

COBWEB DOCUMENTATION

COBWEB INTRODUCTION

COBWEB (Complex Organization and Bifurcation Within Environmental Bounds) represents a different approach to studying adaptation. The COBWEB project aims to computationally simulate the adaptation of autonomous agents in a changing environment. Version 2 of the software is general simulation platform that simulates how individual agents decisions affect the capacity for a population to adapt to variability and changes in the environment. COBWEB is very interactive, allowing the user to set many parameters that either affect the agents or the resource production. Parameters such as energy expenditure, resource growth rate, energy requirements for different actions and the energy derived from different resources are all accessed through a user interface that is very easy to use. Once the parameters have been set, the environment is displayed graphically. The “world” created by the program consists of a two-dimensional grid populated by colored isosceles triangles, black squares and coloured squares. The triangles are agents, whose tallest apex indicates the direction they are facing; the black squares are rocks, which are impassable by the agents, and the coloured squares are food or energy resources that grow at a user-defined rate.

COBWEB was developed to explore how components of systems (people, animals, bacteria, etc) adapt to environmental change and environmental variability. The software contains two major components: (1) a population of agents that move, eat and reproduce and (2) the resources that the agents consume, which are also variable in location. The agents make decisions on when to move, where to move and when to eat. In combination with optional communication between agents and memory, the population is able to adapt to some environments and fail to adapt in others. This process can result in the occurrence of surprising events, whether it is a sudden extinction after a long period of stability, a recovery from what appears to be certain extinction or a sudden change in population for no apparent reason. These surprising events can be expected in any complex system, and their occurrence in this simple abstraction of complexity is a very important aspect of COBWEB software.

The COBWEB agents are powered by an artificial intelligence tool called a genetic algorithm, which is a behavioural strategy. The resources, are simulated with another common AI, a cellular automaton. A genetic algorithm is a computer algorithm that through a process of continual selection, random mutation and recombination, evolves a collection of strings of data in a direction determined by the nature of the selection. Agents are not given goals, fears, navigation algorithms, or plans for survival. Rather, their GA's are initialized to random values, and by natural selection (since there are limited resources), agents that are not well-adapted to

survival are eliminated. COBWEB's agents slide around their two-dimensional world in a non-deterministic way, doing whatever their GA's dictate.

The population evolves in a Darwinian sense, as those agents that are best adapted to the environment tend to reproduce most often, and through mutation during reproduction (the rate of which can be controlled by the user), the system can experiment with new strategies. The agents that are not well adapted to survival are eliminated during the simulation. The user has the option to change the environment in small or large increments at any time during the simulation in order to test the adaptability of the population to change.

COBWEB is a great educational tool for students from ages 9 and up. Students can learn how environmental changes affect social and biological systems by adjusting the initial parameters to explore themes such as survivorship, population growth, resource variability and drought, the costs of energy and cheating. Unique workshops can be developed to explore specific issues such as landscape dynamics (how different populations affect the growth of different plants), invasive species, fertilization, social isolation and the emergence of group behaviour. Ultimately, COBWEB's purpose is to create an evolutionary system complex enough to generate emergent and unexpected results – to reveal information and trends that would be otherwise unattainable.

COBWEB Installation:

COBWEB requires the Java Runtime Environment (JRE) to operate. If you do not already have this installed on your computer, it is available to download for free at <http://www.java.com/en/download/index.jsp>.

Installing COBWEB is as simple as saving the file to your desk top. Once this is done, to launch COBWEB, just double-click on the COBWEB.jar icon. It should launch two windows, the one in the background has a grid on it, and the other has a series of tabs labeled: Environment, Resources, Agents and Food Web from left to right.

COBWEB Operation:

1 The Basics:

When COBWEB is launched, it comes up with two windows, one containing the environment display and the other is an interface that allows the user to manipulate the parameters of the program. The Environment tab allows the user to manipulate the physical appearance of the grid. The Resources tab allows the user to manipulate the growth rate, depletion rate and other characteristics of the food resources the program provides for the agents. The Agents tab allows manipulation of parameters that affect each individual agent such as the amount of energy it expends to move and how it may reproduce. Finally, the Food Web tab controls which agents can eat what resources. Additional explanation of the parameters is provided in later sections. The user may manipulate these parameters any way they wish. The four baselines on the website provide some guidance for starting values. If a parameter value is set too high, the program will not run properly.

Once the parameter values are selected, click 'Save' if you want to save the *parameter file*, for future reference or so that the same combination of parameters can be used again without having to re-input all the values. If you do not wish to save a copy of the parameters, simply click 'OK'. In both cases, the parameters window will close, leaving you with the environment window.

The environment visible in the environment window is dictated by your choice of parameters, however the food resources, stones and agents are randomly placed in the environment grid. Food resources are represented by coloured squares, stones are represented by black squares and agents are represented by triangles with coloured dots in the centre. The colour of the dot on the agent corresponds to its 'favourite food', and the colours correspond to the agent and food types in the parameters window as follows: Type 1 = yellow, Type 2 = blue, Type 3 = green and Type 4 = red.

COBWEB will run its simulation indefinitely, but if you would like to control the time step at which the simulation ceases, simply type the number of the time step you would like it to stop into the box left of the 'Resume' button. To begin the simulation, click on the 'Resume' button. The simulation will begin and you will be able to observe your agents moving about in the environment and the food resources grow and disappear. The 'Speed' toggle allows you to control the speed of the simulation runs. The simulation proceeds slower as the toggle is moved farther to the right.

2 Saving the Data:

The COBWEB simulation can provide two different records of the simulation results. The first record is called a *log file*. To get a log file, click on 'Log' under the File menu, and save your file (filename.xls) **before** the simulation is run. Log file output is in spreadsheet format so it is best if you save it with the file extension of a spreadsheet such as Excel (.xls). When the simulation is complete, simply close the COBWEB window and open your log file if you would like to view it immediately. Log files provide the user with the following for each agent type for every time step, as well as totals for all agent types (referred to as a *tick* in the output):

Column of Output	What It Is
Tick	The number of the time step the data was outputted for.
FoodCount	Total number of squares in the grid occupied by that agent's favourite food
AgentCount	Total number of agents for that type present in the grid.
AveAgentEnergy	The average amount of energy an agent of that type possesses (total energy/number of agents).
AgentEnergy	The total amount of energy possessed by agents of that agent type.
Num. Cheat	For the Prisoner's Dilemma game: the count of agents with the 'cheater' strategy.
Num. Coop	For the Prisoner's Dilemma game: the count of agents with the 'cooperator' strategy.

Note For Num. Cheat and Num. Coop: The default setting is 'cheater' so if the Prisoner's dilemma option is not selected, the agent count and cheater columns will be identical. The agents will not play the prisoner's dilemma game, however, so the presence of these 'cheaters' has no effect on the simulation.

If you desire more information about the agents that were part of your simulation, you may also save a *report file*. To save a report file, click on 'Report' under the File menu and save your report (filename.txt) after the simulation is complete. The report file provides the following information for each agent that existed during the simulation:

Agent Information	What It Is
Agent Number	The agents are numbered by the computer starting at zero. This number is an identifier to track the agent throughout the simulation.
Agent Type	Agent Type is reported as 0,1,2 or 3, which corresponds to agent types 1,2,3 and 4 in the parameters window respectively.
Birth Tick	The time step the agent was generated (born).
Birth Type	Output options as follows: Randomly generated (generated at the beginning of the simulation) Asexual, from agent X (where X = agent number above) Sexual, from Mother: X, Father: Y
Death Tick	The time step the agent died. If the agent was still present at the end of the trial, "Agent didn't die" replaces this line.
Direct Descendants	The number of offspring produced by this agent.
Total Descendants	The total number of agents related by birth to this agent.
Sexual Pregnancies	Number of times the agent was pregnant (for sexual reproduction).
Action statistics breakdown:	What It Is
Steps	The percentage of time steps when the agent moved (took a step).
Turns	The percentage of time steps when the agent turned.
Agent Bumps	The percentage of time steps when the agent bumped into another agent.
Rock Bumps	The percentage of time steps when the agent bumped into a rock.
Strategy	Options: 'Tit-for-tat' or 'Probability'. Again, if the Prisoner's Dilemma option is not turned on, the default state is a tit-for-tat strategy with the first move as cooperation.

Report files also provide tallies of the total number of agents dead, alive, sexually produced, asexually produced (including random generation at the beginning of the trial), offspring

produced, average steps taken by an agent and average time steps lived. These are recorded at the bottom of the report file for each agent type.

3 Other Operating Information:

Provided in this section is an overview of the options under the ‘File’ and ‘Edit’ menus of the main window.

3.1 File Menu

The File menu allows the user to create and edit parameter files as well as save output for each file run.

Option	What It Does
Open	Clicking on Open will allow the user to select a pre-saved parameter file (with a .xml extension).
Create New Data	This option allows the data to create a new parameter file as described in section 2.1 above.
Modify This File	Modify this file will allow the user to edit the parameter file in the middle of a simulation. This is useful when wanting to change the growth rate of food or providing new resources to the agents. ‘Keep Old Agents’ and ‘Keep Old Array’ should be selected in the Environment window when using this option to prevent a new simulation environment from being generated in the middle of the trial.
Save	This allows you to save your current parameter settings as an XML file that can be opened and used in subsequent simulation runs.
Report	Report saves a report file, but it must be done after the simulation is complete.
Log	Log saves a log file, but it must be saved prior to running the simulation in order to capture the data.
Track Agent	Track agent will track and record the motion and decisions of a randomly chosen agent, outputted to a text file. This must also be saved prior to running the simulation in order to capture the data.
Quit	Quit exits the program.

3.2 Edit Menu

The Edit menu allows a user to further customize the environment generated randomly. The Edit menu allows the user to add and remove stones, agents and food resources from the grid.

Choosing Select allows you to add and remove stones, agents and food resources one by one.

Clicking on an empty square will add whatever you have selected to that square, and clicking on

a square where it already exists will remove it. If you would like to remove all of something, choose the appropriate 'remove all' option.

4. Starting Your First Experiments:

In order to run comparative experiments, it is important to set up a baseline parameter set which is reasonably stable and predictable. This requires some experimentation in itself. The parameters do not have units. They can be modified to reflect the behaviour of known system or to try and produce other system-wide behaviours of interest. Usually, it is necessary to adjust the parameters beyond the initial settings to achieve a stable outcome (the population doesn't explode or crash over a period of a few thousand time steps). Once the baseline is set, it may be used to determine how changing different parameters affects the environment. Four default baselines are provided on the website.

When exploring how different parameter changes affect the environment, testing values first spread far apart and then narrowing in on your range (responses) of interest is often an effective strategy. It is also a good idea to test extremes, as you may observe your environment switch from one stable state to another. Be creative and enjoy! The strength of COBWEB is in its abstractness; it can be used to simulate anything from climate change to the development of cancer.

ENVIRONMENT VARIABLES

Grid Settings

Name: Width

Type: Integer

Description: The simulation in ComplexEnvironment consists of a 2D grid. The parameter width specifies the width of the grid in units. When this is changed to add more space to a simulation, the new area will be blank (no food growth, stones, or agents) until filled, and when it is decreased, the simulation elements occupying the destroyed area will be also destroyed.

Name: Height

Type: Integer

Description: The height of the 2D grid in units. (See 'Width').

Name: Wrap

Type: Boolean

Description: The edges of the 2D grid normally act as physical boundaries. When Wrap is enabled, agents can move through them, appearing on the opposite side of the grid. It is similar to how a flat map of the Earth works and is equivalent to the agents living on a torus-shaped world.

Prisoner's Dilemma Options

Name: appears as: Prisoner's Game

Type: Boolean

Description: This turns on the Prisoner's Dilemma option. With this option, agents are randomly assigned a propensity to share energy or steal energy. The decision carries consequences similar to the Prisoner's Dilemma game. If both agents opt to share, their total energy is divided in two. If both agents opt to steal, the both lose 50% of their energy. If one agent decides to steal, this agent obtains 50% of the other agents energy.

Name: appears as: Memory Size

Type: Integer

Description: This allows an agent to remember other agents that stole its energy. As this parameter increases, up to a limit of 10, the agent is able to store a longer list of interactions.

Name: appears as: Energy Based

Type: Boolean

Description: This parameter allows an agent to "see" how much energy is possessed by another agent.

Environment Transition Settings

Name: keepOldAgents **appears as:** Keep Old Agents

Type: Boolean

Description: When keepOldAgents is enabled, all agents that are currently in the simulation will remain unless they are in areas that are removed by an adjustment of the "height" or "width" parameters. Enable this when changing some parameters in the middle of a simulation run and then restarting the same simulation. It should be disabled when beginning a brand new run.

Name: spawnNewAgents **appears as:** Spawn New Agents

Type: Boolean

Description: When spawnNewAgents is enabled, new agents will be spawned in random locations according to the value of the parameter "Agents", which is specified individually for each agent type. This should not be enabled when changing some parameters in the middle of a

simulation run and then restarting the same simulation, unless that is the parameter that you want to change. It should be enabled when beginning a brand new run.

Name: keepOldArray **appears as:** Keep Old Array

Type: Boolean

Description: When keepOldArray is enabled, whatever stones and food growth that are currently present in the environment will remain after the data-file is completely processed in addition to whatever agents or stones may be added. Enable this when changing parameters in the middle of a simulation run and then restarting the same simulation. It should always be disabled when beginning a brand new run.

Name: NewColorizer **appears as:** New Colorizer

Type: Boolean

Description: When NewColorizer is enabled, the routines responsible for coloring the agents will be re-initialized. This should always be enabled at the beginning of new simulation, but you may want to disable it if you are only changing a few parameters in the middle of a run and continuing with the same simulation.

Name: keepOldWaste **appears as:** Keep Old Waste

Type: Boolean

Description: When keepOldWaste is enabled, the waste that is currently present in the environment will remain after the data-file is completely processed. Enable this when changing parameters in the middle of a simulation run and then restarting the same simulation. It should always be disabled when beginning a brand new run.

Name: keepOldPackets **appears as:** Keep Old Packets

Type: Boolean

Description: When keepOldPackets is enabled, the packets that are currently present in the environment will remain after the data-file is completely processed. Enable this when changing parameters in the middle of a simulation run and then restarting the same simulation. It should always be disabled when beginning a brand new run.

Colour Settings

Name: NumColor **appears as:** No. of Colors

Type: Integer

Description: NumColor controls the number of basic colors used by the colorizer algorithm. This must be set to at least 2, and the recommended value is 3 (since there are 3 basic color

components). The colorizer algorithm works by selecting a group of agents currently in the simulation, whose size is given by NumColor, using an algorithm determined by ColorSelectSize (see below) at every reColorTimeStep (see below) ticks of the simulation. It then assigns colors to every agent in the simulation as a combination of NumColor basic colors, where the percentage of each basic color added is the same as the percentage similarity of the agent's genetic array to that of the group of specially selected agents.

Name: ColorSelectSize **appears as:** Color Select Size

Type: Integer (\leq NumColor)

NOTE: For the time-being this is an obsolete parameter that will have no effect on coloring algorithm. In favor of an algorithm that selects agents in a way to find maximum uniqueness, Cobweb is currently using one that simply selects the NumColor agents randomly.

Description:

ColorSelectSize controls the algorithm used for selecting the NumColor agents used in the coloring algorithm. The algorithm will search through all permutations of the available agents to find the ColorSelectSize agents such that the maximum similarity of any two of them and the agents already selected is minimized. This process iterates until NumColor total agents are selected. Thus if NumColor = ColorSelectSize, the chosen agents will be optimal. This may however be too complex an operation and take too much time. To speed things up, ColorSelectSize can be less than NumColor and the algorithm will proceed much quicker.

The optimal version of ColorSelectSize will find the three agents that are maximally different and colour them red, blue and green. The suboptimal version will take three agents at random as the initial three agents and colour them red, blue and green.

Name: reColorTimeStep **appears as:** Recolor Time Step

Type: Integer

Description: ReColorTimeStep controls how often the coloring algorithm selects NumColor new agents to represent the basic colors (that determine the colors of each agent in the simulation). Since the genetic makeup of the simulation will change over time, the NumColor selected agents become poorer and poorer as archetypes for determining agent colors as the simulation runs. The archetype agents will be reselected every reColorTimeStep ticks of the simulation.

Name: ColorizerMode **appears as:** Colorizer Mode

Type: Integer (0 or 1)

Description: The ColorizerMode parameter determines in what of two methods the basic colors are combined to give the color of each agent. In mode 0 the colors are less distinct but are slightly better at representing the magnitude of diversity in the population. In mode 1 the colors are more vibrant and distinct but will remain so even if all of the agents in the simulation are close to identical.

Name: ColorCodedAgents **appears as:** Color Coded Agents

Type: Boolean

Description: When ColorCodedAgents is enabled the agents will appear with colors determined by the coloring algorithm instead of basic red.

Random Variables

Name: randomSeed **appears as:** Random Seed

Type: Long

Description: randomSeed controls the random seed that the simulation will start with. All values generated by the generator's algorithm proceed deterministically from the previous value stored within it. Hence if all of the parameters of a simulation are identical, and randomSeed is also the same, the simulation will always unfold in exactly the same manner. When randomSeed is set to 0, Cobweb will initialize it with the current system time. Otherwise, whatever value provided to it is used. Be careful to have randomSeed explicitly set to 0 in each data-file you wish to open unless you want to control the random unfolding (to say, reproduce a particularly interesting session). To find out what value Cobweb gave randomSeed for a particular session, you can use the "save" feature in the file menu (which saves all parameters to a text file that can be read like any other data file through "open"). By inspecting that datafile generated by "save" you can determine the value. Also, simply by opening that file (making sure that it has randomRocks set to 0 and spawnNewAgents, keepOldArray and keepOldAgents all disabled) you can reproduce your simulation session exactly.

Name: randomStones **appears as:** Random Stones

Type: Integer

Description: randomStones indicates the number of stones (physically impassible barriers to the agents that occupy exactly one spot on the grid) to be placed randomly in the grid. If the simulation is stopped and this value is changed, the new stones will added to the ones already in place.

General Food Variables

Name: dropNewFood **appears as:** Drop New Food

Type: Boolean

Description: When dropNewFood is enabled, food will be randomly added to the environment according the values of the parameter “Food” for each food type. Note: each agent type corresponds to a food type of the same index.

Name: **appears as:** Max Food Chance

Type: Float

Description: This parameter is no longer active

The following parameters are dependant on agent type and must therefore be defined for each type. The special parameter “Index” is used to specify the type number for which all following parameters will be associated. In the following description we will refer to the agent type being manipulated as type “X”.

RESOURCE VARIABLES

Name: Food **appears as:** Initial Food Amount

Type: Integer

Description: Determines the amount of food of type X to be randomly dropped in at the beginning of the simulation unless the parameter “DropNewFood” is disabled.

Name: FoodRate **appears as:** Food Rate

Type: Float (0.0 to 1.0)

Description: The percentage chance that a single square of food of type X will be dropped into the simulation at every time step.

Name: FoodGrow **appears as:** Growth Rate

Type: Float (0.0 to 100.0)

Description: The FoodGrow parameter controls the cellular automaton algorithm that controls food growth. Once the food type to be grown in a given square is randomly chosen, the chance of it appearing there is a percentage given by the number of adjacent squares containing food times FoodGrow. Hence if there are 3 adjacent food-containing squares and FoodGrow is 5, the chance of food appearing in that square is $3 \times 5 = 15\%$.

Name: FoodDeplete **appears as:** Depletion Rate

Type: Float (0.0 to 1.0)

Description: FoodDeplete controls the proportion of food of type X in the simulation that is eliminated every DepleteTimeSteps ticks of the simulation. If set to 0.0, no food is eliminated and the deplete system is effectively disabled for this food type.

Name: DepleteTimeSteps **appears as:** Depletion Steps

Type: Integer

Description: (see FoodDeplete above)

Name: **appears as:** Draught Period

Type:

Description: This sets a length of time for draughts.

Name: FoodMode **appears as:** Food Mode

Type: Integer

Description: FoodMode controls the food depletion and famine system parameters FoodDeplete and DepleteTimeSteps. When FoodMode is set to 0 (the default value), both parameters are checked for validity (both positive and FoodDeplete ≤ 1.0), and if they are not in range, they are randomly initialized with values in the ranges (0,1] for FoodDeplete and (0,100) for DepleteTimeSteps. When FoodMode is set to 1, the two parameters are initialized randomly with values in the ranges (0.3,0.5] for FoodDeplete and (20,40) for DepleteTimeSteps. FoodMode 1 only does this random initialization at the beginning of the simulation (or more accurately when a new file is loaded). Randomly changing the FoodDeplete and DepleteTimeSteps according to some more advanced algorithm DURING the simulation would be an interesting experiment, but the current version lacks this functionality.

AGENT VARIABLES

Name: Agents **appears as:** Initial Num. of Agents

Type: Integer

Description: Agents controls how many agents of type X are randomly placed into the environment when the simulation begins. This parameter has no effect if SpawnNewAgents is set to false.

Name: MutationRate **appears as:** Mutation Rate

Type: Float (0.0 to 1.0)

Description: MutationRate controls the probability that each set of output bits will be randomly initialized to a new value during both asexual and sexual reproduction. Currently, all of the output bits that correspond to each input are either completely changed or not irrespective of the individual meanings of each bit in the standard output.

Name: InitEnergy **appears as:** Initial Energy

Type: Integer

Description: InitEnergy is the amount of energy an agent gains from its asexual parent (or mother, in the case of sexual reproduction). The mother is the one that must build up energy in order to breed and the one that initiates breeding. The father is merely a donor of genetic material and may be miles away when the birth actually takes place [if PregnancyPeriod is high for example]. InitEnergy will be the amount of energy the new agent starts with, and it also is the amount of energy that is taken from the parent after birth.

Name: FoodEnergy **appears as:** Favourite Food Energy

Type: Integer

Description: FoodEnergy is the amount of energy an agent of type X gains from food of the same type.

Name: OtherFoodEnergy **appears as:** Other Food Energy

Type: Integer

Description: OtherFoodEnergy is the amount of energy an agent of type X gains from food of any other type than X.

Name: BreedEnergy **appears as:** Breed Energy

Type: Integer

Description: BreedEnergy is the amount of energy that acquired and maintained (through the pregnancy period) by an agent for asexual breeding to occur.

Name: PregnancyPeriod **appears as:** Pregnancy Period – 1 parent

Type: Integer

Description: PregnancyPeriod is the amount of time (in ticks of the simulation) that an agent will remain “pregnant” after conceiving a child agent asexually. This doesn’t however guarantee that an agent will give birth exactly when the period expires. This is because agents only give birth when they move forward on a given time step, since they place their offspring on the square immediately behind, which is guaranteed to be empty.

Name: StepEnergy **appears as:** Step Energy Loss

Type: Integer

Description: StepEnergy is the amount of energy consumed when an agent moves forward successfully (without bumping into a barrier such as a wall or another agent).

Name: StepRockEnergy **appears as:** Step Rock Energy Loss

Type: Integer

Description: StepRockEnergy is the amount of energy an agent wastes by driving itself into a rock or edge of the grid area.

Name: TurnRightEnergy **appears as:** Turn Right Energy Loss

Type: Integer

Description: TurnRightEnergy is the amount of energy an agent consumes turning right.

Name: TurnLeftEnergy **appears as:** Turn Left Energy Loss

Type: Integer

Description: TurnLeftEnergy is the amount of energy an agent consumes turning left.

Name: MemoryBits **appears as:** Memory Bits

Type: Integer

Description: MemoryBits is the amount of memory bits an agent will use in its genetic array. It adds to both the number of input bits (for which the total size of the array doubles with each addition) and to the number of output bits. The memory works by taking whatever output the agent produces and using part of that for its next input, a kind of “feed-back” mechanism. Setting MemoryBits to 0 effectively disables agent memory.

Name: CommSimMin **appears as:** Min. Communication Similarity

Type: Float (0.0 to 1.0)

Description: CommSimMin is the minimum similarity an agent must have to another agent (of the same type, since agents don't communicate or breed across types) in order for communication to occur. If set to 0, there is no limit on the communication. It might be important to restrict communication between dissimilar agents since they, in a sense, speak a different language.

Name: StepAgentEnergy **appears as:** Step Agent Energy Loss

Type: Integer

Description: StepAgentEnergy is the amount of energy expended by an agent when it drives itself into another agent. This type of contact is of course necessary for agent communication and sexual breeding, so this value shouldn't be set too high.

Name: CommunicationBits **appears as:** Communication Bits

Type: Integer

Description: Along with the parameter MemoryBits, CommunicationBits is the only way for the user to influence the size and composition of the genetic arrays of agents in the simulation. CommunicationBits, much like MemoryBits adds to both the amount of input bits and the amount of output bits. Every time-step, if an agent has received a communication from another agent, the message of that communication (an integer x , $0 \leq x < 2^{\text{CommunicationBits}}$) is used for that agent's next input. If no communication is received, the message is 0. Since the size of an agent's genetic array grows exponentially with the amount of input bits, it is important to keep this number reasonably low. Setting CommunicationBits to 0 effectively disables communication.

Name: SexualPregnancyPeriod **appears as:** Pregnancy Period – 2 parents

Type: Integer

Description: SexualPregnancyPeriod is identical to PregnancyPeriod (described previously), except it used for sexually induced pregnancies.

Name: BreedSimMin **appears as:** Min. Breed Similarity

Type: Float (0.0 to 1.0)

Description: BreedSimMin, like CommSimMin, introduces a restriction on sexual breeding that only allows agents that are sufficiently similar in genetic makeup to engage in the process. If set to 0, there is no restriction.

Name: SexualBreedChance **appears as:** 2 parents Breed Chance

Type: Float (0.0 to 1.0)

Description: If all of the requirements for sexual breeding are met SexualBreedChance is the probability that sexual breeding will take place. If set to 1.0, it will always happen, set to 0, it is disabled.

Name: ASexualBreedChance **appears as:** 1 parent Breed Chance

Type: Float (0.0 to 1.0)

Description: If all of the requirements for asexual breeding are met ASexualBreedChance is the probability that asexual breeding will take place. If set to 1.0, it will always happen, set to 0, it is disabled.

Name: **appears as:** Aging Mode

Type: Boolean

Description: Choosing 'True' will turn on aging. This means that the energy costs of the agent's actions increase over time so that at a certain point the energy costs are infinite and the agent must die. 'False' is the default value and turns aging off. When aging is turned off, an agent may live an indefinite length of time: as long as it gathers enough energy to carry on its life functions. The energy increase is defined by the inverse tangent function.

Name: **appears as:** Aging Limit

Type: Integer

Description: This number sets the point of the asymptote for the inverse tangent function. At this point, the energy cost of living is infinite and the agent must die. It is highly unlikely that agents survive to this number of time steps because the energy costs are also very high in the time steps leading to this point.

Name: **appears as:** Aging Rate

Type: Float

Description: This controls the slope of the inverse tangent function, and therefore the rate at which energy costs increase. The higher the value, the faster the aging rate (the slope of the function becomes steeper faster).

Name: **appears as:** Waste Mode

Type: Boolean

Description: This parameter turns waste production on or off. Set it to false to prevent the agents from producing waste.

Name: appears as: Step waste Energy Loss

Type: Integer

Description: This determines the amount of energy an agent loses by stepping into waste. It behaves in a way similar to stepping into a stone.

Name: appears as: Energy Gain Limit

Type: Integer

Description: This threshold value represents the amount of energy an agent should gain to produce waste. If the value is set for 20 units, for example, then the agent would automatically produce waste after gaining 20 units of energy (if waste mode is set to true and space is available for waste production).

Name: appears as: Energy Usage Limit

Type: Integer

Description: This parameter links agent activity to waste production. It determines the amount of energy to be used by an agent (in actions like movement) before the agent can produce waste. A higher Energy Usage Limit would produce waste less frequently.

Name: appears as: Waste Half-life Rate

Type: Float

Description: This controls the rate of decay of waste. This value determines how long the waste will persist in its current location. At each time step, the following equation is used to calculate the new value for each unit of waste:

$$N(t) = N_0 * e^{-ht}, \quad \text{where } h = \text{Waste Half-life Rate}$$

A half-life of zero causes no decay. A higher half-life value would cause waste to disappear in less time steps.

Name: appears as: Initial Waste Quantity

Type: Integer

Description: This controls the quantitative value of waste as it is produced. Whenever an agent produces waste, it will have the initial value defined here. In combination with Waste Half-life Rate, Initial Waste Quantity determines how long the waste will persist in its current location.

Name: appears as: PD Tit for Tat

Type: Boolean

Description: This turns tit-for-tat strategy on or off. This field is only used once the PD option is enabled. Whenever this parameter is enabled, probability is ignored. The first move for a tit-for-tat player is cooperation.

Name: `appears as:` PD Cooperation Probability

Type: Float

Description: This determines the probability of this agent cooperating in a PD game. If tit-for-tat is disabled, the agent uses this probability to determine bias towards cooperation or defection. The values for this parameter range between 0 (always defects) and 100 (always cooperates).

Name: `broadcastMode` **appears as:** Broadcast Mode

Type: Boolean

Description: Setting to 'True' will turn on broadcasting. This means that whenever an agent decides to initiate a broadcast (if food is found for example), the broadcast will be sent successfully. 'False' is the default value. When broadcasting is turned off, agents of that type will not be able to send or receive broadcasts.

Name: `broadcastEnergyBased` **appears as:** Broadcast Range Energy-Based

Type: Boolean

Description: Setting to 'True' will set the broadcasting range according to the energy value of the agent that initiated the broadcast. When it is turned off, the range will be determined according to the fixed value (`broadcastFixedRange`) set by the user.

Name: `broadcastFixedRange` **appears as:** Broadcast Fixed Range

Type: Integer

Description: This parameter enables the user to set a fixed range for broadcasting packets. If `BroadcastEnergyBased` is set to false, this parameter will be used. The minimum range is 1. To make sure the broadcast covers the whole environment, set this parameter to a value bigger than the grid size (e.g. 100 for a 20x20 environment).

Name: `broadcastEnergyMin` **appears as:** Broadcast Minimum Energy

Type: Integer

Description: This value determines the minimum amount of energy required before an agent can broadcast. Typically this value should be smaller than the cost of initiating a broadcast (`broadcastEnergyCost`).

Name: `broadcastEnergyCost` **appears as:** Broadcast Energy Cost

Type: Integer

Description: This is the amount of energy consumed when an agent broadcasts a message. The minimum value for this parameter is 0.