Supporting Online File for:

Movement, disease and patch exploitation in nesting agent groups

Wayne M. Getz, Richard Salter and Krti Tallam

November 19, 2019

1 Mathematical Details

1.1 General Framework

In this technical presentation all `TEXT` in this font refers to an actual Numerus Model Builder built-in function, but its designation is sufficiently clear so that the algorithm can be implemented in other programming languages.

1. The population initially consists of N_0 agents; but over time, due to births and deaths this number will change and be designated N_t.

2. The landscape is a pixelated $H_\gamma \times H_\delta$ hexagonal cellular array with a doughnut topology. The index pair γ, δ refers to a particular cell in this array. To emphasize that these indices refer to one entity (a particular cell), we tie them together using a hat notation: viz., $\widehat{\gamma, \delta}$. This is particularly useful when we use multiple subscripts to refer to cells in the neighborhood of the cell $\widehat{\gamma, \delta}$.

3. $M << N_0$ of the $H_\gamma \times H_\delta$ cells are identified as nest cells because they are the home cell of, at most, one uniquely identified colony.

4. Initially, every one of the N_0 agent is assigned to a nest cell at random. Thus, initially, there are an average of M/N_0 individuals per colony. The m^{th} of these colonies is referred to as $A_m,\ i = 1, ..., M$.

5. The model has a T_f-unit fast-time (within) cycle that begins with all individuals at home (in their nest cells) at times satisfying `REMAINDER(TIME()/`T_f`)=0,` while `INTEGER(TIME()/`T_f`)` is the number of cycles that have been completed.

6. At each tick of the clock except the last within each fast time cycle—i.e, for times $\tau = 1, ..., T_f - 1$ where
$$\tau = \texttt{REMAINDER(TIME()/}T_f\texttt{)}$$
an individual can stay in its current cell or move to any of a set of cells in set of cells called `BLOCK` (see Eq.1 below). However, when `REMAINDER(TIME()/`T_f`)`=0 the individual moves (jumping, if necessary) back in its home cell.

7. At each tick of the clock, except for the last within-cycle tick, when individuals return home, individuals move to increase their potential to extract resources from cells or claim new cells for their colony according to the priorities laid out in the movement rules section below.

8. Cells have a state vector of dimension $M + 2$, indexed according to the position of the cell in the array using the row index $\gamma = 1, ..., H_\gamma$ and column index $\delta = 1, ..., H_\delta$. The first M elements of each cell vector $\mathbf{c}_{\widehat{\gamma,\delta}}$ contain "fading memories" of the "occupation history" of individuals from each of the M colonies that have occupied this cell over time. In particular, if an individual from colony j occupies the $\gamma\delta$ cell at time t, then the value of the j^{th} element $c_{\widehat{\gamma,\delta},j}$ is increased by 1, but the value itself declines geometrically over time at a rate η_{M}. The $M+1^{th}$ element of $\mathbf{c}_{\widehat{\gamma,\delta}}$, denoted by $c_{\widehat{\gamma,\delta},M+1}$, is a real non-negative number that reflects this cells current resource value. Finally, the element $c_{\widehat{\gamma,\delta},M+2}$ contains the infectiousness value of the cell.

9. Individual agents have a state vector $\mathbf{a}_\alpha$, $\alpha = 1, ..., N_t$. The elements of $\mathbf{a}_\alpha$ are as follows:

   (a) $a_{\alpha,1}$ is a positive integer value that designates the type of individual with the number of possible types reflecting the number of species of interest, their sex, and any other category designator (e.g., if there are two sexes for each of two species, this number may be 1, 2, 3 or 4).

   (b) $a_{\alpha,2}$ is an positive integer value $m = 1, ..., M$ that identifies the colony membership of individual α.

   (c) $a_{\alpha,3}$: this is a positive integer value (or named state) that identifies the disease state of individual α. This can be relatively simple—such as disease states (with a numeric equivalents) S (0), E (1), I (2), V (3). In more detailed cases, we might want to increase the number of categories so the value $a_{\alpha,3}$ might reflect the number of time periods since the individual was infected. The probability of being in state E, I, or V (and even back to S) could then depend on the value of $a_{\alpha,3}$, conditioned on the value of other elements of $\mathbf{a}_\alpha$ if needs be.

   (d) $a_{\alpha,\ell}$, $\ell = 4, 5, ...$ may be continuous state variables that measure the relative size of individuals, their state of hunger, their state of emaciation when food is in short supply, and so on. Here we only use two such variables and refer to $a_{\alpha,4}$ interchangeably as the fitness or biomass variable and $a_{\alpha,5}$ as the stress variable.

1.2 Movement rules

In this section, and in NovaScript, an extension of the JavaScript programming language used to construct Numerus Model Builder, the local cell occupied by an individual is referred to as its `RING(0)` cell. Its six immediate neighbors are its `RING(1)` cells, while all cells that it can reach in a minimum of k moves without jumping over any cells is its `RING(`k`)`.

Recall that $\mathbf{c}_{\widehat{\gamma,\delta}}(t)$ represents the state of the cell occupied by individual α at time t, where $\widehat{\gamma,\delta} \equiv \texttt{RING}_\alpha(0)$ is one way of referring to the cell currently occupied by individual α. With this in mind, we use $\{\mathbf{c}_{\widehat{\gamma,\delta}}(t)|\widehat{\gamma,\delta} \in \texttt{RING}_\alpha(1)\}$ to refer to the set of state vectors of the 6 cells surrounding the current location of individual α. Before moving on, we stress that the designators $\widehat{\gamma,\delta}, \gamma = 1, ..., H_1, \delta = 1, ..., H_2$ are global locators of cells, while $\text{RING}_\alpha(r)$ is a local locator of all cells a distance r from the current cell occupied by individual α.

At the start of each time interval $[t, t+1]$, we first determine whether or not individual $\alpha = 1, ..., N$ remains in its current cell or moves to some other cell in the set $\texttt{BLOCK}_\alpha(r)$, where r is some pre-selected positive integer that determines the furthest distance any single move can be:

$$\texttt{BLOCK}_\alpha(r) = \{\text{All cells in } \cup_{i=0}^{r} \texttt{RING}_\alpha(i)\} \tag{1}$$

The total number of cells in this set is $K = 1+6(1+2+...+r) = 1+3r+3r^2$ (one center cell plus each hexagonal ring centered on this center cell ring has 6 cells more then the its immediate inscribed ring: i.e., 1+6+12+...+6(r-1)+6r cells). We use the notation $\widehat{\gamma,\delta}_i \equiv i$, $i = 1, ..., K$ to index a renumbering of cells in $\texttt{BLOCK}_\alpha(r)$. One renumbering scheme, for example, is to label the center cell $\widehat{\gamma,\delta}$, occupied by individual α, as $\widehat{\gamma,\delta} = \widehat{\gamma,\delta}_1$, the cell to its immediate west as $\gamma, \widehat{\delta+1} = \widehat{\gamma,\delta}_2$, then going counter-clockwise until the last of the cells in $\texttt{RING}_\alpha\texttt{(1)}$ has been labeled 7 (i.e., $\gamma - \widehat{1,\delta+1} = \widehat{\gamma,\delta}_7$ and moving on to the west most cell in $\texttt{RING}_\alpha\texttt{(2)}$ as 8 and so on until all cells in $\texttt{RING}_\alpha(r)$ have been labeled.

The decision for agent α belonging colony ℓ to stay in its current cell or move to some other cell in $\texttt{BLOCK}_\alpha(r)$ is determined using the agent movement rule below that relies on the following three measures:

Distance measure: *Closer is better*

For some decay parameter $\varepsilon > 0$, distance radius ν_i, and maximum distance radius K

$$d_i = e^{-\varepsilon\nu_i},\ i = 1, ..., K \tag{2}$$

represent the distance between the center cell and the i^{th} cell in $\texttt{BLOCK}_\alpha$ when this cell is in $\texttt{RING}_\alpha(\nu_i)$.

Comparative resource measure: *Prefer cells with more resources*

Under our renumbering $c_{\widehat{\gamma,\delta}_i,1}(t)$ is the resource value of the i^{th} cell in $\texttt{BLOCK}_\alpha$ at time. Use this to calculate for $i = 1, ..., K$,

$$\text{If} \quad c_{\widehat{\gamma,\delta}_i,M+1} > 0 \quad \text{then} \quad \rho_{\widehat{\gamma,\delta}_i} = \max\left\{1 - \frac{c_{\widehat{\gamma,\delta}_1,M+1}}{c_{\widehat{\gamma,\delta}_i,M+1}}, 0\right\} \quad \text{else } 0 \tag{3}$$

Comparative occupation-history measure: *Prefer cells less used by other colonies*

Under our renumbering $c_{\widehat{\gamma,\delta}_i,m+1}(t)$ is cell i's $\widehat{\gamma,\delta}$'s memory state with regard to the occupation history of agents from colonies $m = 1, ..., M$ in this cell. For cells $i = 2, ..., K$, this quantity is based on a comparison of the largest occupation history value (in cell i) of $c_{\widehat{\gamma,\delta}_i,m}(t)$ for all $m = 1, ..., M$ in comparison with the occupation history value of agent α in cell i. Recalling that agent α is from colony ℓ) then, for $i = 1, ..., K$,

$$\text{If} \;\; c_{\widehat{\gamma,\delta}_i,\ell+1} > 0 \;\; \text{then} \;\; \omega_{\widehat{\gamma,\delta}_i} = \max\left\{1 - \frac{\max\limits_{m=1,...,M,\; m\neq\ell}\left\{c_{\widehat{\gamma,\delta}_i,m}(t)\right\}}{c_{\widehat{\gamma,\delta}_i,\ell}(t)}, 0\right\} \;\; \text{else } 0 \tag{4}$$

Agent movement rule: *Stochastic trade-off*

For weighting parameters $0 \leq \phi_1 \leq 0$ and $0 \leq \phi_2 \leq 1 - \phi_1$—which represent trade-offs among the relative values of movement distance, resources of cells, and past use of cells—define the cell attractiveness values p_{i}, $i = 1, ..., K$ as

$$p_{\alpha,i} = \frac{(1-\phi_1-\phi_2)d_i + \phi_1\rho_{\widehat{\gamma,\delta}_{\text{I}}} + \phi_2\omega_{\alpha,i}}{\sum_{j=1}^{K}\left((1-\phi_1-\phi_2)d_i + \phi_1\rho_{\widehat{\gamma,\delta}_{\text{I}}} + \phi_2\omega_{\alpha,i}\right)} \tag{5}$$

Now choose among the cells by applying the single drawing multinomial distribution: MULTINOMIAL[1;$p_{\alpha,1}, ..., p_{\alpha,K}$]

1.3 Ecological Dynamics

First, we determine for all cells which individuals occupying each cell stay or move, and then, we compute changes in the resource states of the cells, as well as the fitness that individuals gain or lose from consuming resources due to movement activity.

Movement *Who moves?*

Assume that there are $n_{\widehat{\gamma,\delta}} \geq 1$ in cell $\widehat{\gamma,\delta}$ individuals at times t that are not divisible by T_f. Then for each of these individuals, a multinomial drawing based on Eq. 5 is implemented to see if an individual stays or moves; and, if it moves, to which cell it moves.

STAY dynamics *Cell resources and individual fitness updating*

If $0 < y(t-1) \leq n_{\widehat{\gamma,\delta}}$ individuals stay in cell $\widehat{\gamma,\delta}$ over the time interval$[t-1, t]$ (i.e., these individuals were in cell $\widehat{\gamma,\delta}$ at time $t-1$ and did not move out at time t), then the individual and cell states are updated using the following quantity defined in terms of the maximum resource extraction rate parameter $u > 0$, extraction-efficiency parameter $h \geq 0$, and competition parameter $q > 0$ (see [1, 2], taking into account that $y(t-1)-1$ is the number of individuals in each cell beyond a focal individual for which the extraction expression is being formulated):

$$X(t-1) = \min\left\{\frac{c_{\widehat{\gamma,\delta},M+1}(t-1)}{y(t-1)}, \frac{uc_{\widehat{\gamma,\delta},M+1}(t-1)}{h\big(1+q(y(t-1)-1)\big) + c_{\widehat{\gamma,\delta},1}(t-1)}\right\} \tag{6}$$

If $y = 0$ then $X(t) = 0$. Specifically, the individuals' fitness states $a_{\alpha,4}(t-1)$, $\alpha = 1, ..., N$, are updated for a maximum fitness value $a_{\alpha,4}^{\max}$, a fitness decay factor ψ, and a biomass conversion parameter $0 < \kappa < 1$ using

$$a_{\alpha,4}(t) = e^{-\psi}a_{\alpha,4}(t-1) + \kappa X(t-1)\max\left\{0, \left(1 - \frac{a_{\alpha,4}(t-1)}{a_{\alpha,4}^{\max}}\right)\right\} \tag{7}$$

and the cell $\widehat{\gamma,\delta}$ resource state for resource-intrinsic-growth-rate parameter $r > 0$, reservoir parameter $g \geq 0$ and saturation parameter $s > 0$ is updated, by first defining

$$c^*_{\widehat{\gamma,\delta},M+1}(t) = c_{\widehat{\gamma,\delta},M+1}(t-1) - y(t-1)X(t-1) \tag{8}$$

and then using the equation

$$c_{\widehat{\gamma,\delta},M+1}(t) = c^*_{\widehat{\gamma,\delta},1}(t-1) + r\left(1 - \frac{c^*_{\widehat{\gamma,\delta},M+1}(t-1)}{s}\right)\left(c^*_{\widehat{\gamma,\delta},M+1}(t-1) + g\right) \tag{9}$$

MOVE dynamics *Individual fitness updating*

If an individual α moves out of a cell at time $t-1$, then the individual's fitness (or biomass) decays at a rate $\nu\psi > 0$, where ψ is the decay rate factor introduced in the STAY dynamics section and $\nu > 0$ is the fitness cost of moving relative to not moving; i.e.,

$$a_{\alpha,4}(t) = e^{-\nu\psi}a_{\alpha,4}(t-1) \tag{10}$$

Stress dynamics *Individual not meeting daily requirements*

We model stress dynamics using a variables $a_{\alpha,5}(t)$ for the $a = 1, ..., N$ agents. We note that $a_{\alpha,5}(t) \geq 0$, such that $a_{\alpha,5}(t) = 0$ implies the individual is unstressed with stress increasing as $a_{\alpha,5}(t)$ increases. We assume that without additional stress input, the $a_{\alpha,5}(t)$ relaxes back to zero at a rate c_z. However, if an individual is subject to an additional stress $z_\alpha(t)$ over the interval $[t, t+1]$ then its new stress level at time t will be

$$a_{\alpha,5}(t) = \max\{0, (1-c_z)a_{\alpha,5}(t-1) + z_\alpha(t-1)\} \tag{11}$$

To calculate the amount of stress an individual is subject to in each time interval, we assume that when it moves (i.e. does not take in any resources) then it has a resource deficit of X^*_{type} (i.e., a set of input parameter values subscripted by type, where "type" indicates that these deficits may vary by species, sex, or age, etc.). In this case we assume, dropping the type designation (since we make this parameter the same for both sexes, all disease states, and we have no incorporated age structure), that

$$\text{MOVES}: \qquad z_\alpha(t-1) = c_m X^* \tag{12}$$

where $c_m > 0$ is a conversion constant that depends on the units used to measure resources. On the other hand, if an individual gets to stay in the cell over time $[t, t+1]$, and there are y individuals (including itself) that stay to exploit resources in the cell during this time interval, the amount of resource that the individual extracts, from Eq. 6, is $X(t-1)/y$. We now assume that the extent to which this value falls short of the resource deficit value X^* is the amount of stress incurred over $[t, t+1]$ multiplied by the conversion constant: i.e.,

$$\text{STAYS}: \qquad z_\alpha(t-1) = \max\left\{0, c_m\left(X^* - X(t-1)\right)\right\} \tag{13}$$

Note the the above covers all individuals that stay whether or not they reproduce.

Cell memory dynamics *Which colony dominates particular cells*

The quantities $c_{\widehat{\gamma,\delta},m}(t)$, $m = 1, ..., M$, which contain a fading memory of how many individuals from colony m have visited the cell over time, are updated using $y_m(t)$ to represent the number of individuals from colony m that are in cell $\widehat{\gamma,\delta}$ at time t and a memory fade rate constant $\eta_{\text{M}} > 0$. The updating equation for $m = 1, ..., M$ is:

$$c_{\widehat{\gamma,\delta},m}(t) = y_m(t-1) + e^{-\eta_{\text{M}}}c_{\widehat{\gamma,\delta},m}(t-1) \tag{14}$$

1.4 Epidemiology and Deaths

The epidemiological component is essentially an S (susceptible), E (exposed but not yet infectious), I (infectious), V (recovered with immunity) and D (dead) model [3]. We use the value of $a_{\alpha,2}$ to denote this state and use the following rules to compute whether or not individuals in states S, E, I, and V respectively transition to states E, I, V, S and D in terms

of "transition rate parameters" $\gamma_{\rm E} > 0$, $\gamma_{\rm I} > 0$, $\gamma_{\rm V} > 0$, $\gamma_{\rm S} > 0$ and $\gamma_{\rm D} \geq 0$. We assume a basic death rate over the interval $[t, t+1)$ $\mu_0 > 0$ that increases with stress at rate that is proportional to the stress level $a_{\alpha,5}(t)$, with a constant of proportionality given by $c_s > 0$. In this case:

$$\gamma_{\rm D}(t) = \mu_0\big(1 + c_s a_{\alpha,5}(t)\big) \tag{15}$$

for all individuals in diseases states S, E, I and V. For individuals in disease state I, we assume an additional disease induce mortality rate $\mu_{\rm I} c_{\widehat{\gamma,\delta},1}(t)$ (where for relatively high mortality diseases $\mu_{\rm I} >> \mu_0$) so that for individuals in I, where we add the index I to distinguish it from the rate we have in Eq. 15

$$\gamma_{\rm ID}(t) = (\mu_0 + \mu_{\rm I})\big(1 + c_s a_{\alpha,5}(t)\big) \tag{16}$$

Cell Infectiousness *Pathogen density in each cell*

If $y_{\rm I}(t)$ is the number of infected individuals in cell $\widehat{\gamma,\delta}$ at time t and $\eta_{\rm I} \geq 0$ is an infectiousness fade rate ($\eta_{\rm I}$ close to 0 implies that indirect transmission of pathogens is of considerable importance while $\eta_{\rm I} \to \infty$ implies that direct contact becomes increasingly important for transmission (for a discussion on direct versus indirect transmission, see [4]), then:

$$c_{\widehat{\gamma,\delta},m+2}(t+1) = y_{\rm I}(t) + e^{-\eta_{\rm I}} c_{\widehat{\gamma,\delta},m+2}(t) \tag{17}$$

Transmission or death *Who gets infected?*

A susceptible individual that spends the period $[t, t+1)$ in cell $\widehat{\gamma,\delta}$ will have probability of getting infected at a rate proportional $c_{\widehat{\gamma,\delta},m+2}(t)$, where the constant of proportionality is $\beta > 0$. Thus the per-capita rate or force of infection is:

$$\gamma_{\rm S} = \beta c_{\widehat{\gamma,\delta},m+2}(t) \tag{18}$$

If susceptible, individuals also have a probability of dying over the same time interval at a rate (or force of risk) that is proportional to their stress level $a_{\alpha,5}(t)$ then, in essences, we are assuming that the transition rate out of then using a competing rates approach [3] we obtain for $\gamma_{\rm S}$ and $\gamma_{\rm D}$ given by Eqs. 18 and 15

$$\begin{aligned} \text{Prob. individual makes transition S} \to \text{E} &= \frac{\gamma_{\rm S}\,(1 - e^{-\gamma_{\rm S}-\gamma_{\rm D}})}{\gamma_{\rm S} + \gamma_{\rm D}} \\ \text{Prob. individual dies: i.e., S} \to \text{D} &= \frac{\gamma_{\rm D}\,(1 - e^{-\gamma_{\rm S}-\gamma_{\rm D}})}{\gamma_{\rm S} + \gamma_{\rm D}} \\ \text{Prob. individual not infected: i.e., S} \to \text{S} &= e^{-\gamma_{\rm S}-\gamma_{\rm D}} \end{aligned} \tag{19}$$

Disease transitions *what is an individuals next disease state?*

The transitions from states E to I or D and V back to S or D follows the same formulations of Eqs. 19, except we replace $\gamma_{\rm S}$ with $\gamma_{\rm E}$ and $\gamma_{\rm V}$ in each of the two cases respectively. The transition from state I to V or D, also follows the same formulation of Eqs. 19 except now we replace $\gamma_{\rm S}$ with some $\gamma_{\rm E} > 0$, and we replace $\gamma_{\rm D}$ with $\gamma_{\rm ID} =$ defined by Eq. 16

1.5 Colony Switching

The number of individuals in each colony essentially constitutes a bounded stochastic walk on the non-negative integers. The upper bound is imposed by feedbacks that relate to the effects of increasing death rates and decreasing birth rates through reductions in fitness and increases in stress levels due to competition for limited resources. Such walks are known to ultimately hit zero [5, 6], which is an absorbing state for the number of individuals in each colony, unless colonies themselves can be fueled through emigration.

There are several ways to deal with emigration. A relatively simple approach is for an individual to make a decision to join a colony, when it enters a colony cell and conditions are favorable compared to the individual's current state. For example, the individual on entering a cell can compare its own current fitness and stress levels with those of the individuals residing in the cell being entered.

Here is a very simple approach that we employ in our simulation, based on the idea that if a colony consists of a very few individuals, then competition for resources is less intense in the neighborhood of this colony. In rule articulated below, $\#m$ refers to the number of individuals in colony m:

$$\begin{array}{l}\underline{\text{Rule:}}\ \text{An individual from colony } m', \text{ currently in the nest cell of colony } m,\\ \text{will switch to colony } m \text{ with probability } p_{\text{switch}} \text{ whenever } \#m'/\#m < \rho_{\text{crit}}\end{array} \tag{20}$$

1.6 Reproduction

All individuals in colonies that do not move have an opportunity to reproduce at that start of each new fast cycle (i.e., when $t \bmod T_f = 0$). In particular, any individual who is in disease state S or V, and whose fitness value exceeds $b_{\min}$ will not move in the next time step, but will instead give rise to a new individual. The fitness/biomass value of this reproducing individual will drop (parent) and a newborn will be created with updated fitnesses given by

$$\begin{array}{l}\text{If} \quad \left(t \bmod T_f = 0 \ \text{ and } \ a_{\alpha,4}(t) > b_{\min}\right)\\ \text{then} \left\{\begin{array}{cc} a_{\alpha,4}(t+1) = \phi_{\text{parent}} a_{\alpha,4}(t) & \text{parent fitness} \\ a_{\alpha',4}(t+1) = \phi_{\text{newborn}} a_{\alpha,4}(t) & \text{newborn fitness}\end{array}\right\}\end{array} \tag{21}$$

Note that the newly reproduced individual, α', will belong to the same colony. In addition, we must have $\phi_{\text{parent}} + \phi_{\text{newborn}} < 1$, where the shortfall of this sum from 1 is a cost of reproduction.

1.7 Tables, Parameters and Initial Conditions

Tables referred to in the main text are provide here. A list of all the parameters appearing in the model, the names of these parameters in our NMB code, their mathematical symbols and the equations in which they first appear, their baseline values, and other values used in non-baseline runs are given in Table 1. Values used to define the size of the simulations (size of landscape, length of runs) and the initial values used in our various simulations are given in Table 2. The results of simulations used to assess the effects of colony switching behavior on persistence of populations are given in Table 3

Table 1: Model Parameter Values

Parameter†	**Symbol**	**Eq.**	**Values used**	
			Baseline	**Other**
max_rad	K	2	2	fixed
dist_dk	ε	2	1/3	0.0, 1.0
weight_res	ϕ_1	5	1/3	0.0, 1.0
weight_occ	ϕ_2	5	1/3	0.0, 1.0
max_res_ex_def	u	6	5.0	fixed
eff_ex	h	6	5.0	fixed
comp_ex	q	6	1.0	0.0, 10.0
biomass_conv	κ	7	0.2	fixed
fit_max	$a_{\alpha,4}^{\max}$	7	1.0	fixed
fit_decay	ψ	7	0.1	fixed
res_grow	r	9	0.2	fixed
res_sat	s	9	20	fixed
res_reservoir	g	9	0.1	fixed
move_cost	ν	10	2	fixed
stress_decay	c_z	11	0.05	fixed
res_def	X^*	12	0.5	fixed
stress_scale	c_m	12	1.0	fixed
mem_fade	η_{M}	14	0.3	fixed
stress_death	c_s	15	1.0	fixed
nat_death	μ_0	15	0.01	fixed
disease_death	μ_{I}	16	0	0.03, 0.1, 0.2
infect_fade	η_{I}	17	0.5	fixed
disease_trans	β	19	0.0	0.01, 0.02, 0.03, 0.05
p_switch	p_{switch}	20	0.0	0.8
rho_crit	ρ_{crit}	20	0.3	fixed
p_rep	$b_{\min}$	21	0.8	fixed
parent_fit	ϕ_{parent}	21	0.65	fixed
newborn_fit	ϕ_{newborn}	21	0.35	fixed
latent_inv	γ_{E}	‡	0.5	fixed
infect_inv	γ_{I}	‡	0.2	fixed
immunelose_inv	γ_{V}	‡	0.02	fixed

†max:=maximum, rad:=radius, dist:=distance, dk:=decay, occ:=occupy, res:=resource, ex:=extract, eff:=efficiency, comp:=competition, sat:=saturation, fit:=fitness (same as biomass in our model), def:=deficit, mem:=memory, nat:=natural, trans:=transmission, crit:=critical, rep:=reproduction, inv:=inverse

‡ See subsection on **Disease transmission**

Table 2: Simulation Parameter Values and Initial Conditions

Parameter or Variable	**Symbols and Values**
Landscape dimension	$H_\gamma \times H_\delta = 18 \times 18$
Simulation time	$T = 2000$
colony return time	$T_f = 10$
Number of colonies	$M = 9$
Initial cell memory vector	$c_{\widehat{\gamma,\delta},j}(0) = 0, \;\; j = 1, ..., M^{\#}$
Initial cell resources	$c_{\widehat{\gamma,\delta},M+1}(0) = \mathrm{U}(1,3)^{\dagger\#}$
Initial cell infectiousness	$c_{\widehat{\gamma,\delta},M+2}(0) = 0^{\#}$
Initial number of agents	$N_0 = 350$
Initial agent sex	$a_{\alpha,1}(0)$ is "Female" with probability $p_f = 0.6^{\ddagger}$
Initial agent membership	$a_{\alpha,2}(0)$ randomly assigned to one of M colonies‡
Initial agent disease state	$a_{\alpha,3}(0)$ is S except for one random E‡
Initial agent biomass/fitness	$a_{\alpha,4}(0) = \mathrm{U}(0.15, 0.25)^{\dagger\ddagger}$
Initial agent stress level	$a_{\alpha,5}(0) = \mathrm{U}(0.15, 0.25)^{\dagger\ddagger}$

†U(a,b) is a random drawing from the uniform distribution on $[a, b]$
$^{\#}\gamma = 1, ..., 18$, $\delta = 1, ..., 18$
$^{\ddagger}\alpha = 1, ..., N_0$

Table 3: Times to population extinct for the base line set of parameters (Table 1 except $\mu = 0.02$)

Colony Switching	**Times to extinction for 15 runs**	**Average**†
$p_{\text{switch}} = 0.0$	517, 523, 541, 818, 799, 437, 928, 437, 563, 659, 812, 475, 323, 467, 498	425.0
$p_{\text{switch}} = 0.8$	310, 337, 449, 333, 454, 620, 491, 413, 701, 328, 275, 587, 364, 316, 397	586.5

†The difference is significant at the $p < 0.01$ level: two-sided Mann-Whitney U test.

2 Numerus Model Builder Implementation

Here we provide some of the key details not included in previous publications [1,3] regarding the construction, using the Numerus Model Builder platform, of the agent-based model presented in the main text.

Numerus Model Builder (NMB) employs *NovaScript*, which is embedded in, and extends, Javascript. The NovaScript interpreter is an extension of the ECMA 1.7 Javascript standard. The full range of the Javascript language is available to the NovaScript programmer for specifying the computations required by his or her simulation. It includes facilities for specifying simulations visually, using stock and flow diagrams (familiar to programmers of Stella and Vensim) and other visual tools unique to NMB. Simulations are scoped out using such visual schematic diagrams with equation and other procedural details entered into appropriate windows [1], or textually, via NovaScript. We will refer to the former as a diagram specification; the latter will be referred to as a code specification.

NMB is highly modularized due to: 1) the creation of dynamic agent "Capsules" that interact with their environments through input/output interfaces (more below) and 2) allowing these agents to move over a heterogeneous resource landscape. Movement can either be on a Cartesian plane or from one cell to another on a rectangular or hexagonal array of specifiable dimension [1]. Our model moves agents over hexagonal arrays of variable and dynamic resource cells and was designed to address questions regarding agent and colony populations dynamics under different movement and disease scenarios.

Simulations are built out of objects, called Components. Among the component types are **Primitive Components**, **Simulators**, **Displays**, and **Controls**. The primitive components consist of Stocks, Variables, Sequences, Flows, and Terms. Stocks, Variables and Sequences are used to represent the state of a running simulation. The Stock component may be attached to one or more Flows. Each Flow specifies some change in the current value of the Stock. A Variable is similar, only it is not attached to any Flow; rather its change is determined by a derivative contained in the *Prime* field. Sequences are similar to variables, except that they are discrete, and change is indicated by a *Next* field, directly determining the value of the sequence in the next time interval. Flows and Terms both have *Exp* fields containing their expressions. As mentioned above, Flows affect the value on the Stocks to which they are attached. A Term is used to compute a value using current State values (in Stella, what Numerus Model Builder calls a Term is referred to as a Converter.)

This model is formulated as a set of agents moving over a cellular array. Therefore, as described in our presentation, we require 3 Capsule types: the Cell, Agent, and sirTerrM. (Refer to Sections 3.1, 3.2, and 4 for further detail on each Capsule type.)

2.1 Architecture

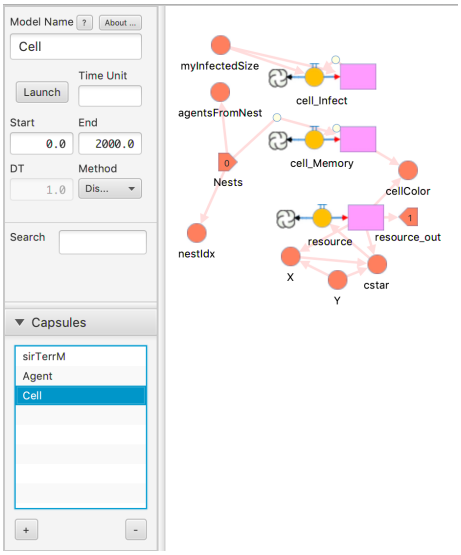
A Numerus model is structured using elements called Simulators. Here is a description of the Simulators used in our model.

Simulators. The term-of-art *Simulator* is used by Numerus Model Builder to describe a container of interacting components. In a running simulation, a Simulator is set to an initial state. As the simulation progresses its next state is determined by its current state and by other simulators with which it may be communicating. The Capsule is the simplest Simulator, and may contain any of the components described in the previous section. The five other Simulator types are array-like *aggregators* comprised of Capsule elements. Each aggregator introduces new functionalities to their constituents, as described below.

Capsule. The role of the Capsule is to contain the components of a simulation and synchronize the progress of the computation. A Capsule holds instances of the primitive components, but may also contain other Capsules and other aggregator Simulators. The top-level construct in any model is a Capsule.

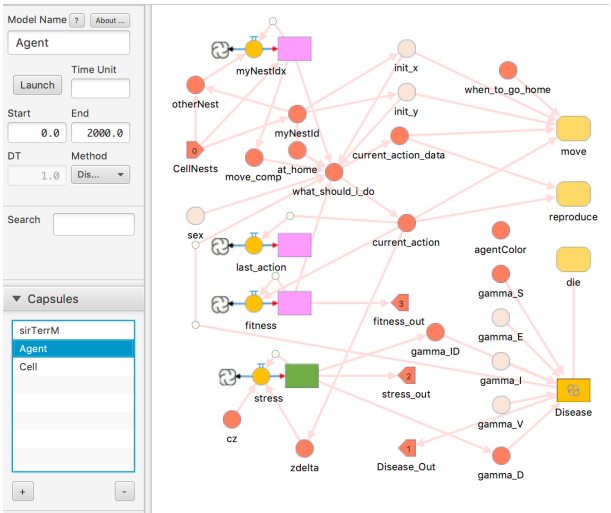
CellMatrix. A *CellMatrix* is a spatial array of individual Capsules, or *Cells* laid out in either a Cartesian lattice or hexagonal grid. Each Cell has a pair of coordinates and can query the CellMatrix to determine the other Cells in its neighborhood, with which it can exchange information. A CellMatrix can also provide an environment for agents to exist when combined with an AgentVector (see below).

In our model each cell contains resources (i.e., plants, fruits, seeds), can act as a nest site associated with a specified colony of agents, and can be occupied by one or more agents who either pass through in a single time period or remain for some time during which they exploit the dynamically changing (through both growth and extraction) resources.



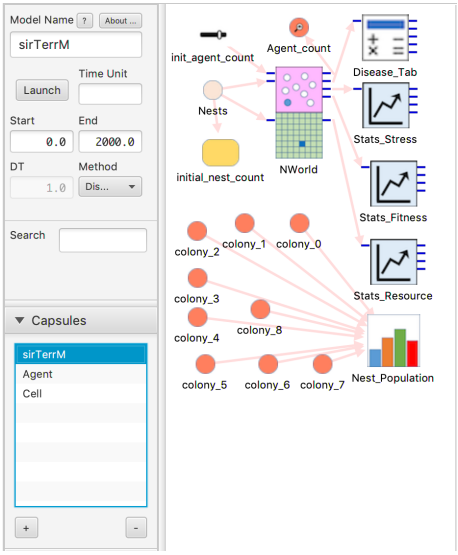
AgentVector. An *AgentVector* contains a dynamic set of Capsules called *Agents* that can exist on some landscape. At any point in time an Agent has a location on that landscape, which will change if the Agent decides to move. Agent motion will generally either use a toroidal topology or reflect off the boundary like a billiard ball (other boundary effects can be implemented). The AgentVector helps Agents find nearby Agents with which they may interact. New Agents may be created and existing Agents may be removed during the lifetime of the simulation.

In our model each Agent has a biomass (a surrogate measure of the agent's individual fitness), a stress level, and a sex (determined at birth). Females may reproduce on regular returns back to their colonies. This works via the parent retaining the majority of the biomass, the offspring acquiring a minority of biomass, and some loss from the cost of the reproduction event itself. Our movement algorithm uses a toroidal topology in order to eliminate boundary effects.



SimWorld. A *SimWorld* combines a CellMatrix with an AgentVector to create an artificial environment in which the Agents of the AgentVector range over the landscape created by the CellMatrix. Consequently, in addition to interacting with each other, Agents may determine the Cell in which they are located and interact with that or nearby Cells.

Pictured below is the top level Capsule of our model, `sirTerrM` showing the SimWorld labeled `NWorld`, which models a set of agents moving over a hexagonal cellular array. Other components in this Capsule are used to initialize the SimWorld and analyze and display results.



Numerus Model Builder also provides 2 additional Simulators for implementing a network spatial topology and using that topology as an agent landscape.

2.2 Model Components

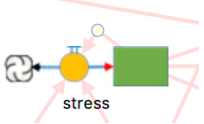
Stocks, Flows and Sequences. Traditional system dynamics modeling is centered on the stock-and-flow construct, whereby a stock, representing some dynamic model value, is updated by adding or subtracting the content of one or more flows (i.e respectively "flowing in or out" of the stock). In the simplest case, with a single flow flowing into a stock, the flow's value is the derivative of the stock's value over time. (When multiple flows are present, the derivative is found by summing the positive and negative flow values). The stock-flow model in system dynamics is especially apt for designing networks of stocks connected by flows that move values between the stocks.

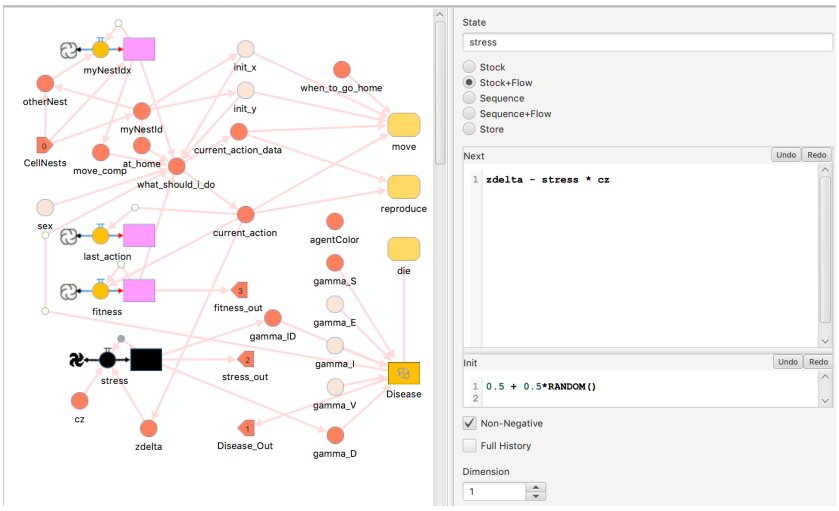
Numerus fully supports the system dynamics paradigm, but adds several convenient modifications. A single Numerus Stock-Flow component may be substituted for a Stock with a single input Flow. This configuration can be particularly useful when modeling differential equations. Numerus also introduces a variation called Sequence, in which the Flow contains the Sequence's *next* value, rather than content to be added to or subtracted from the current value. Sequences do not offer the Runge-Kutta integration algorithms available to Stocks to reduce the error when modeling continuous processes; therefore they are most useful for discrete models.

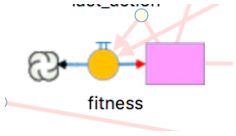
Numerus models are built using drag-and-drop gestures that place components on a design canvas. In the figures below, Stocks and Sequences respectively use lime-green and lavender-pink rectangles, and Flows are represented as yellow circles with arrows connecting to their Stocks.

The sirTerrM model uses both Stocks and Sequences in the Agent Capsule to represent continuous and discrete elements of the model.

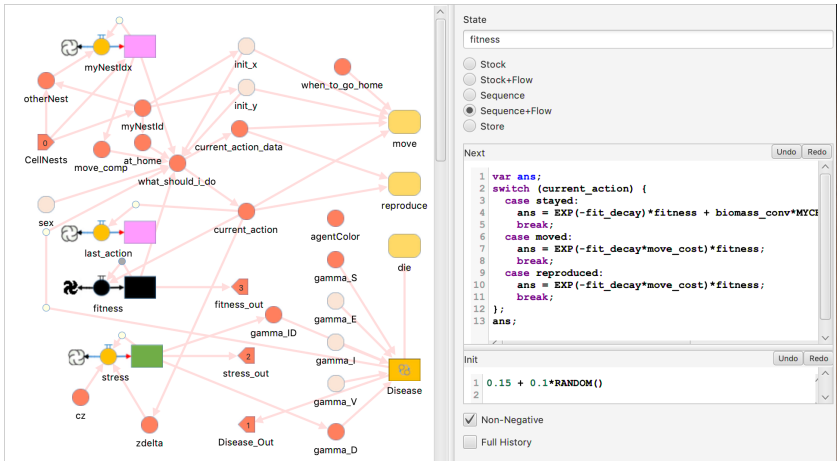
Stock:





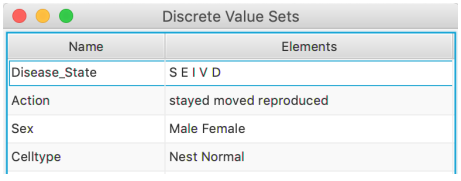


Sequence:



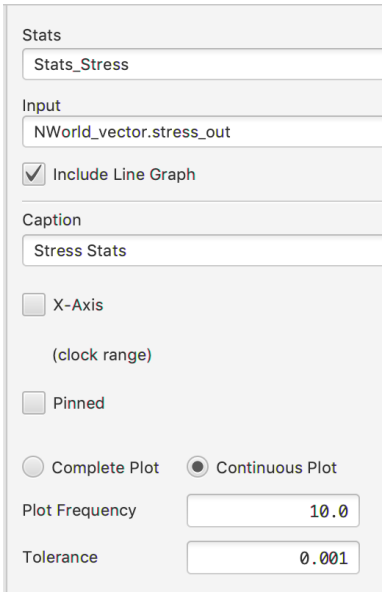
Discrete Value State. The *DiscreteState* component is similar to the Sequence, only its values are drawn from a small discrete set defined for use in the model. In the sirTerrM model, for example, the set of Disease states are represented by the letters S, E, I, V and D (for Susceptible, Exposed, Infected, Immune and Died, respectively). Other Discrete State sets identify agent sex (Male, Female), agent action (Stayed, Moved, Reproduced) and Cell type (Nest, Normal).

DiscreteState transitions are represented as a set of rules governing the transition from each allowable state. In this regard a DiscreteState component implements a finite automaton.



Graphical Display. Numerus provides a wide set of standard data display options (graphs, tables, etc.), including tableaux for visualizing the actions within cells and agents through the choice of color and (in the case of agents) size.

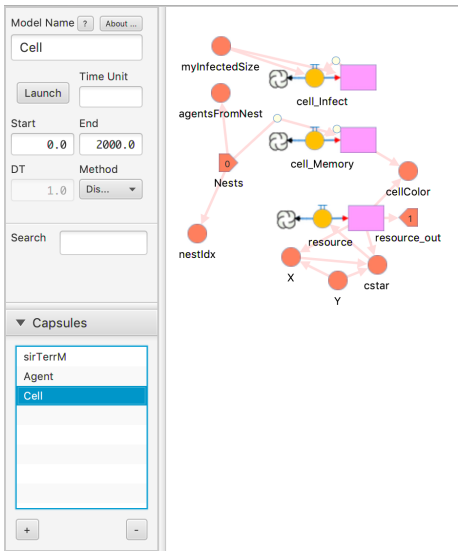
Certain analytical components also include graphs for visualizing their output. The Tabulator component determines at each point in time the number of agents or cells in a particular state. The Stats component computes statistical means and standard deviations for some state value across the set of cells or agents, and displays in a graph and table the running mean and mean ± standard deviation.



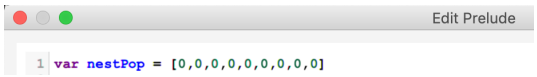
3 Building the Model

3.1 Cell Layer

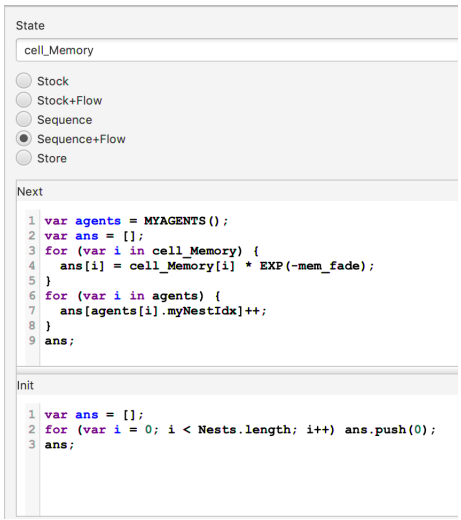
Cell Memory



The population initially consists of N_0 agents, but over time, due to births and deaths this number will change and be designated N_t. To assign this, we go to Project >> Prelude.



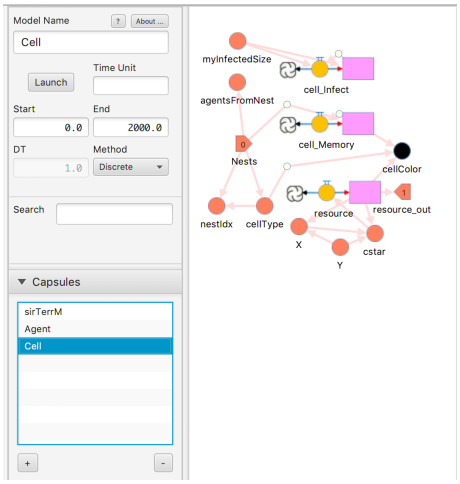
Recall that the quantities $c_{\widehat{\gamma,\delta},m}(t)$, $m = 1, ..., M$, contain a fading memory of how many individuals from colony m have visited the cell $c_{\widehat{\gamma,\delta}}$ up until time t. In our implementation these data are stored in an M-element array in the Sequence `Cell_Memory`. This array is initialized with each slot set to 0. At each point in time first the current value of each slot is aged by a factor of $e^{-\eta_M}$ (where η_M is represented in the program by the parameter `mem_fade`), and then the current population of the cell is distributed to the slots based on their nest of origin.



Note that `cell_Memory` is referenced by an agent during the process of determining its next move. This computation is contained in the agent's `move_comp` term.

Cell Colors

We want the color of a cell in our visualization to reflect properties of the cell. Each nest is assigned a unique color which displays unless the nest is empty, in which case it displays in gray. Otherwise the cell's color is computed to reflect the color of the nest of origin of the most number of visitors, or a neutral color if unvisited. This computation occurs in the Cell term `cellColor`.



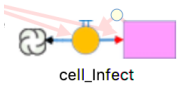
Term
cellColor
Value
Undo Redo

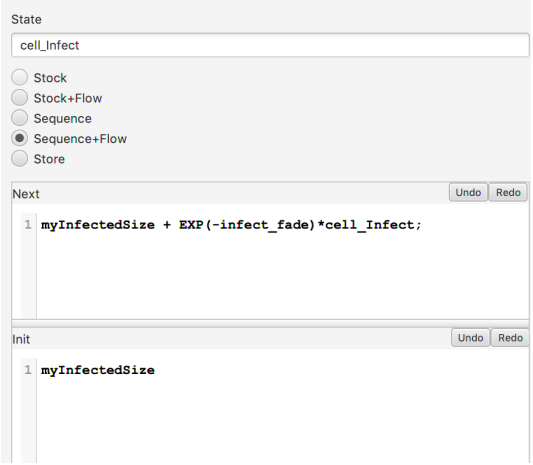
```
var ans;
if (cellType == Nest) {
  ans = (nestPop[nestIdx] == 0) ? RGB(50,50,50) : HSL(nestHSLs[nestIdx]);
}
else {
  var same = true;
  var bestIdx = 0;
  var bestVal = cell_Memory[0];
  for (var i = 1; i < cell_Memory.length; i++) {
    if (cell_Memory[i] != cell_Memory[0]) same = false;
    if (cell_Memory[i] > bestVal) {
      bestVal = cell_Memory[i]
      bestIdx = i;
    }
  }
  var l = Math.min(75, 100-resource);
  if (same) {
    ans = HSL(127, 6, l);
  } else {
    ans = HSL(nestHSLs[bestIdx][0], nestHSLs[bestIdx][1]/2, l);
  }
}
ans;
```

Disease Component

Underlying all dynamical systems modeling of epidemiological outbreaks and endemic diseases are formulations based on the concept of an SEIR progression. Susceptible individuals in disease class S enter disease class E on exposure to a pathogen (i.e., infected but not yet infectious themselves). After a period of latency, individuals in class E then transfer into the class of infectious individuals, I, only to transfer to a recovered or removed class R. However, note that once mortality is included in the model, then disease class R becomes ambiguous because removed individuals now include both recovered and dead individuals. Therefore, we use disease class V for recovered (i.e., "V" for naturally vaccinated individuals that have "recovered with immunity") and D for dead. This SEIVD notation proves useful once SEIR processes are expanded to include births, recruitment, immigration, deaths, and emigration processes.

`cell_Infect`: Each cell has a `cell_Infect` Sequence representing the infectiousness of the cell at time t based on the number of infected individuals that have visited. We require a fade rate to account for an assumed decline in the infectiousness environment of the cell over time brought about by past visitors. If the fade rate is close to 0, this implies that an indirect transmission of pathogens via the cell's environment is of significance, whereas if the fade rate goes towards ∞, then this implies that direct contact is increasingly important for transmission.





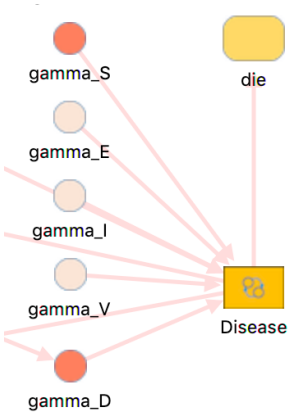
Resource

The third and final Cell state value is the resources it contains for nourishing the agents that visit. The formulae for determining resource dynamics involve both natural growth and agent consumption, and are discussed fully in the paper.

3.2 Agent Layer

Disease Transition States

`Disease Transition Terms`: We include the 5 transition states as Terms: `gamma_S`, `gamma_E`, `gamma_I`, `gamma_V`, `gamma_D`, all of which govern the transitions of the Disease Value State (refer to Section 3 Model Components). Note that `gamma_E`, `gamma_I`, and `gamma_V` are constants derived from parameters. `gamma_S`, which governs the transition from Susceptible to Exposed, depends on the value of `cell_infect` (i.e. the infectiousness of the environment); and `gamma_D`, which governs the transition to Death, depends in part on agent stress.



Movement

Determining movement is a computation that occurs in the Term `move_comp`. Section 6.2 of the manuscript explains movement in great detail. For the purpose of the Supplementary document, we will focus on a few fundamental concepts that determined our choosing of the terms and states for this particular component.

At the start of each time interval $[t, t+1]$, we initially make a case for whether or not individual $\alpha = 1, ..., N$ remains in its current cell or moves to some other cell in the set $\texttt{BLOCK}_\alpha(r)$, where r is some pre-selected positive integer that determines the furthest distance any single move can be.

We determined whether or not agent α belonging to colony ℓ would stay in its current cell or move to another one, based on either Distance Measure (`var dist`), Comparative Resource Measure (`var myResource`), and Comparative Occupation-History Measure (`var otherMaxOccupancy`).

There is also a stochastic trade-off with the agent movement rules, which play a role in determining cell attractiveness values.

Term

move_comp

Value JS R Undo Redo

```
var myCell = MYCELL();
var myRow = myCell.row;
var myCol = myCell.col;
var myBlock = myCell.BLOCK(1, true);
var myResource = myCell.resource;
var myOccupancy = myCell.cell_Memory[myNestIdx];
var act = stayed;
var dest = null;

var prob = []
var tot = 0;
for (var i in myBlock) {
  var dist = EXP(-dist_dk * DISTANCE(myRow, myCol, myBlock[i].row, myBlock[i].col));
  var otherResource = myBlock[i].resource;
  var rho = (otherResource == 0) ? 0 :
       MAX(0, 1-(myResource/otherResource));
  var omega;
  if (myOccupancy == 0) omega = 0;
  else {
    var other_cell_memory = myBlock[i].cell_Memory;
    var otherMaxOccupancy = 0;
    for (var j in other_cell_memory) {
      if (j == myNestIdx) continue;
      if (other_cell_memory[j] > otherMaxOccupancy)
        otherMaxOccupancy = other_cell_memory[j];
    }
    omega = MAX(0, 1-otherMaxOccupancy/myOccupancy);
  }
  prob[i] = MAX(0, (1 - weight_res - weight_occ)* dist + weight_res * rho + weight_occ * omega)
  tot += prob[i];
}
if (tot > 0)
  for (var i in myBlock) {
    prob[i] /= tot;
  }
dest = SELECT(myBlock, prob);
if (dest.myId != myCell.myId) act = moved;
[act, dest];
```

Command

move

Value

```
ans = new Object();
if (when_to_go_home) {
  ans.x = init_x;
  ans.y = init_y;
} else {
  var action = current_action;
  if (action == moved) {
    var dest = current_action_data;
    ans.x = dest.col;
    ans.y = dest.row;
  } else {
    ans.x = CUR_X();
    ans.y = CUR_Y();
  }
}
MOVETO(ans.x, ans.y);
```

Term

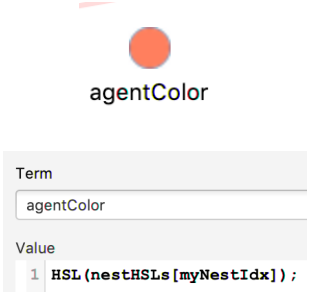
what_should_i_do

Value

```
var ans;
if (sex == Female && at_home && (Disease == S || Disease == V) && fitness > rep_fit_thresh && FLIP(p_rep)) {
  ans = new Array();
  ans[0] = reproduced;
  ans[1] = new Object({
     init_x: LOCATION().x,
     init_y: LOCATION().y,
     fitness: newborn_fit*fitness,
     myNestIdx: myNestIdx
  });
} else ans = move_comp;
ans;
```

Agent Color

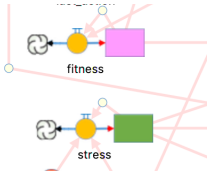
The Agent Color component also exists in order for us to observe distinctions between agents. It is determined by the agent Term `agentColor`, with each agent given the color associated with its home nest.



Fitness

A Fitness Sequence (defined in terms of the maximum resource extraction rate parameter) is also included in the model with the purpose of updating the stress state of individual agents. More specifically, the fitness of an individual is updated over the time interval$[t-1, t]$ for the maximum fitness value, the fitness decay factor, and the biomass conversion values, all of which are broken down thoroughly in Section 6.3, Ecological Dynamics, of the manuscript. If an individual moves out of a cell at time $t-1$, then the fitness is reduced by a fraction that is some multiple of the decay rate.

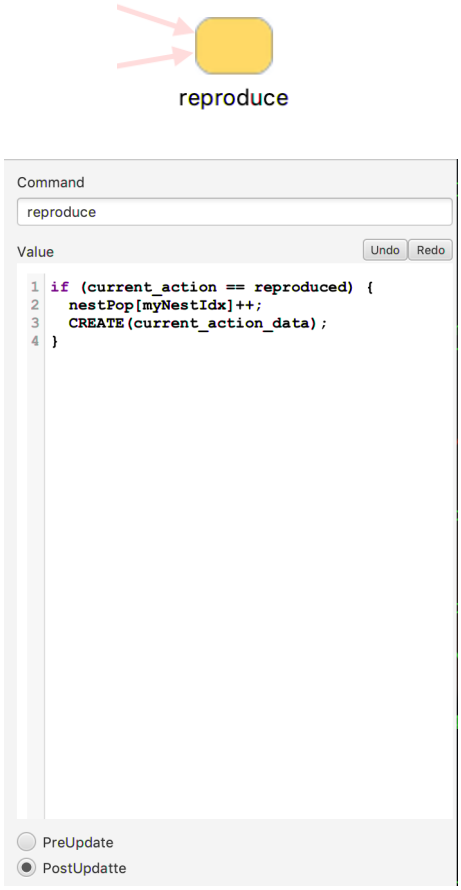
We model Stress Dynamics through the agent's Stress Stock, which contains the quantity $a_{\alpha,5}(t)$ for $a = 1, ..., N$, as described in Equation 11 of Section 6.3 in the paper. We note that the Stress component is initially given a random value between 0.5 and 1.



Reproduction

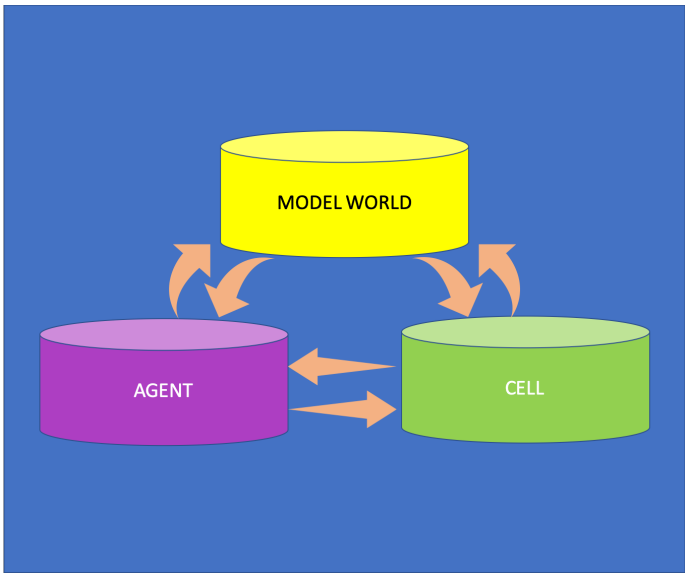
This model assumes nothing about mating structures and male limitations; therefore, only females reproduce, given a particular probability at regular colony-return times, provided they have sufficient fitness and sufficiently low stress values. Individuals die at natural-mortality specified rates, as well as with an additional disease-induced-mortality rate during their infectious period. Note that, as demonstrated in the figure below, Reproduction is a Command that occurs only when an agent selects "reproduced" as its current action. According to the rules of the model, this can occur only when the agent is in its home nest, which recurs on a cyclic basis (see the agent Term `what_should_i_do` for the rules governing this selection).

In our model, we incorporate the effect of pathogens on host population dynamics; therefore, individuals in disease states E and I do not reproduce. Individuals in disease state S or V, and whose fitness value exceeds $b_{\min}$ will not move at the next time step, but instead will give rise to a new individual. Furthermore, newly reproduced individuals will belong to the same colony as their parents.



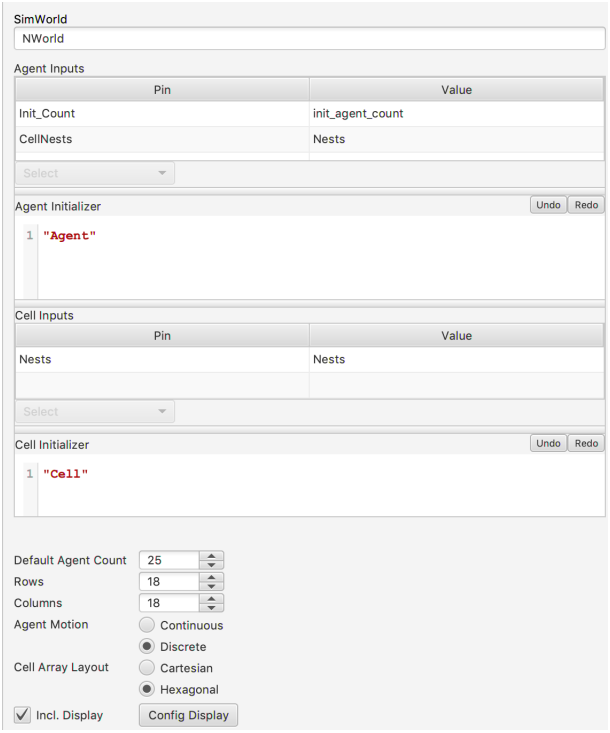
4 sirTerrM Layer

The top-most level includes the SimWorld container `NWorld` to house the cells and agents defined by the latter two Capsules.



4.1 Nests and Colonies

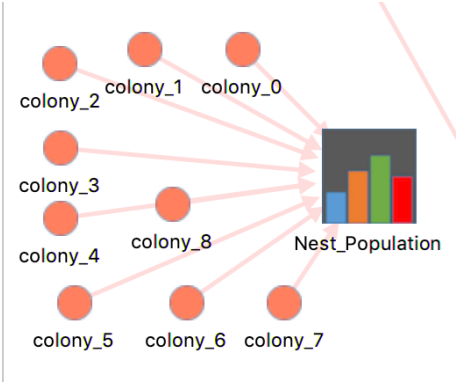
The Prelude contains the globally defined array `nestPop` that keeps track of the population of the 9 colonies used by the model. The Terms `colony_0` through `colony_8` each select a distinct slot of this array for inclusion in the bar graph `Nest_Population` (see below). The locations of the nests are defined in the Term `Nests` using the Numerus Cell ID notation $CY{:}X$, where Y and X respectively indicate the row and column address of the cell. Nest locations are fed to the SimWorld through input pins to both its Agent and Cell constituents, since both need to be aware of these locations.

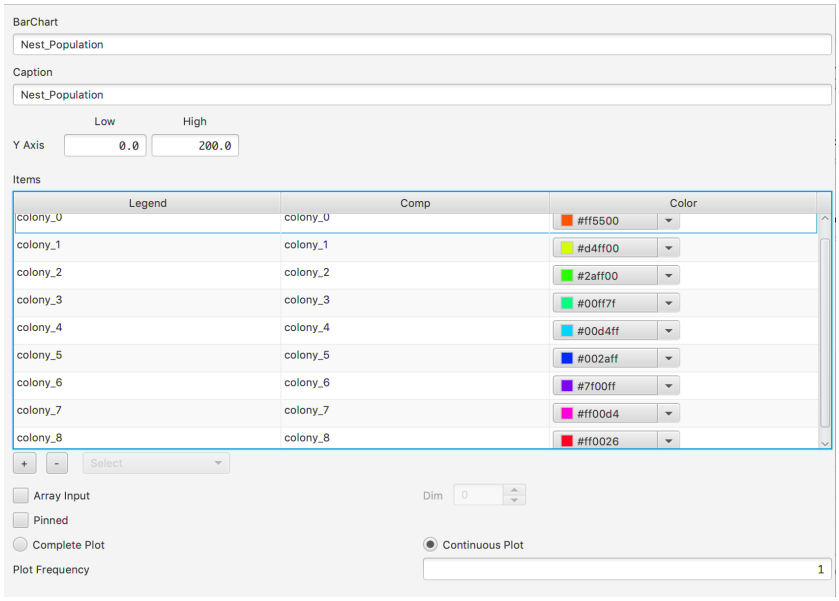


4.2 Data Displays

There are a total of 6 displays showing the state of the running model. The first is a visualization of model activity generated by the `NWorld` SimWorld. The second and third are, respectively, the bar graph mentioned above showing the current colony populations, and line graphs tabulating the current disease state of the agents. The final three displays provide statistical information across the agent and cell populations for Stress and Fitness

(agent); and Resource (cell). The Stats plug-in is used to create graphs showing the mean and mean ± standard deviation of each of these quantities.





5 Text-to-Model Terminology Translation

The tables below provide a translation between the notation used in the text and that employed in the model.

Table 1: Translation of notation between mathematics in text and NMB code

Mathematics in text	Notation	Agent-Level	Cell-Level
N_0, N_t	`agent _count`		
$H_\gamma \times H_\delta$	γ, δ	`rows, columns`	`rows, columns`
M		`CellNests.length`	`Nests.length`
$\omega_{\widehat{\gamma,\delta_i}}$	`move_comp`	i^{th} `cell`	$\texttt{RING}_\alpha(\texttt{r}_\texttt{i})$
K	`max _rad`		`max_radius`
$c_{\widehat{\gamma,\delta_i},m+1}$	`cell_memory`	`myOccupancy`	
A_i	index of an agent's nest list of agents in a nest	`myNestIdx`	`MyNesters`
T_f	`go_home`		
$\texttt{BLOCK}_\alpha(r)$	$\cup_{i=0}^{r}\ \texttt{RING}_\alpha(i)$		

$c_{\widehat{\gamma,\delta}}$
$c_{\widehat{\gamma,\delta},1}$
$c_{\widehat{\gamma,\delta},2}$ $c_{\widehat{\gamma,\delta},m+1}$
$c_{\widehat{\gamma,\delta},m+2}$
	`occupancy`	`resource`
		`[0] to [`$M-1$`]`
		`cell _Infect`
			`infectivity of cell`

ϕ_1 ϕ_2		`weight_res` `weight_occ`	$\phi 1 + \phi 2 \leq 1$
$p_{\alpha,i}$			`trade-off ratio`

$\mathbf{a}_\alpha, \alpha = 1, ..., N_t$
$a_{\alpha,1}$
$a_{\alpha,2}$
$a_{\alpha,3}$
$a_{\alpha,4}$
$a_{\alpha,5}$ (also $z(t)$)
$a_{\alpha,\ell}$, $\ell = 6, 7, \ldots$
	`alpha_5`	(not used)
		`myNestIdx`
		`disease`
		`fitness`
		`stress input`	

Manuscript Notation	Notation	Agent-Level	Cell-Level
h	eff_ex	extraction-efficiency	
q	comp_ex	competition param.	
u	max_res_ex	max resource extraction	
s	eff_ex	saturation	
c_z	c_z		
z	deltaz		
c_m	stress_scale	conversion constant	
b_{min}	rep_fit_thresh	min P(reproduction)	
$b_{\min}$	p_rep	P(reproduction)	
ε	max_ra		
$\texttt{BLOCK}_\alpha(r)$	myBlock		
X^*	res_def	resource-deficit	
ψ	fit_decay		
r	res_grow	resource growth	
s	res_sat		saturation param.
g	res_reservoir		reservoir param.
μ_0	nat_death	natural death	
μ_1	disease_death	mortality	
$y_m(t)$	indiv count who visit cell type		
β	disease_trans		
κ	biomass_conv	value fixed at 0.2	
$\eta_{\mathrm{I}} > 0$			infect_fade
$\eta_{\mathrm{M}} > 0$			memory_fade
γ_{E}	gamma_E	latent_inv	
γ_{I}	gamma_I	infect_inv	
γ_{V}	gamma_V	immuneloss_inv	

6 NovaScript

The NovaScript generated by NMB is attached to the end of this document. The model file is called GetzEtAlPLoSOct2019.nmd, which can be downloaded at

https://www.dropbox.com/s/24j6js6js6h5ur3/GetzEtAlPLoSOct2019.nmd?dl=0

and can be run after downloading a version of NMB at numerusinc.com

References

[1] Getz WM, Salter R, Lyons AJ, Sippl-Swezey N. Panmictic and clonal evolution on a single patchy resource produces polymorphic foraging guilds. PloS One. 2015;10(8):e0133732.

[2] Getz WM, Salter R, Seidel DP, Van Hooft P. Sympatric speciation in structureless environments. BMC evolutionary biology. 2016;16(1):50.

[3] Getz WM, Salter R, Muellerklein O, Yoon HS, Tallam K. Modeling epidemics: A primer and Numerus Model Builder implementation. Epidemics. 2018;.

[4] Alexander KA, Carlson CJ, Lewis BL, Getz WM, Marathe MV, Eubank SG, et al. The Ecology of Pathogen Spillover and Disease Emergence at the Human-Wildlife-Environment Interface. In: The Connections Between Ecology and Infectious Disease. Springer; 2018. p. 267–298.

[5] Ball F, Nåsell I. The shape of the size distribution of an epidemic in a finite population. Mathematical biosciences. 1994;123(2):167–181.

[6] Gordillo LF, Marion SA, Martin-Löf A, Greenwood PE. Bimodal epidemic size distributions for near-critical SIR with vaccination. Bulletin of Mathematical Biology. 2008;70(2):589–602.

```
var fit_max = 1;
var rho_crit = 0.3;
var biomass_conv = 0.2;
var cycle_home = 10;
var eff_ex = 5;
var max_res_ex = 5;
var res_sat = 20;
var res_reservoir = 0.1;
var stress_scale = 0.5;
var prob_f = 0.6;
var p_rep = 0.8;
var parent_fit = 0.65;
var newborn_fit = 0.3;
var max_rad = 2;
var mem_fade = 0.3;
var dist_dk = 0.3;
var infect_fade = 0.5;
var latent_inv = 0.5;
var infect_inv = 0.2;
var immunelose_inv = 0.02;

defineSchema("Cell", {
    specifies: "Capsule",
    methods: {
    },
    components: {
        cell_Memory : {
            specifies: "Sequence",
            count: 1,
            nonNegative: false,
            history: 100,
            initial: function(){
                var ans = [];
                for (var i = 0; i < this.Nests.length; i++)
ans.push(0);
                return ans;
                },
            next: function(){
                var agents = this.MYAGENTS();
                var ans = [];
                for (var i in this.cell_Memory) {
                  ans[i] = this.cell_Memory[i] * EXP(-this.mem_fade);
                }
                for (var i in agents) {
                  ans[agents[i].myNestIdx]++;
                }
                return ans;
                },
        },
```

```
        resource : {
            specifies: "Sequence",
            count: 1,
            nonNegative: false,
            history: 100,
            initial: '2+this.RANDOM(1)',
            next: 'this.cstar + this.res_grow * (1 - this.cstar/
this.res_sat)*(this.cstar + this.res_reservoir)',
        },
        cell_Infect : {
            specifies: "Sequence",
            count: 1,
            nonNegative: false,
            history: 100,
            initial: 'this.myInfectedSize',
            next: 'this.myInfectedSize + EXP(-
this.infect_fade)*this.cell_Infect;',
        },
        cellType : {
            specifies: "Term",
            exp: '(this.Nests.indexOf(this.myId) >= 0) ? Nest :
Normal',
        },
        nestIdx : {
            specifies: "Term",
            exp: '(this.cellType == Nest) ?
this.Nests.indexOf(this.myId) : -1;',
        },
        Y : {
            specifies: "Term",
            exp: function(){
                var agts = this.MYAGENTS();
                agts = _.filter(agts, function(x){return x.last_action
== stayed;});
                var count = agts.length;
                return count;
                },
        },
        cstar : {
            specifies: "Term",
            exp: 'this.resource - this.Y*this.X',
        },
        cellColor : {
            specifies: "Term",
            exp: function(){
                var ans;
                if (this.cellType == Nest) {
                  ans = (nestPop[this.nestIdx] == 0) ? RGB(50,50,50) :
HSL(nestHSLs[this.nestIdx]);
                }
```

```
                else {
                  var same = true;
                  var bestIdx = 0;
                  var bestVal = this.cell_Memory[0];
                  for (var i = 1; i < this.cell_Memory.length; i++) {
                    if (this.cell_Memory[i] != this.cell_Memory[0])
same = false;
                    if (this.cell_Memory[i] > bestVal) {
                      bestVal = this.cell_Memory[i]
                      bestIdx = i;
                    }
                  }
                  if (same) {
                    var l = 50-(this.resource/2);
                    ans = HSL(127, 85, l);
                  } else {
                    var l = Math.min(75, 100-this.resource);
                    ans = HSL(nestHSLs[bestIdx][0], nestHSLs[bestIdx]
[1]/2, l);
                  }
                }
                return ans;
                },
        },
        myInfectedSize : {
            specifies: "Term",
            exp: 'this.COUNT(function(agt){return agt.Disease == I ||
agt.Disease == E;}, this.MYAGENTS());',
        },
        Nests : {
            specifies: "InPin",
            exp: 0.0,
        },
        agentsFromNest : {
            specifies: "Term",
            exp: function(){
                var ans = []
                for (var i = 0; i < this.Nests.length; i++)
ans.push(0);
                var agents = this.MYAGENTS();
                for (var idx in agents) {
                  agent = agents[idx];
                  var nestIdx = agent.myNestIdx;
                  ans[nestIdx]++;
                }
                return ans;
                },
        },
        resource_out : {
            specifies: "OutPin",
```

```
            exp: 'this.resource',
        },
        X : {
            specifies: "Term",
            exp: function(){
                var ans = 0.0;
                if (this.Y > 0) {
                  var v0 = this.resource/this.Y;
                  var v1 = (this.max_res_ex * this.resource)/
(this.eff_ex * (1 + this.comp_ex * (this.Y-1)) + this.resource);
                  ans = MIN(v0, v1);
                }
                return ans;
                },
        },
        disease_trans : {
            specifies: "InPin",
            exp: "mscope.disease_trans",
        },
        disease_death : {
            specifies: "InPin",
            exp: "mscope.disease_death",
        },
        p_switch : {
            specifies: "InPin",
            exp: "mscope.p_switch",
        },
        nat_death : {
            specifies: "InPin",
            exp: "mscope.nat_death",
        },
        weight_res : {
            specifies: "InPin",
            exp: "mscope.weight_res",
        },
        weight_occ : {
            specifies: "InPin",
            exp: "mscope.weight_occ",
        },
        res_grow : {
            specifies: "InPin",
            exp: "mscope.res_grow",
        },
        comp_ex : {
            specifies: "InPin",
            exp: "mscope.comp_ex",
        },
        rep_fit_thresh : {
            specifies: "InPin",
            exp: "mscope.rep_fit_thresh",
```

```
        },
        res_def : {
            specifies: "InPin",
            exp: "mscope.res_def",
        },
        stress_decay : {
            specifies: "InPin",
            exp: "mscope.stress_decay",
        },
        stress_death : {
            specifies: "InPin",
            exp: "mscope.stress_death",
        },
        fit_decay : {
            specifies: "InPin",
            exp: "mscope.fit_decay",
        },
        move_cost : {
            specifies: "InPin",
            exp: "mscope.move_cost",
        },
    },
});

defineSchema("Agent", {
    specifies: "Capsule",
    methods: {
    },
    components: {
        fitness : {
            specifies: "Sequence",
            count: 1,
            nonNegative: true,
            history: 100,
            initial: '0.15 + 0.1*this.RANDOM()',
            next: function(){
                var ans;
                switch (this.current_action) {
                  case stayed:
                    ans = EXP(-this.fit_decay)*this.fitness +
this.biomass_conv*this.MYCELL().X*MAX(0, 1-(this.fitness/
this.fit_max));
                    break;
                  case moved:
                    ans = EXP(-
this.fit_decay*this.move_cost)*this.fitness;
                    break;
                  case reproduced:
                    ans = EXP(-
this.fit_decay*this.move_cost)*this.fitness;
```

```
                    break;
                };
                return ans;
                },
        },
        stress : {
            specifies: "Variable",
            count: 1,
            nonNegative: true,
            history: 100,
            initial: '0.5 + 0.5*this.RANDOM()',
            prime: 'this.zdelta - this.stress * this.cz',
        },
        last_action : {
            specifies: "Sequence",
            count: 1,
            nonNegative: false,
            history: 100,
            initial: 'stayed',
            next: 'this.current_action',
        },
        myNestIdx : {
            specifies: "Sequence",
            count: 1,
            nonNegative: false,
            history: 100,
            initial: 'FLOOR(this.RANDOM(this.CellNests.length))',
            next: function(){
                var ans = this.myNestIdx;
                if (this.p_switch > 0) {
                  if (this.otherNest >= 0 &&
                      nestPop[this.otherNest]/nestPop[this.myNestIdx]
< this.rho_crit &&
                      this.FLIP(this.p_switch)) {
                    ans = this.otherNest;
                    nestPop[this.myNestIdx]--;
                    nestPop[this.otherNest]++;
                  };
                }
                return ans
                },
        },
        gamma_ID : {
            specifies: "Term",
            exp: '(this.nat_death + this.disease_death) * (1 +
this.stress_death*this.stress);',
        },
        gamma_D : {
            specifies: "Term",
            exp: 'this.nat_death * (1 +
```

```
this.stress_death*this.stress);',
        },
        gamma_V : {
            specifies: "Term",
            constant: true,
            exp: 'this.immunelose_inv',
        },
        gamma_I : {
            specifies: "Term",
            constant: true,
            exp: 'this.infect_inv',
        },
        gamma_E : {
            specifies: "Term",
            constant: true,
            exp: 'this.latent_inv',
        },
        gamma_S : {
            specifies: "Term",
            exp: 'this.disease_trans * this.MYCELL().cell_Infect',
        },
        when_to_go_home : {
            specifies: "Term",
            exp: 'this.TIME()%this.cycle_home == 0',
        },
        myNestId : {
            specifies: "Term",
            exp: 'this.CellNests[this.myNestIdx]',
        },
        agentColor : {
            specifies: "Term",
            exp: 'HSL(nestHSLs[this.myNestIdx]);',
        },
        init_x : {
            specifies: "Term",
            constant: true,
            exp: 'this.COL_OF(this.myNestId)',
        },
        init_y : {
            specifies: "Term",
            constant: true,
            exp: 'this.ROW_OF(this.myNestId)',
        },
        CellNests : {
            specifies: "InPin",
            exp: 0.0,
        },
        Disease_Out : {
            specifies: "OutPin",
            exp: 'this.Disease',
```

```
        },
        cz : {
            specifies: "Term",
            exp: 'this.stress_decay',
        },
        zdelta : {
            specifies: "Term",
            exp: function(){
                var ans;
                switch (this.current_action) {
                  case stayed:
                  case reproduced:
                    ans = MAX(0, this.stress_scale * (this.res_def -
this.MYCELL().X));
                    break;
                  case moved:
                    ans = this.stress_scale * this.res_def;
                    break;
                };
                return ans;
                },
        },
        at_home : {
            specifies: "Term",
            exp: 'this.myNestId == this.MYCELL().myId',
        },
        what_should_i_do : {
            specifies: "Term",
            exp: function(){
                var ans;
                if (this.sex == Female && this.at_home &&
(this.Disease == S || this.Disease == V) && this.fitness >
this.rep_fit_thresh && this.FLIP(this.p_rep)) {
                  ans = new Array();
                  ans[0] = reproduced;
                  ans[1] = new Object({
                     init_x: this.LOCATION().x,
                     init_y: this.LOCATION().y,
                     fitness: this.newborn_fit*this.fitness,
                     myNestIdx: this.myNestIdx
                  });
                } else ans = this.move_comp;
                return ans;
                },
        },
        current_action : {
            specifies: "Term",
            exp: 'this.what_should_i_do[0]',
        },
        current_action_data : {
```

```
            specifies: "Term",
            exp: 'this.what_should_i_do[1]',
        },
        stress_out : {
            specifies: "OutPin",
            exp: 'this.stress',
        },
        fitness_out : {
            specifies: "OutPin",
            exp: 'this.fitness',
        },
        move_comp : {
            specifies: "Term",
            exp: function(){
                var myCell = this.MYCELL();
                var myRow = myCell.row;
                var myCol = myCell.col;
                var myBlock = myCell.BLOCK(1, true);
                var myResource = myCell.resource;
                var myOccupancy = myCell.cell_Memory[this.myNestIdx];
                var act = stayed;
                var dest = null;

                // rule 1
                var prob = []
                var tot = 0;
                for (var i in myBlock) {
                  var dist = EXP(-this.dist_dk * this.DISTANCE(myRow,
myCol, myBlock[i].row, myBlock[i].col));
                  var otherResource = myBlock[i].resource;
                  var rho = (otherResource == 0) ? 0 :
                       MAX(0, 1-(myResource/otherResource));
                  var omega;
                  if (myOccupancy == 0) omega = 0;
                  else {
                    var other_cell_memory = myBlock[i].cell_Memory;
                    var otherMaxOccupancy = 0;
                    for (var j in other_cell_memory) {
                      if (j == this.myNestIdx) continue;
                      if (other_cell_memory[j] > otherMaxOccupancy)
                        otherMaxOccupancy = other_cell_memory[j];
                    }
                    omega = MAX(0, 1-otherMaxOccupancy/myOccupancy);
                  }
                  //prob[i] = MAX(0, (1 - (weight_res+weight_occ) *
dist + weight_res * rho + weight_occ * omega))
                    prob[i] = MAX(0, (1 - this.weight_res -
this.weight_occ)* dist + this.weight_res * rho + this.weight_occ *
omega)
                  tot += prob[i];
```

```
        }
        if (tot > 0)
          for (var i in myBlock) {
            prob[i] /= tot;
          }
        dest = this.SELECT(myBlock, prob);
        if (dest != myCell) act = moved;
        return [act, dest];
        },
},
sex : {
    specifies: "Term",
    constant: true,
    exp: '(this.FLIP(this.prob_f)) ? Female : Male',
},
otherNest : {
    specifies: "Term",
    exp: function(){
        found = -1;
        var myCell = this.MYCELL().myId;
        if (myCell != this.myNestId) {
          for (var i in this.CellNests) {
            if (this.CellNests[i] == myCell) {
              found = i;
              break;
            }
          }
        };
        return found;
        },
},
move : {
    specifies: "Command",
    when: "post",
    exp: function(){
        ans = new Object();
        if (this.when_to_go_home) {
          ans.x = this.init_x;
          ans.y = this.init_y;
        } else {
          var action = this.current_action;
          if (action == moved) {
            var dest = this.current_action_data;
            ans.x = dest.col;
            ans.y = dest.row;
          } else {
            ans.x = this.CUR_X();
            ans.y = this.CUR_Y();
          }
        }
```

```
                this.MOVETO(ans.x, ans.y);
                }
        },
        reproduce : {
            specifies: "Command",
            when: "post",
            exp: function(){
                if (this.current_action == reproduced) {
                  nestPop[this.myNestIdx]++;
                  this.CREATE(this.current_action_data);
                }
                }
        },
        die : {
            specifies: "Command",
            when: "post",
            exp: function(){
                if (this.Disease == D) {
                  this.Disease == S;
                  nestPop[this.myNestIdx]--;
                  this.KILL(this.myId);
                }
                }
        },
        Disease : {
            specifies: "DiscreteState",
            dsGroup: "Disease_State",
            stateSize: 5,
            stateNames: ["S", "E", "I", "V", "D"],
            initial: '(this.myId < 1) ? E : S',
            next: {
                S: 'this.SELECT([E,D,S],\n'+
                    '[this.gamma_S*(1-EXP(-this.gamma_S -
this.gamma_D))/(this.gamma_S + this.gamma_D),\n'+
                    'this.gamma_D*(1-EXP(-this.gamma_S -
this.gamma_D))/(this.gamma_S + this.gamma_D),\n'+
                    'EXP(-this.gamma_S - this.gamma_D)]);\n',
                E: 'this.SELECT([I,D,E],\n'+
                    '[this.gamma_E*(1-EXP(-this.gamma_E -
this.gamma_D))/(this.gamma_E+this.gamma_D),\n'+
                    'this.gamma_D*(1-EXP(-this.gamma_E -
this.gamma_D))/(this.gamma_E+this.gamma_D),\n'+
                    'EXP(-this.gamma_E - this.gamma_D)]);\n',
                I: 'this.SELECT([V,D,I],\n'+
                    '[this.gamma_I*(1-EXP(-this.gamma_I -
this.gamma_ID))/(this.gamma_I+this.gamma_ID),\n'+
                    'this.gamma_ID*(1-EXP(-this.gamma_I -
this.gamma_ID))/(this.gamma_I+this.gamma_ID),\n'+
                    'EXP(-this.gamma_I - this.gamma_ID)]);\n',
                V: 'this.SELECT([S,D,V],\n'+
```

```
                    '[this.gamma_V*(1-EXP(-this.gamma_V -
this.gamma_D))/(this.gamma_V+this.gamma_D),\n'+
                    'this.gamma_D*(1-EXP(-this.gamma_V -
this.gamma_D))/(this.gamma_V+this.gamma_D),\n'+
                    'EXP(-this.gamma_V - this.gamma_D)]);\n',
                D: 'D',
            },
        },
        disease_trans : {
            specifies: "InPin",
            exp: "mscope.disease_trans",
        },
        disease_death : {
            specifies: "InPin",
            exp: "mscope.disease_death",
        },
        p_switch : {
            specifies: "InPin",
            exp: "mscope.p_switch",
        },
        nat_death : {
            specifies: "InPin",
            exp: "mscope.nat_death",
        },
        weight_res : {
            specifies: "InPin",
            exp: "mscope.weight_res",
        },
        weight_occ : {
            specifies: "InPin",
            exp: "mscope.weight_occ",
        },
        res_grow : {
            specifies: "InPin",
            exp: "mscope.res_grow",
        },
        comp_ex : {
            specifies: "InPin",
            exp: "mscope.comp_ex",
        },
        rep_fit_thresh : {
            specifies: "InPin",
            exp: "mscope.rep_fit_thresh",
        },
        res_def : {
            specifies: "InPin",
            exp: "mscope.res_def",
        },
        stress_decay : {
            specifies: "InPin",
```

```
            exp: "mscope.stress_decay",
        },
        stress_death : {
            specifies: "InPin",
            exp: "mscope.stress_death",
        },
        fit_decay : {
            specifies: "InPin",
            exp: "mscope.fit_decay",
        },
        move_cost : {
            specifies: "InPin",
            exp: "mscope.move_cost",
        },
    },
});

defineSchema("main", {
    specifies: "Capsule",
    methods: {
    },
    components: {
        NWorld : {
            specifies: "SimWorld",
            count: 25,
            rows: 18,
            cols: 18,
            agentbase: "Agent",
            cellbase: "Cell",
            motion: "discrete",
            layout: "hexagonal",
            ainputs: {
                Init_Count: "this.Super.init_agent_count",
                CellNests: "this.Super.Nests",
            },
            cinputs: {
                Nests: "this.Super.Nests",
            },
        },
        Nests : {
            specifies: "Term",
            constant: true,
            exp: 'this.ASVALUE(\n'+
                '["C3:3","C3:9","C3:15",\n'+
                '"C9:3","C9:9","C9:15",\n'+
                '"C15:3","C15:9","C15:15"\n'+
                ']\n'+
                ');\n',
        },
        Agent_count : {
```

```
            specifies: "Term",
            exp: 'this.NWorld.AData("AGENTCOUNT")',
        },
        colony_0 : {
            specifies: "Term",
            exp: 'nestPop[0]',
        },
        colony_1 : {
            specifies: "Term",
            exp: 'nestPop[1]',
        },
        colony_2 : {
            specifies: "Term",
            exp: 'nestPop[2]',
        },
        colony_3 : {
            specifies: "Term",
            exp: 'nestPop[3]',
        },
        colony_4 : {
            specifies: "Term",
            exp: 'nestPop[4]',
        },
        colony_5 : {
            specifies: "Term",
            exp: 'nestPop[5]',
        },
        colony_6 : {
            specifies: "Term",
            exp: 'nestPop[6]',
        },
        colony_7 : {
            specifies: "Term",
            exp: 'nestPop[7]',
        },
        colony_8 : {
            specifies: "Term",
            exp: 'nestPop[8]',
        },
        initial_nest_count : {
            specifies: "Command",
            when: "pre",
            exp: function(){
                nests = this.Nests();
                for (var i = 0; i < nests.length; i++) {
                  nestPop[i] =
this.NWorld.CELL(nests[i]).MYAGENT_COUNT();
                }
                }
        },
```

```
Nest_Population : {
    specifies: "Plugin",
    base: "PL_Barchart",
    pins: {
        inpt0: "this.colony_0",
        inpt1: "this.colony_1",
        inpt2: "this.colony_2",
        inpt3: "this.colony_3",
        inpt4: "this.colony_4",
        inpt5: "this.colony_5",
        inpt6: "this.colony_6",
        inpt7: "this.colony_7",
        inpt8: "this.colony_8",
    },
    properties: __props__["main"]["Nest_Population"],
},
init_agent_count : {
    specifies: "Slider",
    properties: __props__["main"]["init_agent_count"],
},
Stats_Stress : {
    specifies: "Plugin",
    base: "PL_Stats",
    pins: {
        source: "NWorld_vector",
        In: "stress_out",
    },
    properties: __props__["main"]["Stats_Stress"],
},
Stats_Stress_graph : {
    specifies: "Plugin",
    base: "PL_Linechart",
    pins: {
        inpt0: "this.Stats_Stress.Mean",
        inpt1: "this.Stats_Stress.M_Plus_S",
        inpt2: "this.Stats_Stress.M_Minus_S",
    },
    properties: __props__["main"]["Stats_Stress_graph"],
},
Stats_Fitness : {
    specifies: "Plugin",
    base: "PL_Stats",
    pins: {
        source: "NWorld_vector",
        In: "fitness_out",
    },
    properties: __props__["main"]["Stats_Fitness"],
},
Stats_Fitness_graph : {
    specifies: "Plugin",
```

```
    base: "PL_Linechart",
    pins: {
        inpt0: "this.Stats_Fitness.Mean",
        inpt1: "this.Stats_Fitness.M_Plus_S",
        inpt2: "this.Stats_Fitness.M_Minus_S",
    },
    properties: __props__["main"]["Stats_Fitness_graph"],
},
Disease_Tab : {
    specifies: "Plugin",
    base: "PL_Tabulator",
    pins: {
        source: "NWorld_vector",
        In: "Disease_Out",
    },
    properties: __props__["main"]["Disease_Tab"],
},
Disease_Tab_graph : {
    specifies: "Plugin",
    base: "PL_Linechart",
    pins: {
        inpt0: "this.Disease_Tab.Out_00",
        inpt1: "this.Disease_Tab.Out_01",
        inpt2: "this.Disease_Tab.Out_02",
        inpt3: "this.Disease_Tab.Out_03",
        inpt4: "this.Disease_Tab.Out_04",
    },
    properties: __props__["main"]["Disease_Tab_graph"],
},
Stats_Resource : {
    specifies: "Plugin",
    base: "PL_Stats",
    pins: {
        source: "NWorld_matrix",
        In: "resource_out",
    },
    properties: __props__["main"]["Stats_Resource"],
},
Stats_Resource_graph : {
    specifies: "Plugin",
    base: "PL_Linechart",
    pins: {
        inpt0: "this.Stats_Resource.Mean",
        inpt1: "this.Stats_Resource.M_Plus_S",
        inpt2: "this.Stats_Resource.M_Minus_S",
    },
    properties: __props__["main"]["Stats_Resource_graph"],
},
NWorld_display : {
    specifies: "Plugin",
```

```
    base: "PL_AgentViewerX",
    pins: {
        agentsource: "NWorld",
        cellsource: "NWorld",
        aColorOut: "agentColor",
        cColorOut: "cellColor",
        aSizeOut: "fitness",
    },
    properties: __props__["main"]["NWorld_display"],
},
Agent_count_spy : {
    specifies: "Plugin",
    base: "PL_Spy",
    pins: {
        inpt: "this.Agent_count",
    },
    properties: __props__["main"]["Agent_count_spy"],
},
disease_trans : {
    specifies: "Slider",
    properties: __props__["disease_trans"],
},
disease_death : {
    specifies: "Slider",
    properties: __props__["disease_death"],
},
p_switch : {
    specifies: "Slider",
    properties: __props__["p_switch"],
},
nat_death : {
    specifies: "Slider",
    properties: __props__["nat_death"],
},
weight_res : {
    specifies: "Slider",
    properties: __props__["weight_res"],
},
weight_occ : {
    specifies: "Slider",
    properties: __props__["weight_occ"],
},
res_grow : {
    specifies: "Slider",
    properties: __props__["res_grow"],
},
comp_ex : {
    specifies: "Slider",
    properties: __props__["comp_ex"],
},
```

```
        rep_fit_thresh : {
            specifies: "Slider",
            properties: __props__["rep_fit_thresh"],
        },
        res_def : {
            specifies: "Slider",
            properties: __props__["res_def"],
        },
        stress_decay : {
            specifies: "Slider",
            properties: __props__["stress_decay"],
        },
        stress_death : {
            specifies: "Slider",
            properties: __props__["stress_death"],
        },
        fit_decay : {
            specifies: "Slider",
            properties: __props__["fit_decay"],
        },
        move_cost : {
            specifies: "Slider",
            properties: __props__["move_cost"],
        },
    },
});

MAKE_ENUM("Disease_State","S","E","I","V","D");
MAKE_ENUM("Action","stayed","moved","reproduced");
MAKE_ENUM("Sex","Male","Female");
MAKE_ENUM("Celltype","Nest","Normal");
var nestPop = [0,0,0,0,0,0,0,0,0]

var nestHSLs = [
  [30,100,50],
  [70,100,50],
  [110,100,50],

  [150,100,50],
  [190,100,50],
  [230,100,50],

  [270,100,50],
  [310,100,50],
  [350,100,50],
];
```

```
var __clockParams__ = {
    lo: 0.000000, hi: 2000.000000, max: 2000.000000, dt: 1.000000,
method: "Discrete", step: 1, unit:"null"
}


var __props__ = {
    title: "sirTerrM",
    address: "rms@cs.oberlin.edu",
    date: "Sun Sep 08 08:49:02 PDT 2019",
    paramset: "Param_set_1",
    clockParams: {lo: 0.000000, hi: 2000.000000, max: 2000.000000, dt:
1.000000, method: "Discrete", step: 1, unit:"null"},
    disease_trans: {
        type: "slider", id: "disease_trans", text: "disease_trans",
size: 1.000000, min: 0.000000, max: 1.000000, value: 0, step:
0.010000, unit: ""
    },
    disease_death: {
        type: "slider", id: "disease_death", text: "disease_death",
size: 1.000000, min: 0.000000, max: 1.000000, value: 0, step:
0.010000, unit: ""
    },
    p_switch: {
        type: "slider", id: "p_switch", text: "p_switch", size:
1.000000, min: 0.000000, max: 1.000000, value: 0, step: 0.100000,
unit: ""
    },
    nat_death: {
        type: "slider", id: "nat_death", text: "nat_death", size:
1.000000, min: 0.000000, max: 0.100000, value: 0.005, step: 0.001000,
unit: ""
    },
    weight_res: {
        type: "slider", id: "weight_res", text: "weight_res", size:
1.000000, min: 0.000000, max: 1.000000, value: 1/3, step: 0.010000,
unit: ""
    },
    weight_occ: {
        type: "slider", id: "weight_occ", text: "weight_occ", size:
1.000000, min: 0.000000, max: 1.000000, value: 1/3, step: 0.010000,
unit: ""
    },
    res_grow: {
        type: "slider", id: "res_grow", text: "res_grow", size:
1.000000, min: 0.000000, max: 1.000000, value: 0.2, step: 0.010000,
unit: ""
    },
    comp_ex: {
        type: "slider", id: "comp_ex", text: "comp_ex", size:
```

```
1.000000, min: 0.000000, max: 20.000000, value: 1, step: 1.000000,
unit: ""
    },
    rep_fit_thresh: {
        type: "slider", id: "rep_fit_thresh", text: "rep_fit_thresh",
size: 1.000000, min: 0.000000, max: 1.000000, value: 0.15, step:
0.050000, unit: ""
    },
    res_def: {
        type: "slider", id: "res_def", text: "res_def", size:
1.000000, min: 0.000000, max: 0.500000, value: 0.1, step: 0.010000,
unit: ""
    },
    stress_decay: {
        type: "slider", id: "stress_decay", text: "stress_decay",
size: 1.000000, min: 0.000000, max: 1.000000, value: 0.05, step:
0.010000, unit: ""
    },
    stress_death: {
        type: "slider", id: "stress_death", text: "stress_death",
size: 1.000000, min: 0.000000, max: 10.000000, value: 1, step:
1.000000, unit: ""
    },
    fit_decay: {
        type: "slider", id: "fit_decay", text: "fit_decay", size:
1.000000, min: 0.000000, max: 1.000000, value: 0.1, step: 0.010000,
unit: ""
    },
    move_cost: {
        type: "slider", id: "move_cost", text: "move_cost", size:
1.000000, min: 1.000000, max: 5.000000, value: 2, step: 0.200000,
unit: ""
    },
    main: {
        init_agent_count: {
            pinned: false,
            type: "slider", id: "main:init_agent_count", text: "agent
count", size: 6.000000, min: 0.000000, max: 1000.000000, value:
350.000000, step: 50.000000, unit: ""
        },
        Nest_Population: {
            pinned: false,
            width: 500,
            height: 350,
            autogen: true,
            plotInterval: 1,
            contplot: true,
            XAxis: ["colony_0", "colony_1", "colony_2", "colony_3",
"colony_4", "colony_5", "colony_6", "colony_7", "colony_8"],
            YAxis: [0.000000, 200.000000],
```

```
            caption: "Nest_Population",
            strokes: ["#ff5500", "#d4ff00", "#2aff00", "#00ff7f",
"#00d4ff", "#002aff", "#7f00ff", "#ff00d4", "#ff0026", "#0000ff",
"#ff0000", "#00ff00", "#aaaa00", "#ff00ff", "#00ffff", "#77777777"],
        },
        Stats_Stress: {
            input: "NWorld_vector.stress_out",
        },
        Stats_Stress_graph: {
            pinned: false,
            width: 500,
            height: 350,
            autogen: true,
            multi: false,
            rate: 1,
            lo: __clockParams__.lo,
            hi: __clockParams__.hi,
            contplot: true,
            plotInterval: 10,
            tolerance: 0.001000,
            multipin: false,
            caption: "Stress Stats",
            strokes: ["#00ff00", "#0000ff", "#ff0000", "#aaaa00",
"#ff00ff", "#00ffff", "#77777777"],
            legend: ["Mean", "Mean+Std", "Mean-Std"],
            ylow: [null, null, null],
            yhigh: [null, null, null],
        },
        Stats_Fitness: {
            input: "NWorld_vector.fitness_out",
        },
        Stats_Fitness_graph: {
            pinned: false,
            width: 500,
            height: 350,
            autogen: true,
            multi: false,
            rate: 1,
            lo: __clockParams__.lo,
            hi: __clockParams__.hi,
            contplot: true,
            plotInterval: 10,
            tolerance: 0.001000,
            multipin: false,
            caption: "Fitness Stats",
            strokes: ["#00ff00", "#0000ff", "#ff0000", "#aaaa00",
"#ff00ff", "#00ffff", "#77777777"],
            legend: ["Mean", "Mean+Std", "Mean-Std"],
            ylow: [null, null, null],
            yhigh: [null, null, null],
```

```
        },
        Disease_Tab: {
            input: "NWorld_vector.Disease_Out",
            size: 5,
            base: 0,
        },
        Disease_Tab_graph: {
            pinned: false,
            width: 500,
            height: 350,
            autogen: true,
            multi: false,
            rate: 1,
            lo: __clockParams__.lo,
            hi: __clockParams__.hi,
            contplot: true,
            plotInterval: 10,
            tolerance: 0.001000,
            multipin: false,
            caption: "Disease State",
            strokes: ["#00ff00", "#804d80", "#e64d4d", "#ff9966",
"#1a1a1a", "#0000ff", "#ff0000", "#aaaa00", "#ff00ff", "#00ffff",
"#77777777"],
            legend: ["S", "E", "I", "V", "D"],
            ylow: [null, null, null, null, null],
            yhigh: [null, null, null, null, null],
        },
        Stats_Resource: {
            input: "NWorld_matrix.resource_out",
        },
        Stats_Resource_graph: {
            pinned: false,
            width: 500,
            height: 350,
            autogen: true,
            multi: false,
            rate: 1,
            lo: __clockParams__.lo,
            hi: __clockParams__.hi,
            contplot: true,
            plotInterval: 10,
            tolerance: 0.001000,
            multipin: false,
            caption: "Resource Stats",
            strokes: ["#00ff00", "#0000ff", "#ff0000", "#aaaa00",
"#ff00ff", "#00ffff", "#77777777"],
            legend: ["Mean", "Mean+Std", "Mean-Std"],
            ylow: [null, null, null],
            yhigh: [null, null, null],
        },
```

```
        NWorld_display: {
            caption: "NWorld",
            width: 540,
            height: 540,
            rows: 18,
            cols: 18,
            unit: 30,
            radius: 5,
            useAgent: true,
            useCell: true,
            interactive: false,
            agentSizing: true,
            largeSize: 5,
            smallSize: -1,
            colors: {
                cell:  ["#000000", "#ff0000", "#008000", "#0000ff",
"#00ffff", "#ff00ff", "#ffff00", "#ffffff"],
                agent: ["#000000", "#ff0000", "#008000", "#0000ff",
"#00ffff", "#ff00ff", "#ffff00", "#ffffff"],
            },
            cellValues: [0.000, 1.000, 2.000, 3.000, 4.000, 5.000,
6.000, 7.000],
            numAgentColors: 8,
            numCellColors: 8,
            staticCell: false,
            agentColorAssigned: false,
            hiAgentColor: 7.000000,
            loAgentColor: 0.000000,
            hiCellColor: 7.000000,
            loCellColor: 0.000000,
        },
        Agent_count_spy: {
            caption: "Agent_count",
        },
    },
    Agent: {
    },
    Cell: {
    },
};
```