

Mobidyc

Une plate-forme multi-agents pour la création et l'utilisation de modèles individus
centrés dédiés à la dynamique des peuplements et aux automates cellulaires

Tutorial

Vincent Ginot et Sami Souissi, avril 2003

Avertissement

Remarque importante : *Mobidyc version 2.2 inclut de nouvelles fonctionnalités non présentes dans ce tutorial. En particulier, pensez à utiliser la primitive MathScript pour toutes vos formules mathématiques, en lieu et place de l'ancienne primitive ModifierAttribut. MathScript est un module Java qui requiert Java runtime environnement, version J2SE 1.4.2 ou plus récente (téléchargement sur <http://java.sun.com/j2se>). Consultez l'aide en ligne, notamment la rubrique 'nouvelles fonctionnalités' pour faire connaissance avec la nouvelle version de Mobidyc..*

Ce tutorial est destiné à montrer les potentialités de Mobidyc, un simulateur destiné au biologiste, afin de l'aider à construire et à manipuler des modèles individuels centrés dédiés à la dynamique des peuplements.

Il est bien évident qu'un tel simulateur ne représente pas la seule réponse à la question de la modélisation des peuplements. Il repose en particulier sur le concept d'agent qui implique une discrétisation du temps et de l'espace, et des choix de synchronisation parfois délicats. Dans toutes les situations où les aspects individuels ne sont pas prépondérants, ou lorsqu'un espace hétérogène n'est pas à prendre en compte, il sera presque toujours préférable de garder une approche plus classique, à base d'équations différentielles ou d'algèbre matriciel par exemple, qui offre des résultats numériques sûrs et un formalisme mathématique très utile pour faire l'analyse théorique du modèle. Dans le domaine des modèles à compartiments, c'est-à-dire où le système peut se décomposer en boîtes entre lesquelles transitent des flux, le déjà ancien simulateur Stella™, qui depuis a fait beaucoup d'émules, représente à nos yeux l'archétype de l'outil d'aide à la modélisation, l'utilisateur étant maître de créer et de manipuler son modèle. Mobidyc s'en veut en quelque sorte le pendant, pour les systèmes dont la dynamique dépend fortement d'aspects individuels ou spatiaux, et où la notion d'agent devient de ce fait importante.

Ce travail, initié en 1996 à la station d'Hydrobiologie Lacustre de Thonon les Bains, a été en partie financé dans le cadre du programme CNRS Environnement Vie et Sociétés "Biodiversitas" à travers une action thématique du GIP HydrOsystemes. Un groupe de travail associant principalement, outre l'INRA, le laboratoire Green/TERA du Centre de coopération internationale en recherche agronomique pour le développement (CIRAD), les laboratoires HEA et LIA de l'Institut de la Recherche pour le développement (IRD), et le laboratoire d'informatique de Paris VI (LIP6, Université Pierre et Marie Curie) a fortement contribué à l'émergence des idées contenues dans ce logiciel.

Charte d'utilisation

Mobidyc est un logiciel gratuit à usage non commercial, sous licence GPL. Il utilise le moteur Smalltalk VisualWorks de Cincom, d'usage gratuit pour les applications non commerciales.

Vous pouvez librement télécharger Mobidyc et vous pouvez le diffuser à condition de toujours fournir l'archive complète du logiciel ainsi que la présente charte.

Aucune formalité ne vous est demandée. Mais si vous en avez l'usage, nous vous suggérons de nous retourner vos coordonnées, vos suggestions, et le type de modèle que vous développez, en vous inscrivant sur la liste des utilisateurs que vous trouverez sur notre site. Ceci dans l'esprit de développer progressivement un club d'utilisateurs et de développeurs. Les informations ne seront ni diffusées, ni utilisées à des fins commerciales.

De même, merci de citer la référence au logiciel en cas de publication.

Le code source est également disponible pour ceux qui en font la demande, notamment pour adapter un modèle trop complexe pour le prêt-à-porter standard de Mobidyc, ou pour participer aux développements de nouveaux modules.

Mobidyc ne fait l'objet d'aucune garantie. Tous les risques de son usage sont assumés par l'utilisateur.

Vous testez la première version de Mobidyc : soyez indulgents, mais néanmoins critiques et faites nous rapidement part des difficultés rencontrées et de vos suggestions.

Pour toute information ou nouveauté consultez le site : <http://www.avignon.inra.fr/mobidyc>

Sommaire

La plate-forme Mobidyc	5
Résumé	5
Installation et lancement	5
L'interface principale	6
Présentation générale	6
Le menu principal	6
Création d'un projet	8
Notion de projet	8
L'exemple Sugar Scape, fil conducteur de ce tutorial	8
Créer un espace	8
Donner des attributs (un état) à l'espace	9
Définir un observatoire	11
Créer un agent qui cherche le sucre	13
Lancer une première simulation	16
La visualiser avec le magnétoscope	19
L'agent mange le sucre	19
Le 'scheduler', ordonnanceur de la simulation	22
Visualiser une chronique	24
"Dialogues" entre agents	25
L'agent "tond" le sucre de la cellule	25
L'agent respire et meurt	26
La cellule restaure son sucre	29
Les agents non situés	31
Définition	31
Création d'un agent non situé	31
Création d'une tâche utilisateur	32
Faire jouer plusieurs agents : créer des agents différents à partir du même moule	35
Les agents se reproduisent	37
Assembler des primitives pour former des tâches : l'exemple chèvre-loup	43
Principe	44
Cohérence entre arguments	44
Les différents types d'objets manipulés par les primitives	44
Les différents types de primitives	45
Un loup poursuivant des chèvres	47
De l'importance du mode d'appel des agents (synchronisation)	51
Enchaîner des expériences simulatoires. Le mode 'Batch'	52
Codage traditionnel d'une tâche	57
Gestion de la synchronisation	59
Conclusion	66
Les fichiers utilisateur gérés par Mobidyc	66
Les fichiers projet	67
Les fichiers d'importation ou d'exportation de données	68
Les fichiers Mobidyc d'intérêt pour l'utilisateur	70

La plate-forme Mobidyc

Résumé

Mobidyc, acronyme de MOdélisation Basée sur les Individus pour la Dynamique des Communautés, est un outil destiné à aider un utilisateur non informaticien à créer, utiliser, et faire évoluer ses propres modèles dans le domaine de la dynamique des peuplements. Basé sur le concept agent, il est bien adapté à la modélisation des systèmes pour lesquels des aspects individuels ou spatiaux sont à prendre en compte. Partant de l'agent de base très simple fournit par la plate-forme, l'utilisateur crée ses propres agents, agents de la communauté biologique, agents cellules formant l'espace, ou agents non-situés responsables de tâches plus générales, en ajoutant de nouveaux attributs pour compléter leur état et des tâches pour fixer leur comportement. Pour ce faire, il utilisera les composants prédéfinis et paramétrables du simulateur, mais pourra également créer ses propres tâches en utilisant l'outil d'assemblage de « primitives ». Un outil de codage plus traditionnel des tâches (utilisation directe du langage de programmation Smalltalk) est également proposé. Mobidyc gère les variables quantitatives avec leurs unités, les modes de synchronisation séquentiel et (pseudo)parallèle entre agents, ainsi que l'importation de divers fichiers ASCII pour fixer des scénarios prédéfinis sur certains paramètres et charger automatiquement un peuplement initial ou encore les cartes de valeurs d'une grille environnementale. Si un espace est utilisé, à deux dimensions et selon une grille régulière ou constitué de cellules de formes quelconques (cette dernière option actuellement désactivée), un "observatoire" permet de définir ce que l'on souhaite visualiser durant la simulation. La sauvegarde des résultats, ainsi que la gestion de la synchronisation, est fondée sur le principe que chaque agent est doublé par son "historique" chargé de mémoriser son état (la valeur de ses attributs) à chaque pas de temps. Il est ainsi virtuellement possible de rejouer une simulation, et la visualisation et l'exportation des résultats vers un tableur s'en trouve facilitée. Une procédure, particulièrement utile pour les analyses de sensibilité, permet d'enchaîner automatiquement des expériences simulatoires (batch). Diverses interfaces permettent de filtrer les résultats à sauvegarder.

Installation et lancement

Décompresser l'archive. Elle contient sept répertoires.

Le répertoire image contient les exécutables

Le répertoire projets est vide, il est prévu pour recevoir vos modèles

Le répertoire MB_exemples contient des exemples en français

Le répertoire MB_exemples contient des exemples en anglais

Le répertoire MB_beispiele contient des exemples en anglais

Le répertoire MB_plugin contient les mises à jour du logiciel (corrections, nouveaux modules)

Le répertoire messages contient l'aide en ligne et les informations pour la gestion multi-langue.

Il n'est procédé à aucune installation et Mobidyc ne change en rien l'organisation de votre disque dur. Si vous voulez désinstaller le logiciel, effacez simplement ces répertoires.

Pour lancer Mobidyc, exécuter Mobidyc.im qui se trouve dans le répertoire image puis référez vous à l'aide en ligne ou à ce tutorial.

L'interface principale

Présentation générale

L'interface principale de Mobidyc est composée d'un menu principal et d'une zone de contrôle de simulation (Fig. 1). Mobidyc propose deux modes de simulations. Le premier est interactif et est destiné à la mise au point du modèle. Le deuxième est automatique (procédure Batch) et permet de définir différents plans d'expériences. Un module de dépouillement de ces expériences est en cours de développement avec le logiciel de statistiques 'R'. Nous consulter.

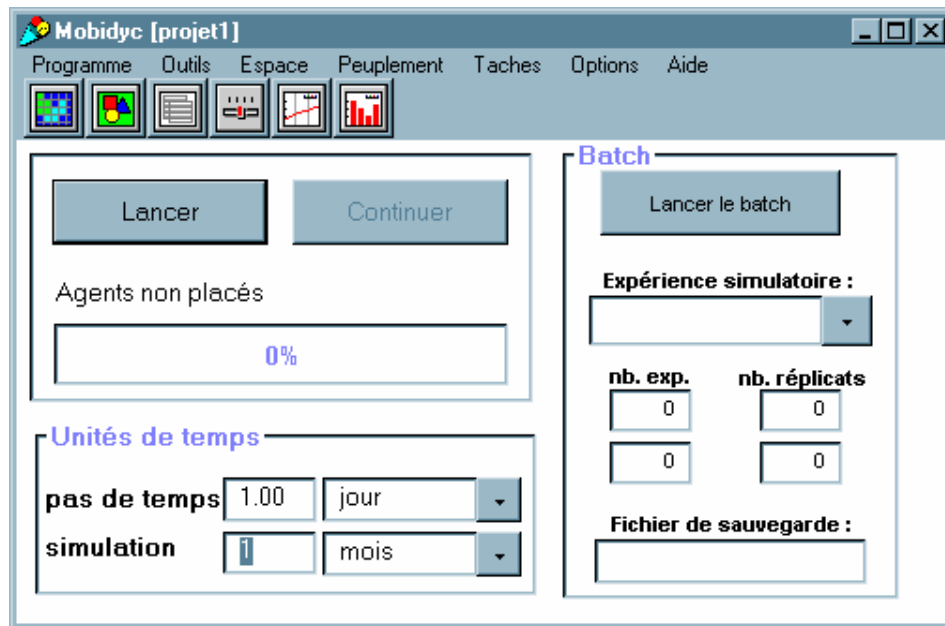
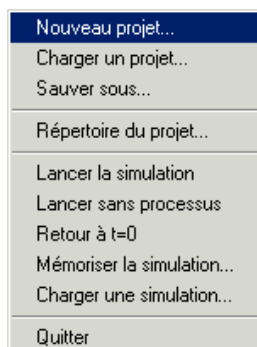


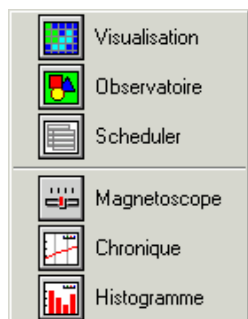
Fig.1. Interface générale de Mobidyc composée d'un menu principal (en haut), des outils de visualisation et de contrôle de simulation (en dessous du menu principal) et de deux zones de contrôle de simulations selon un mode interactif (à gauche) et selon un mode d'expériences simulatoires 'le batch' (à droite).

Le menu principal



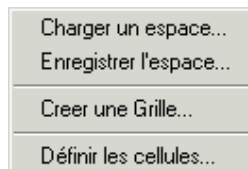
Programme

Ce menu permet de gérer un projet (création, chargement et enregistrement d'un projet). Il permet également de lancer la simulation en deux modes, avec et sans processus. Le premier est l'équivalent du bouton « lancer », le deuxième est destiné aux développeurs pour faciliter le debugage. Retour à t=0 permet de réinitialiser la simulation sans la relancer.



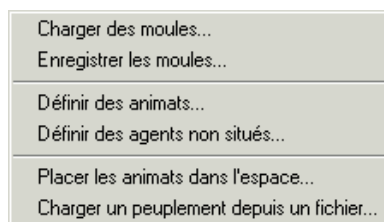
Outils

Plusieurs outils de visualisation et du graphisme sont disponibles sous Mobidyc. Nous découvrons l'utilisation et l'utilité de ces différents outils lors de la présentation des exemples.



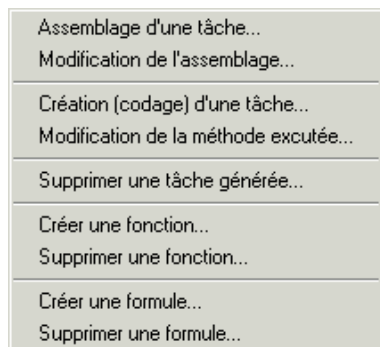
Espace

1- Chargement et enregistrement de l'espace sous la forme d'un fichier texte (fichier .esp). Actions automatiquement effectuées lors du chargement/sauvegarde de votre projet. Ce fichier peut-être importé d'un tableur ou d'un SIG, mais attention de parfaitement respecter le format de Mobidyc. 2- Création de l'espace, une grille avec des formes régulières carrés ou hexagonales (un module « lecture de cellules de formes quelconques est actuellement désactivé. Nous consulter. 3- Définition des propriétés (attributs et taches) des cellules.



Peuplement

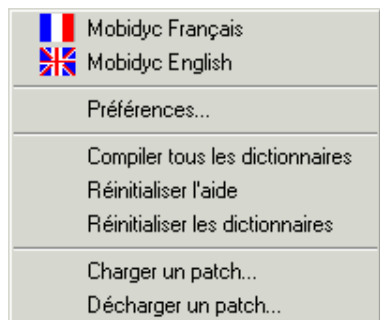
Les autres agents, subdivisés en deux catégories. Les agents situés (ou animats) qui sont les individus au sens classique, et les agents « non situés » souvent utilisés pour des tâches d'intérêt général. Le fichier (binaire) correspondant au descriptif des agents (y compris les cellules) est le fichier .ppl. Ce menu permet également de placer (et initialiser) les animats initiaux, soit manuellement, soit à partir d'un fichier ASCII (fichier .chg)



Taches

Ce menu propose un outil de personnalisation des tâches qui consiste, dans sa version la plus simple, à enchaîner des sous-tâches (que nous appellerons primitives) qui, elles, sont prédéfinies. Mobidyc va en faire une nouvelle classe qui sera conservée au même titre que les tâches prédéfinies. Un outil de codage plus classique existe aussi pour ceux qui connaissent Smalltalk. Ce menu permet également de créer de nouvelles fonctions mathématiques (en plus des fonctions prédéfinies) ou de nouvelles formules paramétrables.

Un nouveau module de composition de tâches, plus souple et plus direct, est en cours de test. Il est chargeable via le répertoire MB_plugin.



Options

Le menu des options permet de lancer Mobidyc en plusieurs langues (actuellement en français et en anglais). Ajouter une autre langue n'est pas très difficile, avis aux amateurs de traduction. Attention, un modèle écrit dans une langue doit tourner dans sa langue d'origine (d'où les deux répertoires d'exemples, français et anglais). Les menus dictionnaires et aides sont pour les développeurs. Charger / Décharger un patch permet de faire évoluer le logiciel entre deux révisions majeurs. Ces modules de corrections / évolutions seront disponibles sur notre site, et à placer dans le répertoire MB_plugin pour pouvoir être chargés et utilisés.

à propos de Mobidyc...
aide

Aide

Une aide en ligne est proposée pour expliquer les différents menus et les différentes étapes de construction d'un modèle sous Mobidyc.

Création d'un projet

Notion de projet

Un projet permet de gérer l'essentiel des fichiers d'un modèle sous Mobidyc. Il gère 5 fichiers :

projet.mbp : fichier binaire du projet

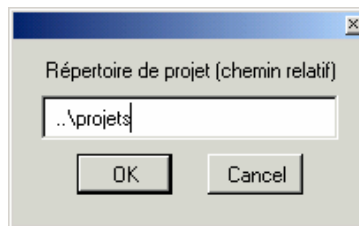
projet.esp : fichier ASCII de l'espace (vous pouvez donc le modifier « à la main » et le recharger)

projet.ppl : fichier binaire du descriptif des agents (les moules)

projet.pdv : fichier binaire du point de vue (comment visualiser les agents)

projet.chg : fichier ASCII de chargement initial des animats, équivalent du .esp. Vous pouvez donc également construire votre fichier de chargement à l'aide d'un tableur par exemple.

Par défaut, Mobidyc se trouve dans le répertoire MB_exemples. Nous vous recommandons de laisser ce répertoire inchangé et de mettre vos nouveaux fichiers dans le répertoire 'projets' de Mobidyc. Menu 'Programme -> changer répertoire du projet'. Vous pouvez mettre un chemin absolu ou relatif. Ex ici pour un PC. Pour un mac faire ::projets, et ../projets sous Unix.



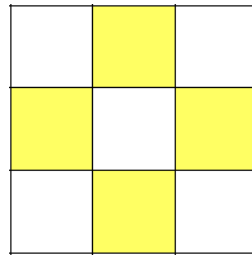
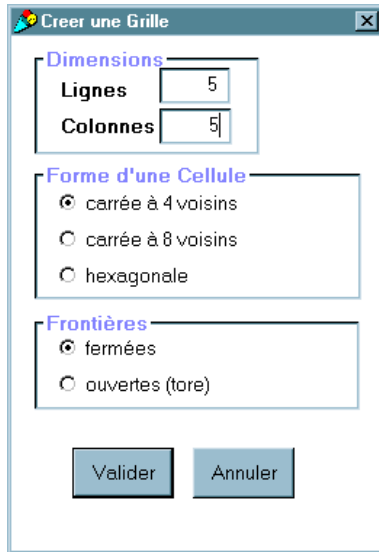
Puis créez un nouveau projet 'Programme -> Nouveau Projet, par exemple 'Tutorial', sans extension. (Par défaut, un projet est automatiquement créé de toutes façons.

L'exemple Sugar Scape, fil conducteur de ce tutorial

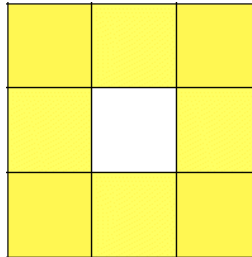
Le premier exemple que nous suivrons dans ce tutorial est celui de "Sugar Scape" développé par des économistes (Epstein & Axtell, 96) : des agents mobiles cherchent et broutent du sucre. Mais ils respirent et peuvent mourir de faim. Seuls les plus chanceux (situés dans les zones les plus riches) ou les mieux adaptés (faible respiration, grand rayon d'action de déplacement) survivent. Ils se reproduisent aussi, selon différents scénarios. Dans sa version la plus complète deux ressources sont disponibles, l'une commune et bon marché (le sucre), l'autre rare et chère (l'épice), et des échanges peuvent s'instaurer entre agents. Les auteurs de ce modèle montrent qu'en partant de mécanismes très simples on observe l'émergence de "phénomènes de société" intéressants. Pour de plus amples précisions sur SugarScape, consultez <http://www.brookings.org/sugarscape/default.htm>

Créer un espace

Un des intérêts de Mobidyc est de permettre la mise au point graduelle de son modèle, en ne le complexifiant que lorsqu'on est sûr de la partie déjà en place. Nous allons donc commencer par développer notre modèle Sugar Scape avec un seul agent, sur un support spatial réduit : une grille carrée 5x5 fermée à quatre voisins. Pour cela menu Espace → Créer une grille




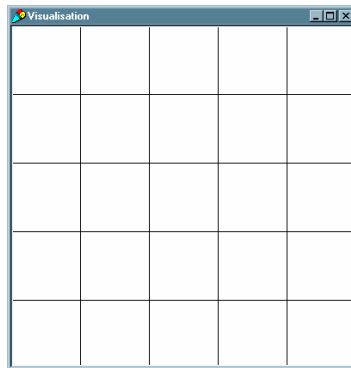
Cellule à 4 voisins



Cellule à 8 voisins

Interface de création d'une grille à cellules carrées ou hexagonales

Valider puis visualiser la grille à l'aide du bouton visualisation (→ Menu Outils) 



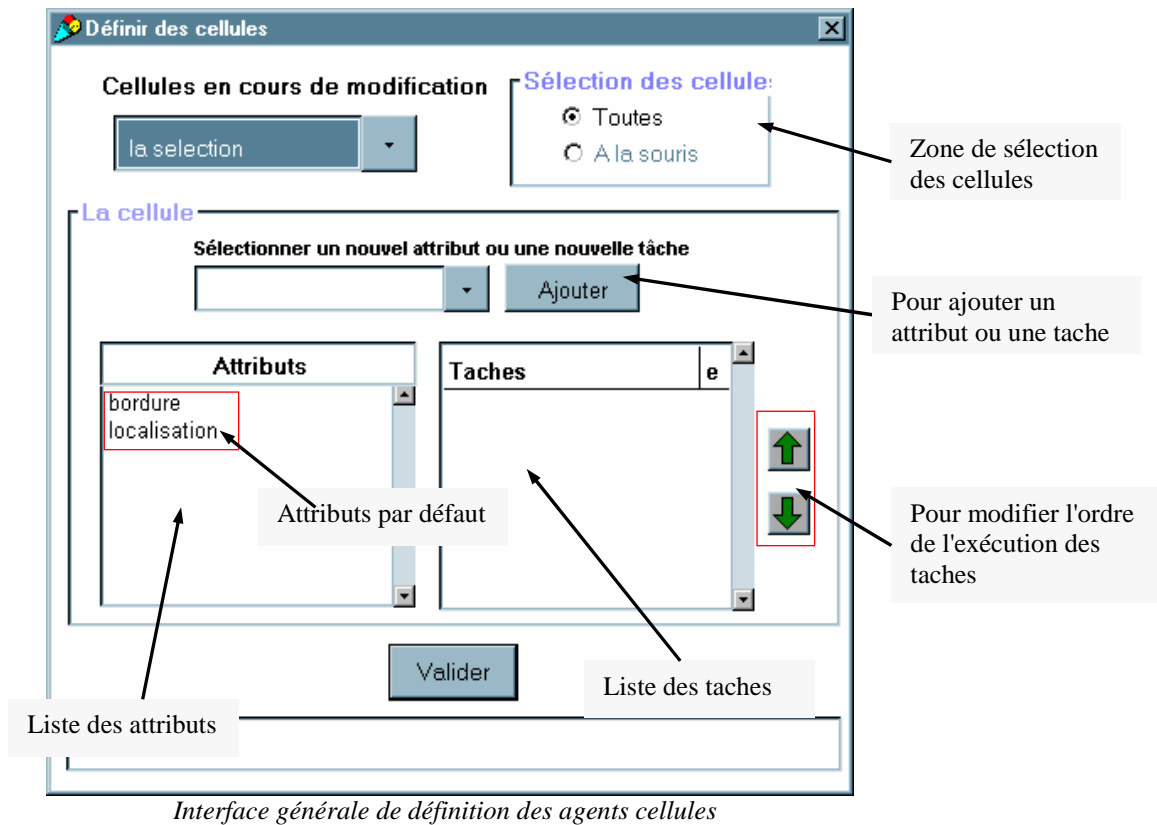
fenêtre de visualisation d'une grille 5 × 5

Gestion de la sélection des cellules dans la grille :

Une cellule = simple clic
 Toutes les cellules = shift clic (ou bouton droit PC), alt clic (MAC), bouton milieu (Unix)
 Additionner des cellules = shift clic
 Toutes sauf = ctrl shift clic
 Rien = cliquer à l'extérieur de la grille

Donner des attributs (un état) à l'espace

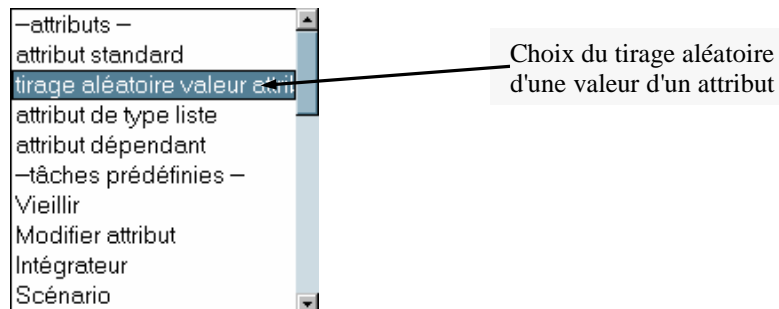
Commençons par mettre du sucre dans les cellules de cette grille. Pour cela il faut créer un attribut 'sucre' qui donnera la quantité de sucre de chaque cellule, et donc appeler la fenêtre de définition des cellules. Menu Espace → Définir les cellules



Noter que pour l'instant, les seuls attributs de nos cellules sont "bordure" (avec deux modalités "1" et "0" selon que la cellule se trouve en bordure de grille ou non) et localisation (identifiant de la cellule qu'il ne faut pas modifier). Nous allons, pour chaque cellule, créer un attribut sucre et lui affecter une valeur. Pour cela vérifier que toutes les cellules sont sélectionnées (bouton radio "toutes" et menu "la sélection (complète)" activé) puis presser "Sélectionner un nouvel attribut ou une nouvelle tâche".

La liste des attributs et tâches disponibles est organisée en trois grands groupes :

- 1- les attributs ;
- 2- les tâches prédéfinies ;
- 3- les tâches créées par l'utilisateur (liste normalement vide au premier chargement de Mobidyc et qui s'enrichit progressivement).



liste des attributs et des tâches prédéfinies disponibles pour une cellule


Il ne faut pas oublier de cliquer sur le bouton 'Ajouter' pour valider le choix de l'attribut ou de la tâche. Dans cette liste, on pourrait sélectionner "attribut standard" et donner un nom et une valeur à cet attribut. Mais toutes les cellules sélectionnées auraient alors le même niveau de sucre. On utilise donc une fonction plus spécifique aux cellules (on ne la retrouvera pas lors de la définition des animats) : "tirage aléatoire d'une valeur". Ajoutez le à la liste des attributs de votre agent, ceci donnera la fenêtre ci-dessous:

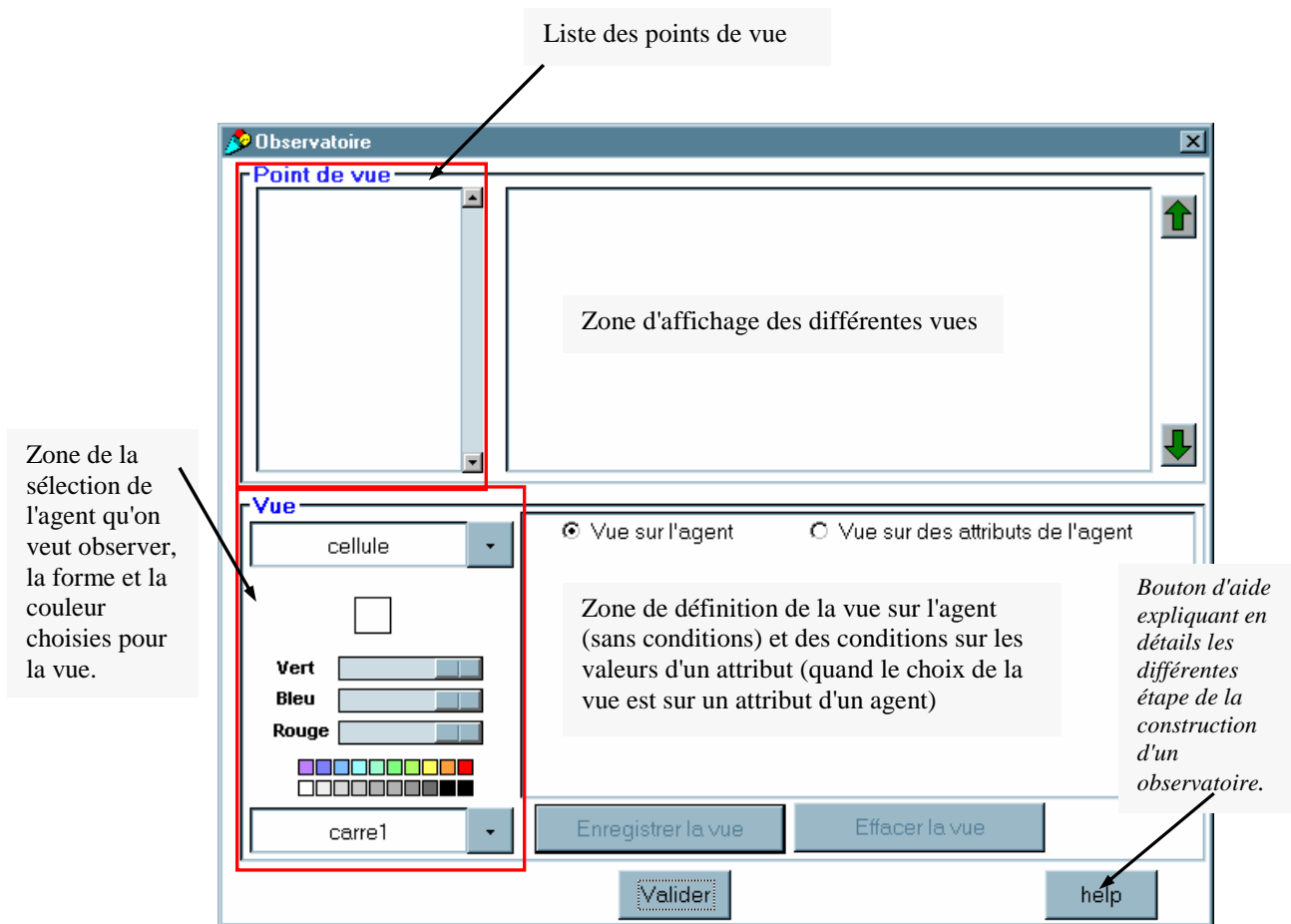
fenêtre du tirage aléatoire d'une valeur d'un attribut 'sucre'

Nommer cette variable "sucre" et un intervalle de valeurs de 0 à 4, conformément à l'exemple Sugar (l'unité n'a pas d'importance). Valider. L'attribut "sucre" doit apparaître dans la fenêtre des attributs des cellules. Quitter en validant les modifications.

- ☞ Une alternative, lorsque les grilles sont grandes ou que l'on travaille sur des données réelles, est de charger directement un fichier de données pour les cellules, issu d'un SIG par exemple. Fichier de type .esp (pour espace)
- ☞ Dans la fenêtre d'ajout de composants, noter également l'option "Scénario" dans tâches prédéfinies (une liste de valeurs dans le temps). Typiquement, un scénario sera la température journalière d'une cellule, importé d'une base de donnée par exemple.

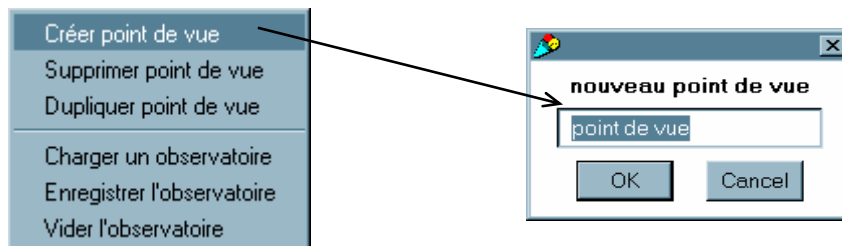
Définir un observatoire

Pour l'instant la grille, bien qu'en principe aléatoirement pleine de 4 niveaux de sucre, est désespérément blanche. Mobidyc ne pouvant pas deviner ce que vous voulez voir, il faut définir son point de vue. Pour cela, appeler l'interface de l'observatoire (Menu Outils → Observatoire, ou bien cliquer directement sur le bouton de l'Observatoire  dans la fenêtre principale). La fenêtre ci-dessous apparaît:



Fenêtre de l'observatoire avant remplissage

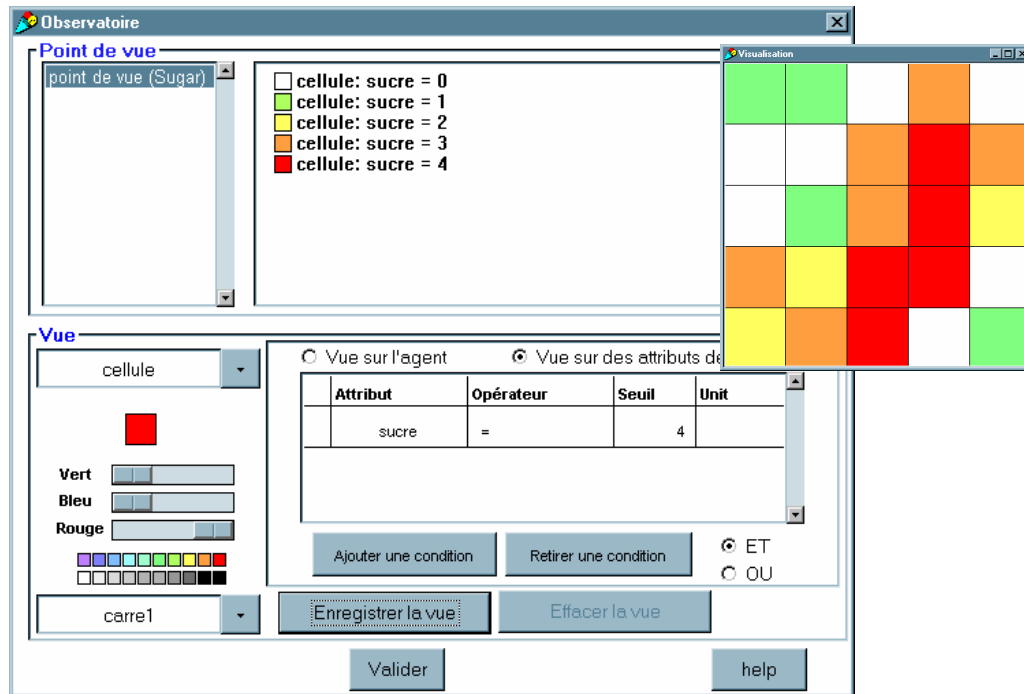
Il faut d'abord commencer par créer un point de vue : Menu contextuel dans la zone "Point de vue" → Créer un point de vue :



menu contextuel de "Point de vue" et le choix d'un nouveau point de vue

Le nom du nouveau point de vue ("point de vue" par défaut) apparaît dans la zone Point de vue (en haut et à gauche).

On veut ici un point de vue sur un attribut d'un agent cellule (le sucre). Cocher ce bouton puis fixer les conditions à remplir sur cet attribut. Fixer une couleur, prédéfinie ou en jouant sur les curseurs. On conseille, pour les cinq niveaux croissants de sucre les couleurs blanc, vert, jaune, orange, rouge, ou alors des niveaux de gris. **Attention, Les éditeurs de tableaux smalltalk sont un peu sourds : cliquez 2 fois dans la case pour pouvoir insérer votre chiffre.** On peut mettre plusieurs conditions (elles sont de type "ET" ou "OU") ce qui permet d'affiner la condition si nécessaire. Quand conditions et couleurs sont correctes, cliquer "Enregistrer la vue". Il doit s'afficher dans la liste des vues. Une fois le tout rempli vous devez avoir quelque chose qui ressemble à la figure ci dessous:



Fenêtre de l'observatoire après remplissage et de la visualisation de la carte

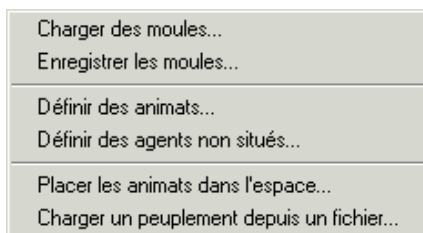
Presser "Valider" et visualiser votre grille (bouton Visualisation du menu "Outils"), vous devez obtenir une carte qui ressemble à celle montrée ci-dessus:

Vous pourrez ultérieurement définir d'autres points de vues, enregistrer et charger un observatoire (en utilisant le menu contextuel). Un observatoire (fichier .pdv) peut comporter plusieurs points de vues. Vous pouvez créer un nouveau point de vue ou bien utiliser l'option 'Dupliquer point de vue' du menu contextuel.

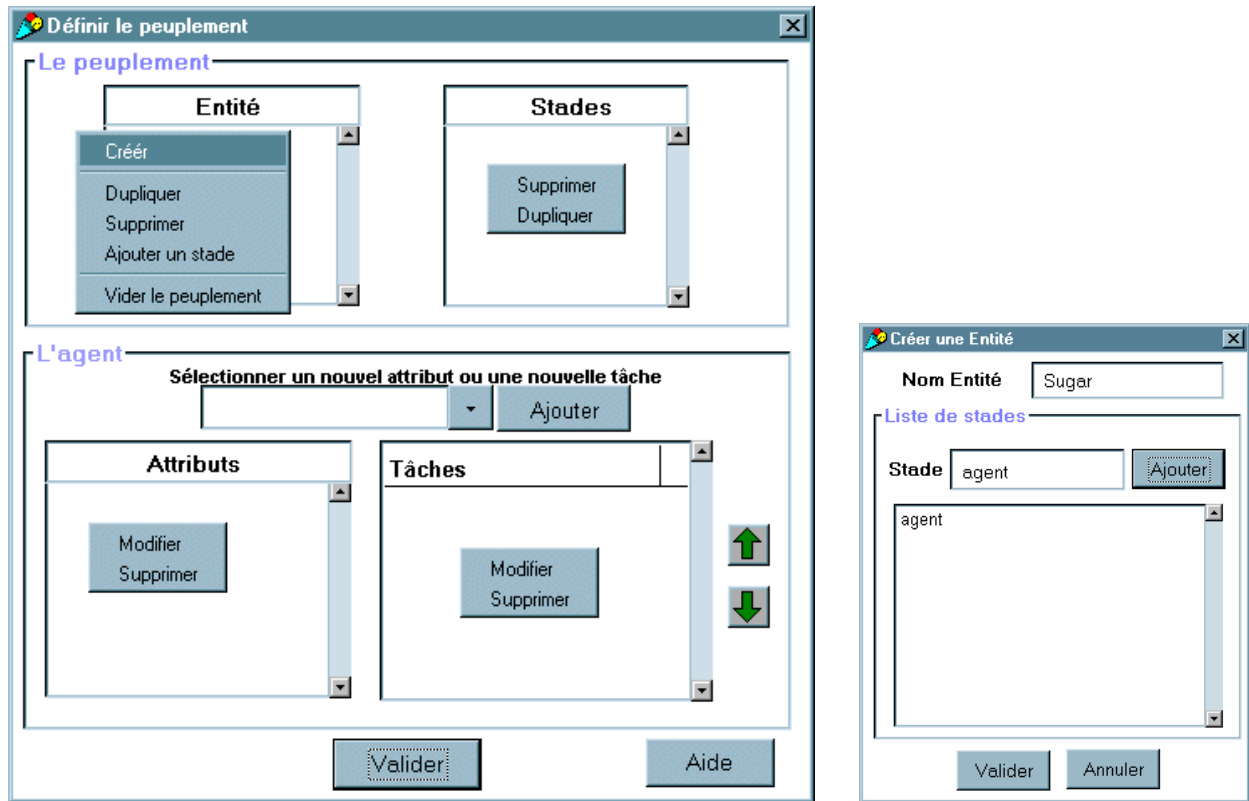
Avant de passer aux animats, enregistrer votre projet, et ayez la curiosité d'éditer votre fichier d'espace 'projet.esp' avec Excel par exemple pour voir son allure, Vous pouvez essayer de le modifier à la main. Sauvez le sous un autre nom (format texte, séparateur tabulation, suffixe .esp), et visualisez vos modifications en chargeant ce fichier via le menu Espace -> charger un espace. Vous pouvez choisir ou non de revenir à votre espace initial en rechargeant le fichier correspondant.

Créer un agent qui cherche le sucre

Maintenant que nous avons un espace et du sucre, nous allons créer un agent qui va partir à sa recherche. Sélectionner "Menu Peuplement → Définir des animats..."



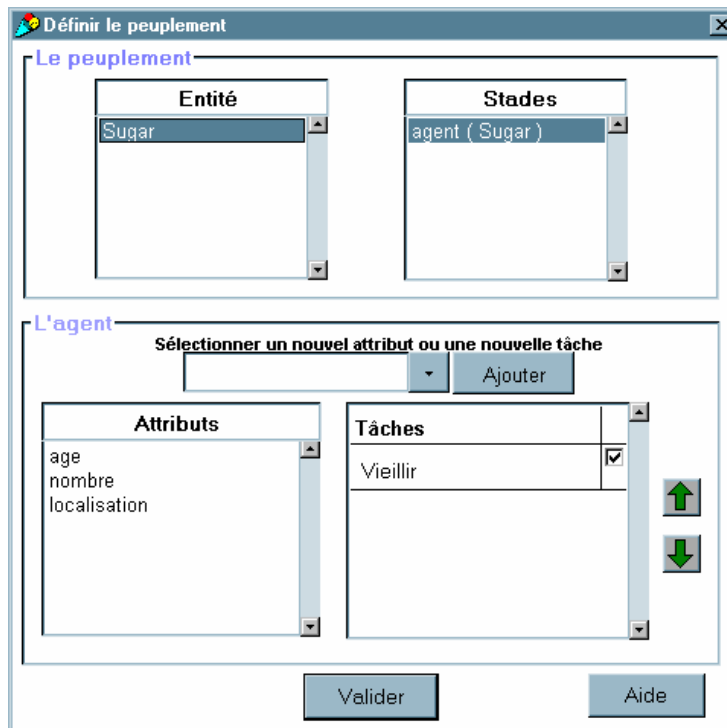
La fenêtre générale de définition des animats apparaît, fort logiquement très semblable à celle de définition des cellules :



Interface générale de définition des animats et les différents menus contextuels pour les différentes zones (Entité, Stades, Attributs et Tâches)

Mobidyc étant destiné à la dynamique de peuplements, les agents sont structurés en entités et stades. Par exemple des stades œufs ou juvéniles de l'entité truite. Nous verrons plus loin que ces stades peuvent être connectés par les tâches "Métamorphose" et "Reproduction". Dans notre exemple, la structure en stades n'est pas utile mais elle existe de toute façon.

Dans Mobidyc, les animats sont des "stades" d'une "entité". Pour créer une entité 'Sugar' ayant un seul stade 'agent', il faut utiliser le premier menu contextuel (zone Entité) et sélectionner la première option 'Créer', vous obtiendrez la fenêtre de droite, avec la définition du nom de l'entité et de ses stades. N'oubliez pas de cliquer sur 'Ajouter' avant de valider votre entité. Vous obtiendrez la fenêtre de peuplement suivante:



Fenêtre de définition du peuplement après la création de l'agent Sugar

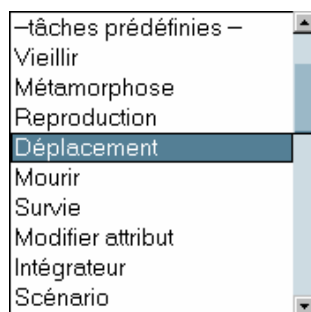
Par défaut tout animat possède les attributs "age", "nombre" et "localisation" ainsi que la tâche "vieillir" qui consiste tout simplement à augmenter l'âge en fonction du pas de temps de la simulation. Cliquer sur la tâche Vieillir, et "modifier" dans le menu contextuel. Vous obtiendrez la fenêtre suivante:



Interface de modification de la tâche Vieillir

Cette tâche est un simple compteur de temps. Vous pouvez modifier son nom et spécifier un autre attribut récepteur.

Pour faire bouger notre agent. Sélectionnez la tâche prédéfinie "Déplacement" dans la zone 'Sélectionner un nouvel attribut ou une nouvelle tâche', puis cliquer sur 'Ajouter'



La fenêtre de définition des déplacements s'affiche:

Interface du définition du déplacement d'un agent du peuplement: agent (Sugar)

Spécifier le rayon d'action (par exemple deux cellules), le fait de considérer ou non la cellule de départ comme une cible potentielle, un filtre de recherche d'un gradient ascendant du sucre, et un tri sur les ex-æquo.

Vérifier que la tâche Déplacement est ajoutée à la liste des tâches dans la fenêtre du peuplement. Vous pouvez visualiser les tâches et les attributs de votre agent. Vous pouvez aussi accéder et modifier ce qui a déjà été fait, ou ajouter de nouvelles tâches et attributs. La petite croix derrière chaque tâche signifie que la tâche sera activée. La décocher désactivera la tâche sans la supprimer (elle ne sera pas exécutée). Cette fonctionnalité est très utile pour tester le comportement de son agent.

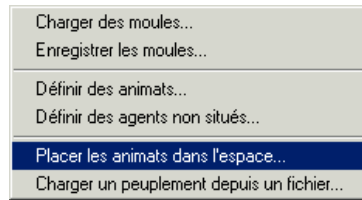
Remarque importante : Dans tout ce qui précède, nous avons dit que nous définissions des animats et des cellules. C'est un abus de langage. Nous avons défini les "moules" de ces agents, c'est-à-dire les structures à partir desquelles seront "clonées" les agents réels. Sauf exception (par exemple lorsque vous avez donné des valeurs de sucre initiales aux différentes cellules), vous ne travaillerez donc jamais directement sur les agents, mais sur leurs moules.

Lancer une première simulation

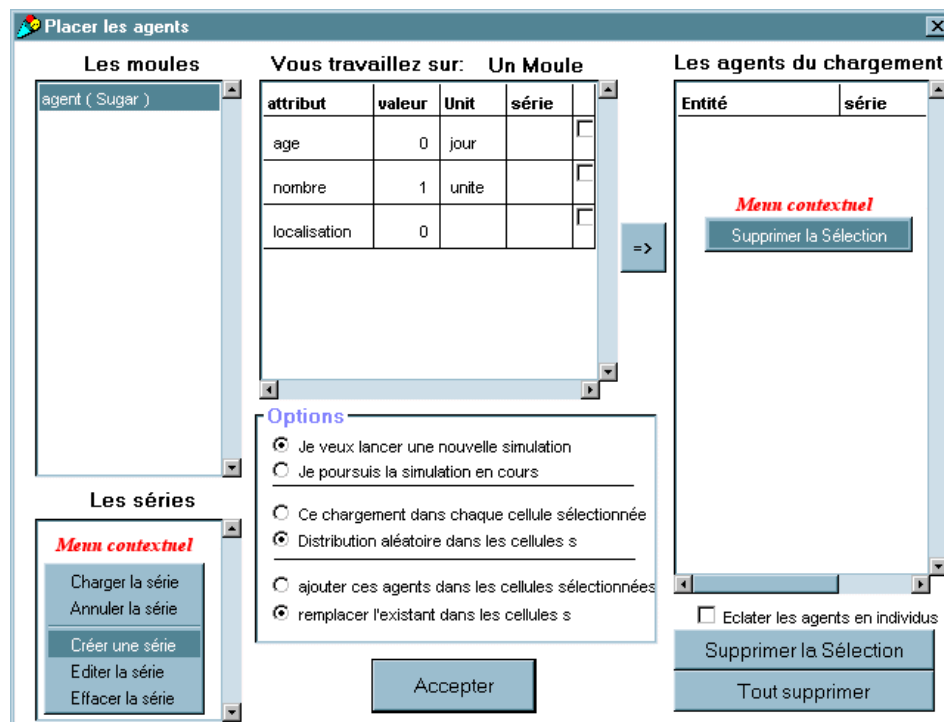
Vous avez maintenant tous les éléments d'un projet: un espace (par défaut, un espace d'une seule cellule est toujours créé), un moule d'animat (qui vieillit et qui se déplace), et un point de vue sur la quantité de sucre de la cellule. Vous êtes prêt à lancer des simulations. Sauvez votre projet par précaution.

Avant de lancer une simulation, il faut spécifier les conditions initiales. Celles des agents cellules sont celles que vous avez définies lors de la définition des cellules. Pour les animats, il faut sélectionner les agents de départ, et les placer sur la grille. Nous allons commencer par placer un seul agent sur une seule cellule.

Sélectionner sur la grille la cellule dans laquelle vous voulez mettre votre agent puis retrouver le menu Peuplement
→ Placer les animats :



La fenêtre de chargement s'affiche. Cette fenêtre est assez complexe. Consultez l'aide le cas échéant. Nous nous contenterons pour l'instant du chargement "manuel" par défaut : les animats disponibles doivent apparaître dans la liste de gauche. Sinon, c'est vraisemblablement que vous avez quitté la fenêtre de définition des animats par un "annuler". Revenez aux étapes précédentes et définissez votre animat. Sélectionner agent (Sugar), vous obtiendrez la fenêtre suivante:




Interface de chargement initial des animat

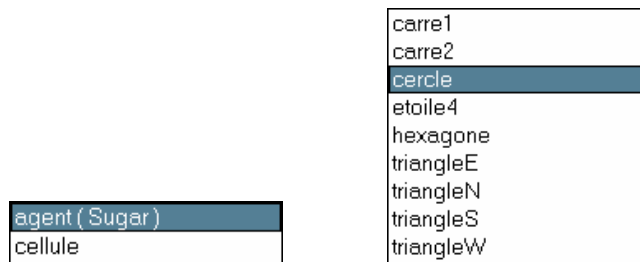
Le tableau central, permet de fixer la valeur initiale des différents attributs de l'animat sélectionné. Nous nous contenterons des valeurs par défaut présentes dans le moule. Cliquez sur la flèche pour copier cet agent dans la liste de droite, qui contient les "moules de chargement" à partir desquels MobidyC va créer les agents initiaux.

Remarque: si vous sélectionnez un moule (tableau de gauche), et que vous modifiez ses attributs (tableau central), vous modifiez les moules des animats, c'est-à-dire leur valeur par défaut. C'est parfois gênant, on peut souhaiter des valeurs initiales différentes des valeurs par défaut, tout en gardant ces valeurs par défaut pour la reproduction ou la métamorphose par exemple. Pour cela, sélectionner un "moule de chargement" (tableau de droite, après avoir cliqué sur la flèche) et modifier les attributs. Le label, au dessus du tableau central vous informe sur ce que vous êtes en train de modifier: le moule de l'animat ou un "moule de chargement". Si vous rappelez ultérieurement cette interface, MobidyC vous redonnera dans cette liste de droite les "moules de chargement" déjà définis.

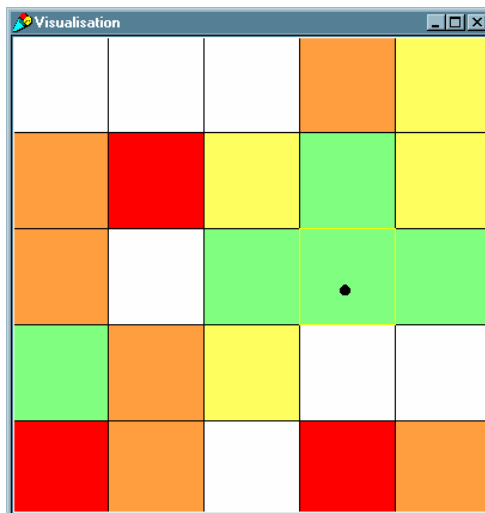
- Pour créer rapidement des individus tous identiques, donnez une valeur différente de 1 à l'attribut nombre, puis cochez l'option 'Eclater les agents en individus'. Si vous ne cochez pas cette option, il sera créé un seul agent représentant un groupe d'individus identiques
- Vous pouvez modifier les conditions initiales des valeurs des attributs avant de sélectionner chaque agent dans le chargement.
- Vous pouvez sélectionner plusieurs agents et les distribuer aléatoirement dans les cellules sélectionnées. Pour cela il faut cocher dans la zone des options le choix 'Distribution aléatoire dans les cellules'.
- Nous verrons plus loin des chargements plus complexes chargements initiaux plus complexes.

Mobidyc revient à l'interface principale. En cliquant sur le bouton "Visualisation" , rien ne s'affiche. Normal, nous n'avons pas encore défini de point de vue pour notre animat. Rappel de l'interface de l'Observatoire et ajouter une vue sur cet agent (globalement, sans conditions sur ces attributs pour l'instant). Par exemple un rond noir.

Pour cela, sélectionnez l'agent sugar dans la zone "vue", puis le cercle dans la liste des formes et la couleur noir:



Enregistrer la vue et valider l'observatoire. Vous obtiendrez une carte comme celle-ci.




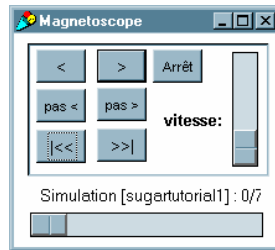
Visualisation de la grille après chargement de l'agent

Remarque : vous pouvez changer la taille relative du point et choisir de ne pas visualiser les bordures des cellules via le menu Option -> préférences.

On garde le pas de simulation de un jour (unité de temps par défaut du simulateur) dont on notera qu'elle est parfaitement arbitraire sur notre exemple. Et on fixe la durée à une semaine par exemple. Cliquer "lancer la simulation" : notre agent doit se précipiter vers les cellules les plus sucrées mais il peut aussi être piégé sur des minima locaux, surtout si son rayon d'action n'est que de 1 avec une grille à 4 voisins. Pour relancer la simulation, presser à nouveau sur "lancer la simulation". Pour poursuivre, presser "continuer la simulation" qui vient d'apparaître.

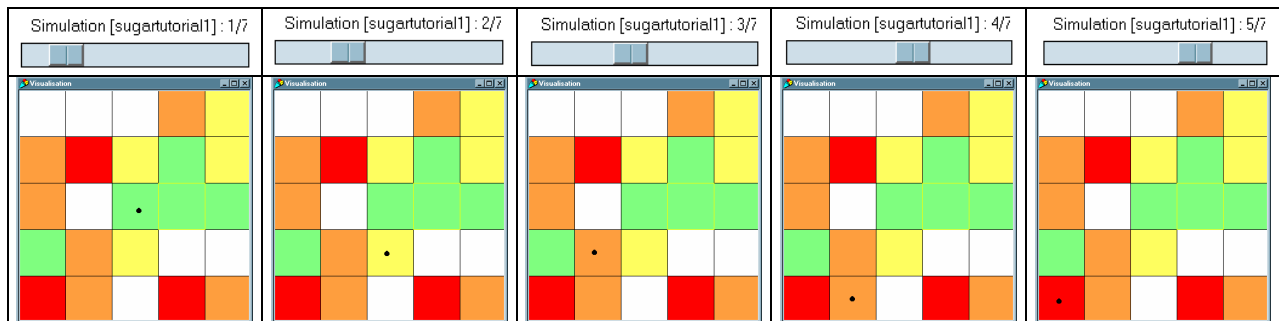
La visualiser avec le magnétoscope

Par défaut, en mode interactif, tout est sauvegardé à chaque pas de temps, et nous pouvons rejouer la simulation à l'aide du magnétoscope . ceci donnera:



Les aspects stochastiques des modèles individus centrés:

Nous pouvons constater sur cet exemple très simple, qu'en réduisant le rayon d'action à 1 cellule, l'agent suivra un gradient ascendant du sucre, il choisira au premier déplacement une cellule verte. Selon le choix de la première cellule le résultat de la simulation va être différent. Par exemple dans le cas où l'agent choisira la cellule verte à gauche le chemin choisi sera le suivant:



Jouer une simulation à l'aide du magnétoscope

L'agent mange le sucre

Notre animat ayant trouvé ce qu'il cherche, il faut le faire manger. Manger se traduit ici par "ajouter la valeur d'une variable (le sucre de ma cellule) à un de mes attributs (mon sucre à moi)". Pour cela commençons par créer l'attribut récepteur, le sucre de l'agent. Retourner dans la fenêtre de définition des animats pour y ajouter un attribut standard :

ajouter un attribut standard "sucre" à l'animat "agent (Sugar)"

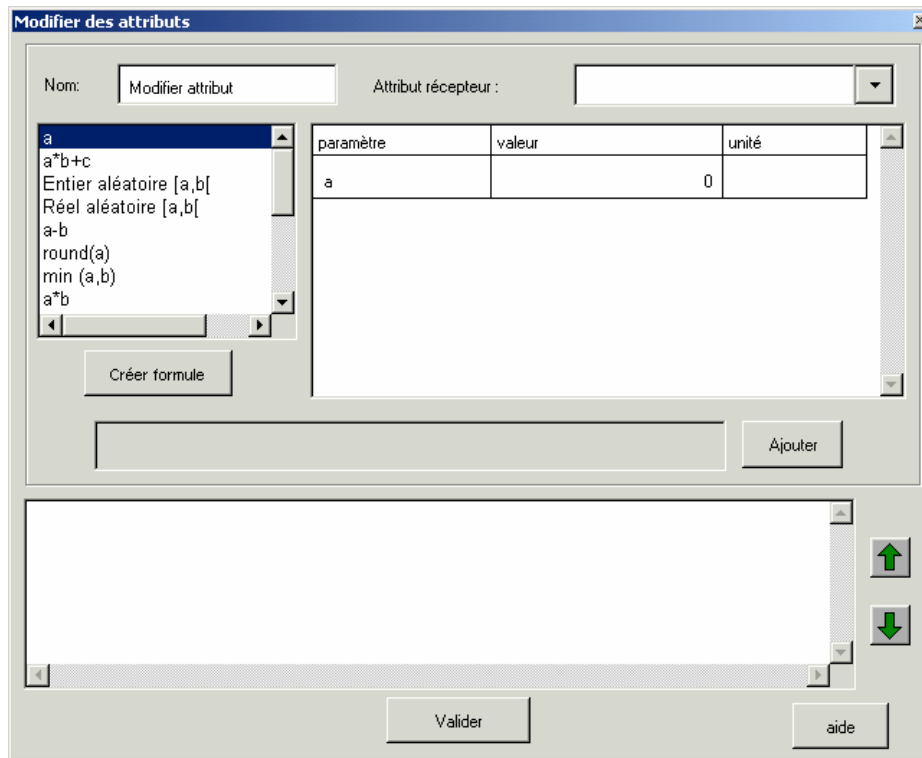
La tâche manger peut se traduire par l'expression mathématique suivante:

le sucre de Sugar = (le sucre de Sugar) + (le sucre de la cellule)

Mais vis-à-vis de l'agent l'expression peut s'écrire:

mon sucre = (mon sucre) + (le sucre de ma cellule)

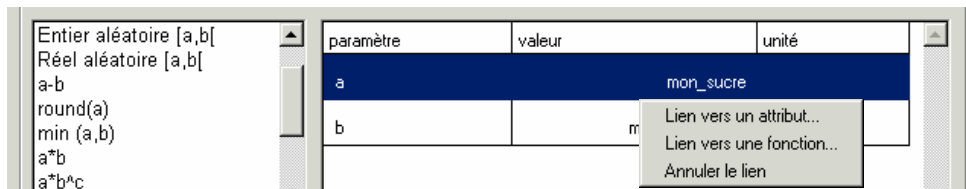
La définition de ce type d'expressions impliquant des attributs d'agent se fait à l'aide de la tâche "ModifierAttribut". Ajouter cette tâche à votre agent. La fenêtre suivante apparaît:



Fenêtre de la tâche "Modifier attribut"

Commencer par donner un nom à la tâche, par exemple "Manger". Dans la zone "Attribut récepteur", vous avez accès à la liste des attributs de l'agent (Sugar) "mon" et aux attributs de tous les agents accessibles par notre animat, et en particulier sa cellule, ainsi que l'agent non situé "simulateur", toujours présent, et qui fournit aux agents des informations sur la simulation : pas de temps, durée, temps écoulé... Choisissez "mon_sucré":

Sélectionnez ensuite la formule, ici a+b (il est possible de créer sa propre formule avec le bouton correspondant). Le paramétrage de la formule s'affiche dans la boîte centrale. Vous pouvez soit entrer une valeur, avec son unité, soit et c'est ce qu'il faut faire, sélectionner la paramètre et utiliser le menu contextuel pour lier sa valeur à celle de l'attribut concerné, par exemple "mon_sucré" pour a et maCellule_sucré pour b.



Définir une dépendance sur le paramètre 'a' (de la formule a+b) à l'aide du menu contextuel

La formule est maintenant complète, il ne faut pas oublier de l'ajouter (en cliquant sur le bouton 'Ajouter') pour qu'elle s'affiche dans liste des opérations mathématiques associées à la tâche manger.

Modifier des attributs

Nom: Attribut récepteur :

Entier aléatoire [a,b]
 Réel aléatoire [a,b]
 a-b
 round(a)
 min (a,b)
 a*b
 a*b^c
a+b

Créer formule

paramètre	valeur	unité
a	mon_sucré	
b	maCellule_sucré	

Valider et vérifier qu'elle apparaît bien dans la liste des tâches de notre agent :

Tâches	
Vieillir	<input checked="" type="checkbox"/>
Déplacement	<input checked="" type="checkbox"/>
Manger	<input checked="" type="checkbox"/>

Liste des tâches de l'agent Sugar

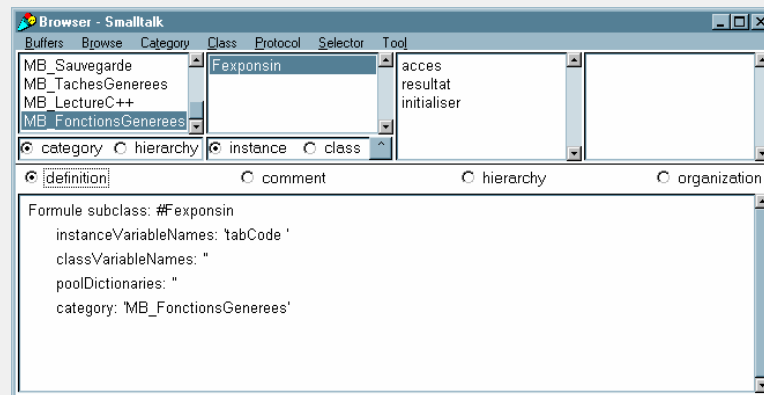
Valider le peuplement. Par sécurité sauvez votre projet de temps en temps.

Remarque : Le bouton créer une formule fait appel à un interpréteur syntaxique de formules mathématiques. Nous pouvons créer de nouvelles formules paramétrables en respectant une certaine syntaxe (essentiellement séparer les opérateurs et les paramètres par des espaces blancs). Par exemple pour ajouter la formule suivante:

$$(a+b)e^{\sin(c)}$$

Il suffit de cliquer sur le bouton 'Créer formule', l'interface suivante apparaît (son remplissage donne le résultat suivant).


Mobidyc créera automatiquement une nouvelle classe 'Fexponsin' de cette nouvelle formule. (Pour les développeurs, cette classe sera rangée dans la catégorie 'MB_FonctionsGenerees' de l'environnement Smalltalk).

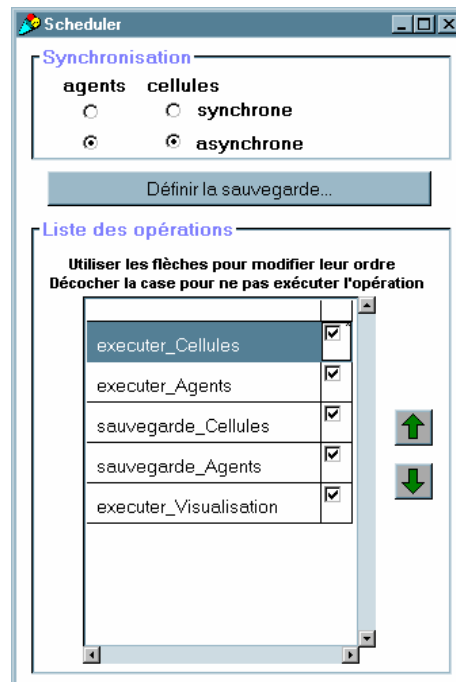


le 'Browser' de Smalltalk montrant la nouvelle classe générée par Mobidyc

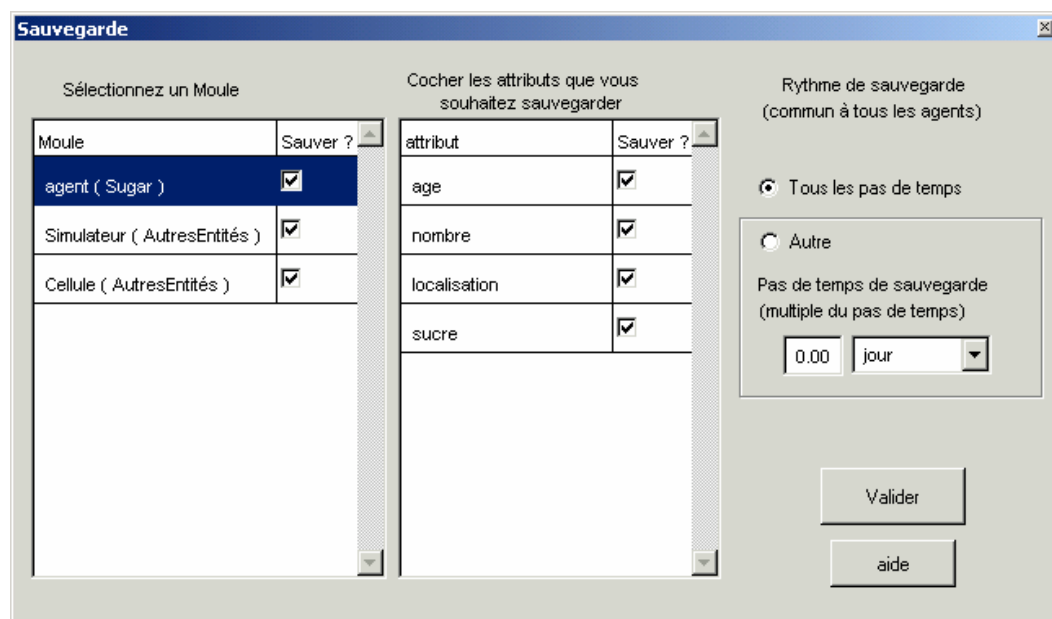
Le 'scheduler', ordonnanceur de la simulation

Avant de relancer une simulation, jetons un oeil sur le "Scheduler", véritable chef d'orchestre de la simulation. Il permet de spécifier le mode de synchronisation (synchrone ou asynchrone) des agents et des cellules, ainsi que modifier l'ordre d'exécution et de sauvegarde des cellules et des animats et des agents non situés. Il permet également d'affiner les options de sauvegarde des résultats.

Sélectionner "Scheduler" , dans le menu "Outils" de l'interface principale :



Pour l'instant nous allons laisser les options par défaut, mais nous allons vérifier que tous les attributs sont sauvegardés. Cliquer sur 'Définir la sauvegarde', vous obtenez l'interface suivante après avoir sélectionné l'agent (Sugar).




Interface du choix de sauvegarde

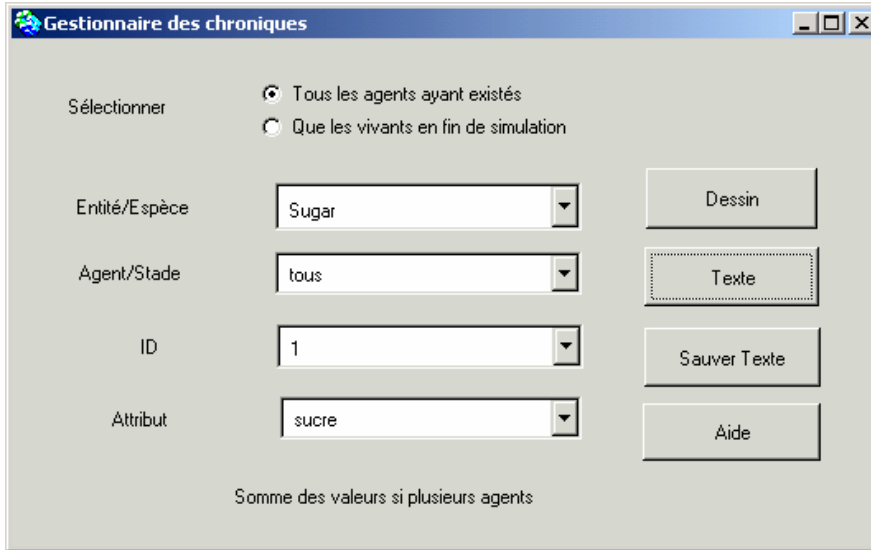
Les attributs s'affichent et tous doivent être cochés par défaut.

Vous pouvez relancer directement une simulation, Mobidyc prenant automatiquement en charge le fait que vous avez ajouté un attribut à votre moule. Il est donc inutile (en principe !) de repasser par la phase de placement de votre agent dans l'espace. Par précaution, toujours sauver votre projet après chaque changement important.

Visualiser une chronique

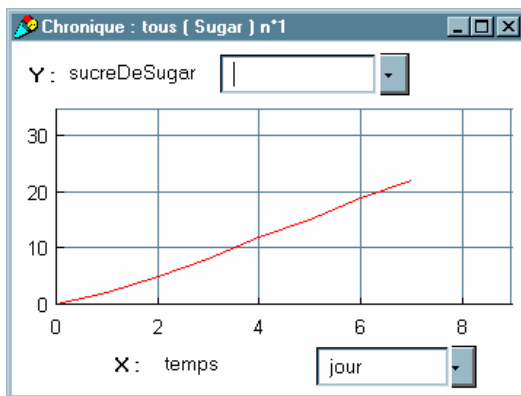
Nous souhaitons maintenant contrôler que notre animat a bien mangé. Pour cela nous allons visualiser la chronique de son stocke de sucre dans le temps :

Cliquez sur le bouton "Chronique"  (ou bien menu Outils → Chronique) pour appeler le gestionnaire des chroniques:

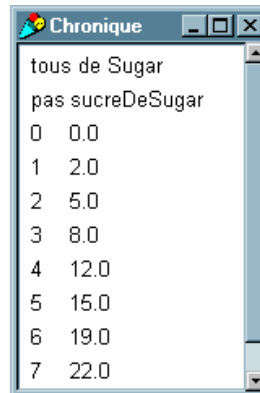


Gestionnaire des chroniques

Nous pouvons visualiser la chronique en cliquant sur "Dessin" ou bien le "Texte":



Chronique de sucreDeSugar sur 7 jours



tous de Sugar
pas sucreDeSugar
0 0.0
1 2.0
2 5.0
3 8.0
4 12.0
5 15.0
6 19.0
7 22.0

Valeurs de la chronique

Le gestionnaire des chroniques permet de suivre un seul individu (agent), nous pouvons sélectionner son identifiant:

ID

Vous pouvez sélectionner une cellule ou un groupe de cellules, la chronique ne représentera alors que ce que c'est passé dans la sélection.

Mobidyc permet de gérer les unités du temps (axe X) et les unités courantes.

Y : sucreDeSugar unite

X : temps

- seconde
- minute
- heure
- jour
- semaine
- mois
- annee

- nombre
- poids
- temps
- temperature
- longueur
- sexe
- booleen

"Dialogues" entre agents

L'agent "tond" le sucre de la cellule

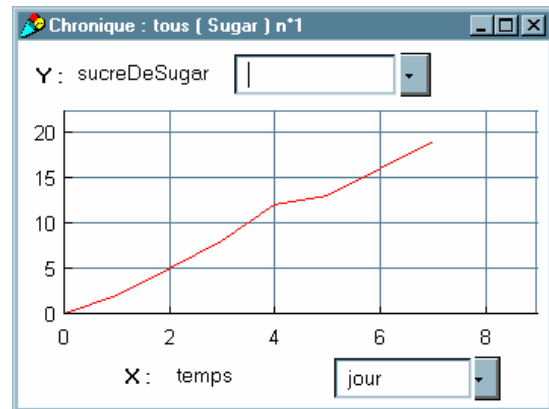
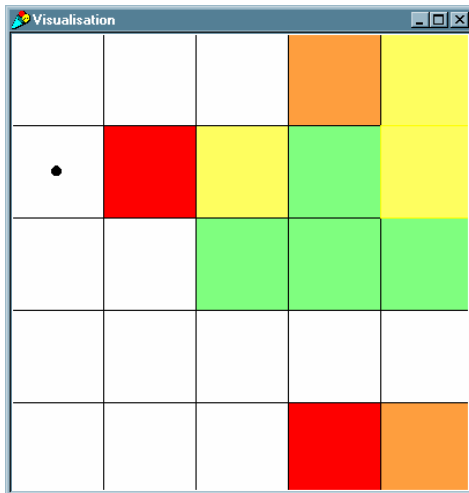
Pour ne pas vexer Lavoisier, la cellule doit être informée qu'un animat vient de consommer son sucre et mettre sa valeur à zéro. Mobidyc, dédié à des agents réactifs, ne gère pas les messages entre agents. Par contre (et cela viole un peu la notion d'encapsulation) tout attribut (mais seuls les attributs) d'un agent est visible ET MODIFIABLE par un autre agent, ou par l'utilisateur qui souhaite l'observer. C'est cette propriété que nous allons utiliser : l'agent va directement modifier l'attribut "sucre" de sa cellule.

Pour cela retournez dans le menu "Peuplement → Définir les agents", et sélectionnez la tâche "Manger" pour la modifier (menu contextuel). L'interface de la tâche ModifierAttribut se réaffiche telle que vous l'aviez laissée. Vous allez ajouter une nouvelle expression : `maCellule_sucre = 0`. Pour cela, sélectionner le bon attribut récepteur puis la formule "a" en laissant la valeur à 0. Ajoutez l'expression à celle qui existe déjà. Vous devez obtenir :

```
mon_sucre := mon_sucre+maCellule_sucre
maCellule_sucre := 0
```

Les nouvelles expressions de la tâche "Manger"

Donnez un nom plus explicite à la tâche, "Tondre" par exemple, puis validez et relancez la simulation. L'agent transmute maintenant en blanc toutes les cellules qu'il touche... sans oublier de stocker son sucre, ce que vous pouvez vérifier comme précédemment.



La carte de distribution du sucre à la fin de la simulation Chronique de la quantité du sucre de l'agent de Sugar

L'agent respire et meurt

Fini de rire, à courir partout notre agent se fatigue et consomme du sucre (il respire). Et si son stocke s'annule il meurt. En ce qui concerne la respiration, rien de plus facile, nous allons réutiliser "Modifier attribut" avec une expression du type $\text{mon_sucre} = \text{mon_sucre} - \text{respiration}$. Reste à définir "respiration". Une solution est de garder une valeur numérique (par exemple 2 sucres par unité de temps) dans la formule mathématique. Dans ce cas, tous les agents auront le même taux de respiration. L'alternative est d'ajouter un nouvel attribut à notre agent, par exemple "respiration", et de lier la valeur du paramètre de la formule à cet attribut. Chaque agent pourra alors contrôler sa propre respiration. C'est ce que vous allez faire. Revenez au menu "Peuplement → Définir les agents → ajouter un attribut standard" et définissez l'attribut "respiration", avec 2 comme valeur initiale.

Complétez ensuite la tâche ModifierAttribut comme ci-dessous.

Modifier des attributs

Nom: Attribut récepteur :

paramètre	valeur	unité
a	mon_sucre	
b	mon_respiration	

Créer formule

Définition de la tâche "Respiration" en utilisant la tâche prédéfinie "Modifier attribut"

Valider votre tâche et vérifier qu'elle s'est ajoutée à la liste des tâches de l'agent Sugar (4 tâches : Vieillir, Déplacement, Tondre et Respiration).

Remarque : Pour éviter de jongler entre les interfaces, vous pouvez saisir manuellement le nom d'un attribut non encore existant dans la tâche ModifierAttribut. Mais attention de ne pas oublier de le définir ensuite, avec la même orthographe. Il est donc très fortement conseillé de toujours définir les attributs AVANT de les utiliser, et de n'utiliser la saisie manuelle des attributs qu'exceptionnellement.

Reste maintenant à définir la mortalité. La mortalité est une tâche prédéfinie. L'agent meurt si certaines conditions, portant sur lui-même, sur sa cellule, ou sur un agent non situé sont remplies. Sélectionner et ajouter la tâche "Mourir". Tâtonnez un peu pour voir comment fonctionne la boîte des conditions, et définissez la condition "mon sucreDeSugar < 0" pour le faire mourir, histoire de laisser une petite chance à l'agent exactement égal à zéro. Accepter la tâche et validez votre peuplement.

Mourir

agent (Sugar) Nom de la tâche:

Si

	Variable	Opérateur	Valeur Seuil	Unité
▶	mon_sucre	<	0	

☒ ET ☐ OU

Alors C'est pas de jeu, je suis mort!

Interface pour la définition de la tâche "Mourir"

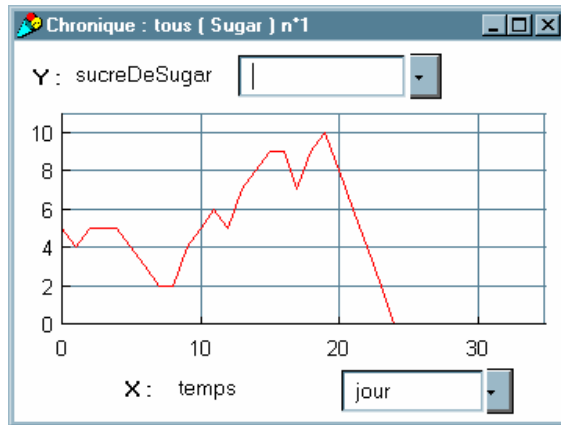
Vous pouvez relancer une simulation mais retournez plutôt dans la fenêtre de placement des agents. Effacez le précédent chargement, et placez un nouvel agent en vérifiant au passage la valeur des attributs initiaux. Vous pouvez gentiment lui donner une réserve de sucre au départ : mettre une valeur, par exemple 5, dans la case correspondante.

attribut	valeur	Unit	série	
nombre	1	unite		<input type="checkbox"/>
sucreDeSuga	5			<input type="checkbox"/>
localisation	0			<input type="checkbox"/>
age	0	jour		<input type="checkbox"/>
resp	2			<input type="checkbox"/>

Liste des attributs de l'agent Sugar dans la fenêtre du chargement d'un peuplement, la valeur de sucreDeSugar a été initialisée à 5.

Lancer la simulation. La poursuivre le cas échéant jusqu'à la mort de l'agent : il doit se volatiliser, mais la simulation se poursuit jusqu'à la fin (durée de la simulation). S'il ne meurt pas, vérifier pourquoi : sélectionner le et éditer son stocke de sucre. Retourner si nécessaire contrôler et modifier les différentes tâches de l'agent dans la fenêtre générale du peuplement.

Et s'il meurt, comment le retrouve-t-on ? En sélectionnant l'ensemble de la grille, l'ensemble des animats ayant existé au cours de la simulation est accessible via le gestionnaire des résultats. Visualiser ou éditer le stocke de sucre de l'animat.



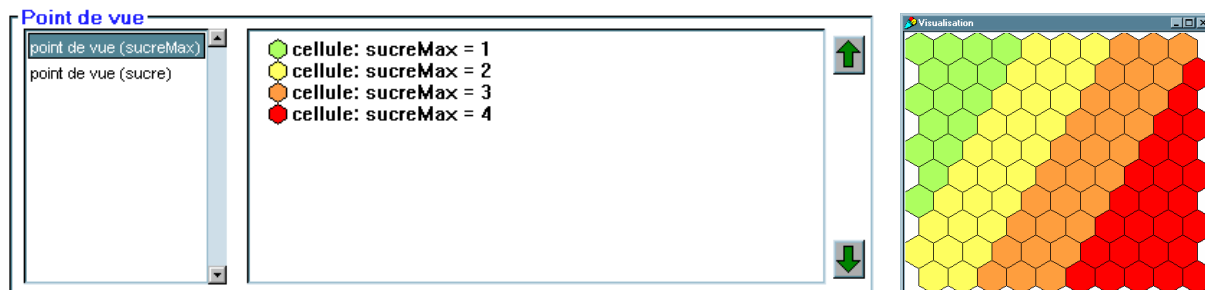
Chronique du stock de sucre de l'agent Sugar, qui est mort le 25^{ème} jour de la simulation

La cellule restaure son sucre

Retournons à nos cellules qui ne souhaitent maintenant qu'une chose, restaurer leur sucre. C'est une tâche un peu plus complexe puisqu'il ne suffit pas d'ajouter une dose de sucre à chaque pas de temps. Il faut s'arrêter lorsque l'on a atteint la capacité maximale de la cellule. Nous allons donc avoir besoins de deux paramètres pour les cellules : une capacité maximum, et le terme de croissance par unité de temps. Fidèles à la logique de Mobidyc, il faut immédiatement penser à en faire deux nouveaux attributs. Pour cela, sélectionner toutes les cellules (ou n'en sélectionner aucune en cliquant à l'extérieur de la grille), rappeler l'interface de définition des cellules et créer un attribut "croit" par exemple, fixé à 1 pour toutes les cellules (gain de 1 sucre par unité de temps). Pour la capacité maximum (ex. "sucreMax"), plutôt que de paresseusement profiter de l'outil de création aléatoire, nous allons prendre le temps de créer un gradient pour structurer un peu l'espace, ce sera plus joli et démonstratif.

Pour cela, partons d'une nouvelle grille plus grande (10 × 10, par exemple). Revenez au menu Espace → Créer une grille, et pour changer, essayons les cellules hexagonales. Nous allons maintenant "peindre" manuellement les cellules. Il faut procéder par étape : sélectionner toutes les cellules qui correspondent à une valeur, puis rappeler l'interface de définition des cellules, vérifier que la case "sélection à la souris" est bien activée, et donner la bonne valeur à l'attribut "sucreMax" (menu contextuel "Modifier" après avoir sélectionné l'attribut). Répéter l'opération pour chaque valeur, par exemple 1, 2, 3 et 4.

Vous remarquerez vite que pour vous en sortir, il faut définir un nouveau point de vue sur les cellules, permettant de visualiser le nouvel attribut "sucreMax". Vous obtiendrez par exemple l'observatoire suivant et la grille suivante après validation :



Nouveau point de vue sur les valeurs de "sucreMax" et visualisation de la grille (10 × 10)

N'OUBLIEZ PAS D'ENREGISTRER DE TEMPS EN TEMPS VOTRE PROJET.

Pour créer la tâche "Pousser", revenez à la fenêtre de définition des cellules. Nous allons encore utiliser la tâche "Modifier attribut" dans laquelle nous enchaîneront deux instructions.

```
1. mon_sucres = mon_sucres + mon_croit
```

2. $\text{mon_sucre} = \text{Min}(\text{mon_sucre}, \text{mon_sucreMax})$

Il faut donc utiliser la formule $a+b$ et la formule $\min(a,b)$, vous obtiendrez la définition des opérations en bas de la fenêtre de la tâche "Modifier attribut" (n'oubliez pas de donner un nouvel nom 'Pousser' à la tâche créée).

Nom: Attribut récepteur :

```
mon_sucre := mon_sucre+mon_croit  
mon_sucre := min(mon_sucre,mon_sucreMax)
```

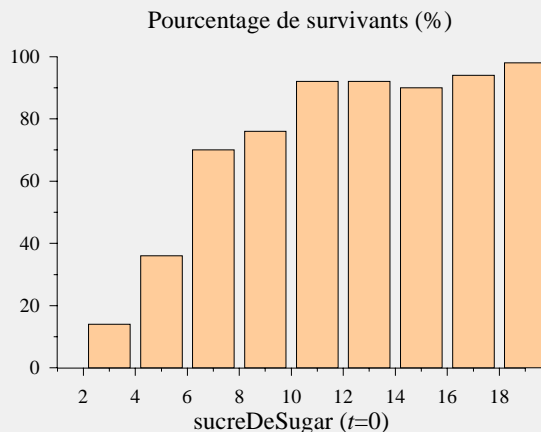
Création de la nouvelle tâche "Pousser" pour restaurer le sucre dans les cellules

Valider la tâche (qui apparaît dans la liste des tâches des cellules) puis la fenêtre de définition des cellules. Placer un agent dans la zone la moins riche. Avant de lancer une simulation il faut revenir à notre ancien point de vue de l'observatoire (vue sur le sucre des cellules). Vérifier que notre agent a une valeur initiale de sucre suffisante pour pouvoir franchir la zone pauvre en sucre (5 par exemple). Vérifiez également que le sucre initial de toutes les cellules est bien 0 (Grille blanche au départ). Le cas échéant, revenez à $t=0$ avec le menu "retour à t_0 ". Nous verrons plus loin comment donner la valeur sucreMax comme valeur initiale aux cellules.

Faites plusieurs simulations sur une période de 30 jours (1 mois). Vous remarquerez que les trajectoires de l'agent sont différentes et que dans certaines simulations l'agent est piégé dans la zone verte (la plus pauvre) et qu'il finira par mourir avant la fin de la simulation. Cela dépend bien sûr du rayon d'action que vous lui avez donné (ici 1) et de son sucre initial.

Exemple d'étude de sensibilité:

A rayon d'action fixé à 1, c'est la valeur initiale de 'sucre' qui déterminera le devenir de notre agent. Il peut être intéressant de faire une petite analyse de sensibilité en faisant varier la valeur initiale de 'sucre' de 3 à 19 tous les 2 unités. Pour chaque valeur 50 simulations ont été réalisées. Le pourcentage de survivants en fonction de 'sucreDeSugar' est donné dans la figure suivante:



Variation du pourcentage de survivants en fonction de la valeur initiale de sucreDeSugar

Les agents non situés

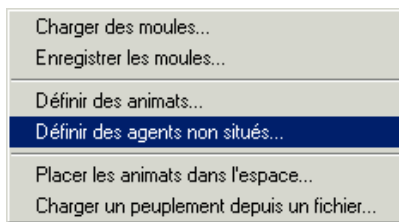
Définition

Nous avons vu deux types d'agents. La troisième famille d'agents sous Mobidyc est celle des agents non situés. Ce sont des agents qui seront souvent chargés de tâches d'ordre général, comme dérouler les scénarios environnementaux, faire des calculs auxquels les autres agents auront accès, collecter les résultats importants de la simulation, ou encore piloter la mémorisation des expériences simulatoires en mode "Batch". Chaque agent de ce type est unique (une seule copie à partir du moule), et est donc toujours accessible, par son nom, par les autres agents. Tous les agents peuvent donc y faire référence à tout moment, et ils sont présents du début à la fin de la simulation.

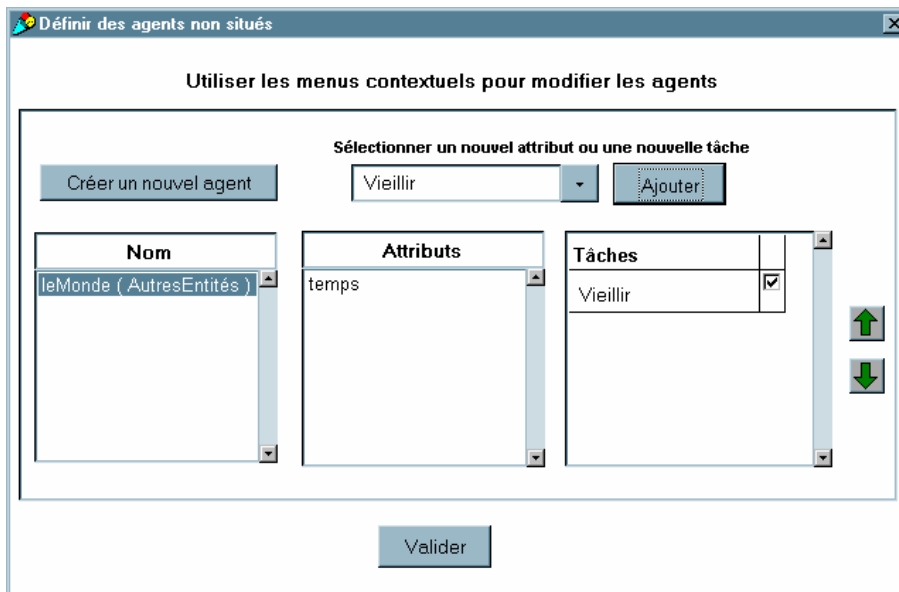
Un agent non situé toujours présent et que vous avez déjà vu apparaître est l'agent "Simulateur", qui rend accessible les variables de la simulation : pas de temps, durée, et temps écoulé. Par esprit didactique, nous allons créer de toute pièce un agent (leMonde) qui va tenir une partie de ce rôle : la référence au temps écoulé. Car nous avons besoin de cette référence au temps pour créer une tâche qui va résoudre notre problème de tout à l'heure : initialiser le sucre des cellules à la bonne valeur.

Création d'un agent non situé

Pour définir un agent non situé il faut appeler le menu Peuplement → Définir les agents non situés:



L'interface de définition des agents non situés apparaît. Elle est fort logiquement semblable aux interfaces de définition des autres agents. Créer l'agent. Par défaut, un agent non situé n'a aucun attribut et aucune tâche. Donnez lui un attribut "temps" et la tâche "vieillir", que vous ferez pointer sur "temps" (par défaut, vieillir propose, et créé automatiquement si nécessaire, l'attribut "age").



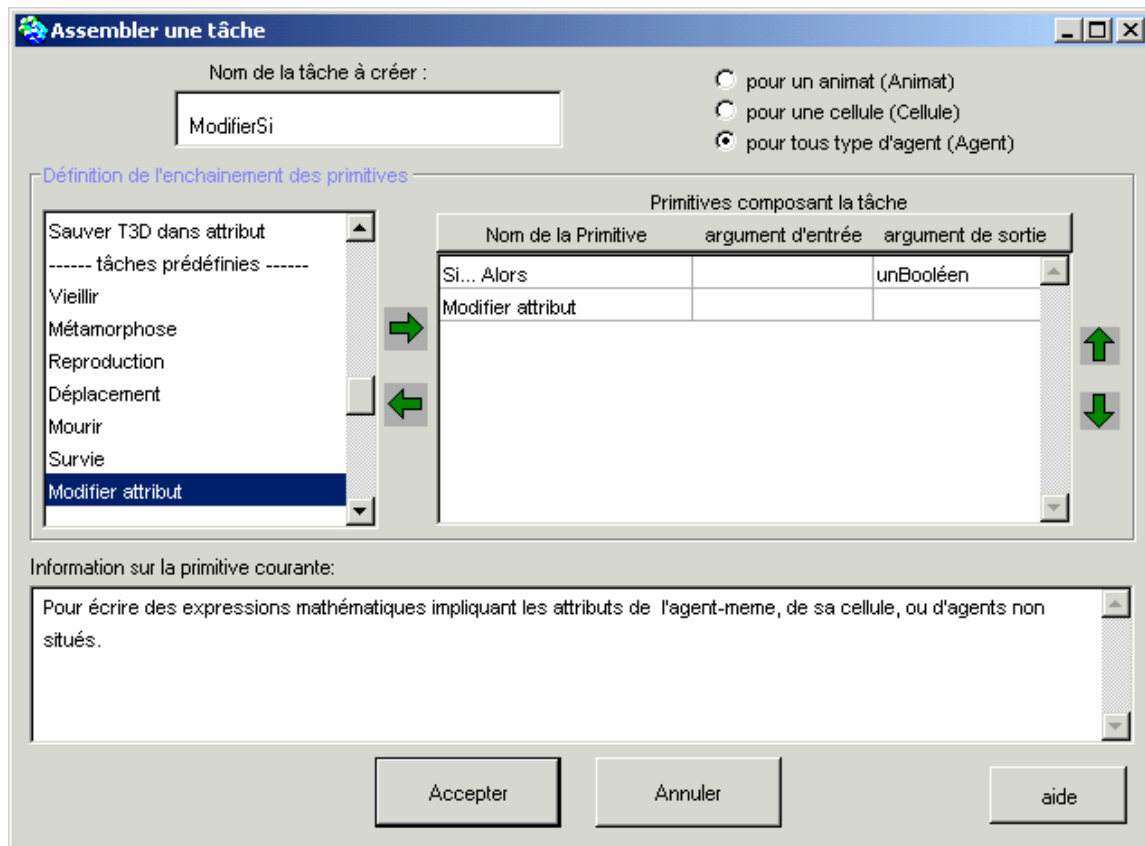
Fenêtre de définition de l'agent non situé 'leMonde'

Création d'une tâche utilisateur

Nous allons maintenant résoudre la question du sucre initial des cellules, en composant une tâche spécifique qui ne s'exécutera qu'au début du premier pas de temps. Car il est évidemment hors de question de "repeindre" le sucre à la main comme nous l'avons fait pour sucreMax. Mais nous verrons plus loin qu'il existe une solution encore plus rapide, qui consiste à passer par le fichier .esp de description des cellules. Il suffit d'éditer ce fichier sous un tableur (excel par exemple), de copier la série de valeurs correspondant à celles de sucreMax, et de recopier cette série dans la colonne sucre. Et le tour est joué.

Mais il est souvent utile de pouvoir définir une tâche d'initialisation et nous allons le faire. Cette tâche va tout simplement associer la primitive "Si...Alors" et la tâche d'initialisation à exécuter. Dans notre cas, ce sera une fois encore "ModifierAttribut" avec l'expression "mon_sucres = mon_sucresMax".

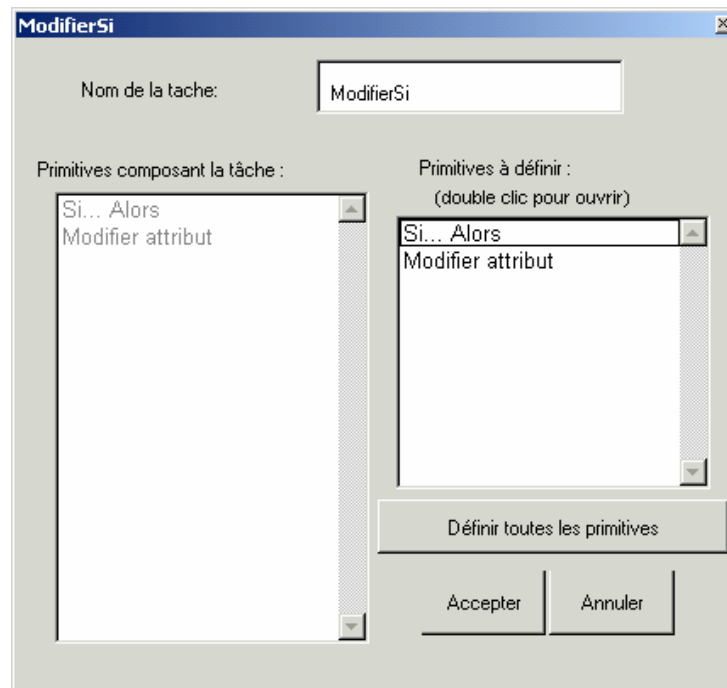
Pour créer cet assemblage, activez le menu 'Tâches → Assemblage d'une tâche'. L'interface d'assemblage des primitives apparaît. Consultez l'aide en ligne pour plus de détails. Commencez par donner un nom à votre tâche, par exemple "ModifierSi", qui explicite bien ce qu'elle fait. Précisez ensuite à qui vous destinez cette tâche. Ce dernier point permet à Mobidyc de trier les primitives disponibles : une cellule ne peut ni se déplacer ni se reproduire par exemple. Mais un "ModifierSi" n'a rien de spécifique, et vous pouvez donc indiquer que cette tâche est exécutable par tout type d'agent. La liste de gauche vous donne la liste des primitives réparties par grands groupes. Sélectionner "Si...Alors" dans le groupe 'Contrôle', puis la tâche "ModifierAttribut" dans le groupe "Tâches prédéfinies". Pour insérer ou retirer une primitive de l'assemblage utiliser les flèches horizontales. Les flèches verticales (à droite) permettent d'ordonner les différentes primitives.



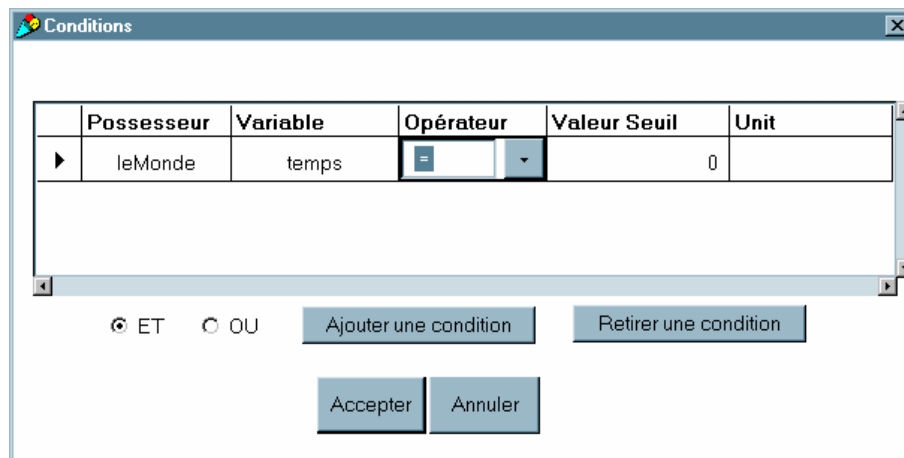
Fenêtre de définition d'une tâche par assemblage de primitives

Accepter la nouvelle tâche. Mobidyc crée le code de cette nouvelle tâche. Si une tâche avec ce nom existe déjà vous êtes avertis, et Mobidyc vous propose, le cas échéant, de retirer l'ancienne tâche des agents qui pourraient la posséder. Ceci pour garder la compatibilité avec la nouvelle tâche que vous venez de créer.

Pour les développeurs, précisons que le code de cette nouvelle tâche est rangé dans la parcelle "TachesUtilisateur" (qui se trouve dans le répertoire image avec ses deux fichiers, (binaire et source), .pcl et .pst), et plus précisément dans la catégorie "MB_TachesGenerees" de Smalltalk. Cette tâche obtient donc le même statut que les tâches prédéfinies, et on l'ajoute aux agents exactement de la même manière que ces dernières. Il nous faut donc maintenant retourner au menu de définition des cellules (Espace → Définir les cellules), et leur ajouter la tâche "ModifierSi" qui se trouve dans le groupe 'tâches générées'. L'interface générique de définition des tâches assemblées apparaît. Elle permet d'accéder aux interfaces spécifiques de chacune des primitives.



Cliquer sur le bouton 'définir toutes les primitives' pour une définition en boucle de toutes les primitives ou bien double clic sur le nom de la primitive. Dans l'interface de Si...Alors, ajouter une condition correspondant au premier temps de la simulation à l'aide de l'agent non situé 'leMonde', c'est-à-dire 0. Vous devriez obtenir le résultat suivant:



Interface de conditions sur le temps de l'agent 'leMonde'

Remarque importante : Quel est le premier pas de temps, 0 ou 1 ? Cela dépend de l'ordre d'appel des agents. Par défaut, l'ordre d'appel est le suivant :

- l'agent non situé 'Simulateur', toujours présent.
- les agents cellules
- les animats
- les agents non situés, dans l'ordre de leur création.

Vous pouvez modifier cet ordre (sauf 'Simulateur' toujours premier, qui d'ailleurs n'apparaît pas dans le scheduler) avec l'ordonnanceur (scheduler). Mais par défaut donc, l'agent 'LeMonde' sera exécuté **après** les cellules, et nous avons initialisé son attribut "temps" à 0. Au premier pas de temps, lorsque les cellules seront activées, 'leMonde_temps' sera donc toujours 0. Si par contre vous faites passer 'leMonde' en premier ou si au lieu de 'leMonde_temps' vous utilisez 'Simulateur_iterationCourante', alors il faudra tester la condition "leMonde_iterationCourante = 1"

Accepter cette condition. Dans l'interface 'Modifier attribut', construisez l'expression comme ci-dessous :

paramètre	valeur	unité	dépend.
a	0.0		d

Validez votre travail. Donner un nom plus explicite à votre tâche, par exemple 'InitCellule', qui doit apparaître dans la liste des tâches des cellules.

Lancer la simulation pour contrôler que tout va bien. Dès le premier pas de temps le gradient doit apparaître conformément à la tâche d'initialisation et l'agent se précipite vers les cellules les plus riches en laissant derrière lui une traînée qui se comble progressivement.

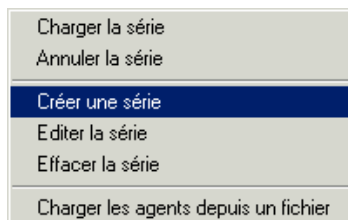
Remarque : Cette tâche d'initialisation est coûteuse puisque la condition sera testée à chaque pas de temps. Il est donc préférable d'initialiser les cellules en éditant le fichier .esp et en modifiant directement la colonne sucreMax. Si le besoin s'en fait vraiment sentir, une évolution de Mobidyc pourrait être de donner aux agents un dictionnaire spécifique pour ces tâches d'initialisation.

Faire jouer plusieurs agents : créer des agents différents à partir du même moule

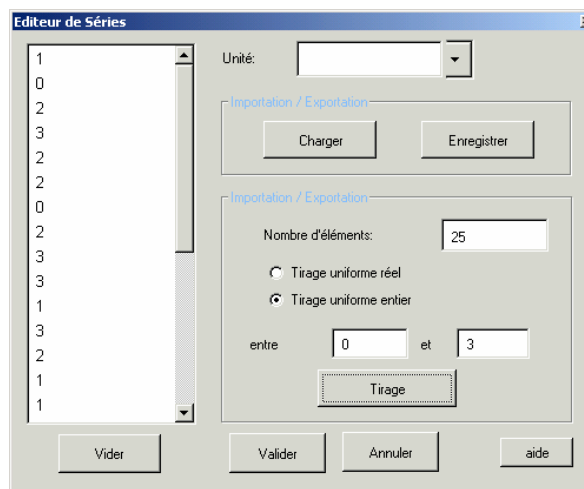
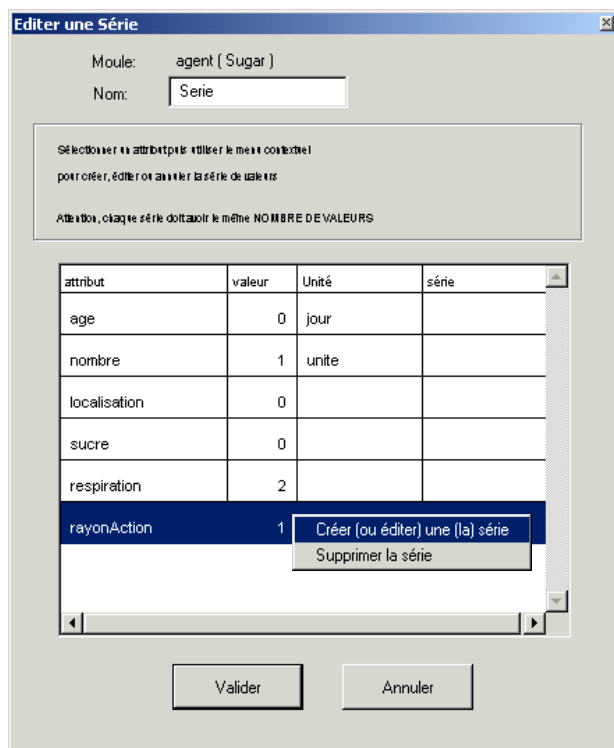
Nous allons maintenant charger plusieurs agents, qui conformément à l'exemple Sugar auront chacun un patrimoine génétique différent pour leur rayon d'action, leur sucre initial et leur respiration. En ce qui concerne le rayon d'action, nous l'avons fixé à 1 dans la tâche de l'agent. Ce rayon est donc pour l'instant commun à tous les agents. Pour permettre l'individualisation de ce paramètre, il faut lui donner un lien vers un attribut de l'agent, que nous appellerons "rayonAction" par exemple. Eviter par prudence de mettre des blancs dans les noms d'attributs. Vous êtes maintenant familier avec l'édition de liens : il est préférable de d'abord créer le nouvel attribut, puis de mettre à jour le lien dans l'interface de déplacement. (comme toujours, menu contextuel dans le champ de saisie du paramètre).

Remarque importante sur les liens : Vous avez le choix entre 'lien vers un attribut' et 'lien vers une fonction'. Vous connaissez le premier. Il va remplacer la valeur du paramètre par celle de l'attribut. Et cela nous suffit ici. L'autre option permet de combiner (addition ou multiplication) la valeur du paramètre avec celle de l'attribut, à travers une fonction mathématique. Donc quelque chose comme $a + f(\text{attribut})$ ou $a * f(\text{attribut})$. Le paramètre est alors composé de deux parties : a qui est commun à tous les agents (et sur lequel nous verrons qu'il est donc facile de faire porter des expériences simulatoires), et $f(\text{attribut})$ qui apporte un complément de valeur, par exemple une composante liée à un scénario de température, ou plus souvent la variabilité de chaque agent. Cette deuxième option est donc souvent très utile.

Valider votre travail sur la tâche de déplacement. Nous allons maintenant créer d'un seul coup 25 agents tous différents et les répartir sur la grille. Sélectionner toutes les cellules, rappeler la boîte de dialogue de placement des animats et supprimer tout chargement précédent. Pour créer ces agents, nous allons utiliser l'outil d'édition de séries de valeurs. Sélectionner le moule de départ (donc notre moule 'sugar'), puis l'item "créer une série" dans le menu contextuel de la boîte 'Séries'. Si rien ne se passe, c'est que vous n'avez pas sélectionné de moule.



L'interface qui s'affiche permet d'affecter des listes de valeurs aux différents attributs de l'agent. Ces listes seront conservées dans un dictionnaire spécifique du moule, le dictionnaire des séries. Donner un nom à la série puis sélectionner successivement les attributs à traiter et appelez l'éditeur de série toujours via le menu contextuel. Exemple ci-dessous pour le rayon d'action, avec le tirage aléatoire de 25 entiers entre 0 et 3.



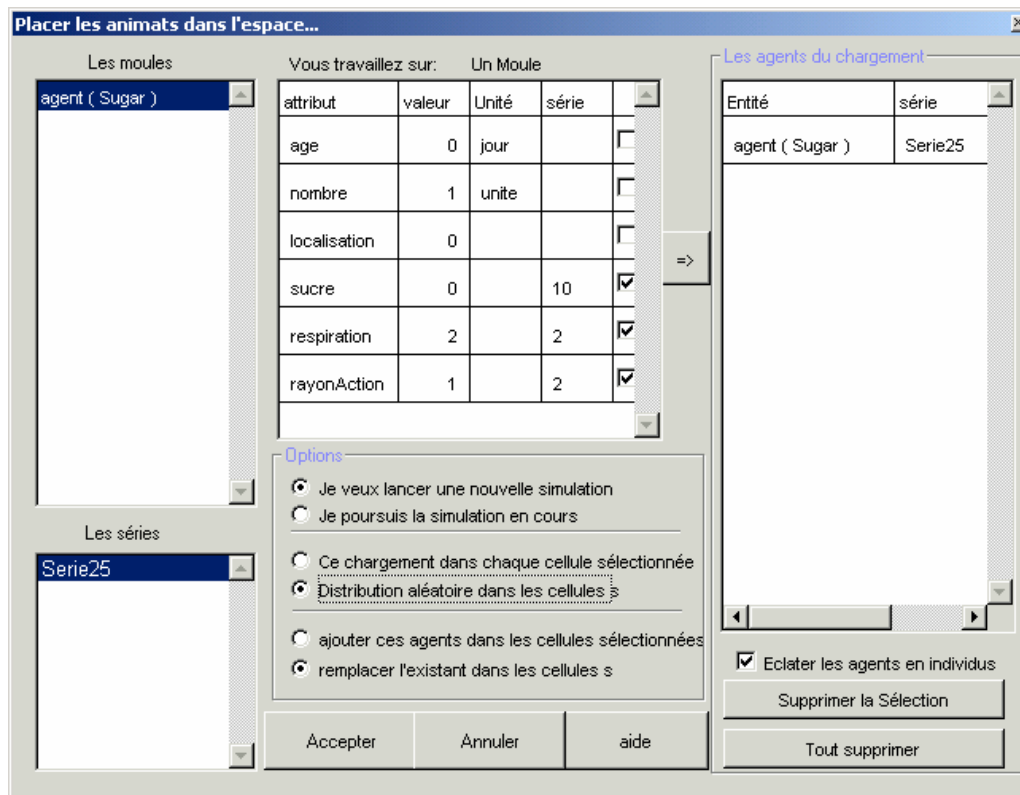
Série aléatoire entre 0 et 3 pour l'attribut rayonAction (25 éléments)

Remarque : Vous pouvez aussi saisir manuellement les valeurs ou les importer d'un fichier textuel ou par copier/coller. Consulter l'aide en ligne pour plus d'information sur cet éditeur.

Valider ce tirage. Attention, si vous renouvelez un tirage, les nouvelles valeurs **s'ajoutent** aux anciennes. Bien effacer toutes les valeurs avant d'effectuer un nouveau tirage pour n'avoir que 25 valeurs dans la liste.

Effectuer la même opération avec tirage entre 1 et 4 pour la respiration, et entre 4 et 10 pour le sucre initial. Appelez votre série "série25" par exemple, puis valider la. Elle apparaît maintenant dans la liste des séries.

Sélectionner la, puis effectuer son chargement à l'aide du menu contextuel 'charger la série'. Vérifier qu'elle est effectivement chargée dans le moule du peuplement (tableau central). Copier cet agent-série dans la liste de droite à l'aide de la flèche. Vous obtenez :



Fenêtre du chargement après création et chargement d'une série de 25 éléments

La petite case à cocher à droite de la série (tableau central) indique si vous souhaitez ou non utiliser la série pour créer les agents. Si elle n'est pas cochée, les agents garderont tous la valeur du moule pour cet attribut (donc la valeur de la première colonne du tableau central), mais la série reste mémorisée pour une utilisation ultérieure. Si c'est l'ensemble des séries que vous voulez désactiver, plutôt que de décocher chaque attribut, sélectionner 'Annuler la série' dans le menu contextuel des séries. **Attention, comme déjà dit, cliquez 2 fois dans la case pour pouvoir insérer votre chiffre dans les tableaux.**

Il faut ensuite dire ce que l'on fait de ces "agents de chargement" :

- Une case permet de spécifier si un agent doit être éclaté en individus. Ceci ne sert que si nombre est différent de 1, on éclate alors ce nombre en autant d'agents distincts, mais tous identiques.
- Deux dichotomies permettent de spécifier si l'on doit ajouter ou remplacer les agents préexistants des cellules sélectionnées, et si l'on doit répéter ce chargement dans chaque cellule ou le répartir aléatoirement. Attention, par défaut le chargement sera dupliqué dans chaque cellule sélectionnée. Bien cocher "distribution aléatoire".
- On spécifie également si ce chargement servira à une nouvelle simulation ou si on continue la simulation en cours.

Remarque : cette procédure de chargement manque encore un peu de souplesse, par exemple il est impossible de spécifier que l'on souhaite un agent unique (mais génétiquement tous différents) par cellule (cas au sens strict de l'exemple original Sugar). Le chargement aléatoire fait que deux agents peuvent se retrouver ensemble.

Valider. Un curseur vous donne l'avancement du travail de chargement. Lancer ensuite la simulation sur un ou deux mois. Le modèle arrive assez rapidement à un équilibre en nombre d'agent : seuls les mieux adaptés survivent.

Les agents se reproduisent

Nous allons maintenant faire se reproduire nos animaux. Ajoutez la tâche "Reproduction" à sugar :

Reproduction

agent (Sugar) Nom de la tâche: Reproduction

Si

	Variable	Opérateur	Valeur Seuil	Unité
▶	mon_age	est multiple de	4	jour

☒ ET ☐ OU
 Ajouter une condition Retirer une condition aide

Alors % des individus se reproduit en

Si

☒ Nombre fixe

☐ Aléatoire entre et

☐ Eclater les descendants en individus
cocher notamment si vous êtes en Individu Centre

(Défaut aucun héritage sauf localisation)

(Défaut aucune post-action)

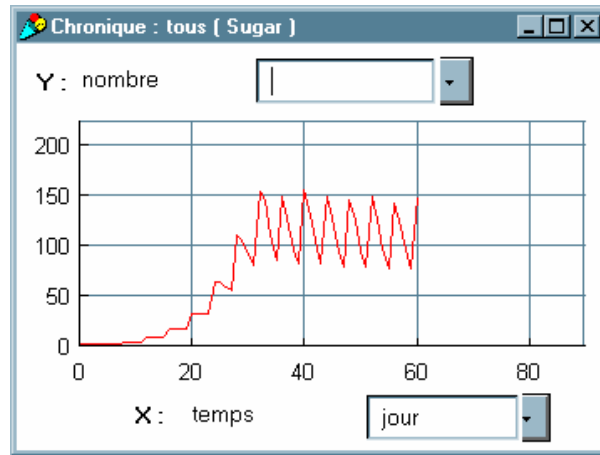
Interface de la définition de la tâche de 'Reproduction'

La reproduction est une tâche complexe qui demande de définir :

- quand se reproduire ? (interface de conditions)
- quel fraction des individus constituant l'agent se reproduit ? (équivalent à quelle probabilité de succès ?)
- en quoi se reproduire ?
- combien de descendants ?
- quelles valeurs initiales pour les descendants ?
- quelle action faire au besoin si on s'est reproduit ?

L'interface des conditions vous est maintenant familière. Demandez à votre agent de se reproduire tous les quatre jours par exemple. Sélectionnez "agent de sugar" comme descendant. Garder un descendant par individu, mais noter que le menu contextuel vous permettrait si nécessaire de lier ce paramètre à une autre variable comme la température de la cellule, le poids de l'agent ou l'âge du capitaine. Cocher par sécurité "éclater les descendants en individus" : au cas où vous cochiez plus d'un descendant, cela créera des agents individuels et non des agents "groupe d'individus". Dans un premier temps nous ne définirons pas d'héritage des valeurs des attributs entre parents et enfants : tout nouvel agent aura comme valeurs initiales les valeurs contenues dans le moule de l'agent. Valider.

Nous allons tester la reproduction en lançant une simulation avec un seul agent au départ. Cliquer sur la cellule où vous voulez le placer, rappeler la boîte de chargement, effacer tout chargement préexistant, et spécifier les attributs du moule de l'agent avec une respiration importante (ex. 3) pour ne pas leur rendre la vie trop facile et limiter le nombre d'agent finaux, un peu de sucre initial (ex. 4) et un rayon d'action égal à 2. Lancer l'exécution sur quelques jours, puis sur un ou deux mois. L'évolution du nombre total des agents devrait avoir l'allure ci-dessous, avec des variations cycliques d'abondance.



Chronique de l'évolution du nombre en fonction du temps

Ce modèle n'est bien sûr pas très satisfaisant : tous les agents sont identiques. Et même si nous étions partis d'agents différents au départ comme au paragraphe précédent, tous leurs descendants auraient été identiques au moule. Nous allons donc affiner l'héritage des attributs. Pour cela rappeler l'interface de reproduction et presser "Définir l'héritage". Les possibilités offertes par cette interface sont :

- Garder pour l'attribut la valeur par défaut (celle du moule : colonne 'valeur', que vous pouvez modifier)
- Choisir d'hériter de la valeur parentale : case 'hérit' à cocher
- Mettre un lien (simple ou via une fonction) entre valeur ou valeur héritée et un attribut
- Sélectionner aléatoirement une valeur dans une liste
- Panacher ces options selon différentes modalités

Essayez d'obtenir la figure ci-dessous, sachant que nous voulons que les descendants tirent aléatoirement leur respiration entre 2 et 5 compris, et leur sucre initial entre 4 et 9 (menu contextuel dans la boîte séries qui doit vous être familière, elle a le même fonctionnement que pour le chargement). Pour ces attributs, vous devez donc entrer les valeurs 2, 3, 4 et 5 dans la série (respectivement 4, 5, ... 9), et vous mettez l'option "1" (tirage simple dans la série) en dernière colonne. Attention, entrer ces séries en colonne, un chiffre par ligne. Pour le rayon d'action, nous voulons quelque chose d'un peu plus raffiné : celui-ci sera hérité de la valeur parentale, mais l'agent aura une chance sur trois d'augmenter ce rayon d'une cellule, une chance sur trois de le diminuer, et une chance sur trois de garder la valeur parentale. Vous allez donc créer pour rayonAction, une série avec les valeurs 1,0, et -1, puis cocher la case "hérité" et l'option "3". Notez que dans la figure ci-dessous la case 'hérit' de 'resp' est (didactiquement !) cochée par erreur. Mais cela n'a aucune incidence du fait de la valeur '1' sur le statut de cette série : la valeur initiale ou héritée n'est donc pas utilisée. Consultez l'aide en ligne au besoin.

Héritage des attributs

attribut	valeur	Unité	herit.	dépendance	Série	Unité	Statut
age	0	jour	<input type="checkbox"/>				0
localisation	0		<input checked="" type="checkbox"/>				0
sucres	0		<input type="checkbox"/>		9		1
respiration	2		<input checked="" type="checkbox"/>		5		1
rayonAction	1		<input checked="" type="checkbox"/>		1		3

Les séries

SerieHeritage

Valeur sera la valeur initiale de l'agent sauf si (de gauche à droite):


- hérit. est cochée, alors l'agent hérite de la valeur parentale
- dépendance (menu contextuel) dans la case dép.: dépendance sur valeur / valeur héritée
- s'il existe une série, alors selon statut :

- 0- la Série n'est pas prise en compte
- 1- une valeur est tirée dans la série
- 2- valeur (ou valeur modifiée) est multipliée par une valeur de la série
- 3- valeur (ou valeur modifiée) est additionnée à une valeur de la série

Accepter

Annuler

Relancer la simulation en partant toujours d'un agent unique. Les animats sont rapidement plus nombreux que dans le cas précédent : ils sont mieux adaptés à leur environnement. Le nombre total d'animats s'affiche en continu, à côté du pas de temps en cours et de la durée totale de la simulation, dans l'interface principale de Mobidyc. Si la population explose, vous pouvez interrompre une simulation avec le bouton 'arrêter', et dans les cas graves avec Ctrl Y. En principe, vous ne perdez rien, mais il est toujours préférable de sauver ses modifications avant de lancer une simulation.

On s'en convainc aisément en sélectionnant l'outil d'animation sur les histogrammes de respiration des agents : les fortes respirations sont systématiquement contre sélectionnées. Mais elles ne disparaissent pas définitivement car des agents à forte respiration naissent à chaque instant. Cliquer sur le bouton du gestionnaire des histogrammes  et préciser ce que vous souhaitez visualiser ainsi que les limites (min et max, le défaut, -1, voulant dire que on laisse Mobidyc chercher) et le pas de l'histogramme. Vous obtiendrez la fenêtre suivante:

Gestionnaire des Histogrammes

Entité/Espèce: Sugar

Agent/Stade: tous

Groupe: tous

Attribut: respiration

Date: 60

Pas: 1

Min: 2

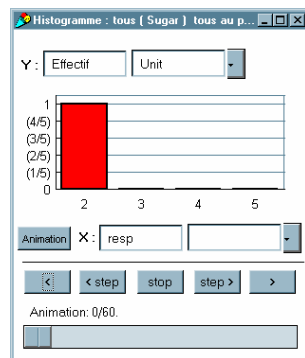
Max: 5

Dessin

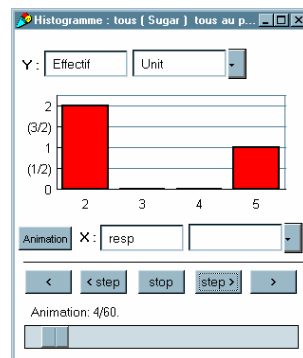
Aide

Cliquez sur le bouton 'Dessin', puis sur le bouton 'Animation' (bug, "animation" ne fonctionne plus sur les PC). Voici un exemple d'histogrammes à différents temps de la simulation.

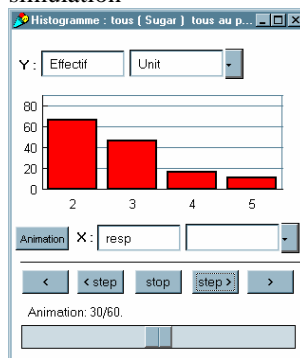
Temps initial (t=0)



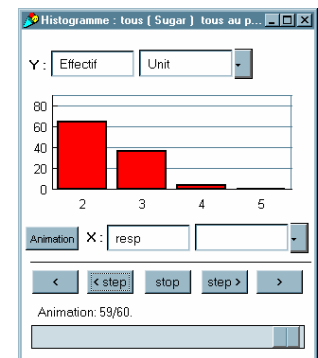
Première reproduction



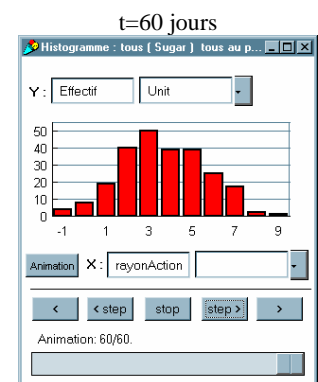
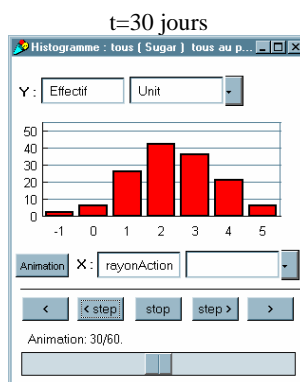
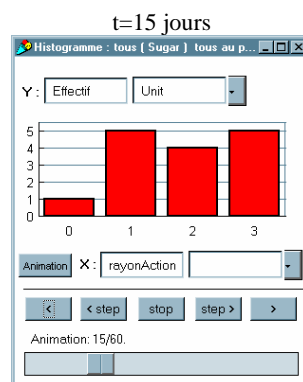
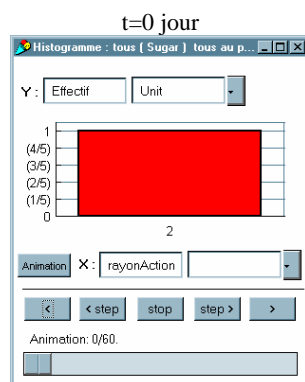
Après un mois de simulation



Vers la fin de la simulation



De la même façon, nous pouvons visualiser les histogrammes du rayon d'action. N'oubliez pas de corriger les limites (min et max) en utilisant par exemple les valeurs par défaut (-1 et -1) qui font une recherche automatique des bornes.



Remarque : pour un rayon d'action négatif l'agent reste sur place (équivalent donc à un rayon d'action égal à 0).

Nous avons pleinement utilisé une qualité importante de Mobidyc, la mise au point progressive d'un modèle afin d'être toujours sûr d'un comportement avant d'en ajouter un autre. Nous pouvons maintenant passer à des simulations en vraie grandeur.

Mais nous allons maintenant payer cette souplesse d'utilisation. Car les calculs seront nécessairement beaucoup plus lents qu'avec un simulateur dédié à un modèle particulier et écrit dans un langage compilé type C++. Pour garder un temps de réponse raisonnable dans le cadre de ce tutorial, nous allons nous limiter à une grille 30x30 et à une centaine d'animats contre une grille 50x50 et 500 agents dans le modèle originel.

Chargez la grille spatiale 'Sugar30.esp' qui est fourni avec le logiciel et qui se trouve en principe dans votre répertoire "projets" (menu Espace -> Charger un espace). Si vous ne l'avez pas encore fait, ayez la curiosité d'aller voir la structure de ce fichier textuel (sous un tableur par exemple) qui donne à Mobidyc toutes les informations pour créer l'état initial de la grille spatiale. Notez que si certains attributs existent dans ce fichier mais pas dans le moule de vos cellules, Mobidyc va (en principe !) créer automatiquement les attributs manquants pour garder la compatibilité avec le moule. Mais ces nouveaux attributs n'étant évidemment pas impliqués dans les tâches, vous n'obtiendrez pas le comportement souhaité pour le modèle. Attention donc à la correspondance entre les noms des attributs. Le cas échéant, modifier les noms dans le fichier Sugar30.esp (sauvegarde texte brut, séparateur tabulation) pour les adapter aux noms d'attributs que vous avez choisis pour les cellules. Ce sera plus rapide que de les modifier dans le moule puis dans toutes les tâches où ces attributs sont utilisés. Remarquez également que l'attribut 'sucre' des cellules est initialisé par la valeur lue dans le fichier. La tâche initCellule est donc inutile, vous pouvez la désactiver.

Créer maintenant une série de 250 animats à placer sur la grille, selon la même procédure que précédemment. Utiliser l'éditeur de séries pour générer aléatoirement une série sur l'âge (entre 1 et 4 pour désynchroniser les reproduction), et sur le sucre initial (entre 4 et 9). Fixons la valeur initiale de respiration à 3 (valeur élevée), pour tous, et un rayon d'action de 1. Les agents initiaux ne sont donc pas très bien armés pour la vie...

attribut	valeur	Unit	série
resp	2		4
age	0	jour	
rapport	0.0		
sucreDeSugar	4		9
rayonAction	2		3
localisation	0		
nombre	1	unite	

Valider votre série, charger la dans le moule de l'agent Sugar, et ajouter cet agent-série dans la liste de droite (après avoir au besoin vidé le chargement précédent). Sélectionner 'nouvelle simulation', 'distribution aléatoire dans les cellules' et 'remplacer les agents existants dans la cellule'. Accepter ce chargement et visualiser la grille avant de lancer une simulation : les 250 animats doivent se répartir aléatoirement sur tout l'espace. Au besoin, utiliser le menu 'Option->préférences pour modifier la taille relative des cellules et masquer les limites de cellules.

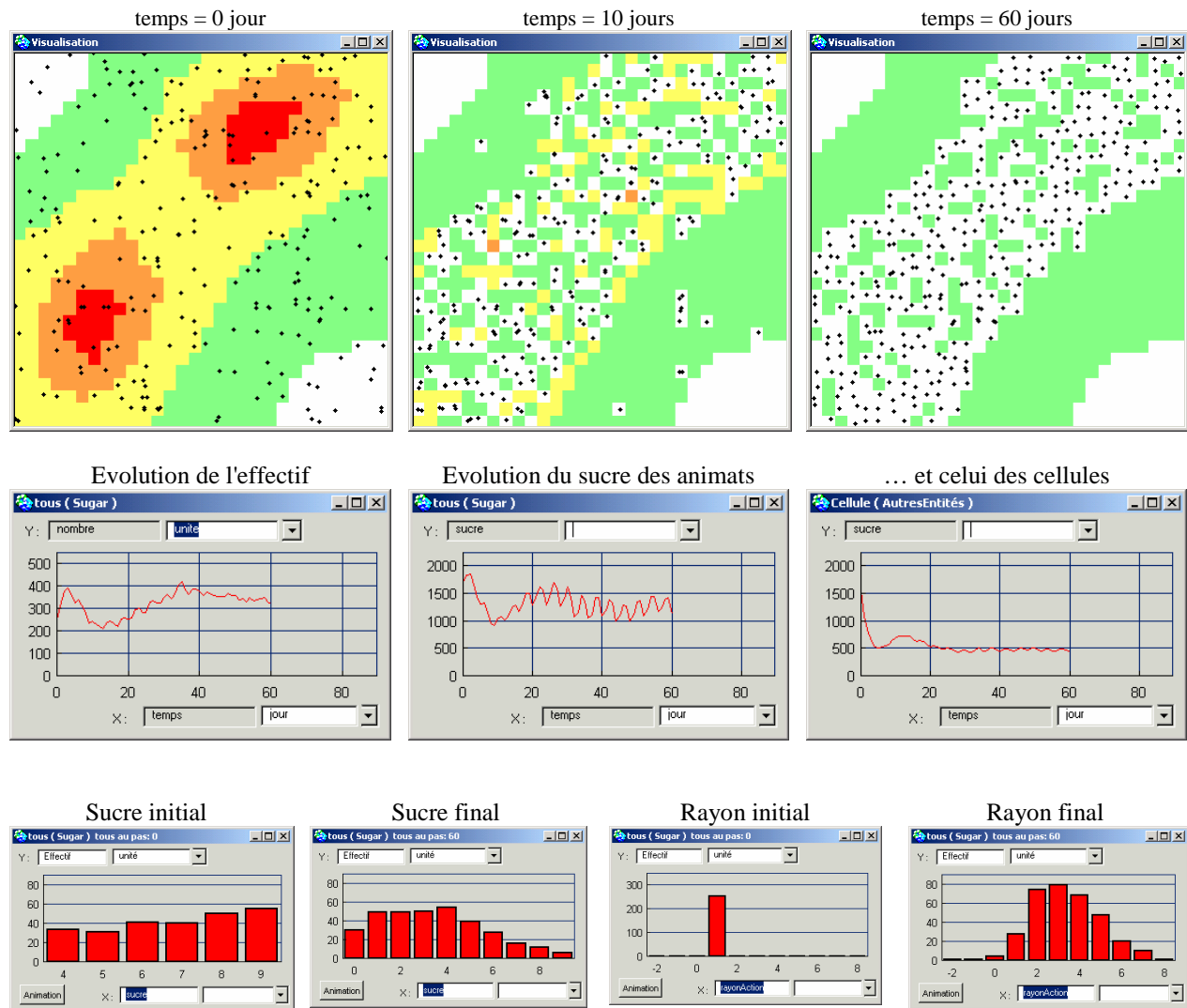
Lancer une simulation sur 1 semaine puis sur 1 ou 2 mois. Les agents se concentrent dans les zones les plus riches. Si votre ordinateur est un peu lent, fermer la fenêtre de visualisation ou décocher la tâche 'Exécuter visualisation' dans le scheduler et allez prendre un café. En fin de simulation, vous pourrez utiliser le magnétoscope pour visualiser la grille et les agents à différents temps de la simulation, ainsi que les chroniques et les histogrammes. Vous constaterez

que la population, après une courte période de croissance, diminue puis remonte, les agents s'adaptant progressivement à leur milieu : les agents à fort rayon d'action et faible respiration supplantent les agents initiaux.

Remarque importante :

Si vous avez des difficultés à visualiser des chroniques ou des histogrammes c'est souvent que :

- Vous n'avez pas sélectionné toutes les cellules : les résultats ne concernent que les cellules sélectionnées.
- Vous avez désactivé l'option de mémorisation des résultats de vos agents dans l'ordonnanceur.
- Vous avez décoché certains attributs dans la rubrique "sauvegarde" de l'ordonnanceur.



Assembler des primitives pour former des tâches : l'exemple chèvre-loup

Nous allons maintenant aller plus avant dans la construction de tâches, en nous intéressant à un deuxième exemple, des loups cherchant à attraper des chèvres qui, elles, cherchent à les éviter. Mais avant cela, regardons d'un peu plus près le principe de fonctionnement des primitives :

Principe

Nous avons vu dans l'exemple SugarScape comment définir une tâche utilisateur très simple. Elle consistait à faire précéder la tâche prédéfinie 'modifier attribut' de la primitive 'Si...Alors'. Une tâche plus complexe consiste à faire des enchaînements plus longs, le résultat d'une primitive, s'il y en a un, étant directement transmis à la primitive suivante.

On peut donc distinguer 3 classes de primitives, selon leurs arguments :

Les primitives amorces. Elles ne possèdent pas d'argument d'entrée, mais elles possèdent un argument de sortie. Elles servent au démarrage d'une tâche.

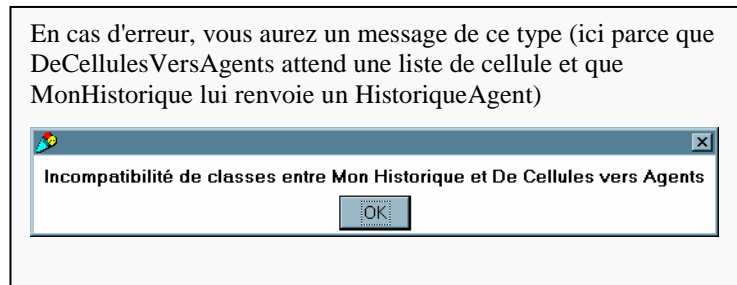
Les primitives centrales. Elles possèdent un argument d'entrée et un argument de sortie.

Les primitives de clôture. Elles possèdent un argument d'entrée mais pas d'argument de sortie. Elles sont souvent utilisées à la fin d'une tâche. Mais une primitive centrale peut également terminer une tâche.

Les tâches sont considérées comme des primitives sans arguments et peuvent donc, comme nous l'avons déjà fait, entrer dans une composition.

Cohérence entre arguments

Lorsqu'on a défini un enchaînement, Mobidyc contrôle sa cohérence. Deux primitives qui échangent un argument doivent se comprendre, l'argument de sortie de la première doit être compatible avec l'argument d'entrée de la deuxième. Mobidyc contrôle le type de l'argument et sa dimension. Suivant la logique 'objet', toute primitive acceptant un type de classe accepte aussi toutes ses sous-classes. En ce qui concerne la dimension, Mobidyc ne gère que les objets simples (dimension 0) et les listes d'objets (dimension 1). Toute primitive sachant gérer une liste doit savoir gérer un simple objet, l'inverse n'étant pas vrai.



Les différents types d'objets manipulés par les primitives

Afin de limiter le nombre de primitives et de simplifier leur fonctionnement, il est important de limiter le nombre d'objets qu'elles auront à manipuler. Nous avons choisi de ne leur faire s'échanger que des agents (ou des structures apparentées) ainsi qu'une structure pouvant contenir des données, le T3D (ou tableau à trois dimensions). Mais avant de détailler ces objets, il faut nous faire une idée du vaste monde dans lequel sont plongés les agents Mobidyc...

Vous avez pu constater que l'élaboration d'un modèle implique trois phases que l'on boucle autant de fois que nécessaire : la définition des moules, la simulation, et l'observation des résultats. Pour ces trois phases, nous avons choisi de mettre en jeu trois structures d'agent particulières, évidemment fortement apparentées : le moule, l'agent réel, et son historique. Ajoutons-y pour faire bonne mesure une quatrième structure, l'image de l'agent, destinée à mémoriser l'état de l'agent à un instant donné, en général en début de pas de temps, pour permettre de proposer un mode synchrone d'appel des agents. Un agent est donc "quadruple", et ces quatre structures sont immergées dans quatre "mondes" différents, eux aussi fortement apparentés (sauf le premier), le GestionnaireDesEntités, le Monde réel, L'Histoire du Monde, et l'Image du Monde. Pour gagner du temps et de la mémoire, l'Image du Monde n'est

activée que si le mode synchrone est demandé (mode qui est donc plus lent que le mode séquentiel), et l'HistoireDuMonde que si la mémorisation des agents est activée.

En langage objet, dire que ces structures sont apparentées, veut dire qu'elles héritent d'une structure commune. En particulier, toutes ces structures d'agents partagent le fait qu'elles ont un nom et des attributs, c'est-à-dire un état. Cet objet commun est le ProtoAgent. Donc toute primitive qui peut se contenter de travailler sur la structure héritée de ProtoAgent peut travailler sur toutes les sous classes de ProtoAgent, c'est-à-dire dans la pratique sur toutes les structures liées aux agents.

Résumé des principales classes impliquées dans les agents Mobidyc. En gras les objets manipulés par les primitives.

Nom	Super Classe	Rôle	Variables principales
ProtoAgent	Objet	Superclasse de toutes les structures d'agents. Gère le dictionnaire des attributs (état)	- dictionnaire des attributs
Imago	ProtoAgent	Gère le dictionnaire des tâches (comportement) et les options de sauvegarde pour l'agent	- dictionnaire des tâches - objet Sauvegarde
Agent	Imago	Superclasse de tous les agents réels. Gère l'accès à l'historique (la mémoire dans le temps) et à l'image (pour le mode parallèle) des agents	- historique - image
Animat	Agent	Ces instances seront les animats	
Cellule	Agent		Dessin voisins
AgentNonSitué	Agent		
ImageAgent	ProtoAgent	Superclasse de toutes les structures de mémorisation d'agents (historiques et images)	agentCorrespondant
HistoriqueAgent	ImageAgent	Classe gérant la mémorisation dans le temps des attributs des agents, en accord avec les options de l'objet sauvegarde	
T3D	Objet	Tableau à trois dimensions permettant de stocker les valeurs d'un attribut d'un ou de plusieurs agents, éventuellement dans le temps, que l'attribut soit de type standard (scalaire) ou de type liste	- Dim1 valeur courante des attributs des différents agents. - Dim2: valeurs dans le temps - Dim3: Utilisée si l'attribut est de type 'liste'

Les différents types de primitives

Nous pouvons maintenant étudier les différentes primitives, classées par grands groupes : amorces, sélection, transition, calcul, fin de tâche, et contrôle. Dans ce dernier groupe, vous remarquerez les primitives 'TantQue' et 'DéroulerlesEntrées' qui permettent de quitter le mode strictement linéaire de l'enchaînement.

Nom	Fonction	Entrée	Sortie
Amorces			
Moi	Renvoie l'agent actif		Un Agent
MaCellule	Renvoie la cellule de l'agent actif		Une cellule
TousLesAnimats	Renvoie tous les animats		Des animats
ToutesLesCellules	Renvoie toutes les cellules		Des cellules
MonVoisinage	Renvoie les cellules voisines de l'agent actif selon un rayon		Des cellules
MonHistorique	Renvoie l'Historique de l'agent actif		Un HistoriqueAgent

MonHistoriqueAttribut	Renvoie un tableau donnant toutes les valeurs prises au cours du temps par l'attribut spécifié		Un T3D
Selection			
TrierSurNom	Sélectionne les animats correspondants au nom spécifié	Des animats	Des animats
TrierSurAttributs	Sélectionne les agents sur des conditions à vérifier sur leurs attributs	Des agents	Des agents
SousSelection	Sélection aléatoire d'une fraction de l'entrée (proportionnelle ou absolue)	Des objets	Des objets
Choix final	Sélection d'un agent selon des conditions sur ses attributs (aléatoire, le plus près, le plus gros...)	Des agents	Un agent
VoisinageCellule	Renvoie le voisinage de la cellule fournie en entrée	Une cellule	Des cellules
Transision			
DeAnimatsVersCellules	Renvoie les cellules des animats	Des animats	Des cellules
DeCellulesVersAnimats	Renvoie les animats présents sur les cellules	Des cellules	Des animats
DeAgentsVers Historiques	Renvoie les historiques des agents	Des agents	Des historiquesAgents
DeAgentsVersValeurs	Renvoie un T3D contenant les valeurs de l'attribut spécifié. Dim: valeur courante des attributs des différents agents. Dim2: valeurs dans le temps (si Historiques). Dim3: Utilisée si l'attribut est de type 'liste' et non 'scalaire'	Des protoAgents	Un T3D
Calcul			
Compte	Compte les objets fournis en entrée et stocke la valeur dans l'attribut spécifié	Des objets	Sortie = entrée
ModifierAttribut	Calculs mathématiques impliquant des attributs de l'agent actif et de l'agent fourni en entrée. Si plusieurs agents en entrée, répète l'opération pour chaque	Des agents	Sortie = entrée
CalculSurT3D	Calculs standards (somme, moyenne, max...) sur une des dimensions (agents, temps, ou valeurs d'un attribut de type 'liste') du tableau. Celui-ci perd une dimension.	Un T3D	Un T3D
Fin de tâche			

SauverValeurs	Si le tableau est de dim 0 ou 1, sauve les valeurs du T3D dans l'attribut spécifié qui doit être de type standard (dim0) ou de type liste (dim1)	Un T3D	Sortie = entrée
DéplacementCiblé	Pour un animat. Déplace l'animat vers la cellule cible. Ou bien éloigne ou rapproche l'animat de la cellule cible selon la vitesse spécifiée	Une cellule	Sortie = entrée
Tuer	Tous les animats fournis en entrée	Des animats	
Contrôle			
SiAlors	Ne continue l'exécution des primitives qui suivent si des conditions sont remplies		
TantQue / FinTantQue	Relance les primitives concernées en boucle tant que des conditions sont satisfaites		
DéroulerLesEntrées / FinDéroulerLesEntrées	Déroule la liste d'entrée et donne les objets un par un aux primitives suivantes. S'arrête en fin de liste ou sur conditions	Des objets	Un objet

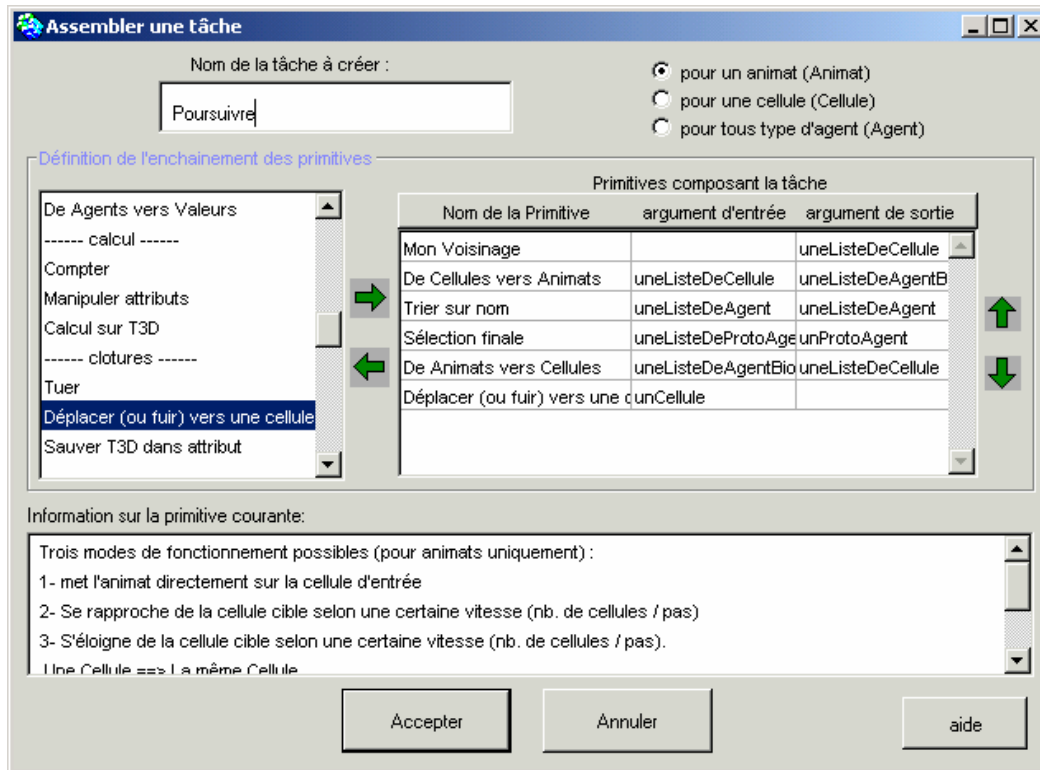
Un loup poursuivant des chèvres

Mais revenons à notre loup. Son comportement peut se décomposer de la manière suivante :

scrutation du voisinage
localisation de la chèvre la plus proche
chercher à s'en rapprocher
si elle est atteinte, la dévorer

Le comportement de la chèvre est très voisin, sauf qu'au lieu de localiser une chèvre, elle va chercher les loups, et chercher à s'en éloigner au lieu de s'en rapprocher...

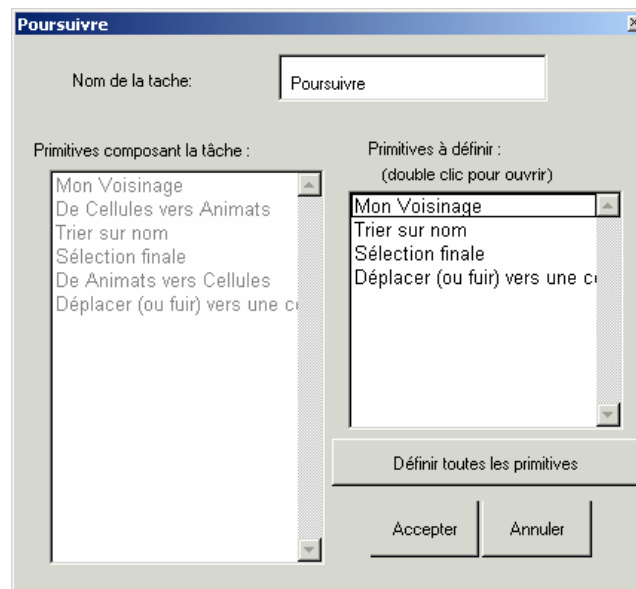
Ce simple enchaînement des actions nous dicte pratiquement l'enchaînement des primitives. Ouvrir la fenêtre d'assemblage des tâches et obtenir le déroulement suivant :



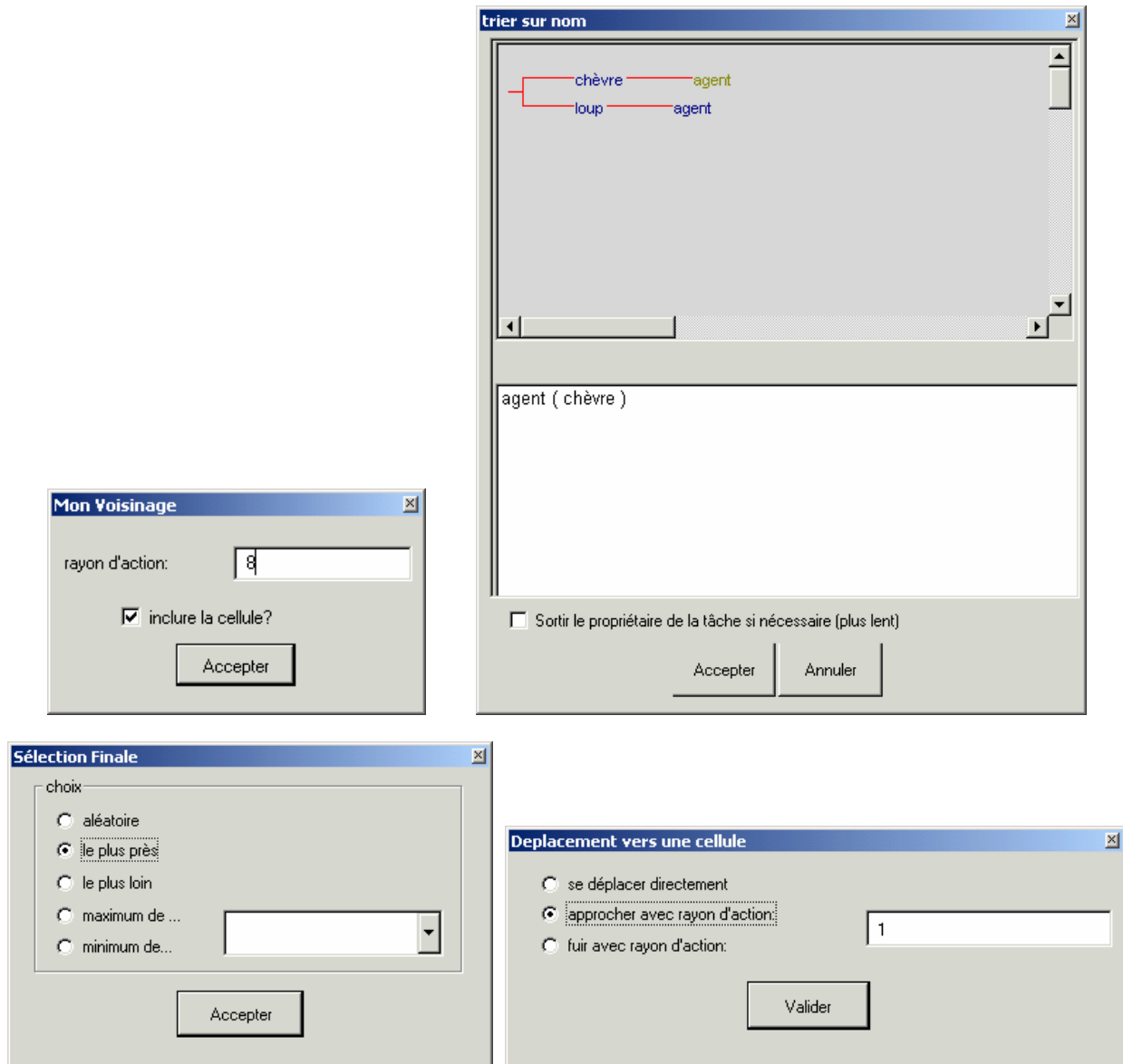
Assemblage de la tâche de déplacement des loups, qui servira également aux chèvres

N'oubliez pas de donner un nom à cette tâche et valider.

Ouvrir la fenêtre de définition des animats et créer vos deux moule (chèvre et loup) qui peuvent appartenir à la même entité ou à deux entités différentes. Commençons par ajouter la tâche 'poursuivre' au loup. Vous devez obtenir la fenêtre suivante :



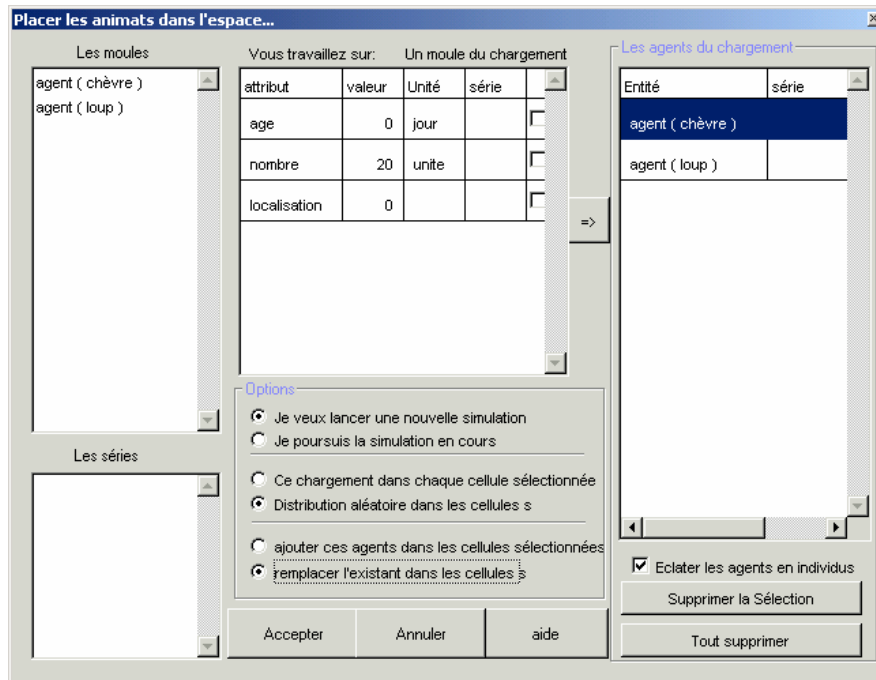
Cliquer sur "définir toutes le primitives", et paramétrer vos primitives comme ci-dessous, c'est-à-dire un rayon d'action de 8 cellules pour le loup, qui cherche les chèvres, qui localise la plus proche, et qui s'en approche d'une cellule par pas de temps.



Paramétrage des 4 primitives de la tâche de poursuite du loup

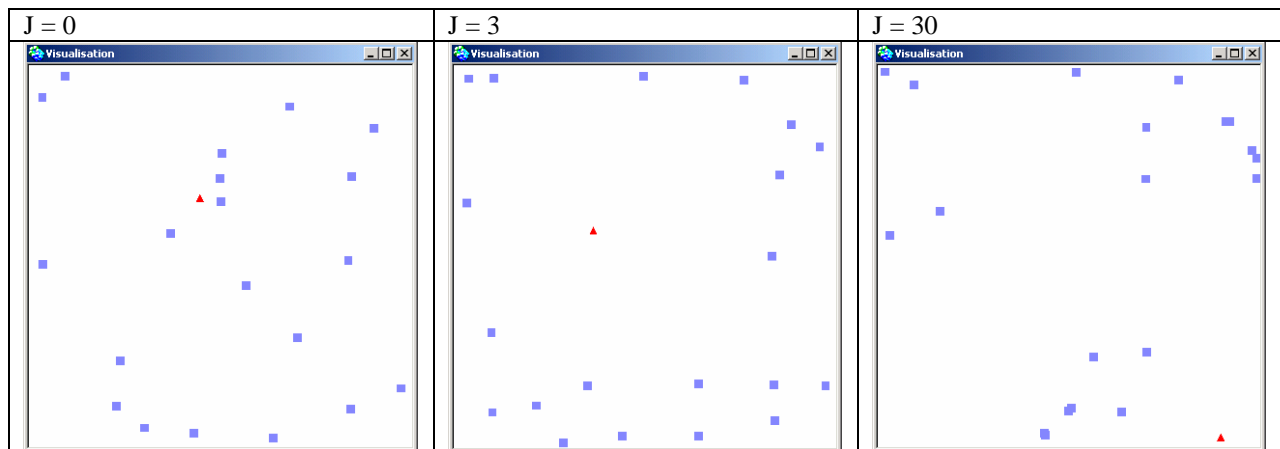
Pour la chèvre, utilisez la même tâche, mais avec un rayon de 5 cellules (elles sont un peu myopes !), et une vitesse de fuite de 2 cellules.

Créez une grille pas trop grosse (15x15, carrée à 8 voisins par exemple), et placez aléatoirement dans la grille une vingtaine de chèvre et un loup. Voici un exemple de fenêtre de placement. Attention de bien cocher 'éclater en individus' pour faire éclater votre groupe de 20 chèvres, et 'distribution aléatoire dans les cellules'.



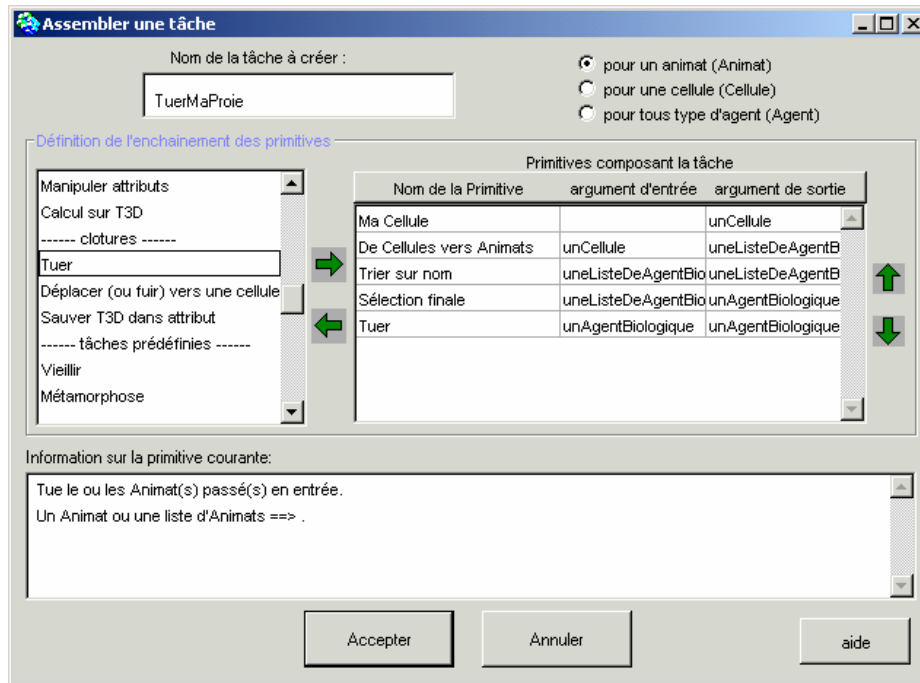
Chargement de 20 chèvres et un loup

Définir un observatoire sur vos animaux, sauver votre projet, et lancez la simulation sur 1 mois. Vous devez obtenir quelque chose qui ressemble à la figure ci-dessous. Au besoin, aller dans Option-> préférences pour modifier la taille relative des agents et masquer les traits de cellules.



Les loups mangent les chèvres

Revenons à notre loup qui doit, s'il rejoint une chèvre, la manger. Notre loup va donc regarder sa cellule pour voir si une chèvre s'y trouve et la manger. Il faut assembler une nouvelle tâche comme ci-dessous :



Assemblage de la tâche de prédation du loup

On remarquera que cette tâche reprend une bonne partie de la tâche précédente. Le loup localise la chèvre deux fois, une fois pour s'en approcher, et une deuxième fois pour voir s'il la atteinte. Cette redondance est une grosse limitation du mode 'linéaire' de l'assemblage des primitives : un objet localisé à l'issue d'une partie du travail ne peut pas être réutilisé dans une autre partie de la procédure. Mais un module de composition plus souple a été développé à partir du récent travail d'un doctorant en informatique. Il permet (en autres) d'utiliser un argument plusieurs fois. Ce module n'est pas encore totalement finalisé, mais peut déjà être utilisé. Il s'agit du module Myctalk, développé par Reza Razavi, qui se trouve dans le répertoire 'plugin'. Nous vous invitons à le tester dès que possible (Options-> charger un patch), vous serez étonnés par sa souplesse.

De l'importance du mode d'appel des agents (synchronisation)

Revenons à notre modèle. Ajoutez la tâche de prédation au loup, qui pointe donc sur la chèvre, et dont la sélection finale (au cas où plusieurs chèvres se trouvent dans la même cellule) est aléatoire. Sauvez votre modèle et relancez votre simulation. Le loup a bien du mal à parvenir à ses fins, les chèvres y voient bien et courent vite ! Mais il peut arriver à coincer une chèvre dans un coin ou sur un bord, aidé en cela par un artefact de la simulation. Par défaut en effet, les animats sont appelés à jouer en mode séquentiel, l'un après l'autre. Et cet ordre est aléatoire, avec retraitage à chaque pas de temps. Il peut donc se faire que le loup puisse s'approcher deux fois de suite d'une chèvre avant que celle-ci ait son tour (et inversement), ce qui aide évidemment beaucoup notre prédateur. Pour vous convaincre de l'importance de la chose, passez vos animats en mode parallèle grâce au Scheduler. Si vos cellules ont huit voisins, vos loups n'attrapent plus les chèvres. Sur cet exemple, la taille de la grille, sa topologie (ouverte ou fermée, nombre de voisins), et le mode de synchronisation peuvent influencer les résultats. C'est souvent le cas dans les modèles à interactions fortes entre agents comme ici.

Fini de rire pour les chèvres, refaites une simulation avec trois ou quatre loups au lieu d'un. Pour cela, plutôt que de supprimer l'ancien chargement et de tout recommencer, rappelez votre 'moule de chargement loup' dans la boîte de droite, et modifiez l'attribut 'nombre' pour le passer de 1 à 3. Vérifiez que vos chèvres sont toujours vingt et que 'éclater en individus' est bien coché, et validez. La chasse des loups est bien plus efficace, et même en mode synchrone ils parviennent à attraper quelques chèvres. Mais que se passe-t-il si deux loups attrapent une même chèvre ? Ici, rien de spécial, la chèvre sera tuée deux fois. Mais si les loups la mangent, et augmentent leur poids en conséquence ? Il y a conflit... et pas de solution unique. On peut imaginer un partage plus ou moins équitable ou laisser un seul des loups en présence s'arroger la totalité de la chèvre. La gestion fine d'un conflit de ressource, autre que la solution de servir aléatoirement un seul des protagonistes (et seule gérée (en principe !) par les primitives), et

toujours une affaire délicate et nécessitera la plupart du temps l'écriture d'une procédure spécifique. Ces procédures pourront éventuellement déboucher sur de nouvelles primitives si elles se révèlent suffisamment génériques. Avis aux amateurs développeurs, ils sont les bienvenus !

Enchaîner des expériences simulatoires. Le mode 'Batch'.

La plupart des modèles individu centrés, même les plus simples, sont trop complexes pour que l'on puisse se faire a priori une idée précise des relations entre les paramètres et la réponse du modèle. Les expériences simulatoires sont là pour cela. Mobidyc propose quatre plans d'expérience, accessibles via le menu contextuel du champ de saisie 'Expériences simulatoires' :

1- Saisie manuelle du plan. Vous entrez manuellement les combinaisons des paramètres à tester. Chaque combinaison peut être répétée autant de fois que souhaité. Si aucun paramètre n'est spécifié, fait une simple répétition des simulations.

2- Exploration paramètre par paramètre. Pour chaque paramètre, vous entrez la gamme de variation à tester, en combien de valeurs découper cette gamme, et le nombre de répliqués par valeur. Exemple, un taux de croissance à tester entre 0.3 et 0.9 g/j, découper en 10 valeurs, chaque valeur étant répliquée 5 fois. Attention à votre découpage pour les paramètres à valeurs discrètes.

3- Plan d'expérience complet (pour ANOVA). Vous entrez les valeurs des niveaux des différents paramètres que vous souhaitez tester, et Mobidyc crée le plan de toutes les combinaisons possibles. Attention, la combinatoire explose vite. Si vous prenez quatre niveaux par paramètres et 6 paramètres, vous obtenez déjà $4*6 = 4096$ simulations, à multiplier par le nombre de répliqués.

4- Plan d'analyse de sensibilité (pour paramètres à valeurs continues uniquement). Vous entrez le nombre total de simulations désirées (au moins 100) et la plage de variation des paramètres numériques continus à tester (défaut, dans un voisinage de 10% autour de la valeur d'origine). Mobidyc fera les simulations en tirant aléatoirement la valeur des paramètres à tester au voisinage de leur valeur par défaut.

Un module de dépouillement vient d'être développé par Nicolas Labat, à l'aide du logiciel de statistiques 'R'. Il prend en entrée le fichier de résultats de Mobidyc, et permet de traiter tous ces plans, avec en particulier la recherche de la meilleure transformation possible de la sortie du modèle, le dépouillement de l'anova, ou le calcul de l'évolution des fonctions de sensibilités du modèle dans le temps. Vous êtes cordialement invités à le tester... et à participer à son développement !

Principe de fonctionnement

Les agents partageants tous le même dictionnaire de tâches (stocké dans le moule des agents), modifier un paramètre dans une tâche, provoque immédiatement une modification de tous les agents concernés par la tâche... ce qui permet de batcher très facilement !

Corollaire, si ce paramètre est lié par une dépendance (simple) à un attribut d'un agent, la valeur du paramètre que l'on souhaite modifier n'est plus accessible car chaque agent en est propriétaire. On tourne la difficulté en utilisant une dépendance complexe (via le menu contextuel) pour le paramètre de la tâche. Mettre une dépendance de type "paramètre + fonction identité" par exemple. En procédant ainsi la valeur du paramètre reste interne à la tâche, et représentera par exemple la valeur moyenne du paramètre sur laquelle on pourra batcher. Et la partie dépendance pointe (ici via la fonction identité) vers l'attribut de l'agent. Notons que ce dernier ne porte alors plus la valeur du paramètre, mais l'écart à la moyenne.

Remarque importante : A l'heure actuelle, seuls les paramètres des tâches sont donc accessibles au mode Batch. Il est donc impossible de faire automatiquement varier les conditions initiales (nombre d'animats, état initial, taille et état de la grille, position initiale des animats dans la grille etc...). Dans certains cas, il est possible de faire passer certaines conditions initiales par des tâches. Mais la plupart du temps, il vous faudra faire ces expériences à la main, en attendant une solution plus élégante. Avis aux amateurs développeurs...

Nous allons illustrer le fonctionnement du batch sur notre exemple chèvre-loup, en comparant l'effet du champ de vision respectif des chèvres et des loups sur le total de chèvres attrapées.

Commençons par définir notre plan (plan simple, saisie manuelle). Nous allons croiser deux valeurs de vision des chèvres et des loups (par exemple 10 et 3) ce qui nous conduit à quatre expériences comme le plan ci-dessous. Répétons ce plan 10 voire 30 fois car la variabilité de nos simulations est grande. Nous aurons donc 120 simulations. Réduire le nombre de réplicats pour ce tutorial si votre machine est lente. Remarquons qu'il s'agit d'un plan complet (nous testons toutes les combinaisons possibles, nous aurions donc pu aussi bien utiliser l'option plan complet. Les fichiers résultats ont exactement la même allure quel que soit le plan. Valider ce plan.

Plan définit manuellement...

Nom de l'expérience : [Valider] [Annuler] [aide]

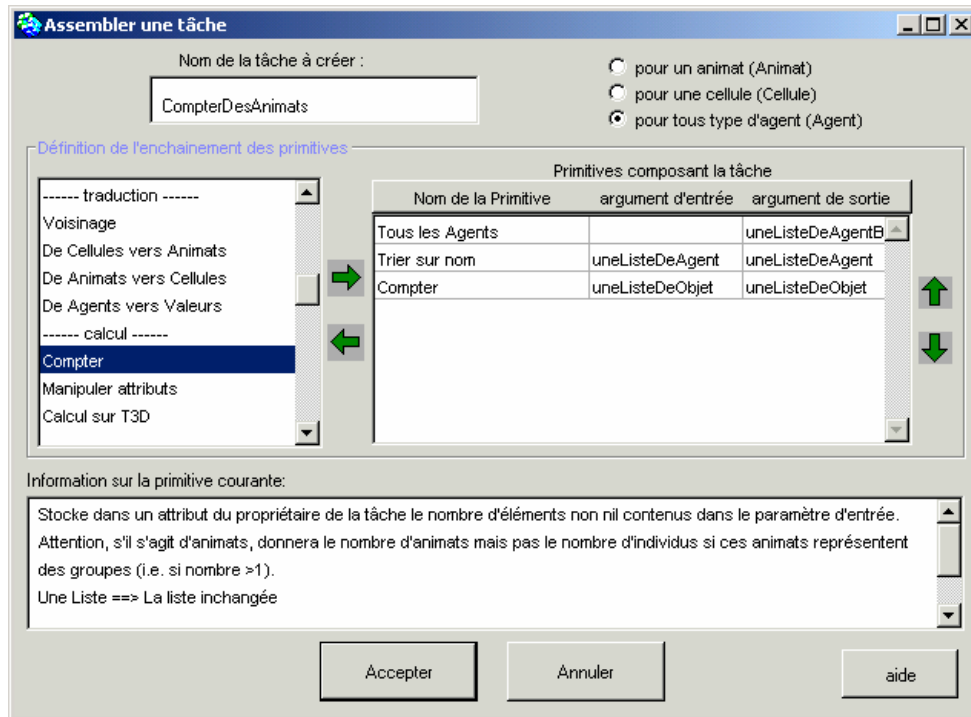
Nombre d'expériences : Nombre de réplicats : [Ajouter/Supprimer paramètres]

	param 1	param 2	param 3	param 4	param 5
	10	10			
	10	3			
	3	10			
	3	3			

	chemin	valeur	unité
param1	chèvre_agent_Fuir_Mon Voisinage_rayon d'action	5	
param2	loup_agent_Poursuivre_Mon Voisinage_rayon d'action	8	

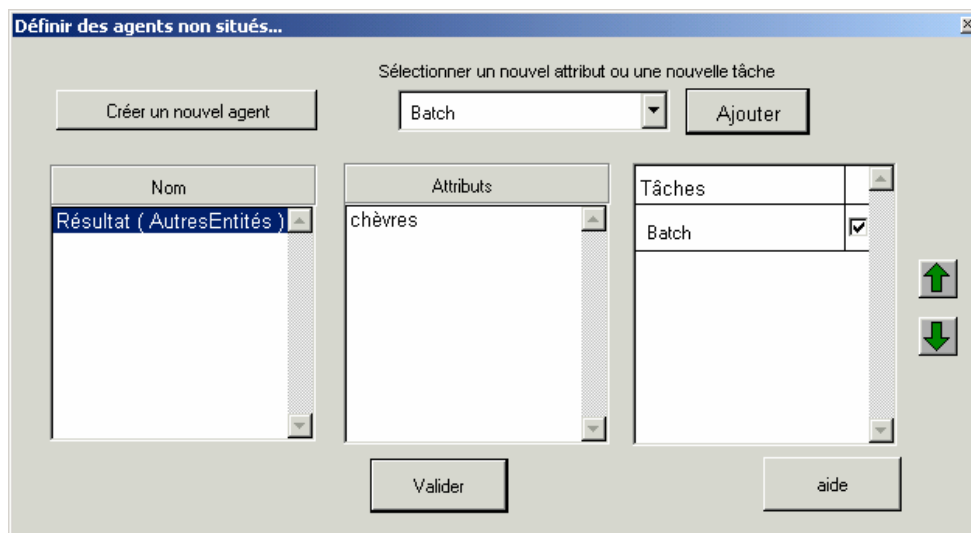
Plan d'expérience à deux paramètres et deux modalités

Il faut maintenant préciser la (ou les) variable(s) de sorties que nous voulons observer. En mode Batch, il est en effet hors de question de tout sauvegarder à tous les pas de temps comme c'est le cas par défaut en mode interactif. Le réflexe 'Mobidygien' est de penser immédiatement qu'on va confier cette tâche particulière... à un agent bien sûr. Le plus logique est de confier cette tâche à un agent non situé, mais tout agent peut la porter. Nous allons donc créer un agent non situé que nous appellerons 'résultat' par exemple, et qui contiendra un attribut 'chèvre' qui portera le résultat que l'on souhaite observer. Ce résultat, c'est la population totale de chèvres en fin de simulation. Il faut la calculer. C'est une tâche très simple, mais qui n'est pas prédéfinie par Mobidyc, nous allons donc la créer à l'aide des primitives et l'appeler 'CompterDesAnimats' par exemple. Remarquez que cette tâche est assez générique, et que vous pourrez facilement l'utiliser dans d'autres modèles.

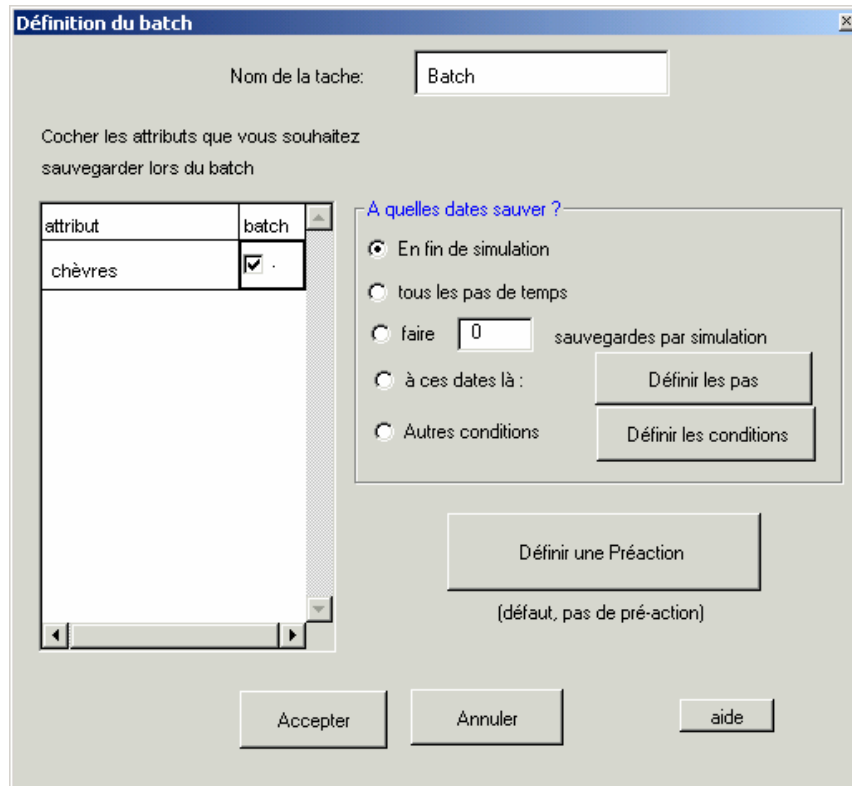


Création d'une tâche capable de trier les animats en fonction de leur nom et de les compter

Créez maintenant votre agent non situé 'résultat', qui doit avoir cette allure là, et ajoutez lui la tâche prédéfinie 'Batch':



L'interface de la tâche batch à l'allure ci-dessous. Vous cochez le (ou les) attributs à mémoriser puis les dates de sauvegarde. Nous nous contenterons ici des informations en fin de simulation.



Remarquez que nous n'avons pas encore donné la tâche 'CompterDesAnimats' à notre agent. Il est en effet inutile de lui faire exécuter cette tâche à chaque pas de temps puisque nous souhaitons juste compter les chèvres en fin de simulation. Nous allons associer cette tâche à la tâche batch via le bouton 'Définir une préaction'. Ce bouton appelle la liste des tâches disponibles, que vous connaissez bien maintenant. Sélectionnez 'CompterDesAnimats' et interfacez-là correctement, cela ne pose aucune difficulté.

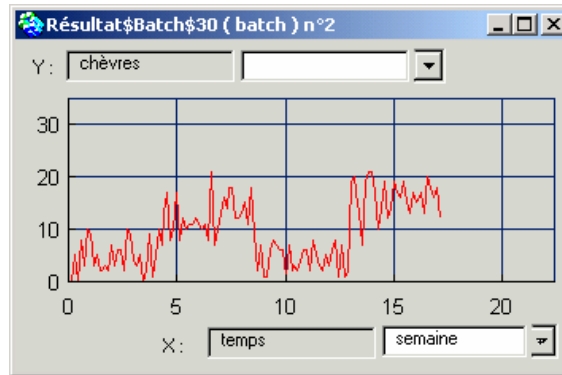
Il vous restes à préciser le nom du fichier de sortie, et vous êtes prêt à lancer vos expériences. Mais avant cela, vérifiez que le placement initial de vos agents est bon, et **surtout sauvegarder impérativement votre projet**. En effet, nous avons vu que le principe du Batch est de modifier les valeurs des paramètres des moules. Ces valeurs sont restaurées en fin de travail. Mais **si la procédure est interrompue**, accidentellement ou volontairement, **cette restauration n'est pas faite**. Il faut alors **immédiatement** recharger votre projet pour récupérer votre paramétrage initial.

Remarques :

- Le bouton 'Arrêter une simulation' ne fonctionne pas en mode Batch. Le moyen de stopper un Batch qui semble s'éterniser est de taper **Ctrl Y**. Rechargez immédiatement votre projet pour restaurer le paramétrage.
- **Vérifiez que le fichier de sortie n'est pas déjà ouvert par une autre application** (typiquement, le dernier Batch que vous avez ouvert sous Excel pour voir son allure !). Cela provoque une erreur de partage.
- La sauvegarde des résultats est automatiquement désactivée, mais pas la visualisation de l'espace. Vos options courantes de sauvegarde seront restaurées en fin de Batch. La visualisation étant souvent très coûteuse en temps, il est souvent préférable de fermer la fenêtre de visualisation. Le faire avant de lancer le Batch, après vous n'avez plus la main.
- Certaines combinaisons de paramètres peuvent entraîner une explosion du nombre d'animats. Le champs 'nombre max d'animats' est là pour l'éviter.

Voir l'aide en ligne pour plus de détails, et référez-vous à la partie 'Fichiers manipulés' de ce tutorial pour connaître la structure du fichier résultat (fichier .dat, qui se trouvera dans votre répertoire de projet).

Lancez votre Batch et allez boire un café. L'interface principale affiche l'avancement du travail. Une fois terminé, vous pouvez rapidement visualiser le résultat, en profitant du fait que la mémorisation du Batch se fait exactement de la même manière que la mémorisation des agents dans le temps, sauf que le temps est remplacé par le numéro de la simulation. Il est créé un 'agent batch' pour chaque tâche Batch (un agent peut donc en posséder plusieurs) et pour chaque pas de temps. Vous pouvez donc visualiser la population totale en sélectionnant l'agent batch, et utilisant l'outil chronique. Vous remarquerez au passage que les chroniques des autres agents sont vides, la mémorisation ayant été désactivée.



Résultats du Batch, avec quatre expériences, chacune renouvelée 30 fois. Le temps représente le numéro de l'expérience.

Typiquement, on dépouille ce genre de résultats avec une analyse de variance et des comparaisons multiples de moyennes. Ayez la curiosité d'aller inspecter votre fichier (qui porte automatiquement le suffixe .dat) dans votre répertoire de projet.

Batcher sur une liste

Et si ce n'est pas la simple valeur finale qui m'intéresse, mais la courbe de disparition dans le temps de mes chèvres ? Quitte à faire ensuite des comparaisons de courbes plutôt que des comparaisons de valeurs finales ? L'option 'sauvegarde à chaque pas de temps' de la tâche Batch n'est pas très pratique, car l'information sera dispersée dans une multitude de petits tableaux. Une meilleure idée est de faire mémoriser non pas un scalaire, mais une liste, à notre agent batch. Pour cela, revenir à votre agent résultat, et lui ajouter un attribut de type 'Liste', par exemple, 'chronique'.

Il faut ensuite mettre à jour cette chronique à chaque pas de temps. Pour cela, nous allons donner la tâche 'CompterDesAnimats' à notre agent 'Résultat' de manière classique, et faire pointer le résultat sur l'attribut 'chronique'. L'agent va donc compter les chèvres à chaque pas de temps, et chaque résultat ira s'ajouter à la liste. Il ne reste plus qu'à revenir dans la tâche 'Batch', et à cocher 'chronique' pour qu'il soit mémorisé. Laissons l'attribut 'chèvres' tranquille, ainsi que la post-action, même si nous obtiendrons la même information à la fin de chaque liste.

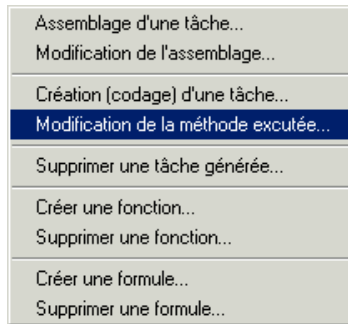
Il ne reste plus qu'à baisser le nombre de réplicats de l'expérience (menu contextuel 'ouvrir l'expérience' pour la rappeler, histoire d'aller plus vite), sauver le projet, vérifier que le fichier résultat n'est pas ouvert par une autre application, et lancer le nouveau Batch. En sortie, vous devez avoir un fichier .dat qui ressemble à celui-ci (exemple avec trois réplicats).

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	HISTORIQUE	22													
2															
3	NPARAMETRE	date	\$1_rayon d'a	\$2_rayon d'a	chèvres	chronique									
4	UNITE:	jour													
5	VALEURS:														
6		1.0	10.0	10.0	7.0	chronique#1									
7		2.0	10.0	10.0	8.0	chronique#2									
8		3.0	10.0	10.0	0.0	chronique#3									
9		4.0	10.0	3.0	6.0	chronique#4									
10		5.0	10.0	3.0	13.0	chronique#5									
11		6.0	10.0	3.0	19.0	chronique#6									
12		7.0	3.0	10.0	2.0	chronique#7									
13		8.0	3.0	10.0	3.0	chronique#8									
14		9.0	3.0	10.0	4.0	chronique#9									
15		10.0	3.0	3.0	18.0	chronique#10									
16		11.0	3.0	3.0	19.0	chronique#11									
17		12.0	3.0	3.0	17.0	chronique#12									
18	NPARAMETRES VALEUR:														
19	chronique#1	21.0	21.0	20.0	20.0	16.0	14.0	14.0	14.0	14.0	13.0	11.0	11.0	9.0	8.0
20	chronique#2	24.0	23.0	22.0	21.0	19.0	17.0	15.0	14.0	14.0	14.0	14.0	13.0	13.0	12.0
21	chronique#3	21.0	21.0	20.0	18.0	16.0	15.0	14.0	12.0	10.0	9.0	9.0	8.0	7.0	6.0
22	chronique#4	21.0	21.0	20.0	20.0	20.0	20.0	20.0	19.0	17.0	16.0	16.0	16.0	15.0	14.0
23	chronique#5	21.0	21.0	20.0	18.0	16.0	14.0	14.0	13.0	13.0	13.0	13.0	13.0	13.0	13.0
24	chronique#6	23.0	22.0	21.0	21.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0
25	chronique#7	22.0	22.0	21.0	21.0	19.0	19.0	16.0	15.0	13.0	13.0	12.0	10.0	10.0	10.0
26	chronique#8	23.0	21.0	21.0	20.0	19.0	18.0	17.0	16.0	15.0	15.0	13.0	12.0	12.0	11.0
27	chronique#9	22.0	22.0	22.0	21.0	19.0	19.0	18.0	17.0	17.0	15.0	14.0	14.0	14.0	13.0
28	chronique#10	22.0	22.0	21.0	18.0	18.0	18.0	18.0	18.0	18.0	18.0	18.0	18.0	18.0	18.0
29	chronique#11	23.0	23.0	22.0	22.0	21.0	20.0	20.0	20.0	20.0	19.0	19.0	19.0	19.0	19.0
30	chronique#12	22.0	22.0	21.0	20.0	20.0	20.0	18.0	18.0	18.0	18.0	18.0	17.0	17.0	17.0
31															
32															
33															
34															

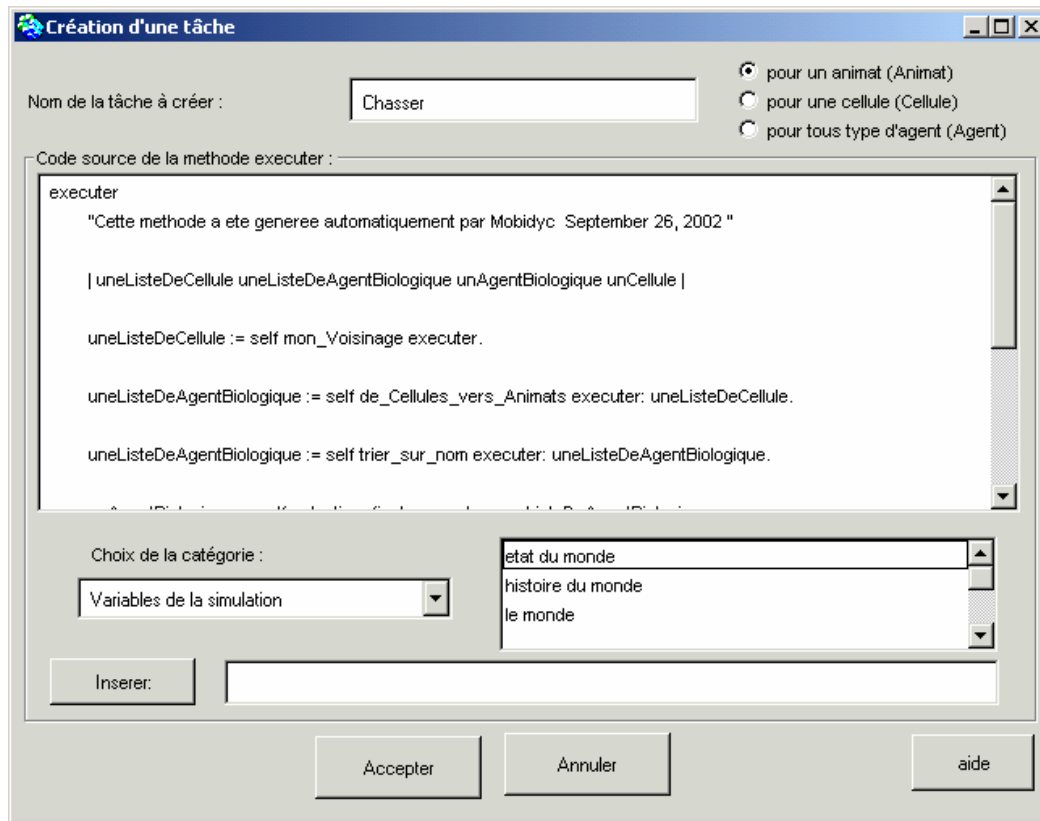
Les chroniques apparaissent en ligne (incomplètes ici). Pour plus de précisions sur la structure de ce fichier, voir la section 'Fichiers manipulés'.

Codage traditionnel d'une tâche

Même en version 'utilisateur' standard, Mobidyc, qui s'appuie sur le langage interprété Smalltalk, permet à un utilisateur de personnaliser ses tâches via un éditeur de code. Nous allons l'illustrer en reprenant notre tâche de poursuite, et en lui ajoutant 'en dur' la partie capture qui fait pour l'instant l'objet de la tâche spécifique 'TuerMaProie'. Ce faisant, nous partons d'un canevas existant (la tâche poursuite). Un gros avantage de Mobidyc est de pouvoir mixer assemblage de primitives et codage. Car il serait idiot de se passer des primitives sous prétexte d'écrire du code. Par contre, si nous souhaitons garder notre ancienne tâche 'Poursuivre' en l'état, il nous faut d'abord rappeler l'interface d'assemblage de tâche sur 'Poursuivre' et changer son nom en 'Chasser' par exemple. Ce faisant nous avons donc dupliqué 'Poursuivre'. Pourquoi ? A cause d'un petit bug qui fait que l'interpréteur de code aurait "oublié" (du fait du changement de nom, si on était directement parti de 'Poursuivre') que cette tâche fait également appel à des primitives. Nous pouvons maintenant sélectionner l'item correspondant à la modification d'une méthode d'exécution (toujours dans le menu 'tâches').



Et sélectionner 'Chasser' dans la liste qui s'affiche. Vous obtenez la fenêtre suivante :

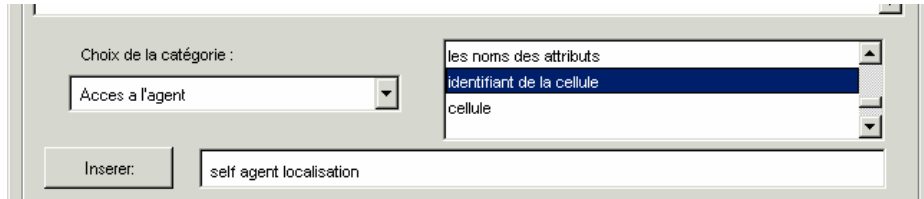


Donc, ne surtout pas changer le nom de votre tâche, la référence aux primitives serait perdue. Cette interface présente un éditeur de texte Smalltalk. Vous y trouvez "des commentaires", | les variables locales |, puis le texte de la méthode générée par Mobidyc pour 'Poursuite'. Vous reconnaîtrez sans peine j'espère (sinon on est mal partis !) la séquence d'appel des primitives. Allez en bas de la séquence, et repérez au passage notre malheureuse chèvre : c'est la variable 'unAgentBiologique' (le compilateur n'a pas encore totalement digéré l'appellation 'animat' que nous utilisons maintenant...).

Evidemment, vous ne connaissez sans doute rien à Smalltalk, et encore moins les entrailles de Mobidyc. Comment diable allons-nous faire pour vérifier si le loup à bien atteint la chèvre et si oui la manger ? Pour vous aider, certains raccourcis bien utiles ont été préprogrammés, listés par catégories. Une fois sélectionnés, ils peuvent être insérés dans le code au niveau du curseur (bouton 'Insérer').

Nous voulons d'abord regarder si loup et chèvre sont dans la même cellule. En vous promenant dans les catégories de code, vous remarquerez vite que l'on parle d'agent, et d'identifiant d'agent. En particulier, dans la catégorie 'Accès à

l'agent' vous avez l'item 'identifiant de la cellule'. On demande bêtement à l'agent le numéro de sa cellule. Accéder physiquement à la cellule est plus difficile, le code correspondant se trouve en dessous, dans 'cellule'.



Revenir à la fenêtre de codage, mettre le curseur à la fin et insérer ce bout de code. Une ligne Smalltalk, c'est très simple, ce n'est qu'une suite de messages qui s'adressent à un objet initial et qui se termine par un point. Et chaque message renvoie un objet, à qui s'applique le message qui suit. C'est d'une simplicité biblique. 'self agent localisation' ce lit donc comme suit : on part de 'self' qui est donc un objet. C'est ni plus ni moins la tâche 'Chasser' elle-même, que nous sommes en train d'écrire. 'agent' est un message 'Mobidyc' que toutes les tâches et primitives possèdent par défaut. Elle renvoie l'agent en cours d'exécution, donc le loup puisque c'est à lui que cette tâche est destinée. Puis nous avons le message 'localisation', qui est donc envoyé à l'agent actif, et qui nous donne l'identifiant de sa cellule. Mais nous pouvons également adresser ce message 'localisation' à l'objet chèvre, c'est-à-dire à la variable 'unAgentBiologique' que nous venons de repérer. Nous avons donc les identifiants des cellules de la chèvre et du loup. Pour faire un test d'égalité, il faut connaître un peu de Smalltalk, et bien s'imprégner de la logique Smalltalk : on envoie le message 'ifTrue:' à l'objet 'Condition' généré par l'égalité. Et entre crochets, se trouve la portion de code à exécuter. Il s'agit ici de tuer tout simplement la chèvre. Vous ne trouverez pas comment tuer un agent dans les listes proposées (elles sont à mettre à jour...). Il faut envoyer le message 'meurt' à l'animat concerné. Nous allons donc écrire (utilisez le menu contextuel pour avoir accès au copier/coller de l'éditeur) :

```
(self agent localisation = unAgentBiologique localisation) ifTrue: [unAgentBiologique meurt ]|
```

C'est presque terminé. Il reste à ajouter un petit contrôle (contrôle qui était automatiquement effectué par les primitives). Car que se passe-t'il si aucune chèvre n'est localisée ? Peut-on envoyer 'localisation' ou 'meurt' à rien (qui se dit 'nil' en Smalltalk) ? Non bien sûr. Il faut donc vérifier qu'une chèvre a bien été renvoyée par la primitive 'SelectioFinale'. Le code complet s'écrit :

```
unAgentBiologique isNil ifFalse: [
    (self agent localisation = unAgentBiologique localisation) ifTrue: [unAgentBiologique meurt ] ]|
```

Sauter une ligne dans une commande n'a aucune importance. C'est le point qui marque la fin.

Cette fois c'est terminé. Cliquer sur 'accept' dans le menu contextuel de l'éditeur pour que l'interpréteur vérifie votre syntaxe. Puis valider. Votre tâche 'Chasser' est prête. Installez-la dans votre loup, et supprimez (ou désactivez dans un premier temps) les tâches Poursuivre et Tuer. Sauvez votre projet et lancez une simulation. Cela devrait marcher...

Gestion de la synchronisation

Vous avez sans doute remarqué, dans les exemples de code, des références à 'agent' et à 'étatAgent', ainsi qu'à 'leMonde', et à 'étatDuMonde'. 'agent' et 'leMonde' font références à l'agent réel. Le préfixe 'état' lui, fait référence soit au réel, en mode séquentiel (asynchrone) d'appel des agents, soit à l'image sauvegardée à la fin du pas de temps précédent, si les agents sont en mode parallèle (synchrone). Votre code devrait toujours pouvoir s'adapter aux deux modes de synchronisation, on ne sait jamais. Le principe est simple. Toutes les affectations de valeurs (en gros tout ce qui se trouve le plus à gauche d'une ligne Smalltalk), doivent porter sur l'agent réel (il ne sert à rien de changer une image). Et tout ce qui est 'lecture d'information', doit porter soit sur le réel, soit sur l'image, en fonction de la synchronisation. Donc faire référence à 'état', le scheduler se chargeant de mettre ce qu'il faut derrière selon la synchronisation choisie.

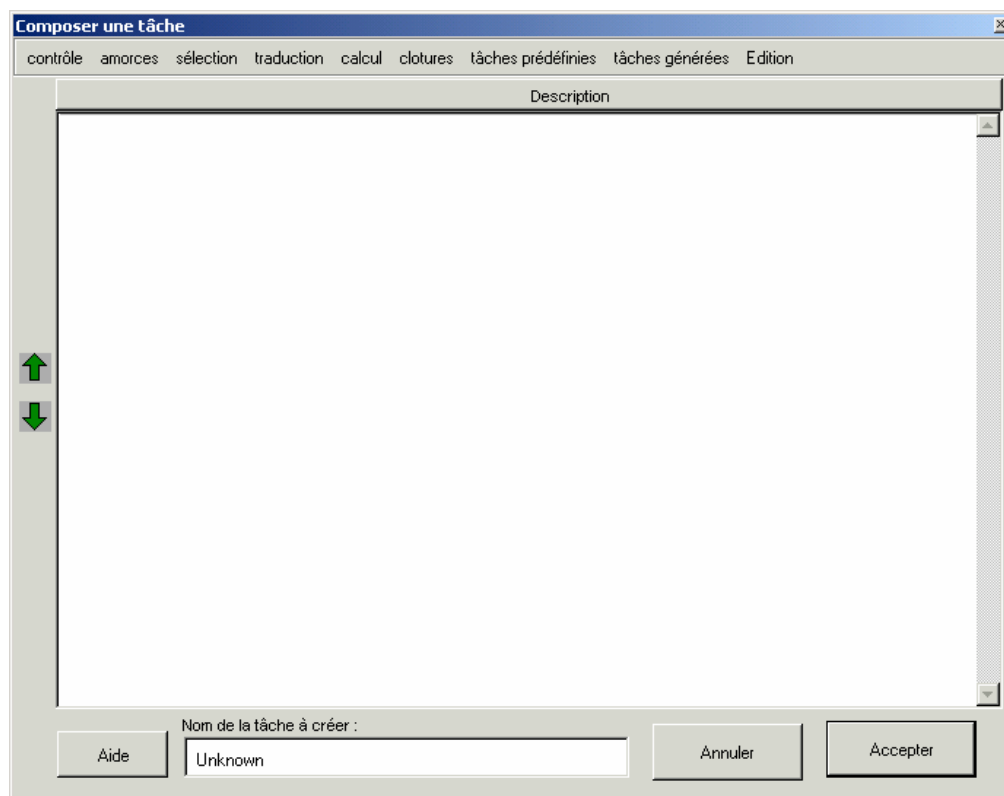
Mais dans notre exemple, nous n'avons rien à faire. Seule la chèvre doit être pointée via 'état, et c'est ce que font les primitives. 'unAgentBiologique' fait donc bien référence à l'image de la chèvre. Mais peut-on faire mourir une image ? Oui, Mobidyc, transmet l'information 'meurt', à l'agent réel, et le tue, tout en vérifiant qu'il n'a pas déjà été mangé (gestion par défaut de ce conflit de ressources). Mais vous pouvez forcer une image à vous rendre l'agent réel en lui envoyant le message 'agentCorrespondant'.

Et pour clore cette introduction au codage des tâches, une dernière remarque. Une tâche codée s'exécute entre 2 et 6 fois plus vite qu'une tâche assemblée. Il peut donc être intéressant de coder ses tâches alors que les primitives pourraient suffire. Il est alors conseillé de mettre au point son modèle avec les primitives, puis de le coder, en vérifiant qu'on obtient le même résultat. Ceci, parce que, en principe, les primitives sont plus sûres. C'est aussi leur grand intérêt, une erreur de programmation est très vite faite. Et cela permet en outre de débuser les bugs éventuels des primitives...

Construire une tâche à l'aide d'une procédure Myctalk

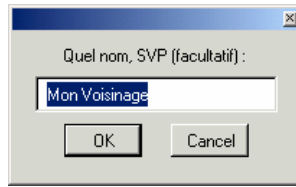
Nous allons maintenant utiliser une troisième manière de construire des tâches. Nous avons vu que l'enchaînement linéaire que nous avons utilisé n'est pas très souple. C'est pourquoi une nouvelle procédure a été développée, la composition Myctalk. Cette procédure étant encore en test, elle est fournie comme un patch, patch que vous pouvez télécharger à tout moment si vous constatez qu'il engendre des problèmes de fonctionnement de Mobidyc. Pour charger Myctalk, sélectionnez 'Options -> Charger Un patch'. La parcelle Dycalk est automatiquement chargée en même temps que Myctalk. Mobidyc est maintenant prêt à utiliser Myctalk. Nous allons une fois de plus donner sa tâche de chasse à notre loup.

Retournez à la fenêtre de composition des animats et sélectionnez le loup. Un nouvel item est apparu dans la liste des composants, l'item "procédure". Ajoutez-le au loup. L'interface de composition Myctalk s'ouvre :



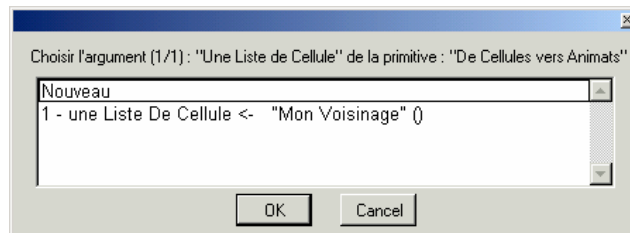
Interface de composition des procédures Myctalk

Vous reconnaîtrez sans peine dans la barre de menu les différents groupes de primitives. La procédure est très proche de celle que nous avons utilisé. Commencez par sélectionner la primitive de départ, 'MonVoisinage'. Une boîte de dialogue s'affiche, vous invitant à personnaliser le nom de la primitive si vous le souhaitez. Cela peut être utile pour améliorer la lisibilité de la séquence. Nous pouvons ici garder l'appellation par défaut :

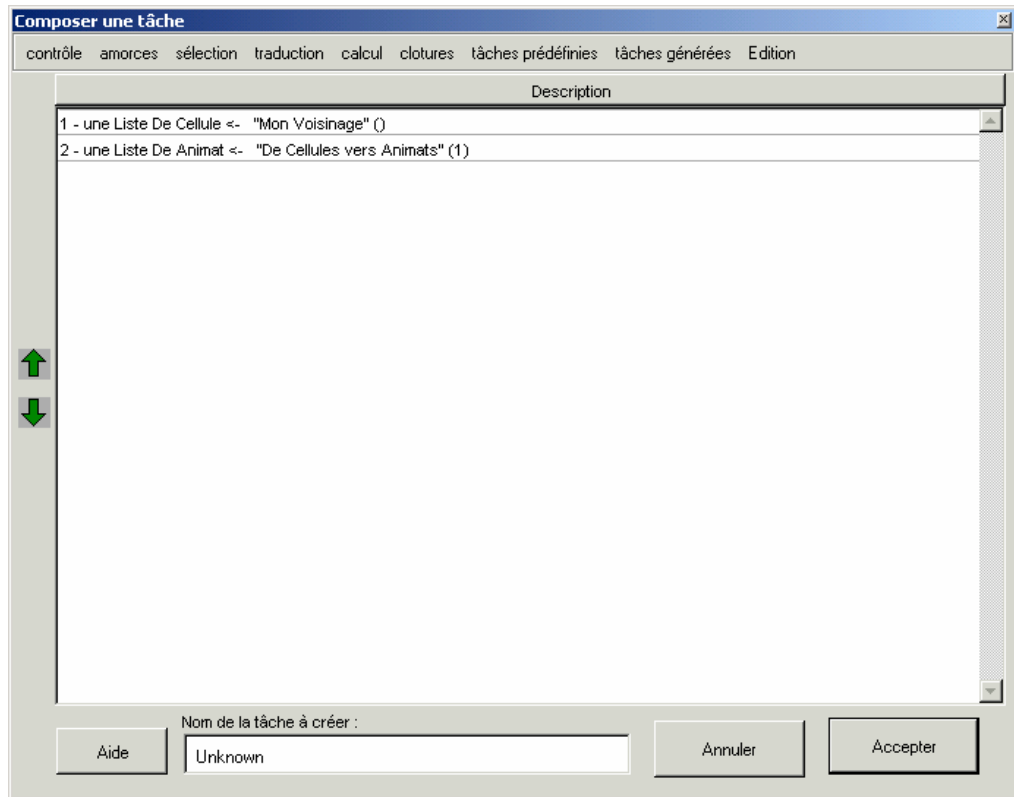


Validez. Vous obtenez ensuite la boîte de dialogue de paramétrage de la primitive. Gardez un champ de vision de 8 cellules. Au passage, vous noterez qu'avec Myctalk, on édite la séquence et on paramétrise les primitives lors d'une seule et même opération.

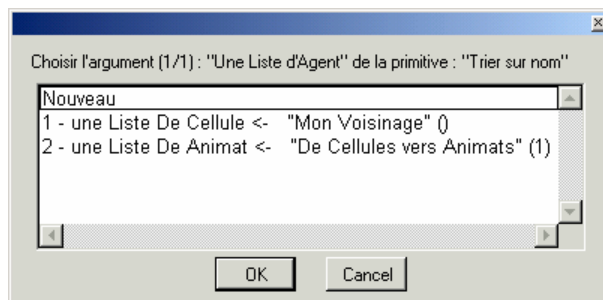
Sélectionnez la primitive suivante, 'DeCellulesVersAnimats', et gardez son nom par défaut. La procédure vous demande ensuite de sélectionner un argument d'entrée. C'est ici que réside la différence majeure avec la composition classique de Mobidyc : l'argument d'entrée n'est plus nécessairement l'argument de sortie de la primitive précédente.



Ici, pas de surprise, nous sélectionnons l'argument de sortie de 'Mon Voisinage', et nous obtenons :

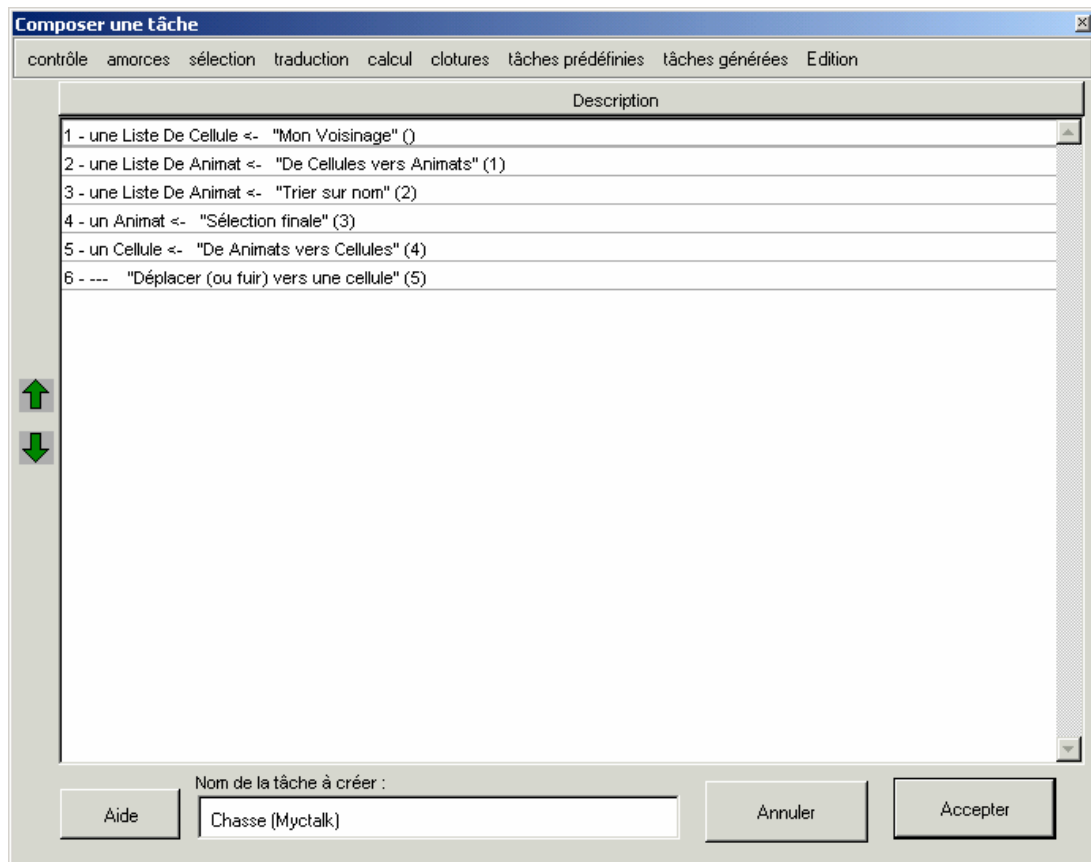


Nous en sommes maintenant à trier les chèvres. Sélectionnez la primitive 'Triez sur nom' et vous obtiendrez la liste suivante comme arguments possibles en entrée :

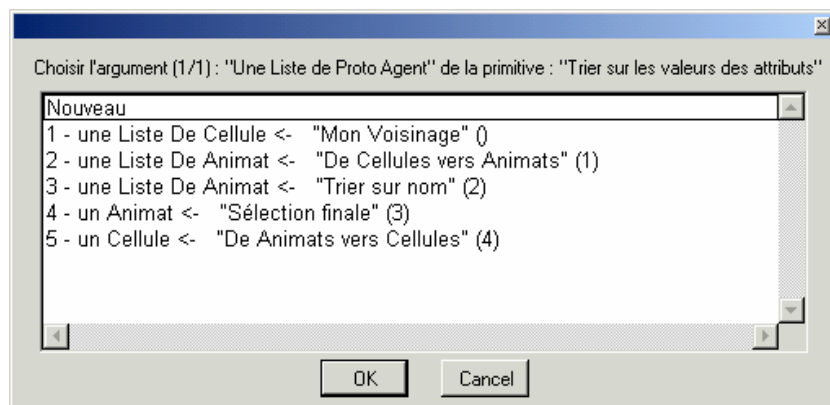


Vous remarquerez que cette liste est assez mal filtrée, la primitive 'TrierSurNom' attend en principe des animats. Sélectionnez l'argument (2) et paramétrez la primitive pour trier les chèvres.

Continuez avec la primitive 'SélectionFinale', qui prendra la sortie de 'TrierSurNom' comme argument d'entrée et que vous paramétrez pour sélectionner la chèvre la plus proche. Puis, comme nous l'avons fait précédemment, enchaînez avec la primitive 'DeAnimatsVersCellules' et la primitive 'Déplacer(ouFuir)VersUneCellule'. Comme tout à l'heure, le loup s'approche avec une vitesse de 1 cellule par pas de temps. Nous avons terminé la première étape, celle de la poursuite. N'oubliez pas de donner un nom à la procédure, par exemple 'Chasse (Myctalk)', vous devez obtenir la fenêtre suivante :



Nous devons maintenant regarder si le loup a bien atteint sa proie. Or Myctalk nous permet de reprendre la chèvre localisée à l'étape (4). Le test se fait comme tout à l'heure avec la primitive 'TrierSurDesValeursdAttributs'. Vous obtenez comme liste d'arguments possibles :



Sélectionnez la chèvre, i.e. l'argument 4, puis spécifiez la condition sur la localisation comme ci-dessous :

Tri sur attributs

loup (Chasse)

	Variable	Opérateur	Valeur Seuil	Unité
	mon_localisation	=	son_localisation	

☒ ET
 ☐ OU

☐ Sortir le propriétaire de la tâche si nécessaire (plus lent)

Nous avons presque terminé. Il ne reste plus qu'à ajouter la primitive de clôture 'Tuer'. Mais attention de tuer la bonne chèvre, c'est-à-dire l'argument 7, et non pas le 4, sans quoi notre loup sera capable de tuer la chèvre à distance!

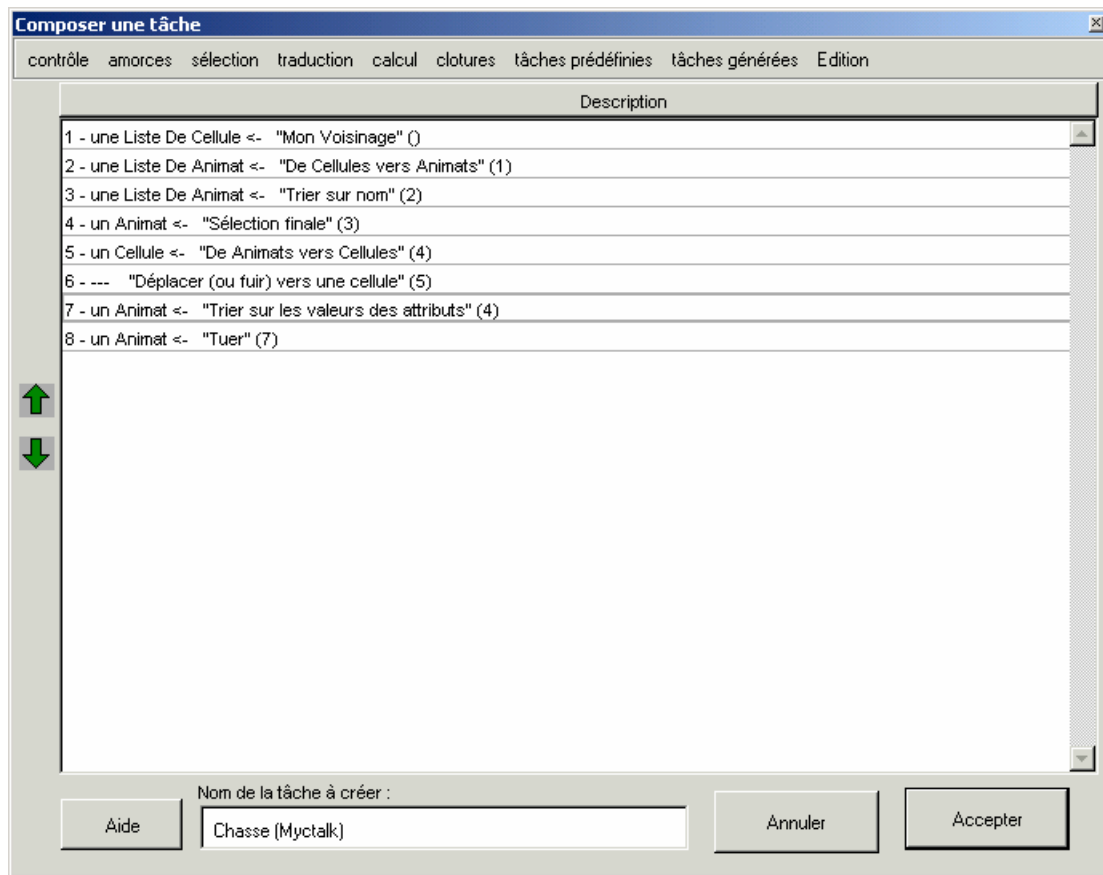
Choisir l'argument (1/1) : "Une Liste d'Animat" de la primitive : "Tuer"

Nouveau

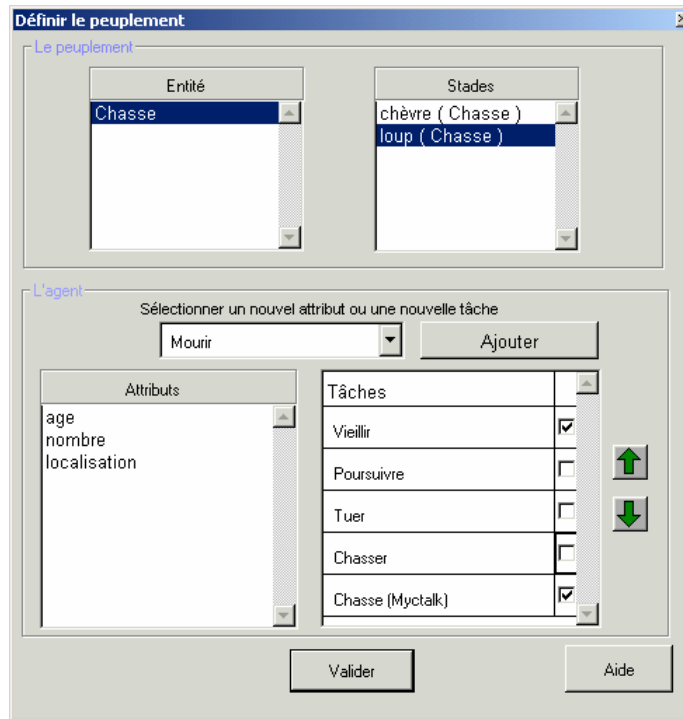
- 2 - une Liste De Animat <- "De Cellules vers Animats" (1)
- 3 - une Liste De Animat <- "Trier sur nom" (2)
- 4 - un Animat <- "Sélection finale" (3)
- 7 - un Animat <- "Trier sur les valeurs des attributs" (4)**

OK Cancel

Et voici la tâche terminée :



Vous avez donc maintenant 3 manières de chasser dans votre loup. Le couple Poursuivre/Tuer de la composition classique, la tâche Chasser que nous avons complétée par du codage en dur en partant de la tâche assemblée Poursuivre, et enfin la procédure Myctalk que nous venons de composer.



Quelques remarques sur l'usage de Myctalk :

- 1- Si vos modèles comprennent une composition Myctalk, il est impératif que le patch Myctalk soit chargé avant de charger votre projet. Si vous avez chargé Myctalk, ce dernier est automatiquement rechargé à chaque lancement de Mobidyc. Si vous souhaitez ne plus utiliser ce module, déchargez le, ainsi que son petit frère Dyctalk, au moyen de 'Option -> Décharger un Patch'. Ou plus radical, jetez le fichier mobidyc.plg du répertoire image : il contient la référence aux patches que vous avez chargé.
- 2- Utilisez le menu 'Edit' (en haut à droite) pour modifier les primitives (ou double clic pour modifier). Si vous retirez une primitive, attention de toujours respecter la cohérence de la séquence sous peine de générer une erreur. Dans la pratique, supprimez les primitives dans l'ordre inverse de celui où vous les avez ajoutées.
- 3- Le copier/coller ne fonctionne pas bien pour les procédures Myctalk.

Conclusion

Vous avez fini ce tutorial. Vous êtes maintenant en principe suffisamment armé pour vous lancer seul dans vos propres modèles. Merci de nous faire part des erreurs et des manquements à ce tutorial. Et si vous vous sentez l'âme pédagogue, n'hésitez pas à le compléter, ou encore à décrire vos propres exemples. Nous les mettrons en ligne sur le site, avec les fichiers projets correspondants.

Les fichiers utilisateur gérés par Mobidyc

Mobidyc utilise à la fois des fichiers binaires et des fichiers textuels. **Les fichiers binaires** contiennent des objets Smalltalk complets. Avantage : ils sont immédiatement disponibles, dans l'état où vous les aviez laissés. Inconvénient : ils doivent correspondre bit par bit à la version qui les a créés. Dans la pratique cela veut dire qu'ils ne seront vraisemblablement pas rechargeable dans une version future (ou antérieure) de Mobidyc, même si nous essayons au maximum de maintenir la compatibilité... **Les fichiers textes** sont plus robustes et peuvent être modifiés par l'utilisateur. Tous peuvent provenir d'un tableur par exemple, exportation au format texte, séparateur tabulation. Attention, la première ligne contient (en principe !) des informations sur le type de codage ASCII utilisé. **Ne pas le modifier manuellement.** Si votre fichier provient d'une autre plate-forme que la votre (par défaut, c'est l'incontournable codage PC...), et que votre éditeur le lit mal (accents, sauts de ligne), Mobidyc en principe sait le

lire. Chargez ce projet, puis sauvez le sous Mobidyc. Les fichiers doivent maintenant être dans un codage compatible avec votre machine. Cette ligne contient également la langue du projet. Mobidyc doit tourner dans la langue correspondant à celle du projet, il n'y a pas encore de traducteur pour passer un projet d'une langue à l'autre.

Les fichiers projet

- **Le fichier .mbp, fichier projet.** Il contient la référence à tous les autres fichiers et tout ce qui est lié à la gestion de vos simulations. Il sauvegarde en particulier l'ordonnanceur (scheduler), les préférences et les expériences simulatoires.

- **Le fichier .ppl, fichier des moules.** Ce fichier contient l'objet "gestionnaire d'Entités" du simulateur, c'est à dire principalement les moules de tous les agents.

- **Le fichier .esp, fichier de l'espace.** Ce fichier contient le descriptif de l'espace (taille de la grille, type de cellules etc.), puis la valeur initiale des attributs de toutes les cellules. Donc une ligne par cellule. Vous pouvez facilement l'éditer pour changer 'à la main' les caractéristiques de votre grille. Par exemple pour changer la taille de l'image, ou la topologie de la grille, Mobidyc ne permettant pas de le faire actuellement (pas de bouton 'modifier l'espace', et cela manque un peu...).

```
..\projets\TutorialSugar.esp      ISO-8859-1      1      fr
30
30
carre4
noTore
400
agent      Cellule ( AutresEntités )
localisation      bordure sucre  croit  sucreMax

1      1      0      1      0
2      1      0      1      0
3      1      0      1      0
4      1      0      1      0
5      1      0      1      0
6      1      0      1      0
7      1      1      1      1
8      1      1      1      1
9      1      1      1      1
...
```

Exemple de fichier .esp. La ligne vide sous le nom des attributs contient en principe les unités, mais aucune n'est définie dans cet exemple. Localisation fait référence au numéro (identifiant) de la cellule.

- **Le fichier .pdv, fichier des observatoires.** Ce fichier contient un "Dictionnaire des points de vue" qui rassemble l'ensemble de point de vues que vous avez pu définir sur les agents et leurs cellules permettant une visualisation sur les cartes 2D.

- **Le fichier .chg, fichier du chargement initial des animats.** Ce fichier ressemble beaucoup au fichier de l'espace. Il contient le descriptif initial des animats, Il sera créé autant d'agents que de ligne de valeurs. 1^{ère} ligne, rappel du nom du fichier et du codage ASCII. 2^{ème} ligne, mot réservé "agent", puis le nom complet de l'agent. 3^{ème} ligne, nom des attributs. 4^{ème} ligne unité (optionnelle) des attributs. Lignes suivantes, valeurs des attributs. On passe à l'agent suivant en reprenant le mot réservé "agent" et le nom de l'agent suivant Et ainsi de suite.

```
..\projets\TutorialSugar.chg      1      ISO-8859-1      1      fr
agent      agent ( Sugar )
age nombre localisation  sucre  respiration  rayonAction
jour unite
2      1      483      5      3      1
4      1      629      5      3      1
1      1      423      8      3      1
4      1      601      6      3      1
4      1      871      6      3      1
```

...

Deuxième exemple d'un fichier décrivant un peuplement avec deux types d'agents. Fichier entièrement construit sous tableur selon des données réelles, puis chargé dans Mobityc via le menu 'Peuplement -> charger depuis fichier'.

```
..\projets\jun-86 1 ISO-8859-1 1 fr
agent juvénile ( daphnie )
taille age
mm jour
0.416054 4.4270336
0.443163 4.60161821
0.446647 4.6242222
0.458449 4.70107892
0.469063 4.77057959
...
1.16996 10.4144229
1.18014 10.5168675
1.18984 10.6152195
agent adulte ( daphnie )
taille age maturité
mm jour jour
1.21663 10.890684 0
1.22767 11.0058782 1
1.23685 11.1024284 2
1.25486 11.2939034 0
...
```

Le fichier .pcl (et .pst), fichier de sauvegarde de vos Tâches. Ces fichiers sont deux, l'un contient le code binaire (pour gagner du temps au chargement du projet), l'autre le code source au format XML. Ils contiennent la sauvegarde des tâches formules et fonctions que vous avez pu générer au cours de vos projets. Ces fichiers, qui sont donc associés à chaque projet, remplacent fichier TachesUtilisateur.pcl (et .pst) des versions antérieures de Mobityc. Ce fichier est néanmoins conservé (dans le répertoire image) pour la compatibilité des anciens modèles. Mais dès que vous sauvez un ancien modèle, les fichiers nomProjet.pcl et nomProjet.pst seront automatiquement créés et complétés.

Ces sept fichiers (si tous présents) représentent l'intégralité de votre projet. Attention de bien tous les inclure (de préférence dans une archive zip), si vous souhaitez transférer votre modèle sur une autre machine.

Les fichiers d'importation ou d'exportation de données

- **Les fichiers de séries XY (.sxy).** Ce sont des fichiers qui décrivent une série abscisse / ordonnée, typiquement un scénario dans le temps. C'est également dans ce format que sont sauvegardées les chroniques lorsqu'on analyse les résultats. Exemple "température.sxy" qui donne les températures journalières du Léman sur plusieurs années consécutives. Un fichier scénario peut faire plusieurs milliers de ligne, il n'y a pas de limites a priori.

```
..\projets\température
temps température
jour degre C
0 6.4
1 6.8
2 6.5
3 6.1
4 6.2
5 6.3
6 6.5
...
```

La première ligne donne le nom du scénario, la deuxième le nom de l'abscisse (mot réservé "temps" pour un scénario dans le temps) et le nom de l'ordonnée (nom de l'attribut touché par le scénario). La troisième ligne donne les unités, puis viennent les valeurs. Les valeurs seront calculées pour tout X par interpolation ou bien par palier, Y conservant alors sa valeur pour tout X $[X_{i-1}, X_i]$. Dans le cas d'un scénario le pas de temps peut donc être indépendant du pas de temps de simulation, et même variable. Mais ceci impose que la première valeur de temps soit 0, afin de permettre

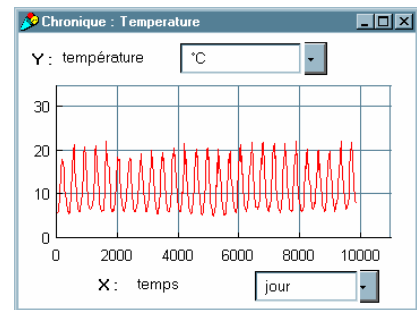
l'interpolation sur tous les temps possibles d'une simulation. Voir l'aide en ligne de la tâche scénario pour plus de détails.

Interfaces de saisie d'une série XY. Notez les options de bouclage, les options d'interpolation et le graphisme en ligne. L'éditeur (boîte centrale) permet de saisir les valeurs, mais le menu fichier permet d'importer un fichier texte qui répond au format ci-dessus et il est également possible de faire directement un copier coller des valeurs dans l'éditeur.

Interface de la tâche de scénario

Editeur de séries

Chronique des températures mensuelles



- **Les fichiers vecteurs de valeurs (.npa).** Ce sont des fichiers qui décrivent un vecteur de n valeurs (un NParametre dans Mobidyc). Typiquement une liste de valeurs décrivant la variabilité d'un attribut dans une boîte de chargement ou une boîte de gestion de l'héritage. Première ligne, nom du vecteur, puis unité et valeurs. Ici le début d'un vecteur de 1000 valeurs du paramètre de croissance d'un agent corégone selon une loi normale.

```
K
annee^-1
0.5
0.352
0.462
0.455
0.694
...
```

- **Les fichiers Batch (et HistoireDuMonde) (.dat).** Ce sont les fichiers mémoires d'une série d'expériences simulatoires, ou d'une simple simulation. Ces deux fichiers ont exactement la même structure, mais dans le cas des expérience simulatoires, l'attribut 'jour' fait référence au numéro de la simulation et des (pseudo)attributs supplémentaires apparaissent, précédés du signe \$. Ce sont les paramètres dont les valeurs ont été modifiées lors du Batch. Si plusieurs tâches Batch, ont été définies, ou si vous avez mémorisé les informations à différents pas de temps, vous obtiendrez plusieurs blocs de données séparées par la ligne 'HISTORIQUE AGENT: nomAgentBatch'. Le dernier chiffre (ici 22) donne le nombre de lignes du bloc. Les attributs à valeur standard (scalaire) sont dans le premier tableau, les attributs de type liste (ici chronique) sont donnés ensuite. Attention de ne pas dépasser 256 valeurs sous peine de ne pas pouvoir relire ces attributs liste (mais cela dépend du tableur avec lequel vous allez traiter ces données).

```
'HISTORIQUE AGENT: Résultat$Batch$30 ( batch ) NUMERO:2 22
```

```
NPARAMETRE: date $1_rayon d'action $2_rayon d'action chèvres chronique
UNITE: jour
VALEURS:
1.0 10.0 10.0 7.0 chronique#1
2.0 10.0 10.0 8.0 chronique#2
3.0 10.0 10.0 0.0 chronique#3
4.0 10.0 3.0 6.0 chronique#4
5.0 10.0 3.0 13.0 chronique#5
```

6.0	10.0	3.0	19.0	chronique#6							
7.0	3.0	10.0	2.0	chronique#7							
8.0	3.0	10.0	3.0	chronique#8							
9.0	3.0	10.0	4.0	chronique#9							
10.0	3.0	3.0	18.0	chronique#10							
11.0	3.0	3.0	19.0	chronique#11							
12.0	3.0	3.0	17.0	chronique#12							
NPARAMETRES VALEUR:											
chronique#1	21.0	21.0	20.0	20.0	16.0	14.0	14.0	14.0	14.0	13.0	11.0
chronique#2	24.0	23.0	22.0	21.0	19.0	17.0	15.0	14.0	14.0	14.0	14.0
chronique#3	21.0	21.0	20.0	18.0	16.0	15.0	14.0	12.0	10.0	9.0	9.0
chronique#4	21.0	21.0	20.0	20.0	20.0	20.0	20.0	19.0	17.0	16.0	16.0
chronique#5	21.0	21.0	20.0	18.0	16.0	14.0	14.0	13.0	13.0	13.0	13.0
chronique#6	23.0	22.0	21.0	21.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0
chronique#7	22.0	22.0	21.0	21.0	19.0	19.0	16.0	15.0	13.0	13.0	12.0
chronique#8	23.0	21.0	21.0	20.0	19.0	18.0	17.0	16.0	16.0	15.0	15.0
chronique#9	22.0	22.0	22.0	21.0	19.0	19.0	18.0	17.0	17.0	15.0	14.0
chronique#10	22.0	22.0	21.0	18.0	18.0	18.0	18.0	18.0	18.0	18.0	18.0
chronique#11	23.0	23.0	22.0	22.0	21.0	20.0	20.0	20.0	20.0	19.0	19.0
chronique#12	22.0	22.0	21.0	20.0	20.0	20.0	18.0	18.0	18.0	18.0	18.0

Remarque : La version utilisateur actuelle ne permet pas de sauver le fichier 'HistoireDuMonde' d'une simulation standard en mode interactif. Mais cette option peut être ajoutée si le besoin s'en fait sentir.

- **le fichier sauvegarde d'une simulation (.sim).** Binaire, optionnel. Si vous souhaitez sauvegarder l'ensemble d'une simulation pour une analyse ultérieure (parce que la refaire tourner serait trop coûteux en temps), utilisez la procédure 'Programme -> Mémoriser la simulation'. Vous pouvez ensuite la recharger à tout moment, vous vous trouverez exactement dans la même situation qu'à la fin de la simulation initiale. Attention, les fichiers .sim sont souvent gros, notamment si vous avez tout mémorisé à chaque pas de temps. Recharger un tel fichier peut nécessiter presque autant de temps que rejouer la simulation !

Les fichiers Mobidyc d'intérêt pour l'utilisateur

- **Le fichier d'initialisation mobidyc.loc** (répertoire image). Contient la langue et le répertoire de projet par défaut. Si vous le détruisez, vous obtiendrez la même fenêtre d'invite que lors du tout premier lancement.

- **Le fichier des mises à jour logiciel mobidyc.plg** (répertoire image). Contient la liste des ressources additionnelles que Mobidyc doit charger lors de son lancement. Initialement vide, il s'enrichit des nouveaux modules ou des modules de corrections d'erreurs qui doivent impérativement se trouver dans le répertoire 'plugin'. Vous pouvez le détruire, les ajouts ne seront plus chargés.

- **Les fichiers 'plugin' de mise à jour logiciel** (répertoire plugin). Ces fichiers sont de même type que les fichiers .pcl et .pst des tâches utilisateurs. Ils contiennent le code (et le source) des mises à jour logiciel ou des modules qui pourraient être développés. Ils se trouvent dans le répertoire 'plugin'. Nous vous engageons à charger et tester le module Myctalk / Dyctalk, qui permet une composition plus souple des primitives.

- **Le fichier mobidyc.cha** (répertoire image). Fichier ressources temporaires de Mobidyc, sans utilité pour l'utilisateur non programmeur. Vous pouvez le détruire, il sera automatiquement recréé.

- **Les fichiers messages, fichiers de traduction entre langues** (répertoire messages). Le répertoire 'messages' contiennent l'essentiel des dictionnaires de traduction, le reste étant rassemblé dans l'objet 'International' de Mobidyc. Ces fichiers sont organisés en répertoires, un répertoire par langue. Ne pas modifier les noms de ces répertoires. Comme la plupart des fichiers Smalltalk, ces fichiers sont doubles. Le fichier .idx contient la forme binaire (compilée), et le fichier .lbl la forme ASCII. Là encore, attention aux options de codage ASCII. Si vous souhaitez modifier ces fichiers pour modifier certains labels affichés par Mobidyc, c'est possible, mais il est préférable de nous consulter pour avoir la marche à suivre. Partez de la version rtf de ces fichiers, supposée plus portable. Et si vous êtes courageux et que vous souhaitez **traduire Mobidyc dans une autre langue**, c'est très possible et pas si long. Cela vient d'être fait pour l'Allemand. 'mobidyc.lbl' (fichier principal) contient les labels des interfaces, 'warning' les labels des boîtes d'alerte. **'aide' contient le fichier d'aide en ligne de Mobidyc.** Il n'existe qu'en français et en anglais. La

encore les traducteur / correcteurs sont bienvenus ! Ces fichiers ont le même nom dans les différentes langues.
Attention de ne pas les mélanger sous peine de surprises.