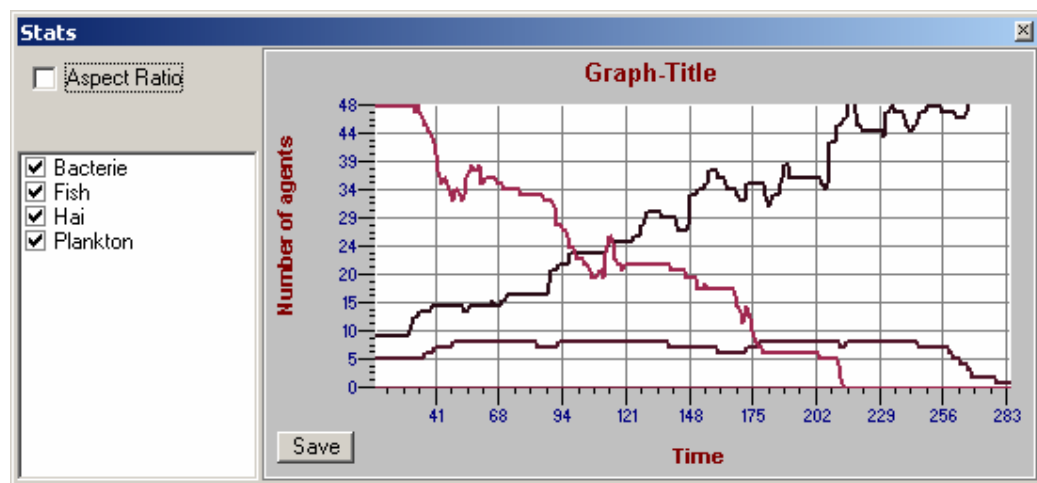
Les deux derniers cadrans servent à placer des agents dans le système. À l'aide du premier, on sait insérer type par type, tandis que le deuxième permet de placer en un clic une certaine configuration.

La fenêtre principale affiche l'environnement dans lequel sont plongé les animats. Chacun d'eux



Le titre indique à quel instant se trouve la simulation de même que le nombre total d'agents présent et actifs dans le système. Aussi peut-on y lire si actuellement il fait jour ou nuits.

En bas à gauche se trouve une fenêtre contenant les courbes représentant le nombre de chaque type d'individu par unité de temps. Le bouton «Save» permet de sauvegarder le graphique. En cochant ou décochant dans la liste à gauche les types d'individus, les courbes apparaissent ou disparaissent. Une fois la simulation arrêtée, on peut agir à l'aide de la souris sur le graphique en l'étirant par exemple. La case «Aspect ratio» indique si les mouvements doivent se faire en gardant les proportions ou non.



La dernière fenêtre fonctionne comme sortie pour toute sorte de messages. Elle indique ici la cause de la mort de certains agents. Cela permet par exemple, de comprendre pourquoi une simulation a eu telle et telle fin et donc de changer les paramètres correspondants.