Primordial Soup (c) 2002
By Orrery Software

PSoup Help System Texts

Table of Contents

Primordial Soup (c) 2002
By Orrery Software

PSoup Help System Texts
TABLE OF CONTENTS

Primordial Soup (c) 2002
By Orrery Software

PSoup Help System Texts

Welcome

Welcome to the help system for Primordial Soup.

This document is a re-formatted collection of the articles contained in the hypertext-based help system provided with PSoup Version 1.0. The purpose is to act as an alternative reference source, for those users who prefer to do their reading away from the computer screen. For publication via paper, the hypertext links have been removed. However, extensive finding aids have been provided in the form of a table of contents and an index. The organization of topics follows the layout in the table of contents of the on-line help system.

The help system is set up, not only to help you navigate around the PSoup application, but also to help you better understand the terminology that is peculiar to PSoup, and to help you understand the applicability of PSoup terminology and processes to analogous biological terminology and processes.

The help system offers two ways to enter. Entry via the primary help page gives you fast and direct access to technical descriptions of menu selections, dialog panels and toolbars, as is typical for all windows applications. Entry via the Table of Contents of the help system also gives you direct access to a collection of articles called the 'Theory and Practice' library. These articles attempt to situate the PSoup implementation in terms of analogous biological systems and also in terms of similar classes of computer systems.

To invoke the help system at any time, left-click the help button in the ‘Help’ toolbar at the top right of the PSoup screen (the button with the yellow question mark) or press the ‘F1’ key at the top left corner of your keyboard.

Getting Started

It is presumed, at this point, that you have PSoup installed on your computer and ready to go. If not, see the README.TXT file found in your installation package.

Once you have PSoup installed and loaded, it is very easy to set the evolutionary processes in motion. There are three easy steps:
Step 1 - Use the 'Scenario' command in the main menu to select a suitable scenario. You can use the mouse, or you can use ALT-key combinations, as indicated by the small underscores in the various commands and scenario names.
Step 2 - Some scenarios take a while to load, so wait for it. Eventually, a screen called 'About this scenario' will pop up. Read the screen. Use the mouse to scroll down through all of the material. It has several sections such as purpose, references, things to note, and exercises. The references are to detailed articles found in this help system. This information, in the references, may be needed to understand the technical workings of the scenario, or to master the analytical tools applicable to the scenario, but it is suggested that the references be accessed after you have watched the scenario at least once. Under things to note you will learn how the scenario is designed, and what to watch for as the scenario unfolds. When you have read the panel in the screen 'About this Scenario', then press the OK button, and the scenario will appear.
Step 3 - Start the run. To do this, either (1) click on the 'Go SloMo' button, or (2) use the 'Evolution, Go SloMo' command, or (3) Use the 'Evolution, Go' command.

The 'Go SloMo' command is controlled by a governor which prevents more than 17 moves per second, to make the motion intelligible on very fast computers. The 'Go' command has no speed controls on it.

To access the full 'Help System' including all of the references, click on the yellow question mark, and check the 'Contents' tab.

Note - Not all help text is in the help system. From time to time pertinent information is displayed in pop-up screens that appear when needed. The 'About this scenario' screen is an example. To access this information at any time, use the 'Help - About This Scenario' command, or press the 'Sc' button.

PSoup Theory and Practice

Basic PSoup

A PSoup Primer (PSoup Theory and Practice)

The Basics - The following description applies specifically to a Level 1 bowl of PSoup, and applies with some elaboration, to higher levels. A student should be sure that (s)he understands these basic concepts before moving on to the higher levels of PSoup.

The PSoup – The action in this application will happen in a logical bowl of 'PSoup'. The choice of the word 'PSoup' is intentional. PSoup stands for "Primordial Soup", and is named after the pre-Cambrian waters in which life first developed over a period of some 2.5 billion years. We join the story somewhat after the time when the first autotrophs (photosynthesizing algae) appear and just at the moment when the first holotrophs (algae-eating protozoa) appear, and the biogenetic processes are kicking into high gear. However, PSoup should not be viewed as an approximation or simulation of the real-world primordial soup. PSoup is, rather, a logical entity existing in a logical world where logical evolutionary forces are drawn into play. When PSoup was being designed, there was very little effort expended to replicate the laws of physics or chemistry that would be necessary to simulate what happens in a real-world primordial soup in a convincing fashion. And this demonstration, this PSoup, is intended to be not just convincingly argumentative (in favor of the validity of the concepts in the theory of evolution) but to be absolutely demonstrative of evolutionary processes. Therefore, it must be clear that 'PSoup' is a logically abstracted system which stands on its own as a theater in which real evolution occurs.

Scenarios - PSoup is able to demonstrate a variety of different environments (or configurations) in which evolutionary processes operate. Each instance of a bowl of PSoup, whatever the configuration, is called a scenario. PSoup comes with a number of canned scenarios, but also offers the ability for the user to develop and save his/her own scenarios. The 'tinker' commands allow the user to modify the configuration of the bowl of PSoup, including the rules of chemistry used behind the scenes, as well as the genetic structure of each of the denizens of the bowl. In Level 1, the tinker commands have limited capabilities. By the time a student achieves Level 6, the tinker commands are fully enabled.

Cells – Distance, location, or space in PSoup is not continuous, but rather, it is discrete. That means, there is a smallest indivisible unit of distance. The smallest unit of two-dimensional space is a one-by-one square called a cell. If you view the denizens of PSoup to be analogous to single-celled organisms, then a cell might correspond to a real-world distance of, roughly, a micron. However, if you view the denizens of a bowl of PSoup to be analogous to, say, elephants or whales, then a cell might be viewed as being several miles in width. A bowl of PSoup is a rectangle composed of square cells laid side by side completely filling the rectangle.

Nutritive Mud – A bowl of PSoup is considered to have a supply of nutritive logical mud equally distributed among all of the cells. In this implementation of PSoup, every cell has nutritive mud. In some scenarios there is a patch of cells in the middle of the bowl of PSoup (called the oasis) which is super-nutritive. By default, a bowl of PSoup is a closed energy system which holds within itself a fixed total amount of energy. This energy moves from mud to lower entities to higher entities via logical processes which correspond distantly to natural processes such as movement, feeding, reproduction, and death. It is possible to convert any bowl of PSoup into an open energy system in which there is a continuous supply of energy coming in, and a continuous leakage of energy out again, in closer simulation of a real-world ecosystem. In a closed energy system, the mud stores unused energy before it is distributed as algae. In an open energy system, the energy is put into the mud before it is distributed, and it dissipates when used.

Algae Placement – Each cell is able to contain one or more logical entities. Two types of logical entities exist in a bowl of PSoup: algae and bugs. At startup, the algae are scattered randomly throughout those cells containing the nutritive mud. Usually, this includes all cells in the bowl of PSoup. Each cell can have

only one unit of algae in it. Each unit of algae has a fixed number of units of energy (40 units). A unit of algae is called a colony of algae, named after the colonies of blue-green algae that were common in the Pre-Cambrian era. The energy held by a unit of algae does not change over time (see 'clocking' below). As algae are distributed, energy for their placement is drawn from the nutritive mud. When all of the energy supply in the mud is exhausted, the distribution of algae is complete. In a closed energy system, the total amount of energy in the system (mud, algae and bugs) does not change throughout this process. In an open energy system, the total energy in the system changes over time depending on the difference between the inflow of energy and the outflow of energy.

Bug Placement - At startup of each scenario, a pre-determined number of bugs are scattered randomly throughout the bowl of PSoup according to pre-determined rules. These bugs can be placed in any cell, whether or not it contains nutritive mud. A bug may occupy the same cell as is occupied by a colony of algae, but it may not occupy a cell already filled by another bug. This is a seed population of bugs. At startup of the scenario, each bug has a pre-determined amount of energy, all bugs having the same amount. As each bug is added, the energy contained in the bug is added to the total energy of the system. When the distribution of the bugs is completed, the total energy in the system is computed (mud, algae, bugs) and fixed. From this point forward energy is tracked. In a closed energy system, it never changes. In an open energy system the rate of inflow of energy is either fixed at a steady rate or varies over time. The rate of dissipation of energy from the system is determined by the consumers, the bugs.

Clocking – Life in a bowl of PSoup evolves over time. In this logical PSoup, time is not continuous, but, rather, it is discrete. That is, there is a smallest indivisible unit of time which is arbitrarily called a PSoup second. It is also called a 'tick', as in a tick of a clock. Larger units of time are the minute (60 PSoup seconds), hour (60 minutes), and day (24 hours). These larger units of time are used to make accounting for large spans of time easier, but the application moves forward one tick at a time. During a tick of the clock, the application performs two functions, one of which has many sub-functions. The following list briefly describes each (sub-)function in the order they happen. Following paragraphs describe the details of each (sub-)function. Here are the (sub-)functions performed by the application at each tick of the clock:

- Pre-Per-Second Processing (initialization of all parameters)
- LOOP: For all bugs currently alive, allow each bug to perform the following life functions:
    - Pre-Per-Bug Processing;
    - Sensing;
    - Movement;
    - Feeding;
    - Reproduction;
    - Death; and
    - Post-Per-Bug Processing (Distribute one algae, if enough energy exists in the mud); and
- Post-Per-Second Processing (Distribute fresh algae, update display panels).

Life Functions – For each tick of the clock, each bug has the opportunity to complete five life functions which are sensing, moving, feeding, reproducing and, optionally, dying. Energy may be transferred or consumed by each. The actions taken under subsequent functions within this tick may be affected by the results of the previous functions.

Sensing – In levels 1-3 of PSoup all bugs are totally senseless. Of course, the computer program has access to all knowledge of the logical structure of the bug, and of the entire bowl of PSoup. This information is NOT available to each bug as it makes its moves about the bowl of PSoup in search of algae. In higher levels, as the bugs evolve senses, their ability to sense their surroundings and react to them is strictly driven by a random number generator as interpreted by their genes. The bugs have evolved these abilities through, again, natural selection, and no guidance or interference in the processes of natural selection was undertaken.

Movement – Each bug makes a decision to move into a neighboring cell, then moves. The decision about into which cell it should move is based entirely on information coded in it's genes. In levels 1-3, it cannot sense the contents of any of the cells around it. A neighboring cell is one of the eight cells which surround the cell in which the bug exists. The decision process is described in more detail below under 'genetic

coding'. Movement costs energy at a rate of 3 units per cell (an amount which may be modified under genetic control). This is called the Energy Per Move parameter, or EPM. For each tick of the clock, each bug moves one cell and expends 3 units of energy. This energy is removed from the store of energy in the bug, and is returned to the nutritive mud from whence it came (or, in the case of an open energy system, dissipated from the system). It is not stored in the current cell, but rather goes into a generic store of energy in the mud, from where it is re-distributed as algae in the last process. If the bug tries to move into a cell occupied by another bug, or it tries to move out of bounds past the edge of the bowl of PSoup, it is bumped back into its original cell or a neighboring cell. For example, if a bug at the left edge of the bowl tries to move down and left, it will move down but not left, as it is prevented from going out of bounds.

Eating – In this process, if the bug is not already full (i.e. has more than 1500 units of energy), a bug eats any colony of algae found in the cell that it has just occupied. This threshold of 1500 Nrg units is called the Maximum Energy Per Bug, or EPB parameter. When a bug eats a colony of algae, the algae is removed from the cell, leaving only the nutritive mud that was there originally. The energy from that colony of algae is added to the energy store of the bug that ate it. Note that:
- This bug has just occupied this cell, due either to a move, or to being bumped back into it;
- This cell may have been just vacated by another bug, which may have eaten the algae colony there;
- The cell that this bug has just vacated may or may not contain a colony of algae, depending on whether (a) this bug was not hungry during the feeding cycle of the previous tick of the clock, or (b) a fresh colony was deposited there since the bug ate one (in another bug's turn, or in the 'distribute fresh algae' process at the end of the previous tick of the clock).

Reproduction – If the bug meets certain criteria, it will reproduce by fission (i.e. mitosis). That is, it will split into two daughter bugs, and the mother bug will cease to exist. There are two types of criteria that a bug must meet to be able to reproduce. First, the bug must be mature. In other words, it must be at least 800 ticks of the clock old before it will be allowed to reproduce. This parameter is called the Reproduction Age Threshold, or RAT parameter. Second, the bug must be healthy. In other words, it must have at least 1000 units of energy before it will be allowed to reproduce. This is called the Reproduction Energy Threshold, or RET parameter. The bug must meet both criteria to be able to reproduce. The mother's energy is split evenly between the two daughter bugs. At the moment of reproduction, each of the daughter cells suffers a genetic perturbation, i.e. a random mutation of one of its genes. In Level 1, there is a 100% probability that each daughter bug will suffer such a mutation. See 'genetic coding' for more details on the mutation process. Each daughter is 0 ticks old and must age and gather energy if it is to eventually reproduce itself. Each daughter has the orientation of its mother. (See below for more about orientation.)

Death – If the bug meets certain criteria, it will die. That is, it will cease to exist, and all remaining energy, if any, will be returned to the nutritive mud for later re-distribution as colonies of algae (or, in an open system, will be dissipated and lost from the system). There are two types of criteria that may cause a bug to die. Breach of either threshold will cause death. First, the bug will die if it is more than 1600 ticks of the clock old. This is called the Death Age Threshold, or DAT parameter, Second, the bug will die if it has less than 30 units of energy and therefore cannot sustain itself. This is called the Death Energy Threshold, or DET parameter.

System-wide Parameters – Six key parameters have been mentioned which are common for all bugs system-wide, in Level 1.

- DAT - Death Age Threshold	1600 PSoup seconds
- DET - Death Energy Threshold	30 Nrg units
- RAT - Reproduction Age Threshold	800 PSoup seconds
- RET - Reproduction Energy Threshold	1000 Nrg units
- EPM - Energy Per Move	3 Nrg units
- EPB - Maximum Energy Per Bug	1500 Nrg units

Starting at Level 2, these are coded as genes in a group of genes called the regulatory genes (a type 2 chromosome), are individual to each bug, and are subject to evolutionary pressures.

Note the energy in a bowl of PSoup is measured in Nrg units; 1 Nrg unit being equivalent to 1/40th of a colony of algae.

Distribute Fresh Algae – The nutritive mud generates a new crop of algae in cells previously void of algae, to a maximum of one colony per eligible cell. Only those cells with nutritive mud can generate algae. When the energy in the mud is exhausted (i.e. falls below 40), no more algae can be distributed. Each colony contains the same amount of energy as was used in the original distribution of algae, i.e. 40 units each. Note that algae may be distributed into a cell containing one or more bugs. Those bugs will move out of that cell on the next tick of the clock, leaving it behind. The algae will be available for the first bug that moves into the cell on the next tick of the clock.

Genetic coding of the Transport Genes – Each bug has nine movement control genes called the Palmiter genes. Not coincidentally, if you include the cell in which the bug currently resides, there are nine cells within a distance of one cell of the bug. The Palmiter genes pre-wire a bug's decision-making process with respect to movement, and, ultimately, its movement behavior. Each of the nine genes contains two numbers, a base number and an exponent. The strength of the gene is computed when you raise the base to the power of the exponent. When we talk about the strength of a gene, we always mean this computed value. In PSoup, a gene always expresses itself as a strength.

At level 1, gene strengths may be any positive real number which can be expressed as a power of 2. The sum of the strengths of all Palmiter genes is, therefore, also a positive real number, which, for this discussion, we shall call '**S**'. Suppose we number and name the nine Palmiter genes as '**gene0**' through '**gene8**' (**gene0** is the first gene, **gene1** is the second gene, and so on), and suppose, further, that the strength of each gene is named '**s0**', etc. up to '**s8**' respectively. We can view **S** (the sum of the nine strengths) as a segment of a number line, and we can view the strength of each gene, as a segment of a number line. We can lay the **sn** segments in order, end-to-end, along the **S** segment, **s0** starting at zero at the beginning of **S**, and the top end of **s8** coinciding precisely with the top end of **S**. We then have a tiling of the line segment **S** such that each and every positive real number in the segment **S** is mapped to one and only one gene. If we then use the computer's pseudo-random number generator to generate a positive real number '**p**' between zero and **S**, and use the above mapping to determine which gene is associated with that number, we have a process by which we can randomly select one of the nine Palmiter genes based on their relative strengths. This process is used to determine which gene will express itself and control the bug's movements on any given turn. The execution of this process is called a decision, although it is clear there is no free expression of will when a bug makes such a decision. All such decisions are made via the interaction of the bug's Palmiter genes and the pseudo-random number generator. The probability that any given gene will be selected for expression is the strength of that gene over the sum of the strengths of all Palmiter genes. This rather elegant concept was invented by Dr. Michael Palmiter, and he has graciously agreed to let these genes bear his name in PSoup.

Bug Movement mechanics – Each bug has a head. The bug also has an orientation such that the head is pointing towards one of the eight neighboring cells. There are therefore exactly eight possible orientations, each 22.5 degrees apart from each other. This distance between orientations is called a turn. When a bug has a chance to move, it goes through a simple process: first it decides to rotate its orientation by anywhere from 0 to 7 turns; second, it rotates by that many turns; third, it steps forward into the cell that it is now facing. It occupies the new cell and maintains the orientation that brought it into the cell. It then eats whatever algae may be in the new cell and thereby increases its energy store.

Bug Movement Decisions – A bug makes a decision to rotate through a number of turns by randomly selecting one of its nine transport control genes. Eight of these genes control turning. One of these genes, called variously **gene8** or the **Stand Still** gene, represents a decision not to turn and step forward, but rather to simply stand still in the cell it currently occupies. When a Palmiter gene has been selected for expression, the bug follows the instructions coded thereby. The coding of the genes of the bug affect its decisions about how many 'turns' to rotate to the right before stepping forward. Each gene represents a number of turns, and, if allowed to express itself, will cause the bug to rotate exactly that many turns. During movement, only one of the nine genes will express itself, and it will cause the bug to turn exactly

that many turns to the right. Note that, in Level 1 of PSoup, the **Stand Still** gene is inert and plays no role in a bug's movements. It is introduced to the user in Level 3.

Here are some examples. Each gene is named. For discussion, let's call them **gene0**, **gene1**, **gene2**, **gene3** etc. up to **gene8**. Each gene causes the bug to turn a number of turns to the right or to stand still, as described above. **Gene0**, if expressed, causes the bug to rotate by zero turns. **Gene7**, if expressed, causes the bug to rotate seven turns to the right.

- If all genes have a value of 1, which is true for 'Pristine' bugs at startup, then all genes have an equal probability of expressing themselves, and the bug will express all genes with equal probability. Its movements will be erratic and unorganized;

- If **gene0** has a strength of 8, and all other genes have a strength of 2, they will have an aggregate gene strength of 14. The total is 14 + 8 = 22. The dominant gene will then determine roughly 36% of the moves, computed by 8/22 = 0.36.

- If in a mature bowl of PSoup **gene0** has a strength of 256, and all other genes have a strength of 2, they will have an aggregate strength of 14. The total is 14 + 256 = 270. The dominant gene will then determine roughly 95% of the moves.

Mutation of Palmiter genes - As stated above, each Palmiter gene consists of two numbers: a base number which, by default, is equal to '2', and an exponent which, by default, is equal to '0'. The strength of a gene is computed when you raise the base to the power of the exponent. The exponent ranges in value between -150 and +150. A gene may therefore be coded as a real number greater than zero. The minimum strength is (2^-150). The maximum strength is (2^+150). At startup for the basic scenario 1A, all Palmiter genes of all bugs have a genetic strength of exactly one (2^0). For each bug, then, the sum total genetic value of all eight active genes is eight (remember, **gene8** is inert in Level 1). When a mother bug successfully reproduces, each daughter bug has a 12.5% probability of mutation during fission. When a daughter bug mutates, one of its Palmiter genes is randomly selected, and then the component of this gene which is used as the exponent is randomly adjusted either upwards or downwards by a value of exactly one. Genes always therefore have a positive real strength.

Standard Mutation - More generally, every gene has a strength. A standard mutation causes a change in the value of the log of the strength, using a base 2 logarithm, of exactly one, either upwards or downwards.

Distribution of Strengths - For those students with a rudimentary knowledge of the behavior of random processes, it is easy to see that the values of the exponents should cluster around zero (the default starting point) with a normal distribution around it. Over time, the width of the distribution curve should broaden, as the accumulation of random shifts drive a few genes to extreme values. But the average value of the exponent, when taken across all Palmiter genes of many bugs, should remain at zero. However, if there is some kind of selection process in place which favors some movement patterns over others, then those bugs with less-favored movement patterns (a less-favored pattern of Palmiter gene strengths) will be selected against, and the distribution of gene exponents should shift towards the favored values.

When considering such a 'random walk' you might think that it is not possible to attain high exponent values until after many generations. In reality, since natural selection strongly favors certain patterns of movement over others, some Palmiter genes obtain very high exponent values indeed. This is symptomatic of the very high degree of evolutionary pressure that is placed on the Palmiter genes in PSoup.

PSoup's Edges – The edges of a bowl of PSoup are a major characteristic of the environment in which our bugs evolve. Bugs can interact with nutritive mud (indirectly through the algae), algae (by eating it), other bugs (by bumping into them or blocking their passage), and edges (by bumping into them). That's it. PSoup has a VERY simple environment, when compared to natural environments. PSoup offers a student the ability to control the way bugs interact with edges, to better examine how the edges affect the evolution of bugs. There are two modes of interaction, defined by the 'wrap' toggle. Wrap may be turned on or off.

- **Wrap On** - When wrap is turned on, the edges of the bowl of PSoup are wrapped around logically to touch each other. The cell at the very top of any column of cells is considered to be just below the cell at the bottom of the same column. Similarly, the cell at the extreme right of any row of cells is considered to also be immediately to the left of the left-most cell. From a topological point of view, when wrap mode is

'on', a bowl of PSoup is not a rectangle, but rather the surface of a torus. A student may choose to turn wrap on to eliminate any edge effects, such that a bug's movement patterns are not disturbed by arbitrary nearness of the edge of the PSoup.

- **Wrap Off** - When wrap is turned 'off', then a bug is unable to see or avoid the edge, and unable to penetrate it. Bugs which continually drive themselves into a corner and bump their little heads on the edge until they exhaust themselves do not reproduce. A student may choose to turn wrap off to examine the effects of such limitations on the evolutionary output.

Take a Swim in the Soup

Let's take a swim in a basic bowl of PSoup. Pour yourself a bowl of 1A (Scenarios menu, Level 1, (A) - The Basics), and we'll dive in and swim, one lap at a time (pun intended).

Now, before you start, it is suggested that you shrink this help page down to roughly 1/4 of the screen size. To do this: (a) Be sure the help page is neither maximized nor minimized. There are three buttons at the top right corner of the help page. There are also three at the top right corner of the PSoup program. You are in fact running two programs at the same time, PSoup, and the PSoup Help System. (You will see this if you look at the status bar at the bottom of the screen. There are two broad buttons there that you can use to switch between the two programs.) In the top right corner, click the middle button for the PSoup Help System. If the Help system now fills the entire screen, click it again. (b) Change the width. Use the mouse to point to the left edge of the Help System window. When a two-headed arrow appears, press and hold the left mouse key, drag the edge of the window to the right. (c) Change the height. Point to the bottom edge of the Help System window, and when the two-headed arrow appears again, press and hold the left key and drag the bottom edge upwards. (d) Reposition the window. Point the mouse to the middle bare area of the title bar at the top of the Help System window, press and hold with the left mouse key, and drag the window down into the bottom right corner of the screen. You should now be able to see most of the PSoup screen, and the Help System. As you work with PSoup the Help System will temporarily disappear behind PSoup, but you can pop it to the front again by clicking on the appropriate bar in the Windows status bar at the bottom.

Before we start, note the situation. We have a bowl of PSoup measuring 25 cells by 10 cells (check the legend panel). We have four pristine bugs. A Pristine bug is one for which all Palmiter genes have a strength of 1 (check the Current bug genetic profiles). Pristine bugs are grayish, and all legs are of equal length (all directional rotations have equal probability). Each bug has a name and a number. It's name and number are combined to produce a full name. The name is analogous to a family name. All daughters will carry the same name. The number is analogous to a given name. Each bug has a unique number. Use the 'Advance by 1 bug' button to click through the four bugs. Watch the Current Bug genetic profile to see their names. The energy loading factor in this scenario is 100%, and edge wrap is turned off (check the legend panel). Don't worry about edge wrap; it will be explained in scenario 1C.

First Lap About the Bowl

Now, run scenario 1A for roughly 1 PSoup minute and 50 PSoup seconds. To do this, use the Evolution toolbar and press 'SloMo'. This activates the slow motion advance (limited to 17 Moves per bug per second). Watch the status panel and, when it reaches roughly 1 minute and 40 seconds press 'Stop'. Then press the 'Advance by one second' button until you are at 1 minute and 50 seconds, give or take a few seconds.

Check the Family Tree. Each of the four bugs is represented by a vertical bar graph which grows with each second that passes. For the first two minutes, you can see the growth, second by second. After two minutes, this graph is shrunken vertically, and the growth is shown every four minutes. When we pass the two-minute post on our swim, you will see this shift in the family tree display.

Check the Energy Chart. To activate the energy chart, press the 'Nrg' button, or use the 'Options' menu to 'Display the Energy Chart'. The chart will appear in place of the family tree panel. The energy chart provides us with two graphs overlaid on top of each other.

The first is a line graph which tracks the number of bugs living in the bowl of PSoup as time progresses. Note that, in the chart, there is a black horizontal bar at [4]. This shows that we started with, and still have, four bugs in our bowl of PSoup. The square brackets indicate scale numbers associated with the 'number of bugs' line graph. The smallest number on the graph is [0], and the largest is [20]. The horizontal bar is unlabeled, but the value is [4].

The second graph is a stacked bar chart. Each vertical bar in the background represents one PSoup second, and provides us a snapshot of the PSoup Energy System. As per the legend, the colored portions of the bar represent energy in the nutritive mud, in the algae, and in the bugs, the total height of the bar indicating the total energy in the bowl of PSoup as a system. The numbers outside of the brackets provide the scale for these bars. You will note that we started with a substantial amount of energy in the nutritional mud. This is energy that had not been distributed as algae, and was therefore unavailable for consumption by the bugs. Within the first 20 PSoup seconds or so, this energy was distributed as algae.

You will also note that the bugs started with a small amount of energy (800 units each, for a total of 3200 units) equal to just under 1/4 of the total energy in the system. They have scurried around, and, somewhere around the 1 minute mark, managed to fill themselves up with roughly 1500 units of Nrg each, for a total of 6000 units. In the real world, energy is measured in ergs. In PSoup, energy is measured in Nrg units.

Now, activate the status panel and the energy chart (by pressing the appropriate buttons in the toolbars, or using the Options menu), then advance the scenario one second at a time (by using the '1Sec' button, or the 'Evolution' menu) until you get to 2 PSoup minutes and 1 second.

In the energy chart, there are now three vertical bars. One is a snapshot at second 0; one is a snapshot for minute 1, second 0, and the third is a snapshot of the energy system for minute 2, second 0. From now on, the energy chart is updated once every minute. Now, switch to the 'Family Tree' (by pressing the 'Tree' button, or using the 'Options' menu). In the family tree, the vertical bars have collapsed to show a life span of less than 4 PSoup minutes. The bars will stay at this length until the bugs reach the age of 8 minutes and 0 seconds. After that, they will not grow vertically until after each four minutes is elapsed.

Now, be sure the status panel is showing, and advance the bowl to 7 PSoup minutes and 50 seconds so you can watch the next transition. Advance one second at a time up to 7 minutes and 59 seconds (7m, 59s). Watch the family tree closely and advance by one more second to 8m, 0s. Did you see the bars grow by one pixel in length? Now, switch again to the energy chart and advance by one more second to 8m, 1s. Did you see the extra bar added to the chart?

Bugs are not able to undergo fission until they are 800 PSoup seconds old. That's 13 minutes and 20 seconds (13m, 20s). Advance the bowl of PSoup to 13 minutes and 10 seconds. Don't daydream or you'll miss it.

Be sure the status panel and the family tree panel are both activated. Advance the bowl of PSoup one second at a time to 13 minutes and 19 seconds. Don't go too far or you'll miss it. Now, advance the bowl one BUG at a time, and watch the number of bugs counter in the status panel. As each bug takes a turn, the number of bugs jumps by one. After four clicks, you will now have eight bugs, and the counter will turn over to 13m, 20s. Now, advance by one bug again, eight times. Watch the names in the genetic profile, and watch the bugs in the bowl of PSoup. In the previous second, all four bugs underwent fission, and produced two daughter bugs, each pair of which were occupying the same cell. This is the only circumstance in PSoup which can cause two bugs to occupy the same cell. Each daughter is now under 'forced move' orders to move into any neighboring cell having 0 or 1 bugs until it finds itself as the lone occupant.

Note that the family tree chart has not changed. It is updated every minute, so it will not reflect the change until 14m, 0s. The energy chart, likewise, is updated every minute, so it will not reflect the change until 14m, 0s. Activate the family tree and advance to 13 m, 50s. Advance second by second to 14m, 1s.

Note the new bar in the energy chart. Activate the family tree again. Note that the four original bars have now been terminated, and eight new bars have started. Each daughter is attached to its mother's bar by a horizontal line.

Second Lap About the Bowl

This generation of bugs will be ready to reproduce when they reach an age of 13m and 20s. This should happen when the bowl of PSoup is aged to 26m, 40s.

Advance to 26m, 30s, then advance second by second to 26m, 41s, watching the bugs in the bowl and the status panel.

Advance to roughly 28 m, 0 s, and stop again. Check the family tree, and the energy chart.

You may now have 16 bugs. You may have fewer than 16 bugs. The 'Current Bug' genetic profile will be pointing at the last bug in the Bug List as displayed during the processing of the previous PSoup second. To advance it to display the first bug in the list, you must encourage that first bug to take it's turn. Advance the bowl by 1 BUG (using the appropriate evolution toolbar button, or by using the 'Evolution' menu). Note the name of the bug. Note the color of the Palmiter profile (the eight-legged picture in the clock-like picture within the genetic profile). Note the color of the first vertical bar in the family tree. They should be the same. If there are fewer than 16 bugs in your bowl of PSoup, you will notice that the family tree shows you which bugs have failed to achieve fission. Advance, bug by bug, through the entire list of bugs. Note the changes to the Palmiter profiles. Bugs appear to have randomly attained longer and shorter legs. The length of the leg is the relative probability that that gene will be expressed on any given turn. The profile is self-scaling, so the longest leg is always extended out to touch the edge of the 'clock', and the other legs are proportionately displayed. There will always be at least one longest leg visible, until we introduce the Stand Still gene in Level 3. The Stand Still gene does not appear in the Palmiter profile, in the 'clock'.

Third Lap About the Bowl

Now let's move forward more quickly. De-activate the Current Bug genetic profile. It displays every turn of every bug in every second, and slows progress. You deactivate it by pressing the 'Cur' button in the 'Options' toolbar, or by using the 'Options' menu. Activate the energy chart. Advance the bowl to roughly 42 PSoup minutes. We will now have completed three generations and should have members of the fourth generation of bugs visible in the bowl of PSoup.

Looking at the energy chart, we can see that life was not easy for the third generation. Some were born late, due to malnutrition of their mothers. Some mothers may have starved or died without issue, due to chronic malnutrition. The energy available to the bugs has sunk to a low level. If malnutrition caused the second generation of bugs to falter, it has hit the third generation even harder. If food was available aplenty, then you might expect close to 32 bugs to appear in generation 4. Typically, generation 4 has 20 bugs at start. Competition for food has now become fierce. There are many winners in this competition. But, there are also many losers. For the losers, the penalty is total extinction of their genetic line. For the winners, the prize is the chance to continue to compete for resources in future generations.

Looking at the family tree, we can see significant gaps developing. Those third generation bugs which have successfully completed fission will be proudly displaying their offspring in the family tree. Those which have failed to fission will appear as elongated bars. Those that died without issue, either due to old age or starvation, have been elided from the tree. We see four trees, in fact, one for each of the original bugs, each starting to look like a deformed antler.

Laps Four, Five and Six

Fast forward now to the end of generation 6, the beginning of generation 7. At 13m, 20 seconds per generation, that's, um, 80m, 0s, or, rather, 1 PSoup hour, 20 PSoup minutes, and 0 PSoup seconds. Let's go to roughly 1h, 22m, 0s.

Looking at the energy chart we see that life is difficult for every bug starting at generation 4. The population rises and falls slightly, but it hovers around 20 bugs. The available energy in the algae also rises and falls, but it hovers around 10,000 Nrg units. We seem to have reached some kind of steady state, some kind of equilibrium, from the point of view of the energy system.

Looking at the family tree, we should see four trees, one for each of the original bugs. You may see only three at this stage, or even just two. They are starting to look like antlers with zigzag trunks.

Laps Seven, Eight and Nine

For the last three laps, you'll be on your own. Run the scenario out to 2h, 0m. This will complete 9 generations of bugs in PSoup, as measured by the Reproduction Age Threshold (RAT), which is set at 800 PSoup seconds.

Make notes in your three-ring binder. Note the values in the Palmiter profiles for each bug.

About Bug Coloration

PSoup is designed to make use of color such that a user can better understand what is happening in a bowl of soup while the action is underway. We have therefore refrained from making color a phenotypic characteristic (i.e. one on which evolutionary forces act).

On the Technical Side

First, you need to understand a little about how a common personal computer handles colors. There are a variety of ways that a computer may use to render colors. PSoup has been designed to work with one of these modes, and colors under other modes will vary (sometimes radically) from specification. It is VERY HIGHLY recommended that you set your computer to the proper mode. Under the 'Help' menu item, in the 'About PSoup' command, the best viewing mode is noted. To set the best viewing mode, follow these steps:

- left-click the 'Start' button in the lower left corner of your screen;
- in the pop-up menu that appears, left-click on 'Settings';
- a second pop-up menu will appear; left-click on 'Control Panel';
- a large window will appear with many icons; double-click on the 'Display' icon;
- a 'Display Properties' dialog will appear; left-click on the 'Settings' tab in the top right corner;

On this tab there are two properties that need to be properly set. First, in an area labeled 'colors' there is a drop-down list with, typically, four entries. PSoup works best with the entry 'High Color (16 bit)'. Second, in an area labeled 'Screen area' there is a slider. Move the slider sideways until the label underneath the slider says '800 by 600 pixels'.

When the above two selections have been made, left-click the 'Okay' button at the bottom of the dialog. If you have made changes (and not just confirmed settings that were already correct) you will get an additional dialog with two options: restart your machine, or do not restart your machine. You do not need to restart your machine at this point, but either option is acceptable.

Rendering Colors

The computer renders colors by mixing three primary colors in varying intensity. The three primary colors for a computer are NOT related to the primary colors of nature, so do not confuse them. The three primary colors in a personal computer are Red, Green and Blue, and the connection between your computer and the monitor is therefore called an RGB connection. Each pixel on your screen has one and only one color, at any given moment in time. Each primary color can have 256 different intensities, so the total number of colors possible for a pixel is 256 x 256 x 256. If all three primary colors have an intensity of 0 for a given pixel, the resulting color in the pixel is black. If all three have an intensity of 256, the resulting color in the pixel is white.

Two Modes of Coloration in PSoup

In PSoup, the color of a bug is determined by a parameter called the 'Bug Coloration Mode'. This parameter can be accessed by the user in the 'Tinker' menu, under 'Tinker with the Environment'. The two modes are:

- By Palmiter Genotype (by gene); and
- By Root Inheritance (by root).

'By Gene' Coloration Mode

In this mode, each bug has its own distinctive color that is determined by its genes. While the color is genetically determined, it is non-adaptive in any and every PSoup scenario. In higher levels, when bugs

develop the ability to see, COLOR plays NO ROLE whatsoever in the evolution of the bugs. The bugs do evolve a type of camouflage, but the bug's color is not a factor in this.

The purpose of the coloration of the bugs, by gene, is to enable matching the bug in the bowl of PSoup to the branch of the family tree in which it is represented. In a multicultural bowl of PSoup, in which many genotypes exist at the same time, this works well. In a mono-cultural, all bugs look the same.

The details of the algorithm used to compute the color are too complex for presentation here, but in general, a bug with a strong FL gene will appear bluish, a bug with a strong FR gene will appear reddish, and a bug with a strong B gene will appear greenish.

Most colors are of a pastel shade.

Similar colors between bugs imply similar Palmiter phenotypes. Convergent evolving Palmiter phenotypes will have similar colors.

Similar colors do not imply common genetic origins.

'By Root' Coloration Mode

In this mode each primary branch of the family tree has its own color, and all descendants of a 'root' bug will inherit the same color. PSoup has a palette of 503 colors that it uses for the root colors. These colors are randomly assigned to each root of the family tree. It may be that two roots are assigned two similar colors, but no two roots get the same color.

Colors may be dark, bright, or pastel, or dull.

Similar colors between two roots do not imply similar genetic structures, but rather common genetic origin. Divergent branches of a family tree will share the same color.

In many canned scenarios the roots are colored by PSoup to have meaning. This is particularly true of the scenarios of Levels 4 and 5. Many identical root bugs will be given the same root colors in these scenarios. This enables the user to watch the behavior of different types of root bugs and their offspring. In general, if the user decides to tinker with a scenario, color by root, and not use the original bugs, then PSoup will generate random colors for each root bug, and all descendants of that root will have the same color. If the user tinkers, uses the original bugs, and colors by root, then the root colors will be preserved and used in the scenario.

About Paddies

A paddy is a feature of PSoup terrain, like the edge of the bowl, or an oasis in a Desert Oasis Ecosystem (DOE), which is used to create alternative environments, alternative ecological niches, in which evolutionary forces shape the bugs. A paddy is a rectangular subset of the bowl of PSoup. A plain bowl of PSoup, by default, is considered to have one paddy consisting of the entire bowl. A bowl of PSoup with two or more paddies has the paddies arranged side by side, and numbered, from left to right.

Terraced Paddies

If you access the 'Tinker' menu, the 'Tinker with the Environment' command, you will notice mention of paddies in several places in the 'Environment' tab. Under 'terrain type' you can choose between 'Terraced Paddies' and 'Independent Paddies'.

Imagine terraced paddies to be arranged down the side of a mountain, the left-most paddy being up on the side of the mountain, and the right-most paddy being on the plain downwind of the prevailing winds. The paddy high up on the side of the mountain will receive large amounts of rain often. The paddy down on the plain will be arid. In the same fashion, when PSoup tries to place a new colony of algae, it first tries to place the colony of algae in paddy 0, on the left. If that fails (due to the existence of a colony of algae in that specific randomly chosen position), then it attempts to place the colony in the same location in the next paddy to the right. If that fails, it attempts a placement in the next paddy. If all attempts fail, then the algae is not placed in this turn and the energy remains in the nutritive mud until the next bug has completed a turn, at which time PSoup tries again to place the algae.

In order for a colony of algae to be placed in the right-most paddy, there must be a colony of algae in the identical position in each paddy to the left. So, the left-most paddy is like a mountain rain-forest. The right-most paddy is like a desert.

When you select 'Terraced Paddies' in the 'Environment' tab, four other sections are reconfigured:
- Number of Paddies - you can choose the number of identical side-by-side paddies which will be put into the bowl of PSoup. The number of choices is restricted by the PSoup bowl size, and changes for different sizes.
- Probability of Paddy Jump (POPJ) - you can change the probability that a bug will accidentally fall from one paddy to the next right paddy, if and as it bumps into the right-hand wall of the paddy. A POPJ of 10 means that a bug has a probability of 0.010 that it will successfully move from its current paddy into the paddy on the right, if it tries to do so. Bugs cannot move from paddies on the right to paddies on the left. I.e. they can move down the mountainside, but not up the mountainside. Populations can migrate only from the left to the right.
- Bug Distribution Mode - You can choose to distribute all bugs, at the beginning of a scenario, into paddy 0, the left-most paddy, or, you can choose to spread them over all paddies. If you choose to spread them over all paddies, the number of bugs at start must be a multiple of the number of paddies. This mode of distribution is particularly useful for independent paddies.
- Algae Distribution Mode - For terraced paddies, the algae distribution mode is predetermined to be 'Cascade' which means, as described above, the algae is distributed to the left-most paddies first, and it cascades to the right-most paddies only on overflow, as defined above.

Terraced paddies provide different ecological niches in which a student can explore the possibilities of speciation within a monoculture.

Independent Paddies

Independent paddies are similar to terraced paddies, but not the same.

- Number of Paddies - as for terraced paddies, you may choose from a number of selections;

- Probability of paddy jump (POPJ) - for independent paddies is pre-set to 0.0, and cannot be changed. There is no migration of bugs between paddies under any circumstances.
- Bug Distribution Mode - is preset to ‘across all paddies’ and the number of bugs at start must be a multiple of the number of paddies.
- Algae Distribution Mode - is preset to ‘across all paddies’.

Independent paddies provide multiple side-by-side identical environments in which a student can explore convergent and divergent evolution.

Wrap and Paddies

When you have more than one paddy, the 'Wrap' feature behaves differently. The 'Probability of Paddy Jump' (POPJ) factor determines whether bugs go from paddy to paddy. Under no circumstances can a bug jump from the left-most paddy to the right-most paddy, or vice versa. When 'Wrap' is turned on, bugs can go past the upper boundary of the paddy, in which case they re-appear at the bottom of the paddy. Similarly, if a bug goes down past the bottom it re-appears at the top. When 'Wrap' is turned off, they cannot do this. Here is a summary, for each type of situation. T = Top. B = Bottom. L = Left. R = Right.

	Wrap On	**Wrap Off**
One Paddy	Wrap at T, B, L & R	No Wrap
Independent	Wrap at T & B only.	No Wrap
Cascade	Wrap at T & B.	No Wrap at T & B.
	Jump to next right paddy only. No jumping from right-most.	N/A.

Sensing and Paddies

At Level 4 and higher, bugs can develop the ability to sense algae, other bugs and edges. They also develop the ability to modify their movement behavior based on the information sensed. The effect of the paddy boundaries vary depending on the circumstances, as described above. The ability to sense these boundaries, or past these boundaries, also varies:
- In general, if bugs cannot pass a boundary, that boundary can be sensed by a bug which has developed such capability. If bugs can pass a boundary, it cannot be sensed.
- This means that, in general, if 'Wrap' in on, the top and bottom boundaries cannot be sensed no matter how many paddies, or the type. If wrap is off, the top and bottom boundaries can be sensed.
- If there is only one paddy, the left and right boundaries behave like the top and bottom boundaries, and the 'Wrap' feature controls them.
- If there are independent paddies, all vertical boundaries can be sensed.
- If there are cascading paddies (a) the left-most and right-most boundaries can be sensed; (b) all vertical boundaries which constitute the left wall of any paddy can be sensed by enabled bugs which are in the paddy to the right of that vertical boundary, but (c) all vertical boundaries (except the right-most) which form the right wall of a paddy cannot be sensed by enabled bugs which are in the paddy to the left of that vertical boundary.

With cascading paddies, all internal vertical boundaries are semi-permeable to bugs moving to the right, but impermeable to all bugs moving to the left. They cannot be sensed by any bugs moving to the right, but they can be sensed and avoided by bugs moving to the left.

This has some important implications for the bugs. If a bug can sense an edge, as per the rules above, then it cannot sense any algae or bugs beyond the boundary. However, if it cannot sense the edge, then, for that bug, the edge does not appear to exist, and it can sense (if enabled to do so) any algae or bugs which may be beyond the edge.

You will notice, in Level 4 and higher, when bugs can sense algae on the other side of the boundary of a cascading paddy, that the bugs pile up at the right-most side of the paddy trying to get to the greener grass on the other side of the fence. The number of bugs which successfully jump to a lower paddy is much

greater than chance would predict. This is an evolved behavior which may (get at greener paddy) or may not (irreversibly moved into lower paddy with harsher conditions and more intense competition in the long run) have an advantage. Watch for this phenomenon.

Simulation Versus Demonstration

PSoup demonstrates evolution in action. When you watch the PSoup screen what you see is real evolution of logical bugs as they gradually improve their resource gathering effectiveness over time. The evolution that you can see in front of your eyes is REAL evolution. It is not simulated evolution. It is not emulated evolution. It is a demonstration of real evolution in the same way that a demonstration TV in a furniture store is a real TV.

In the discussion that follows, we are going to split a few hairs, but there is an important point to understand, so please excuse the hair splitting.

In abstract terms, evolutionary processes may exist in several places. It happens that most biologists believe that evolutionary processes are the primary mechanism by which biological organisms have developed in the natural sphere here on Earth. To be sure, the evolutionary processes used in PSoup are distantly modeled on those which biologists have observed in nature. However, the bugs of PSoup and the organisms that live in the natural sphere are radically different. They play out their lives in a radically different physical environment, with radically different rules of chemistry and physics, and the very basis of the nature of existence is radically different. Organisms in nature are carbon-based physical objects that occupy space and consume energy, and over time, adapt to their physical environment. The bugs of PSoup are abstract data structures that temporarily occupy memory resources in a computer, are maintained via the steady consumption of electrical energy, and over time, adapt to their environment, which itself is a set of abstract data structures. The differences are immense.

While evolutionary processes may be based on an abstraction of processes seen to be operating in the natural sphere, they are worthy of study in their own right, and they are being applied to the solution of a very wide range of problems in today's technology. In this respect then, PSoup is a computer data base management system, of sorts, in which data structures (called bugs) are entered into a tournament, vie for the right to be copied (reproduced) by gathering data points (Nrg Units) more effectively than contemporary bugs, undergo change at the moment of reproduction (via change of some of the numeric elements of the data structures), and collectively adapt to the challenges facing them via the process of PSoup selection (similar to natural selection).

Without belaboring the point, the data structures called bugs evolve over time. This is real evolution.

However, noting that the evolutionary processes of PSoup are based on natural processes, and intentionally so, it is clearly possible to draw many analogies between what happens in PSoup and what happens in nature.

PSoup is designed (a) to demonstrate evolutionary processes when applied to data structures and (b) to be sufficiently analogous to natural evolutionary processes that a student can obtain clear insights into natural evolutionary processes via the study of PSoup.

PSoup is NOT designed to be a source of basic knowledge about biology. While it is analogous to the real world, the world of PSoup is radically more simple than the real world. For example, a student would be in error if he thought that an understanding of the crossover phenomenon in PSoup made the study of such phenomenon in real world unnecessary. While they are similar, they are not the same.

In summary, PSoup demonstrates evolution in action. The evolutionary processes in PSoup are analogous to the evolutionary processes in nature. PSoup does not prove that evolution operates in the natural sphere, but it does prove that simple processes of change and selection can lead to evolution.

The PSoup Analogy

Throughout the PSoup application the rectangular area of the screen in which the bugs reside and play out their lives is referred to as a bowl of PSoup. The analogy of course is to a bowl of mineral-rich waters in which primordial organisms spawn. This analogy is exchanged, for some scenarios, to a more appropriate analogy. For example, there are a series of scenarios called the 'Desert Oasis Ecosystem' scenarios in which an evergreen patch of algae has a much higher probability of growing algae than the rest of the bowl. A student may view such scenarios as being analogous to an oasis in the midst of a desert, or a garden in the midst of a patch of weeds, or, as an underwater hot spring in the midst of a mineral-poor deep-ocean trench. In short, the use of the PSoup analogy and terminology is not intended in any way to limit the imagination of the researcher in finding the possible applicability of the learnings from PSoup. The continued use of the PSoup pun, and the associated terminology, is intended to remind the student that PSoup is analogous (or applicable) to the real world, but is not a simulation of the real world.

Here are several variant analogies for various ecological settings used throughout PSoup:
- Primordial Soup;
- Desert/Oasis;
- Terraced Paddies on a mountainside, ranging from rain forest to desert;
- Easter Island, an isolated open energy system;
- Independent Paddies, as might be seen in a fish hatchery;
- Finite/Infinite State Machines (not an analogy, but rather, a view of PSoup);

Are PSoup's bugs truly primordial?

That is truly hard to decide. Over time, the meaning of primordial changes. We have the never-resolved problem of the missing links.

Primordial means first in order. The use of the word implies an unbroken lineage of life from some first (primordial) ancestor down to the present-day fauna. This makes sense to most paleontologists in theory. (There are a few that argue that, from the moment that sexual reproduction first appears in a genotype, the idea of a 'first ancestor' becomes meaningless.) However, the fossil record is notoriously sparse in its record of most such lineages, and the specimens of the primordial ancestors are particularly difficult to obtain. Furthermore, while it is quite possible to identify ancient fauna, classify them, estimate their age. But there is no way to determine whether they are indeed ancestral to any later organisms, or whether they are members of a temporary and soon-to-be extinct branch on the tree of life. The overwhelming majority of branches on the tree of life became extinct. So, for each fossilized creature that we look upon, we know that a successful lineage extended up from the root(s) to that point, but can also feel confident that we are looking at a design that ultimately failed. So, in terms of the natural world, we can identify ancient organisms, but it is VERY difficult to determine whether any of those we have found and identified are indeed 'primordial' in the true sense of the word.

If we substitute the word 'ancient' for 'primordial' we can feel somewhat more comfortable in saying that PSoup is analogous to ancient life. However, it is also just as analogous to modern life. In PSoup we stick with the use of the word 'primordial', in spite of its failings and misdirection, because it offers the opportunity for a good pun.

What other analogies are used in PSoup?

For some 2.5 billion years before the Cambrian explosion, blue-green algae dominated the earth. Both ancient and modern forms of blue-green algae are prokaryotic. This means they were/are simple single-celled organisms having few or no internal organelles. They consume minerals and produce energy via photosynthesis (in general). The algae in PSoup is analogous to such blue-green algae.

PSoup uses many other analogies to describe the phenotypic behavior of its bugs. Examples are parasite, prey, predator, host, mate, bush, dandelion, potato and others. Please note that these analogies are merely used as colorful ways to present what are otherwise dry and simple rules, rules which determine when a bug can detect, pursue, and feed upon or mate with another bug, when and how it reproduces, and how it manages energy. All of these phenotypic characters are strictly controlled by a bug's genetic makeup in interaction with the rules. For example, the behavior of a bug may be parasitic, predatory, or benign depending entirely on the group of contemporaries with which it is in competition.

Speed Controls in PSoup

PSoup has a multitude of features which use up CPU cycles and slow down a run. If you want to execute a run at top speed, say overnight, you might want to take some of the following actions. For each suggestion, the circumstances under which you might not want to use it is given.

1 First, and possibly, most obviously, the 'Go SloMo' button has a speed governor attached which limits the action to a maximum of 17 PSoup seconds per real second. This allows the appearance of true 'cruising' when cruisers cross the screen. If you want a run to proceed at a faster speed, use 'Go' instead of 'Go SloMo'.

2 Curiously enough, the next hint is 'move the mouse cursor off of the toolbars and menu bar, place it somewhere among the PSoup panels, then leave it alone'. When the mouse is hovering over a toolbar or menu bar, the operating system is sending out a continuous stream of queries to all toolbar buttons or menu bar items asking them if they have been clicked yet. The answer is always no. This background traffic of 'messages' can slow the speed of a PSoup run by almost 50%. The same is true, but to a lesser extent, if you simply move the mouse around on the screen. This generates messages in the background with respect to 'where is the mouse now'? Don't move the mouse around, and things will speed up a lot.

3 Display only those panels you need to have displayed to track progress. There are two actions happening in the background. First, the data object that is there to contain the data needed to update the panel is filled with data. This takes some CPU power, and is unavoidable. Second, PSoup checks whether the user has requested each panel to be presented, and, if yes, that panel is drawn. All panels are thus checked. Only those that are turned on are drawn. Drawing a panel requires a lot of CPU power, especially if there are graphics. In order of the size of the drain on CPU resources, the following panels drastically slow a run:
(a) Some panels are updated for every function of every turn of every bug. Specifically, the 'Life Functions Panels' are updated five times per turn of a bug.
(b) Some panels are updated for every turn of every bug. The Current Bug Genetic Profiles, are updated for every bug.
(c) Some panels are updated every PSoup second. The 'Average Bug Genetic Profiles' are updated every second. And the 'Family Tree' and 'Energy Chart' are updated every second, but only during the first two PSoup minutes. The 'Status', 'Legend' and 'Counts' panels are all updated every second, but these non-graphic panels use very little of the CPU resources.
(d) Some panels are updated every minute, or every couple of minutes. All other panels fall into this category.

4 Historic data collection happens once every generation, which, typically, is every 4-15 PSoup minutes. This uses very little CPU time. Data collection for allele counts and allele values is on by default. Data collection for pairing squares is off by default.

For long runs which will be monitored, it is suggested that graphic panels which are updated once per second or more often be turned off.

For long runs which are not monitored (e.g. overnight runs) all panels be turned off.

Be careful about running PSoup minimized, as it may become confused about panel locations and bomb. Nevertheless, this is a possibility. In minimized mode, PSoup runs on the lowest priority. But if the computer is working on a high-priority but low CPU task, such as word processing, then PSoup will run very quickly in minimized mode as all graphic updates are avoided by the operating system.

Background To Scenarios

About Genotypes and Phenotypes (PSoup Theory and Practice)

This discussion focuses on the Palmiter genes, the nine genes found in the type 1 chromosomes, eight of which are visible in a Level 1 bowl of PSoup. However, the concepts are equally applicable to all genes used in PSoup. This article should be read in conjunction with review of Scenario 1B - A Gallery of Bugs.

Each bug in Scenario 1B has nine Palmiter genes (also called transport genes) located together in a type one chromosome. Eight of these genes (named gene0 through gene7) rotate the bug in preparation for a step forward. The ninth, called gene8, or the 'stand still' gene, causes the bug not to turn and step, but rather to simply stand still. This ninth gene has been deactivated in Scenario 1B and can be ignored.

Simply put, a genotype is a unique set of gene strengths. Two bugs are said to have the same genotype if and only if all corresponding genes have equivalent gene strengths. I.e. gene0(Bug1) has a strength equal to gene0(Bug2), etc., up to gene8(Bug1) = gene8(Bug2). If one of the set of corresponding genes has a difference of strengths, then the bugs are not of the same genotype.

Each of the Palmiter genes are simple genes, and, in Scenario 1B, have a finite number of possible values. The base is fixed at 2.0. The exponent can range in value from -150 to +150, but must be integral. This means that the exponent can have any of 301 different values, and the gene can have any of 301 different strengths.

How Many Genotypes?

A Level 1 bug has eight active Palmiter genes and one inert Palmiter gene (the Stand Still gene, which does not change values nor play any role in movement decisions). No other genes are active. There are 301^8 (that's 301 raised to the power 8) possible combinations of values. That works out to approximately 6.73801 x 10^19 different possible genotypes, or 10 million trillion genotypes.

A unique genotype for a Level 1 bug can then be represented as an ordered 8-tuple, where all coordinates are elements of the set of 301 allowed strengths.

What is a Phenotype?

A phenotype is the outward expression of a genotype. Genes control behavior. The exhibited behavior that derives from a genotype is the phenotype.

Let's take a concrete example. Each gene has a probability of expression, during any given turn of the bug, that is proportionate to its strength when compared to the sum of all strengths. This can be written as Pn = Sn / (S0 + S1 + S2 + S3 + S4 + S5 + S6 + S7). Pn is the probability that gene number n will be expressed on this turn. Sn is the strength of gene number n. S0, etc. are the strengths of each of the eight numbered genes respectively. The following table presents the base, exponent, strength and probability of expression for each of the eight genes in a typical genotype.

Gene	Base	Exponent	Strength	Prob Of Exp
gene0	2	3	8	.381
gene1	2	0	1	.048
gene2	2	1	2	.095
gene3	2	0	1	.048
gene4	2	0	1	.048
gene5	2	1	2	.095
gene6	2	1	2	.095
gene7	2	2	4	.190

This means that this genotype will cause the bug to step forward without turning 38% of the time. 19% of the time the bug will make seven turns to the right, then step forward. And so on. When a bug undergoes mutation, one of the exponents goes up or down in value by 1. All of the percentages shift accordingly. So it is clear that the behavior changes somewhat as the genotype changes. But what happens if all of the exponents are increased in value by one. Here's a table showing the result.

Gene	Base	Exponent	Strength	Prob Of Exp
gene0	2	4	16	.381
gene1	2	1	2	.048
gene2	2	2	4	.095
gene3	2	1	2	.048
gene4	2	1	2	.048
gene5	2	2	4	.095
gene6	2	2	4	.095
gene7	2	3	8	.190

In this case, the probability of expression of each gene is the same as for the genotype in which all corresponding genes had an exponent with a value of exactly one less. If we shift all of the exponents, in tandem, upwards or downwards by exactly the same amount, the distribution of probabilities does not change.

The table of probabilities of expression describe the probabilistic search pattern that is determined by the genotype. This table of probabilities is (a description of) the phenotype. We can write a phenotype as an ordered 8-tuple of probabilities, in which each coordinate is a real number between 0 and 1, all coordinates add up to 1, and each coordinate represents the probability of expression of the corresponding gene.

The C1 genetic profile of each bug displays both the genotype (strengths of each gene) and the phenotype (% probability of expression).

How many phenotypes are there?

An exact answer to this question would take a great deal of work. We can make some estimates, however. For each genotype, we can determine a set of genotypes that would produce precisely the same phenotype by the simple expedient of translating all exponents in step through the range of possible values.

The pristine genotype has all exponents equal to 0. It has a characteristic search pattern in which on any turn a step in any direction is equally likely. A bug with all exponents equal to 1 would have the same phenotype, and the same characteristic search pattern. A bug with all exponents equal to 2 has the same phenotype. Etc. There are 301 possible values of the exponents, and 301 genotypes that exhibit this phenotypic search pattern.

Let's look at another set of genotypes. Let's look at the daughter of a pristine bug which has one exponent equal to 1, all others equal to zero. There are only 300 such genotypes that give rise to the same phenotypic behavior. Because the exponents in the genotype have two values, we have one less optional genotype produced by translation of the original genotype. The set of all genotypes which have exponents valued at either 0 or 1 is 256 (2^8). Two of these are pristine (all 0's, or all 1's). So we have 254 'almost pristine' genotypes with exponents limited to values of 0 or 1. Each of these genotypes determines a unique phenotype. Each of these phenotypes has 300 possible genotypes that produce it.

In summary, it appears that:
- there is one (1^8) phenotype in which all exponents are the same.
- there are 256 phenotypes in which all exponents differ from each other by a maximum of 1.
- there are (3^8) phenotypes in which all exponents differ from each other by a maximum of 2.

We postulate that there are (300^8) phenotypes, or 6.561 x 10^19 phenotypes, in total.

That's a lot of phenotypes.

Why Identify and Study Phenotypes

Evolutionary forces cannot see or sense genotypes, and act on genotypes indirectly. They act on phenotypes directly. If a gene is suppressed (as gene8 is in Level 1) then it plays no role whatever in determining a bug's behavior, has no effect on search patterns or success/failure rates, and is hidden from the forces of evolution. There is no selection for or against this gene.

In a gross manner, we can consider phenotypes to be the visible manifestations of the invisible genes. To take an example from the natural world, the color of a cat is phenotypic. The coloration of the cat will have a direct bearing on its success as a hunter. The color is determined by genes. Many genotypes may result in the same effective phenotype. Evolution selects those cats with an effective phenotype. If there are other color genes in one of the genotypes, color genes which are suppressed and not expressed in the phenotype, they will be accidentally selected along with the expressed genes.

What are the Practical Implications for PSoup?

In PSoup a user has the ability to use the TinkerBug Wizard at any time and tinker with the genes of any bug. As you tinker with the genes, you are modifying the genotype. The TinkerBug Wizard also displays some (not all) of the phenotypic characters that are determined by the genotype. Each tab in the TinkerBug Wizard clearly distinguishes between genotypic and phenotypic items. You will note that you cannot modify the phenotype directly, but only via the modification of the genotype.

NOTE: The TinkerBug Wizard is available through the intermediate or advanced user interfaces, which can be activated via the 'Control Panel' under the 'Options' menu.

You will also note that some phenotypic characters (e.g. C1, C2, C3/4 Penalties) are not always visible in the TinkerBug Wizard tabs, or on the genetic profiles. The decision to use or suppress the use of these penalties is a system-wide decision, not a bug-by-bug genetic decision. The toggles that turn them on or off are, therefore, considered environmental. Their appearance is controlled by the 'Tinker with the Environment' Command. Once it is determined that they will appear and be used, the determination of their values is a bug-by-bug determination, under the control of the genotype.

PSoup and Edge Effects

Scenario 1C provides us with a simple example of something called edge effects.

The Edges are a Factor in Evolution in PSoup

The world of PSoup is analogous to the real world, but it is extremely simple in comparison. In a real world environment there are innumerable ways for an organism to interact with the environment. In PSoup, there are only four, as follows:
1 - Nutritive mud - a bug interacts with the nutritive mud indirectly through the algae. Algae may grow only where nutritive mud is located. In all scenarios the nutritive mud grows everywhere, but in some scenarios there is an 'oasis' with super-nutritive mud.
2 - Algae - a bug eats algae, if it happens to stumble across some.
3 - Other bugs - a bug may not enter a cell currently occupied by another bug. If it attempts to do so, it will be gently bumped back into its original cell. It will nevertheless have expended energy equivalent to the strength of its EPM (Energy Per Move) gene.
4 - The edge of the bowl of PSoup. If the wrap feature is turned off, a bug bumps against the edge of the bowl, expends energy, and has its search pattern disrupted by influences for which it may not be well prepared.

So, the edge plays a role that is worth examining.

Let's think through an example that will help us to understand what kind of role the edge may play. Let's presume that we have a bowl of PSoup with 20 strongly typed Arrows. By this we mean the F gene (gene0) has a strength of 8 or more, and the other genes have a strength of less than 0.125. Such a bug will tend to go in a straight path. About once every eight steps it will break its pattern, take a new direction, then carry on. This bug will be able to travel far and fast. However, eventually it is going to reach the edge of the bowl of PSoup. What happens to it then?

The answer is, well, that depends on what special rules are invoked when it reaches the edge. In scenario 1A we have one set of rules. In scenario 1C we have a different set of rules.

No Wrap

In Scenario 1A we have implemented the 'No Wrap' option. If you go to the 'Tinker with the Environment' tab on the Tinker Command, you will see that Scenario 1A has the 'Do not Wrap at Edge' option selected. However, if you load Scenario 1C and check the tinker tab, you will see that it has the 'Wrap at Edge' option selected.

What rules are used in the No Wrap option? In Scenario 1A, if a bug attempts to step over the edge of the bowl of PSoup, it is (gently) bumped back into the cell from whence it came. In detail, when the bug takes its turn moving, it consults with the pseudo-random number generator and selects a Palmiter gene to be expressed, and turns the appropriate amount. Let's say gene1 is expressed so it turns 1 turn to the right, then steps forward. However, it bumps its little head on the bowl at the edge of the PSoup, and is gently pushed back into the cell that it occupied just a moment ago. It retains the new orientation, but does not move.

What are the implications for the bugs? Well, for Twirlies, or even for Cruisers, this is not a great problem. Every few turns they have changed their direction by 180 degrees, edge or no edge, so this is a momentary disruption to their built-in genetic search path, as determined by their Palmiter genes. Their birthright (their search path) is tainted, but not disrupted. The story is different, however, for Arrows. Arrows charge straight ahead, and occasionally turn slightly left or right, but rarely turn about a full 180 degrees. When an Arrow reaches the edge, it continues to adjust its direction slightly, from time to time, but it keeps banging its little head against the edge of the bowl and going nowhere. All the while, it is expending energy and

finding very little in the way of algae. Most Arrows simply head crash into an edge, follow it to a corner where they become hopelessly trapped, and expire of exhaustion. Scenario 1A is hostile to Arrows.

Wrap

Scenario 1C has a different set of rules. In scenario 1C the top edge of the bowl is considered to be connected to the bottom edge of the bowl. If a bug is in column 3 of the bowl of PSoup and attempts to step directly upwards, it will find itself magically transported to the bottom cell of the same column at the bottom, and all algae found there is available for feeding.

The effect is that the genetically determined search pattern of any bug, whether they are Twirlies, Cruisers, or Arrows, is able to be executed without interference of any kind by the edge.

Other Solutions

We could have implemented any number of other options as edge effects. The bugs could jump out of the bowl and expire. They could suffer transposition of all of their Palmiter genes, flipping them left to right. They could gain or lose energy. We chose to offer only two options for handling the edge of the bowl.

Are there Other Edges to Worry About?

Edge effects look like a small thing. They are a ***** BIG THING *****.

A simple bowl of PSoup appears to exist in only two dimensions, and have two sets of edges: (a) top and bottom; and (b) left and right. We have two sets of rules, one for each dimension: vertical and horizontal. We have implemented homologous sets of rules in Scenarios 1A and 1C, but there are other terrain types in PSoup that have different wrap rules for top/bottom and left/right edges. In particular, when you investigate 'Cascading Paddies' you will see that the 'Wrap At Edge' option has a different meaning.

We say it appears to exist in only two dimensions. This appearance is deceptive. Scenarios 1A and 1C exist, logically, in roughly 660 dimensions. Each dimension has an extent (or width), and each extent has two edges (a top and a bottom of the extent, if you like). PSoup implements a set of rules to handle each and every one of these edges,

Where does this number 660 come from? For a detailed discussion of these concepts see the articles on 'PSoup as a Finite State Machine'. Here is the short version.

When you analyze a bowl of PSoup, it is useful to identify all of the data elements that can change from second to second. Call these data elements the parameters which define the current state of the bowl of PSoup.

The location of a bug is one of these parameters.

Or, is it two? Or five hundred!?! As far as the bug is concerned, it is two: row and column. If it gets into a corner, and bangs its head, the corner is two-dimensional. However, from the computer's point of view, it is one dimension, as each cell is a member of a one-dimensional array, and the rules of movement simply give it the appearance of two dimensions. The point of this little digression is, the identification of dimensions is a bit of a tricky business and subject to disagreement. The following argument splits hairs and dimensions finely.

The state of a cell with respect to containment of algae is a dimension. The cell at (0,0) either contains algae, or it does not contain algae. This is a dimension of PSoup with an extent of 2 and a value equal to the number of colonies contained. We have decided that cell (0,0) will not be able to hold fewer than 0 or more than one colony. We could have allowed 300 colonies in that cell. We could have allowed negative numbers (Venus fly traps that suck energy out of bugs). We SET THE RULES for this cell, determined the extent of the dimension, and determined edge effects. So, what are the edge effect rules. If PSoup tries to

put a second colony of algae into this cell, it fails, and the energy stays in the nutritive mud. Simple! If a bug enters cell (0,0) and there is no algae there, the bug neither loses nor gains energy. Simple! A rule for the top of the extent, and a rule for the bottom of the extent.

Cell (0,1) is a different dimension, with a potential to have a different extent and different edge effects. We could allow more colonies of algae in this cell, and have fractional access (a bug eats half the colony). We have decided to implement the same extent for this second cell, and for all 250 cells, but each cell represents a dimension of the PSoup system, each dimension has a pre-defined extent, and pre-defined rules for handling the edges.

So far, our bowl of PSoup has, not one, not two, but 250 dimensions.

Each cell can also have 0, 1 or 2 bugs in it. This is another dimension for each cell. We now have 500 dimensions.

It happens that we have implemented the same edge effect rules for many of these blocks of dimensions, making it appear simple, but the dimensions exist.

Each bug has eight active Palmiter genes, eight dimensions per bug, twenty bugs per bowl, that's 160 more dimensions. A Palmiter gene can have 301 different values (with an exponent ranging from -150 to +150). There is a rule for handling exponents that go out of range. They are the same for all simple genes, bug they don't have to be.

There are other minor sources of dimensionality in PSoup, but they are of minor import. The point has been made. 250 + 250 + 160 = 660 dimensions, each with extents and edges defined, and edge effects defined.

Some other Examples of Edge Effects

Here are some edge effects that are interesting to study:
- Super-charged bowls of PSoup (more than 100% loading factor) in which overcrowding occurs.
- Cascading paddies with wrap on/off.
- Low energy bowls of PSoup; how low can you go and still maintain a stable population?

Here are a few to avoid:
- Maximum of 500 bugs in the PSoup data base.
- Maximum of 1500 nodes in the family tree.
- ??

How does PSoup Handle Edge Effects

Notification:
Edges can be a source of bias in the evolutionary processes. The interference of edges must be made visible, if it occurs. The Event Counts panel displays the number of counts of breaches of key edges. This panel is active via the Advanced User Interface only. For more information see the help text on the Event Counts panel.

Consistency:
Many edges are a fundamental part of the design of a system and play an immense role. In PSoup the effect of these edges is greatly simplified through the use of consistency:
- All dimensions respecting cells and algae are handled in an identical fashion.
- All dimensions respecting cells and bugs are handled in an identical fashion.
- All dimensions involving gene strengths are handled identically.

Optional rules:

The edges of the bowl of PSoup offer a simple means to experiment with edge effects and identify the impact on the evolutionary processes that are the subject of PSoup. The selection of optional rules is under the control of the user.
- Wrap or No Wrap at the edge of a bowl of PSoup.
- Probability of Paddy Jump.

Avoidance of edges:
- Restricted transitions;
- Hidden edges.

This last pair of bullets requires explanation. You can view PSoup as a multidimensional cube. Let's focus on a single bug. Each dimension (gene) has an extent (301 possible values). A bug is a cube of 8 dimensions, and it defines part of the overall dimensionality of PSoup. How do we want to handle the edges. We could wrap the top to the bottom (not a good idea) or we could bump the value back into bounds when it goes out of bounds (good idea). We could also set up processes that bias the state of the bug such that a step out of bounds is very unlikely (best idea). This approach is called restricted transitions in PSoup, and is used to control edge effects on genetic mutations. These are implemented as optional rule sets and are called the C1 penalty, the C2 penalty, and the C3/4 penalty. The C1 penalty prevents genes in type 1 chromosomes from taking on extreme values. The C2 penalty does likewise for genes in type 2 chromosomes, and the C3/4 penalty does the same for genes in type 3 and type 4 chromosomes.

Hidden edges are those edges which have been handled in such a way that key characteristics of the system being studied are not affected by transition across the edge. Wrap of top/bottom and left/right are examples of hidden edges.

Summary

Edge effects are ubiquitous and important. PSoup has been designed to simplify all edge effects, apply common rules for handling edge effects across many homologous dimensions, and make them invisible. Measures such as wrap and chromosome-related penalties are used to control edge effects under the direction of the user.

PSoup and Terrain Effects

In this article we look very briefly at the implementation of terrain effects in PSoup. It is intended that this article be read in association with Scenario 1D.

In the world of PSoup things are very simple, intentionally so. The forces of evolution are driven by the interaction of the bugs with their environment. There are only four ways a bug can interact with its environment:
- A bug can interact with the nutritive mud;
- A bug can interact with the edges of the bowl;
- A bug can interact with algae;
- A bug can interact with other bugs.

Two of these, mud and edges, are terrain effects.

The edge of the bowl causes a type of terrain effect which is also an edge effect. This is discussed in great detail under 'PSoup and Edge Effects'.

In a Level 1 bowl of PSoup (e.g. in Scenario 1D) bugs cannot sense anything - not mud; not algae; not other bugs. Furthermore, the mud is ubiquitous, so how can it affect the evolution of the bugs, you may ask.

The bugs interact with the nutritive mud indirectly, through the algae. In Scenario 1D we have an 'oasis', a special patch of mud which has first call on placement of any algae colony. This means that the computer makes one attempt to randomly place a new colony of algae into the oasis before making a placement attempt in the wider extent of the bowl. If the computer randomly chooses a cell in the oasis already containing a colony of algae, then and only then is the colony placed outside of the oasis.

This means that, in the early stages of a run, the oasis will be greener than the surrounding terrain, call it the dessert. Bugs which do not travel far (e.g. Pristine bugs, or Twirlies, or Jitterbugs) will benefit from a constantly renewed source of algae. However, over time, most renewed colonies of algae will eventually, in all probability, be bounced out of the oasis and will land out in the dessert. When this happens, the oasis will eventually become barren and the local denizens will starve. However, with the rise of wayfaring bugs such as cruisers and arrows, the far-flung colonies of algae will be found and eaten and renewed back in the oasis.

One would expect, then, that there would be a long-term balance of power between den-sticking bugs and wanderers.

So, even though the bugs are totally senseless to the existence of this patch of super-nutritive mud, it drives the evolutionary development of two codependent species of bugs. This is the power of a simple terrain effect.

PSoup will offer the user the opportunity to experiment with two other types of terrain effects. The user will be able to establish side-by-side paddies which offer identical environments in which bugs can flourish, and experiments in convergent and divergent evolution can be carried out. The user will also be able to establish cascading side-by-side paddies in which bug in the left-most paddies are protected from parasitism or predation by bugs in those paddies further to the right, but also benefit from a first call on all new colonies of algae. In these scenarios, the forces of evolution cause the bugs in some paddies to become more herbivorous, while the bugs in other paddies must become parasitic or carnivorous.

In the real world, geography and terrain play a huge role in the process of speciation. In PSoup, the user has the opportunity to use simple terrain effects to come to understand the evolutionary forces that cause this to happen.

Survival of the Fittest

In this article, we look closely at the processes which drive evolution in PSoup, and the meaning of well-known catch phrases such as 'natural selection' and 'survival of the fittest'. This article should be read in conjunction with viewing of '**Scenario1E - Survival of the Fittest**'.

"I have called this principle, by which each slight variation, if useful, is preserved, by the term Natural Selection."

Sir Charles Robert Darwin (1809-1882), The Origin of Species, 1859, Chapter 3

First we shall think our way through the stages that are exhibited by a typical bowl of PSoup when it is first generated.

Let's Take an Example

Suppose we have a large bowl with 40% energy loading factor and 30 bugs at start. There will be 24,000 Nrg units in the bugs (800 each) and 324,000 Nrg units in the nutritive mud available for eventual distribution in the bowl of PSoup. Suppose we have a DOE (Desert Oasis Ecosystem) scenario, wrap off. The laws of probability say that there will be some collisions as the colonies of algae are initially distributed, and each such collision will defer the distribution of that colony to a later turn, so there is a high probability that most of the energy will remain in the nutritive mud, and a lesser amount will be distributed as algae.

So, in the starting state, the bowl of PSoup has a small number of bugs, each partially energized, a large store of energy in the nutritive mud which has not yet been made available to the bugs, and a smaller store of energy in the algae which is available to any bug that chances upon it.

Before we start the PSoup clock ticking, let's do a little arithmetic to size up this bowl of PSoup. These bugs have their RET (Reproduction Energy Threshold) set at 1000 Nrg units. If we assume that in a mature bowl of PSoup all bugs have on average 1000 Nrg units, then we can predict how many bugs a mature bowl of this PSoup might contain. Why do we pick the RET? The logic is, in a mature bowl of PSoup the bugs have not reproduced either because they are underage or under-energized or both. Without putting too fine a point on it, if those that are over-aged are under-energized, and those that are under-aged are over-energized, on average they would have roughly RET. Using this rule of thumb, we expect we might be able to support roughly 348 bugs in this bowl of PSoup. In practice, the number will settle at about 300, so our estimate is slightly high, but this approach to estimating the number of bugs that can be supported tends to work.

If we can support 300 bugs, and we are starting with 30, there is more than enough energy for everybody to survive. But they must each manage to stumble across enough food to survive.

Let's start the clock ticking.

The RAT (Reproductive Age Threshold) of our bugs is set at 800 PSoup seconds, or 13 minutes and 20 seconds. Let's call this length of time a generation. During the first generation the bugs stumble around and gather algae. At the same time, the rest of the energy stored in the nutritive mud is distributed as algae and the energy in the mud drops close to zero (some integer less than 40). One or two of the thirty bugs may die, but the probability is high that they will all have more than sufficient energy to fission when they come of age. At 13 minutes and 20 seconds, almost all bugs will fission, jumping the number of bugs to 60. A few which happen to be under-energized may have a slightly delayed fission.

We are now going through the second generation. There is enough algae for everybody, but the bugs are all close to the DOE, and there are bare patches appearing. Events are as they were for generation 1. A

few bugs die; and a few suffer a delayed fission experience, but at 26 minutes and 40 seconds most undergo fission to produce generation 3, something less than 120 of them. Typically, you might get 90.

For generation 3 there is still plenty of algae to go around, but the bugs are tightly packed into a corner of the bowl. There will be significant starvation. At 40 minutes, generation 3 undergoes fission and produces generation 4 bugs. If there were not starvation, we would now have 240 bugs. Instead we might have 95-100.

For generation 4 it would be an overstatement to say there is plenty of algae for everybody, as there are bare patches, and patches of plenty. Since the bugs cannot see or sense the algae, they have no means of knowing that they are in a bare patch. Unless their genes have endowed them with a pattern of movement that takes them out of the bare patch, they will have a shortage of food. Sure, on average, there is enough food for all, but most is in the DOE, and the rest seems to be stacking up at the far end of the bowl, away from most of the bugs. At 53 minutes and 20 seconds there will be a burst of fission events, followed over the next several seconds by a minor number of delayed fission events. Something like 120 generation 5 bugs will be born. We should have 480. Our population of bugs is roughly 1/3 of what it should be.

Before we continue with the story of generation 5, let's take stock. The bowl of PSoup now has some bare areas that have a high density of bugs, and some green areas with a low density of bugs. Of course, the oasis is evergreen and it has both lots of bugs and lots of algae. The bugs themselves have undergone four stages of mutation. Each bug has up to four Palmiter genes that are distinctly different from their parents, which cause them to follow a differently shaped search path in their search for algae. At first, the shape of the search path did not matter much, because there was lots of algae for everybody. But over time, this has been changing, and the shortage of algae in some areas is acute. It has already caused the death without issue of several bugs, either by starvation (DET) or old age (DAT).

What has been happening with the energy. All bugs started in the oasis. All bugs have an EPM (Energy Per Move) set at 3. For generation 1, with every tick of the clock, 90 units of Nrg were expended by the bugs. Two colonies of algae were distributed, and 10 units of Nrg went into the nutritive mud for later distribution. When the algae is distributed, the computer first attempts to place it in the oasis. If that fails, due to collision with an existing colony of algae, then it attempts to place it in the desert region in a random spot. If it happens to land somewhere near the oasis, then a bug wandering out of the oasis will soon gobble it up. If it happens to land at the other end of the desert, far from the oasis, it will be a long time before it is found by a bug. During these first five generations there has been a trend of consumption of algae near the oasis and deposition of algae away from the oasis. This is not an error in the random distribution of algae. It is an artifact of the behavior of the bugs. Generation 2 consumed 180 units of Nrg per turn, much of which was exported to the back country. As the number of bugs increases, the rate at which algae is exported from the oasis region to the nether corners of the bowl increases.

So at the same time that we have seen a build up of bugs in the neighborhood of the oasis we have seen a shift of resources away from the oasis. Those bugs unlucky enough to have wandered outside of the oasis (remember they cannot see or sense algae until they eat if) have a higher likelihood of starving, unless they have wandered very far away, where the density of bugs is less. Who is likely to wander? Pristines and Jitterbugs are likely to stay more or less in one place. The oasis is a safe haven for them, as the algae is dense there. Dodgers, Arrows and Cruisers are likely to wander. The oasis is a potent source of food for them, but given their tendency to wander out of it and not find their way back, it can be a death trap.

With generation 5 something remarkable starts to happen. Up until now, every generation has shrunk in size between its inception and its denouement. Generation 5 stays flat. The delayed fission events from generation 4 roughly equal the deaths by starvation, and the number of bugs stays flat. There is still a jump in numbers at the demarcation second between 5 and 6, but generation 7 grows in size, and the demarcation between generations 6 and 7 is not there. By generation 8, there are roughly 300 bugs, and the numbers hover there ever after.

Something else happens with generation 5. We start to see a clear distinction between the genotypes of the bugs that dominate the oasis, and those that dominate the desert. Twirlies and Jitterbugs clearly are

predominant in the oasis. If any wander outside of the oasis they soon die. Cruisers, Arrows or Dodgers dominate the desert. If any wander into the oasis they benefit briefly, maybe enough to fission, but soon depart on their wandering ways.

After generation 8 a curious trade-off develops between these two species. For more on this, read the article on 'The Gaeia Effect'

Survival of the Fittest

In a mature bowl of PSoup, consider the bugs currently alive at a given PSoup second to be a generation. In this generation of bugs it is highly unlikely that any two bugs will have exactly the same Palmiter genotype. The genotype determines a characteristic algae search pattern. All bugs are expending energy at the same rate. All bugs are searching to find sufficient energy to reproduce themselves. Suppose two bugs are very similar, differing only in a single gene. Suppose that this gives one of the bugs a 1% improved chance of finding algae. They both have 800 seconds. In that time, the bug with the slight advantage will gather 8 more algae than the other bug, or 320 units. If there is a shortage of energy, the advantaged bug has a better chance of reproducing. If a bowl can support only 300 bugs, and there are already 300 bugs in the bowl, 50% of all offspring will either not be born, or die of starvation or old age after birth. Those that have a slightly higher probability of gathering algae, have a higher probability of surviving.

Is it always the 'most fit' bug that survives. Probably not. But if a less fit bug happens to reproduce two less fit daughters, will they also be so lucky, buck the trend, and survive to reproduce? Probably not. The odds are that, eventually, the fit genetic lines will supplant the unfit genetic lines.

Survival strategies

This raises mind pictures of survivalists in khaki clothes carrying weapons and planning their survival.

It is sometimes useful to personify things and say that a bug has a superior survival strategy. The bug with the superior strategy succeeds.

This wording can also be misleading. Note, these bugs are mindless and cannot strategize. Bugs are driven by the instinct that is coded into their genes. A non-deterministic (or probabilistic) search pattern is coded into their genes, and this search pattern (as interpreted by the pseudo-random number generator) is the survival strategy. It was fixed immutably when the bug was born. The interaction of this search pattern with the bug's circumstances (other bugs, algae, edges) determines its success.

The luckiest bug survives, and a huge dollop of luck springs from its genes.

The most fit genotype, whether or not it is currently present in the population, will emerge and dominate, until it is surpassed.

Nevertheless, it is often useful to say things like, 'Jitterbugs have a superior survival strategy for life in the oasis'.

The Environment Favors Survival Strategy X

This mode of talk personifies both the environment and the bug. Bugs are mindless and have no strategy, as stated above. Environments are heartless, and favor nobody. Circumstances either increase the probable success of certain search patterns, or they decrease the probable success. If enough bugs with a given search pattern are exposed to precisely the same environment (will never happen) then the probabilities will be turned into hard numbers of successes and failures.

However, the environment, which consists of algae, bugs, edges, and algae distribution patterns, changes from second to second. It is never the same for any two bugs. It is like a river, always flowing, always in flux.

Nevertheless, it is often useful to say things like, 'The oasis favors Jitterbugs and Twirlies, while the desert favors Cruisers, Dodgers and Arrows.'

Natural Selection

This conjures up pictures of mother nature lovingly tending her garden and choosing to hybridize this rose over that rose.

Nature is not loving. It is heartless, mindless, and non-personal.

Natural selection is not something that someone or something does. It is a phenomenon that happens. Or rather, it is a phrase used to identify a complex of outcomes.

Nature exhibits local characteristics that implicitly define the circumstances for survival of a population of organisms. The characteristics include geographic, chemical, and biotic features, and more.

PSoup is similar. The local characteristics consist of the relative location, density and behavior of other bugs, the location and density of colonies of algae, the algae distribution patterns (e.g. Oasis first), edge effects, and stores of unused energy in the nutritive mud. Those bugs which possess a probabilistic search pattern (Palmiter genes) which would be the most effective, which is the best fit to the circumstances presented, have the best probability of survival. When one such bug actually survives and reproduces, it has been 'selected'. When a bug fails to reproduce itself, but rather dies without issue, it has not been selected. This is PSoup selection, the analogy to 'natural selection'.

Bugs select themselves for reproduction by winning in the competition for resources. Evolution is most rapid when resources are reasonably plentiful, and the competition is fierce.

Returning to Charles Darwin

In Charles Darwin's own words:

"Each (species of bug) is striving to increase in a geometrical ratio… each has to struggle for life and to suffer great destruction… The vigorous, the healthy, and the happy survive and multiply… From the war of nature, from famine and death, the most exalted object which we are capable of conceiving, namely, the production of the higher animals, directly follows. There is grandeur in this view of life, with its several powers, having been originally breathed by the Creator into a few forms or into one, and that… from so simple a beginning endless forms most beautiful and most wonderful have been and are being evolved. (The Origin of Species, 1859, Ch 3).

PSoup and Control of Life Functions

This section should be read in conjunction with the PSoup primer. The material here must be understood by a Level 1 student before (s)he moves to level 2, but the presentation addresses Levels 1 through 3.

Life in a bowl of PSoup is controlled by a number of parameters. Some of these parameters play a role only when the bowl of PSoup is being initially set up, prior to the start of a run of a scenario, and play no further role in the evolution of life in the bowl. Such parameters are said to play a role in determining the initial conditions, or starting state. Other parameters are used to control processes that are used by PSoup on each turn, but are not specific to any bug. These parameters are called operational parameters. Finally, some parameters are used by every bug on every turn for each and every tick of the PSoup clock. These parameters are said to control life functions.

Some parameters which control initial conditions are:
- the size of the bowl of PSoup (i.e. how many cells wide, by how many cells tall);
- the terrain type (i.e. plain bowl, desert/oasis, cascading paddies);
- number of bugs at start, and bug distribution mode (after start, bugs appear in the cell occupied by the mother);
- wrap mode;
- initial energy of bugs at start of a scenario (by default, always 800 Nrg units);
- amount of energy in the nutritive mud;
- and others.

Some operational parameters are:
- algae distribution mode (normal, Oasis first, steady, or variable);
- probability of paddy jump (for independent paddies);
- rules of PSoup chemistry.

The distinction between initial condition parameters and operational parameters is fuzzy. The distinction between these and the Life Function parameters is NOT fuzzy.

There are six life function parameters, and each has a name and an acronym. A student must memorize these if (s)he is to gain any depth of understanding from PSoup.

In Level 1 of PSoup these parameters are constant (never change) and are identical for all bugs. Therefore, in Level 1, they are akin to system-wide operational parameters. However, in Level 2 of PSoup and higher levels we encode these parameters as genes. As such, they are subject to the same chemical rules for mutations as the Palmiter genes, and subject to evolutionary pressures. At startup for a scenario all bugs may have the same strengths for these six genes, but as mutations accumulate and time progresses, there is a buildup of changes, and PSoup selects for those bugs with the best combination of strengths for these genes. In other words, these are no longer system-wide operational parameters but become personalized genetic strengths on which evolutionary forces exert pressure.

These six parameters/genes vary greatly in strength at startup , as follows:

Gene Name	**Normal Starting Strength**
Death age threshold (DAT)	1600 PSoup seconds
Death energy threshold (DET)	30 Nrg units
Repro age threshold (RAT)	800 PSoup seconds
Repro energy threshold (RET)	1000 Nrg units
Energy per move (EPM)	3 energy units
Maximum energy per bug (EPB)	1500 Nrg units

These default strengths are based on the original program, Simulated Evolution, written by Dr. Michael Palmiter. The name given to each of these parameters is new to PSoup, but each represents a value taken from the original program. These strengths were, by some method, determined by Dr. Palmiter to be acceptable to provide a good demonstration of evolutionary forces. If we subject these control parameters themselves to evolutionary forces, they will drift in strength.

DAT - on the turn in which the bug reaches this age, if it is not able to reproduce, it dies of old age. Bugs which have a longer maximum life span (or DAT) may reasonably be expected to have a higher chance of reproducing, all other genes being of equal strength.

DET - on the turn in which the bug's energy sinks below this level, if it is not able to replenish its store of energy, it dies of hunger. Bugs which have a lower minimum energy requirement (DET) may reasonably be expected to have a higher chance of reproducing, all other genes being of equal strength.

Note that a bug may die due to either DAT or DET.

RAT - on the turn in which the bug reaches this age, presuming that it is sufficiently energized (see RET), it will reproduce. Bugs which are able to reproduce at a younger age, all other things being equal, have a greater probability of out-competing their contemporaries in the contest for resources.

RET - on the turn in which the bug achieves this level of energy, presuming that it is of sufficient age (see RAT), it will reproduce. Bugs which are able to reproduce at lower energy levels, all other things being equal, have a greater probability of out-competing their contemporaries in the contest for resources.

Note that a bug must meet both criteria (RAT & RET) to reproduce.

EPM - on each turn, a bug expends this amount of energy as it moves. A bug which stands still due to the expression of gene8 (the Stand Still gene) will expend half this amount of energy (a bonus for standing still). A bug which attempts to move but is blocked (by another bug or an edge) nevertheless expends EPM of energy. Bugs which have a lower EPM will waste less energy on movement and will conserve more energy for reproduction, and thereby have a greater probability of out-competing their contemporaries in the contest for resources, all other things being equal.

EPB - on each turn, if the energy store of a bug exceeds this amount, it ceases to be hungry and ceases to feed, until the energy store is reduced below this level again. Bugs which have a high EPB will pass on more energy to their daughters, or have a greater reserve to carry them through lean times, thereby increasing their probability of out-competing their contemporaries in the contest for resources.

Because these genes regulate the life functions of the bugs, they are collectively called the regulatory genes. Each of these regulatory genes will be coded as a simple gene, similar to the coding of the Palmiter genes, in which a base and an exponent are used to compute the strength. By default, the base is 1.1. Note that the base is now 1.1, not 2.0, and the exponent can be any real number, not just integers. Why use a base of 1.1? Somewhat arbitrarily, we postulate that a 10% change in gene strength per mutation gives us a good balance between fast adaptation and graininess of solutions within the solution-space.

Using the above default strengths as startup values will allow the PSoup program to replicate the behavior of Dr. Palmiter's program at Level 1. But encoding these parameters as genes will allow PSoup to exhibit more sophisticated adaptability in the higher levels in which these parameters are allowed to come under evolutionary pressure.

PSoup and Plant-Like Phenotypes

How Does Gene8 Work?

Gene8 is a Palmiter gene, similar to the other eight in many ways. When a bug takes its turn, gene8 is summed with the other genes, and the pseudo-random number generator selects a gene for expression. If gene8 is expressed, the bug (a) does not turn (b) does not move, and (c) expends EPM/2 units of energy.

Suppose we want to emulate plant-like behavior in a bug in a bowl of PSoup. How might we do this? In a Level 3 bowl of PSoup we are introduced to a ninth Palmiter gene. In effect the eight Palmiter genes seen in Levels 1 & 2 can each be associated with a neighboring cell. Eight neighboring cells; eight genes. But plants don't move into neighboring cells, and with the approach seen in Level 1, a bug MUST move on every turn. By adding a ninth gene which, when expressed, says 'don't move', then we at least have the opportunity to evolve bugs that exhibit little or no movement. This gene is variously called gene8, or the Stand Still gene, and is represented in a PSoup genetic profile as 'SS'.

Do Bugs Ever Actually Stand Still?

No. Due to the way the Palmiter genes work, no bug ever stands still forever. All genes have a positive strength, although it may be very very small. There is therefore a very remote possibility that any gene, no matter how small, will eventually express itself. So, even if a bug has a very very strong gene8 and tiny strengths for all other genes, it will occasionally move.

Forced Moves

Bugs that are inclined to be sessile may otherwise be obliged to move under what is called a 'forced move'. In PSoup the natural state of a cell is to be occupied by 0 or 1 bugs, but not by 2. However, when a bug undergoes fission or gives birth, there will momentarily be two bugs in the cell previously occupied by the mother. Any bug which is found to be in a cell with another bug is forced to attempt a move out of the cell. The bug is first given an opportunity to move out of its own volition and under normal PSoup rules. If the bug fails to leave the cell, then PSoup scans for suitable neighboring cells. A suitable cell is any cell with either 0 or 1 bug in it. PSoup then chooses a random cell from among those available, and moves the bug into the suitable cell. Note that the bug may have gone into yet another doubly-occupied cell, and may face a forced move again if the other bug does not leave in the mean time. When any such gypsy bug finally finds an empty cell of its own, it will no longer be under forced move orders, and will only move when its genes say it must, and when it can. If a bug under forced move has no empty or singly-occupied cells into which it can move, it stays where it is until next turn, when it will try again.

When Might Immobility be an Advantage?

In bowls with very low levels of energy loading, bugs with efficient search patterns dominate. In bowls with very high levels of energy loading, bugs do not need to search for algae; it comes to them. In such a bowl, bugs which conserve energy (low EPM) have an advantage. Note that, when the Stand Still gene is expressed, a bug expends only half of EPM, whereas any other Palmiter gene consumes a full EPM each time it is expressed. The combination of low EPM and high Stand Still gene makes for a very energy efficient bug.

In the Gallery

In Level 3 you are introduced to a gallery of genotypes with plant-like names: the bush, the dandelion, the potato. Each of these has a strong Stand Still gene and all other Palmiter genes are weak. The difference between the three genotypes is in their C2 genes:

- Bush - A bush has enhanced stamina.
- Dandelion - A dandelion has enhanced fertility.

- Potato - A potato has enhanced agility. While this appears to be nonsense, it results in a plant with high capacity to collect and store energy, which sounds a bit like a potato.

About Chromosome Penalties

PSoup has the ability to evolve its denizens endlessly. If you were start a simple scenario, say Scenario 1A, and run it and never turn it off the bugs would continue to evolve forever. A stable phenotype would quickly emerge, but subtle variations on that phenotype would continue to occur, and the evolutionary forces would select among those variations, driving the development of ever more extreme values. Those genes of deleterious value would be made ever smaller in strength, and those of benefit would be made ever greater, in proportion. Ultimately the computer would no longer be able to represent the strengths, as they would be too small or too large to be accurately rendered.

These problems are a type of edge effect. They are handled in PSoup in two ways.

Large Number Thresholds

The first is not very subtle. If a mutation occurs which produces a number which is too large or too small (i.e. beyond pre-determined thresholds) then that mutation is disallowed. Effectively, the evolutionary pressures hit an abrupt immovable wall. This avoids computer failure, however, the resulting edge effect so overpowers any evolutionary pressures that further development is of little interest.

The thresholds are:
- Base numbers of genes must never be less than 1 or greater than 50;
- Exponents of genes must never be less than -150 or greater than +150;

Admittedly, that leaves a lot of room, and it would be a rare incident when PSoup oversteps these thresholds, but not so rare as you might imagine. Under strong evolutionary pressures, bugs can breach these thresholds in a few hundred generations.

Evolutionary Penalties

The second approach is more subtle. PSoup uses evolutionary penalties to steer evolving bugs away from the edges of the logical space in which they exist. In a fully functional Level 6 bug, there are three types of chromosomes and three types of chromosome-related penalties which limit the evolution of the bugs.

The C1 penalty is computed from the gene strengths in type 1 chromosomes. A type 1 chromosome contains a set of nine Palmiter genes. The C1 penalty is calculated as [the difference in strength between the strongest and weakest genes] divided by [a scaling factor]. This gives us a positive number proportional to the spread in strengths, and inversely proportional to an arbitrary scaling factor. This number is added to EPM (energy per move) on each turn and the bug that is assigned this penalty must expend that much additional energy on each and every turn. Note that, if the scaling factor is small, this can be a very stiff penalty. The evolutionary pressures on EPM is very high, and any small penalty applied there will certainly be noticed.

The C2 penalty is computed from the gene strengths in type 2 chromosomes. A type 2 chromosome contains a set of six regulatory genes (DAT, DET, RAT, RET, EPM, EPB). The C2 penalty is computed as the absolute value of [the sum of the stamina, the fertility, and the agility] divided by [a scaling factor]. Note that each of these three numbers may be positive or negative. When you sum them, the sum will be either positive or negative. The absolute value of this, is, of course, a positive number. The C2 penalty is applied to the EPM, as is the C1 penalty.

The C3/4 penalty, introduced in Level 5, is [the difference in strength between the strongest and weakest capability types] divided by [a scaling factor]. This positive number is applied to the EPM, as for the other two penalties.

The default scaling factors are:

- C1 Penalty 20,000
- C2 Penalty 200
- C3/4 Penalty 20,000

The wide variance in size is due to the scaling factors already built into the calculations for stamina, fertility and agility. The scaling factors are under user control and can be set at the beginning of any scenario by using the 'Tinker with the Environment' command.

About PSoup's Energy Systems

In PSoup, energy is measured in Nrg Units. One Nrg Unit is equal to 1/40th of a colony of algae.

The energy system in PSoup has three components. Energy may be stored in any one of the three:
- nutritive mud - which is grayish in color (although the color varies from paddy to paddy), and which stores all loose energy. Energy in the nutritive mud is like energy bound up in minerals in the natural world. It is generally unavailable to bugs for use until it has been processed by algae. The energy in the nutritive mud can be any real number from 0 to infinity (or as big a number as your computer can handle).
- algae - which live in greenish colonies, one colony per cell maximum. A colony of algae has exactly 40 Nrg Units of energy. Energy travels from the ubiquitous mud into the algae by a process of random distribution which is under the control of the parameter called 'Algae Distribution Mode'. Energy which goes into the algae from the mud stays in the algae forever, unless eaten by a bug.
- bugs - which are multi-colored. Bugs eat algae and/or other bugs. When bugs move they expend an amount of energy equal to their EPM gene strength (a gene in the type 2 chromosome). EPM stands for Energy Per Move. If the Stand Still gene was expressed, the energy expended is EPM/2. When bugs die, all energy remaining in them is expended.

What is a Closed Energy System?

In a closed energy system there is an energy cycle. No energy enters or leaves the system during the run of a scenario. When bugs expend energy, it returns to the nutritive mud to be re-distributed as algae. The only energy entering the nutritive mud comes from bug moves or bug deaths. All expended energy is so recycled.

The nature of the PSoup energy system (open or closed) is determined by the Algae Distribution Mode. The energy system is closed if and only if the Algae Distribution Mode is 'normal' or 'cascade' or 'Oasis first'. In normal mode, the algae is distributed randomly across the bowl of PSoup, whether it is a single paddy, or many paddies. In cascade mode, there must be more than one paddy, and the algae is distributed to the high priority paddies first, the priority order starting with the highest on the left and decreasing to the lowest on the right (see About Paddies). In 'Oasis first' mode, the algae is distributed to the oasis first, and, failing placement, it is distributed to the desert areas.

A closed energy system is self-regulating, in that the inflow of energy into the nutritive mud exactly equals the expenditure of the bugs. There is never an excess or a shortage in this respect. Closed energy systems are characterized by populations that grow rapidly to some level, and then hold steady at that level, give or take a few bugs. Closed energy systems are remarkably stable.

Note that, in the natural biosphere, there is no such thing as an energy cycle. In PSoup, there is, when you have a closed energy system. The Nrg units cycle through the system, from mud to algae to bugs and back to the mud.

What is an Open Energy System?

In an open energy system there is no energy cycle. Energy enters the system in the form of sunlight striking and warming the nutritive mud. Energy is captured by algae and stored. Energy transfers from the algae to the bugs when the bugs eat the algae. When bugs move or die, energy is expended, and lost from the system. When the energy is lost, it is dissipated, and never returns to the system.

The PSoup energy system is open if and only if the Algae Distribution Mode is 'steady' or 'variable'. In steady mode, a fixed number of Nrg units are inserted into the nutritive mud at the beginning of each and every PSoup second. This happens whether energy is being dissipated or not. In variable mode the amount of energy that is inserted into the system each second is calculated on a second-by-second basis. The user specifies the average distribution rate, the length of a season, and the maximum percentage variation from

the average. PSoup uses these three parameters to create a sinusoidal pattern of incoming energy. The full cycle of the sine wave takes four seasons to complete. This is vaguely analogous to the pattern of sunlight in temperate zones in the biosphere in the annual cycle.

In open energy systems, the inflow of energy is independent of the outflow of energy. These systems are not self-regulating, in this respect. It is possible to have huge stores of energy built up in the nutritive mud in periods when supply outstrips demand, or to have empty plains of algae-free cells, when demand outstrips supply. Open energy systems are characterized by wild swings in the population levels, and often result in the total collapse and extinction of the population of bugs.

Note that, in the natural biosphere, there is one open energy system. Energy is received from the sun (or from geothermal sources), is used temporarily by a few organisms, then is dissipated into space.

About Combat Profiles

Combat in PSoup is not visible, it happens behind the scenes. However it is complex. It is managed via a mechanism called a combat profile.

Combat profiles play a key role in four out of five of the life functions of each bug which has any capability activated:

Sensing - When bug A senses an edge cell, a colony of algae, or another bug, a combat profile is built for each such sensed item.

Edges and algae have simple profiles. Mud is ignored.

Bugs have complex profiles. The first time two bugs meet, two profile stubs are built: one for bug A taking initiative against bug B (call this the AB profile) and one for bug B taking initiative against bug A (call this the BA profile). These stubs contain comparisons of the genetic strengths of the bugs for all nine potential capabilities. Suppose we look as 'SIGHT'. Bug A has a fight character and a flight character, call them Afi and Afl. Bug B similarly has Bfi and Bfl. In profile AB we store (Afi - Bfl) or 0, whichever is greater. In profile BA we store (Bfi - Afl) or 0, whichever is greater. We do this for all nine capabilities. In addition, the relative distance and direction from A (the bug with the initiative) to B is stored in the AB combat profile and nine 'distance-reduced' premiums are added to the AB combat profile. This profile is no longer just a stub. When these two bugs meet again later, the stubs are located and re-used, only the temporary distance and direction-related information is updated, and only for the bug with the initiative.

Moving - When bug A moves, it uses its Palmiter genes to determine the direction of the move, but the strengths of these genes have 'premiums' added to them before they express. Bug A goes through its list of active combat profiles, and, for each one, computes an appropriate premium to be added to an appropriate gene. This is dependent on the location and direction of Bug A, and the location and direction of the entity profiled.

Let's work through a few examples. Suppose that bug A is in cell (8, 10), i.e. column 8, row 10. Suppose also that bug A is pointing 'UP'. Bug A would handle profiles as follows:

Example - Edge Cells - Note that, while edge cells can be detected by a Level 4 bug, they cannot decide to avoid them unless and until the 'THINK' capability has been activated. The student user is not yet allowed to endow a bug with such ability. Let's assume that 'THINK' has been activated, and that bug A has detected edge cells at (5, 7), (5, 8), (5, 9), (5, 10), (5, 11), (5, 12), and (5, 13); seven in total. Bug A has seven combat profiles for edge cells. For each combat profile, a 'decision' is made to avoid the edge, i.e. to move away from it. If bug A expressed gene7 (FL) it would point to, and move towards, the edge cell at (5, 7). However, if it expressed gene3 (BR), it would point away from, and move away from the edge cell at (5, 7). So, bug A sums up all of the distance-reduced premiums in the combat profile for the (5, 7) edge cell and adds them to the strength of the Palmiter gene gene3. This increases the probability that the bug will go away from this cell. The size of the increase in probability is determined by the strength of the genes. Similarly, each combat profile is processed. At the end of this exercise, the bug has an enhanced probability that it will move away from the edge.

Example - Algae Colonies at (6, 7) and (7, 7), two in total. Bug A has two combat profiles for algae. For each combat profile, bug A determines which gene needs to be given a premium to take it towards that cell, computes the distance-reduced premium, and adds that to the value of the selected Palmiter gene.

Example - Mate at (6, 8). Bug A 'has an instinctual urge' to pursue the potential mate, and enhances gene7 with a distance-reduced premium computed from the combat profile.

Example - Prey at (6, 10). Bug A 'has an instinctual urge' to chase the prey, and enhances gene6.

Example - Predator at (6, 12). Bug A 'has an instinctual urge to run away from the predator, and enhances gene1.

Example - Host at (8,12). Bug A 'has an instinctual urge' to approach the host, and enhances gene4.

The original search pattern that was encoded in the Palmiter genes is still there, but it has been heavily overlaid with information coming from the bug's capability genes, information that is specific to the current situation in which the bug finds itself. In this circumstance, type 1 chromosomes are largely irrelevant, and the genes in the type 3/4 chromosomes take charge. The bug has many 'instinctual urges' along the lines of 'if I go this way, it's good/bad'. Each such 'urge' is recorded as an **I**nstinctual event, and the student can watch them rack up at an astounding rate in the Event Counts panel. Note that the bug still HAS NOT MOVED. It has merely instinctively processed the situational information available to it.

Now, bug A uses the usual method to decide which Palmiter gene to express, and proceeds with its move. Note that the probability of selection of each Palmiter gene has been modified by the capability genes. If you turn on the 'Life Function' panels, and pace through a bug's move one function at a time, and watch the C1 genetic profile, you will see the base Palmiter profile projected during the sensing function, and the altered Palmiter profile projected after the moving function has been executed.

Feeding - Bug A now finds itself in a cell (called its current cell) and facing, or pointing towards, another cell (called the target cell). Bug A now goes through the following logic:
- If not hungry, stop feeding and do nothing;
- If the target cell has a prey or host or parasite bug or potential mate, attack it, and combat ensues. Note that a host bug may attack a parasite, and eat it, but only if it is in the target cell. A bug neither pursues nor avoids parasites, but eats them opportunistically, as if while cleaning itself;
- If the target cell is empty, attack it 'just in case' it is occupied by an undetected prey or host or parasite bug or mate. If there is an undetected prey or host or parasite bug, combat ensues. Note that a host bug may eat a parasite. If it is occupied by a suitable mate, mating occurs;
- If no combat has occurred yet, check the current cell to see if there is any algae therein, if yes, eat it.

Prey and parasites are usually eaten in toto, if attacked successfully. Host bugs are bitten (severely) but normally not killed. A host bug will die only if its energy store goes below its DET. When attacked, prey or host bugs can defend themselves with a shell or spines (TOUCH capability) or a powerful odor (SMELL capability) or a poisonous taste (TASTE capability). Bugs that use the SMELL or TASTE defenses are partially eaten, and the combat profile is altered to indicate declining interest by the attacking bug. The combat profile contains probability factors which are genetically determined and which are used to compute the probability of success of each specific attack. The pseudo-random number generator is used to determine who wins an attack.

Reproducing - At Level 4, bug A uses fission to reproduce. The combat profiles play no role, yet. At higher levels, information that was computed and stored in the combat profile during sensing is recalled and used during mating. If the combat profile does not exist (e.g. for undetected mates who happen to be in the target cell) then the bugs mate opportunistically, without the benefit of a combat profile. It takes a micro-second longer, but it's still fast.

Dying - The combat profiles play no role.

The Evolution of the Senses

In Level 4 there are five scenarios each of which introduces a genotype with pre-endowed capabilities. To learn more about each capability type, follow the links below. To learn more about capability genes in general, see **About Capability Genes**, or see **Type 3/4 Chromosomes**.

The introductory scenarios are:

Scenario	**Capability Type(CType)**
- Scenario 4A - Out of Sight, Out of Mind	SIGHT
- Scenario 4B - Smell Like a Pig	SMELL
- Scenario 4C - Hear Like a Shrew	HEAR
- Scenario 4D - Royal Taste (Feeding Frenzy)	TASTE
- Scenario 4E - A Touch of Death	TOUCH

In addition, in PSoup there is a sixth sense used by XOver and OVULE-enabled bugs to sense potential mates. These are introduced in:

Scenario	**Capability Type(CType)**
- Scenario 5A - Recombination Via XOver	XOVER
- Scenario 5B - Recombination Via Mating	OVULE

What phenotypic characteristics are associated with each capability, and how do they interact? That is the topic of this article.

First, obviously, each bug gains the named sense for those capabilities enabled by its genes. Each sense has two sides, an aggressive side (called the fight character) and a defensive side (called the flight character).

SIGHT

Sight is the most powerful sense.

Fight - A seeing bug can sense a scan area of 7 cells by 7 cells. It can see edges (edge cells of the bowl, or left-hand edges of paddies), mud, algae, and any bug whose appearance is not sufficiently camouflaged.

Flight - A seeing bug can also camouflage itself against detection by sight. Only seeing bugs have this defense.

SMELL

Smell is not as powerful as sight. A bug with only smell cannot detect other bugs.

Fight - A SMELL-enabled bug can sense a scan area of 7 cells by 7 cells. It can smell mud or algae. It cannot smell edges or other bugs.

Flight - A smelling bug can use powerful odors as a defensive strategy in a melee, or it can prevent being smelled by another SMELL-enabled bug.

HEAR

Hear is similar to smell. A bug with only hear cannot detect algae.

Fight - A hearing bug can sense a scan area of 7 cells by 7 cells. It can hear edges (by echo-location) or any bug whose sound is not sufficiently masked.

Flight - A hearing bug can be stealthy, and avoid being heard by other hearing bugs.

TASTE

Taste is a weak senses.

Fight - A tasting bug can sense a scan area of 3 cells by 3 cells, i.e. only covering those cells into which it might move on the next turn. It can taste mud and algae.

Flight - A tasting bug can use powerful taste (poisons) to repel attackers, or to reduce their damage.

TOUCH

Touch is a weak sense.

Fight - A touching bug can sense a scan area of 3 cells by 3 cells. It can touch edges, or any bug which has not sufficiently camouflaged its texture. It can use claws and teeth in a melee to gain advantage when attacking.

Flight - A touching bug can avoid detection by touch. It can use its shell or spines to defend itself when attacked.

XOVER

XOver provides the ability to sense potential mates. This sense is active only if this bug is in heat. An XOver-enabled bug is in heat at the moment of birth and stays in heat until it has successfully stolen a set of chromosomes from another suitable target bug. This need to find a suitable mate is a high priority activity. Once a mate is found and genes are successfully stolen, the bug is no longer in heat and the ability to sense potential mates disappears. Potential mates must be:
(a) - alive; that is to say, these bugs are not choosy;
(b) - haploid;
(c) - of suitable complexity; such that $0.5\ C < X <= 2.0\ C$ where X is the complexity of the target bug and C is the complexity of the XOver-enabled bug in heat and looking for a mate; and
(d) - not masked by a large XOver flight capability.

Fight - An XOver-enabled bug can sense a scan area of 7 cells by 7 cells. It can sense any eligible mates which have not sufficiently camouflaged their status (by a strong XOver flight capability).

Flight - An XOver-enabled bug can avoid detection by bugs with weak mating instincts (i.e. a weak XOver fight capability).

OVULE

OVULE provides the ability to sense potential mates. This sense is active only if this bug is mature. An OVULE-enabled bug is mature if it has (a) previously mated, or (b) achieved an energy level greater than 0.6 of its RET. Mature bugs search for a mate if they are not yet pregnant. This need to find a suitable mate is a high priority activity. Once a mate is found and impregnation occurs, the motivation changes slightly. The bug now searches for and pursues other bugs currently in heat and attempts to impregnate them.

For an OVULE-enabled bug **in heat**, a suitable mate is:
(a) - of suitable complexity; such that $0.5\ C < X <= 2.0\ C$ where X is the complexity of the target bug and C is the complexity of the OVULE-enabled bug in heat and looking for a mate;
(b) - OVULE-enabled;
(c) - mature;
(d) - not masked by a large OVULE flight capability;
(e) - of similar feeding habits.

For an OVULE-enabled bug **not in heat**, a suitable mate is:
(a) - of suitable complexity; such that 0.5 C < X <= 2.0 C where X is the complexity of the target bug and C is the complexity of the OVULE-enabled bug in heat and looking for a mate;
(b) - OVULE-enabled;
(c) - mature;
(d) - not masked by a large OVULE flight capability; and
(e) - In heat;
(f) - of similar feeding habits;

Fight - A mature OVULE-enabled bug can sense a scan area of 7 cells by 7 cells. It can sense any eligible mates which have not sufficiently camouflaged their status.

Flight - A mature OVULE-enabled bug can avoid mate-based detection by bugs with weak mating instincts.

About Sensing Algae

All holotrophic bugs sense algae if they are standing on top of it. Any non-carnivorous hungry bug which enters a cell containing a colony of algae will sense and eat the algae. Carnivorous bugs can neither sense nor eat algae. However, bugs which have the ability to sense algae at a distance (via SIGHT, SMELL or TASTE), also have the ability to move towards any sensed colony of algae, in the hopes of getting there first and eating it. Each of these three senses work independently of each other.

About Sensing Bugs

Bugs which have SIGHT, HEAR or TOUCH can sense other bugs. Bugs which have XOVER can sense suitable mates. Mature bugs with OVULE can sense suitable mates. If it is bug A's turn, and bug B is within bug A's scan area, bug A will detect bug B if, and only if, for at least one of the five bug-detecting senses (SIGHT, HEAR, TOUCH, XOVER or OVULE), bug A's fight strength is greater than bug B's corresponding flight strength, checked on a per capability basis. If bug B does not have that sense activated, then its flight strength for that sense is considered to be 0. To completely avoid detection by a fully-enabled Bug A, Bug B must have a strong flight capability for all five capabilities. As Bug A scans, it will then build a combat profile for each detected bug in its scan area. Each of these senses work independently of each other, but once a bug is detected, it is immaterial by which sense it was detected.

When bug A has detected another bug, it decides whether this bug is a parasite, prey, a predator, a host, or of the same herd as bug A. Herd bugs are then further classified as potential mates, or of no interest. Bug A may then decide to approach or run away from the detected bug. Bug A will pursue mates, prey or hosts. Bug A will flee from predators.

For more technical information about bugs sensing bugs, read the article on 'About Combat Profiles'.

Advanced Reproductive Modes (PSoup Theory and Practice)

In Level 5 there are two scenarios each of which introduces a genotype with pre-endowed capabilities. To learn more about each capability type, follow the links below. To learn more about capability genes in general, see **About Capability Genes**, or see **Type 3/4 Chromosomes**.

The introductory scenarios are:

Scenario	**Capability Type(CType)**
- Scenario 5A - Recombination Via Crossover	XOVER
- Scenario 5B - Recombination Via Mating	OVULE

Sensing Mates

Note that XOver and Ovule enable a sixth sense, the ability to sense a mate. Here is how it works.

XOver - If a bug is XOver-enabled, is of age, and has not yet stolen suitable genetic material to execute the cross over, then it becomes sensitive to potential sources of genetic material. Any bug of suitable complexity (i.e. is haploid and for which the complexity X is in the range $0.5C < X <= 2.0C$ where C is the complexity of the XOver-enabled bug) is suitable as a source, and will be sensed as potential mate. XOver-enabled bugs will pursue such potential mates with the intent to steal a copy of their genes.

OVULE - If an OVULE-enabled bug is of age, has sufficient energy, and is not pregnant, then is will search for a suitable mate. Any bug which is of suitable complexity (see above), is OVULE-enabled, is of age, and shares the same feeding habits will be pursued for its sperm. Even if the OVULE-enabled bug is pregnant, it, being hermaphroditic, will pursue other suitable bugs in heat and attempt to impregnate as many as possible.

If XOver-enabled or OVULE-enabled bugs are also able to sense other bugs by SIGHT, HEAR or TOUCH then the objects of their desires are pursued all the more vigorously.

What other phenotypic characteristics are associated with each capability, and how do they interact? That is the topic of this article.

First, obviously, each bug gains the capability enabled by its genes. Each capability has two sides, an aggressive side (called the fight character) and a defensive side (called the flight character).

XOver

Fight - An XOver-enabled bug can sense, pursue and steal the genetic material of any other bug of similar complexity.

Flight - An XOver-enabled bug can also defend itself against detection by another XOver-enabled bug, assuming that it cannot otherwise be sensed. (Question? Would evolution favor a strong defense, or no defense?)

An XOver-enabled bug which is ready to fission, but which has not yet captured genetic material from another bug, will undergo regular fission.

Ovule

Fight - An Ovule-enabled bug can sense, pursue, and initiate mating with other Ovule-enabled bugs of similar complexity.

Flight - An Ovule-enabled bug can also defend itself against detection by another OVULE-enabled bug, assuming, again, that it cannot otherwise be sensed.

An Ovule-enabled bug which is ready to give birth but has not yet been fertilized, will undergo virgin birth via parthenogenesis.

About Combat Profiles

Combat profiles are used to determine the success or failure of an attempt to sense a mate.

Higher Functions

In Level 6 there are two scenarios each of which introduces a genotype with pre-endowed capabilities. To learn more about each capability type, follow the links below. To learn more about capability genes in general, see **About Capability Genes**, or see **Type 3/4 Chromosomes**.

The introductory scenarios are:

Scenario	**Capability Type(CType)**
- Scenario 6A - Herding	HERD
- Scenario 6B - Thinking	THINK

Note that, while HERD and THINK may be activated by the capability genes, they may be ineffective. A bug cannot execute a herding or thinking action unless it can detect other entities. These higher functions are only effective if in combination with the ability to sense other entities.

What phenotypic characteristics are associated with each capability, and how do they interact? That is the topic of this article.

First, obviously, each bug gains the capability enabled by its genes. Each capability has two sides, an aggressive side (called the fight character) and a defensive side (called the flight character).

Herd

Fight - A non-hungry herding bug prefers the company of other bugs of similar complexity.

Flight - A non-hungry herding bug prefers to be left alone.

Think

Fight - A thinking bug can (a) avoid edges and (b) plan a best route when a decision is made.

Flight - A thinking bug can diminish the value of the plans of an attacking bug.

PSoup and Emergent Behavior

Emergent behavior is a characteristic of complex systems. A complex system is said to exhibit emergent behavior if its overall behavior is more than would be expected based on a review of the behavior patterns of the constituent components. Or, the whole is more than the sum of its parts.

In the physical world liquid and gaseous turbulence can be viewed as emergent behavior. A study of the kinetics of individual molecules will not give you insight into turbulent movements. Even a thermodynamic study of averaged behavior will give you scant insight into this domain of knowledge. Recent advances in the study of non-linear dynamic systems and complex systems are now starting to provide some new and deeper insights.

In the natural organic world, human consciousness is considered an emergent property. You can examine muscle, bone, blood vessels, neurons and electrical activities, but these do not give any hint as to how consciousness might arise. The flocking of birds, the schooling of fish, the social order of ant and bee colonies, the herding of cattle, the social (in the broadest sense) interactions of people; these are all emergent properties.

PSoup is able to demonstrate emergent behavior, but it is often masked in the hurly-burly of competition for survival. It is most clearly evident in those scenarios in which the distracting noise is eliminated and the opportunity to recognize the emergent behavior is enhanced. In Scenario 5B ('Recombination Via Mating') and Scenario 6A ('Herding, an Emergent Property') we have those conditions.

Feeding Frenzies

Let's examine Scenario 4D first. In this scenario we have carnivorous birds and foul-tasting worms. The birds have a dilemma. The worms, their only source of food, are truly foul-tasting, and the birds have no defense against this. When a bird first bites a worm, it spits out 80% of the worm (having consumed 20%). It now remembers that this worm is awful. The worm also gains some ability to resist further attack from this bird. The bird may bite the worm several more times. Eventually, the worm is able to resist all attacks from this bird, due to its TASTE defense, and suffers no more damage from this bird. In the mean time, the bird continues its attacks for a few more times, until it learns that it is both futile and foul to eat this worm. The worm leaves somewhat damaged, but still alive. The bird leaves somewhat fuller, but having one less worm that it can effectively attack.

Watch what happens as the scenario unfolds. We eventually get to a state in which there are many birds and not many worms. The birds see the worms and flock to them. The worms fight back, rather effectively. Eventually, the attacks of the birds become useless, but the sated birds cannot depart the feast due to the press of hungry birds outside the ring. We then get situations in which one or two worms are seen at the center of a flock of birds, severely damaged, but only receiving a occasional peck from a bird.

This type of flocking around a food source is analogous to a feeding frenzy in nature. It can be considered an emergent behavior, as it is created by unrequited greed and hunger in a large crowd.

Mating Swarms

Let's examine Scenario 5B next. In this scenario all bugs have a complexity of 10. Complexity mutations are not allowed so all bugs will always have a complexity of 10. They are OVULE-enabled and all are able to sense, pursue and mate with other bugs. All are SMELL-enabled, so all have equal ability to forage for and find green algae. Each of these bugs is always on the lookout for another bug in heat. When it detects such a bug within three cells of its own position, it is inclined to move towards that bug, and mate with it. If it is also in heat, it will also attract the attention of other bugs, and will draw them towards itself, even as it pursues the bug in heat.

In summary, when these bugs are in heat, they pursue any possible mate. When these bugs are not in heat, they pursue only those possible mates that are in heat.

Once contact is made and mating is complete, the mated bugs are no longer in heat and no longer attract each other.

In the normal hurly-burly of a bowl of PSoup, a bug may come into heat at any time, and, if lucky, will be detected by another suitable mate and will be mated. If not, when it comes time to reproduce, it will fertilize itself and reproduce via parthenogenesis. The mating ritual is normally brief and perfunctory, lasting 2-3 PSoup seconds at most. The participants then lose interest in each other and wander off in search of other algae, or other bugs in heat.

However, in Scenario 5B, because we have all bugs starting life in synchronization with each other, there are sudden changes of behavior en mass. All bugs start in heat. All immediately pursue each other, and amass into large mating swarms, much like midges. As the mass grows in size, the attractive property grows in strength, for those bugs within 3 cells of distance, and they become stuck together in a large blob. After 2-3 moves most are impregnated, and a few try to wander off, but most remain trapped by the bugs in heat still being drawn into the swarm. Eventually, the last few bugs in heat are impregnated, and all bugs immediately lose interest in each other and the swarm disperses. The bugs now pursue algae in competition with each other. However, as soon as they come of age, they give birth and come into heat again. The mating swarms reappear.

Herds of Herbivores

Similarly, in Scenario 6A we have bugs that are of equal complexity but have no interest in each other for sexual purposes. They reproduce by normal fission, having neither XOver nor OVULE capabilities enabled. They are herbivores, and are content to search for food until sated, then wander around while they digest. To achieve this they are SMELL-enabled. When they get hungry, they move off in search of more food. However, when sated with food, they are inclined to locate and position themselves with respect to any bug of comparable size. To position themselves, they take two possible types of action:
- If they have a more dominant HERDing capability, they align their direction with the less dominant bug and then try to travel at a comfortable distance beside the other bug (one or two spaces distant).
- If they have a less dominant HERDing capability, they take a leadership role by forging straight ahead, not turning left or right, making it easier for herding bugs to align with them and travel beside them.

All bugs start hungry. All are equally adept at finding food. Food starts as plentiful, so there is no real competition. All become sated at about the same time. All start looking for companions with whom they may wander around while they digest their food. Herds start to form. All become hungry again at about the same time, and the herds disperse again.

The desire to digest food near a friend bug is directed by the HERD capability genes. The formation of large herds of wandering herbivores is an emergent property that depends on the synchronization of hunger and satiation cycles and the availability of food.

PSoup and Boids

The following discussion gets quite technical. For those with an interest, carry on.

The herding capability in PSoup is modeled somewhat distantly on the flocking concept developed by Craig Reynolds in 1989, and implemented in his program called 'Boids' in which Birdoids take flight in flocks. In that program virtual birds flock together, fly together in a flock, and navigate around obstacles as a flock.

In PSoup we want bugs to organize into 'herds' and travel across country much like a herd of bison might. But in PSoup we do not want to make herding behavior a priori the behavior of our bugs, as it is for the Birdoids. We want to make it a capability that will develop only when the circumstances provide an

evolutionary advantage to bugs having the capability. From a technical point of view, there are two issues to address. First, there must be a set of genes that activate the behavior. This was simple. Second, the herding behavior must be integrated into all of the other motivating factors that drive a bug's behavior, which is expressed through the Palmiter genes.

PSoup defines each HERD-enabled bug to have a social momentum. You can think of social momentum as an urge, or instinct, to keep moving. Do not view it as either speed or physical momentum. Bugs have only two speeds: go (1 cell per second) when any of the first eight Palmiter genes are expressed, and stop, when the stand-still gene is expressed. So speed and physical momentum are not sufficiently useful discriminators for our purposes.

Let's review how the Palmiter genes work. For simplicity, suppose the stand-still gene is inactive and has a strength of zero. It then plays no role. Let's call the first eight genes the 'go' genes for the purposes of this discussion. Each go gene has a gene strength, and the relative strength of that gene when compared to the sum of the strengths of all eight active Palmiter genes determines the probability that the bug will turn and step in the direction of the gene. Let's look at an example. Suppose that each of the go genes has a strength of 2, except for gene 'FR', the 'Forward Right' gene, which has a strength of 4. The probability that the FR gene would be expressed is 4 / (2 + 4 + 2 + 2 + 2 + 2 + 2 + 2) = 2 / 18. This bug has an equal probability of heading in any of seven directions, but has double the probability of heading directly forward and to the right.

Now, let's endow this bug, call it Bug A, with the HERDing capability. The social momentum of this bug is the strength of the last Palmiter gene that was expressed. If the last move of the bug was to step straight forward, then the social momentum is 2. If the last move was to turn one turn to the right then step forward, the social momentum is 4. It is now time for the bug to move again. It scans a 7x7 area of cells in its immediate vicinity and notes all bugs that are of similarly complexity. That is, their complexity X is between half the complexity C of this bug, and twice the complexity of this bug. (0.5 C < X <= 2.0 C). All such bugs are noted as fellow members of the herd. Suppose there are three, which we will call Bug B, Bug C and Bug D. Each of these has social momentum equal to the strength of the last Palmiter gene to be expressed. Suppose the momentum is 2, 4 and 2 respectively. For each of the three of these bugs, Bug A petitions the bug with a request to join its herd, and, depending on the success or failure of the petition, takes one of the following two actions:
1. - If accepted, Bug A (a) notes the direction of travel of the other bug and notes the social momentum of the other bug. It then adjusts its own appropriate Palmiter gene effect upwards by the amount of the social momentum of the other bug. Suppose it was the FL gene. It now has an effective strength of 2 (its original strength) + 2 (other bug's momentum) = 4. (b) It also notes the distance between itself and the other bug and adjusts an appropriate gene effect upwards to increase the probability that it will move to the correct distance from the other bug. Suppose it was the FR gene, which now has an effective strength of 6.
2 - If rejected, Bug A uses its own social momentum to strengthen the probability that it will go straight forward. Its F gene then has a strength of 4.

After reviewing all eligible bugs, several of its Palmiter genes will have greatly enhanced effective strengths. The bug now makes its move. The random-number generator picks a Palmiter gene to be expressed, and the bug turns and moves. The probability has been increased that the bug will behave either as a leader, or as a follower, in the herd. But, there is always the chance that an un-enhanced gene will be expressed and the bug will continue as per normal. If you watch the herds move, you will see a bug occasionally try to break from the herd. Some succeed.

However, think about what happens after a few turns. Suppose one leader (a) enhances his forward gene and then steps forward. It now has larger than normal social momentum, and is pointing out its direction. Next, suppose a follower uses that larger than normal social momentum to align itself with the leader. We now have two out of the four with a common direction and larger than normal social momentum. Eventually, the social momentum becomes enormous and spreads. As the newly formed herd moves out across the plains, other bugs are swept up in the social momentum and join the wandering herd.

Note that, at any time, any bug might express an un-enhanced gene, step out of range of the social momentum (3 cells) of the herd and break from the herd. This is the lot of bugs with low HERDING instincts on the edges of the herd.

HERDing behavior is expressed only if the HERD-enabled bug is not hungry and not being pursued or pursuing another bug (e.g. for mating). HERDing behavior is the lowest rung on the ladder of priorities for the bugs. So, what happens if you place a ravenous lion into the midst of a large herd of wandering antelope? Check out scenario 6B.

PSoup and Biased Mutations

In the natural world, the question is often asked as to whether the bias towards certain kinds of mutations influences the course of evolution. This is a very complex question, which cannot be resolved by reference via simple analogy such as we have implemented in PSoup. However, it is possible to gain a little insight into the nature of the question with a few experiments in PSoup.

DNA is coded as strings of four types of complex molecules. Mutations of DNA present themselves as altered strings. The alteration can be anything as simple as the deletion of one of these complex molecules, or its replacement by another of the three types. Or, it can be a complex mutation that breaks up one or several such strings and re-connects them in an altered order. These are classified as crossing over, inversions, multiple copies, etc.

Some mutations would seem to offer a high probability of producing a new strand of DNA that is useful (e.g. crossing over) while others would seem to have a high probability of simply disabling the DNA (deletions). Due to the rules of chemistry, and the circumstances of replication and maintenance of DNA, certain types of mutations are more likely to occur than others, and certain types of mutations are more likely to disable the DNA than others. Since the mutation of DNA provides the raw materials used by evolution to bring about change, the question must be asked, does the intrinsic bias towards certain types of mutations affect the course of evolution? At face, it would seem that it does, and it must. The deeper questions then are: why, and how?

Again, the genetic encoding used in PSoup is not intended in any way to emulate the physical encoding of DNA, so we cannot use PSoup to better understand DNA encoding. We can use PSoup to explore the response of an evolutionary system to biased mutation within its own system of genetic encoding.

Suitable Parameter Settings

We are going to execute a few thought experiments, but before we do, let's handle some technicalities required to set up the experiments.

Suppose that we have a standard Level 5 bowl of PSoup filled with Pristine bugs, that is, bugs for which all Palmiter genes have a strength of 1. Wrap is turned on. De-activate the Stand-Still gene. Turn off the 'probability of complexity mutations', and start all bugs with complexity equal to zero (i.e. no capability genes activated in the type 3/4 (C3 & C4) chromosomes). When we start the run, the genes in the type 1 and type 2 (C1 and C2) chromosomes will have the potential to mutate. So, even though this is a Level 5 bowl of PSoup, it will effectively operate at Level 3, the highest level before C3/C4 genes become active.

In the natural world, it is estimated that mutations occur only once in every 50,000 opportunities. In PSoup, they occur once in every eight opportunities (12.5%). This is vastly more often. The purpose is to demonstrate evolutionary forces in real time, as you wait, so the parameter is set at the highest value that will support the rapid but stable adaptation of the PSoup bugs. If you feed mutations too quickly into the hopper of natural selection, it does not have the clear opportunity to make a choice, and the results become chaotic and unpredictable. If you feed mutations in too slowly, the processes of natural selection eventually run out of options to work on and evolution is narrowly confined. See 'Scenario 5D - Evolution Without Mutation' for an examination of this issue. For our purposes for these thought experiments, we will leave the 'probability of mutation', call it M, set at the default value, M = 12.5%.

Next, you may recall that genes in PSoup have three parts: base, exponent, and strength. PSoup can allow mutations of either the base or the exponent, but the default at lower levels is that only the exponent can mutate. To simplify our thought experiments, we will set the 'probability of exponent mutation' to 100%, meaning that the exponent will mutate, when appropriate, and the base will not.

There is one final parameter in the PSoup rules of chemistry, and it is this parameter that we will be altering in the following thought experiments. The 'probability of mutation up' determines whether there is a bias towards an increase or decrease in the strength of the gene when it mutates. For reference, call this parameter 'U'. [Note that all such parameters range between 0 and 1000, and are multiplied by a factor of 0.1% to convert them into a percentage, which ranges from 0% to 100%.] If we set U to 500 (50%) then there is an equal probability that the exponent will mutate upwards or downwards. In either case the value will change by exactly one unit. So, for example, if the value of the exponent is zero (the default value at start, giving the gene a strength of 2^0 = 1), then, when it mutates, it will have a value of -1 or +1, each resultant value having an equal probability. If we set U at 250 (25%) then a mutation upwards by 1 would occur 25% of the time, and a mutation downwards would occur 75% of the time.

These probabilities compound on each other. The probability of mutation M is 12.5%. The probability of mutation upwards is invoked only when it has been determined that a mutation must occur. Then, and only then, U is invoked to determine whether the value of the exponent goes up or down by 1. So, if M = 12.5% and U=25%, 3.125% of the time a simple gene will increase in strength, 9.375% of the time the simple gene will decrease in strength, and 87.5% of the time it will not change. Since each bug has 8 active Palmiter genes, the chances are that one gene will suffer a mutation upwards, or downwards, as appropriate, on the birth of each bug.

Some Thought Experiments

In our five thought experiments we will set the value of U at five different values: 0% (0), 25% (250), 50% (500), 75% (750) and 100% (1000).

U = 50%

Let's start with 50% (500). We go into the 'Tinker with the Environment' command of the 'Tinker' menu, select the 'PSoup Rules of Chemistry' tab, and set the 'Probability of mutation upwards' to 500. We have a setup closely approximating a standard Level 3 bowl of PSoup. In 5-10 generations the Palmiter genes in the C1 genetic profile of the average bug start to show a definite pattern. The forward-facing genes become elongated, and the backwards-facing genes become shortened. While PSoup has no doubt faithfully produced an equal number of upwards mutations for the backwards-facing genes, as for the forwards-facing genes, nevertheless, there is clear selection of short at the back, and long at the front.

If there were no competition for resources and if all bugs had enough energy to reproduce as soon as they are old enough, then, by the third generation most Palmiter genes would be mutated for all bugs, and by the 6th generation there should be an array of genes of all strengths pointing in all directions. But, due to competition, half of the mutations are weeded out at every generation, and we have a rapidly emerging pattern of selected genes.

U = 100%

Set the 'probability of mutation upwards' to 100% (1000). In this instance, all mutations of the backwards-facing genes will strengthen the genes, and these would be bad. In the previous experiment only 50% of the mutations of these genes were in the wrong direction. This would lead you to think that a setting of U=100% is therefore going to slow down, if not stop, evolution. However, all mutations of the forwards-facing genes will also strengthen those genes, and these would be good. So the strong bias towards increasing the strength of these genes has some positive potential (lengthening the forward-facing genes) and some negative potential (no shortening the backwards-facing genes).

The phenotype of the bug determines the effectiveness of the bug. The C1 phenotype determines its search pattern, and the effectiveness of the search pattern. (We can safely ignore the effects of the C2 genes if we limit ourselves to a few generations). Is it possible that two bugs with widely different C1 genotypes can have the same C1 phenotype? Yes. If all of the exponents of the corresponding C1 genes are displaced by exactly the same amount, then the C1 phenotypes will be identical, and the search patterns will be identical.

Therefore, while it is very unlikely that our bugs in experiment #2 can produce an average C1 genotype close to the average C1 genotype of the bugs in experiment # 1, nevertheless, it is possible to produce an average C1 phenotype that is identical in every way from either experiment.

U = 0%

In our third experiment, we set U = 0% (0). In this instance, all mutations of the forwards-facing genes will shorten them (which is bad, since they will be selected against), and all mutations of the backwards-facing genes will shorten them (which is good, since they will be selected for). The situation is then similar to the situation in experiment # 2, but in reverse.

Again, it is possible, within a few generations, to produce the same average C1 phenotype in this experiment.

The question comes to mind, is it of equal evolutionary value (however you would measure that) to have (a) a lengthened F gene or (b) a shortened B gene? If either of these changes provide the identical advantage to a bug, all other things being equal, then we would expect the emergence of the same C1 phenotype at the same rate of development under a value of U = 0% as we saw at U=50% or U=100%. This in fact seems to be the case.

U = 25% or 75%

These two experiments are a little more complex to think through, although they produce the same results. After a few generations, it is still possible to produce the same C1 phenotype as produced in the other three thought experiments, even though the bias in the mutations is different.

But, What About the C2 Phenotype?

Let's revisit the type 2 (C2) chromosome and its genes. In the case of the C1 genes, the phenotype was determined by comparing the relative strengths of the genes. The C2 phenotype is determined with reference to external physical constants:

- DAT Second
- DET Nrg unit
- RAT Second
- RET Nrg unit
- EPM Nrg unit
- EPB Nrg unit

If these C2 genes increase or decrease in absolute strength, relative to the reference physical constants, then the phenotype changes. This was not true for the C1 genes.

So, let's repeat the five thought experiments but focus on the C2 genes. The results are in the following table. 'F' means that it is expected that evolutionary forces would select for this mutation, all other things being equal, and 'A' means that it is expected that evolutionary forces would select against this mutation. 'N' means not sure or no specific selection for or against. The table for the C1 genes is also included for contrast.

Gene names are listed across the top. Values of U are down the side.

	F	**FR**	**R**	**BR**	**B**	**BL**	**L**	**FL**
100%	F	F	N	A	A	A	N	F
75%	F	F	N	A	A	A	N	F
50%	N	N	N	N	N	N	N	N
25%	A	A	N	F	F	F	N	A
0%	A	A	N	F	F	F	N	A

	DAT	DET	RAT	RET	EPM	EPB
100%	F	F	A	A	A	F
75%	F	F	A	A	A	F
50%	N	N	N	N	N	N
25%	A	A	F	F	F	A
0%	A	A	F	F	F	A

Again we see that the propensity for the evolutionary forces to select for or against a new gene changes as we change the bias of mutation. What is harder to see is whether it will change the C2 Phenotype. If we use the three derived phenotypic characters Stamina, Fertility, and Agility to categorize the response, it provides some insight.

Stamina is a linear combination of the strengths of the DAT and DET genes. Bugs with high stamina survive longer during hard times before succumbing to natural death (DAT) and are better able to endure shortage of energy in hard times (DET). If U is above 50%, then more mutations of DAT and DET will be selected for, with a resultant increase in stamina.

Fertility is a linear combination of the strengths of the RAT and RET genes. If these genes increase in strength, then the fertility drops. Bugs with a high Fertility give birth at a younger age (RAT) and need less energy reserves to give birth (RET). If U is below 50%, then more mutations of RAT and RET will be selected for, with a resultant increase of Fertility.

Agility is a linear combination of the strengths of the EPM and EPB genes. These genes respond differently, and are selected differently, depending on their strength. A bug with a higher EPM consumes energy more quickly and is therefore less likely to collect enough energy to live and reproduce, all other thinks being equal. A bug with higher EPB is able to gather a greater store of energy before reproduction, and can therefore produce stronger (more energetic) daughters. This is an advantage especially when in transition between a time of plenty and a time of shortage. If U is above 50%, new EPM genes are more likely to be selected against, and new EPB genes are more likely to be selected for. If U is less than 50%, the responses are reversed.

So, in contrast to the impact on the C1 phenotype, which we saw to be nil, biased mutations seem to alter the C2 phenotypic characters which would emerge under the pressures of evolutionary forces.

What Can We Learn From This?

In summary, biased mutations affect the C1 genotype that emerges, but they do not affect the C1 phenotype. This is because the C1 phenotype is self-referential. Biased mutations affect both the C2 genotype and phenotype that emerges. This is because the phenotypic characters are based on absolute external PSoup system values (PSoup second and Nrg unit).

In short, an intrinsic bias in the mechanisms of a mutation may have a profound impact on the nature of the raw material provided to the forces of evolution, the mutated genes, causing significant changes in the selected and emergent phenotypes, or it may have no impact. It is highly dependent on the means by which the genotype is translated into a phenotype.

You are encouraged to repeat the five thought experiments using 'Scenario 5C - Biased Mutations'. Repeat for each of the five values of U, and execute 10 generations of each. In your notebook, note the values of the C1 and C2 genetic profiles of the average bug at the end of each generation (i.e. when the C3/C4 profile shows the generation click over into the next integer). You should have 14 readings for each generation (8 C1 genes plus 6 C2 genes), and 11 such sets of readings (generation 0 up to generation 10).

Advanced PSoup

About Capability Genes

A capability gene represents the ability for an organism to grow a specific type of physical mechanism such as an optic or olfactory nerve. A suitable collection of such mechanisms would form an organ or organelle that has evolutionary value. An organ might have any of the following functions: seeing, hearing, tasting, smelling, sensing by touch, enabling genetic crossover, enabling sexual reproduction, herding or thinking.

Complexity

The third and fourth chromosomes have similar structure but different purpose. The third chromosome, called the type 3 chromosome, or C3, contains genes which may enable a variety of capabilities. The fourth chromosome, called the type 4 chromosome, or C4, contains genes which may enable a defense against other bugs with enabled capabilities. For example, a combination of C3 genes might enable a bug to see other bugs and pursue them as prey. A combination of C4 genes might enable camouflage so that SIGHT-enabled bugs cannot see this one, and cannot pursue it.

The capability genes are stored in the type 3 and type 4 chromosomes (C3 & C4). There are 26 types of capability genes (also called GTypes, in the terminology of PSoup), labeled 'A' through 'Z', and each chromosome contains exactly one of each gene type . Each capability gene is either activated or trashed. A trashed gene can be viewed as analogous to trash DNA in an organic chromosome. These trashed genes are inactive and are not counted as true alleles of the gene. When a trash gene becomes activated, it starts to play a role in the biochemistry of the organism, and it starts to appear in the PSoup tools (allele counts, values, etc.). A trashed gene is assumed to have a strength of 0 when measuring distances (cluster analysis).

The complexity of a type 3/4 (C3/C4) chromosome is the count of active genes. The lowest complexity possible is 0, meaning all 26 genes are trash. The highest complexity possible for a C3/C4 chromosome is 26, meaning one of each GType is active. Any complexity in between 0 and 26 is possible.

A bug in PSoup may be haploid or diploid. For more information, see the article About Haploid and Diploid Forms in PSoup Theory and Practice.

The 'complexity' of the bug is taken by adding the complexity of each of its active C3 and C4 chromosomes. A haploid bug has one C3 chromosome and one C4 chromosome. A diploid bug has two C3 chromosomes and two C4 chromosomes. The complexity of these chromosomes is strictly additive. The maximum complexity possible for a haploid bug is 52. The maximum complexity possible for a diploid bug is 104.

The default complexity is zero (i.e. zero C3 & C4 capability genes are active) and this rises as capability genes are activated in various C3 & C4 genes. The capability genes are activated and added to the C3/C4 chromosomes via mutation of the trash genes under the influence of the PSoup chemical rules.

So, the overall number of capability genes is called the complexity of the bug. Complexity changes via mutations. There are two fundamentally different types of mutation that can affect the capability genes. The complexity mutations affect the number of capability genes. The genic mutations affect the strengths of the components of the capability genes.

The complexity mutation rate determines the ability of an organism to add or delete capability genes. By default the complexity mutation rate is the same as the mutation rate for simple genes, or 12.5%. Therefore, 12.5% of the time, when an organism reproduces, it will either add or delete a capability gene from its C3/C4 chromosomes, determined on a chromosome-by-chromosome basis. For example, if 100 haploid bugs reproduce, roughly 12.5% of the bugs will have an altered complexity of the C3 chromosome,

and 12.5% will have an altered complexity of the C4 chromosome. If a chromosome has 0 capability genes, it must add one. If it has 26 capability genes, it must delete one. In all other circumstances, there is a 50/50 chance that it will either add or delete a capability gene (determined by the rules of chemistry). If it must delete a capability gene, one is randomly selected and deleted. If it must add a gene, it will randomly select one from the list of inactive trash genes, and activate it.

If a bug has zero capability genes in the list (the starting condition for many bugs) then the first gene is constructed with default values. All genes which are added due to a mutation of complexity are constructed with the default values, except that the gene type is selected randomly.

A capability gene is a standard gene having four parts displayed as an ordered quadruple (gene type, dominance type, base, exponent). The gene also has a strength, which is computed using the base and exponent. Strictly speaking, the strength is not a part of the gene, but the computer computes and stores the value once in the same gene structure for the sake of efficiency. For economy of words, we shall write this as (***G***, ***D***, ***B, E, [S]***).

'***G***' is a positive integer representing one of 26 possible mechanisms that an organism can evolve. The number 26 is arbitrary, and happens to be the number of letters of the English alphabet, so each gene type can be represented by a letter of the alphabet. (This lets us say that these bugs can evolve the ability to spell.) In a C3 chromosome, the genes represent 26 aggressive mechanisms. In a C4 chromosome the genes represent 26 defensive mechanisms.

D is the dominance type of the gene.

B is the base of a power. It has the default value of 2 in all C3/C4 capability genes.

E is the exponent of a power. It has the default value of 0 in all C3/C4 capability genes.

S is the strength, computed as ***B*** raised to the power of ***E***. It has the default value of 1 in all C3/C4 capability genes.

When a capability gene undergoes mutation ***G*** and ***D*** do not mutate. ***B*** and ***E*** do mutate under the control of the rules of chemistry. ***S*** is computed after ***B*** or ***E*** mutate.

The acquisition of new gene types is via the addition of a random gene under complexity mutations.

Capabilities

A capability type (CType, in PSoup jargon) is enabled when an organism is able to grow an appropriate organ (or develop an appropriate defensive trait). An organ grows when two or more of the capability genes which spell the associated word happen to exist in the organism's capability gene list. The gene types which spell the associated word are called the activating gene types for that capability.

Capability	**Associated word**
capability to see	SIGHT
capability to hear	HEAR
capability to smell	SMELL
capability to taste	TASTE
capability to sense by touch	TOUCH
capability to exchange genes	XOVER
capability to reproduce sexually	OVULE
capability to herd	HERD
capability to think	THINK

Potency

The number of times that any letter appears in the above list indicates the potential potency of that type of gene. For example, if a bug has all nine capabilities activated, it may nevertheless have only a single ‘T’ gene in its capability list. This gene has enabled four capabilities, and its strengths will be counted into the bugs various strengths four times. On the other hand, the bug similarly may have only a single ‘C’ gene, which enables a single capability, and is counted only once.

Here are the potencies of each gene type (GType):

GType	Potency
A	2
B	0
C	1
D	1
E	6
F	0
G	1
H	5
I	2
J	0
K	1
L	3
M	1
N	1
O	3
P	0
Q	0
R	3
S	3
T	5
U	2
V	2
W	0
X	1
Y	0
Z	0

Note that ‘SMELL' may be activated by a single ‘L’ gene that 'L' gene has double potency in SMELL. Similarly, TASTE is activated by a single ‘T’ gene, and that gene has double potency in TASTE.

Inert genes

A number of genes have zero potency. They contribute towards the overall complexity of the bug, but have no potency in aggressive or defensive actions. These genes are said to be inert. These gene types are used in by the Tinker Bug Wizard to add complexity when no additional potency is wanted. There is evolutionary pressure to add complexity, and therefore accumulate as many capability genes as possible. However, there is no evolutionary pressure of any type on the strengths of these insert genes. The inert gene types are B, F, J, P, Q, W, Y, and Z.

Activation of Capabilities

A bug’s ability to effectively use its organs is determined by the strength of the genes. For example, the ability to see is determined by summing the strengths over all of the activating gene types in the C3 chromosome. If a gene is inactive, it has a strength of zero. These sums do not change over the life of the organism, and are fixed at time of fission (or at birth). So, for example, if an organism has an S and an I gene in the C3 chromosome, SIGHT capability has strength equal to the sum of the strengths of the two activating genes. If a different bug has an H and a T gene in the C4 chromosome, SIGHT defenses are activated equal to the sum of the strengths of the activating genes. If the SIGHT defenses are greater than

the SIGHT capability, then the second bug is invisible to the first. When an organism starts life, and all mutations have been computed and put in place, it is then possible to evaluate each physical capability and determine cumulative strengths for that organism for each potential capability and defense, and those strengths will be with that bug for its life.

Evolution of and competition between C3/C4 chromosomes

The approach described above sets the stage for competition for survival of the fittest BETWEEN C3 AND C4 CHROMOSOMES within the genome. When bugs reproduce sexually, chromosomes can migrate throughout a population of bugs, generation after generation. At the same time, chromosomes can change their content and configuration slowly through the accumulation of mutations and stolen alleles. A bug cannot have an unlimited number of chromosomes. Those chromosomes which offer the most evolutionary advantage to the bugs will multiply. Those that offer inferior capabilities to the host bug will disappear from the gene pool.

In order for a capability to be of evolutionary value, it must affect a life function. Our bugs have only five life functions, being:
- Sensing
- Moving
- Eating
- Reproducing
- Dying

To add some similitude to real biological capabilities, by default we interpret the fight or flight strength as a physical attribute, and apply it to one of the five life functions. The following table outlines the interpretations given and the function affected:

Capability	Fight Interpretation	Flight Interpretation	Function Affected
SIGHT	Good Eyes	Camouflage	Sensing, Moving
HEAR	Good Ears	Stealth	Sensing, Moving
SMELL	Good Nose	Foul odor	Sensing, Moving, Repro
TASTE	Eat foul food	Foul taste	Sensing, Moving, Feeding
TOUCH	Sharp Claws	Hard Shell	Sensing, Moving, Repro
XOVER	Initiate Crossover	Resist Crossover	Repro
OVULE	Initiate mating	Resist mating	Repro
HERD	Herding instinct	Loner instinct	Moving
THINK	Planning	No function	Moving

In addition, certain capabilities are associated with certain predilections towards food types. By default, all bugs are holotrophs. That is to say, they all eat algae. However, the capability genes can modify that.

Bugs which can hear or touch, i.e. bugs for which the C3 chromosome has enabled HEAR and/or TOUCH are able to eat other bugs, and are no longer able to eat algae. These bugs are carnivorous. Bugs which can smell or taste are able to eat only algae, and are said to be herbivorous. In fact, their food source is the same as the holotrophic bugs. Bugs which can see can eat either algae or bugs, and are said to be omnivorous. Any bug which is both herbivorous and carnivorous, is omnivorous. All holotrophic bugs reproduce by fission and cannot mate. Bugs which are carnivorous, herbivorous or omnivorous and which are also sexually reproduced MUST mate with their own kind, within the allowed range of complexity. So, the C3 genes can create mating barriers across which genes cannot flow. It is therefore possible to evolve distinct species, in every sense of the word.

More on Complexity

COMPLEXITY is used to define the prey/neutral/predator relationships. COMPLEXITY is a positive (or zero-valued) integer equal to the number of capability genes. COMPLEXITY is therefore also a tag which

identifies those bugs which are potentially able to mate with each other. Only bugs of similar COMPLEXITY may mate. Upon initiation of a standard PSoup run, all bugs have no (zero) capability genes, and therefore a COMPLEXITY of zero. In the first generation of bugs, 12.5% of bugs (based on the mutation rate) will have exactly one capability gene. These bugs are therefore 'potential mates'. However, only those with the activated OVULE capability can actually transfer genes in a sexual encounter. When the OVULE capability is first activated, the bug immediately becomes diploid, and all chromosomes are doubled. This means that bugs must have a COMPLEXITY of at least 4 (2 activating genes in two C3 chromosomes in a diploid bug) before mating can be activated. Since COMPLEXITY determines the predator/prey relationship, there should be tremendous pressure on this characteristic to grow larger, up until parasitic behavior develops.

There are actually three types of paired relationships governed by Complexity:
- Parasite/host relationship - this relationship exists if one bug is more than four times the size of the other bug.
- Predator/prey relationship - this relationship exists if one bug is more than twice the size of the other bug.
- same herd - this relationship exists between any two bugs which are not in the other two relationships. Bugs in the same herd are generally roughly equal in complexity. Specifically, neither is less than half the size of the other.

Bugs which are of the same herd may view each other as being potential mates, potential members of a common herd, or bugs of no interest.

About Haploid and Diploid Forms

This is a brief technical article describing the meaning of, and implementation of, haploid and diploid forms in PSoup.

Haploid bugs have four or less chromosomes, depending on the achieved Level current at the time that a scenario is run. Diploid have eight chromosomes, arranged as four pairs, one of each type. The following table outlines the possibilities:

Level	Haploid	Diploid
1	1 C1	N/A
2	1 C1, 1 C2	N/A
3	1 C1, 1 C2	N/A
4	1 C1, 1 C2, 1 C3, 1 C4	N/A
5	1 C1, 1 C2, 1 C3, 1 C4	2 C1, 2 C2, 2 C3, 2 C4
6	1 C1, 1 C2, 1 C3, 1 C4	2 C1, 2 C2, 2 C3, 2 C4

Do not confuse the achieved Level with the nominal level of a scenario. Achieved level is increased by passing the Level Advancement Tests (LATs) or by using the control panel to change the level. If you run Scenario 1A with an achieved level of 1, you will have haploid bugs with one chromosome, and that's all you can have. The nominal level of Scenario 1A is 1, as the name suggests. However, if you raise the achieved level to 2 and re-run Scenario 1A, then the bugs will be haploid with two chromosomes each. If you raise the achieved level to 6 and re-run the same scenario, even though the nominal level is still 1, the bugs will start as haploid bugs with four chromosomes, but will soon evolve to become diploid bugs with eight chromosomes.

Only OVULE-enabled bugs are diploid. All diploid bugs are OVULE-enabled. If a bug suffers a mutation such that the OVULE capability is canceled, then the bug becomes haploid and half of the chromosomes (the paternal half) are discarded. If a haploid bug suffers a mutation such that the OVULE capability is enabled, then the maternal chromosomes are copied to double the total number of chromosomes. The diploid bug now has two identical sets of chromosomes.

Complexity - Haploid bugs can attain a maximum complexity of 52. Diploid bugs can attain a maximum complexity of 104.

Haploid bugs can have at most 52 active capability genes in the C3/C4 chromosomes. Diploid bugs, having two C3 chromosomes and two C4 chromosomes, have twice this number of genes.

Cross-Over - Cross over involves two sets of four chromosomes. I.e. in a cross-over event, eight chromosomes are involved.

When a haploid bug undergoes cross-over, the four chromosomes of the XOver-enabled bug are used as one set, and the copy of the four chromosomes stolen from the donor bug are used as the other set. The order of events is:
(a) theft of genetic material (Note that the donor bug is not damaged or changed);
(b) cross-over is attempted, chromosome by chromosome, for all four types, to produce a gamete with four chromosomes;
(c) discard stolen chromosomes;
(d) fission of gamete to produce two daughters;
(e) disappearance of mother.

If no genetic material could be stolen, steps (a) through (d) are skipped and simple fission is used in their place.

When a diploid bug undergoes cross-over, the four maternal chromosomes are crossed with the four paternal chromosomes. The spermatic chromosomes do not participate. The spermatic gamete is a selection of one of each type of chromosome from each of the four pairs of maternal and paternal chromosomes in the mating partner (the donor bug). The order of events is:
(a) mating, in which the bug obtains a spermatic gamete with four chromosomes from the donor; the bug now has its own 8 chromosomes in 4 pairs, plus 4 spermatic chromosomes in a gamete;
(b) if XOver is enabled, the maternal and paternal chromosomes undergo attempted cross-over; if XOver is not enabled, then this step is skipped;
(c) a gamete (ovum) is formed by selecting one chromosome of each pair from the maternal and paternal chromosomes; by default, the non-selected chromosomes also form a gamete, which is discarded; the gamete effectively is a haploid gamete;
(d) conjunction of the haploid ovum and the haploid sperm to produce a diploid zygote with eight chromosomes;
(e) the zygote is released as the daughter; if this is the first daughter, the bug lives to reproduce again; if this is the second daughter, the mother gives everything to the daughter and dies.

Note that, if XOver is enabled, there is always a possibility that chance will prevent actual cross-over. If the 'Probability of XOver' parameter is set low, most attempts at cross-over will fail to produce any change.

PSoup and Alleles

This is a technical article on the details of implementation of genetic alleles in PSoup. It is intended that this article be read in conjunction with the use of the 'Display Allele Counts' and 'Display Allele Values' features (under the Options menu) or with Scenarios 6F and 6G - Population Size and Allele Loss (One and Two).

Before jumping into the technical details, there must be a clear understanding of the differences between real genes and PSoup genes.

In the natural world, a gene is a set of DNA code that determines an identifiable phenotypic character. A gene may consist of a single contiguous string of DNA, or several non-contiguous strands of DNA. A gene may have extraneous material inserted into its midst.

In PSoup a gene is a pair of numbers S and D.

S is the strength of the gene. It is in the range from 50^-150 to 50^+150 (that's 50 raised to the power of -150 and +150 respectively). The strength S is computed using two control numbers B and E. B is used as the base of a power function. E is used as the exponent of a power function. The value of the power function S = B^E is called the strength of the gene and is denoted by S as described above. B ranges from 1 to 50. E ranges from -150 to +150. There is no intended analogy between strength of a gene in PSoup and any known quantitative measure of real genes in the natural world.

D is the dominance type of the gene. D is in the range from 0 to 999. If two corresponding genes are both active in a diploid bug, then the gene with higher dominance type is used and the other gene is ignored. If two corresponding genes in a diploid bug have different strengths and equal dominance types, then PSoup randomly increments the dominance type of one of the genes, making it dominant and the other recessive.

The only intended and clear analogy between PSoup genes and real genes in the natural world is the control of phenotypic characters by the genes. All other details about the implementation of genes in PSoup are incidental to the computer system.

In the real world, a gene which controls a particular phenotypic character (say, the color of eyes of cats) will exist in several different varieties or values or forms. For example, if we look at the DNA of two cats, looking at the sets of DNA that constitute a gene for eye color, the sets of DNA in similar locations in similar chromosomes constitute two copies of the same gene for eye color. These two copies of the gene may be identical, in which case both cats should exhibit the same phenotypic eye color character, or they may be similar but not the same in every detail, in which case the cats may exhibit different colors of eyes. If the two set of DNA are identical, we say they are the same allele. If the two sets of DNA are different, we say that they are two different alleles of the same gene. In general, different alleles of a gene exhibit as different variations of a character in a phenotype.

In PSoup, every gene has a type which is determined by the chromosome in which it resides and the location of the gene within the chromosome. Type 1 chromosomes (C1 chromosomes) contain nine Palmiter genes (also called transport genes). All C1 genes are simple genes being identical in structure but differing in function, as they control different characters in the phenotype. Type 2 chromosomes (C2 chromosomes) contain six regulatory genes that control the life functions of a bug. All C2 genes are simple genes, and again they have differing functions depending on location in the chromosome. Type 3 chromosomes (C3 chromosomes) contain 26 genes. Type 3 chromosomes enable aggressive capabilities. Type 4 chromosomes (C4 chromosomes) contain 26 genes, which enable defensive capabilities.

In PSoup, two genes are considered to be the same gene, the same type, if they occupy the same position in the same type of chromosome. This is vaguely analogous to real genes in the natural world. If two copies

of the same gene have different strengths, we say they are different alleles. If the two copies of the two genes have the same strength, we say they are the same allele of the gene.

In PSoup, bugs are either haploid (meaning all chromosomes are unique) or diploid (meaning all chromosomes exist as pairs of similar types of chromosomes). This is also true of all organisms in the natural world which have nuclei in their cells. Some single-celled organisms do not have nuclei and are not referred to as haploid or diploid. In PSoup, a haploid bug may have one each of a C1, C2, C3 and C4 chromosome, which may all be doubled if the bug is diploid. Only bugs which are 'OVULE-enabled' can be diploid.

In PSoup, a haploid bug may have as many as 67 genes, one allele each, or 67 alleles. A diploid bug may have 134 genes, one allele each, or 134 alleles.

In the natural world, two alleles of the same gene usually establish a dominant/recessive order. If one allele dominates the other in animal A, then the same allele is dominant in animal B. In PSoup, this is not the same. If two alleles of a gene exist in a diploid bug in PSoup, then one may be intrinsically dominant by reason of past history. If however, they are equal in dominance, PSoup arbitrarily chooses one to be dominant and increases its dominance factor. In all future meetings involving this gene with increased dominance, it will function at the incrementally higher dominance. However, due to the arbitrary method of selection for increased dominance, PSoup may arbitrate between the same two alleles in two different bugs and assign the dominant role to opposite alleles in different bugs. It may be that allele X dominates allele Y in bug A, but allele Y dominates allele X in bug B.

Careful setup of a scenario in PSoup allows us to explore allele-related dominant/recessive issues (e.g. Mendel's genetics) but in general, alleles may not exhibit dominant/recessive order consistently in general PSoup scenarios.

PSoup offers the ability to randomize the genes of a bug. This process randomly assigns a new strength to each active gene in the bug. The base value of the gene is left untouched. The exponent is given a new value. If O is the old value, and N is the new value, then N is within the range $0.875O <= N <= 1.125O$. 100 evenly spaced values are identified within this range, and the strength is assigned one of these 100 selected values. In this way, if there is a population of 100 bugs with identical genes, and you randomize the genes of each bug, the new population of bugs will exhibit up to 100 different alleles for each gene. The phenotype of all resultant bugs will be similar to, but not the same as, the phenotype of the original bugs. This randomization process is used to generate variety in the populations of bugs in Scenarios 6F and 6G.

In the natural world, in most diploid organisms, for each gene there are a relatively small number of dominant alleles which control the typical phenotype of the species, and a large number of recessive alleles which are passed on from generation to generation. This pool of unexpressed genes (the gene pool) is the raw material, the library of collected useful genetic mutations, that gives a species the collective ability to respond adaptively to changes in circumstance. An adaptive response is often needed within a few generations or the species faces extinction. Mutations cannot provide the new genetic offerings quickly enough to meet this demand. Natural variation of the alleles in the latent gene pool can and does meet this demand.

Competition for survival exists at several levels in PSoup:
- **Bugs** - The bugs with the best array of phenotypic characters will out-compete their contemporaries and reproduce more quickly and/or efficiently.
- **Chromosomes** - The chromosomes which carry the best array of phenotypic characters will enable the bugs which carry them to out compete their contemporaries. Chromosomes in haploid bugs are committed to the bug and to each other. There is no process to de-link the chromosomes. They survive together, and evolve together, or go extinct together. Chromosomes in diploid bugs can (a) contribute to their own survival through their array of dominant genes or (b) contribute to their future role in adaptive response through their recessive genes. Chromosomes in a diploid population compete directly and fiercely for

survival. Their dominant genes must be in a beneficial configuration, and their non-beneficial genes must be recessive, or natural selection will weed their owners out of the population.

- **Gene Types** - Unlike the C1 and C2 chromosomes, the C3 and C4 chromosomes are not fully populated by default. The gene types in a C3 or C4 chromosome interact with each other in ways that may be beneficial or harmful. In a sense, there is competition between gene types, as not all gene types may be desirable, and some combination of gene types may be selected against. For gene types to persist, there must be some benefit, and they must therefore find a favorable configuration inside a chromosome, or face extinction.
- **Alleles within a Gene Type** - The alleles which cause exhibition of beneficial characters contribute to the survival of the chromosomes and bugs which carry them, and therefore ensure their own survival. If XOver is enabled, alleles of a gene may jump from chromosome to chromosome, looking for a better configuration. In a haploid population, alleles can jump from a chromosome in one bug to a chromosome of the same type, and gene of the same location, in another bug. In a diploid population, alleles can jump from the maternal chromosome to the paternal chromosome and vice versa. Over time, combinations of alleles which contribute the most to the survival of strong chromosomes, and strong bugs, will accrete into powerful chromosomes. For alleles to compete effectively, they must be in a diploid population (OVULE-enabled) and able to move from chromosome to chromosome (XOver-enabled).

Several scenarios within PSoup are designed to allow a student to explore the behavior of alleles and the competition between chromosomes, gene types and alleles:

- Scenario 5D - Evolution Without Mutation
- Scenario 6E - Migration of Genes
- Scenario 6F - Population Size and Gene Loss (One)
- Scenario 6G - Population Size and Gene Loss (Two)

Several tools are provided in the PSoup Options menu and Tinker menu to enable a student to examine the genes in a population of bugs:

- 'Display Genes' button of the 'Tinker with the Bugs' command/dialog;
- C1, C2 and C3/C4 genetic profiles of each current bug;
- C1, C2 and C3/C4 genetic profiles of the average bug in a population;
- Historic charts which track allele counts and values over time;
- Cluster Analysis, with genetic profiles by cluster; and
- Population distribution charts, which plot the distribution of alleles within the population.

The Role of Chance

In PSoup, there is no such thing as luck. There is no such thing as free will. Everything that happens in any and every scenario is the result of the clockwork interplay of the genetics of the bugs, chance, and circumstance.

Genetics of the bugs

A bug's genes are fixed unalterably at birth and do not change at all throughout its life (unless the TinkerBug Wizard puts his rod into the soup).

Circumstance

The circumstances in which a bug is situated comprise only those things which can affect its behavior or development. The following is intended to be a complete list, but it may leave out some subtle influences:

- edge effects such as the edge of the bowl of soup (out of bounds for bugs), maximum value for the base number in each simple gene (= 50), maximum and minimum values for the exponent numbers in each simple gene (= +150 and -150 respectively), maximum number of bugs supported in PSoup (=500), the maximum number of tree nodes allowed in the family tree (= 1500), maximum number of days in PSoup (= 30,000; an untested boundary), and possibly other implementation-specific edges.
- permanent terrain effects such as an oasis, cascading paddies and the probability of paddy jump; independent paddies; muddy vs. non-muddy patches (not implemented in this version).
- temporary terrain effects such as the current location of algae.
- indirect factors such as the nature of the energy system (open or closed).
- the current location of other bugs (blocking movement, offering donations of genetic material, posing threats as parasites or predators, offering a source of energy such as hosts and prey).
- the rules of chemistry and physics that are implicitly embedded into the PSoup environment (e.g. bugs move one cell at a time, they cannot fly, cells are square vice hexagonal, the processes governing cross-over of genes). Included in these rules are the thresholds that are used by the pseudo-random number generator to choose between alternate process options.

The Role of Chance

This is an important concept, key to the understanding of PSoup and what it teaches us.

The outcome of each scenario is pre-determined when the scenario is set up, except for the intervention of a pseudo-random number generator, which is invoked in four instances, as follows:

some physical laws are probabilistic - in PSoup only one is. The probability of a jump from a left-hand cascading paddy to the right-hand cascading paddy is pre-determined, but the actual jump is determined by the pseudo-random generator.

- some chemical laws are probabilistic - in PSoup most of these are controlled by the parameters, the thresholds, with which a user may tinker in the 'Tinker with the environment' panel. When certain chemical processes are invoked, the pseudo-random number generator is used to select between options.
- some interactions between bugs are moderated by the pseudo-random number generator, based on probability thresholds derived from comparisons of their genes. To be explicit, if one bug, in the clockwork routines of PSoup, happens to be found in the circumstance that it is moving into the square already occupied by another bug, it is considered to be attacking the other bug (feeding, stealing genes, mating). A probability of success threshold is determined by reference to the genes of the two bugs, and the pseudo-random number generator is used to decide the outcome.
- when a bug 'decides' how much it will turn before it steps forward, or when it 'decides' to stand still, the probability of each action is determined by the strength of the associated Palmiter gene, but the decision is made by the pseudo-random generator.

The computer does not favor one potential action of a bug over another, except when the genes of the bug, in interaction with the genes of other bugs and the pseudo-random number generator, determine that it must be so.

Nevertheless, pristine bugs, with no intelligence whatever, can evolve to become efficient gatherers, effective fighters, capable of social interactions (herding) and semi-intelligent behavior (choosing a better escape route).

Contingency

Contingency is a key concept in evolutionary theory and must be understood.

Let's do a couple of mind experiments with a Level 1 bowl of PSoup and a first year university physics lab.

In PSoup, each scenario is set up with the help of the pseudo-random number generator. It determines the placement of algae and bugs at startup, and possibly some details about the genetic makeup of the bugs, if required for the scenario. The probability that two scenarios will start the same is vanishingly small. However, to simplify our discussion, we can arrange to have a startup scenario replicated precisely through the simple expedient of saving a copy before use. Suppose you open a Level 1 scenario in PSoup with twenty bugs at start and save it before any PSoup time passes. It's clock is set at d0h0m0s0. You can now reload that scenario as many times as you wish, and replay the exact identical startup scenario as many times as you wish.

This kind of precise repeatability of starting conditions is a luxury we can enjoy in PSoup which is not available in ANY science in the real world, and certainly not in those sciences which deal with historical events. The theory of evolution falls into this category. In a physics experiment we can set up an experiment to precisely duplicate the starting conditions of a similar experiment, but there is always some scale of interpretation at which the two sets of starting conditions differ. It is often possible, in physics and chemistry, to choose a scale of interest such that differences in starting conditions are of negligible importance, and then produce useful repeatable experiments. Our knowledge of the laws of motion and chemistry came from such experiments. However, when we change the scale to cover the very large (celestial mechanics affecting stars and galaxies) or the very small (subatomic interactions) or the very prolonged (evolution of life) or the very brief (discrete particles of time) we do not have the ability to set up or make meaningfully repeatable experiments.

Returning to our reloaded bowl of PSoup. Suppose we reload it 100 times and run it out to 3 PSoup days each time. The number of possible trajectories of any such run is so incredibly immense, so unimaginably vast, that words or numbers cannot describe it. We literally do not have the mathematical notation available to practically write such large numbers. They would be numbers with trillions of digits before the decimal raised to the power of numbers with trillions of digits. Similarly, the probability that any two runs of this scenario would produce precisely the same outcome would be so small as to be equal to zero by all practical or imaginable measures. However, having completed the exercise we will have 100 trajectories all starting from the same condition and all ending at time 3d0h0m0s.

At the same time, a first year physics student is doing some carefully controlled experiments on the conversion of potential energy to heat energy. He/she carefully drops a metal bar 100 times through a nearly perfect vacuum onto a metal plate, recording the change in temperature of the bar and the plate each time. In each drop, the bar is carefully placed in its starting position, the temperature is measured, and the density of the vacuum is checked. Starting conditions are always precisely the same, within the tolerances of the equipment available to the experimenter. If measured with sufficient accuracy, the result will be 100 sets of results which are closely grouped. Using an averaging technique, the results will be combined and compared with theoretical predictions, and any variations from the predicted outcome will be sought and explained. The action of the macroscopic laws of thermodynamics will have been evident in the experiment.

Even though the exact outcome of any of our 100 runs of the PSoup scenario is unpredictable, there will be a grouping of certain characteristics of the results. The final number of bugs will be similar. The genotype of all bugs across all independent runs of the scenario will be tightly grouped. Variances from the 'norm' can be examined and possibly explained. Even though the PSoup scenarios are driven by fundamentally random and unbiased decisions arbitrated by the pseudo-random number generator, and no trajectory can possibly be repeated in two runs, nevertheless, the macro characteristics of the outcome is predictable. The action of the law of averages (mathematics) and the laws of natural selection (evolution) are evident in the experiment.

Let's focus on a single bug which is alive and living in the bowl of PSoup at the end of one of the runs. The history of the bowl of PSoup follows a trajectory through PSoup time, and that history included the history of each and every ancestor of this bug, and the history of each and every other bug that has inhabited the bowl since second 0. The current location of this bug is contingent upon the chance collision of this bug with other unseen bugs. The current strength of this bug is contingent upon the chance discovery of algae colonies that were not discovered first by other bugs. The age of this bug is contingent upon the birth date and success of its parent. The genetic makeup of this bug is contingent upon the chance mutations and successful feeding and reproduction of each of its ancestors. In some small way, the current status of this bug is contingent on EVERY action arbitrated by the pseudo-random number generator since the start of the run, EVERY placement of algae, EVERY step made by EVERY bug, EVERY mutation of EVERY gene, whether successful or not, as these determined the environment in which the bug and the it's ancestors competed for success.

In PSoup the history of a bowl of PSoup can be reduced to a list of discrete identifiable events all arbitrated by the pseudo-random number generator. The precise history of a bowl can be relived by recreating these events. In fact, that is roughly how the 'Record' function of PSoup works. In the real world, there are contingencies at all levels, at all scales, and not only can we not recreate the exact starting conditions, we cannot list and re-enact the events upon which the outcome is contingent. Nevertheless, many events in the natural world are highly predictable, just as they are in PSoup. It is this predictability that scientists formulate as natural laws.

Suppose our single bug, being three days old in a Level 1 bowl of PSoup, has the evolved phenotype of a right-handed cruiser. That is, it's F Palmiter gene (gene0 in C1) has 20% of the total C1 gene strength and its FR Palmiter gene (gene1 in C1) has 75% of the total C1 gene strength, leaving only 5% for the other 6 genes. What is the probability that one of its progeny in the far distant future will be a pristine bug, with all C1 genes having an equal 12.5% of the C1 strength. Such an outcome would be contingent upon a very large number of very improbable events. So, while there are a very large number of historic trajectories which can be projected into the future to arrive at this outcome, the probability of its realization is vanishingly small, virtually nil. On the other hand, if we had looked at a bug in the second generation of the bowl, the probability that one of its progeny would be pristine would have been very high.

All that the bug is, on day 3, is due to the weight of myriad small events piled on top of each other in the history of the bowl. The future status of each of its progeny is equally contingent on those same historic events, as well as more which have not yet happened.

So, What is Contingency in PSoup?

- In any non-determinate system (probabilistic) such as PSoup the establishment of the trajectory of the system through its space of possible trajectories is determined one second at a time.
- There is a probability associated with the transition of the system from its current state to any possible next state, the sum of these probabilities is one, and only one transition is possible.
- Transitions are unidirectional, i.e. when one of the possible next states is realized, the system cannot return to the previous state.
- At each step, the realization of one possibility, the move into one possible next state, excludes or seriously reduces the probability that other possible future states can be realized as part of the trajectory.

- The current status of the system, the bowl of PSoup, is determined by the historic trajectory. That is, it is determined by the weight of a very many small outcomes of many probabilistic processes, however probable or improbable each outcome may have been, none of which can be revisited or undone.
- And, **most importantly**, the future trajectory of the system is significantly limited by the history of the trajectory to date, and the laws of PSoup which govern the outcome of all future small probabilistic events.

In short, the present is contingent on the past, no matter how improbable or undesirable that past might be, and the future is contingent on the present, and the natural laws that seem to govern change. Most theoretically possible future states may never happen, however desirable they may be. All currently existing presentations of real possibilities (e.g. conscious man) are merely the evidence of one possible historic trajectory which can never be repeated.

Why is Contingency Important?

In the examples given so far, contingency seems to be important only at the microscopic level, and does not seem to appreciably affect the longer-term trajectory. The last sentence of the prior paragraph hints at a higher importance.

An understanding of contingency is necessary if we are to properly understand the historic trajectory of any developing system. If we view life on earth as a developing system, we must understand the role of contingency.

Through paleontology and archeology we can discover the actual trajectory of the organic system we call life, and the social system we call civilization. But how do we extract from this knowledge information about the laws of nature? Or what can it tell us about our role in nature? Did the historic trajectory follow the most probable course? Was the arrival of man on earth inevitable, or an accidental and improbable outcome caused by the elimination of the dinosaurs? Can you overcome the effects of contingency and plot the future trajectory of a probabilistic system?

A Rock in a Stream

There are two extreme views about the long-term impact of a probabilistic event. One view can be called the 'rock in the stream' idea. There is a turbulent stream running down the mountainside. In the middle of one more level section is a rock which protrudes above the surface of the water. The water flows around the rock, which causes some white water in front, a bit of a hydraulic behind, and some wavelets for a distance downstream. The rock is the event in time, which is the water flowing past it. The event caused an immediate and large change at that point in time when it occurs, but as time passes, things return to normal until the impact of the event is invisible. To be sure, the molecules of water have been rearranged due to the presence of the rock, and the local impact is clear, but the long-term impact is minimal.

This view can be seen in the Level 1 bowl of PSoup. We start with 20 pristine bugs. We know from experience that only one will succeed, and that eventually the progeny of the other 19 will die out, the lines becoming extinct. In three days the dominant phenotype will be a cruiser. We can place a large rock in the stream by adding, say, jitterbugs or Twirlies to the mix of bugs at an early stage. The ultimate outcome, however, on day 3, will not change. We will have the same number of cruisers in the bowl either way. We see this effect in the natural world as well, in the phenomenon of convergent evolution. The ecological niche on the edge of the world's continental shelves has been occupied by plesiosaurs, reptiles and mammals (porpoises), all having the same phenotypic characteristics, but all having radically different origins. The rock in the stream is the removal of the dominant species in the niche. Let enough time go by and the niche is filled by another species. 'The molecules have been rearranged but the long-term impact is minimal'.

The Butterfly Effect

The other view is called the 'Butterfly Effect'. In this view, the beat of the wing of a distant butterfly eventually affects the future of all weather, and through it, all life on earth. The location of every molecule

of air is contingent on the prior location and status of all of the other molecules with which it has come into contact. However gently the butterfly might move its wing, the molecules around it are disturbed, and those next to them, and those next to them. Unnoticed, the disturbance travels outward with ever increasing breadth but ever decreasing strength, moving with the speed of sound (the speed at which molecules jostle each other) until every molecule within the sphere of the earth has reverberated ever so slightly in response. Just one of those ever so small responses can alter the course of a probabilistic event which was contingent upon it, resulting in a significantly different trajectory for the history of the earth.

In PSoup we can see the impact of this type of effect. In our Level 1 bowl of PSoup, what kind of cruiser will be dominant on day three, right cruisers or left cruisers? It will be one or the other, but not both. Suppose you started with a population of 10 right cruisers and 10 left cruisers, who would dominate after 3 days? Both types have an equal probability of success. In fact, it is hard to think of a reason why both cannot succeed compatibly together. They do not. At some point early in the history of the bowl a bug undergoes mutation which gives it slight tendency to turn left or right. The effect, at first is as small as the beat of a butterfly's wing. The advantage is almost nil. However, there is a new balance of probabilities, the existence of which is/was contingent on the appearance of that advantageous mutation at that precise moment in time. And over time, as the bugs compete for resources, and as natural selection removes the losers, the handedness of the phenotype becomes associated with success and is preserved.

Any minor event which triggers, or helps to trigger, a growing cascade of events of great import might be viewed as an example of the butterfly effect.

Back to Physics and Butterflies

We see a manifestation of the butterfly effect in physics, although it is not generally viewed that way. This example is rather contrived, but bear with it. Suppose our first year student modifies his thermodynamics experiment somewhat. Suppose he adds a steel three-sided prism to the bottom plate and drops the metal bar so that it strikes the upward edge of the prism. Rather than thumping onto the steel plate as before, it now strikes the prism, bounces to one side or the other and rolls until it hits the edge of the bell jar in which the experiment is being executed. No matter how carefully, how accurately, the student lines up the bar before dropping it, it will fall and bounce to one side or the other. We can imagine the bar falling, striking the prism, and coming to rest on top of it like a bird on a perch, but we know from practical experience that this is not remotely likely. Suppose we imagine a little genie who can ensure that the bar is perfectly aligned, and falls perfectly, and comes to rest exactly balanced on the prism. No matter how perfectly aligned the rod and prism are, at some microscopic level the bar will respond or reverberate to air turbulence (such as remains in the bell jar), or heat turbulence (in the prism) or gravitational or electrical fields, such that, momentarily, the center of mass will be to one side or the other, and eventually, the bar will fall from the prism. As the bar gradually gains speed and falls from its perch, we see the macroscopic effect of the microscopic events which cumulatively toppled it.

Do we have organic examples of the butterfly effect in the phenomenon of ecological escape (e.g. African killer bees, fire ants, zebra mussels, English sparrows, earwigs, man, etc.).

To Be or Not To Be?

How do we view man as a product of evolution?

According to the Rock in the Stream theory, we might be the inevitable pinnacle of an ever progressive development.

According to the Butterfly Effect theory, we are the accidental and temporary product of a turbulent probabilistic trajectory which faced many forks with equally probable branches, the selection of which was influenced beyond reasonable expectation by small (contemporary and prior) mundane chance events.

Reality probably lies somewhere in the middle. PSoup can be used to mentally explore some of the implications of these profoundly interesting ideas.

About Speciation

The question naturally arises, are we able to evolve separate species of bugs in PSoup, and under what conditions would multiple arise?

What is a Species?

It has been generally recognized since Darwin wrote his book 'On the Origin of Species' in the mid 1800's that species are not immutable, and that, in fact, for many plants and animals, it is very difficult to clearly define a species. Whatever the definition of a species, it is almost always to find organisms which test the definition.

In the natural sphere, a species is generally defined as (A) the smallest group of organisms among which gene flow is possible. This definition is applied only with difficulty to organisms which reproduce non-sexually, and for which gene flow between organisms is not meaningful or not possible. Many of the bugs in PSoup, and certainly all of the bugs found in levels lower than level 5, fall into this category. For such organisms, a less satisfactory definition must be used. For such organisms, they are considered the same species if (B) they have a common phenotype; i.e. common structure and behavior as life proceeds.

We use definitions similar to definitions A and B, above, in PSoup.

Before we explore the implications for PSoup, let's take some time to understand what it means in the natural sphere.

In its narrowest sense, definition 'A' means that two organisms are of the same species if they are geographically able to meet, physiologically able to mate, are inclined to participate in the act, and can produce viable fertile offspring.

In its more general sense, two organisms are of the same species if there is an unbroken chain of possible matings between them. Suppose snake A is the same species as snake B. Let's write that as A=B. Suppose that snake B = snake C. And then suppose that snake C = snake D. Then there is (potential) gene flow between them, and they are of the same species. It doesn't matter that snake A lives in Canada and snake D lives in Mexico. If there is a potential gene flow, over time, from A to the descendants of D, then they are of the same species. You can imagine it this way. A mates with B producing Ab. Ab mates with the daughter of C to produce Abc. Abc mates with the grand-daughter of D to produce Abcd. Suppose, at the same time, Dcba, the grand-daughter of D, mates with the grand-son of A. Genes have flowed from A to D and/or from D to A, so they are in the same species. Kinda messy, huh!

With these definitions in hand, let's look at PSoup, focusing, at first, on those bugs which do not have XOver or Ovule enabled.

Every Fissioning Bug defines its own Species!?

For those bugs which have not developed the ability to recombine genetic material either through cross-over or through mating, their genes flow nowhere but to their daughters. Under definition A, strictly speaking, each such bug is a species in its own right. Biologist have the same difficulty in identifying species of bacteria that reproduce chiefly via fission. In this case, bugs which look the same and act the same are considered to be the same species, and bugs which look different and act different are considered to be different species. This takes us to definition 'B'. In PSoup we use a 'standard mutation' as a measure of distance between bugs. Bugs which cluster close to each other are then considered of the same species, and clusters which are widely separated are considered different species. This sounds good, but cluster analysis is very much an art, and not a science. The bottom line, if the bugs are haploid and XOver is not enabled, genes do not flow and definition B is used. Cluster analysis is an experimental tool built into PSoup to allow determination of species based on phenotype and genotype differences.

What is a Sub-species?

In biology there is also a concept of sub-species. Two organisms are considered to be in different sub-species if they are clearly in the same species, using the above definition, but due to geographic separation or for other reasons, the gene flow is temporarily cut off, or the phenotypes are distinctly different. This is a very messy concept, and there are many species and sub-species designations which are in dispute by learned and knowledgeable men.

In PSoup we use a concept similar to this. Those bugs which are of the same species can categorize each other by size. Complexity is used as a proxy for physical size. Each bug can therefore categorize any other bug it meets as to sub-species; being one of parasite, prey, same, predator, or host. A bug, if so inclined, will chase and eat prey, and will avoid being host to a parasite.

We also use feeding habits to separate bugs into four categories: algivores (a made-up word), herbivores, carnivores and omnivores. Bugs which are able to exchange genetic material will choose to do so only with other bugs which share their feeding habits.

Is this true speciation? Restrictions due to complexity restrict the gene flow but do not stop it. This has the ability to establish sub-species/species which differ in behavior. Such differences meet the definition of species under definition B. Restrictions due to feeding habits are absolute and do define separate species according to definition A.

What happens when we develop two stable genotypes, such as is often the case in the Desert Oasis Ecosystem (DOE) scenarios? Each genotype is adapted to a geographic region (the oasis vs. the desert). This is reasonably close to the intent of the biological distinction between species under definition B, and in this case we can say we can have true species.

What about XOver and Ovule?

XOver enables gene flow from all bugs (of similar size) to those which have XOver enabled. If all bugs have XOver enabled, then they are all of the same species, because two-way gene flow between all bugs is enabled. If only some bugs have XOver enabled, then all bugs which are of the same size (complexity) as some XOver-enabled bug would be considered as in the same species. It is possible to evolve two contemporary 'definition A' species with XOVER, especially if feeding habits split the population.

Ovule enables two-way gene flow between all OVULE-enabled bugs of similar feeding habits and size (complexity). All such bugs are in the same species. If Ovule-enabled bugs of a given feeding habit exist at all sizes, then they form one species, as there is gene-flow between them all. If there is, however, a size barrier of sufficient width between two Ovule-enabled groups of bugs which share a common feeding habit, then we would certainly have two such species. This situation might arise, e.g., if some Ovule-enabled parasites stayed small to avoid predation, and some Ovule-enabled hosts stayed large to effectively prey on mid-sized bugs that were not Ovule-enabled.

When might multiple sub-species arise in PSoup?

The conditions which would encourage multiple contemporary species are:
- multiple terrain types with variations in the rate of replenishment of algae;
- a sufficiently high population (number of bugs) such that a reasonable number of extreme variants survive and reproduce;
- the availability of multiple strategies for the effective gathering of energy;
- the availability of multiple strategies for reproduction;
- behavioral barriers to mating;
- the ability to jump out of an evolutionary rut from time to time to explore other genotypes, without a severe penalty;

- a self-regulated sharing of energy between two or more genotypes such that all are encouraged to survive without direct competition.

PSoup provides:
- Flat terrain (not suitable for speciation), the Desert Oasis Ecosystem (suitable for two species), cascading paddies (potentially the best terrain for speciation), and independent paddies (good for non-interacting species development);
- Up to 500 bugs (small population, at best, but that's it!);
- Algae eating (all bugs), herbivores (smell algae, taste algae), carnivores (hear bugs, touch bugs), omnivores (see, hear & smell, touch & taste);
- mating barriers between bugs of different sizes, different repro modes, different feeding habits;
- Differing RAT & RET values, differing repro modes (fission, XOver, Birth);
- A continuous state space with smooth transitions rather than a discrete state space with precipitous transitions, an open energy system with variable inputs, adjustable rules of chemistry which can favor the generation of new genetic material; and adjustable scenarios on the edge of chaos;
- evolutionary defenses (camouflage, shells and spines, stealth mode, poisonous taste) to prevent predation and enable the sharing of energy between predator and prey.

Does Speciation Occur in PSoup?

It is easy to use PSoup to design scenarios which initially hold two species which interact. If evolution is allowed, virtually all of these are unstable and one species tends to evolve to obtain an advantage resulting in the extinction of the other. The PSoup arena is too small, and the rate of mutation and evolution too hyped up, to sustain multiple species for an extended period. If evolution is turned off, then stable scenarios with two or more species can be established.

In long evolutionary runs sub-species and species appear and disappear regularly. It is difficult to identify them, but with care they can be spotted, the action stopped, and the population analyzed for structure. What are the clues that might give a hint about the existence of two species:

Population distribution charts - if there are two clusters of values in any of the population distribution charts for C1 genes, C2 genes or C3/C4 capabilities (don't bother with the gene-level charts for C3 and C4), then there are two sub-species. For example, suppose that gene1 of the C1 charts showed values around 8 and 1 with a gap between, then we know that the alleles of this gene have diverged in value within the population, and potential sub-species have emerged. Similarly, if the population distribution chart for the C3 'SIGHT' capability showed values clustered around 10 and 250 with a gap in between, then we know that the alleles of this capability have diverged.

Cluster Analysis - Cluster analysis is good for identifying two, or maybe three clusters at most. If the population distribution charts hint that some clustering is happening, then use cluster analysis to break the bugs into groups. If there is a reference distance roughly equal to 1/4 to 1/3 of the diameter of the population which resolves the population into two or three or four clusters, that is very good indication of speciation.

Dissection of Representative individuals - If cluster analysis has indicated a good clustering of the population, then proceed to dissection. Use the 'Display Genes' button of the Tinker Bug Wizard to examine the genetic makeup of one bug of each cluster to determine whether they are potentially of different species. The questions to ask yourself are (a) is two-way gene flow possible (b) are there intermediate forms through which two-way gene flow is possible (c) is phenotypic behavior distinctly different?

PSoup and The Hardy Weinberg Law

In the first decade of the 20th century there was rapidly growing interest in the nature of heredity. Several biologists working separately re-discovered the insights that Mendel, the Austrian monk, had learned and reported more than 40 years earlier.

In short, they discovered that hereditary traits are not passed on as blendable qualities, but rather that they are passed as distinct qualities that appear to occur in organisms two at a time, one dominating the other and repressing its appearance.

For example, if Mendel took a pure population of red-flowered peas, and a pure population of white-flowered peas and crossed them, he did not get pink-flowered pea plants. What he got was all red-flowered pea plants. The white-flowered population seemed not to be represented in the resulting set of offspring. However, when he then took this first generation of offspring and divided them into two equal groups, and crossed the two groups, his second generation of peas were 75% red-flowered and 25% white flowered.

Mendel postulated that the tendency to be red or white-flowered was each stored in a distinct types of 'hereditary factor'. He said that each plant contained two such factors, and that it may have two of one kind, two of the other kind, or one of each. He said that the red factor was dominant over the white factor, so when the two types were in the same plant, the red factor was expressed, and the white factor was repressed.

In the late 19th and early 20th century these findings were re-discovered and confirmed. We now know that these 'hereditary factors' discussed by Mendel are contained in the genes, which reside in chromosomes in the nuclei of the pea plants cells. The different kinds of hereditary factors affecting the same phenotypic characters are called alleles. So, for example, he was working with two alleles, red and white, of the gene that controls phenotypic flower color.

We can distinguish between the genetic constitution of a single organism, and the constitution of the population of which the organism is a representative. Let's focus on a single gene, flower color, and two alleles, red (R) and white (W). An organism can have at most two alleles of that gene, one received from it maternal parent, and one received from its paternal parent. So the percentage of R alleles is one of 0%, 50% or 100. However, if we look at the population as a whole (and let's assume that the total population is very large, consisting of exactly 10.000 individuals) then the percentage of R alleles can be any value between 0 % and 100 %. The percentage of R alleles over the total number of alleles (R&W) is called the gene frequency for R.

The total number of alleles for flower color in the population is [2 alleles per organism] x [10.000 organisms] = 20,000 alleles. Suppose that 80% of these are red, and 20% are white. Then we say that the frequencies for alleles R and W are 0.8 and 0.2 respectively. We can then ask, what genotypes are likely to be found in the next generation, and we assume random pairing of the alleles with each other. We can answer this question with a variant of the well-known Punnett square. Consider the moment when each gamete is formed, and fertilization takes place. Looking at the eggs, we know that 80% of all eggs will have an R allele, and 20% will have a W allele. Similarly, 80% of the sperm will have an R allele and 20% of the sperm will have a W allele. The Punnett square looks like this:

		Sperm	
		0.8 R	**0.2 W**
Eggs	**0.8 R**	0.64 RR	0.16RW
	..0.3 W	0.16 WR	0.04WW

With these considerations, we know that the next generation must therefore contain 64% RR organisms, 32% RW organisms, and 4% W organisms. These are called the genotype ratios.

Let us suppose now, for each of calculation, that there are now 100,000 organisms in the population. What percentage of the alleles are now R, and what percentage are W? To compute the number of R alleles we take [0.64 x 100,000] + [1/2 x 0.32 x 100,000] or 80,000 alleles, or 80%. Similarly, to compute the number of W alleles we take [1/2 x 0.32 x 100,000] + [0.04 x 100,000] or 20,000 alleles, or 20%.

[Note: PSoup implements a tool called a pairing square, which is similar to a Punnett square, but not the same. A Punnett square is a commonly used tool which is used to predict the next generation based on the gene frequencies of the current generation. A pairing square is a tool unique to PSoup which tabulates the current genotype ratios regardless of the gene frequencies of the previous generation. The Pairing Square feature of PSoup allows the student to track genotype ratios from generation to generation. For more about the relationship between Punnett squares and Pairing Squares, and their use in PSoup, see the article "About Pairing Squares in PSoup".]

Assuming that all individuals mature and reproduce, we see that gene frequencies should not change from generation to generation, and genotype ratios should not change from generation to generation.

You should find the above statement troubling. Of course they should change. If they don't change, there is no evolution.

This issue was researched independently in the first decade of the 20th century by G.H. Hardy and W. Weinberg, and their findings are now called the Hardy-Weinberg Law.

A population for which the gene frequencies and genotype ratios stay constant from generation to generation is said to be in genetic equilibrium. Such equilibrium need not be confined to two alleles. A population may have many alleles of a gene, and the gene frequencies and genotype ratios will be computable. If these remain constant, the population is said to be in genetic equilibrium.

The Hardy-Weinberg Law says that "under certain conditions of stability both gene frequencies and genotype ratios remain constant from generation to generation in sexually reproducing populations."

The conditions of stability are"
- the population must be large enough to make it highly unlikely that chance alone could significantly alter gene frequencies;
- mutations must not occur, or else there must be mutational equilibrium;
- there must be no immigration or emigration; and
- reproduction must be totally random.

In other words, there will be a departure from equilibrium, and evolution of the population, if any of the following occurs:
- the population is small enough to suffer from genetic drift, the random variance of the gene frequencies due to chance;
- mutations are occurring;
- there is immigration;
- there is emigration; or
- there is non-random reproduction.

PSoup can be used as a laboratory to explore some of these conditions, to better understand them.

Size of Population

In the natural world, a population of less than 10,000 breeding organisms is considered to be at risk of evolution due to **genetic drift**. This may not sound serious, but it is. This is evolution which is not driven by adaptation and natural selection, but driven by chance and circumstances. It is non-adaptive change, and it usually does not serve the population well. In such small populations the number of alleles per gene usually declines rapidly to two or three, and often there will be only a single allele represented for some

genes in the entire population. When this happens, the population has lost its ability to adapt to changes in environment or circumstance. The final genotype may not even be the best genotype for the current circumstances, as natural selection has had too little time and material to properly form and select the best genotype based on the alleles available.

We can use PSoup to explore this aspect of the Hardy-Weinberg Law in Scenarios 6F and 6G, "Population Size and Gene Loss (I and II)". In these scenarios we have two populations with similar allelic constitution, except one population is 1/4 the size of the other.

In PSoup ALL populations are small, as we have a maximum number of bugs set at 500. This is only 5% of the size considered safe in nature. However, it is very difficult to make direct comparisons between natural ecosystems and PSoup systems. We have souped up the environment and the bugs in PSoup to ensure that evolution happens FAST. We can accomplish in a few generations in PSoup things that would normally take many many generations to complete in nature. Nevertheless, 500 bugs is a small population in PSoup, and such populations do exhibit genetic drift. It is not possible to produce a truly stable multiple allele population in PSoup, due to this genetic drift.

However, we can track and measure the relative increase of speed of loss of alleles when we cut a population of bugs to 1/4 of its size. That is what we do in these two scenarios.

No Mutation, or Mutational Equilibrium

In nature mutations happen all of the time. You cannot stop them from happening. Its a fundamental characteristic of DNA that it can recover when disrupted, mend itself and carry on. When the mended DNA looks different from the original DNA, a mutation has happened. If that mutation results in functioning active DNA, then a new allele has been added to the gene pool. It is estimated that one mutation occurs in every 50,000 - 1,000,000 reproductive events. The slow accumulation of such mutations in the gene pool over hundreds or thousands of generations is the source of the variety of alleles in a population. A population with many alleles for each gene is well-prepared to respond adaptively to changes in environment or circumstances.

Mutations cause changes to gene frequencies (i.e. allelic frequencies) and disturb genetic equilibrium.

Mutations between existing allele values change the gene frequencies but do not add new alleles. If all mutations of alleles result in no long-term change in the gene frequencies, then **mutational equilibrium** is considered to exist. It doesn't take a rocket scientist to figure out that this will be very rare indeed. So, we will disregard the issue of mutational equilibrium.

When mutations occur continuously, if slowly, and continue to add new alleles to the population, this can cause a shift in the gene frequencies. Such a shift in gene frequencies, as for genetic drift, is not driven by natural selection or adaptation to environment or circumstances. It is driven the chemistry of DNA, the availability of disruptive influences, and the frequency of usable mutations produced. This phenomenon is called **mutation pressure**.

PSoup enables you to investigate the phenomenon of mutation pressure. In Scenario 5E "Biased Mutation Rates" we examine the effect of mutations which are persistently biased. For example, for C1 chromosomes we know that large B genes (that for Backwards, gene4)are BAD, and small B genes are good. But what if the entire population has large B genes and the only type of mutation allowed in the rules of chemistry are those which cause larger genes. In this scenario we do not specifically examine how such biased mutations take us away from genetic equilibrium. We examine how evolution responds to such perversely biased mutations. For more information see 'PSoup and Biased Mutations' in PSoup Theory and Practice.

For an example of a PSoup population which is sexual, homozygous in all genes (i.e. there is one and only one allele of each and every gene in the population) and in which no mutations occur, see scenario 5D. This scenario is designed to test Hardy-Weinberg Law with respect to non-random reproductive pressures,

but it can also serve for this purpose. The default population is designed to be in perfect Hardy-Weinberg equilibrium. Run the default population as is. Then restart the scenario, go into 'Tinker with the Environment' but rather than changing the reproductive bias, just simply turn up the rate of mutation. Set it to 0.125. Then re-run the scenario. (Don't forget to click on the 'Use existing bugs' box after setting the mutation rate, but before leaving the tinker dialog.) Watch the allele counts and allele values and pairing squares change over the generations. You will then see how mutations disturb the equilibrium and allow natural selection to move the population away from the point of equilibrium.

So, in summary, there are two aspects to mutations.
- Biased mutation rates can cause mutation pressure, which is a pressure that pushes a population away from equilibrium, for good or for ill. This is evolution which has a large component which is non-adaptive.
- Non-biased mutation can provide the fuel, the phenotypic variants, which natural selection can use to move a population away from equilibrium. This is adaptive evolution.

Immigration

Immigration is due to the influx of organisms from outside of the population. These organisms must, of course, be able to mate with the members of the population and produce viable offspring. If such immigrants come from a population which has a different set of gene frequencies, or even new alleles, then they will disturb the genetic equilibrium. This is, perhaps, self evident.

It is possible, again using the default population of Scenario 5D (Biased Reproductive Rates) to test this. Run the default generation for a while to convince yourself that they are indeed in genetic equilibrium. They are homozygous, having only a single allele per gene, and mutations are turned off, so of course they are in equilibrium. Now, go into the 'Tinker with the Bugs' dialog and add a new allele. Next generation, do the same. Next generation, do the same. You will see the gene frequencies shift, of course.

The more interesting question is 'How do the genes migrate into a population. Scenario 6E "Migration of Genes" is designed to explore this issue. In this scenario we have a population which was in genetic equilibrium, except that a few bugs carry one variant allele which is decidedly beneficial. You can track the progress of the new gene as it is welcomed into the population and establishes its place in the gene pool.

Emigration

The effect of emigration is different. If organisms that leave the population (never to return) are somehow randomly chosen, then the genetic equilibrium will not be disturbed. If, however, there is some phenotypic character that makes some bugs more likely to leave than others, then that allele will be depleted more than others, and the genetic equilibrium will be disturbed. It is hard to imagine a natural situation in which the emigration of organisms from a population would be totally random. In all cases, any slightest characteristic which would allow one organism to avoid removal when another cannot avoid it would make the removals non-random. In most cases, therefore, it would seem that emigration will disturb genetic equilibrium.

However, the effect is more immediate than for immigration. With emigration the full effect is immediately felt. The wandering/dangerous alleles are immediately missing and removed from the population.

In the case of immigration, the new alleles immediately modify the gene frequencies, but if and as they are absorbed into the population, especially if they are new to the population, they can continue over many generations to adjust the balance of equilibrium.

PSoup does not have the tools to examine emigration specifically. It is possible to see emigration in action sometimes. If you have two or three cascading paddies and an omnivore or herbivore is in a left paddy, and food is running scarce, you will see it running back and forth along the right side of the paddy until it escapes into the greener pastures on the other side of the paddy's edge. It has emigrated out of the population in the left paddy into the population in the right paddy, never to return. There is a one-way flow

of genes from left to right. However, the current implementation of PSoup does not have the ability to examine population statistics within a single paddy, so the effect cannot be tracked easily.

Random Reproduction

The members of a population enjoy totally random reproduction if all organisms are absolutely equal in all phenotypic characters that might in any way affect their probability of survival. This would only be the case if the population is homozygous in all genes (both alleles the same) and there were no variant alleles in any member of the population for any gene. NOT LIKELY.

The default population for Scenario 5D "Biased Reproductive Rates" is such a population. It lives in genetic equilibrium. It is designed to be run (a) in default mode, and then to be run in two other modes in which (b) there is a range of SMELL-enabling alleles, and a mating instinct to mate with the best nose and (c) there is the same range of SMELL-enabling alleles, and a mating instinct to mate with the worst nose.

The selection of the two options is through the 'Tinker with the Environment' dialog, and is effected via a change in the rules of chemistry of love making (a kind of chemistry in PSoup). Selection of the option (i) creates a range of alleles for the relevant genes, and (ii) toggles the instinct to be careful, when selecting a mate, about noses.

This introduces a non-random aspect to reproduction (multiple alleles that affect the ability to find resources, in competition with the contemporaries in the population) but also adds a bias in selection of a mate.

This is a tricky one. Can you predict what the results will be when natural selection goes to work on the (c) population?

Evolution Without Mutation

That brings to a conclusion the review of the Hardy-Weinberg Law, but we have one more consideration to look at. We've looked at the factors that disturb a state of genetic equilibrium and cause evolution to occur, but we need to put them into perspective.

A population can be in genetic equilibrium if and only if it is genetically very pure. Pure means having few or one allele for every gene.

Evolution works when a range of variant phenotypes exist within the population, and natural selection selects next-generation survivors with gene ratios different from existing gene ratios in the current population.

Mutation, immigration and emigration change the gene frequencies, and this may slow down or speed up the rate of change under natural selection.

Even if natural selection is predisposed to select next-generation survivors with gene frequencies consistent with existing gene frequencies, in small populations genetic drift and mutation pressure will cause shifts in gene frequencies, a loss of alleles and loss of adaptability.

PSoup as a Finite State Machine

Finite State Machines

A Level 1 bowl of PSoup may be viewed as a finite state machine (often shortened to FSM). A finite state machine is a computational model which is often used to formally model a real-world system, or to solve certain types of complex problems. Finite state machines also are of intrinsic interest in their own right, and have been studied seriously by mathematicians for many years.

What is a Finite State Machine?

A finite state machine is a system which can exhibit a number of discrete states of existence, each clearly distinguishable from the other, and which can change itself from one state of existence to another via the application of a transition function, or transition process.

In simple terms, a finite state machine is:
(a) a finite list (either explicit or implicit) of all possible states of existence that the system might theoretically display (states of a system are explained later);
(c) a transition function or process that is used consistently to cause a change of state of the system; and
(b) a pre-selected starting state.

Here are a few examples of some very simple natural 'systems' that can be modeled using finite state machines:
- the pendulum of a grandfather clock has two states (left and right); a transition process (swing to the other state under the forces of gravity and the internal spring) and a starting state (either left or right);
- a light bulb with a standard switch activated by a pull-chain connected to it has two states (on and off); a transition process (flip the switch by pulling on the chain); and a starting state (either on or off);
- a standard 52-card deck of playing cards has 52! (that's 52 factorial) states (each possible sorting order of the cards); a transition process (shuffling); and a starting state (the order of the cards on leaving the factory);
- a tic-tac-toe board has 3^9 (that's three raised to the ninth power, or 19,683) states in which each of the nine cells has either an X, an O, or nothing; a transition process (add an X or an O to an empty cell); and a starting state (all cells empty).

The pendulum and the light bulb can be represented by a VERY simple FSM which has one dimension with an extent of 2. The transition function is a simple toggle which results in the FSM moving to the only other state each time it is executed.

The deck of playing cards can also be represented by a one-dimensional FSM, but this time the extent along that one dimension is vastly larger. Each possible sorting order of the deck of cards is a different state of existence. There are 52! Such sorting orders. To calculate 52! you multiply 52 times 51 times 50 times 49 … times 3 times 2 times 1. This is approximately equal to 8.1 times 10 raised to the power 67 (8.1E67).
There is another difference. The transition function (a thorough shuffling of the cards) is non-deterministic, but rather probabilistic. That is to say, the transition function has an equal probability that it will cause the system to occupy any state, even the current state. It is possible to shuffle a deck of cards very well and end up with the same sort order, although it is not very likely. With the pendulum or the light bulb, the transition function is deterministic. If you know the current state, and you apply the appropriate transition function, you know exactly what the next state will be. With the deck of cards, you only know the probable next state (all are equally probable) but you cannot know the next state until the transition function has completed execution.

The tic-tac-toe board is also interesting. First, the FSM is not one-dimensional, it is nine-dimensional. Each cell can independently have a value of X, O, or N (for naught or nil). But there are nine cells. This means that there are nine independent parameters which describe the state of the system (one parameter for

each cell) and three possible values for each parameter. We can imagine the collection of states to form a cube-like 3x3x3x3x3x3x3x3x3 structure in 9-dimensional space, with a total number of states equal to 3^9, or 19,683. Such a collection of states in an n-dimensional cube is called the state-space of the system. The location of any state can be described by an ordered n-tuple (an ordered set of n integers) which position the state in a Cartesian coordinate system. Suppose we number the cells of the tic-tac-toe board from left to right and from top to bottom, giving each cell a number starting from 1.

1	2	3
4	5	6
7	8	9

The number of the tic-tac-toe cell determines the location in the ordered n-tuple. Suppose also that an empty cell is annotated as a zero (0), and X by one (1), and an O by two (2). Then a board with a single X in the central cell of the board, and all other cells empty, would be located at (0,0,0,0,1,0,0,0,0) in the state space. An empty board would be located at (0,0,0,0,0,0,0,0,0), and a board with all O's would be located at (2,2,2,2,2,2,2,2,2).

We have now defined a state space for an FSM which simulates a game of tic-tac-toe. There are a finite number of dimensions (9). Each dimension has a finite extent (in this case, all dimensions have an extent of three). Each state of the system is discrete from all others (non-overlapping in any way) and unique.

Now, let's look more closely at the transition function. Up above, we defined the transition function as 'add an X or an O to an empty cell'. While this transition function will move the system from the starting state to an eventual end state in which all cells are filled, in nine moves, it does not simulate a game of tic-tac-toe very well. The moves of alternate players are not simulated, and the game-end conditions (three in a row) are not simulated. How can we simulate a game of tic-tac-toe? Let's formulate a new transition function, then see how it works.

tic-tac-toe transition function number 2:
- first, add an X to the first empty cell, then alternately add O's and X's to the next empty cell until the system contains three X's or three O's in a row, at which time all applications of the transition function add neither X's nor O's, leaving the system in its current state.

If we now run a 'game' of tic-tac-toe, X plays in cell 1, O plays in cell 2, X plays in cell 3, etc., until X plays in cell 7, at which time X has three in a row in cells 3, 5 and 7. The FSM continues to apply the transition function which says 'add neither X's nor O's, leaving the system in it's current state'. It doesn't matter how long you leave the FSM running, the 'game' is over. This is much closer to a true game of tic-tac-toe, but it is deterministic in nature. Each and every 'play' is 100% pre-determined by (a) the starting state and (b) the transition function. The system moves through the state-space along a pre-ordained unchangeable path (called an orbit). How can we add variability to the outcome?

tic-tac-toe transition function number 3:
- first, add an X to one of the empty cells according to the rules of selection, then alternately add O's and X's to empty cells (again, according to the rules of selection) until the system contains three X's or three O's in a row, at which time all applications of the transition function add neither X's nor O's, leaving the system in its current state;
- the rules of selection are '(a) select any empty cell, all empty cells having equal probability of selection'.

Transition function number 3 will simulate the moves of an actual game of tic-tac-toe. Finally, we can add strategy to the playing of the game by improving the rules of selection to read: '(a) organize all empty cells into four non-overlapping possibly empty groups as follows (i) those empty cells which complete a 'three-in-a-row' sequence; (ii) cell 5; (iii) cells 1, 3, 7 and 9; (iv) cells 2, 4, 6, 8; (b) select any empty cell from among those groups, going through the groups in priority order, until a non-empty group has a cell to select; and (c) if the first non-empty group encountered contains more than one empty cell, all such cells have equal probability of selection.

These 'rules of selection' are called the 'instructions' for the transition function, and one can vary the instructions to the transition function. Other 'instructions' are the input from the computer's pseudo-random generator, which are used to determine the correct choice between options which have probabilities assigned.

We now have an FSM that will simulate a pair of skilled players in a game of tic-tac-toe. Note that, X always goes first, and X will always select cell 5. O will have four cells to choose from, selecting a corner cell. Play will continue to an inevitable draw. We have a non-deterministic FSM which nevertheless has, due to the constraints placed on the transition function, a deterministic outcome, in terms of wins and losses.

For the sake of what follows with PSoup, let's take this discussion one step further. Suppose that we can vary the starting conditions. Instead of starting with an empty tic-tac-toe board, suppose that we have one X and one O randomly placed on the board prior to start. The same transition function can be applied, but we have a random starting state (from within a list of allowed starting states). With this technique we can open up the variety of paths (or orbits) through state-space that a game might take.

But this definition of the FSM provides for a number of 'possible' states of the tic-tac-toe board which will never exist in any run. For example, any state in which the number of 1's and 2's in the 9-tuple differ by 2 or more will never come up. The 9-tuple (1,1,1,2,2,2,2,2,0) has 3 X's and 5 O's. The O's have had more turns than they deserve. The difference between 5 and 3 is 2. This is a 'possible' state which is unreachable by the transition function from any acceptable starting state. Under transition function 1, which had very few restrictions, every possible state in the state-space was reachable via repeated application of the transition function. Under the other transition functions a very many of the possible 19,683 states are unreachable unless the starting state is carefully chosen. Even then, some states may only be expressed as an actual board configuration if they are chosen as the starting state.

Note that a deterministic transition function applied to a start state defines an explicit path of states in the state space. Similarly, a probabilistic transition function applied to a start state defines a set of probable paths. These paths may fork and never rejoin, or they may rejoin. Each state which lies on one of these probable paths has a probability that it will eventually be exhibited in the game as it unfolds. Those that do not lie on such a probable path will never be exhibited.

In summary, by looking at these simple applications of FSMs, we have briefly reviewed:
- state spaces;
- dimensions and extents;
- deterministic and non-deterministic (probabilistic) transitions functions;
- impact of changing starting state.

Now let's look at a really simple bowl of PSoup.

Can we list all the states that PSoup might display?

Let's address this question looking at the most simple bowl of PSoup:
- a Level 1 bowl;
- 10 cells wide by 4 cells deep;
- wrap is turned off;
- terrain is normal flat terrain;
- closed energy system; and
- the total energy is 4,800 Nrg units.

This energy loading can be achieved with a loading factor of 100% (1600 Nrg units) and four bugs (3200 Nrg units) at start.

Practice with a Simple Example

Just before we tackle this 'most simple bowl' of PSoup, let's practice with a very simple FSM. Suppose we have a finite state machine (FSM) with only three parameters that change. Call them A, B and C. Suppose further that A can have only three values: 0, 1 and 2; B can have only four values: 0, 1, 2 and 3; and C can have only five values: 0, 1, 2, 3 and 4. We then say that A has an extent of 3; B has an extent of 4; and C has an extent of 5. If we allow all three parameters to range through all possible values, we have a complete list of all possible states that this little FSM can exhibit. We could describe each state fully with an ordered triple (a, b, c) where a is a value of A, b is a value of B and c is a value of C.

To compute the number of states possible we look at the three-dimensional cube of cells, with extent 3 times 4 times 5. There are a total of 60 cells in this FSM's state space.

In the abstract, to count the number of possible states we:
- identify all of the parameters that might change when the FSM changes state, each parameter defining a dimension of the state space;
- for each parameter, identify the number of possible values it can take on (i.e. the extent of the parameter);
- multiply all of the extents together.

To list all of the possible states we:
- list all of the parameters that can change;
- assign arbitrary order to the list of parameters;
- create a list of ordered tuples in which each element must be a value drawn from the extent (or range) of the corresponding parameter;
- list all such tuples;
- the total number of tuples will equal the number computed above, and is equal to the number of cells in the multi-dimensional cube that is the state space of the FSM.

Apply This Idea to PSoup

First, look at the bowl. There are 40 cells. Each cell can have a colony of algae, or no colony of algae. Since the loading factor is 100%, it is possible to have a colony of algae in every cell at the same time. This makes the calculation easier for us, but makes the total number of states of the bowl very high. For each cell there are two possible states. For the entire bowl, there are 2^40 possible states. That is 2 raised to the power 40. In base 10 that's 1.09951 times 10^12 possible states, or just over 1 trillion states.

Next, look at a single Level 1 bug. Gene8 is inert, so there are 8 active genes (the Palmiter genes from gene0 through gene7). At least one other parameter, the age of the bug (in PSoup seconds) is important. This gives us a total of 9 parameters per bug. All of the genes of the Type 2 chromosomes are common and fixed in value, so they do not change throughout state space, and therefore do not define different states. Each Palmiter gene has a fixed base of 2, which does not change throughout state space, so it does not define different states. Each gene has an exponent which can range from -150 up to +150, 301 possible states. The total number of possible states, then, for a bug, is 301^8 (i.e. 301 raised to the power of 8) times 1600(extent of age parameter). In base 10, that's 1.07808 times 10^23 states, or just under 112 billion trillion states.

We're not finished. Each cell can hold up to two bugs. While it is not likely, we need to see whether we have enough energy to support a colony of algae in every cell plus two bugs in every cell. DET says that any bug with less than 30 Nrg units will die. A bug needs a minimum of 31 units, therefore, to live. Since we have 3200 units available, after all cells are filled with algae, we have enough energy left over, THEORETICALLY, for 103 bugs. More than enough for eighty.

At this point you may be wondering, how would you get two bugs into each cell (by fission), and would new daughters have only 31 units of energy (only if they were gridlocked and could not move for many turns, which is true), and if they are this hungry, how would all of the cells have algae (only if on the immediately previous turn all cells were filled simultaneously)? What this line of questioning tells us is that, there are some states in PSoup's state space which are only accessible if we use them as a start state, or otherwise select the start state to make them possible. The same situation exists for our FSM of tic-tac-toe.

The issue here is that we are about to include some states which may not be accessible via the transition function, even though we can describe them. We saw this in the tic-tac-toe example above.

So, each cell can have two bugs. That's 80 possible bugs with 10^23 variant states per bug. The total number of states due to the bugs is ((10^23)^80) or roughly 10^1843 possible states. When we combine this with the roughly 10^12 possible states of algae in the bowl, and the total number of states in the state space of a most simple bowl of PSoup rises to roughly 10^1855. Our universe is roughly 10^17 seconds old. If we enumerated one state of PSoup per second for the age of the universe, we would not have begun to list the entire list. 10^1855 is a truly BIG number. We cannot even say that its an ASTRONOMICAL number. Astronomical numbers pale and disappear by comparison.

So, in total, the most simple bowl of PSoup can exhibit as many as 10^1855 different possible states of existence, give or take a few.

Note the following:
- PSoup is very simple when compared to the real world, but, even the most simple bowl of PSoup is able to exhibit a truly immense number of possible states of existence.
- We cannot explicitly list all of the possible states of a simple bowl of PSoup, however, we can implicitly describe them all, and have done so above.

What are these states?

A state is a single comprehensive description of a bowl of PSoup AT THE END OF A PSoup SECOND. To be sure, as a PSoup second unfolds (as the bugs take their turns, bug-by-bug) the bowl of PSoup goes through intermediate states which look identical to the end-of-second states. There is a difference, however, which is important in PSoup, and possibly in other finite state machines. Between end-of-second boundaries, some bugs have had a turn and some have not, and at this point the computer has given a slight advantage to the bugs that come first in the list. A key quality of PSoup is that the computer never accidentally gives a bug an unfair advantage. So, we define a state as a configuration of the system in which all transition processing is completed.

Note that most PSoup functions and features are inaccessible when the transition process is partially complete. Click the 1Bug or 1Fn buttons and watch what happens to your menu bar and toolbars. They all go dead except for those that progress the unfinished work of the transition function. Continue clicking the 1Bug or 1Fn buttons until a second is complete. Accessibility to all items is restored.

More generally, it is a key concept in finite state machines that time is discrete and moves forward one tick at a time. Partial ticks have no meaning.

PSoup's State Space

The collection of all possible states is called the state space, of the finite state machine. It can be viewed as a large multi-dimensional space consisting of little multi-dimensional cubes, something like a Rubik's cube. A bowl of PSoup can only exhibit one state of the state space at a time. That is, it can only exhibit the defining characteristics of one single state at the start/end of any tick of the PSoup clock. So one little multi-dimensional cell of state space is occupied, the rest of the 10^1855 cells are empty.

Note that each different sized bowl of PSoup at any given level has a different state space. Note that similarly sized bowls of PSoup which are at different levels have different state spaces. Note that, furthermore, two similarly sized bowls which are at the same level (e.g. level 3) but which have different rules of chemistry or which differ in certain other parameters will have different state spaces. The state space is determined by (some of) the layout parameters.

Note also that we are now using the word 'cell' in two ways. A cell in a bowl of PSoup is one of the square 2-dimensional areas which make up the bowl of PSoup. There are 40 cells in our most simple bowl of PSoup. A cell is also a potential state in state space. There are roughly 10^1855 cells or states in the state

space of our most simple bowl of PSoup. Each time we use the word cell, it will be qualified as to its meaning.

What is the starting state?

We just saw that a system (an FSM) can only occupy one cell of its state space at a time. When the FSM is first built, we must (somewhat arbitrarily) decide into which cell of state space it should be placed. That cell is called the starting state, and the bowl of PSoup is exhibiting its initial conditions for this run.

Here's the same idea using different words. A bowl of PSoup can only exhibit one configuration, one variant of its parameters, at a time: algae here but not there; so many bugs; such-and-such genotypes. Before we start a scenario running, we must build a bowl of PSoup with a valid set of parameters. When we have finished building the scenario, but have not yet run the first PSoup second, the bowl is in its starting state. The initial conditions are fixed.

The starting state is determined by the computer when a scenario is first displayed. The starting state is parametrically described before the bowl is laid out. The starting state is fixed when the bowl is completely laid out. Let's understand this by looking at a number of parameters, and how they are converted into a single starting state:
- One parameter, e.g., is the size of the bowl. This is straight forward. The parameter determines the cell size required for a bowl of that size and lays out the cells and the mud in the proper dimensions.
- Next is the energy loading factor. If it is set at 0% there will be no algae in any cell, and the total number of possible starting states is drastically reduced. However, if it is set at 100% it is possible that each and every cell can start with a colony of algae (although not very likely). By setting this parameter, we establish a constraint on the starting state of the bowl of PSoup, and limit the number of available states from which we can choose our starting state. (Note that we also limit the range of states accessible via the transition function.)
- Another parameter is the number of bugs at start, in our example it is 4. This is a little more complex. Of the 10^1855 possible states, there are many possible states which have more than or less than 4 bugs. These are ruled out. However, there are still trillions of possible states with 4 bugs. This parameter has defined a group or class of possible starting states, and by randomly assigning the four bugs to four specific cells, it effectively chooses a small subset of those trillions of states as the starting state.
- What about the contents of the genes of the four bugs. There are trillions of possible states for each bug. For each scenario, there is a process invoked to set a combination of genes for each bug. The bug's genes are set before the bug is placed into the bowl. As each gene is set to its target value, we narrow down the list of states that might still remain as the one starting state. In our example, all bugs are pristine, all gene exponents having a value of 0. As we set each gene's exponent, we rule out all states in which that particular exponent had a different value in its range (extent) of values.
- Another parameter is the Algae Distribution Mode. As we randomly distribute each colony of algae into the bowl of PSoup (or decide not to distribute it), we successively determine a more and more narrow subset of all possible states, until, when the last colony of algae is finally placed, the starting state of this bowl of PSoup is fully determined.

When a scenario is ready to be started, PSoup is in one, and only one, of the some 10^1855 possible states. This is called the starting state, or initial state. The initial conditions are now fixed.

Note that the user can specify a starting state in parametric terms only, and the computer translates that into the selection of a single starting state. It is understood that the user can use the tinker command to change number of bugs, to change the values of the gene's exponents, and even to re-locate the bugs. However, the user can only specify the 'Algae Distribution Mode' and the 'Energy Allocation Factor' but cannot explicitly locate each colony of algae in a cell, or in the mud. The actual starting location of each colony of algae is done by the computer under the instructions of the pseudo-random number generator. This is why we say that at best, the user can only specify the starting state parametrically.

What does the transition function do?

The transition function is a function or process that works on input to produce output. One of the inputs is a bowl of PSoup in a known state. The second type of input is called instructions. The output is a bowl of PSoup in a known state.

We could define the transition function like this: the transition function takes a system which exhibits a valid state of existence and causes a change of state, such change being under the control of external instructions.

A finite state machine (FSM) then operates like this. The system is in a starting state. The transition function begins to perform its operations with the goal of causing the system to jump from its current state and enter another state. It reads its instructions. It transforms the system to occupy another cell in the state space. One tick of the machine clock has now passed. The transition function starts its operations again, reading instructions and modifying the system, causing yet another change of state.

As the machine clock ticks, the system moves from cell to cell within the state space. For any tick of the clock the 'from' and 'to' cells may be adjoining, in some sense, or may occupy radically different locations within the state space.

Deterministic and Non-deterministic transition functions

A deterministic transition function has a pre-determined set of fixed invariant instructions. The result of application of the transition function is pre-determined. For every state in the state space which might be used as input to the transition function, there is one and only one output state possible. In this case, ***the starting state pre-determines all future changes of state in the system under the action of transition function***.

A non-deterministic transition function has a set of instructions which vary. The way that the instructions vary is independent of the state of the system, and is expressed in probabilistic terms. For any given state in the state space which might be used as input, there may be more than one possible output state.

PSoup uses a non-deterministic transition function which is under the control of an instruction set which has a strong probabilistic component generated by the pseudo-random number generator. In anthropomorphic terms, whenever the PSoup transition function needs a variant instruction, it goes to the pseudo-random number generator, gets its instruction, and then carries on in its work to transform the bowl of PSoup from one state to the next.

What is the difference between the Initiation Function and the Transition Function?

PSoup has two functions which have roles that are similar, but not the same. First, it has an initiation function. This function is used to interpret the startup parameters, to go to the pseudo-random number generator when a variant instruction is required, and to correctly select a starting state for the PSoup finite state machine. This function is invoked under four circumstances. The initiation function is invoked when:
- PSoup is first turned on or uses the File/New command; or
- the user requests a new scenario using the 'Scenario' menu item, or the DOE toolbar button; or
- the user selects the 'ReSeed' menu item to restart a scenario; or
- the user uses the 'Tinker with the Environment' menu item to cause some or all of the parameters to be changed.

In all of these cases the initiation function chooses startup parameters (and the process used to choose those startup parameters may vary greatly depending on the mode of invocation), interprets those parameters with instructions, when required, and lays out the bowl.

However used, the initiation function selects a start state for PSoup.

The transition function assumes that PSoup occupies one and only one state in the state space, and moves it from state to state.

How does PSoup's Transition Function Work?

The transition function is described in detail under various topics, but in outline, here it is. For each tick of the PSoup clock, the transition function executes three steps:
- a pre BugLoop process;
- a BugLoop process; and
- a post BugLoop process.

The pre-BugLoop and post-BugLoop processes are also called pre-PerSecond and post-PerSecond processes. The BugLoop process addresses each bug in turn in order of their appearance in a linked chain called, surprisingly, a BugList. For each bug, the transition process executes these seven sub-processes:
- pre-PerBug;
- the Sensing Function;
- the Moving Function;
- the Feeding Function;
- the Repro Function;
- the Dying Function; and
- the post-PerBug process.

When all of this has been done, PSoup has been successfully transition from one state to another.

In PSoup it is possible to execute a partial transition function (e.g. advance by 1 Bug) but all escape routes for the user are disabled until the transition function is fully executed (e.g. via advance by 1 Sec). When the transition function needs a variant or probabilistic instruction (e.g. for selecting a Palmiter gene for expression, for other 'events' as defined in the 'Event Counts' panel, for the distribution of algae, for the resolution of combat) it gets it from the pseudo-random number generator.

Number of dimensions

As mentioned before, the state space for our most simple bowl of PSoup is multi-dimensional. On the face of it, our most simple bowl of PSoup has at least 760 dimensions. For each cell, algae provides one dimension, and two bugs provide 18 more, for a total of 19 per cell. With 40 cells, the total number of dimensions is 19 times 40, or 760, (excluding various minor parameters).

This view of PSoup may appear to be needlessly complex. At startup of our sample bowl of PSoup there are ONLY FOUR bugs. What does it mean to have 720 dimensions of the state space devoted to descriptions of the bugs' genes when we only need 4x9=36 dimensions for the four bugs at start-up. This is a difficult issue and we will not go into all the details here. Suffice it to say that the state space must be sufficiently broad in its definition to cover all possible states (or configurations) of a bowl of PSoup. From the point of view of a mathematician, each cell of a bowl of PSoup always contains a potential of two bugs. That potential exists, whether or not it is actualized. That potential adds dimensions to the space of all potential states. When a bug comes into being through fission, or ceases to exist due to death, certain parameters may change values, some dimensions may gain or lose relevance (to the definition of the state, and to the next application of the transition function), but the number of dimensions in the full description of the state space remains constant, and high.

Distance between states

Within this state space, it would be useful to define a distance function such that we know how far it is between two states. Without being too rigorous, we are going to do that now.

Two states in a PSoup state space are said to be adjacent (i.e. roughly one unit of state-space distance apart in each dimension) if, for each and every dimension, the difference in the value of the parameter is less than or equal to the maximum unit of change possible in a single pass of the transition function.

What does this mean in practice? In our most simple bowl of PSoup, a gene changes value when it mutates. The exponent changes in value by an amount of DPG (Delta Per Gene), which is set to equal one. So, the exponent goes up in size by one, or down in size by one. It has 301 possible values. So, with this in mind, two states are adjacent if all corresponding genetic parameters (the 640 exponents of the Palmiter genes) are a distance of no more than one apart. With respect to algae, since these dimensions (40 of them) have an extent of 2, the transition function can only add or remove a colony, so the concept of adjacency is moot. With respect to the age parameters of the 80 bugs, a bug can change by one second at a time, so the concept of adjacency is meaningful and obvious.

Let's add the condition that a state cannot be adjacent to itself.

Let's try to size the number of adjacent states.

At the high end, without going into the details of the computation, it is estimated that a state in our most simple bowl of PSoup can not have more than ((3^760)-1) or 10^334 adjacent states. This looks like a lot of adjacent states, and truly, it is, but it is an infinitesimal fraction of the total number of 10^1855 states. The '-1' represents the removal of the current state. This includes all of the states which might precede this state, and all those that might follow it. So the estimate of the number of 'next' adjacent states would be 10^334 / 2, which is, still, due to rounding, roughly 10^334.

At the low end, we expect there to 4 or more bugs in the bowl. Each bug can move to 8 cells, unless they interfere with each other (assume they don't). That's 8^4 possible moves, or 4096 moves. We have 9^4 possible mutations, or 6561. In total there are, practically speaking, in typical circumstances, only 4096 * 6561 or 3 * 10^7 next states.

The state space for a (most simple) bowl of PSoup is truly immense, with a mind-boggling array of optional adjacent next states for every state.

Let the formula for the distance between two states be the Manhattan Block formula, i.e. the sum of the differences in coordinate values taken on a dimension-by-dimension basis. E.g. in a two-dimensional system D = (x + y) where x = (x2-x1) and y = (y2-y1), the changes in value of the x and y parameters respectively. This formula is based on the answer to the question, how far is it from this corner to that corner in a city map? Count the city blocks in each dimension and add.

Then the nearest adjacent state would be exactly one unit of distance away (i.e. one colony of algae, or one DPG away) and the most distant adjacent state would be 760 units of distance away (i.e. a change on one in each dimension).

Invertibility of the transition function

A transition function is said to be invertible for a given pair of states if, for some set of instructions, state A can be transformed into state B, and, with another set of instructions, state B can be transformed into state A. The transition function of PSoup is definitely NOT invertible. For example, bugs age, and that aging processes is irreversible. A bug's age plays a role in the determination of state, and the execution of the transition function. There is no set of instructions which will make a bug younger. As another example, bugs expend energy while moving, and that expenditure of energy is irreversible, since it is redistributed as algae immediately, and cannot be recalled.

Orbits and Radiation Paths

In an FSM which has a deterministic transition function, the selection of a start state automatically defines a set of states called an orbit. The orbit of a state A is the set of all states in the state space which can be exhibited by repeated application of the transition function an arbitrarily large number of times. You remember that, with a deterministic function, there is one and only one possible output state B when the function F is applied to A. We say B = F(A). B is uniquely determined by F and A. Similarly, C = F(B) is uniquely determined by F and B. And so on. F and A uniquely determine an orbit, a thread of such

adjacent states running through the state-space. The orbit must eventually loop back onto itself, or hit the edge of state space and come under the control of edge effects built into the transition function (see the article on PSoup and Edge Effects).

In an FSM which has a non-deterministic transition function, the selection of a start state automatically defines a set of states which we shall call the radiation path. The radiation path of a state A is the set of all states in the state space which can be exhibited by repeated application of the transition function an arbitrarily large number of times. With a non-deterministic transition function, there are many possible output states B when the function F is applied to state A. Let B be the set of all output states of F(A). Similarly, C = F(B) is the set of all possible output states when the function F is applied to any state in set B. D = F(C). E = F(D). Etc. F and A uniquely determine a radiation path for A, a set of interconnecting, forking, threads of adjacent states running through the state-space.

PSoup is non-deterministic, so PSoup has a radiation path. As time moves forward, tick-by-tick, the transition function takes the current state as the input state, selects among those possible adjacent output states, and moves PSoup into that output state. Of all of the possible output states, one is actualized. Looking into the past, PSoup has followed a single orbit. Looking into the future, PSoup has a radiation path, which offers a myriad of possibilities.

Let's define a radiation thread as any possibly connected path of states within the radiation path which might be actualized. Every state which is an element of the radiation path is an element of at least one radiation thread. The starting state is an element of every radiation thread.

PSoup's Radiation Path

So, we can picture in our minds, as a PSoup scenario unfolds in time, the system (bowl of PSoup) is moved from state to adjacent state, along a contiguous path, which is one path among those in the radiation path. And there is a natural direction in which the movement will occur. For example. for every state there will be adjacent states in which all bugs are younger, and adjacent states in which all bugs are older. Some with fewer pregnant bugs, and others with more younger bugs. You can imagine that, at every point, multi-dimensional state space has an uphill and a downhill direction, and the direction of motion is downhill. You can picture the system, the bowl of PSoup, represented as a ball bearing rolling gently down a sloped hill, following a path of least resistance. Since the aging of bugs is irreversible, any adjacent state in which all bugs are one second older may be considered to be downhill from the current state, and any adjacent state in which the same bugs are one second younger can be considered to be uphill. Adjacent states in which the bugs do not meet these conditions are not accessible by the transition function in one turn (or possibly ever). Other effects are irreversible. The expenditure of energy by movement is irreversible, in the transition function cannot restore that amount of energy to a bug in a single turn. The acquisition of energy through the eating of a colony is, similarly, irreversible in a single turn. You cannot ever put the colony back.

For every state in the PSoup state space there is a set of adjacent states. This set of adjacent states is divided into four subsets by the transition function: (i) the current state; (ii) those that can be converted to the current state via one step of the transition function; (iii) those that can be produced from the current state via one step of the transition function, and (iv) all the rest.

The starting state is randomly determined within certain given parameters. However, the assignment (or selection) of the starting state immediately organizes the state space into a complex series of intertwining overlapping paths collectively called the radiation path. It is also conceivable that the selection of the start state would establish a subset of states which do not lie on the radiation path, and can therefore never be exhibited by the system, given this starting state.

Here are couple of interesting questions to ponder:
- Assume a most simple bowl of PSoup as defined above, a typical start state A, and the standard transition function F. Is the state A revisited somewhere downstream on its own radiation path? What conditions or assumptions would you need to make to assert that it is, or is not?

- Again, assume a most simple bowl of PSoup as defined above, a typical start state A, and the standard transition function F. Is the radiation path equal to the entire state-space, or is there some subset Z of the state space which is not on the radiation path? What conditions or assumptions would you need to make to answer this question?

Contingency

Fundamentally, the key concept behind contingency is this, the current state occupied by the bowl of PSoup is contingent upon the prior history of states through which the system has passed. Let's explore that idea.

We understand that the calculations carried out about above respecting adjacent 'next' states were very rough indeed. Nevertheless, for a thought experiment, let's use the number 10^7 as our estimate for all practical purposes of the number of adjacent next states for each and every state in the radiation path.

Let's ahead 80,000 seconds. Looking into the future, there is a radiation path, of which each thread has a defined orbit of 80,000 states through which the system might pass. Given that a generation of bugs will reproduce every 800 seconds, more or less, (recall that this is a Level 1 bowl of PSoup, RAT is set at 800 for all bugs, and RAT is a system-wide constant unchanging over time) we will then have the 100th generation of bugs in the bowl, roughly speaking. We know, from experience, that we will have a population of Twirlies in the bowl. Do we have left Twirlies or right Twirlies?

Suppose the bugs are right-handed. From the point of view of the bugs of the 100th generation, their history is fixed. At some time in their history something happened that headed them down the path towards right-handedness, and they and all of their progeny will ever after be right-handed. Their current state is contingent on all events in their history that brought them to this state. The future history of their progeny is, in all probability, limited to right-handedness. Past history has limited the options for the future.

At what point was it that something happened to move them towards right-handedness. The answer is probably like this. They started in a balanced state in which there was no hint of handedness. When the very first generation of survivors split and their daughters underwent mutation, in all probability, they tilted slightly to one side or the other. At this point, after the first round of mutations, there was probably a very slight bias towards one type of handedness over the other. The only type of circumstances in which this would not be the case is if all mutations involved only Palmiter genes 0 and 4 (F and B). For some time, over the next few generations, they probably wobbled slightly, crossing back over the summit of the potential hill that divided the two ways. Eventually, however, the system tilted far enough from equilibrium to start a long slow fall from the heights into the valley that lead them to the 100th generation, which was confirmed in its handedness.

In summary:
The present is contingent on the past.
The future is contingent on the present.
At some point in the prior history of the system, a small insignificant change, a single improbable (but very possible) outcome, may have been the determining event in the selection of two widely diverging paths.

Some evolutionary theorists believe that contingency has significant implications for the interpretation of past events. Others feel that it is of only mild relevance. PSoup shows that both views may be right, each in their own circumstances. For more about contingency, read 'PSoup and the role of chance'.

More About FSMs

For more information about PSoup as a Finite State Machine, try scenario 2F, and read the article ‘PSoup and Attractor Sets’.

PSoup and Attractor Sets

Note that this article is the second in a series. It is highly recommended that students read the article 'PSoup as a Finite State Machine' prior to reading this article. It is also recommended that students study the performance of Scenario 1F, which carries the same title.

So, PSoup can be viewed as a finite state machine. Each instance of a bowl of PSoup defines a state space. The state space has a finite number of discrete states which the system is able to exhibit, which can be organized as a multidimensional cuboid (i.e. a cube in more than three dimensions). Each dimension represents a parametric variable which describes the space, and each variable can have a finite number of discrete values. The total number of possible states is computed by simply counting the extent of each dimension, and multiplying the results: length x width x height x d4 x d5 x d6 … .

The starting state is selected, within parametric limits, randomly. The selection of a starting state (initial conditions) immediately organizes the state space into a radiation path, a complex of threads each of which is a series of states through which the bowl of PSoup might transition due to the repeated application of the transition function an arbitrarily large number of times. Transition along some dimensions may be reversible, but transition along other dimensions is not immediately reversible, so the radiation path will have (many) irreversible segments, forcing a direction of flow.

However, for the most simple bowl of PSoup, the total number of possible states is astounding, numbering roughly 10^1855 (that's 10 with 1854 more zeros added). The number of dimensions of the state space is estimated at over 760. The number of states which are adjacent to any given state is estimated to be roughly 10^334 (that's 10 with 334 more zeros added), and the total number of possible paths, or radiation threads, that a system might take through the state space would be so far beyond 'astronomical' that there is no word to describe the immensity.

How do we make sense of this indescribable complexity?

A 3D model of a state space

It is, of course, difficult to visualize anything with more than three dimensions. So let's temporarily simplify our 760 dimensional state space and consider it to have only three dimensions of importance. For the sake of generality, we will not specify which three. We will assume that the extent of each dimension is the same, so we have in effect a cube of, say 100 states by 100 states by 100 states. Call this the 3D model.

Consider the start state, for the sake of discussion, to be the state which occupies one of the eight corners of the cube. Now stand the cube on its side, such that the start state is at a top corner of the cube. Visualize each radiation thread as a thread of linked states which can be drawn through the cube. The radiation path, which is the aggregate of all such threads, spreads through the cube, branching regularly to become many threads, those branches merging with other branches to become single threads, and branching again. Some threads may cycle back upon themselves after many steps. All segments of all threads are irreversible defining a necessary direction of flow.

Note that a radiation thread in the 3D model is a two-dimensional thread with a three-dimensional extent.

Observations on a simple bowl of PSoup

Examine the behavior of the bowl of PSoup in Scenario 2F. As our most simple bowl of PSoup rolls downhill in its state space, we notice two ultimate outcomes. Either the denizens of the bowl become right Twirlies, or they become left Twirlies. At roughly 2 PSoup hours we see one genotype or the other starting to emerge and dominate. By 4 hours, the dominance is almost assured. We get one or the other, but not both. In addition, the number of bugs always approaches and stays near 7. Sometimes there will be 6 bugs,

or 5, or maybe 8, 9 or 10, but the number hovers around 7. This is in a bowl which was intentionally designed to be able to hold up to 80 bugs in a pinch, and certainly as many as 40. You can the history of the number of bugs by viewing the energy chart in which the number of bugs is plotted. So, we can observe that, while there is a vast number of possible states in this simple bowl's state space that are theoretically possible, the probability that most of them will be exhibited at any time is so minuscule as to be totally negligible.

Abusing the model

Let's use the 3D model to visualize what might be happening. First, we will abuse the model badly to build a concept, then we will rethink the concept and disabuse the model somewhat in the process. This is called 'working with the ideal', or cynics might call it 'fudging' the solution.

In the terms of our 3D model, visualize a two-dimensional manifold along which the system moves. You can think of the radiation paths as forming a pliable 2D sheet of plastic which is gently contoured, hanging from the corner where the starting state is located, and draping itself variably throughout the rest of the cube. The ball bearing (i.e. the system, which is the bowl of PSoup) rolls downhill along the surface of the plastic sheet (i.e. is transformed via the transition function to exhibit or actualize states on a radiation thread which runs along the surface of the sheet). When it hits a wrinkle, or a rut, it changes direction, bouncing and swaying as it follows the curved ruts in the plastic, moving ever downwards, until it lands in some pocket in the plastic and comes to rest.

In the plastic sheet there is an upside-down Y-shaped rut. The bowl starts at the end of the leg of the upside-down Y. A some time it comes to the branch and takes either the right branch (leading to right Twirlies) or it takes the left branch (leading to left Twirlies). Once it has taken one of these branches, it is not very likely that it will ever be able to jump out of this branch and go into the other. Not likely, but not impossible.

On it's trip to its ultimate resting position the ball bearing (the system) has, at times moved uphill due to momentum and prior history, but its general motion was downhill, driven by gravity, following either the right or left rut.

If you did not change the shape of the plastic, and you started a second ball bearing rolling downhill from the same starting position, or one very near to it, would it end up in the same place? The probability is high, but it would depend on what ruts exist in the plastic sheet, how deep they are, how close they are together. Why does the ball never travel all the way back up to the top, even though there are paths in the plastic sheet that lead that way? There are forces, pressures, (momentum, pliability and elasticity coefficients) which can conspire to resist the impact of gravity temporarily, but ultimately the ball will take a downward path driven by gravity.

If you run the ball bearing down the same slope many times, let's say 100 times, you will note that there are a very few number of practical outcomes. In terms of a state space, there are a small number of states towards which the system inevitably moves.

While the general slope of our imaginary plastic manifold is tilted, with a clear uphill/downhill distinction, there are ruts in the manifold such that some paths have a very high probability of transversal, and some paths have an extremely low probability of transversal. The rut is not formed by a single thread but by a large number of threads which lie close to each other, and in fact merge and branch and cut across each other regularly, and which all exhibit certain defining characteristics in all of their component states (e.g. contain left or right twirlies and have number of bugs within a defining range).

Disabusing the Model

In our abuse of the model, we relied heavily on an analogy with gravity, plastic, steel balls, etc. Let's remove that and rethink the model.

- Replace the concept of gravity as a driving force with the concept of high-probability as a preferred state.

- Replace the concept of momentum with the concept of irreversibility.
- Replace the concepts of ruts, valleys, and hills with low probability branches verses high probability branches.
- Replace the concept of movement of a ball bearing with the transition of a system from state to adjacent state moderated by the transition function.
- Replace the two-dimensional plastic sheet with a cobweb of radiation threads filling the entire cube.
- Replace the pliability and friction coefficients with the vagaries of a pseudo-random number generator.

That was easy!

Nevertheless, there is a limited number of outcomes, and the system will follow highly-probable paths through the cobweb of threads until one of those outcomes is achieved.

So, what are these ruts?

We can say that our simple bowl of soup seems to have two such ruts. One exhibits 7 left twirlies, and the other exhibits 7 right twirlies. In the early stages of this bowl of PSoup, there is one rut only. The ball bearing rolls evenly downhill. In fact the slope is so gentle at first that the ball tends to roll randomly in any old direction, as if it is starting on an almost flat plateau. However, the ball soon settles into a rut such that only bugs with strong forward genes exist, and all bugs with strong back-pointing genes are eliminated. These states are on less probable paths (uphill paths) and the probability of their eventual exhibition grows less and less as the ball rolls further and further downhill. This pattern (strong forward genes) emerges at about 1 PSoup hour (after the fifth generation of bugs is born). At about two hours, the rut splits into two ruts and the ball takes one rut or the other, but not both. The stronger the genotype becomes left-typed, the less likely it is that the ball will ever make the jump into the other rut). Both ruts run downhill, but the probability ridge between them grows ever higher, making the jump between ruts less and less likely.

Here is a curious phenomenon. As long as the total energy in the system is the same, 4800, it doesn't matter how many bugs you start with, the outcome is the same. Also, it doesn't matter whether you start with pristine bugs or other bug styles, the outcome is the same. In terms of our 3D model, the starting state is unimportant. Eventually the ball finds one of the standard ruts and rolls to its inevitable outcome.

Here is a challenge. Using Scenario 2F as a basis, can you tinker with the environment such that you have a 10x40 bowl of soup, no wrap, 4800 units of energy in total, which exhibits a stable genotype other than a left or right twirlie within 1 PSoup day?

Attractor Sets

BE VERY CLEAR, the following is speculative and attempts to apply recent developments in complexity science to PSoup.

Background - Chaos theory and the complexity sciences have developed concepts through the study of deterministic systems which exhibit chaotic behavior. PSoup is probabilistic (i.e. non-deterministic) so it is not at all clear that the concepts developed through the study of deterministic systems are applicable to a probabilistic system such as PSoup. Nevertheless, it appears that the same ideas can give us great insight into the behavior of PSoup, so we press ahead.

In a deterministic dynamic system:
- a state space is defined in the same way that we have defined the state space for PSoup, however, the extent of each dimension is continuous, and the number of states is infinite.
- for every state, there is only one possible outcome when the transition function is applied, that is, the outcome is 100% determined.
- two broad categories of motion through state space are possible: (a) non-recurring motion, in which the orbit of motion never enters the same state twice, and eventually exits from the state space (implying incomplete definition of the state space) and (b) recurring motion, in which one or more states are visited by the orbit one or more times.

- recurring motion is the only type of motion of interest.
- the set of all states which recur in an orbit are called the attractor set of the orbit.
- periodic motion is motion which returns to precisely the same state after a constant number of applications of the transition function. The cardinality of the attractor set is equal to the period of the motion.
- quasi-periodic motion is motion which, when launched from a state in the attractor set, returns after a precise period of time to a state arbitrarily close to the launch state. The orbit periodically follows a path extremely close to a previous period of the orbit, but not reproducing that previous orbit or crossing it or overlaying it at any time.
- chaotic motion is motion which, when launched from a state in the attractor set, follows an endless path through the attractor set but never exhibiting any type of periodicity for any prolonged interval of time.

We now borrow some concepts from Chaos Theory. In Chaos Theory, the solution space is usually continuous. In an FSM the state space (the solution space) is discrete with a countable finite number of possible states. Nevertheless, the concepts developed and used in the study of chaotic phenomena are of some help in illuminating the activities in a bowl of PSoup. The main concept we want to borrow is the idea of attractor sets.

Simply put, an attractor set is a pool or puddle, a low basin, in the state space.

We have three obvious attractor sets in our 'most simple bowl of PSoup':
- Set 'L' - One attractor set includes all those states that exhibit roughly 7 bugs with a left-twirlie genome.
- Set 'R' - One attractor set includes all those states that exhibit roughly 7 bugs with a right-twirlie genome.
- Set 'N' - There is at least one more attractor set; the set of all states in which there are no bugs.

Let's look at set 'N' first. If the system were ever to enter this state, it would become stuck in this single state forever. Bugs do not spontaneously generate. This attractor set is a set of one state, in which all cells have algae and 3200 units of energy are in the mud. The transition function moves the system from this state back into this state. There are many states which have no bugs, and all are in a basin which leads to this attractor set. For example, if only 50% of cells contain algae, the transition function would probabilistically fill algae into the cells, step by step, until all cells are filled and the attractor set is achieved. Within this basin (consisting of all states with zero bugs), all orbits lead towards set N. To be sure, the basin of N might even include many states in which all existing bugs are very sickly, but there is some chance that these states would show recovery and the orbits would eventually lead away from N. When the system loses its last bug, it crosses the event horizon of the set N and is inevitably drawn to N. That's why it is called an attractor set.

Here is the proposition - whatever the initial conditions, whatever the start state of our most simple bowl of PSoup, the orbit will lead eventually and inevitably to one of the three attractor sets. Sets 'L' and 'R' are the two sets of great interest to us from an evolutionary point of view. Let's now look at an orbit which does not immediately take us into the black hole of 'N'. Place the system into any acceptable starting state (4 healthy bugs). As the system moves from state to state under the action of the transition function it follows ruts to their inevitable end in a low basin. It is as if the system is attracted to this basin. Typically, the system will traverse across the state space until it reaches one of the two basin, and then it will fall into the basin, and then forever after circle around in the basin. Those states which are repeatedly exhibited by the system as it moves around in the region of the basin are said to be the attractor set.

With a View to Formalizing the Concept of Attractor Sets in PSoup

Before we proceed, lets understand a few differences between deterministic systems and PSoup:
- In a deterministic system, if a state ever occurs twice in an orbit, then the orbit is of necessity a periodic orbit. In PSoup, a system may occupy precisely the same state many times (10^7times) or more before it every repeats the same exit. Even then, there is little probability that it will follow a previously trod orbit for more than a few steps.
- In a deterministic system, chaotic motion is easily distinguishable from periodic or quasi-periodic motion. There are new tools and techniques being developed regularly which improve a mathematicians

ability to do this. In PSoup, it is proposed that all three types of motion are possible, but the tools for distinguishing the different types of motion would need to be applied with a moderating layer of statistical analysis, and the results would difficult to interpret. We would need to redefine the meaning of periodic, quasi-periodic and chaotic so that they apply to a finite and discrete state space and a probabilistic transition function. Tall order, but, fundamentally, motion having the nature of periodic, quasi-periodic and chaotic motion can be seen in PSoup scenarios.

But, having noted it, we shall ignore these difficulties, in the time-honored tradition of physical scientists, and carry on with our discussion.

The following definitions are specific to PSoup state spaces and are probably not in use in other literature about FSMs. For more formal and widely used definitions and terminology, students should consult appropriate sources. Each definition defines a subset of the PSoup state space.

A **radiation thread** T is a countably finite subset of the state space consisting of a string of adjacent states generated one by one by the application of the transition function, beginning at the starting state. A radiation thread delineates one possible outcome, one possible trajectory, of the infinitely repeated action of the transition function.

View a radiation thread as a thread, with a direction.

Note that, as a subset of the PSoup state space, T is necessarily countably finite. Also note that it is the product of an infinite number of applications of the transition function. Therefore, it MUST loop back upon itself. At some point the output of the transition function must be a state which has already previously been identified as an element of the thread T.

Suppose ‘S’ is a state which is an element of a radiation thread ‘T’. Now, consider the set ‘D’ of all adjacent states which can be produced from S through the application of the transition function ‘F’ once and once only. For discussion’s sake, let’s say there are 50,000 such adjacent states just one step downstream from S. The actual number cannot be determined without a detailed review of the current state. So we use 50,000 in this discussion. No state which contains any bugs is downstream from itself, as any such contained bugs would be one second older, which implies a different state. So let us assume that a bowl of PSoup in state S contains at least one bug. When the non-deterministic function F is applied to S, one and only one element of D is selected and realized. Each element of D has a probability of selection, and the average probability of selection is 1/50,000. Suppose we order these 50,000 elements of D in order of probability of selection, such that those with the highest probability of expression come first, and those with the lowest possibility of expression come last. If any have equal probability of selection, then they are lumped together as equals.

We can then choose any probability P (any percentage between 0% and 100% inclusively) and find the subset of D which, in aggregate, exhibits a probability greater than or equal to P that one of its elements will be selected upon application of the function F to the state S. We simply add the probability of each element of D, starting with the highest probability and working downwards, adding all ties all at once, until we have a number greater than P. Call this subset of D ‘H’, meaning the **P set of highest-probability outcomes** of the application of F(S). We so define H(S).

We can now define the set Q iteratively as the **P highest-probability radiation path** of the state S, consisting of (a) the P% set of highest-probability outcomes of S and (b) the P% set of highest-probability outcomes of any of its own elements. If a state U is an element of Q, H(U) is a subset of Q.

It stands to reason that a 1% HPRP (highest-probability radiation path) would be narrow and follow only the channel of the deepest rut. On the other hand, a 99% HPRP would overflow the banks of the widest valleys and submerge all but a few random high hills.

An **P attractor set** is the set of all states S which are contained in their own P% HPRP.

Note that S is NOT and element of H(S). S is an element of the P% HPRP of itself only there is a path within the P% HPRP which wraps back onto S.

Theorems - We speculate that:
- a bowl of PSoup will exhibit periodic behavior if, for some suitable P, the P-attractor set of the orbit forms a single toroid (n-dimensional torus) in state space, or consists of a single disc of adjacent states.
- a bowl of PSoup will exhibit quasi-periodic behavior if, for some suitable P, the P-attractor set of the orbit forms two or more toroids linked but having distinctly different periods.
- a bowl of PSoup will exhibit chaotic behavior if, for some suitable P, the P-attractor set of the orbit forms branching web of discrete but continuous paths.

By toroid, we mean a torus in more than three dimensions, i.e. a closed continuous path of variable width which has a direction and closes upon itself.
By 'suitable P' we mean a value of P which produces the appropriate shape of attractor set. You can think of P being the height of the water table in a broad deep valley filled with rolling hills:
- if P is too low, say, close to zero, then the ponds in the valley are all dry, and the P-attractor set would be the null set.
- if P is a little higher, then the lowest spot is under water, and only the most-probable transitions are in the attractor set.
- if P is raised a little higher, several ponds might appear, joined to the original pond by little valleys, and some less-probable paths would now be included in the attractor set.
- if P is raised a little higher, the passes between hills are filled, and the pond has become a lake with one or more islands, and many improbable paths are now included in the attractor set.
- if P is set a 1, or close to 1, then the entire space is now one pond, and the attractor set includes almost all states in the state space.

Keep in mind that it is perfectly possible, at any time, for a bowl of PSoup with a highly evolved group of denizens to follow an orbit through state space that ultimately produces a population of pristine bugs. Possible, but absolutely improbable. The definition of the P-attractor set solves this problem.

Continuing with this analogy, periodic (like) motion would appear in a single pond, or a lake around an island.
Quasi-periodic (like) motion would appear in a lake with two or three islands.
Chaotic (like) motion would appear in a lake with many islands.

An **attractor set** is a rut, a pool, a high-probability set of possible future states which will be realized again and again as the transition function is applied an infinite number of times.

Why talk about attractor sets?

For every natural system, it is possible to build a logical finite state machine that emulates the natural system to some identifiable degree of accuracy. Every such finite state machine has attractor sets. Attractor sets usually consist of immense collections of states which can be described implicitly by a few simple selection rules.

For example, one attractor set of our most simple bowl of PSoup can be described as the set of all states which:
- contain 7 left-handed twirlies, give or take a few;

The other attractor set of our most simple bowl of PSoup can be described as the set of all states which:
- contain 7 right-handed twirlies, give or take a few;

In spite of the immense complexity of the PSoup state space, we can say with certainty that roughly 50% of all runs will eventually exhibit one of the states belonging to this attractor set, and roughly 50% of the runs will eventually exhibit one of the states belonging to the other attractor set. The system is attracted towards

these sets of states, and has a high probability of exhibiting the defining characteristics of one set or the other.

These same concepts are useful in modeling weather patterns, biological systems, ecological systems, economic systems, chemical reactions, industrial processes, hydraulic turbulence, and more. Any natural or man-made system can be represented as a finite-state machine with attractor sets.

If we can describe attractor sets for probabilistic systems, and we can better understand them, we can more effectively use our models to make predictions about the real-world systems being modeled.

In Summary

How can we measure the complexity of a finite state machine:
- we can count the number of states in the state space, which is the product of extents for each dimension.
- we can estimate the number of radiation paths, which is determined by the complexity of the transition function.

How can we manage and effectively study the complexity of a finite state machine?
- identify attractor sets;
- parameterize the definition of the attractor sets, such that a few easily comprehended rules describe probable behaviors of the system under the action of the transition function;

For more information about PSoup as a Finite State Machine, read the article ‘PSoup and Chaos’.

PSoup and Chaos

Note that this article is the third in a series. It is highly recommended that student's read the articles
- 'PSoup as a Finite State Machine'; and
- 'PSoup and Attractor Sets'
prior to reading this article. It is also recommended that student's study the performance of Scenarios 1F and 2F, which carry the same titles. This article should be read in conjunction with viewing of Scenario 4F.

We could have equally entitled this paper, 'PSoup as an Infinite State Machine'. Starting at Level 2, PSoup allows the mutation of the base of all genes, as well as the exponent. In Level 1, a single gene could have 301 different possible strengths as the exponent, an integer, ranged in value from -150 up to +150. In Level 1, the base value in every gene was always 2. Starting in level 2, the base value could have any REAL value greater than 1.0 and less than 50. Also, the exponent could have any REAL value in its range, being no longer restricted to integer values.

PSoup is no longer a 'Finite state machine'. If you thought the number of potential states was large before (~10^1855), it has now risen to an uncountably infinite number.

How does this change PSoup?

The state space for PSoup no longer consists of discrete cells, but rather a disturbing combination of discrete and continuous dimensions. Algae either occupies a cell, or it doesn't. So, if we view the algae status of cell (0,0) as a dimension, it has an extent of 2, with two discrete values. On the other hand, where gene0 used to represent a single dimension with extent of 301 discrete values, it now represents two dimensions (base and exponent can vary through their respective ranges) each of which has an infinite continuum of possible values.

This means that:
- over time, as a PSoup scenario progresses, you will see gene strengths which are not powers of 2 but rather having any value.
- There will be less graininess in the states exhibited, and greater ability on the part of the PSoup system to respond to gentle evolutionary pressures, allowing for the definition of shallow ruts and creeks (figuratively speaking) where only flat desolate plains existed before in the angular features of the state space.
- High-probability radiation paths will be bounded by smoothly rolling hills rather than vertical cliffs, so escape from one such path into another will be more possible, allowing for a greater expression of diversity as scenarios unfold.

Does this detract from our insights into PSoup gained from FSMs, and into evolution?

No!

While the number of possible states has gone from a countable finite number to an infinite number, it does not invalidate the concepts of state spaces and attractor sets. In fact, since the common definition of attractor sets is used in application to continuous state spaces rather than discrete state spaces, we move one step closer to making orthodox Chaos Theory applicable to PSoup. Closer, but not there. PSoup is still probabilistic rather than deterministic.

What about Chaos?

In recent decades there has been an upsurge of interest in chaos, and chaotic systems. For many years, from Newton's day up until the latter part of the 20th century, chaotic system were simply viewed as uninteresting degenerate cases of non-chaotic systems. The most interesting case goes back to Sir Isaac Newton himself.

To formulate a description of gravity and its behavior he invented a new branch of mathematics called calculus or analysis. Using this new mathematics, he described how two masses interact under gravity, orbiting around each other, opening the understanding of celestial mechanics, and ushering in a new age of observation, research and scientific progress. Calculus has since been used in many branches of science to solve many difficult problems.

However, to Newton's great dismay, he was unable to use calculus to describe the motion of three masses. Two was the maximum he could handle. He worked on the so-called 3-body problem for some time before giving it up. For years many mathematicians and physicists of all stripes tried their hand at solving the 3-body problem, to no avail. Finally, in the early 20th century it was proven that the 3-body problem is insoluble using calculus or any other analytic algebraic techniques. We were faced with the stark fact that there is a simple natural system which cannot be described in a simple algebraic formulae.

It was eventually realized that three bodies interacting under the influence of gravity form a system which exhibits chaotic behavior, and it is a characteristic of chaotic systems that algebraic solutions do not exist.

Hydraulic turbulence was studied for many years by scientists and mathematicians before it was realized that turbulent systems also are systems that exhibit chaotic behavior.

Systems which exhibit chaotic behavior are fundamentally different, in some sense, from those which do not. Their predictability is different. Their susceptibility to description using algebraic techniques is different. And yet, many systems which are non-chaotic are simply a special simple case of the more general chaotic systems. A 2-body gravitational system is little different from a 3-body system. The ordered flow of a liquid is little different from the turbulent flow of the same liquid under pressure.

Why study chaos?

Chaos is all around us, and we don't recognize it. Complexity scientists now are beginning to realize that only a very small subset of physical phenomena are amenable to analysis using algebraic techniques. Our great success in physics, in particular, has been due to the initial step, 'assume an ideal' situation, after which an analytical solution is sought. Even then, in many cases, the equations can be drafted, but no solution can be found. Trying harder does not always work. As in the case of the three-body problem, the differential equations are easy to draft, but the solution does not exist.

When you turn on a tap, the flow of water is regular, when you turn it almost off, the drip, drip, drip of the tap is chaotic in nature, with an unpredictable pattern of drips.

When healthy, the human hearts beats regularly, when in fibrillation, it beats with a chaotic pattern.

Stock markets are predictable, like Newton's two bodies, in times of good fortune, but exhibit chaotic behavior in times of social crisis.

Soldiers on the battlefront, when faced with extreme danger, exhibit chaotic responses.

And the most famous example of all, weather cannot be predicted accurately more than five days in advance by the professional weatherman, but can be predicted a year in advance by almanac writers. Why? Mid term weather phenomena (i.e. those that are from 5 to 60 days in the future) exhibit chaotic behavior, long-term weather phenomena (i.e. those driven by the seasons) do not. Weather phenomena that develop within a five day period are chaotic, but they develop slowly enough that general predictions can be made, with probabilities of success, that are good for a few days only.

What's this got to do with PSoup?

Most of the scenarios in PSoup are specifically designed so as to NOT exhibit chaotic behavior. Those scenarios that are allowed to be chaotic, usually end quickly in the death of all denizens. Chaotic systems are difficult to predict, difficult to regulate, and difficult to understand.

However, evolutionary systems in the natural sphere exhibit chaotic behavior on a disturbingly regular basis. Some of the most strange and fascinating characteristics of natural flora and fauna can be attributed to the chaotic, turbulent, unimaginably complex ever-changing landscape of contours in the state space in which evolution plays. Try to imagine what turbulent eddies produced the giraffe, the elephant, the hummingbird, or the human mind.

It is therefore highly useful to understand and explore the implications of chaos for the PSoup system.

How is Chaos controlled?

Most systems which exhibit chaotic behavior do so when pushed past some threshold:
- a 2-body gravitational system is non-chaotic, a 3-body system is chaotic, but only under certain circumstances;
- a stream on a flat plain is non-chaotic; a stream running down a slope is chaotic;
- a person watching a sunset is non-chaotic; a person in extreme danger is chaotic;
- a bowl of PSoup with a regulated supply of energy is non-chaotic; a bowl with a non-regulated supply is chaotic, but only under certain circumstances.

In the natural sphere, energy does not have a cycle. Energy comes from the sun, and is dissipated into space. The flow of energy is unregulated. More to the point, when looking at a small locale, a small slice of our biosphere of dimensions say 100 miles by 100 miles by 200 feet, the inflow and dissipation of energy is totally irregular and unregulated.

In PSoup there is an energy cycle. There are two types of system, with respect to energy regulation:
- in a closed system, the total energy is fixed and does not change; if the energy level is high enough, the biomass can be very stable, and evolution proceeds slowly along predictable paths;
- in an open system, the total energy is not fixed, but the rate of inflow of energy is strictly regulated. The rate of dissipation is left to chance. The resources available to the bugs is determined by the amount stored in the mud and algae, more than the rate of inflow. The size of the store is contingent on past history, so there is a powerful feedback system affecting the resource levels. Evolution is buffeted by rapidly changing circumstances, and survival of the fittest is diluted by chance and circumstance. New evolutionary possibilities are opened up, even as old ones are lost.

In a closed system, PSoup ensures that there is always an appropriate supply of energy appearing as colonies of algae, and it is self-regulating. If too many bugs are born, then they hold more energy, there is less in the algae, and the competition for resources becomes intense. Only the most efficient bugs gather enough energy to survive. If many bugs die, the energy immediately shows up as algae, and the competition for resources is greatly relaxed, allowing all of the bugs, regardless of their efficiency, to build up their numbers. Here is a challenge. Design a closed system with 100% energy loading factor and 10 bugs at start that will suffer the death of all bugs. It cannot be done. In fact, in a closed system it is possible to load the energy levels so high that there is never any competition for resources, all bugs remain equally inefficient due to overcrowding, and no interesting evolution takes place. That is a system you can design.

In an open PSoup system, it is possible to have wild swings between periods of high competition, and periods of great abundance of resources. If the population of bugs is small, resources build up faster than they can be dissipated. The bugs feed easily; all reproduce; there is a population explosion. There is a huge hidden store of energy in the nutritive mud which seems inexhaustible. Dissipation rates start to exceed the acquisition rates, but the store is apparently endless, and the population continues to grow until the store is used up. Then the algae are all consumed. Then massive starvation sets in as all energy moves from algae to bugs. The population plummets, and only the most efficient, and the luckiest, survive. The implicit governor that controls these swings in a closed system has been removed, and the choice is no

longer one of the designer, but a random result of the transition function as it drives the system along a radiation path in its state space. In those terms, you can envision the high-probability corridors of the state space of an open energy system in PSoup as being wider and more shallow than the corridors of a closed system, with a greater likelihood that the system will vary widely from the norm, or jump out of the corridor altogether, and zoom off in some unexpected direction. Is this starting to sound like chaotic behavior? Does it sound like real-world biotic populations?

Can we Detect Chaos?

There are several ways we might be able to detect chaotic behavior in a system. Unfortunately, all of these techniques have been developed for deterministic systems. PSoup is probabilistic, and some the techniques carry a caution that determinism is assumed.

The one technique we have chosen to implement in PSoup is the 'Bifurcation Diagram'. For more information about how to use the bifurcation diagram features, see the 'Display Bifurcation Diagram' command under the 'Options Menu', or carefully read the HowTo screen that is associated with the command. Note, the command in the options menu is only available for scenarios which have an open energy system with a variable incoming energy level. Go to scenario 4F for a canned scenario, or use 'Tinker with the Environment' to open up the energy system. The 'Bifurcation Diagram' command with then be activated, and you will be able to enable data collection, and start.

What is a Bifurcation Diagram? -- If you have a non-linear deterministic dynamic system which sometimes displays chaotic behavior, it is possible to choose a control parameter C and an output parameter O such that a plot of O Vs C presents a characteristic bifurcation diagram. Typically, at the extreme range of C, O always has one value which varies slightly as C varies. This portion of the graph looks like a straight line, or a mildly curved horizontal line. However, eventually C hits a critical value, and the line abruptly splits (bifurcates) into two lines. The change is dramatic. One branch climbs quickly upwards and levels off as the lower branch drops quickly then levels off. About the time that they are both approaching level, each line bifurcates again and we have four lines, not two. Further bifurcations occur at rapidly increasing speed until it is not possible to distinguish the number of lines. Careful analysis shows that, as C approaches a critical value, the number of lines approaches infinity. Past that critical value, more complex bifurcation patterns interplay with each other.

The key points:
- control (C) and output (O) parameters well chosen;
- multiple readings of (O) taken at each value of C;
- for some part of the range of C, single value for O;
- for some part of the range of C, two values for O;
- for the rest of the range of C, more than two values for O.

How Do We Apply This to PSoup? -- We have three problems to solve.

(1) What type of bowl of PSoup is likely to exhibit chaotic behavior? We believe that a bowl of PSoup which has an open energy system and variable (seasonal) energy input rates with extreme changes between the seasons is most likely to display chaotic behavior. The seasonal energy influx creates periodic behavior. The fixed value of the RAT also creates a rhythm with a periodic quality, although it is a doubling of population every RAT seconds. The mismatch of the synchronization of these two periodic factors causes turbulent collapses of the population, with restarts the population growth at random moments within the seasonal cycle.

(2) What parameters can be chosen out of a possible 760 parameters or more? We choose the season length (SL) as the control variable. We choose number of bugs active as the output variable.

The number of bugs active during a very very long seasonal cycle, i.e. a seasonal cycle which extends over many generations of bugs, will modulate itself to follow the seasonal cycle. If we measure the number of bugs only at the low point of the seasonal cycle, then we should get the same number of bugs (give or take

a few) every cycle. Suppose RAT = 800 (the default). Then 8000 seconds would be ten generations. If the season length is 8000, then for ten generations the population will slowly rise, for twenty generations it will slowly fall, for twenty it will rise, and so on, twenty generations at a time. It will always fall to the same level (give or take a few) at the low point in the season. We say 'give or take a few' because this is a probabilistic system, and there is always room for variation due to statistical variance.

(3) How do we handle the variance due to the probabilistic nature of PSoup? We ignore it, trusting that patience will give us a credible result.

Another Thought Experiment

Imagine that we have a bowl of PSoup with 40 bugs. The bowl of PSoup has an open energy system with an energy input rate of 160 units per second (enough to support 40 bugs, with EPM of 4 units per bug per move). However the seasonal variation is at 80%, so in the height of summer the energy arrival rate is 289 units per second (able to support a population of 72 bugs) and at the low of winter the energy arrival rate is 31 units per second (able to support a population of 8 bugs). All mutation is turned off, and the gene pool is pure pristine bugs. This eliminates clouding of the results through unwanted adaptations.

Now, imagine that we do a run with SL set to 8000 PSoup seconds. At second 0 there are 40 bugs (give or take a few, we won't say that anymore, but understand that its there). At second 8000 there are 72 bugs. At second 24,000 there are 8 bugs. We take our first reading. 16,000 seconds later there are 72 bugs. 16,000 seconds there are 8 bugs again. We take our second reading. Every 32,000 seconds thereafter we take a reading and get a consistent 8 bugs. After ten readings are taken, we stop.

Now, imagine that we reduce the season length (SL) to 7,000 seconds. Take another 10 readings. Again we get 8 bugs at every reading.

Continue this process with SL decreased by 1,000 seconds each time. Eventually you will be taking a reading for a season length of 1,000 seconds. What happens now? We start with 40 bugs, but as those bugs are maturing and getting ready to fission, the rate of input has climbed to almost 250 units per second. There is a tremendous build-up of energy in the mud. The population soars to 80, and there is plenty for everyone. The next generation starts to mature, but at 1000 seconds the rate of energy arrival starts to drop. At 1600 seconds the second generation undergoes fission, and the population jumps to 160. Energy consumption is now very high. Energy input continues to slow. Bugs start to starve. At 3000 seconds energy arrival is very low, enough to support only 8 bugs, but we might have substantially more than that, or substantially less, depending on how far, and how fast, the population has collapsed. We no longer have the slow adjustment of the population to a gently changing climate. We have the catastrophic collapse of the population with a barely adequate incoming supply of new energy. Say the population dropped only to 8 bugs, and not 0 or 1 or 2. At 3800 seconds there will be 16 bugs. At 4600 seconds there will be 32 bugs. At 5000 seconds there will be high summer, with an energy input rate of 289 units per second, enough to support 72 bugs, but there will still only be 32 bugs chewing it up.

Stop now, and note a few things:
- the last time, at high summer, there were 80 bugs aged roughly 200 seconds.
- this time, at high summer, there are 32 bugs aged ?

The current age of this generation of bugs is determined by the conditions of the previous collapse. The current number of bugs is determined by the conditions of the previous collapse. A population collapse is a very messy thing. The next collapse is contingent on the timing of the previous collapse, and is also contingent on the number and health of the bugs that survived the previous collapse. Each collapse essentially resets the initial conditions for the next inevitable population collapse.

For this run with SL = 1,000 seconds, the readings will be ten numbers ranging from 0 to 200. We believe that these results are the results of a deterministic and chaotic process working behind the scenes, which is masked by conversion to discrete results (there are no fractions of a bug) and probabilistic processes.

Somewhere between a season length of 7,000 and a season length of 1,000 the sedate periodic behavior was replaced by chaotic behavior. By taking readings for every 1,000 seconds in that interval, we can identify the range in which the chaos starts. Say that we note the increase at 3,000 seconds. We can then do another series of runs with season lengths changing every 100 seconds. We can then zero in on the precise value of SL at which bifurcation takes place, and take this as evidence that chaos is present in this bowl of PSoup.

Poincaré Section

In the above process we have used a tool called a Poincaré section. It is a tool used to analyze recurring motion in a state space. Choose a planar section which cuts across the state space, and in fact cuts across the attractor set of the orbit you wish to study. The section must of course be of a dimension smaller than the space itself. Then, as the system follows the orbit, it will from time to time impact on the planar section. A periodic orbit will impact the section in precisely the same point, or set of points, each time. A quasi-periodic orbit will impact the planar section with the regularity of clockwork, but the point of impact will drift over time. Chaotic motion will impact the planar section in an irregular pattern that is predictable in neither time nor space, but in a visible pattern nonetheless.

As you vary the control variable through its range, the Poincaré section should indicate periodic motion of one period (one impact point), then of period two (two impact points), then of period four, rapidly increasing to an infinite number of impact points, at which value chaotic motion will appear. After that, as C continues to move through its range, the Poincaré section will vary widely between periodic motion of many periods, and chaotic motion. If you collect the results of all of these impacts on the Poincaré section, and plot them on a curve against C, you get the bifurcation diagram.

We used a one-dimensional Poincaré section, number of bugs.

The Effects of Chaotic Behavior on Speciation

Note that this article is the fourth in a series. It is highly recommended that student's read the articles
- 'PSoup as a Finite State Machine';
- 'PSoup and Attractor Sets'; and
- 'PSoup and Chaos'

prior to reading this article. It is also recommended that students study the performance of Scenarios 1F, 2F and 4F, which carry the same titles. This article should be read in conjunction with viewing Scenario 6D.

It is also recommended that students first read the article 'About Speciation'.

The interaction between the energy cycle in an open energy system and the population of bugs CAN exhibit chaotic behavior.

Suppose we have a seasonal cycle which is very long in comparison to the average life span of a bug. The life span of a bug is determined by RAT (reproductive age threshold). A generation of bugs is typically equal to, or just greater than, the average value of RAT. We then have a seasonal cycle of energy input which rises ever so gradually and falls again ever so gradually. The amount of energy in the bugs and in the nutritive mud also rises and falls with the seasons in a regular rhythm. However, if we have a seasonal cycle which is relatively short with respect to the life span of the bugs, we get a huge build-up of energy in the nutritive mud, followed by an immense population explosion, followed by starvation and a population crash. While the input of energy follows a regular cycle, the energy available to the bugs varies all over the scale, with immense ecological repercussions.

There is a critical seasonal length which is at the threshold between non-chaotic and chaotic behavior. Can you figure out what it is? Just approximately.

What are the implications for PSoup?

In a non-chaotic bowl of PSoup the evolutionary pressures are constant. Every bug is able to produce two offspring. There is room for only one. On average, one will survive and one will not. Those bugs that are the most fit, the best adapted to compete with their contemporaries, will probably survive. Each generation will evolve in the direction determined by the evolutionary pressures. In the terms of our High-Probability Radiation Paths (HPRPs), and using the imagery of our simple 3D model, the system is following the course of the HPRP between gently rolling hills as it moves towards a (distant) attractor set of ultimate states. The course of evolution is not exactly pre-determined (e.g. the 1F bowl of PSoup has two attractor sets with equal probability of realization). However, the course of evolution is predictable in a probabilistic sense.

In a chaotic bowl of PSoup the evolutionary pressures are waxing and waning in an unpredictable way. Now there is more than enough food for everyone to survive, and the population explodes in size. Then there is a severe shortage and only the cream of the crop, and the most fortunate, survive. Unpredictability creeps in two ways. For a period of time, no adaptations are negative, and many which would normally have little probability of survival are allowed to compound the error by piling mutation upon mutation without deleterious effect. It is as if the tide of water has risen above the banks of the probabilistic valley which held it in and has flooded out into the surrounding countryside, looking for new courses.

Then, the source of energy diminishes at a rate that causes a high rate of deaths. If some of those bugs which were washed over the banks have found a deeper valley (higher probability of survival), then they will out-compete their contemporaries when the food shortage gets severe. Or, we may have two genetically distinct species survive, each in its own niche (not likely in PSoup, happens all the time in nature).

So, we have unpredictability coming in through the temporary survival of non-competitive variations which may lead to the generation of competitive variations which are genetically several steps removed from the original stock.

The other source of unpredictability is simply serendipity. PSoup has a very small number of bugs. In nature, any population of less than 10,000 individuals is considered in danger. In PSoup, there is never any more than 500 bugs. When you have a precipitous decline in the number of bugs, the survivors are not necessarily the most fit. Random chance starts to play a greater role in the determination of who lives and who dies. The survivors may be less than the best representatives of their contemporary generation. Some great genes may have been lost forever. This effect is called **genetic drift**, and it is a source of evolutionary change in small populations under conditions of severe competition.

How would you recognize the effects of chaos?

A bowl of PSoup which is chaotic might exhibit some of the following traits:
- The level of energy in the nutritive mud would rise and fall precipitously in a cycle with varying unpredictable period.
- The number of bugs would rise and fall in a cycle of unpredictable period.
- Terrain effects (desert/oasis and paddies) or social barriers (diploid bugs of variant feeding habits) would tend to produce variant species, detectable via cluster analysis or study of the population profile charts.
- The number of effective alleles would rise and fall (assuming mutations are turned on), detectable via study of the allele charts.
- Produce a bifurcation diagram, using the supplied tool, and look for the telltale shape in the resulting graph (works some times).

PSoup as an Evolutionary Algorithm

PSoup is an example of the application of an Evolutionary Algorithm (EA), an iterative approach to finding the solution of complex problems. EAs have their roots in an approach called Monte Carlo Simulations, named, no doubt, for the casinos of Monte Carlo. It is useful to examine how PSoup is similar to, and different from, Monte Carlo simulations if we want to understand exactly how and why PSoup does what it does.

What is a Monte Carlo Simulation?

Monte Carlo simulations are special computer programs which are used to evaluate strategies in games of chance. Take blackjack for example. The game of blackjack has some simple rules which are easy to learn. The rules are carefully designed such that the house usually wins, but not always. The player wins just often enough to keep him (her) coming back. However, while the rules are simple, there are many strategies that a player may use to try to increase his (her) odds of winning, and the value of a strategy changes as the game unfolds. A casino would want to analyze the rules against every possible player strategy to be sure that the odds of winning cannot be consistently manipulated against the house.

There are two approaches for the casino. They can hire a mathematician to rigorously identify and evaluate the odds for every strategy, and keep the mathematician on hire for those occasions when a new player appears to be exhibiting a new strategy, or it can use a computer to simulate a player playing, say, 800 hands of blackjack using the new strategy, and, based on the law of large numbers, assume that the results are close to those a mathematician would produce.

When is a Monte Carlo Simulation Useful?

In essence, a Monte Carlo simulation has two main phases:
- in phase one an analyst models the internal processes of the game (or system) to be simulated. In all games of chance there is a randomizing mechanism (e.g. shuffled cards, rolled dice, spinner) that introduces a significant probabilistic factor into key decision points in the processes. The analyst must identify the basic probabilities of all of the various outcomes at each such probabilistic decision point (e.g. probability of rolling a 1 on a 6-sided dice is 1/6; probability of drawing a Qh in a deck of cards is 1/52, etc.) When the model is complete, and all of the % probabilities are in place
- the model is executed a large number of times with different reasonable starting conditions and a record is kept of the results of each run.

While the ideas used in Monte Carlo simulations first arose in the gaming arena, they are now commonly used in such diverse fields as physics (to evaluate the range of possible results from probabilistic processes) or economics (to predict possible outcomes of policy changes).

Many complex systems (e.g. economic systems) are probabilistic in nature and highly variable in outcome, given very similar starting conditions. In essence, for these systems, it is possible to have virtually identical starting conditions in two instances, and yet have different outcomes due to the probabilistic non-deterministic nature of the internal processes. What is usually observed, however, is that the outcomes follow a distribution pattern which is partially driven by starting conditions and partly driven by chance. If it is possible to analyze such a system and identify the key probabilistic decision points in the internal processes, and assign probabilities to the various outcomes in each case, then these decisions can be emulated by a random-number generator and the overall behavior of the system simulated.

In a game of chance, lets say a game of cards, the key ‘decision points’ that are probabilistic in nature occur when the dealer draws a card from the unused cards in the deck and deals them to a hand. The processes which are not probabilistic are (a) the rules of the game and (b) the strategy of the player. In a Monte Carlo simulation, the random-number generator is used to determine what card is dealt. Then the model plays the hand according to the rules of the game and the pre-determined strategy of the player. When hundreds of

'players' have been simulated in thousands of hands, the record of wins and losses indicates the value of the strategy.

In What Way is PSoup like a Monte Carlo Simulation?

Consider a simple 4 x 10 bowl of PSoup in which we start with eight similar but not identical bugs. In this bowl, the edges wrap around (so bugs do not bump their little heads on the edge if they get too close, and edges do not interfere with the bugs' innate search patterns), and there is a fertile oasis somewhere in the central part of the bowl. Suppose that the Energy Allocation Factor (EAF) is 1.0, and the Default Energy Level (DEL) is 800, so we know there are 1600 Nrg units in the bowl (40 cells times 40 Nrg units per algae colony) and 6400 Nrg units in the bugs (8 bugs times 800 Nrg units per bug) for a total of 8000 Nrg units. This is enough for 8 bugs to live and reproduce with (since the default Reproductive Energy Threshold (RET) is 1000 Nrg units). So, as the evolutionary lines develop within this bowl of PSoup, for every two daughters born, one will die without issue. Suppose that all Palmiter genes are identical for all bugs (i.e. all are pristine bugs) except for a small perturbation in each. In Bug 0 gene 0 has a strength of 2.0 instead of 1.0; in Bug 1 gene 1 has a strength of 2.0, etc., and in Bug 7 has gene 7 with a strength of 2.0.

Which bug has the better survival strategy encoded into its genes, to prepare it for this particular struggle for survival?

This would be a very difficult question to analyze mathematically. There are at least three aspects of this situation that will cause serious work for a mathematician:
- There are only 40 cells in this bowl of PSoup, and that gives us an average of only four empty cells per bug into which it can step without (a) bumping into another bug or (b) finding an empty cell just vacated by another bug, with the algae removed. They are almost certainly going to bump into each other and interfere with each other's effective search paths.
- Next, starting location may provide a significant advantage. A bug which finds itself in a corner in the beginning and pointed into the corner, will be at a disadvantage to those initially located in the middle of the bowl.
- and last, but certainly not least, each bug's search path is probabilistic in nature, not fixed, so on any given move a bug just might always choose the best path, through sheer dumb luck, in spite of what its genes say.

Nevertheless, it is not unreasonable to imagine that one of these bugs has a genetic advantage over the other bugs, all other things being equal. We know that in any bowl of PSoup there is a virtual certainty that one of the bugs' progeny will come to dominate the bowl, and the other lines of bugs will become extinct. Is it purely random chance that determines the winner, or do the genes play a significant role?

There may be many ways to address this question. We will look at (or, at least, discuss) three. The first approach is a Monte Carlo simulation. The third is the Evolutionary Algorithm (EA) approach. The second is somewhere in between.

(A) - A True Monte Carlo Simulation

We will start with the bowl of PSoup described above. Call it a standard bowl.

With this standard bowl:
- We will do 81 'runs' in which the bugs are pitted against each other.
- We will deal a 'hand' to each bug by randomly placing each of the eight bugs into the standard bowl of PSoup, thereby establishing a typical starting situation for the bowl. (Note that we DO NOT methodically go through every possible starting situation. We would then be forced to execute a very very large number of runs, and any partial set of runs would be strongly biased.)
- We will establish success criteria. At the end of 800 PSoup seconds, the bug(s) with the most energy win. The Reproductive Age Threshold (RAT) is set to 800 PSoup seconds, so the finish line in this little mini-competition is the moment when the most successful bug would first be eligible to undergo fission.

- We will tally the results of the 81 runs, and, the bug with the most wins is deemed to be the bug with the best survival strategy.

Note that in PSoup we have not actually implemented a true Monte Carlo simulation. If we had, we would be able to automatically run a scenario 81 times, stopping it each time at 13 minutes and 20 seconds (i.e. just before the bugs reproduce) to determine the winner and log the results. Treat this as a thought experiment.

(B) - An Almost Monte Carlo/Almost EA Simulation

Suppose we set the probability of mutation at 0%. Therefore, all daughters will be precise replicas of their parents. Suppose we also increase the size of the bowl of soup by a factor of three in each direction, so the bowl is now 12 by 30 instead of 4 by 10. Suppose we start with nine of the eight types of bug, for a total of 72 bugs. The total energy per bug is the same (1000 Nrg units per bug). The total area of the bowl is nine times bigger, there are nine times as many bugs, and we can effectively run nine runs in parallel. The most effective strategy in the smaller bowl should be just as superior in this bowl of PSoup. Call this the expanded bowl.

With this expanded bowl:
- We will do only nine 'runs' in which the bugs are pitted against each other. However, the nine runs will not be discrete. The offspring of the bugs which win in the first run will automatically and immediately be entered into the next run. The nine runs will be considered complete when the last bug in generation nine either dies or matures and undergoes fission.
- We will deal the first 'hand' to each bug by randomly placing each of the seventy-two (nine times eight) bugs into the expanded bowl of PSoup, thereby establishing a typical starting situation. (Note again that we DO NOT methodically go through every possible starting situation. We would then be forced to execute a very very large number of runs, and any partial set of runs would be strongly biased.)
- This time, instead of establishing success criteria, we will establish fitness criteria. At the age of 1600 PSoup seconds, the bug(s) which have not yet collected enough to fission are considered unfit to continue and are eliminated from the competition. The Death Age Threshold (DAT) is set to 1600 PSoup seconds, so the finish line in this little mini-competition is the moment when the unsuccessful bugs would die of old age.
- At the moment a mother bug produces two offspring, she effectively deals the new 'hand' to her two daughters, so there is no need for the dealer to become involved. Each daughter picks up the 'hand' dealt to her by her mother, and carries on in the competition. For her, the clock started counting the second she was born.
- When the ninth run is done, we enumerate the number of descendants of the original eight bugs that now exist in the 10th+ generation. The original bug with the most offspring is considered the winner, and is deemed to have been the bug with the best survival strategy.

Note that in PSoup we have not actually implemented the above scenario, although it would be an easy one for the student to set up. Again, treat this as a thought experiment.

It is instructive to examine how this approach is different from the true Monte Carlo simulation.
- First, we have replaced 'per heat' winning criteria with elimination criteria. To be sure, there are winners which receive intrinsic rewards in each run. The effective bugs don't starve, and they reproduce early. At the end of the first generation (i.e. when the last bug of generation one either dies or undergoes fission, the unidentified winners may have already undergone their second round of fission, and be starting run #3. By the time we finish run #9, it is conceivable that some bugs may be well into generation 11 or 12, having effectively lapped the competition.
- We have replaced the overall winning criteria (most heats won) with a different measure of success; most offspring alive.

Mathematical purists would argue, correctly, that this is not the same. In this approach a line of bugs which has a very slightly inferior strategy might nevertheless make some wildly fortuitous steps in the early runs,

putting it ahead in the race, giving it enough headway to win. One single run of nine generations of 72 bugs will be less trustworthy than 81 independent runs of 8 bugs.

(C) - An Evolutionary Algorithm (EA) Simulation

If we take the bowl of PSoup used in example (B) above, and make one change, we will have an evolutionary algorithm. We are now going to allow mutation of genes, and our process now becomes an EA.

Before we proceed with this discussion, we need to split a hair. 'Evolutionary Algorithm' (which in the literature is often written as the acronym EA) is a technical term used to describe a class of processes to solve difficult mathematical problems such as the (now cliché) traveling salesman problem. If we include mutation of genes, PSoup's processes can be classified as an evolutionary algorithm. However, in PSoup, as in nature, it is possible for evolutionary changes to occur in the absence of any mutation of genes. Evolutionary processes can be used to enhance the frequency of pre-existing genes in a large population, without recourse to any new mutations. So here is the hair to be split: (a) turning on the mutation of genes is necessary to convert the above example of PSoup into a true EA, in the technical meaning of the term; but (b) mutations need not be turned on in PSoup to demonstrate evolutionary processes.

Here is what happens now in the same 12 by 30 bowl of PSoup examined (in a thought experiment) in example (B) above. As each generation 1 bug matures and fissions, it produces two daughters which are no longer identical, genetically, to their mother. The losers of the competition in run #1are still eliminated, as before, but the winner's strategies are no longer represented in the competition in run #2. Those strategies, those sets of Palmiter genes, have been replaced by strategies which are similar, but not the same. When run #2 is completed, and run #3 starts, the strategies (sets of Palmiter genes) represented will again be similar, but not the same. When run #9 is completed, there will be clear winners, but what will it tell us about the original strategy. Will we know which of the eight original strategies is the best?

Again, the mathematical purist would say that the issue is greatly clouded by the history of the run. Each bug, in its generation, had to compete against a random assortment of misbegotten offspring of prior-generation bugs which, at best, can only be described as non-losers.

Unfortunately, we have proven nothing about the original eight strategies. However, we may have found one or several strategies which are substantially better than any of the original strategies. There will be branches of the family tree that clearly and unquestionably dominate the bowl, regardless of their lowly origins.

Summary of the Three

Monte Carlo simulations and Evolutionary Algorithms allow you to evaluate the effectiveness of different strategies in a game of chance, or in other probabilistic processes.

A large bowl of PSoup (in which mutations are turned off) has many similarities to a Monte Carlo simulation. However, the winning criteria (fitness) is more open than the winning criteria in a Monte Carlo simulation (success = most energy), and overall success is clearly marked by a dominantly large population.

A large bowl of PSoup (in which mutations are turned on) demonstrates an EA, an evolutionary algorithm.

A Monte Carlo simulation is able to tell you, relatively precisely, which pre-determined strategy is the best, and by how much.

An Evolutionary Algorithm is able to start with almost any old strategy as input, and build on that to find a better strategy, until, ultimately, it produces a best strategy.

Speculative and Experimental

The Gaeia Effect

This is also called the Gaeia Hypothesis, or the Gaeia Effect.

In ancient mythology, Gaeia was the name of the goddess, mother earth. She was the personification of the biosphere. Some worshippers believed that the biosphere was actually a living organism.

In 1976 James Lovelock and Lynn Marjulis put forward a fascinating hypothesis that the earth's biota shape and control the abiotic environment, creating the necessary conditions for life to continue to flourish. This hypothesis is called the Gaeia hypothesis.

"The Gaeia hypothesis states that the lower atmosphere of the earth is an integral, regulated, and necessary part of life itself. For hundreds of millions of years, life has controlled the temperature, the chemical composition, the oxidizing ability, and the acidity of the earth's atmosphere." (Marjulis, L and J. Lovelock, 1976, "Is Mars a Spaceship, Too?", Natural History, June/July pp 86-90).

The Gaeia hypothesis is disputed by many, and considered untestable, and therefore unscientific, by many critics. However, even if it continues to be found untestable, this does not prove it wrong. It remains an unproven hypothesis.

What are the implications for PSoup?

In PSoup, it is easy to create scenarios in which the biota (bugs), in the course of their struggle for survival, create and maintain the conditions that are optimal for their survival. In this implementation of PSoup this has not been extended to include control over the rules of chemistry of PSoup, but that extension can easily be added.

Gaeia in the DOE

The Desert Oasis Ecosystem (DOE) scenarios present the best example of a Gaeia effect at work. Jitterbugs, Twirlies, and plant-like bugs (Level 3) are well adapted to life in the oasis, but cannot live in the desert regions of this bowl of PSoup. On the other hand, Dodgers, Cruisers and Arrows are well adapted to life on the desert plains, and, while they do regularly foray into the oasis, their sojourn there is usually brief. Theirs is a life of hunting and gathering, not gardening.

In a mature bowl of DOE PSoup, there is typically a large population of Twirlies in the oasis, and a smattering of Cruisers racing across the desert. There is a curious balance between the Twirlies and the Cruisers. If either sub-species fails, the other sub-species suffers.

This is how it works. Algae is placed in the oasis first, and is re-directed out into the desert only if the randomly chosen location(s) in the oasis are already occupied with algae.

The Cruisers require that there be a steady stream of algae into the desert, so they require (a) consumption of the algae in the oasis at a sufficient rate to ensure it is re-cycled out to the desert, and (b) sufficient numbers of algae colonies in the oasis at any time to allow colonies to be diverted to the desert. The Twirlies that inhabit the oasis fill this function if they are numerous enough to pump enough energy through the system to overflow the oasis, but, not so numerous that they clear all algae out of the oasis, thereby starving the desert dwellers.

The Twirlies require that all colonies of algae remain close to the oasis, because they cannot travel. They need (a) a steady stream of algae arriving in the oasis, and (b) an efficient means of retrieval of that energy

that has escaped out into the desert. The Cruisers provide the retrieval mechanism. Every unit of energy that they collect is eventually sent to the oasis for attempted placement.

How does this Demonstrate a Gaeia Effect?

If one sub-species fails, and becomes extinct, the other suffers. Over time, via divergent evolution, the missing sub-species is replaced and balance is restored. We have a situation in which the biota affect the way the resources are distributed. If the balance between the types of biota goes too far out of balance, then the result is good for one and bad for the other, causing a change in distribution of resources, causing a return to balance. It would be an interesting exercise to try to set up an oscillation between the two sub-species such that the relative numbers of each rise and fall in opposite phase to each other.

Gaeia is Built In

In fact, a type of Gaeia effect is built into the default setting of most scenarios in PSoup. The PSoup ecosystem as a whole is by default a non-dissipative non-linear energy system. This means, in layman's language, that energy is never lost to friction or other types of dissipation. Every Nrg unit that is expended by the movement of the bugs or that is released upon the death of a bug is put back into the ecosystem via the ubiquitous nutritive mud. If the bugs become too numerous, there is less food per bug and some die, restoring the balance. If there are too few bugs, resources are plentiful and more survive. Since we have a non-dissipative closed energy system, it would be very rare indeed for any of the canned scenarios to have all bugs die.

However, we do have a few scenarios in which this type of subtle Gaeia effect is turned off. In the Easter Island Catastrophe, for example, we have a more realistic simulation of life as we know it. It is very possible for these bugs to all die out, and it took a great deal of care to craft a scenario in which this was not an inevitable result. Why is this? In this scenario we have a constant rate of introduction of energy, which is not dependent on the consumption rate of the bugs. The system is dissipative. If we have a surge in population (which occurs at the beginning) then energy is dissipated at an extremely high rate, much faster than it is coming in. If the total energy in the system is exhausted, and new energy is trickling in at a slow rate, then the bugs may not be sufficiently able to find enough food to survive.

In short, if the energy system is closed, it is always possible to set the energy level of the ecosystem high enough that a population of pristine bugs are guaranteed to survive, no matter how inefficient they are at gathering food. If the energy system is open, then the energy level of the system can vary widely, and it is possible that any population of bugs be so successful as to cause their own extinction, no matter how effective they are at finding food.

About the Cluster Analysis Function

Cluster analysis is an art, not a science. While there are many recognized techniques, none are based on fundamental statistical principles, and few are broadly applicable.

With that in mind, PSoup has implemented a home-grown approach to cluster analysis. The purpose is to provide the student with one more tool to slice and dice a population of bugs in a bowl of PSoup to better understand the evolutionary forces and the response of the population of bugs.

Simply put, PSoup plots the location of each bug in the population in a 33-dimensional space and looks for clusters of bugs within that space.

The 33 dimensions are C1 genes (9), C2 genes (6), C3 Fight capability types (9), and C4 Flight capability types (9). Capability types are used for the C3 and C4 chromosomes rather than the 26 genes, because we want to group the bugs with similar phenotypes, and because it substantially reduced the computations needed. It is both better and faster.

Unit of Distance

Distance between genes is measured in standard mutations. Two genes are said to be one standard mutation apart from each other if their strengths, when expressed as a power of 2, differ from each other in their exponent by exactly one. Suppose one gene has a strength of A, and a second gene has a strength of B. A can be written as 2 raised to the power of 'a' where a = Log(base2)A. Similarly B can be written as 2 raised to the power of 'b' where b = Log(base 2)B. The distance D between gene A and gene B is then the absolute value of the difference of the exponents, or D = |a-b|. Typically, if a gene starts with a strength of A, it must undergo at least D mutations to be transformed into a gene with a strength of B.

Distance Function

Distance between bugs is measured using the 'Manhattan Block' distance function. On each of the 33 dimensions, gene type by gene type, or capability type by capability type, the distance is measured. The distance between the bugs is then determined by simply adding up all of the distance values. Note that each distance between genes is positive, so the total distance is always positive. Note that we do not use the Cartesian distance formulae (root of the sum of the squares of the distances) as the process of mutation is very pedestrian in nature. It requires two mutations to move a bug a distance of two standard mutations, and the Manhattan Block distance function reflects this correctly.

Distance Table

PSoup starts the analysis with the construction of a distance table. Suppose, for the purposes of the discussion, that there are N bugs in the population being analyzed. PSoup constructs an N x N matrix of distances. Only one half is filled, as the other half is redundant. The distance from each bug to itself is 0 standard mutations, so N cells need not be filled. The total number of distances recorded in the table is therefore N(N-1)/2 distances. If there are two bugs in the population, there is only one distance recorded. If there are 500 bugs in the population, 124,750 distances are recorded. Each recorded distance is tagged with the two bugs associated with it.

Sorted Unique Distances List

Next PSoup sorts these distances into order and then copies the unique values into a shorter sorted list of distance records.

Reference Distances

In simple terms, a cluster of bugs is a collection of bugs which are near to each other. We determine nearness by a reference distance. If our reference distance is very short, say less than the shortest distance between any two bugs, then each and every bug is in a cluster of one, and we have N clusters. On the other hand, if our reference distance is very large, say, longer than the longest distance between any two bugs, then all bugs are in a single cluster. Neither of these clusterings affords us any new information. So the best reference distance must be somewhere between the shortest and longest distance recorded in the list of unique distances. Unfortunately, we can choose a reference distance anywhere within this range, and it will give us a clustering with anywhere from 1 to N clusters. How do we tease out of the data a 'best' clustering, with the greatest promise of some analytical insight into the structure of the population of bugs.

PSoup handles this decision by not deciding. It breaks the range of distances into twenty equal ranges and establishes 21 proposed reference distances. It then goes through the sorted list of unique distances and notes the actual distance in the list found to be closest to the proposed reference distance. That real distance is then substituted for the proposed reference distance. The result is an ordered list of 21 evenly spaced reference distances drawn from the entire list of unique reference distances. Each such distance is then tweaked slightly to make it just a little bit larger. Then we are certain than no pair of bugs happens to lie exactly one reference distance apart. The first and the last reference distance are close in value to the maximum and minimum distances between bugs in the population.

If there are fewer than 21 unique distances in the list of unique distances, then PSoup makes do with what is available, and an appropriate number of reference distances are extracted for use.

For Each Reference Distance

PSoup now goes into a loop and performs the following computations and manipulations for each reference distance.

Initial clustering - PSoup goes back to the sorted list of distance records, the one with N(N-1)/2 entries, in which each entry is tied back to a pair of bugs. Starting with the closest pair of bugs, PSoup checks to determine whether these bugs are closer than the reference distance. If yes, then they are placed into the same cluster. If no, they are placed into separate clusters. The next longer distance record is retrieved and the bugs processed. If they are within the reference distance of each other, then they can go into the same cluster. However, one of these identified bugs may already be assigned to a cluster (since each bug is tied to (N-1) records, so, if one of the bugs is already assigned to a cluster, both bugs are now put into that cluster. If both bugs are already assigned to different clusters, then they are left where they were.

The technique described above tends to perform well when applied to globular clusters, but performs poorly when applied to stringy clusters. It is believed that PSoup processes produce globular clusters. However this technique performs poorly when distributions are fractal in nature or logarithmic, which are potential problems. Some of the following techniques have been introduced to address this weakness.

Cluster Table

At this point all bugs have been inserted into one and only one cluster in a ragged cluster table. The rows in this table are clusters. The columns are bugs. Some clusters will have only two bugs. No clusters will have a single bug, due to the process used. Some clusters will have many bugs. The maximum number of clusters is N. The minimum number is 1.

Iterative testing

PSoup now starts a series of iterative tests. An iteration includes a test for each bug in the table. An iteration is complete when all bugs have been tested once. If, in the course of an iteration, it has been determined that some bug should be moved to another cluster, the results are noted, the move is made and another iteration is undertaken. If no changes came out of an iteration, the process is stopped, and PSoup notes the result and moves on to the next reference distance.

This is now each bug is tested, to determine whether it is placed in the best possible cluster. The bug is removed from the cluster in which it is located (presuming that it is not the only bug in that cluster) and then it is placed (a) into a cluster by itself, and (b) into each other existing cluster, in turn. Suppose there are C clusters. Then PSoup is testing this bug in C+1 clusters, one of which is a singleton cluster in which this bug is the only occupant. PSoup examines a 'Shrink Ratio' in each of the C+1 situations to determine which of these situations provides the best clustering. If the original cluster provides the best result, the bug is returned to that cluster and PSoup moves on to the next bug. If a different cluster provides the best result, the bug is placed into that cluster, PSoup notes that a change has occurred, then PSoup moves on to the next bug. Changes are made immediately.

The 'Shrink Ratio' (SR) is a ratio computed by dividing the adjusted mean distance between bugs by the unadjusted mean distance between bugs. The unadjusted mean distance is computed by adding all of the distances in the distance table and dividing by N(N-1)/2. The adjusted mean distance is computed by adding all of the adjusted distances between the bugs, and dividing by N(N-1)/2. The adjusted distance between two bugs is computed as follows:
- if the bugs are in the same cluster, the adjusted distance is simply the distance between them; but
- if the bugs are in different clusters, the adjusted distance is computed as A = sqrt(D * RefDistance)
where A is the adjusted distance, D is the unadjusted distance, and RefDistance is the reference distance.

Consider bug B and bugs W, X, Y and Z. Suppose bug W is a distance w from B, X is a distance x from B, Y is a distance y, and Z is a distance Z. Suppose that W and X and in the same cluster as B, and Y and Z are in a different cluster, two clusters in all. Finally, suppose that w and y are greater than the reference distance, and x and z are less than the reference distance.

Cluster	**Bugs**
0	B, W, X
1	Y, Z
2	Nil

We want to determine whether B should remain in the cluster where it is, or be move to cluster 1, or, be placed by itself in cluster 2.

The shrink ratio is the ratio of two sums of N(N-1)/2 terms. By moving B from cluster to cluster we change the value of the adjusted terms, but those terms not involving B do not change, so we can focus on the four terms w, x, y and z only.

When we put these four terms into the adjusting formula we get curious results:

w' = sqrt(w*RefDistance)	Since w is greater than RefDistance, w' is less than w.
x' = sqrt(x*RefDistance)	Since x is less than RefDistance, x' is greater than w.
y' = sqrt(y*RefDistance)	Since y is greater than RefDistance, y' is less than y.
z' = sqrt(z*RefDistance)	Since z is less than RefDistance, z' is less than z.

Depending on where we place B, the numerator of the shrink ratio will contain four terms as follows:

Cluster	**Terms**
0	w, x, y', z'
1	w', x', y, z
2	w', x', y', z'

The sum of these terms which is least determines the best location for bug B. Theoretically, any could be the best, it just depends on how much bigger or smaller than the reference distance each of w, x, y and z, are, and how they play against each other.

For this reference distance, the arrangement of bugs into clusters which provides the lowest value of the shrink ratio is deemed to be the best clustering.

Overall, this approach has some drawbacks. The testing of a single bug at a time leaves room for overlooking a situation where two bugs are in the wrong cluster, and moving a single bug does not reduce the value of the shrink ratio, but moving two would. In other words, this technique can effectively find a local minimum for the shrink ratio, but may not find a minimum or downward gradient which is as close as two bugs away. However, if we note that this testing technique is applied to a table that is expected to be globular and largely correct as given, this approach should work.

When the shrink ratio has achieved a minimum and further testing results in no more moves of bugs from cluster to cluster, PSoup goes on to the next reference distance.

Ranking the Reference Distances by Results

We now have (up to) 21 reference distances with associated shrink ratios indicating the best clustering results possible from each. How does PSoup decide which is the best reference distance?

You will note that the best shrink ratios achieved are associated with the shortest reference distances, and the worst (all at a value of 1.00) are associated with the longest reference distances. Basically, there is a distance, equal to about 1/2 of the diameter of the population, at which all bugs are deemed to be in the same cluster. So, while shrink ratios are useful for determining when the iterated testing for a reference distance should stop, they are not very useful for identifying the best reference distance. Another factor is introduced to handle this.

Call the reference distance R. If R is equal to the minimum distance between bugs, we expect to have N clusters (assuming no bugs are identical). If R is equal to the maximum distance between bugs, we expect to have 1 cluster. Being simple-minded about it, if R is halfway between the minimum and maximum values, can we expect (N-1)/2 clusters? Can we determine an expected number of clusters for each and every value of R in the range? Sure, simply by imputing a straight-line slope. We can then determine, for any given value of R, whether the clustering algorithm is performing better than expected, or worse than expected. We can compute the Cluster Factor (CF) as the ratio of the actual clusters found divided by the expected clusters to determine the performance for any given value of R. PSoup does this.

To make a final determination, PSoup multiplies the Shrink Ratio (SR) by the Cluster Factor (CF) to produce the Cluster Ration (CR). The reference distance which gives the lowest value of CR is deemed to be the best.

How do We Interpret This?

Clusters are a figment of the human imagination. You will not find a good definition of a cluster if your search the literature. Or rather, you will find many different definitions. A cluster is a pattern of distribution. Our minds search for patterns and often find them whether or not they exist. PSoup, similarly, searches for patterns, finds them and presents them.

A low value of SR for a reference distance would indicate that PSoup has produced clusters that represent actual clusters in the population. An SR value of 1.0 indicates failure.

A low value of CR for a reference distance would indicate that PSoup has managed to find fewer such clusters than expected on a straight-line projection, and this is a good thing. If the bugs are truly randomly distributed with no real clusters, you would expect the algorithm to find new clusters at each level as the reference distance shrank. A non-random distribution, i.e. a cluster in which the bugs are more densely packed, would appear as a reduced number of clusters, with more bugs per cluster.

PSoup automatically does a complete cluster analysis as per the above routines, and presents the results to the student. However, PSoup offers the student the ability to explore the results of cluster analysis for all reference distances used, and to explore the intermediate results of each iteration of the clustering algorithm. The genetic profiles of each cluster can be displayed for review by the student.

Cluster Analysis and Speciation

Cluster analysis has been included in the PSoup array of tools to enable a student to better determine whether speciation has occurred in PSoup. The definition of a species in PSoup is left up to the denizens of the bowl. Cluster analysis can be used to sort cruisers from Twirlies and jitterbugs, to sort bushes from potatoes, to sort carnivores from herbivores, to sort predators from prey. In short, cluster analysis can be used to study genotypes and phenotypes. In some instances competing phenotypes may diverge sufficiently within a bowl of PSoup that they could legitimately be called separate species. In those instances, PSoup will be able to pick out two clear clusters with characteristic genetic profiles.

PSoup and Pairing Squares

This article offers a brief review of Mendelian genetics and the analytic tool called a 'Pairing square'. It is intended that this article be read in conjunction with 'Scenario 5C - Mendel's Peas'. The article includes a worked example of a Punnett Square for Mendel's Peas, as found in Scenario 5C, and shows how pairing squares relate to Punnett Squares. Pairing Squares are unique to PSoup, and may not have applicability outside of the framework of PSoup. They are nevertheless very useful in the analysis of the dynamics of evolution of dominant alleles of critical genes.

Background

In the mid 1800s an Austrian monk by the name of Gregor Mendel undertook a series of genetic experiments using ordinary garden pea plants in the gardens at his monastery.

His peas had many distinguishing characteristics that could be used to sort them into groups. Seven of these characteristics were of particular interest to him because, for each characteristic, there were two clearly different presentations. The seven noteworthy characteristics were:

Characteristic	**Presentations**
Seed cover	round vs. wrinkled
Seed color	yellow vs. green
Flower color	red vs. white
Pod shape	inflated vs. constricted
Pod color	yellow vs. green
Flower location	axial vs. terminal
Stem length	short vs. long

In Mendel's experiments he methodically crossed plants of various types, as described above and noted the results. At the time, one contemporary theory on inheritance held that the results of such crosses would be a blended trait. For example, if you crossed a tall pea plant with a short pea plant, you should get a mid-sized pea plant. Or, if you crossed a red flowered pea plant with a white-flowered pea plant, you should get a plant with pink flowers. But it was clear from observation that the predicted intermediate variants did not exist.

Mendel did notice a very curious phenomenon. If you chose any one of these characteristics and crossed plants with the two different aspects of the same trait, the first generation of offspring showed only one trait. If you then crossed the offspring of these plants, all of which now present the same trait, the second generation offspring will present both traits again. In other words, one of the two traits was not presented in the first generation of offspring, but was presented in the second generation. What's more, the trait which disappeared in the first generation was presented in about 25% of the second generation.

The results could be quantified. Suppose you have 50 red-flowered pea plants and 50 white-flowered pea plants. Cross the red with the white. The first generation of offspring will be all red. Divide these arbitrarily into two groups and cross the plants in one group with the plants in the other group. The second generation offspring will have some red-flowered plants and some white-flowered plants. In fact, the plants come out in a ratio close to 3 red to 1 white, or 75% red and 25% white. This same phenomenon occurred whichever characteristic was studied.

Mendel proposed that each pea plant possessed two discrete hereditary factors for each of these seven characteristics, and that when a gamete was formed, these factors separated so that each gamete possessed only one of the two hereditary factors. A fertilized seed would therefore have two such factors again, one from the male parent plant, and one from the female parent plant. He further proposed that one of the two hereditary factors is dominant, suppressing the other factor when both exist in the same plant. So a plant

having both a red-flowered factor and a white-flowered factor would present with red flowers, the white factor having been suppressed.

A Punnett Square

In modern biology there is a standard tool or technique used to describe this result; the tool is called the Punnett Square. Lets run through the experiment with red and white-flowered peas one more time, but this time using Punnett Squares to explain what is happening.

In the zeroth generation, we start with 50 red-flowered pea plants, and 50 white-flowered pea plants. We assume that each is a 'pure' strain, i.e. that Mendel had crossed red with red and white with white for several generations prior to this to produce these pure strains. We then know that each red-flowered pea plant in the zeroth generation had two identical alleles, R and R, to control the color trait. We also then know that each white-flowered pea plant had two identical alleles, W and W, to control the color trait. We can represent the critical section of the pea plant's genome as either RR for the red-flowered plants, or WW for the white-flowered plants.

To produce the first generation of offspring, Mendel crossed the pure red with the pure white. For each union of ovum with pollen, a random allele was chosen from the red-flowered plant's genes, and a random allele was chosen from the white-flowered plant's genes. 50% of the time, you get the first R, and 50% of the time you get the second R. The same for the W.

We can then draw a Punnett Square to predict the population of pairings in the offspring, based on a known set of pairings in the parent generation.

Punnett Square showing the first generation offspring of two pure strains.

	W	**W**
R	RW	RW
R	RW	RW

The two different alleles of the pure red zeroth population are shown as the Rs in the left-most column. The two different alleles of the pure white zeroth population are in the top row. There are in total four different ways that you can select one allele from one parent and one from the other. The four spots in the middle show what pairing of alleles is found in the offspring for that selection of alleles. In the first generation, they are all the same. (Note that order is NOT important.) Since R is the dominant allele, all pea plants appear to be red-flowered, but all harbor the recessive gene W.

Next, Mendel split the 1st generation offspring into two equal parts and crossed group one with group two. Suppose the used the pollen from group 2 to fertilize the flowers of group 1. Group 2 was not fertilized. It does not matter at all how he split them up, as they are all identical. However, now, each ovum has either an R or a W allele, so if makes a difference which is chosen. 50% of the time, the ovum will have an R allele, and 50% of the time it will have a W allele. Similarly for the pollen. Our Punnett Square for the second generation now looks like this.

Punnett Square showing the second generation offspring of two pure strains.

	R	**W**
R	RR	RW
W	RW	WW

Note that 25% of the offspring are pure red-flowered, 50% are hybrid red/white in which red is dominant, and 25% are pure white-flowered.

Percentage representation of second generation offspring of two pure strains. This assumes that the zeroth generation was 50% red and 50% white.

	R	**W**
R	25%	25%
W	25%	25%

We now call Mendel's 'hereditary factors' genes, and we know that they are contained in DNA in matching pairs of chromosomes. During meiosis, that is, during the formation of a pair of gametes, the two chromosomes separate. Each gamete therefore contains a single chromosome of the pair. When two gametes are united to form a fertilized seed, the full complement of pairs of chromosomes is restored.

Terminology

GENES - Organisms have genes encoded into strands of DNA located in the nuclei of their cells. The word 'gene' can actually be used with at least three different meanings. Within this documentation there has been an attempt made to be clear about which meaning is intended each time it is used.

First, it can mean a specific strand of DNA or a collection of strands of DNA which is capable of performing a certain function. The word is rarely used with this level of specificity, but in some places it has this meaning. In PSoup, the concept of gene is encoded in a C++ object or structure called a simple gene, and sometimes these structures are combined into pairs, each pair called a compound gene, or hyper gene. In this sense, a bug in PSoup can have anywhere from 8 genes to 446 genes, or encodings.

Second, it can mean a function or purpose. It is said that the human genome has roughly 30,000 genes. This does not mean 30,000 physical strands of DNA. This means that scientists have identified roughly 30,000 types of coding structures in human DNA which appear to be functional genes. Every living cell of every human being on earth has the same 30,000 genes. In fact, since we share 98% of our DNA code with chimpanzees, we can find roughly 29,000 of these same genes in every chimpanzee on earth. Gene is therefor used in a collective sense, and this is its usual meaning. When scientists say they have discovered a gene, they mean they have discovered a type of structure in the DNA which, due to its content and relative location in the DNA, performs a certain function of interest to the scientists. In this sense, a bug can have from 8 genes to 223 genes, or types of encodings.

Third, it can mean a specific variant of one of the 30,000 genes. When we say that a person carries the sickle cell anemia gene, we mean that that person has a variant of one of the 30,000 which has the effect of causing this disease. That person will have one or two instances of this sickle cell anemia gene (depending on whether he is homozygous or heterozygous in the gene) in every cell of his body. In scientific language, these are usually called alleles of the gene. There is a sickle cell anemia allele, and there are other alleles of the same gene which do not cause the disease. A bug in PSoup can own at most two alleles of each gene, those two alleles selected from a list of possible alleles which is practically uncountable.

TYPES OF CHROMOSOMES - The strands of DNA which hold the genes are typically divided into groups or pieces which are called chromosomes. A chromosome is a single group of genes which are physically linked as part of a single strand of DNA. Within the nucleus of each cell of the organism there is an identical set of chromosomes, that is, the type and number of chromosomes is the same in every cell of the organism. The type of a chromosome is determined by the coding of the DNA in it, that is, by the type, number and order of the genes (and other DNA) that is in the chromosome's DNA strand. Speaking simplistically, if two chromosomes have the same number and type of genes in the same order and the strands of DNA are therefore more or less the same, then the two chromosomes are the same type.

HAPLOID - The nuclei of the cells of most organisms typically contain more than one chromosome. If all of these chromosomes are of a different type, the organism is said to be haploid.

DIPLOID - If the nuclei of the cells of an organism contain two similar sets of chromosomes, that is if all chromosomes exist in pairs, two of each type, then the organism is said to be diploid. For example, the human genome is housed in 38 chromosomes comprised of 19 distinctive pairs. Pairs of chromosomes can be identified because they have the same number and type of genes located in the same order. This does

not mean however that they are identical. Since one chromosome is inherited from one set of grandparents via the mother, and the other chromosome is inherited from the other set of grandparents via the father, each type of gene is potentially represented by a different variant of the gene. The human genome is diploid.

PSOUP'S IMPLEMENTATION OF HAPLOID AND DIPLOID

In PSoup all bugs found in Levels 1 through 4 are haploid, as per the following table:

Level	Chromosome types	Chromosome counts
1	C1	1 chromosome only.
2	C1, C2	1 each, total of 2.
3	C1, C2	1 each, total of 2.
4	C1, C2, C3, C4	1 each, total of 4

In PSoup, for Levels five and six, a bug may be either haploid or diploid. Haploid bugs reproduce via fission which may be first preceded by crossing-over of genes. Diploid bugs reproduce by sexual reproduction (or parthenogenesis, if mating has not occurred for a mature bug). At these levels, haploid bugs have eight chromosomes, to wit, one each of C1, C2, C3 and C4 chromosomes. The C3 and C4 chromosomes usually differ in the type of genes contained, but this is not necessarily the case. At these levels, diploid bugs have eight chromosomes, to wit, two each of C1, C2, C3 and C4 chromosomes in four pairs.

In natural DNA all genes (first meaning of the word) are coded via four organic compounds which are strung in sequence.

In PSoup, all genes are coded as a strength and a dominance type. These two values determine the gene intrinsically. However, the genes derive their impact on the phenotype of the bug via the chromosome which they occupy and the location within the chromosome. For example, a gene (first meaning of the word) with a strength of 1 and a dominance type of 999 would be active, and possibly even important, if located in the first position of a C1 chromosome (i.e. if it was Palmiter gene 0). The same gene values would be immediately, horribly, terminally fatal if located in the first position of a C2 chromosome (i.e. as the DAT or Death Age Threshold). All bugs having these gene values in this position would die at the age of one second. The same gene values located in the first position of a C3 chromosome would enable an 'A' Fight gene, which would provide the bug with an incomplete hearing and tasting organ. Other genes would have to be developed to enable these useful abilities. This gene location would however increase the bug's complexity by one. The point here is, the strength and the dominance type are important, but the location (type of chromosome and position within the chromosome) play the greatest role in determining the effect a gene will have.

ALLELE - In the natural world, when a single characteristic such as seed cover texture presents two different forms such as smooth and wrinkled, then we say that the gene controlling this characteristic has two different alleles. In PSoup each gene has a strength, and a dominance type, and these are used to identify different alleles of the gene. For example, when comparing two different instances of the C1 chromosome, if the strength of the gene in position 0 differs, then there are two different alleles of gene 0 represented in these two chromosomes.

PSoup offers several tools to analyze the alleles within a population of bugs. In most of these, two instances of a gene are considered to be the same allele if:

(a) they occupy the same location in chromosomes of the same type; and
(b) they have the same strength.

In the analysis using pairing squares, there is an additional determinant. To be considered the same allele:

(c) two instances of a gene must also have the same dominance type.

DOMINANCE TYPE - In the natural world, two alleles of a gene which occur in a single individual interact by becoming either dominant or recessive. For example, in the crossed pea plants described above which had both red-flowered and white-flowered hereditary factors (the first generation pea plants), the red-flowered factor was dominant and the white-flowered factor was recessive. So, all of the first generation pea plants appeared as red, even though they carried this hidden recessive factor. PSoup uses a generalization of this idea. Each gene in PSoup has a dominance type which is a number between 0 and 999. For each gene in a bug's genome, PSoup handles dominance as follows:
- if the strength of corresponding genes is the same, dominance is irrelevant;
- if the strength differs, the gene with the higher dominance type is used, the other is ignored; and
- if the strength differs but the dominance type is the same, PSoup randomly selects one of the pair of genes and increments the dominance type by one, then uses the dominant allele.

Note that this implementation of dominance type can be used to duplicate Mendel's results, but may vary from the phenomenon of gene dominance as found in the natural world under some circumstances.

ALLELE COUNT - In PSoup it is possible for a gene to have very many alleles. Every gene in PSoup has a strength and every different value for strength is considered a different allele. There is a huge range of possible values. In practice, the maximum number of bugs allowed in a bowl of PSoup is 500. If there are two different alleles per gene per bug, then a single gene (say gene 0 in a C1 chromosome) can have at most 1000 alleles represented in the population of bugs within the bowl of PSoup at any given time. Of course, since the majority of genes are non-mutated copies of pre-existing genes, the actual number of alleles per gene is almost certainly less than 1000.

Tools Available

The 'Population profile charts' provide you with a pictorial view of the distribution of alleles for the genes in the current population only. Each bar chart shows the distribution of alleles over the range of possible values. Using these charts a user can identify possible speciation by noting clustering of values of alleles around two distinct centers. There are 85 charts covering 67 genes and 18 capabilities in ten panels as follows:
- distribution of alleles for nine C1 genes (one panel);
- distribution of alleles for six C2 genes (one panel);
- distribution of alleles for twenty-six C3 genes (three panels);
- distribution of alleles for twenty-six C4 genes (three panels);
- distribution of nine capabilities; both fight and flight; eighteen in all (two panels).

The 'Allele Counts Chart' provides you with a count of the number of alleles in the population for each of the 67 types of genes. Since there are at most 500 bugs in the population, the largest count possible is 1,000 per gene, or 67,000 for all genes. This chart collects data over a period of time and records the results for up to 13 generations of bugs. This chart let's a user see the total depth of the gene pool for each gene, and the overall trend towards more numerous or fewer alleles, by gene.

The 'Allele Values Chart' provides you with an historic record of the average allele for each gene. For each of the 67 genes, the average value of all extant alleles is recorded from the 'Average Bug' profile. Data for 13 generations are kept. The user can reproduce and examine the Average Bug profile for any generation, and can review trend lines. This chart allows the user to identify any overall trends in evolution, but masks any speciation.

The 'Cluster Analysis' feature provides you with the ability to segregate any population of bugs into groups representing potential sub-species. It is understood that this tool is similar to a bat, when a scalpel is required, but no scalpel is available.

The 'pairing square' chart provides you with an analysis of the successful combinations of genes (maternal and paternal) for each of the 15 types of C1 and C2 genes, for up to 13 generations. It allows you to explore the competition between dominant and recessive alleles, by gene type, and chart the history of any allele over the course of time.

Why Does PSoup not Implement a Punnett Square?

Given a known pure strain of organisms, and controlled fertilization, the Punnett Square is a useful tool to predict the characteristics of the next generation. As such, it is of great use in biology labs in which new plant strains are developed. However, in a bowl of PSoup, as in the open fields of the natural world, the population of organisms is a higgledy-piggledy lot of plants with often far more than two alleles to be concerned about, and no control over who fertilizes whom. The implementation and presentation of a Punnett Square then becomes very complex, and difficult to explain in a simple report. We have a detailed worked example of a Punnett Square below, for those who wish to understand it. However, PSoup implements only the collection of allele counts which would be used as input to a Punnett Square exercise, or would be used to verify the predictions of a Punnett Square exercise.

PSoup has implemented a matrix analysis tool called a pairing square, which has some distinct differences from the Punnett Square, but which is easier to produce and more generally useful in the hurly-burly of a bowl of PSoup.

Here are three Pairing Squares for the first three generations for which Punnett Squares were drawn above.

Pairing Square for the zeroth generation of Pea plants, showing 50% of the population having an RR pairing, and 50% having a WW pairing.

	R	**W**
R	50%	0%
W	0%	50%

Pairing Square for the first generation of Pea plants, showing 50% of the population having an RW pairing, and 50% having a WR pairing. Note that order has some meaning in a pairing square.

	R	**W**
R	0%	50%
W	50%	0%

Pairing Square for the second generation of Pea plants, showing 25% of the population having each of the four types of pairings. Note that order has some meaning in a pairing square.

	R	**W**
R	25%	25%
W	25%	25%

What are the Differences between a Punnett Square and a Pairing Square?

Punnett Square	**Pairing Square**
Two generations per square.	Only one generation per square.
Dynamic projection of population.	Static picture of population's allele pairings.
Cross of two known strains.	Tabulation of all strains in a population.
Implied control on fertilization.	Fertilization technique irrelevant.
Nx2x2, N=# of strains in pop.	NxN; N=# of alleles in pop.

Note, Punnett Squares are widely used. Pairing Squares are unique to PSoup. However, pairing squares are the most useful in PSoup.

Pairing Square - Generation #0

	R	**W**
R	50%	0%
W	0%	50%

50% of population has RR; 50% has WW.

Punnett Square - 0-1 Transition

	W	**W**
R	RW	RW
R	RW	RW

When you cross the RR strain of generation 0 with the WW strain of generation 0, all outcomes for generation 1 have RW or WR. Punnett squares take no notice of order.

Pairing Square - Generation #1

	R	**W**
R	0%	50%
W	50%	0%

In the first generation, all organisms have either RW (50%) or WR (50%). Pairing Squares collect and present information about the source of the alleles (Maternal or Paternal). This is not possible in biology, but it is possible and interesting in PSoup.

What does the 'Pairing Square' Chart Offer?

Data across generations - When data collection for the Pairing Square is enabled, data is collected for each generation. The amount of time considered to be one generation is the current value of RAT (Reproduction Age Threshold) as found in the 'Average Bug Genetic Profile' at the time that the reading is made. This time, recorded in seconds, is rounded up to the next nearest whole minute, and the next reading is taken after that many minutes have passed. Data for up to 13 generations is recorded. When a further reading is taken, the oldest reading is discarded, all readings are shifted to the left by one to make way for the new reading, and then the new reading is added on the right.

Data for Each Gene Type - For each reading, data is collected for each of the 15 gene types found in the C1 and C2 genes.

Masthead data - At the top of the chart some general information is presented about the population of bugs examined. The total number of diploid bugs in the population is given, the number of instances of genes examined (always two per bug), and the total number of distinct alleles found (always less than the number of genes examined).

Table of Alleles - For any given gene type, an instance of a gene is identified as presenting a distinct new allele has either a distinct strength or a distinct dominance type. For example, if two instances of the C1 gene0 gene both have the same strength and the same dominance type, they are considered instances of the same allele of C1 gene0. However, if either the strength or the dominance type is different, then they are considered to be instances of two different alleles. The table of alleles presents a sorted list of all of the distinct alleles found in the population of diploid bugs (note, haploid bugs were excluded) at the time that the reading was taken. The count of the number of instances of each distinct allele is given. The total of all counts should equal the number of diploid bugs times two, as there are two instances of each gene type in each diploid bug. This table also assigns a temporary name to each allele which is used in the following tables for reference. Note that the temporary names are only good for a single reading. If you wish to compare results across readings, you need to make a few notes and make comparisons manually. This table has the following columns:
- allele name - a temporary name as discussed above, looks like 'A0001';
- allele strength - the strength of the allele;
- allele dominance type - the dominance type of the allele; together with the strength, this determines an allele uniquely;
- frequency of this allele and the percentage of genes presenting this allele, computed as the frequency over the total number of genes examined;

- the number of bugs which carried this allele, and the percentage of bugs presenting this allele, computed as the number of bugs presenting over the total number of diploid bugs.

This shows the penetration of the allele into the total population of available genes, and into the total population of available bugs. It is sometimes instructive to plot these penetration rates for an allele over time. This, of course, must be done manually.

Dominance Table - This is a two dimensional table in which the dominance type of each allele in the table of alleles is compared to the dominance type of every other allele in the same list. This table is 50% redundant, in that if A0001 is dominant over A0002, it must be indicated at both places in the table. Therefore, the table is always symmetrical. If you want to know whether A0001 is dominant over A0004, look at the cell which is at the cross between the 1st row and the 4rth column. It will show either A0001, or A0004, or equivalent (indicated as 'Equiv.'). The equivalence indicator will only arise if the alleles, since each attained this dominance type, have never occurred together in the same bug, and therefore the dominance has never been arbitrated.

Table of Allele Cross Counts - This is a two-dimensional table again, with the same structure and organization as the Dominance table. This table shows us the number of times that a given pairing of alleles has occurred in the population of bugs. For example, if a bug presents two instances of the same allele A0004, then the cell at the cross between row 4 and column 4 would have a 1 added to it. If another bug presented A0001 as its maternal gene, and A0003 as its paternal gene, then the cell at the cross of row 1 and column 3 would have a 1 added. The total count in all cells must add up to the number of diploid bugs in the population. This table will not normally be symmetrical, varying from symmetry either due to chance or to bias in the genes (e.g. a gene governing sexual aggression would cause an asymmetry, however we do not include the C3 genes in this analysis). However, every row should have a count of at least one, and every column should have a count of at least one.

Table of Allele Cross Percentages Square - This is a third two-dimensional table, with the same structure and organization as the previous two. This square presents the results of the table of allele counts as percentages of the total number of diploid bugs in the population.

How would Mendel's Experiment Look using Pairing Squares in a bowl of PSoup?

You can see for yourself if you run Scenario 5C. However, here is a translation of how it might work.

In Scenario 5C all of the C2 genes come in two values which differ by 1%. So, for example we have two and only two alleles of the DAT gene. One is at 400. The other is at 404. For ease of discussion, let's call the 400-allele of the DAT gene 'R', and lets call the 404-allele of the DAT gene 'W'. We can then continue with our terminology.

Zeroth Generation - Pairing Square for the zeroth generation of Pea plants, showing 50% of the population having an RR pairing, and 50% having a WW pairing.

	R	**W**
R	50%	0%
W	0%	50%

Note that this is identical to the pairing square drawn above when translating his results into pairing squares.

Using a Punnett Square to Predict the First Generation - Note that when Mendel crossed his red peas with his white peas, he did not allow any RRxRR fertilization, or WWxWW. When he fertilized his zeroth generation, he only allowed RRxWW fertilization. However, in Scenario 5C we have a population which is 50% RR and 50% WW but they are allowed to freely fertilize each other.

|| **R** | **R** || **W** | **W** ||

R	|| RR	| RR	|| RW	| RW	||
R	|| RR	| RR	|| RW	| RW	||
W	|| RW	| RW	|| WW	| WW	||
W	|| RW	| RW	|| WW	| WW	||

We see that three strains, RR, RW and WW will be produced. But in what ratios? It looks like 25%, 50% and 25%, but is it?

Using a Pairing Square to Predict the First Generation - When an RR strain looks to be fertilized, there are 9 RR and 10 WW bugs to choose from:
- 9/19 of the RR population will be crossed RRxRR; and
- 10/19 of the RR population will be crossed RRxWW;
- similarly, 9/19 of the WW population will be crossed WWxWW; and
- 10/19 of the WW population will be crossed WWxRR.

For the RR population, there are 10 bugs; (9/19)x10 bugs will cross as RR; this is out of a population of 20 bugs; the percentage of bugs therefore expected to be involved in RR crosses is 100 x (9/19) x 10/20, or 100 x (9/19)/2.

23.7% = 100 x (9/19)/2.
26.3% = 100 x (10/19)/2.

Pairing Square for the first generation of Pea plants, produced in a bowl of PSoup and allowed to fertilize freely.

	| **R**	| **W**	|
R	| 23.7%	| 26.3%	|
W	| 26.3%	| 23.7%	|

Note the stark difference between this result and the result Mendel got when he controlled who fertilized whom. We now have a population which is:
23.7 % RR;
23.7% WW; and
52.6% RW.

Using a Punnett Square to Predict the Second Generation - To complete Mendel's experiment we would have to zap (eliminate) all of the offspring of the naughty parents who played with their own strain instead of with the other strain. We cannot do that any more than we could prevent such naughtiness. We must forge ahead, as nature must do generation by generation.

A Punnett Square to take us to the next generation would be a 3x3 matrix of 2x2 matrices, with 36 little cubbyholes. Properly executed, it could tell us the probable makeup of the next generation:

	|| **R**	| **R**	|| **R**	| **W**	|| **W**	| **W**	||
R	|| RR	| RR	|| RR	| RW	|| RW	| RW	||
R	|| RR	| RR	|| RR	| RW	|| RW	| RW	||
R	|| RR	| RR	|| RR	| RW	|| RW	| RW	||
W	|| RW	| RW	|| RW	| WW	|| WW	| WW	||
W	|| RW	| RW	|| RW	| WW	|| WW	| WW	||
W	|| RW	| RW	|| RW	| WW	|| WW	| WW	||

Surprisingly, if we count up the cubbyholes of each kind, we get:
- RR - 9/36;
- RW - 18/36;
- WW - 9/36.

But, again, this is misleading. We did not have equal numbers going in. If we had equal numbers of each strain going in, then we would expect each cubbyhole to have equal weight, but THEY DO NOT HAVE EQUAL WEIGHT. What is the expected ratio of the three strains after two generations of unrestricted fertilization?

There are a couple of ways to work this out. One way is to build a 36x36 matrix of probabilities and then use them as weights. We start with a 3x3 matrix. There are three strains, RR, RW and WW, which can cross with each other, so we can set up a 3x3 matrix to determine how many organisms are expected to participate in each type of cross. First, lets determine how many of each strain exist in the bowl. It is NOT ENOUGH to know the percentages in a small population. Since we exclude self-fertilization (as it is not likely in this scenario), if there are N bugs in the RR population (e.g.) then each bug has only (N-1) opportunities to participate in an RRxRR cross, so the percentage of self-crosses is corrected. The percentage size of the correction varies with the size of N, even though the absolute size of the correction (i.e. 1) does not. The size of the zeroth population was 20 bugs. The size of the first population is 40 bugs. The number of bugs, by strain, is:
- RR - 9.47;
- RW - 21.05;
- WW - 9.47, which adds to 40;

Note, we work in fractions of a bug, as these are statistically expected numbers, not actual bugs. Let's maintain the accuracy until we have to actually realize bugs.

The nine entries are computed as follows:
1 - RR cross RR - For any given RR bug, there are (9.47-1) possible RRxRR pairings of a total of (40-1) possible pairings; the expected number of RR bugs that would therefore be expected to participate in an RRxRR cross would be [the number of RR bugs] * [the probability of an RRxRR pairing] which is [9.47] * [(9.47-1)/(40-1)] or 2.058 bugs; as a percentage that's 100 * (2.085/40) = 5.15%.
2 - RRxRW - For any given RR bug, there are 21.05 RW bugs which pairing might occur of a total of (40-1) possible pairings; the number of RR bugs participating in such pairings is [9.47] * [21.05/40] or 5.114 bugs; as a percentage of 40, that's 12.78%;
3 - RRxWW - For any given RR bug, there are 9.47 WW bugs with which pairing might occur of a total possible (40-1) pairings; the number of RR bugs participating in such pairings is [9.47] * [9.47/40] or 2.301 bugs; as a percentage of 40, that's 5.75%;
We work through the next six in similar fashion; in short form:
4 - RWxRR - 9.47 possible RWxRR pairings of 39; 5.114 RW bugs participate; that's 12.78%;
5 - RWxRW - (21.05-1) possible RWxRW pairings of 39; 10.825 RW bugs participate; that's 27.06%;
6 - RWxWW - 9.47 possible RWxWW pairings of 39; 5.114 RW bugs participate; that's 12.78%;
7 - WWxRR - 9.47 possible WWxRR pairings of 39; 2.301 WW bugs participate; that's 5.75%;
8 - WWxRW - 21.05 possible WWxRW pairings of 39; 5.114 WW bugs participate; that's 12.78%;
9 - WWxWW - (9.47-1) possible WWxWW pairings of 39; 2.058 WW bugs participate; that's 5.15%;

Let's put those percentages into a 3x3 matrix:

	RR	**RW**	**WW**
RR	5.15	12.78	5.75
RW	12.78	27.06	12.78
WW	5.75	12.78	5.15

We're almost there. Now, for each of these nine possible types of crosses, there are four possible outcomes with equal probability of occurrence. A standard 2x2 Punnett square handles each, and, in fact, we have

those in the large Punnett Square above. The above 9 percentages represent the probability of each of the nine types of pairings. If we divide these percentages by 4, we have the probability of each of the 36 possible outcomes of crosses between the 40 generation 1 denizens of our bowl of PSoup.

Let's add 'em up.
RR - (5.15%) + (.5*12.78%) + (.5*12.78%) + (.25*27.06%) = 24.70%
WW - (5.15%) + (.5*12.78%) + (.5*12.78%) + (.25*27.06%) = 24.70%
RW - (2*5.15%) + (.5*12.78%) + (.5*12.78%) + (.5*12.78%) + (.5*12.78%) + (.25*27.06%) + (.25*27.06%) = 50.60%

That's how a Punnett Square works!

Pairing Square for the second generation of Pea plants, produced in a bowl of PSoup and allowed to fertilize freely.

	R	**W**
R	24.7%	25.3%
W	25.3%	24.7%

This means:
- 24.7% of the bugs in the first generation have a maternal 400-allele in the DAT gene and a paternal 400-allele in the DAT gene;
- 25.3% of the bugs in the first generation have a maternal 400-allele in the DAT gene and a paternal 404-allele in the DAT gene;
- 25.3% of the bugs in the first generation have a maternal 404-allele in the DAT gene and a paternal 400-allele in the DAT gene;
- 24.7% of the bugs in the first generation have a maternal 404-allele in the DAT gene and a paternal 404-allele in the DAT gene;

In Summary

Punnett Squares are commonly used by biologists to predict the future distribution of alleles in a population based on a known current distribution of alleles in a population.

Pairing Squares are unique to PSoup and are simply used to present the current set of pairings of alleles in a bowl of PSoup. Students with knowledge of Punnett Squares can use a pairing square as input to a Punnett Square exercise. The can get very complicated.

A Pairing Square shows the percentage of the total population of diploid bugs which carry each possible pairing of alleles, and distinguishes between maternal and paternal loci for the allele. A population which carries 5 alleles of a gene will therefore be described in a 5x5 pairing square.

An historic series of pairing squares gives you a view into the dynamics of a changing population, and the competition between and development of alleles.

For the Teacher

Lesson Organization

This article covers possible modes of delivery, and possible lesson content.

What PSoup is NOT

PSoup is not intended to replace a biology text book. If a student needs to learn about a number of facts and theories quickly, a good text book is the best source. Understanding of evolutionary processes is improving rapidly, and PSoup does not profess to present the most up-to-date views, or even to present all sides of any view. Any serious student of evolution must check the learnings from PSoup against other credited sources.

What PSoup IS

PSoup is a laboratory in which REAL evolution can be invoked in an experimental environment. Hypotheses can be formulated, models constructed, experiments designed, results tabulated, models tested, and hypotheses confirmed or overturned.

PSoup has many scenarios which are more-or-less remotely analogous to real-world situations, and can be used as an arena to clarify ideas and opinions about what is happening in the real world.

Content Delivery Tools

PSoup has a number of built-in tools for providing learning opportunities, such as:
- canned scenarios, each with a description of purpose, references (in the PSoup Help System), things to watch for and think about, and exercises.
- an extensive PSoup Help System containing technical instructions about the functions and features of PSoup.
- a broad series of articles in the 'Theory and Practice' section which explore the concepts and questions raised respecting evolutionary processes, and the implementation in PSoup.
- a package of templates for some of the lower level scenarios, which are to be used for data collection for the experimental exercises. The file name is MyPSoupNotes.RTF. It is intended that users would construct their own experiments, and each such experiment would need its own specific data sheets. These templates are intended both for use in the canned exercises, and as examples of what should be drafted by the student for other user-determined experiments.
- a couple of sample recordings, one on the basics of PSoup, and one respecting the Easter Island scenario. These are stored in a directory called 'Recorder'. The teacher can create other recordings and add learning stops with tutorial text.
- a fridge full of old bowls of PSoup, for use in some of the exercises, if needed. Many exercises say something to the effect of 'run the scenario up to one PSoup hour, take notes, then run it for an additional two hours.' To save the teacher some time, and to add some predictability to a probabilistic process, a number of partial runs have been saved. Each run is named by its scenario number followed by the day, hour and minute time stamp of the aged bowl of PSoup. E.g. a bowl of soup derived from scenario 1A which is 1 day, 2 hours and 3 minutes old would be called 1A_01D02H03M.PSP. The teacher may choose to add other old bowls of PSoup to the fridge. Do not confuse the fridge with the normal place for storage of intermediate bowls of PSoup, which is the PspBowls directory.
- the ability to tinker with the parameters of any scenario, creating new scenarios. In conjunction with the recording feature, a teacher can create new canned scenarios which are loaded via the recorder rather than with the Scenario command.
- canned Level Advancement Tests (LATs). These check that a student has learned the necessary ideas needed to operate PSoup well at the next higher level. These tests can be augmented or replaced by the teacher. Such teacher-built tests are stored in a directory called PspLats (PSoup Level Advancement Tests) and can be edited or deleted. The canned tests are generated by PSoup and are not stored in that directory.

- the ability to do wide-open analyses of populations of bugs using pairing squares, cluster analysis, or historic trend analysis on counts of alleles, or average values of alleles.

Modes of Delivery

PSoup is designed to be used in several modes, as the need requires:
- short teacher-led discussion based on a pre-recorded run, either using one of the two canned recordings, or using a teacher-build recording with edited stops.
- student-operated viewing of a pre-recorded run, reading scripted stops, and answering questions in a workbook for later review by a teacher. No sample recordings are provided for this mode of use. The teacher would need to record, analyze, script and edit a run to make the recording for this mode of use.
- student-operated run of a canned scenario under teacher supervision, with completion of an exercise such as those provided with each scenario, using the data collection templates and making notes in the PSoup Experimenter's Notebook. When enough scenarios are completed, the student advances to the next level by completing the canned or teacher-built LAT. With this mode of use, if the teacher wants full control of the student's progress, the teacher must load the software and set the mode to student mode each time the student is to use the software. The default mode is teacher mode.
- open operation by a student, with responsibility in the hands of the student to use the recordings and LATs to properly time advancement to the next level. Execution of canned exercises using canned scenarios to gain the information needed to advance.
- wide-open experimental mode, in which the student is challenged to construct hypotheses, construct experiments to test those hypotheses, execute the experiments, and analyze the results. For this to work, the student must understand the PSoup environment very well. It is recommended that articles in Theory and Practice be read, and the LATs be completed for a level before experiments are constructed at that level.

Possible Lesson Content

There are three types of lessons (or learning opportunities) designed into PSoup. For discussion in this paper, these three types are labeled as types A, B and C.

- A - The first kind is specific to PSoup, and these address the technical issues about how PSoup operates, and how a teacher or student can effectively use PSoup's capabilities. These facilitate the other learning opportunities.
- B - The second type of lesson is with respect to how evolution and natural selection work. It is the goal of PSoup to provide a laboratory in which real (but logical) natural selection is at work, and students can adjust the parameters and circumstances to investigate the changes in the evolutionary processes. While PSoup does not emulate any known natural biological system, it is nevertheless designed to be highly analogous to biological systems. Students who are familiar with basic biology should see similarities and obvious differences between the logical world of PSoup and the natural world of biology. The lessons learned from the study of PSoup's evolutionary processes should be indirectly applicable to any study of natural biology.
- C - The third type of lesson is with respect to the nature of and behavior of the classes of logical systems, such as finite state machines, evolutionary algorithms, and Monte Carlo simulations, of which PSoup is a member. PSoup is not designed to provide broad coverage of such systems, but rather, in providing broad coverage of evolutionary processes, it provides a small selection of such systems. Students of complex systems should find PSoup to be an interesting environment in which they can generate a few different sample systems for study. Lessons in this group examine some of the concepts and terminology used to study such complex systems.

In the following table, the proposed lessons are grouped according to each of the six PSoup levels. Within each level, the lessons are labeled as to type: A, B or C. For each type of lesson, when applicable, reference is made to a scenario or to an article in 'PSoup Theory and Practice' (T&P). The PSoup Theory and Practice articles are part of the PSoup Help System.

Level 1 Lessons

A - What is happening in the bowl (Scenario 1A; T&P: Take a swim in the soup)
A - Palmiter Genes (in chromosome 1), how they work (Scenarios 1A and 1B; T&P: A PSoup primer)
A - Edge effects - the edge of the bowl - what it is and how to modify it (Scenario 1C; T&P: PSoup and edge effects).
A - Explanation of the PSoup pun (Scenario 1C; T&P: About the PSoup analogy).
A - Terrain effects - the Desert/Oasis Ecosystem (DOE) - what it is and how to modify it (Scenario 1D; T&P: PSoup and terrain effects).
A - Understanding the family tree, energy chart, chromosome 1 population profiles, and C1 genetic profiles for the current and average bug (Scenarios 1A, 1B and 1C).
A - Introduction to DAT, DET, RAT, RET, EPM and MEPB as common system-wide parameters that regulate life functions (all scenarios; T&P: A PSoup primer).

B - Demonstrations of changing genotypes as organisms compete for resources and adapt to survive (Scenarios 1A, 1C and 1D).
B - The concepts of genotypes and phenotypes (Scenario 1B; T&P: About genotypes and phenotypes).
B - The concept of alleles and profiles of allele distributions in a population (Scenario 1C);
B - De-confusing evolutionary terminology (Scenario 1D; T&P: Survival of the fittest)
B - Understanding the interplay of genes, circumstance, and chance (Scenario 1D; T&P: The role of chance)
B - PSoup as an unequivocal demonstration (Scenario 1D; T&P: Demonstration vs simulation)
B - How environment determines the outcome of natural selection, as edge effects and terrain effects are changed (Scenarios 1C and 1D).
B - The emergence of species (Scenario 1D; T&P: About speciation).
B - Predicting the outcome of evolution (Scenario 1E).

C - PSoup as a finite state machine (Scenario 1F; T&P: PSoup as a finite state machine).

Level 2 Lessons

A - Introduction to DAT, DET, RAT, RET, EPM and MEPB as individual parameters under the control of regulatory genes in chromosome 2 (Scenario 2A) and subject to evolutionary pressures (Scenarios 2B, 2C, 2D and 2E; T&P: PSoup and life functions).
A - Controlling the speed of runs (Scenario 2B; T&P: About speed controls).
A - Stamina, Fertility and Agility as non-phenotypic characters (Scenarios 2B, 2C and 2D).
A - Introduction of cascading paddies as ecological niches - how they work (Scenario 2E; T&P: About paddies).
A - Changes to the chromosome 1 population profiles (Scenario 2A).
A - Understanding the chromosome 2 population profiles, and the chromosome 2 genetic profiles for the current and average bug (Scenario 2A).

B - More about evolutionary pressures on genes, specifically on the regulatory (chromosome 2) genes. Which ones will go up in value and which ones will go down in value? (Scenario 2A).
B - More genotypes and phenotypes (Scenarios 2B, 2C, 2D and 2E).

C - PSoup and attractor sets (Scenario 2F; T&P: PSoup and attractor sets).

Level 3 Lessons

A - Introducing Gene8 of the Palmiter genes, the Stand Still gene (Scenario 3A, 3B and 3C).
A - Plant-like phenotypes in PSoup (T&P: PSoup and plant-like phenotypes).
A - Stamina, Fecundity, A - Introducing the need for, and use of, chromosome 1 penalties, for controlling edge effects (Scenario 3D; T&P: PSoup and edge effects).
A - Introducing the need for, and use of, chromosome 2 penalties, for controlling edge effects (Scenario 3D; T&P: About chromosome penalties).
A - Open and closed energy systems (in PSoup) and changes to the energy chart, for open systems (Scenario 3E; T&P: About PSoup's energy systems).

A - Independent paddies and their use for tests of parallel evolution (Scenarios 3F and 3G; T&P: About paddies).

B - Sessile or plant-like organisms with restricted mobility (Scenarios 3A, 3B and 3C).
B - Pleiotropic genes and Stamina, Fecundity and Agility as phenotypic characters, acting through the chromosome penalties (Scenario 3D).
B - Population collapse (Scenario 3E).
B - Convergent evolution (Scenario 3F).
B - Divergent evolution (Scenario 3G).
B - Ecological Niches - each set of environmental conditions leads to predictable evolutionary results (Scenarios 3F and 3G).

Level 4 Lessons

A - The capability genes, including the 'fight' genes in chromosome 3 and the flight genes in chromosome 4, and how they work (Scenarios 4A, 4B, 4C, 4D and 4E; T&P: About capability genes, about combat profiles).
A - Five senses (sight, hearing, smell, taste and touch), how they work in PSoup, and interact in PSoup(Scenarios 4A, 4B, 4C, 4D and 4E).
A - Feeding habits; algivore, herbivore, carnivore and omnivore feeding habits. (Scenarios 4A, 4B, 4C, 4D and 4E; T&P: The evolution of the senses).
A - Understanding the chromosome 3/4 genetic profiles, the chromosome 3/4 population profiles, the sensory + tinker tab (all scenarios).
A - Life function panels - how they work.(all scenarios).
A - The need for and use of the chromosome 3/4 penalty (all scenarios).

B - 'Awareness of the environment' as a phenotypic driver for evolution, and the supremacy of chromosome 3 over chromosome 1 (Scenarios 4A, 4B, 4C, 4D and 4E).
B - Algivore, herbivore, carnivore and omnivore 'feeding habits' as a phenotypic driver for evolution (Scenarios 4A, 4B, 4C, 4D and 4E).
B - Predator/prey relationships, parasite/host relationships as phenotypic driver for evolution (Scenarios 4A, 4B, 4C, 4D and 4E).
B - The emergence of new species (Scenarios 4A, 4B, 4C, 4D and 4E; T&P: About speciation).

C - Emergent behavior - feeding frenzy (Scenario 4D; T&P: PSoup and emergent behavior).
C - PSoup and Chaos, experimental production of a bifurcation diagram (Scenario 4F; T&P: PSoup and chaos).

Level 5 Lessons

A - Advanced reproductive modes, cross-over and sexual reproduction (Scenario 5A and 5B; T&P: Advanced reproductive modes).
A - Modifying the chemical rules behind PSoup (Scenario 5D).

B - Cross-over of genetic material between haploid organisms (Scenario 5A).
B - Recombination of genetic material via sexual reproduction, and introduction of haploid and diploid organisms (Scenario 5B; T&P: About haploid and diploid forms).
B - The emergence of new species (Scenarios 5A and 5B; T&P: About speciation).
B - Alleles (T&P: PSoup and alleles).
B - Mendel's pea plant experiments (Scenario 5C).
B - Hardy-Weinberg and non-random reproduction due to mating preferences (Scenario 5D; T&P: PSoup and Hardy-Weinberg).
B - Hardy-Weinberg and mutation pressure (Scenario 5E; T&P: Biased mutations, PSoup and Hardy-Weinberg).
B - Evolution in a population in which no mutation occurs (Scenario 5F; T&P: PSoup and Hardy-Weinberg)

C - Emergent behavior, mating balls (Scenario 5B; T&P: PSoup and emergent behavior).

Level 6 Lessons

A - Advanced functions, only activated if one of the senses is activated, herding and thinking (Scenarios 6A and 6B; T&P: The higher functions).
A - Seasonal variation - how it works (Scenario 6D).

B - The evolution of higher functions such as herding and thinking (Scenarios 6A and 6B).
B - Hardy-Weinberg and the immigration/emigration of genes; migration of beneficial genes throughout a population and the emergence of new species (Scenario 6C; T&P: PSoup and Hardy-Weinberg).
B - The impact of seasonal variations (Scenario 6D).
B - Hardy-Weinberg and genetic drift, loss of alleles in small populations (Scenarios 6F and 6G; T&P: PSoup and Hardy-Weinberg).

C - Emergent behavior, the best example, herding (Scenario 6A; T&P: PSoup and emergent behavior).
C - The effects of chaotic behavior on speciation (experimental, Scenario 6D; T&P: Chaos and speciation).

About the Recorder Function (Recorder menu)

This article provides an overview of the operation and purpose of the recorder function. For information about each command in the 'Recorder' menu, see 'Recorder command'.

PSoup is a probabilistic system. Given two identical bowls of PSoup (which is likely to happen only if you load the same saved bowl of PSoup twice), it is very very unlikely that the events that would unroll as you execute a run in each such bowl would be the same. In fact, the future history of each of the two bowls of PSoup would very probably start to diverge in the very first PSoup second of independent history. Any randomly placed algae would almost certainly go into different cells. Any randomly taken steps by the bugs, would almost certainly go in different directions. Any mutations or other actions under the control of the pseudo-random number generator would almost certainly be different.

This poses a problem for teachers who wish to use PSoup in a classroom setting in which time is short, and the ability to display certain evolutionary phenomena on demand is critical. PSoup addresses this problem with the 'Recorder' function and its associated list of commands.

Allegorically, the recorder function of PSoup is similar to the operation of a single-track tape recorder. You can record a PSoup session (run within a bowl of PSoup) and store the tape in a library of tapes. You can select a tape and edit it for use by a student in a classroom setting, or in a self-study program. If/when a tape is no longer of use, delete it from the library and discard it.

What can be recorded?

The recording function is intended to record all actions within and around a single bowl of PSoup between the time (PSoup second) when the recorder is turned on, and the time (PSoup second) when the recorder is turned off again.

PSoup automatically records every critical change within a bowl of PSoup such that the run can be replicated precisely on playback. The focus is on precise repetition of the events within the bowl, but certain commands affecting the display panels around the bowl of PSoup are also recorded, to enable some limited control of the presentation by the teacher or user preparing the recording.

The following types of events are recorded:
- Distribution of algae;
- Bug move;
- Algae eaten;
- Bug attack;
- Bug birth;
- Bug death;

In addition, the following commands can be recorded:
- Display Average Bug;
- Display Current Bug;
- Display PSoup Status;
- Display Scenario Legend;
- Display Event Counts;
- Display Family Tree;
- Display Energy Chart;
- Display Life Function Panels; and
- Display Population Profiles.

When a command is recorded, PSoup notes which panel is turned on or off, and records that instruction. When the recording is played back, PSoup will issue the save command at that moment in the run, but

PSoup does not prevent a user from issuing similar commands from the keyboard during a playback. This allows a user to have full use of the investigative tools of PSoup during a playback, but also allows limited control over presentation by the person recording the run.

The following commands cannot be recorded:
- All file commands;
- All scenario commands;
- All Evolution commands;
- All advanced Options commands;
    - Activate Readings of Allele Counts;
    - Display Allele Chart;
    - Display Cluster Analysis Report;
    - Bifurcation Diagram;
    - Control Panel;
- All Tinker commands;
- All Recorder commands; and
- All Help commands.

All events are stored in a master 'Event File' which can be viewed by the user during playback.

What are the modes of operation of the recorder?

From the point of view of the recorder, there are four modes of operation:
- Normal PSoup mode - in which the recorder is turned off and not operational;
- Record mode - in which all critical events are captured and recorded for later playback;
- Edit mode - in which a previously recorded run can be edited for the insertion of pedagogical information; and
- Playback mode - in which a recording is played back for instructional purposes.

What is the life cycle of a recording?

PSoup is normally in 'Normal' mode.

To make a recording:
- Select the scenario you wish to record.
- Use the Tinker menu to make any needed adjustments before you start to record. Set the control panel to 'Teacher' mode.
- Adjust the display panels to appear as you want them to appear when the recording starts.
- Finally, advance the age of the bowl of PSoup (if necessary) to the point in (PSoup) time at which you want to start the recording.
- Use the 'Start Recording' command to enable the recorder and put it into Record mode.
- You are now recording all significant events. Advance the age of the bowl of PSoup. Stop when appropriate and re-adjust the display panels around the bowl. Do you want to focus on the 'Energy chart' for a while? Enable it at the right time. If you are uncertain what panels should be active at a given time, leave them as is and you will have a chance to instruct the student to over-ride the settings at time of playback.
- When a recording is completed, use the 'Stop Recording' command to terminate the recording session and put the recording into the library of available recordings.

PSoup has now returned to 'Normal' mode.

To edit a recording:
- Be sure PSoup is in 'Teacher' mode. If PSoup is not in Teacher mode, exit from PSoup and re-enter the program. Teacher mode is the default mode for PSoup when it first starts up.
- When you are certain that PSoup is in Teacher mode, use the 'Load a recording' command to select a recording from the library of available recordings and load it. PSoup is now in edit mode.

- It is suggested that you edit in three passes. In the first pass, watch the evolution of the bowl of PSoup and note any trends or events that have pedagogic value. Make note of the timing of each. In the second pass, carefully research the roots of each event, and the exact timing, making notes as you go. Decide at what points you want to insert stops, and how you want to instruct the students proceed from stop to stop. Include settings for the various panels, and exercises the students could perform, or things to watch for. You are, in effect, studying the recording in detail and planning a learning experience for the student. In the third pass, edit the masthead and insert and edit each of the teaching stops. All edits are effective immediately, so there is no need to save the edited recording.
- To close the edited recording, use the 'Close this recording' command.

PSoup has now returned to 'Normal' mode.

To replay a recording:
- Be sure PSoup is in 'Student' mode.
- Use the 'Load a recording' command to select and load the recording. PSoup is now in 'Playback' mode.
- Follow the instructions in the masthead screen and in each of the teaching stops, performing any actions and exercises recommended therein.
- When the recording is completed, use the 'Close this recording' command to close the recording and return to 'Normal' mode.

PSoup has now returned to 'Normal' mode.

Some hints

Note that, when in record mode, any and all changes to the display panels are recorded, and cannot be deleted from the recording later. Do not do investigations with the record mode turned on, as every change in the panels will be recorded and played back. There are two alternate approaches to handling of these changes in panels. You can make panel display changes during the recording session IF YOU KNOW WHAT IS EXPECTED TO HAPPEN and you have planned the changes carefully and precisely. Or, you can enable a standard set of panels before starting to record, and use the teaching stops (inserted later) to instruct the student as to which panels should be enabled when.

The recorder function is intended for pedagogic purposes, and not investigative purposes. While the maximum size of recording supported is greatly dependent on the configuration of the computer on which PSoup runs, any sizable recording (of length, say, greater than two PSoup hours) may cause the computer to crash. If you wish to use recordings of a run over any great length of time (in PSoup time) it is suggested that you use several independent recordings.

If you wish to try to trap interesting phenomena for later study using the recorder, it is suggested that each target run be recorded in segments of suitable size. Each segment can then be loaded independently for later analysis.

Genetic Map

In PSoup we start with a very basic genetic structure at Level 1 and work towards a two genetic maps by Level 5. In the following, the possible genomes are labeled with a capital alphabetic letter, and the table indicates where each genome first appears. The genome is available at level and at all higher levels.

Genome label	Genome type	Level of first appearance
A	Chromosome 1, 8 transport genes One chromosome in total.	1
B	Chromosome 1, 8 transport genes Chromosome 2, 6 regulatory genes Two chromosomes in total.	2
C	Chromosome 1, 9 transport genes Chromosome 2, 6 regulatory genes Two chromosomes in total.	3
D	Chromosome 1, 9 transport genes Chromosome 2, 6 regulatory genes Chromosome 3, 26 fight genes Chromosome 4, 26 flight genes Four chromosomes in total.	4
E	Chromosome 1, 9 transport genes Chromosome 2, 6 regulatory genes Chromosome 3, 26 fight genes Chromosome 4, 26 flight genes Four chromosomes in total. Plus temporary set of 4 stolen cross-over chromosomes	5
F	Maternal chromosome 1, 9 transport genes Maternal chromosome 2, 6 regulatory genes Maternal chromosome 3, 26 fight genes Maternal chromosome 4, 26 flight genes Four maternal chromosomes in total. Paternal chromosome 1, 9 transport genes Paternal chromosome 2, 6 regulatory genes Paternal chromosome 3, 26 fight genes Paternal chromosome 4, 26 flight genes Four paternal chromosomes in total. Plus temporary set of 4 spermatic chromosomes.	5

In addition, at levels 4, 5 and 6, we gradually add new phenotypic functions for some of the genes introduced at level 4. The phenotypic functions are turned as follows:

Level	New phenotypic features for chromosomes 3 and 4
4	SIGHT, SMELL, HEAR, TASTE, TOUCH
5	XOVER, OVULE
6	HERD, THINK

Checking the genome

If at any time you wish to view the genes in a bug, use the 'Tinker with the Bugs' command of the Tinker menu, select the bug you wish to examine, and click on the 'Display Genes' button.

Primary Page and Menu Structure

Primordial Soup Help - Primary Page

[Actual text from the primary page.]

Welcome to the help system for Primordial Soup. This help system is set up, not only to help you navigate around the PSoup application, but also to help you better understand the terminology that is peculiar to PSoup, and to help you understand the applicability of PSoup terminology and processes to analogous biological terminology and processes.

Below you see a list of menu items which correspond to the menu options offered a user in the main screen of PSoup. If you left-click on any underlined item in the list below, you will be presented with more detail. Within the help system, any underlined word is clickable, and will display additional information if clicked.

You will also notice, above, on the toolbar across the top of the help system screen, a button marked 'Contents'. Either left-click this with your mouse, or use an 'Alt-C' key combination, to pull up the full table of contents of the help system. The menu commands are replicated there as well as here, but there is much more there to see as well.

To invoke this help system at any time, left-click the help button in the 'Help' toolbar at the top right of the PSoup screen (the button with the yellow question mark) or press the 'F1' key at the top left corner of your keyboard.

Main Menu Commands

File menu
Scenarios menu
Evolution menu
Options menu
Tinker menu
Tests menu
Recorder menu
Help menu

File menu commands

The File menu offers the following commands:

New Creates a new bowl of PSoup.
Open Opens an existing bowl of PSoup.
Save Saves a bowl of PSoup using the same file name.
Save As Saves a bowl of PSoup to a specified file name.
Exit Exits PSoup.

Scenarios menu commands

The Scenarios menu offers a list of 'canned' scenarios for each active level. Canned scenarios are available as follows:
Level 1 - Five scenarios
Level 2 - Five scenarios

Level 3 - Five scenarios
Level 4 - Five scenarios
Level 5 - Two scenarios
Level 6 - Three scenarios

Two types of scenarios are presented: (1) those which are designed to help the user better understand how PSoup works, and what can be done with it, and (2) those which are designed to provide insight into analogous biological processes. Users may use the canned scenarios as the basis for their own scenarios using the 'Tinker' commands.

Evolution menu commands

The Evolution menu offers the following commands:

ReSeed Re-start the scenario.
Pause Stop the PSoup clock, and enable all appropriate menu items.
Go Start the PSoup clock, and disable all appropriate menu items.
Advance By One Second Advances the PSoup clock by one PSoup second. All bugs take one turn each.
Advance By One Bug One bug takes a complete turn. It completes all five life functions.
Go Slow Mode Advance at a maximum rate of three PSoup seconds per real second.
Advance By One Function One bug takes a partial turn. It completes a single life function.

Options menu commands

The Options menu offers the following commands:

Display Average Bug Cycles through the three genetic profiles for the average bug.
Display Current Bug Cycles through the three genetic profiles for the current bug.
Display PSoup Status Toggles the PSoup Status panel on and off.
Display Scenario Legend Cycles the Scenario Legend panels.
Display Event Counts Toggles the Event Counts panel on and off.
Zero Event Counts Zeros all event counts.
Display Family Tree Toggles the Family Tree panel on and off.
Display Energy Chart Toggles the panel for the Energy Chart on and off.
Display Life Function Panels Toggles the panels for the Life Functions on and off.
Display Population Profiles Cycles through the panels for the Population Profiles.
Activate Readings of Allele Counts/Valuess Toggles auto reading on and off.
Display Allele Counts Displays a dialog with the allele chart enclosed.
Display Allele Values Displays a dialog with the allele chart enclosed.
Display Cluster Analysis Report... Opens dialog to perform cluster analysis.
Bifurcation Diagram Enable or disable data collection for presentation of a bifurcation diagram.
Sounds Toggles event sounds on and off.
Control Panel Invokes the Control Panel dialog.

Tinker menu commands

The Tinker menu offers the following commands:

Tinker With The Environment Invokes a dialog by which you can modify the parameters of the ecological system in which the bugs evolve.
Tinker With The Bugs Invokes a dialog by which you can modify the acquired characteristics

and genetic code of each bug in turn.

Tests menu commands

The Tests menu offers five Level Advancement Tests (LATs) by which a student can open up new capabilities in PSoup. Each test consists of a 10-question multiple-choice quiz. The questions are drawn from a store of over thirty questions specific to the current level. If you have run all of the scenarios at the current level, and read and understood all about this scenario for each scenario, then you will easily pass the test.

When you open the Tests menu, you will be offered a list of available tests. Typically, you will have access to all tests for previous levels plus the current level. As a student user, you must pass the test at the current level to have access to new capabilities. These new capabilities include new scenarios, new abilities to tinker with either the environments or the bugs, new or more fully enabled genetic code in the bugs, new environmental settings, and, of course, a new test.

When you choose the test available to you, you will be presented with two dialogs. The first dialog provides some brief instructions for completion of the test. The second dialog presents the test itself.

In addition, in Teacher mode PSoup offers the ability to modify or completely replace the canned test material.

Recorder menu commands

The Recorder menu offers a variety of commands in the following groups:

A. Commands to manage existing recordings:
Load a recording Load an existing recording into PSoup from the library of available recordings, in preparation for playback.
Play 1 second (F10) Play back one second (the next second) from the currently loaded recording.
Play 60 seconds Play back sixty seconds (the next sixty seconds) from the currently loaded recording.
Play to next stop Play back advances from the currently loaded recording, second-by-second, until the next 'Teaching Stop' is reached. Stop the playback at the next teaching stop and display the teaching stop.
Play to end (F9) Play back from the currently loaded recording, not stopping until the end of the recording is reached. This initiates a background thread, similar to the 'Go' or 'SloMo' commands, which can be terminated using the 'Pause/Stop' command (F8).
Pause/Stop (F8) Terminate the background thread which is playing back the recorded advances. This puts PSoup back into step-by-step playback mode and undoes the 'Play to end' command.
Close this recording Close the currently loaded recording, and return PSoup to normal operational mode. This undoes the 'Load a recording' command.

B. Commands to manage the creation/deletion of recordings:
Start recording Switch PSoup out of normal operational mode into 'Record' mode. All changes to the bowl of PSoup and its associated display panels will be recorded in an event file and may be played back at a later time. A recording must be created using this command before it is placed in the library of available recordings.
Stop recording Stop recording all events in the bowl of PSoup, terminate the recording session, close the recording and place it in the library of available recordings. It can now be edited or

played back.
Delete a recording Select a recording from the library of available recordings, and delete it from the library. Once deleted, it will no longer be available for edit or playback.

C. Commands to manage the pedagogic contents of a recording:
Edit the masthead Load the masthead into an editable screen and present it to the teacher for entry/change of content. The masthead should contain any credits, and general instructions to the student about how to best benefit from the use of this recording.
Insert a stop here Insert a 'Teaching Stop' here, at this point in the recording. Stops can be inserted during step-by-step playback, if PSoup is in 'Teacher Mode'. See the Control Panel for more information about teacher mode. The teaching stop should give guidance to the student.
Edit this stop Load an existing teaching stop into an editable screen and present it to the teacher for entry/change of content.
Delete this stop Delete this stop from the recording. The stop, and all contents, will be deleted and will no longer be available for use by teacher or student.
View event file Take a peek behind the scenes in a recording.

Help menu commands

The Help menu offers the following commands, which provide you assistance with this application:

About This Scenario Offers you a detailed description of the purpose and interesting points to consider respecting the active scenario.
Welcome Invokes the Welcome dialog presented at the beginning of each session.
About PSoup Displays the version number of this application.
Nag Screen Displays the Nag Screen.
Register Provides information on how to upgrade a demonstration copy of PSoup to a full-featured copy, or how to order additional full-featured copies.
Credits Displays information about the authors.

File Menu

New command (File menu)

Use this command to create a new bowl in PSoup.

You can open an existing bowl of PSoup with the Open command.

Open command (File menu)

Use this command to open an existing bowl of PSoup. You can open only a single bowl at a time. If you wish to save the current bowl of PSoup for later use, use the Save As command first. See Save As ...

You can create new documents with the New command.

File Open dialog box

The following options allow you to specify which file to open:

File Name
Type or select the filename you want to open. This box lists files with the extension you select in the List Files of Type box. All PSoup bowls saved to file have the three letter extension of '.psp' as a default.

List Files of Type
Select the type of bowl you want to open:
PSoup Bowl (*.psp)

Drives
Select the drive in which PSoup stores the bowl that you want to open.

Directories
Select the directory in which PSoup stores the bowl that you want to open.

Save command (File menu)

Use this command to save the active bowl of PSoup to its current name and directory. When you save a bowl of PSoup for the first time, PSoup displays the Save As dialog box so you can name your bowl of PSoup. If you want to change the name and directory of an existing bowl of PSoup before you save it, choose the Save As command.

Save As command (File menu)

Use this command to save and name the active bowl of PSoup. PSoup displays the Save As dialog box so you can name your bowl of PSoup.

To save a bowl of PSoup with its existing name and directory, use the Save command.

File Save As dialog box

The following options allow you to specify the name and location of the bowl of PSoup you're about to save:

File Name

Type a new filename to save a bowl of PSoup with a different name. PSoup adds the extension you specify in the Save File As Type box.

Drives

Select the drive in which you want to store the bowl of PSoup.

Directories

Select the directory in which you want to store the bowl of PSoup.

Exit command (File menu)

Use this command to end your PSoup session. You can also use the Close command on the application Control menu (top left corner) or the system exit button (X in the top right corner).

Scenario Menu

Scenario 1D

Scenario 1D is the Desert Oasis Ecosystem (DOE) scenario.

At each level there is a 'Desert Oasis Ecosystem' scenario. On the options toolbar, left-click on the button showing a small rectangular bowl of PSoup with a patch of green in it. This will invoke the 'Desert Oasis Ecosystem' scenario at any level. The Level 1 Desert Oasis Ecosystem (DOE) is listed in the 'Scenarios' menu, the other DOEs for the higher levels are not, but are accessible through the toolbar alone.

Level 1 (Scenarios menu)

PSoup offers six levels of increasing sophistication and complexity. Each level offers canned scenarios with pre-determined goals, and explanatory material. The scenarios listed under the Level 1 menu are those available at level one and recommended for study prior to attempting the first Level Achievement Test. The Scenarios menu allows you to load only those scenarios which are at a level of achievement equal to, or less than, your current level. To learn more about the Level Achievement Tests (LATs) see Level Achievement Test dialog. To learn more about the explanatory material which goes along with each scenario, see About This Scenario.

Level 2 (Scenarios menu)

PSoup offers six levels of increasing sophistication and complexity. Each level offers canned scenarios with pre-determined goals, and explanatory material. The scenarios listed under the Level 2 menu are those new scenarios added after passing the first Level Achievement Test. The Scenarios menu allows you to load only those scenarios which are at a level of achievement equal to, or less than, your current level. To learn more about the Level Achievement Tests (LATs) see Level Achievement Test dialog. To learn more about the explanatory material which goes along with each scenario, see About This Scenario.

Level 3 (Scenarios menu)

PSoup offers six levels of increasing sophistication and complexity. The six levels also offer canned scenarios with pre-determined goals, and explanatory material. The scenarios listed under the Level 3 menu are those new scenarios added after passing the second Level Achievement Test. The Scenarios menu allows you to load only those scenarios which are at a level of achievement equal to, or less than, your current level. To learn more about the Level Achievement Tests (LATs) see Level Achievement Test dialog. To learn more about the explanatory material which goes along with each scenario, see About This Scenario.

Level 4 (Scenarios menu)

PSoup offers six levels of increasing sophistication and complexity. The six levels also offer canned scenarios with pre-determined goals, and explanatory material. The scenarios listed under the Level 4 menu are those new scenarios added after passing the third Level Achievement Test. The Scenarios menu allows you to load only those scenarios which are at a level of achievement equal to, or less than, your current level. To learn more about the Level Achievement Tests (LATs) see Level Achievement Test dialog. To learn more about the explanatory material which goes along with each scenario, see About This Scenario.

Level 5 (Scenarios menu)

PSoup offers six levels of increasing sophistication and complexity. The six levels also offer canned scenarios with pre-determined goals, and explanatory material. The scenarios listed under the Level 5 menu are those new scenarios added after passing the fourth Level Achievement Test. The Scenarios menu allows you to load only those scenarios which are at a

level of achievement equal to, or less than, your current level. To learn more about the Level Achievement Tests (LATs) see Level Achievement Test dialog. To learn more about the explanatory material which goes along with each scenario, see About This Scenario.

Level 6 (Scenarios menu)

PSoup offers six levels of increasing sophistication and complexity. The six levels also offer canned scenarios with pre-determined goals, and explanatory material. The scenarios listed under the Level 6 menu are those new scenarios added after passing the fifth Level Achievement Test. The Scenarios menu allows you to load only those scenarios which are at a level of achievement equal to, or less than, your current level. To learn more about the Level Achievement Tests (LATs) see Level Achievement Test dialog. To learn more about the explanatory material which goes along with each scenario, see About This Scenario.

At Level 6 PSoup is fully enabled. There are no additional LATs.

Evolution Menu

ReSeed command (Evolution menu)

The ReSeed command is used to re-start the scenario. All toggles are reset to the Scenario's default positions. It is as if you are running the scenario for the first time.

Pause/Stop command (Evolution menu)

Use this command to stop the PSoup clock and enable all of the sub-menu items and toolbar buttons. PSoup executes in background mode on low priority when a scenario is being run. This means that it is possible to start a scenario, then minimize the program and load some other application, say a word processor, and use it in the foreground. PSoup will only use unused computer cycles. However, if a user were to execute a command in foreground mode while a scenario was running in background (say start a new scenario, or toggle a display panel on or off) there would be a collision between the PSoup foreground and background processes. To avoid this, all foreground actions (except the Pause/Stop action and the Draw process) are disabled while a scenario is running.

Some scenarios take a long time to draw (i.e. those with a large number of densely-packed cells). Drawing is done in foreground mode, and all other actions are put on hold while drawing takes place.

While paused, you can tinker with the scenario and even re-use the bugs (e.g. you might want to adjust the rate of distribution of energy in an open bowl of soup, or change the rules of chemistry). Be sure you don't change the 'number of bugs at start' (that causes the scenario to reset) to time 0. Be sure you select 'Reuse Bugs' rather than 'Generate new bugs'. You can also tinker with the bugs' genes, and/or re-position them.

When you are done tinkering, re-start the scenario with the 'Go' command.

Or, you can select a new scenario, build your own scenario, or try the LAT.

Go command (Evolution menu)

When you select a new scenario, load a scenario from file, tinker with a scenario, or otherwise use the various PSoup commands, the bowl of PSoup is dormant. To start the PSoup clock, and start the evolutionary action, issue the Go command, or use the Go button on the Options toolbar. For more about the G0/Pause/Stop commands, see the Pause/Stop command

The 'Go' command has several forms. 'Go' itself causes the clock to start running until all bugs are dead. If all of the bugs continue to live, then the clock continues to run. If it clocks more than 30,000 PSoup days, it rolls over and restarts at 1 PSoup second and continues. You may, additionally, advance the clock by one second, by one bug, or, even, by just one life function of one bug.

Advance By One Second command (Evolution menu)

Use this command, in place of the Go command, to start the PSoup clock ticking, but only for one tick of the clock.

Advance By One Bug command (Evolution menu)

Use this command in place of the Go command to start the action in the bowl of PSoup. Note, however, that PSoup will advance by one bug's turn only. If there is only one bug in the bowl of PSoup, this is equivalent to an advance by one second. At the end of a second, the last bug to take a turn is displayed in the Current Bug genetic profile. To see the first bug's profile, advance

by one bug. To see each bug in turn, advance through the list of bugs one bug at a time.

You will note that, when you have scrolled part way through the list of bugs, many of the other menu items and control bar buttons are disabled. These are for commands that can only be issued ‘between seconds’. To re-enable these commands, use Advance by One Second to finish the processing for the complete second.

Advance By One Function command (Evolution menu)

Use this command in place of the Go command to start the action in the bowl of PSoup. There are five Life Functions that a bug executes during each turn: Sensing, Moving, Feeding, Reproduction and Dying. When this command is issued PSoup will advance by one Life function, i.e. a fraction of one bug’s turn only. If the Life Function panels are toggled on, then you will see the details of the bug’s changing status and the decisions being made, one function at a time. If the panels are turned off, the command will seem ineffective, but the action is happening, one function at a time, invisibly to the user. You will note, if you issue the command often enough (seven times per bug) that the PSoup clock is, in fact, moving forward a function at a time.

You might wonder, why seven commands when there are only five life functions. The Life Function show before and after status for each bug for each function. This requires six stops to view before the first, and after the last. There is a seventh step required to flush the data from the previous bug and load the data for the current bug.

You will note that, when you have scrolled part way through the functions, and part way through the list of bugs, many of the other menu items and control bar buttons are disabled. These are for commands that can only be issued ‘between seconds’. To re-enable these commands, use Advance by One Second to finish the processing for the complete second.

Go Slow Mode (Evolution Command)

This has the same effect as the ‘Go’ command, but it is regulated to a maximum speed of 3 PSoup seconds per real second. This allows a user to follow the action as the bugs cruise around the screen.

Options Menu

Display Average Bug command (Options menu)

Use this command to cycle the Average Bug display through all of its available panels, or to turn its display off. There are at most three such panels.

Display Current Bug command (Options menu)

Use this command to cycle the Current Bug display through all of its available panels, or to turn the display off. There are at most three such panels.

Display PSoup Status command (Options menu)

Use this command to toggle the PSoup Status panel on or off.

Display Scenario Legend command (Options menu)

Use this command to cycle through the available PSoup Legend panels, or to turn them off.

Display Event Counts command (Options menu)

Use this command to toggle the PSoup Event Counts display panel on or off.

Use the 'Zero Event Counts' to return all counts to zero and clear the Event Counts Panel.

Each type of count will only appear in the panel is it is other than zero. Before each run, the event counts panel is automatically zeroed, and appears as empty. When the first second of the new run has been executed, event counts should start to appear (unless there have been no recordable events). Event counts accumulate over time unless manually zeroed. When you zero all counts, the panel clears.

Display Family Tree command (Options menu)

Use this command to toggle the Family Tree display panel on or off.

Display Energy Chart command (Options menu)

Use this command to toggle the Energy Chart display panel on or off.

Display Life Function Panels command (Options menu)

This command will cause the display of the five Life Function Panels:
- Sensing function.
- Moving function.
- Feeding function.
- Reproduction function.
- Dying function.

All genetically derived behavior must, of necessity, be reflected in the life functions of the bug. If not, they play no role in the evolution of the bug. (A bug's color is a characteristic which, as an example, plays no role in the bug's life functions).

Those changes in the behavior of a bug which are derived from changes in its genes are said to be phenotypic.

If you advance PSoup by the '1 Second' button, all Life Function panels for all bugs will flash slowly before your eyes. Slowly, but too fast to read. If you advance PSoup by the '1 Bug' option, all five panels will be updated and held in front of you for your viewing pleasure. If you

enjoy the drama of watching the bug's turn unfold one arduous step at a time, then we have the button for you. Use the '1 Function' button to advance PSoup by one Life Function only for the current bug only. Most Life Function panels have a before and after section. For example, in the Moving Function panel, before the move, you can see where the bug is, its surroundings, and its direction. The Current Bug profile will be updated with the most current information respecting the Palmiter genes and any movement premiums applied due to sensing of friends or foes. The decisions facing the bug, and the means available to the bug to respond, will be available in various panels, if you know how to find and interpret them. Then, press the '1 Function' button, and the bug will make its move. The folly, or the wisdom, of the bug, will be displayed.

Similarly, each of the five panels provides a before and after snapshot of the bug's status for each of the five Life Functions.

Display Population Profiles command (Options menu)

This command invokes the display of population profile charts, one for each gene or gene type or capability type.

A population profile chart identifies the (groups of) alleles of each gene existing in the current population, and draws a bar chart showing the percentage carrying each allele. For C1 and C2 genes, every bug carries one allele of each gene, so the percentages necessarily add up to 100%. For C3/C4 genes, the percentage of the population carrying any one GType or CType will vary, so the percentages will not add to 100%.

The charts are self-scaling. If only one allele exists, a suitable range of values around the existing value will be chosen, and the bar graph with a single bar will be displayed. If two alleles exist, two bars will appear at distant ends of the range. If three or more alleles exist, then three or more bars will be displayed.

Activate Readings of Allele Counts/Values command (Options Menu)

The allele charts are two dialogs which pop up and present tables of readings. The readings are taken automatically (a) when the automated readings are first activated via this command, and (b) every RAT seconds (at most 15 minutes apart) until the readings are deactivated again via this command.

An allele is a gene with a specific strength. For example, if gene0 (F) of the Palmiter genes has a strength of 1, that is an allele. If several bugs all have gene0 Palmiter genes with the same strength, they are said to possess a common allele. If another bug has a gene0 Palmiter gene with strength 2, that is a different allele.

There are times when you want to know how many alleles of a specific gene exist in the population of bugs. This is what the allele counts chart tells you, plus it provides a historical perspective.

There are times when you want to know what the average value of a gene is over time. This is what the allele values chart tells you. The data for the allele values chart is drawn from the 'Average Bug' genetic profile, so the data is not unique to this panel, but the historical perspective is.

These charts complement the Population Distribution charts which provide graphic representations of the distribution of the point-in-time values of all current alleles of a gene.

Note that, if two different Palmiter genes (e.g.) have the same strength, they are nevertheless considered to be two different alleles, as they have a different purpose which is determined by their position in the chromosome. It is considered that two genes constitute two instances of the same allele only if they occupy the same position in the same type of chromosome, and if they

have the same strength.

When a reading is taken PSoup does the following:

For the allele counts chart -
- for each of the nine Palmiter genes, determines how many alleles exist;
- computes the total number of Palmiter alleles;
- for each of the six regulatory genes, determines how many alleles exist;
- computes the total number of regulatory alleles;
- for each of the 26 capability gene types (GTypes), determines how many alleles exist;
  - for the fight sub-components;
  - and for the flight sub-components;
- computes the total number of capability fight alleles that exist, the total number of capability flight alleles, and the grand total number of capability alleles;
- computes the percent change in the total allele count since last reading, and the total percent change since the first reading in this series of readings. Note that only the most recent thirteen readings are kept but the total percent change refers to the change since the first actual reading when the ability to take readings was first turned on.

For the allele values chart -
- for all gene types in all chromosome types, a record is made of the genetic profile of the average bug, for later reconstruction and analysis.

A reading is taken when the allele readings are first activated. Based on the average value of the RAT gene in the population of bugs, a reading is taken every RAT seconds thereafter, or every 900 seconds (15 minutes), whichever is less. RAT is used to increase the rate of readings in those cases in which bugs reproduce quickly. On average, a new generation of bugs appears every RAT seconds and the allele counts and values will have changed.

The charts hold a maximum of thirteen readings. When the chart is full, the oldest readings are discarded to make room for the new ones.

Display Allele Counts command (Options menu)

This command activates the allele chart which displays up to thirteen recent readings of counts of alleles.

If all mutation rates are set to 0, then the count of alleles should decrease, or, at best, remain steady. A decrease in the count represents the depletion of the gene pool. An increase in the allele count indicates the development of a richer gene pool.

The user can optionally cause the display of a line graph of the total allele counts. There are four lines in the line graph. The bottom line shows the count of C3 Flight alleles. The next line is stacked on the first. The distance between the first and second line is the count of C3 Fight alleles. The total distance from the axis to the second line is the sum of the C3 Flight and C3 Fight allele counts. Similarly, the third line indicates the C2 allele counts, stacked on top of the previous two counts. Finally, the fourth line indicates the C1 allele counts, stacked on top of the previous three counts. This fourth top line therefore is the total historical allele count.

The 'HowTo' button causes the display of a brief instruction on the use of the allele counts chart.

The 'Display Chart' button causes the display of up to thirteen readings (the most recent readings) taken contiguously. Note that if the data collection capability is turned off, all data is discarded. When the data collection capability is turned on again later, a fresh slate of readings is started.

The 'Display Graph' button causes the display of the line graph.

Display Allele Values command (Options menu)

This command activates the allele values chart which displays up to thirteen recent readings of values of alleles.

This is a companion chart to the Allele Counts chart.

The 'HowTo' button causes the display of a brief instruction on the use of the allele values chart.

The 'Display Chart' button causes the display of up to thirteen readings (the most recent readings) taken contiguously. Note that if the data collection capability is turned off, all data is discarded. When the data collection capability is turned on again later, a fresh slate of readings is started.

The 'Display Graph' button causes the display of up to 48 different line graphs. The user may flip through the various line graphs by clicking on the spin button by the edit window. Each graph represents a line from the chart, or several lines from the chart. Here is a list of the graphs available:

- Generation - The average generation of the population of bugs at the time of the reading.
- Number of Bugs - The size of the population of bugs at the time of the reading.
- Individual C1 Genes 0 through 8 - The average strength of each gene over time.
- All C1 Genes - Showing the average value of all C1 genes scaled to the same scale.
- C1 Penalty - Showing the recent history of the C1 Penalty.
- Individual C1 Genes 0 through 8 as percentages of total C1 strength - The average contribution of each gene to the bug's search pattern.
- All C1 Genes as percentages of total C1 strength - Showing the average contributions of all C1 genes scaled to the same scale. All percentages add to 100%.
- Individual C2 Genes 0 through 5 (DAT, DET, RAT, RET, EPM, EPB) - The average strength of each gene over time.
- All C2 Genes - Showing the average value, normalized. All values are divided by the default starting value for the gene in a pristine bug. This provides a view of the relative trends.
- C2 Penalty - Showing the recent history of the C2 Penalty.
- Stamina, Fertility and Agility - Each in a separate chart, showing the recent history of each.
- Individual C3 CTypes (capability types) - One graph for each of the nine capability types sight, smell, hear, taste, touch, xover, ovule, herd and think. Each graph shows three lines, fight, flight and total.
- All C3 Fight CTypes - Showing the relative strengths of the fight components of the capability types over time, all scaled similarly.
- All C3 Flight CTypes - Showing the relative strengths of the flight components of the capability types over time, all scaled similarly.
- C3 Penalty - Showing the recent history of the C3 Penalty.
- Complexity - Showing the history of the average complexity of the population over time.

The 'Display Profile' button causes the display of the three genetic profiles (C1, C2 and C3) for the selected reading. The user has the opportunity to flip through all thirteen sets of profiles looking for general trends in the average phenotype.

Display Cluster Analysis Report ... (Options menu)

The user should read the article in PSoup Theory and Practice on 'PSoup and Cluster Analysis'.

There are a number of controls in the this dialog which the user can use to control the processing and to explore the nature of the clustering of the genetic material in the space of possible genetic variations in PSoup. The controls are grouped into three groups:

- Finding options;
- Selecting an Option; and
- Reviewing the Selection.

The 'HowTo' button displays a brief description of how the cluster analysis feature works.

The 'By Bug' radio button serves to reset the dialog back to the start. The 'By Chromosome' feature has not yet been implemented and is inaccessible. Note that, if you have completed the processing of a large population, and it has taken a long time, do not click on the 'By Bug' button. It will discard the work already done and force you to re-do it.

The 'OK' button discards all work and returns PSoup to normal operational mode.

Finding Options

The 'Build Distance Tables' builds an N x N table of distances between the bugs in the population (where N is the total number of bugs currently alive). The distance function is the 'Manhattan Block' distance function in which each gene type is viewed as a dimension and distances in each dimension are added together. D = a + b. The normal formula (D = ((a^2) + (b^2))^0.5) is NOT used. This is similar to the distance a pedestrian must walk to get from place to place in a city, as opposed to the distance a crow must fly. Distances are measured in 'standard mutations' which is a change of exactly one in the exponent when the strength is written as a power of 2. Two genes are one unit apart if their strengths, when written as powers of 2, differ by one in the exponents. The table is half of a square, to reduce redundancy. The distance between a bug and itself is not recorded. So the distance table has N(N-1)/2 entries. If a bowl of PSoup has a full complement of 500 bugs, there will be 124,750 entries in the distance table.

The 'Start Analysis' button sets in process an involved series of events in which PSoup searches for the best clustering of the bugs. PSoup identifies up to 20 unique reference distances, and for each reference distance it produces a best set of clusters. (See the above named article for details on the process). The length of time required for this analysis climbs exponentially as the number of bugs (N) grows. The process cannot be stopped once started. It is suggested that analysis of large populations be done in overnight runs. Play with the capability on a variety of small populations before attempting a large run. Note that the results of a large run are discarded when (a) you click the 'OK' button or (b) you click the 'By bug' button.

The analysis automatically selects a best option and jumps straight to 'Review the Selection' mode. The user can of course select another option.

The 'Display Log' button causes the log of discovered options to be presented to the user in the edit window. The options are:
- up to 20 reference distances; and
- for each reference distance, as many iterations of the clustering algorithm as were required to achieve a stable 'best' option for that reference distance.

Select an Option

When a user adjusts the spin buttons for reference distance or for iteration, then PSoup discards the currently selected option under review in 'Review the Selection' mode and goes into 'Select an Option' mode.

The 'Reference Distance' spin button allows the user to select one of from up to 20 reference distances determined by PSoup to be representative. Each such reference distance is identified in the log, and represents a real distance between bugs in the population.

The 'Iteration' spin button allows the user to select a target iteration at which the clustering

algorithm will stop. This is preset to the last iteration when the reference distance is chosen, but the iteration spin button can over-ride this. There are circumstances when this is advisable. If the last iteration appears to be degenerate for a given reference distance (actual clusters is 1 but expected clusters is greater than 1) then a previous iteration will be more informative. Similarly, if the number of clusters varies radically on the way to a 'best' cluster, you might inspect the intermediate iterations to determine why.

The 'Re-compute Iteration' button causes PSoup to re-apply the clustering algorithm for the selected reference distance up to the target iteration, stopping at that iteration. This button causes a mini log to be written and displayed. The user can use this temporary log to check that the correct iteration was completed. The main log, showing all reference distances and all iterations is always available, and it contains the mini log as a subset. The mini log is discarded at the next action by the user. This button also causes PSoup to switch into 'Review the Selection' mode.

Review the Selection

In PSoup parlance, a selected option is in fact a ragged table of bugs in which each line is a cluster and each cluster contains a number of bugs. This table may or may not have a different structure depending on the reference distance used to produce it, and the number of iterations of the clustering algorithm executed against the table. There are two ways a user can review the contents of the table. A report detailing the statistics about each cluster can be produced and reviewed. Or, a set of three genetic profiles can be produced for each cluster and viewed.

The 'Build Report' button causes PSoup to prepare a report about the contents and structure of each cluster in the selected option.

The 'Display Report' button causes the report to be displayed for review. The report contains simple descriptive statistics for the population in general, and similar statistics for each cluster in the table.

The 'Cluster No' spin button allows the user to select a cluster from those in the current table to be presented as a panel of three genetic profiles.

The 'Display Profiles' button causes PSoup to display the currently selected profiles.

Note that, for technical reasons, PSoup sometimes erases part of the profiles. This is because PSoup uses two windows for display of a variety of reports and profiles, and some combinations of buttons cause temporary over-writing. Clicking on the Display Profiles button will restore the image.

Display Pairing Square (Options Menu)

This command calls up a dialog which is used to initiate and manage processes which collect data and present them for analysis.

The routines provided here are designed to be used specifically with Scenario 5C - Mendel's Peas, but they will work with any scenario which contains diploid (i.e. OVULE-enabled) bugs.

To use this feature, simply turn on data capture, then allow the bowl of PSoup to run until up to 13 generations have passed. Then return to the Pairing Square dialog and examine the data collected.

Or, you can simply use the Pairing Square processes to take an immediate reading of the current population of bugs, and discard the reading when you are done with it.

Whether you take one reading on the current population only, or several readings over several generations of bugs, each reading presents 15 charts. There is one chart for each of the nine Palmiter (C1) genes, and one chart for each of the six Transport (c2) genes.

Each chart produces a listing of all of the alleles found for the gene, and then presents three types of pairing squares for the alleles. You are presented with a square table showing which allele is dominant over which allele, on a pair by pair basis. You are then presented with a square table showing the count of every possible pairing of the alleles. Finally, you are presented with a square table of percentages, indicating what percentage of all pairings fall into each category.

These tables can be used to trace the emergence or disappearance of alleles in a diploid population, the development of dominant and recessive characters, and the effects of Mendelian pairing of hereditary factors.

For more information about the use of this feature see the article About Pairing Squares in PSoup Theory and Practice.

Bifurcation Diagram ... (Options menu)

This command calls up a dialog which is used to initiate and manage the processes required to produce a bifurcation diagram showing the incidence of chaotic processes in PSoup. For more information about bifurcation diagrams, see the article 'PSoup and Chaos' in the 'PSoup Theory and Practice' section of this Help System.

The routines provided here are designed to produce a bifurcation diagram specifically for Scenario 4F. They have been generalized such that they will work under more general circumstances in any scenario in which the algae distribution mode set to 'Variable'.

Production of one bifurcation diagram will take a long long time. Be prepared to wait. You may run it at night, saving the interim results each morning, and reloading and continuing the runs each evening. On a 120 MHz machine, such a run would require 3-4 nights.

In PSoup, our bifurcation diagram is a scatter plot of readings of the 'Number of Bugs' taken regularly once per seasonal cycle for the first ten cycles of each of a pre-determined number of runs. That's X runs, 10 readings per run, for a lot of readings. Each reading, on average, requires four PSoup hours. So, to produce one good bifurcation diagram requires approximately 40X PSoup hours, and a minimum recommended number for X is 15. Typically, the points on the scatter plot lie on a line which bifurcates (splits in two) at that value when turbulence and chaos intrude themselves into the systems processes.

If PSoup were a deterministic system, we might expect a precise bifurcation diagram with carefully drawn branches. However, PSoup is a probabilistic system, and probabilistic systems are rarely precise, and typically messy. For those users who are not familiar with bifurcation diagrams, it is suggested that you study a few examples taken from deterministic dynamic systems first. For those who are familiar with bifurcation diagrams from deterministic systems, be prepared to work for it. A single set of runs may take days. The result may not be easily recognizable as a bifurcation diagram. That's life.

A seasonal cycle consists of four seasons. So, if the 'Season Length' is set to 1000 seconds, then a full cycle is 4000 seconds long. We must collect a set of 10 readings at each value of Season Length as we vary the value of this variable over the appropriate range.

A Caution

This feature of PSoup is not for the impatient.

If PSoup were a deterministic system, we might expect a precise bifurcation diagram with carefully drawn branches. However, PSoup is a probabilistic system, and probabilistic systems are rarely precise, and typically messy. For those users who are not familiar with bifurcation diagrams, it is suggested that you study a few examples taken from deterministic dynamic systems first. For those who are familiar with bifurcation diagrams from deterministic systems, be prepared to work for it. A single set of runs may take days. The result may not be easily recognizable as a bifurcation diagram. You might have to try several times to get a clearly recognizable result. If good fortune frowns upon you, you may never produce a recognizable result. That's life.

How do we produce a bifurcation diagram?

1. Select a suitable scenario. The algae distribution mode MUST be set to 'Variable' or the Options menu will not give you access to the 'Bifurcation Diagram' dialog. Note that the results will be dependent on the configuration of the bowl, the rules of chemistry, and the design of the environment, but WILL NOT be dependent on the design of the original inhabitants. In all subsequent runs after the first, the original denizens are replaced by blind cruisers at startup.

2. Immediately go into the Bifurcation Diagram dialog and enable data collection. If you have advanced PSoup by even a fraction of a PSoup second you will not be able to do this. If you have (accidentally) advanced, reseed.

3. Read the "How To' screen.

4. Adjust the 'Season Length', the season increment, and the number of runs. The full set of values of season length for each of the intended runs is presented. Be sure you agree with this list.

5. Click OK to exit the dialog.

6. Click 'Go', then wait for it to complete the indicated number of runs. If, in one of the runs, all bugs die, PSoup will move on to the next Season Length and start. Each run will be entitled with respect to the run number and latest reading taken.

7. If you need to interrupt a run, press 'Stop' and then save. Remember the filename used to save the run. Later, you can reload the file and continue the run. A single run might require several days of processing. It can be done in overnight runs using this technique.

8. If at any time (after run 3) you wish to view the partially completed bifurcation diagram, interrupt the processing as per step 7 above and SAVE THE FILE. Then open the 'Bifurcation Diagram' dialog and click on 'Display Graph'. If at this point you click on 'Disable', you will lose all collected data and must restore it from the saved file.

9. When the last run is complete, the complete set of data can be viewed as per step 8 above.

Note that, in Scenario 4F all mutations are turned off. In this way, the consistency of the system is maintained between readings and between runs. This gives us results much closer to the mathematically pure ideal bifurcation diagram as seen in most textbooks on chaos. If you allow mutations, the bifurcation diagram is much more complex and difficult (i.e. virtually impossible) to interpret, but also much more typical of natural biological systems which develop under both turbulent and evolving circumstances.

It is understood that the current tools for the production of bifurcation diagrams are barely flexible enough, and efficient enough, to be instructive. A full set of useful capabilities will have to wait for a later release of PSoup.

Sounds command (Options menu)

Many key events such as the birth or death of a bug, the eating of a colony of algae, or other such events, are heralded by a unique sound. Over time you may come to know the sounds and listen for them to better understand what is happening within your bowl of primordial soup. However, over time, you may also come to find these sounds distracting, or ever irritating. Especially if you are doing an overnight run and the computer is within earshot of your bedroom. The Sounds command allows you to suppress the sounds.

Control Panel command (Options menu)

The Control Panel is a simple dialog box which allows a teacher to pre-set the 'Achieved Level' for the program before passing control over to a student, should the teacher wish to do so. The Achieved Level is the level of the LAT passed most recently (and saved).

Here is how it works. When PSoup is loaded, it is automatically in teacher mode. All Levels of PSoup are accessible. A teacher or student may immediately access any and all features of PSoup. However, if the teacher goes to the control panel, sets a particular level, and left-clicks on the button for 'Student Mode', then PSoup immediately freezes itself at the assigned level. The only way to return to teacher mode is to terminate the program and restart it.

Loading a previously saved scenario will change the Level Achieved, but it will not change the Teacher/Student Mode. This means that, as soon as a student has passed an LAT he may choose to save a scenario at the new higher Achievement Level, thereby removing the need to pass the LAT again in the future. Or, alternatively, the teacher can restart the software and set the higher level before turning the student loose again.

In this way, a teacher in a classroom setting may have some control over what the students see and are able to do. However, it is also recommended that both teachers and students first use PSoup in student mode and learn its features from the ground up, one level at a time. As a student progresses through the levels, the scenarios and LATs work together as a tutorial to introduce the PSoup's program features and their applicability in bite-sized chunks.

Tinker Menu

Tinker With The Environment command (Tinker menu)

This Tinker command allows you to make modifications to the environment of your bowl of PSoup. This command invokes a tabbed dialog which, when fully enabled, has two tabs:

The first tab is labeled 'Environment'. For more details about the possible modifications, see Environment Tab .

The second tab is labeled 'PSoup Mutation Rules'. For more details about the possible modifications to the mutation rules, see Mutation Rules Tab.

Environment Tab (Tinker With The Environment dialog)

This is the most complex set of controls in PSoup. The controls are inter-related in many ways. PSoup is designed such that a user cannot do something nonsensical or impossible. The controls change as you modify other controls to limit the offerings to those which are consistent with the settings of the other controls. If there is a control shaded out (de-activated) but visible, it means you have chosen another control value which has pre-empted any option here. Only the key relationships between controls are documented below. It is up to the user to experiment with this tab and discover the possibilities.

Scenario Title - The user can change the name of the scenario. The tab opens with the name of the current scenario here, but with the letters '**UD**' inserted at the front, to indicate it is a **U**ser-**D**etermined scenario. Give the scenario a name, and then save it to disk. You may then load it and run it as often as you wish.

PSoup Bowl Size - There is a variety of sizes which may be selected. A very large size is slow, but offers plenty of room for a lot of bugs. Be wary of overflow on number of bugs with a large bowl. PSoup does not support more than 500 bugs. An overflow condition is noted by an OBu event in the event counts panel. A small bowl is fast but simple. Some terrain types (Desert Oasis Ecosystem, Cascade) will not work well with small bowls. Of course all bowls occupy the same area of the screen. When we say a large bowl, we mean more cells are fitted in by making the cell size smaller, and the cell density greater.

Terrain Type - There are four possible terrain types. Normal terrain type is just a plain old bowl of PSoup. Desert Oasis Ecosystem terrain type is a plain bowl of PSoup with a little oasis of greenery in one corner. Algae is distributed to the oasis first, and the rest of the bowl afterwards. Terraced paddies are side-by-side. Algae is distributed to the paddies, one after the other, from left to right. The analogy of the Primordial Soup is abandoned here in place of a mountainside. The leftmost paddy is high up a mountainside and receives a lot of rain. The rightmost paddy receives only the rain left over that has not fallen on the upper paddies. Independent paddies are side-by-side, but bugs are unable to jump between them.

Wrap - Wrap is either on or off. When wrap is off, the bowl of PSoup is a rectangular plane. When wrap is on, the bowl of PSoup is a toroid. When the terrain type is a Paddy type, then wrap on makes it a cylinder, rather than a toroid.

Number of Paddies - Just what it sounds like. For a terrain type that allows paddies, the user can determine how many paddies there will be from the options presented. You will note that the number of paddies is always a divisor of the width of the bowl. All paddies are therefore of precisely the same width.

Probability of Paddy Jump (POPJ) - If the terrain type is 'Independent Paddies' then POPJ is 0.0.

If the terrain type is Normal or Desert/Oasis the POPJ is irrelevant. For Terraced Paddies, the POPJ is the probability, on any given turn, having attempted to move from the left side of a barrier to the right side of a barrier, that a bug will successfully jump from the left paddy to the right paddy. If this is set at 1, all bugs will quickly move to the rightmost paddy where they will starve. If this is set at 0, you effectively have independent paddies. Low values create a terrain in which variant sub-species may develop and thrive in harmony.

Number of Bugs at Start - Just what it sounds like. But there are complications. There are a variety of restrictions on the number of bugs that a scenario can support at startup. These restrictions are handled automatically by PSoup. As you make various selections, if you see that this number has changed, you've crossed one of the limitations. For example, if you choose terrain type 'Desert/Oasis' and Bug Distribution Mode of 'Oasis First' then the number of bugs at start MUST be less than the number of cells in the oasis. There are six such restrictions which are not listed here.

Bug Distribution Mode - Options are To Paddy 0, Across All Paddies, Cascade and others. PSoup will offer those modes which make sense for the terrain type chosen.

Bug Re-use Mode - PSoup offers to let you generate a slate of new bugs, with default genetic codes, or to re-use the existing bugs, if you want to preserve the genetic codes of the current bugs but want to put them into a new scenario. You can re-use existing bugs only if you have set the number of bugs at start to equal the number of bugs currently in the PSoup. This may restrict the types of scenarios into which you can place the bugs. For example, the terrain type "Independent Paddies" requires the same number of bugs in each paddy, so 9 bugs (e.g.) can go only into a 3-paddy or 6-paddy terrain.

Bug Coloration Mode - This does not affect the bug's evolution, but does make some aspects of bug-watching easier. 'By Gene' colors each bug according to its C1 genotype. 'By Root' colors each bug according to its association with a root bug in the family tree.

Stand Still Gene. This is the eighth Palmiter gene (counting starts at 0) and it allows sessile or sedentary forms of bugs to evolve. This set of radio button puts the activation of the Palmiter gene under user control. 'Inert' deactivates the Stand Still gene. 'Activated' does as it says.

Energy Loading (%) - This determines the amount of energy that is loaded into the nutritive mud ready for distribution as algae. A value of 40% (e.g.) means that enough energy will be loaded into the mud to cover 40% of the bowl of PSoup. Note that there is additional energy in the bugs, so the total amount of energy in the bowl of PSoup is the sum of the energy in the mud plus the energy in the bugs.

Supercharge It/Discharge It - These two buttons allow a user to raise the scale such that up to 300% energy loading is possible. This is available only at higher levels.

Algae Distribution Mode - This is similar to the Bug Distribution Mode but with a significant difference. The bug distribution mode is used initially to distribute the bugs, and after that it is not used again during the run of the scenario. The Algae Distribution Mode is used after every turn of every bug (to attempt to place a single colony of algae) and at the end of every PSoup second (to attempt to place all of the rest of the energy still in the nutritive mud). Several options are available. Those which are relevant to the terrain type are offered. If you choose 'Steady' or 'Variable' (not implemented yet) modes, then you will be offered an opportunity to modify the energy distribution rate.

C1 Penalty - This section allows the user to enable or disable the use of the C1 Penalty during the run of a scenario, and allows the user to increase or decrease the size of the divisor used to scale the impact of the penalty. If the divisor is too low, all mutations will be deadly. If the divisor is too high, the penalty will be of little effect.

C2 Penalty - This section allows the user to enable or disable the use of the C2 Penalty during the run of a scenario, and allows the user to increase or decrease the size of the divisor used to scale the impact of the penalty. If the divisor is too low, all mutations will be deadly. If the divisor is too high, the penalty will be of little effect.

Nrg Unit Distribution Rate - Energy is distributed as PSoup Nrg units. 40 Nrg Units equal one colony of algae. If you choose a 'Normal' or 'Desert/Oasis' terrain type, then you will be offered an algae distribution mode of 'Steady' or 'Variable'. In either case, the PSoup energy is converted from a closed energy system to an open energy system. In a closed energy system, the total energy in the system never varies. In an open system, the energy is fed into the system at a steady rate, and leaves the system as it is used up (by movement or death of the bugs). The energy unit distribution rate is the number of units of energy per PSoup second introduced to an open system. A low value will produce desert-like conditions in which competition for resources is extreme. A high value will produce rain-forest-like conditions in which competition for algae does not really happen, and the real competition is between carnivorous bugs.

Total Energy - This section merely displays the computation of the total energy available to the system when it first starts, whether it is open or closed.

Reset Original Values - Use this button when you have lost track of what changes you've made in this tinkering session, and you want to start over again.

Reset Default Values - Use this button when you want to entirely scrap the current tinkering settings and start again with a scenario template of settings.

PSoup Mutation Rules tab (Tinker With The Environment dialog)
This tab allows a user to tinker with the basic rules of PSoup chemistry. This goes to the very heart of evolutionary processes. While competition for resources (in PSoup, Nrg) drives evolution, and interaction with the environment directs it, the mutation of genetic material provides the raw material, the variant alleles, that allows the incredible advances, and the incredibly intricate responses. This tinker facility puts an extremely powerful capability in the hands of the user, enabling the emulation of a wide variety of biological processes and situations which are of great interest to evolutionary theorists, but which cannot be controlled in reality.

This tab has three main sections as follows:
- Basic constants of PSoup Chemistry (3)
- Advanced constants of PSoup Chemistry (3) and
- Controls on genetic recombination (3).

In each of these sections there are three controls that a user can change, for a total of nine. Each section is described in detail below.

Basic Constants of PSoup Chemistry

The three constants addressed in this section control the mutation of simple genes. You recall that a simple gene has two components, a base number (1 < b <= 50) and an exponent (-150 <= e <= +150). The strength of the gene is computed as the result when 'b' is raised to the power 'e'. All C1 genes start with a base value of 2 and an exponent of 0 giving a strength of 2^0 or 1 (unless, of course, you have used the TinkerBug Wizard to modify them).

Mutation in Level 1 - Special case

In Level mutation processes are not under the control of the following rules. These rules start to be applied in Level 2, and at all higher levels as applicable. In Level 1 only eight genes are not inert, that is the Palmiter genes (excluding the Stand Still gene). When fission occurs, one of the

genes is selected to mutate, and the exponent rises or falls by exactly one. It is 100% certain that one and only one gene will mutate (in any given daughter). There is exactly a 12.5% probability that a specific gene will mutate. There is a 50/50 probability that the exponent will rise/fall by a value of 1. These rules of chemistry replicate the approach used by DR Palmiter in his original concept. Starting at Level 2 PSoup has implemented a generalized approach which encompasses the above but puts it under user control.

Gene, To Mutate, Or Not?

Probability of Mutation - The first basic control determines the probability that a gene will mutate or not. The default value of this control is 0.125. I.e. there is a 12.5 % probability that each simple will mutate. PSoup generates a random number between 0.0 and 1.0 FOR EACH gene, and if the number is less than this threshold value, the gene mutates. On the surface this looks like the rule used in Level 1, but there is an important difference. There is no certainty that a gene will mutate, and it is remotely possible that all genes will mutate. If this value is set to zero, no genes will mutate, and all evolution will be via recombination of existing genetic material. If this value is set to one, every gene will mutate every time fission occurs and there will be chaotic instability in the gene pool.

Probability of Exponent Mutation - The second basic control determines whether the base or the exponent will mutate. One or the other changes in any given mutation, but not both. The default is 100.0 % probability that the exponent will mutate. If the user wants either base or exponent to mutate, set this to 50%. The default replicates the approach used in Level 1. Mutation of the base is required to achieve a fine level of graininess for the C2 genes which start with very high exponents. If the user wants only the base to mutate, set this control to zero. This is not advisable, but possible. Mutation of the base involves a formula which has singularities at 1 and -1, so mutation of any base values between -2 and 2 is proscribed. Setting this control to zero has the potential to dampen most or all mutations. From the perspective of the bug, it is immaterial whether the base or the exponent mutates, the new strength of the gene will be the same in either case. However, if the base mutates, a whole new range of allele values is made possible for future generations. The formulae have intentionally been developed to eliminate any bias that might be introduced via this control. If a user wishes to introduce bias, see the next control.

The population profiles behave differently depending on the value of this control. With a value of 100%, the 'population profiles' graph the distribution of the values of the exponents. With any other value, the population profiles graph the distribution of the values of the strengths.

Probability of Mutation Upwards - The third basic control determines whether the value of the base/exponent rises or falls. The default value of this control is 50%. This is analogous to a bias within a real ecosystem that might favor certain types of mutations over others. The user may choose to produce more mutations that have reduced strength, or more mutations that have increased strength, and see what the impact is on course of development of the bugs.

Advanced Constants of PSoup Chemistry.

We now address the second group of controls, those which control the mutation of C3 and C4 capability genes. These controls come into play only at Level 4 and higher, when the C3/C4 genes are no longer held inert. Again there are three controls.

Probability of mutation of complexity via addition or deletion of genes - This first control is used to determine whether or not the complexity of the Tinker Bug a will be modified. The default value is 0.125. PSoup generates a random number between 0.0 and 1.0. If the number is less than the threshold value, then the complexity is modified, going either up or down by 1. Complexity is equal to the number of genes in the chromosome, so a gene must either be added or deleted.

Probability of Added gene - The second control is brought into play only when it has been determined that complexity must change. It determines whether a gene will be added or deleted. If it is set to 1, mutation of complexity will always result in the addition of a gene. If it is set to 0.0, mutation of complexity will always result in the deletion of a gene. A value of 0.5 results in an unbiased mix of additions and deletions. Then implicit evolutionary pressures will select for longer or shorter lists of genes.

Controls on Genetic Recombination

There is one control in this section. When bugs reproduce via XO-Fission or via Birth, spermatic genetic material is mixed with the genetic material from the mother under the control of these parameters.

Probability that a gene will cross over during an XOver event. - An XOver event occurs as part of XO-Fission. The mother's genetic material is mixed with the father's, based on this rule. For each chromosome the genetic material is viewed as a chain of genes. A suitable break point in the chain is determined. Then, chromosome by chromosome, the die is rolled and a random number is chosen. If the random number is less than the threshold value (0.125 is the default threshold value) then the genes from the father and mother are mixed. The daughters receive (e.g.) the first part of the chain from the mother, and the second part of the chain from the father. For each of the four chromosomes there is a 12.5% chance that crossover will happen, and an 87.5% chance that it will not.

An example of Cross-over

Mother				Father			
C1	C2	C3	C4	C1	C2	C3	C4
M1 M2	M1 M2	M1 M2	M1 M2	F1 F2	F1 F2	F1 F2	F1 F2

C1, C2, C3 and C4 are chromosomes. M1/M2 are segments of genetic material from the mother's code. F1/F2 are segments of genetic material from the father's code.

Pseudo-random number generator output:
- C1 - No crossover
- C2 - Crossover
- C3 - No crossover
- C4 - No crossover

Daughters Get			
C1	C2	C3	C4
M1 M2	M1 F2	M1 M2	M1 M2

For C2 only, the daughters gets the first segment from the mother and the crossover segment from the father.

Note that, since bugs are hermaphroditic, a bug which can deliver strong spermatic material to many mates will do more to spread its genes than a bug which produces two daughters by birth. Many mothers can be forced to reproduce daughters using the father's genetic material. This opens the possibility of a battle for survival between genes, in which the resource being sought is the bodies of mother bugs, and the means of success is alliance (by association in a chromosome) with a strong chromosome of aggressive sexual genes.

Reproductive Bias

This feature is intended to be used with '**Scenario 5D - Biased Reproductive Rates**', but can be used with any scenario. Three options are available, bias upwards, no bias, or bias downwards.

- No Bias has no effect.
- Selection of 'Bias Upwards' has two effects. First, it causes all of the SMELL genes in all bugs to be randomly adjusted to be between 7/8ths and 9/8ths of their former values, with a flat distribution. In other words, each allele of each such capability gene is treated as a member of a set of 100 possible alleles ranging in value from 7/8ths of the original value to 9/8ths of the original value. The original value is then replaced by one of the 100 possible values, each having an equal probability of selection. Second, it flips a switch such that all SMELL-enabled bugs exercise mating preferences when choosing a mate, and opt for a mate with a better nose.
- Selection of 'Bias Downwards' is the same as bias upwards except that the mating preference is for a bug with a poorer nose, i.e. lower strength of the SMELL capability.

Reset Buttons
There are two additional buttons. If you have tinkered with the rules, and wish to undo your current tinkering, left click on the 'Reset Originals' button. If you want to restore the PSoup default values, left click on the 'Reset Defaults' button.

Tinker With The Bugs command (Tinker menu)

This command invokes a dialog in which you select a bug with which you want to tinker, (hereinafter called the 'Tinker Bug'), invokes a Wizard which enables you change just about everything you might possibly want to change respecting the bug (which Wizard is called the TinkerBug Wizard), and, upon return from the Wizard, the option of using or discarding the altered bug.

For more information about the Tinker Bug Selection dialog see Tinker Bug Selection .

For more information about the TinkerBug Wizard see TinkerBug Wizard .

The Tinker Bug Selection dialog (Tinker With The Bugs command)

When you issue the Tinker With The Bugs command you invoke this dialog which is used to select the bug with whose genes you plan to tinker. From this dialog, you can:
- Move forwards or backwards through the list of all bugs currently alive in the bowl of PSoup, highlighting each in turn as a potential target for tinkering.
- Invoke the TinkerBug Wizard.
- Choose to either accept (Inject this tinkered bug back into PSoup) or reject (refresh this bug's original values) the changes made by the TinkerBug Wizard.
- Return to the main menu by pressing the OK or Cancel buttons (Note, these buttons only return you to the main menu, and neither button can undo a change, once it has been injected into PSoup. If you do not inject the tinkered bug, OK and Cancel both discard the changes. If you do inject the tinkered bug, OK and Cancel both accept the change.)

You may also see the details of the genetic code of the currently selected bug by using the 'Display Genes' button.

The TinkerBug Wizard (Tinker With The Bugs dialog)

When you are in the Tinker With The Bugs dialog and you press the button labeled 'Start the TinkerBug Wizard' it will invoke a tabbed dialog with many tabs. At this point, a bug has been selected for modification, and has been placed in a special holding cell. The TinkerBug Wizard then enables you to modify the Tinker Bug in a large variety of ways. Each tab enables the modification of some aspect of your chosen Tinker Bug. The following tabs are available:

- The 'Welcome/Instructions' tab - This is always the first tab available to you in the TinkerBug Wizard. It provides generic instructions on the use of the various tabs and maintains them immediately at hand.
- The 'Acquired Characteristics' tab - This tab is used to change the characteristics of the bug which are not genetically determined.
- The 'Gallery of Tinker Bugs' tab - This tab is used to quickly alter the genetic makeup of Tinker Bug to match one of several templates. You will also have the ability, in higher levels, to endow the Tinker Bug with more sophisticated capabilities.
- The 'Palmiter Genes' tab - This tab is used to tinker with the detailed genetic code in the nine Palmiter genes.
- The 'Regulatory Genes' tab - This tab is used to tinker with the detailed genetic code of the six regulatory genes.
- The 'Sensory+ Genes' tab - This tab is used to tinker with the detailed genetic code of the (up to) 200 capability genes.

The Acquired Characteristics tab (TinkerBug Wizard)

This dialog box allows the user to modify a number of characteristics of a bug which are not derived from its genotype.

A genotype is the configuration of numbers in a bug's genetic code. A phenotype is a characteristic behavior pattern that is governed by the genotype. Many genotypes may give rise to the same phenotype. An acquired characteristic is one which does not derive from genotype, and is therefore in some sense the opposite of a phenotypic characteristic. The length of a dog's tail is a phenotypic characteristic, as it play a different role in a dog's behavior depending on the length. However, if the dog's tail is bobbed, the length of the tail is acquired. There may be some dispute, from biologists, as to whether things like location or age are, in the biological sense, acquired characteristics. For the purposes of PSoup, they are. Users should refer to a good dictionary of biological terms for a better understanding of acquired characteristics.

There are six sections in this dialog.
- Energy.
- Location.
- Direction.
- Age.
- Color.
- Reproductive Mode/Status.

Energy
In PSoup all evolution is driven by the competition for resources. One resource, in fact, and that is energy. A user can modify the amount of energy in a bug in this section. If the energy is too low (below the Death Energy Threshold or DET as determined by the appropriate C2 gene) the bug will die in the first turn in PSoup. The energy cannot go above the Maximum Energy Per Bug (EPB gene in C2).

Location
A bowl of PSoup is a rectangular grid. It consists of a number of square cells arranged in rows (across the screen) and columns (up and down the screen). Each row has a number and each column has a number. The leftmost column is labeled 0. The topmost row is labeled 0. A 10x4 bowl of PSoup (e.g.) has 10 columns and four rows. The user will be able to set the column to any number between 0 and 9, and the rows at any number between 0 and 3. You can imagine that, as you change the numbers, you are physically pushing the bug around in the bowl. If you bump into another bug, or into the edge of the bowl, the TinkerBug Wizard will not let it move. You will need to go around the obstacle.

Direction
A bug can have one of eight directions, labeled as follows:
- 0 = Up.
- 1 = Up Right (UR).
- 2 = Right (R).
- 3 = Down Right (DR).
- 4 = Down (D).
- 5 = Down Left (DL).
- 6 = Left (L).
- 7 = Up Left (UL).

If you display the Life Function panels during the move of a bug, the bug's direction is displayed under the Moving function both before and after a move. The direction is also visible on the bug as a black arrow pointing from the center of the bug in the appropriate direction.

Do not confuse the direction of the bug with the C1 gene being expressed. Gene 0 of the C1 chromosome is called the Forward (F) gene. A bug point down which exercises the F gene still points down. A bug pointing down which exercises the Back (B) gene, will point Up after the move. In other words, the exercise of a Palmiter gene changes the direction, and so determines the new direction, but the it turns the bug from its old direction.

Age
A bug's age cannot be greater that the Death Age Threshold (DAT) as determined by the DAT regulatory gene in the C2 chromosome. A bug's age cannot be less than zero.

Color
In PSoup color is for the benefit of the user only, and it does not play any role in the behavior of the bugs. It is therefore an acquired characteristic. In biological organisms, color is phenotypic.

There are two color modes implemented, and a third slated for a later release. Coloration 'By Gene' uses some trigonometric formulae to mix the three standard colors of red, green and blue. With this approach, each C1 Phenotype has its own shade of color. Coloration 'By Root' uses a list of pre-selected colors and each branch of the family tree is colored with a common color.

Reproductive Mode/Status
Reproductive mode is phenotypic, not acquired. The reproductive status is acquired. In this section a user is able to change the reproductive status. To change the reproductive mode, use the Gallery tab, or the Sensory+ tab. There are three possible reproductive modes.. The potential status values vary by mode, as follows:

- Fission - Immature (age < RAT), Mature (age > RAT but Nrg < RET), and Ripe (age > RAT and Nrg > RET).
- XO_Fission, - Immature (age < RAT), XdOver (carrying cross over material), mature (age > RAT but Nrg < RET), and ripe (XdOver, age > RAT, Nrg > RET).
- Birth - Low Nrg 1 (Nrg < 0.6 RET, pre 1st birth), InHeat 1 (Nrg > 0.6 RET, Not mated, pre 1st birth), Pregnant 1 (age < RAT, mated), and At Term 1 (age > RAT, Nrg > 0.75 RET, mated, ready to give 1st birth), Low Nrg 2 (Nrg < 0.3 RET, pre 2nd birth), InHeat 2 (Nrg > 0.3 RET, Not mated, pre 2nd birth), Pregnant 2 (Nrg < 0.5 RET, mated, pre 2nd birth), and At Term 2 (Nrg > 0.5 RET, mated, ready to give 2nd birth). A bug must progress through these eight stages in order to produce two daughters by birth. The mother dies in childbirth on the birth of the second daughter.

A user can select a repro status in the drop-down list and force the bug to meet the conditions. The TinkerBug Wizard will adjust the age, adjust the energy, or apply or remove spermatic material as required to make the acquired characteristics match the profile for the requested reproductive status. The spermatic material used is derived from the mother using the 'EffectParthenogenesis()' routine. Any bug which achieves the age of maturity (RAT) and meets

all of the conditions to produce offspring via either XO-Fission by via Birth, but lacks the required genetic donation from another bug, uses EffectParthenogenesis to fertilize itself, thus avoiding an evolutionary penalty due to delayed procreation. This enables the emergence of XOver and Birth as repro modes when there are very few practitioners. Fertilization by parthenogenesis has the machinery of XOver or of Birth but the genetic value of normal fission.

The Gallery of Tinker Bugs tab (TinkerBug Wizard)

This dialog box allows you, the user, to generate interesting genotypes for your bugs relatively quickly.

There are six areas in this dialog box, as follows:

- Unknown C1/C2 genotype.
- Mobile Genotypes.
- Enhanced Dodgers (+C2).
- Sessile Genotypes (+C2).
- Capability Endowments (C3/C4).
- Complexity (C3/C4).

These are described in order below.

Unknown C1/C2 genotype

This single radio button is not under user control. If the user selects a C1/C2 genotype from the gallery of C1/C2 genotypes the TinkerBug Wizard turns this button off. Try it! If you switch to another tab (Palmiter genes, or Regulatory genes tabs) and alter the C1 or C2 genotype, this button will be turned back on by the TinkerBug Wizard.

Mobile Genotypes

This section offers the user a menu of six C1 genotypes which may appear in any low-level bowl of PSoup. The types are Pristine, Jitterbug, Twirlie, Cruiser, Arrow and Dodger.

Enhanced Dodgers

This section offers the user a further menu of 4 additional genotypes. In all of these, the C1 genotype is that of a Dodger. In each case, the C2 genotype has been altered for enhanced Stamina, Fecundity, Agility, and finally, in Superbug, all three are enhanced.

Sessile Genotypes

When the Stand Still gene is introduced in Level 3, this section appears and offers the user a menu of three sessile (or possibly more accurately, sedentary) genotypes. In each of these, the standard 8 Palmiter genes have a very low value and the Stand Still gene has a dominant strength. In practice, these bugs exhibit very plant-like behavior, something like an anemone might. Three genotypes are offered, labeled Bush, Dandelion and Potato. In this instance (one of many) we abandon the ‘Primordial Soup’ metaphor and try to emulate more advanced biological behavior. The Bush has high stamina. The Dandelion has high Fecundity. The Potato has high potential to store energy (visible as high Agility).

Note that sessile bugs are NEVER totally sessile. The transport genes always have some residual strength, and therefore always have some possibility of expression during a turn, resulting in movement. However, because of the restriction on the number of bugs in a cell, typically, only the bugs at the edge of a mat of plant-like bugs are in fact able to move. There is an exception to this. When bugs fission, two daughters occupy the cell formerly occupied by the mother. One of these must move, can move into any cell with 0 or 1 bugs, and must keep

moving until it finds an empty cell in which it can attach itself. This emulates the behavior of sessile organisms such as coral, oysters, or, perhaps, maple trees (via their seeds).

Capability Endowments (C3)

This section allows the user to endow bugs quickly and easily with basic C3/C4-derived capabilities. For each of the nine possible capabilities, there are four controls:

- A setup button, on the left.
- A drop-down list for selection of the strength of the C3 gene in each GType gene.
- A drop-down list for selection of the strength of the C4 gene in each GType gene.
- A + button which injects the appropriate genes into the C3/C4 gene list.

Note that the genotype does not actually change until the + button is pushed.
To enable a capability, such as SIGHT, first left-click on the setup button, then click on the arrow in each drop-down list and choose one of low, medium or high. Low sets the exponent at -1; medium sets the exponent at 0; high sets the exponent at 1, for strengths of 0.5, 1.0 and 2.0 respectively. Then click on the + button. This will inject two genes into each of the C3 and C4 chromosomes of the TinkerBug, as selected. The genes injected into C3 and C4 are those required to enable the capability. E.g., for SIGHT, two genes of GType S, I, G, H or T will be injected. Two such genes will go into each of C3 and C4, for a total of four. You will notice that the complexity has risen as this happens. You can adjust the drop-down list selections before you click the + button. When you click the + button the injection will occur and the selection controls disabled. Further endowments for this capability are restricted here, and must be done via the Sensory+ tab.

Complexity (C3)

Complexity is the number of genes in the C3 and C4 chromosomes. Each gene has a gene type (called the GType) which can be any Capital letter from the English alphabet. Some GTypes contribute to the enablement of a Capability Type (called a CType). There are nine such aggressive CTypes for the C3 chromosome and nine matching defensive CTypes for the C4 chromosome. A CType is enabled if the chromosome contains at least two of the GTypes required to spell part of the name of the CType. Some GTypes (such as the GType ‘T’) contribute to the enablement of several CTypes (such as SIGHT, TOUCH and TASTE) while others (such as the GType ‘Z’) do not contribute to the enablement of any CType. Such GTypes are said to be inert.

When you press the ‘Add Complexity’ button, you add an inert gene to the chromosome. This changes the complexity of a bug without any partial enablement of a CType. Why would you do this? The bugs use complexity as a means to determine relative sub-species differences. A bug with less than 1/4 the complexity is considered a parasite, etc.

As a user progresses to higher levels, the features described above are added to the dialog. At Level 1, only the Mobile Genotypes are available. At Level 2 the Enhanced Dodgers become available. At Level 3 the Sessile Genotypes become available. At Level 4 the C3/C4 sections appear, but only with the controls for the five senses. At Level 5 the reproductive CTypes XOVER and OVULE are added. At Level 6 the higher function CTypes HERD and THINK are added.

The Palmiter Genes tab (TinkerBug Wizard)

This dialog box lets you tinker with the specific genes in the C1 chromosome. There are two sections labeled ‘C1 - Genotype’ and ‘C1 - Phenotype’, which are described in order below.

C1 - Genotype

Each of the nine Palmiter genes are named (by their acronyms) and displayed here. While PSoup has the ability to mutate either the base or the exponent (at levels 2 and higher) the user is limited to modifications of the exponent only. The default value of the base is 2. If the user sees a base value other than 2, then you are tinkering with a bug that is not generation 1. It has undergone at least one mutation. Try it! Run a scenario for several generations, then tinker with the bugs.

To the immediate right of each exponent value is a small control with two arrows, one pointing up and the other pointing down. To modify the exponent, left-click on one of the arrows. The strength of the gene will automatically be changed by the TinkerBug Wizard.

For diploid bugs, you must select between the maternal C1 chromosome and the paternal C1 chromosome. For haploid bugs, these controls disappear.

C1 - Phenotype

This area displays SOME of the phenotypic characteristics of a bug which derive from the C1 genes. There are many possible genotypes which can produce the same phenotype. There is a column describing the percentage probability than a specific Palmiter gene will be expressed in any given move. The distribution of percentages determines the nature of the probabilistic path that a bug will follow. Here are a couple of examples.

Suppose that the 'Forward' (F) gene has an exponent of 3 (strength of 8) and the others have an exponent of 0 (strength of 1 each). Then the F gene represents 50% of the total strength. ON AVERAGE, 50% of its moves will be straight forward, and 50% of its moves will be in a random direction.

Now, suppose that the exponent for the other 8 genes is -3 (strength of 0.0125). Now the F gene represents well over 80% of the total strength. Four out of five moves will be directly forwards. One out of five moves will be in a random direction. This bug will also wander aimlessly, but it will travel in a much wider pattern.

Try It! Tinker with two bugs as described above, and watch their movements.

As part of the phenotype, the C1 penalty is also displayed. PSoup favors those bugs which have a C1 Penalty close to zero. To tinker with the formula used to compute the C1 Penalty, or to activate/deactivate it, see the 'Tinker with the environment' command.

This dialog box appears at Level 1, but the Stand Still gene, which was not part of Dr. Palmiter's original concept, is suppressed, as is the C1 Penalty. The C1 Penalty is added at Level 2. The Stand Still gene is added at Level 3.

The Regulatory Genes tab (TinkerBug Wizard)

This dialog box gives you the ability to make specific modifications to each of the six Regulatory genes found in the C2 chromosome.

This panel has three sections labeled 'C2 - Genotype', 'Defaults' and 'C3/C4 - Genotype', which are described in order below.

C2 - Genotype

Each of the six regulatory genes is named (both full name and acronym) and the values of the base and exponent for each is presented. While PSoup allows both base and exponent to mutate, the user can modify only the exponent. The small control with two arrows, one pointing up and the other pointing down, is called a spin control. The exponent can be altered by left-

clicking on one of the arrows in the associated spin control. By default, the base is equal to 1.1. If you see a number other than 1.1 in the base, then you are tinkering with a bug that is NOT generation 1 and has undergone some mutation. Try It!

For diploid bugs, you must select between the maternal C2 chromosome and the paternal C2 chromosome. For haploid bugs, these controls disappear.

Defaults

This section simply lists the default values for each of the regulatory genes.

C2 - Phenotype

This section displays SOME of the phenotypic characteristics which derive from the genotype. In particular, the Stamina, Fecundity, Agility and C2 Penalty are displayed.

This dialog box first appears at Level 2. Stamina, Fecundity and Agility are first displayed at Level 3. The C2 Penalty first appears at Level 3.

The Sensory+ Genes tab (TinkerBug Wizard)

In this tab you can modify each of the individual genes that make up the C3 and C4 chromosomes of the bug's genetic material. In other words, you can manually endow a bug with specific capabilities which are derived from the Sensory+ genes in the C3 and C4 chromosomes.

A bug can have from zero to 104 'capability' genes in the C3 and C4 chromosomes.

There are three groups of controls in this dialog box, labeled 'C3/C4 - Genotype', 'Operations' and 'C3/C4 - Phenotype'.. These are described in order.

C3/C4 - Genotype

There are many clickable items within the C3/C4 - Genotype group of controls. (I.e., if you left click on them with the mouse, they initiate an action.)

The GType indicator can have a value from A to Z. Click on the down arrow in the drop-down list and select a gene type from the list.

There are three clickable spin controls for the selected gene. The C3 and C4 genes are similar to the Palmiter genes (C1) or Regulatory genes (C2) in that each has a base and an exponent which are used to compute the strength of the gene, and a dominance. Either the base or the exponent can be modified to produce an infinite variety of alleles. Note that if the exponent is between -2 and 2 the base cannot be modified. The modification algorithm has to avoid a singularity.

For diploid bugs, you must select between the maternal C3/C4 chromosome and the paternal C3/C4 chromosome. For haploid bugs, these controls disappear.

Select between fight and flight chromosomes with the drop-down list for '3rd Chromosome' (Fight capability genes) and '4rth Chromosome' (Flight capability genes).

The section below displays individual GTypes and their relative strength for the selected chromosome. The strength of the genes in this chromosome (e.g. maternal, 3rd chromosome) is displayed. The button is pushed in for those types which are present in the chromosome. The button is pushed out for those not in the chromosome. Clicking on a button will select that gene for modification, but will not activate or deactivate the gene. Use the buttons for that.

Operations

There are two operational buttons.

Use the ‘Activate’ and ‘Deactivate’ buttons to turn genes on or off.

C3/C4 - Phenotype

This area indicates SOME of the phenotypic characteristics which are derivative of the selected genotype. None of the controls are clickable. I.e., if you click on any of these buttons, nothing will happen. Try it!

Those capabilities which have been activated are indicated by push-buttons. If the button is in, the capability is activated. If the button is out, the capability is not activated. The strength of the fight and flight components are shown separately. The fight component is the sum of all strengths of all fight (C3) genes for those Gene Types required to enable the capability. For example, to enable SIGHT, you need at two of GTypes S, I, G, H and T. Try it! Delete all genes for a bug, then add to C3 one of each. As each GType is added, the appropriate button below will be pushed in. When the capability is enabled, the TinkerBug Wizard will push in the appropriate capability button.

Any capability which enables detection of algae will also enable the ‘Sense algae’ button (e.g. SMELL). Any capability which enables the detection of bugs will also enable the ‘Sense Bugs’ button (e.g. TOUCH). Any combination of capabilities which enables a bug’s ability to sense and chase a mate will enable the ‘Sense Mates’ button (e.g. SIGHT and OVULE). Try it!

This dialog box first appears at Level 4. Only the five senses appear at this level. At Level 5, the enhanced reproductive capabilities OVULE and XOVER are added. At Level 6 the higher functions of HERD and THINK are added.

The Display Genes Button

This button is available in two places in PSoup. You will find it in the 'Tinker With the Bugs' dialog under the Tinker menu, and you will also find it on the 'Display Pairing Squares' dialog under the Options menu.

This button presents a report for any selected bug in the current population of bugs. The report tells you everything there is to know about the bug's genotype, and a great deal about the phenotype, and more.

For example, this report includes ploidy (haploid or diploid), number of active chromosomes, the value of the dominance type, base, exponent, and strength of every active gene, complexity, the name of the bug's mother (and father, when appropriate), the name of the donor of any cross-over genetic material or spermatic material.

Test Menu

Level 1 Test command (Tests menu)

Use this command to invoke the Level Achievement Test (LAT).to gain access to Level 2.

Level 2 Test command (Tests menu)

Use this command to invoke the Level Achievement Test (LAT).to gain access to Level 3.

Level 3 Test command (Tests menu)

Use this command to invoke the Level Achievement Test (LAT).to gain access to Level 4.

Level 4 Test command (Tests menu)

Use this command to invoke the Level Achievement Test (LAT).to gain access to Level 5.

Level 5 Test command (Tests menu)

Use this command to invoke the Level Achievement Test (LAT) to gain access to Level 6.

Level 6 Test command (Tests menu)

Use this command to invoke the Level Achievement Test (LAT) to complete Level 6.

Edit a Test... (Tests menu)

This command, accessible only in 'Teacher' mode, gives a teacher the ability to build and/or modify the six Level Achievement Tests (LATs). This feature is self-guiding. That is to say, there are a number of self-explanatory screens available to you as you use the feature, making it easy to develop an understanding of the capability of the tool, and skill with its use.

Here is what you are offered, in brief:
- six canned LATs which you can use, by default, without any action on your part.
- ability to copy any canned LAT to a test file and modify it by adding, deleting or changing questions.
- ability to discard any modified test at any time and return to the canned test.
- for those with skill with the computer, the ability to manage several alternate tests for each level, inserting the desired version onto the student's workstation prior to his/her use of the test.

Level Achievement Test dialog (Tests menu)

When you invoke a Level Achievement Test (LAT) at any level you will be presented with a multiple choice test of ten questions drawn randomly from a store of thirty or more questions specific to the current level.

To prepare for the LAT, here are some of the things that MAY be required:
- Run each canned scenario at the current level.
- Read the ‘About this scenario’ panel for each one.
- Answer any question raised.
- Carefully watch each scenario as it unfolds.
- Run each scenario several times to see what changes, and what doesn’t change.
- Tinker with the parameters in the scenarios to see how things change.

Recorder Menu

Load a recording (Recorder menu)

For an overview of the intended use of the recording function in PSoup, see the article 'About Recordings in PSoup' in the 'PSoup Theory and Practice' section.

This command allows the user to select a recording from among the library of saved recordings, and to load the selected recording in preparation for playback. Playback is done via any of the four following commands: Play 1 second; play 60 seconds, play to next stop, play to end.

If PSoup is in teacher mode, the loaded recording may also be edited. This includes editing the masthead, and inserting, editing and deleting teaching stops.

When playback is completed, the recording is returned to the library via the Close command.

Play 1 second (F10) (Recorder menu)

For an overview of the intended use of the recording function in PSoup, see the article 'About Recordings in PSoup' in the 'PSoup Theory and Practice' section.

This command is enabled when a recording has been loaded. It plays back one second of pre-recorded events from the recording's events file. An event can be placement of algae, move of a bug, attack by one bug on another, or a user-defined change of panel (e.g. changing the family tree panel out and replacing it with the Energy chart).

A teacher who plans to edit a recording must use the Play 1 second and Play 60 seconds commands to position the recording at appropriate places for the insertion of new stops.

Play 60 seconds (Recorder menu)

For an overview of the intended use of the recording function in PSoup, see the article 'About Recordings in PSoup' in the 'PSoup Theory and Practice' section.

This command is enabled when a recording has been loaded. It plays back sixty seconds of pre-recorded events from the recording's events file. An event can be placement of algae, move of a bug, attack by one bug on another, or a user-defined change of panel (e.g. changing the family tree panel out and replacing it with the Energy chart).

A teacher who plans to edit a recording must use the Play 1 second and Play 60 seconds commands to position the recording at appropriate places for the insertion of new stops.

Play to next stop (Recorder menu)

For an overview of the intended use of the recording function in PSoup, see the article 'About Recordings in PSoup' in the 'PSoup Theory and Practice' section.

This command is enabled when a recording has been loaded.

It plays back events from the event file until the next teaching stop has been reached. There is always a default teaching stop at the end of every recording, so this command behaves like the 'Play to end' command in the absence of teaching stops.

This command, as well as the 'Play to end' command, causes PSoup to start a background thread. The recording will proceed automatically until the next teaching stop is reached, unless the user presses the 'Pause/Stop' button on the toolbar or the F8 function key. These actions cause the background thread to cease, and PSoup returns to manual mode.

Play to end (F9) (Recorder menu)

For an overview of the intended use of the recording function in PSoup, see the article 'About Recordings in PSoup' in the 'PSoup Theory and Practice' section.

This command is enabled when a recording has been loaded.

It plays back events from the event file until the end of the recording has been reached. There is always a default teaching stop at the end of every recording, so this command behaves like the 'Play to next stop' command in the absence of intervening teaching stops.

This command, as well as the 'Play to next stop' command, causes PSoup to start a background thread. The recording will proceed automatically until the end of the recording is reached, unless the user presses the 'Pause/Stop' button on the toolbar or the F8 function key. These actions cause the background thread to cease, and PSoup returns to manual mode.

Pause/Stop (F8) (Recorder menu)

For an overview of the intended use of the recording function in PSoup, see the article 'About Recordings in PSoup' in the 'PSoup Theory and Practice' section.

This command is enabled when a recording has been loaded and when a background thread is automatically playing back pre-recorded events. This command causes the background thread to cease its operations, and the PSoup recorder function returns to manual view/edit mode.

Close this recording (Recorder menu)

For an overview of the intended use of the recording function in PSoup, see the article 'About Recordings in PSoup' in the 'PSoup Theory and Practice' section.

This command is enabled when a recording has been loaded into the recorder function in preparation for playback or edit.

This command returns the recording to the library of available recordings, and returns PSoup to normal operational mode. The recorder is no longer in control of the bowl of PSoup.

This command is the opposite of the 'Load a recording' command.

Start recording (Recorder menu)

For an overview of the intended use of the recording function in PSoup, see the article 'About Recordings in PSoup' in the 'PSoup Theory and Practice' section.

This command is used to create a recording and store it in the library of available recordings. Once a recording has been started, most significant events, whether in the bowl of PSoup, or in the panels around it, is recorded in an events file for later playback. Certain advanced user actions are not so recorded. The user cannot record actions respecting the Allele charts, cluster analysis, or bifurcation diagrams. The user cannot record Tinker actions, file-level actions (load, save, etc.) or other such actions which extend beyond the execution and observation of a single run of PSoup.

However, the user can record most other non-advanced actions under the 'Options' menu, and should make an effort, during the recording process, to ensure that the panels are activated which will make for the best observation of the final recording. There is some minor ability to override this using the 'Teaching stops' which can be inserted during the edit phase.

Stop recording (Recorder menu)

For an overview of the intended use of the recording function in PSoup, see the article 'About Recordings in PSoup' in the 'PSoup Theory and Practice' section.

This command is enabled by the 'Start recording' command, and is used to terminate the current recording session and return the recording to the library of available recordings.

Delete a recording (Recorder menu)

For an overview of the intended use of the recording function in PSoup, see the article 'About Recordings in PSoup' in the 'PSoup Theory and Practice' section.

Use this command to select a pre-existing recording from the library of available recordings, and delete it from the library. Once deleted, that recording will no longer be available to PSoup in either teacher or student mode.

Edit the masthead (Recorder menu)

For an overview of the intended use of the recording function in PSoup, see the article 'About Recordings in PSoup' in the 'PSoup Theory and Practice' section.

This command is enabled when a recording is loaded and PSoup is in 'Teacher Mode' (See the control panel under the 'Options' menu).

This command enables the user to edit the masthead for content. The masthead is saved immediately on change, so be certain you want to make the changes before you do so.

To prevent a masthead from being changed, put PSoup into 'Student Mode' prior to loading the recording.

Insert a stop here (Recorder menu)

For an overview of the intended use of the recording function in PSoup, see the article 'About Recordings in PSoup' in the 'PSoup Theory and Practice' section.

This command is enabled when a recording is loaded and PSoup is in 'Teacher Mode' (See the control panel under the 'Options' menu).

This command enables the user to insert a teaching stop at this particular PSoup second in the recording. In all subsequent playback sessions using this recording, PSoup will stop at this point and display the information stored in the teaching stop.

Edit this stop (Recorder menu)

For an overview of the intended use of the recording function in PSoup, see the article 'About Recordings in PSoup' in the 'PSoup Theory and Practice' section.

This command is enabled when a recording is loaded, PSoup is in 'Teacher Mode' (See the control panel under the 'Options' menu), and a pre-existing stop is current.

This command enables the user to edit the current teaching stop for content. The teaching stop is saved immediately on change, so be certain you want to make the changes before you do so.

To prevent a teaching stop from being changed, put PSoup into 'Student Mode' prior to loading the recording.

Delete this stop (Recorder menu)

For an overview of the intended use of the recording function in PSoup, see the article 'About Recordings in PSoup' in the 'PSoup Theory and Practice' section.

This command is enabled when a recording is loaded, PSoup is in 'Teacher Mode' (See the control panel under the 'Options' menu), and a pre-existing stop is current.

This command enables the user to delete the pre-existing teaching stop at this particular PSoup second in the recording. In all subsequent playback sessions using this recording, PSoup will no longer stop at this point. The contents of the teaching stop will be discarded.

View event file (Recorder menu)

For an overview of the intended use of the recording function in PSoup, see the article 'About Recordings in PSoup' in the 'PSoup Theory and Practice' section.

This command is enabled when a recording has been loaded. It enables a user to scan the events recorded in the events file. It may be useful for a user to note the timing of key events to plan the placement of teaching stops. You are warned, however, that a recording of a complex bowl of PSoup contains a very large number of events. You may find yourself looking for the proverbial needle in a haystack.

Help Menu

About This Scenario (Help menu)

Use this command to display information about the current scenario. Often you will find the answers the to questions of the Level Achievement Tests (LATs), or, at least, hints, in these screens. The information available through this command changes with each scenario run.

To prepare for the LAT, run each scenario for the level, carefully read the ‘About this scenario’, and try to answer any questions raised.

Nag Screen dialog

If you have a demonstration version of PSoup, or if you (for some unexplainable reason) choose to run a full-featured version of PSoup in demo mode, then you will quickly become familiar with the Nag Screen. This device is used to irritate you sufficiently; i.e. just enough so that you will be motivated to pay for this wonderful program.

Welcome command (Help menu)

Use this command to display the Welcome panel seen when PSoup is initially started.

About PSoup command (Help menu)

Use this command to display the copyright notice and version number of your copy of PSoup.

Register command (Help menu)

Use this command to display information about upgrading the limited-feature ‘Demonstration’ version to a full-featured version.

Credits command (Help menu)

Use this command to display information about the authors.

PSoup's Toolbars

About PSoup's Toolbars

PSoup has four toolbars, as follows:

- the Main toolbar;
- the Options toolbar; and
- the Evolution toolbar; and
- the Help toolbar.

The four toolbars are part of the overall PSoup Graphical User Interface (GUI, pronounced gooey). PSoup, for the most part, uses standard Microsoft Windows GUI features as deployed in Windows 98. For a complete description of the PSoup GUI, see PSoup GUI.

A toolbar is a very small window which holds one or more clickable buttons. Each button is linked to a command which may or may not be on the menu of commands for the application which are accessible through the menu bar. In the case of PSoup, one of the buttons (the DOE button) is represented by a menu item at Level 1, but for higher levels the Option toolbar's button is the only means available to the user to invoke the 'Desert Oasis Ecosystem' scenarios.

When you click on a toolbar button in PSoup it will respond in one of three ways:

- some spring back up immediately. The help toolbar buttons work this way. The button is depressed while you hold the left mouse button down and it springs back up into position when you release the mouse button. The command is issued to the computer at the moment the button springs back up.
- some stay pressed in, and do not spring back out immediately, but if you left-click on them a second time, they spring back out. These buttons are called 'toggles'. They represent two opposing commands, one of which undoes the work of the other. For example, when PSoup first starts, you will notice that the status button on the Options menu is already in. This means that the Status panel is already on display. If you click this button, it will spring out, and the status panel will disappear. If you click it again, the status panel will re-appear.
- some stay pressed in, and do not spring back out until they have been clicked several times. These buttons are complex toggles which have more than two possible states. These buttons control the display of several panels which can overlay each other. Multiple presses of the button cycles the button through its various on states and, finally, to its off state. At Level 1 all of the buttons (except DOE) on the Options toolbar function as simple toggles. At higher levels most of them function as complex toggles.

All toggles are associated with menu commands. When the button is 'in' (i.e. the associated panel is being displayed) the menu command will also be checked (i.e. a small check mark will be displayed to the right of the command). Try it. Left-click on the Options menu and note which commands are checked and which are not. Check the toolbar buttons on the options toolbar. The checked menu items correspond to the pressed buttons. Note that, at the higher level, such a menu command will need to be issued several times (as many as six for the population profile charts) to cycle through all of the on states and turn the panel off. The toolbar, in this instance, is a GREAT time-saver.

When PSoup opens, the four toolbars are resting side-by-side in a thin monochrome bar immediately below the menu bar. Each toolbar has a small handle on its left-hand side. This handle can be recognized as a vertically dimpled edge. If you grab the toolbar by its handle and drag it to the middle of the screen (i.e. point the mouse to the dimpled handle, press down on the left mouse button, and drag the toolbar to the middle of PSoup, then let the left-hand mouse button up again) it becomes a floating toolbar. Normally, it is a docked toolbar. In PSoup you can dock the three toolbars only at the top of the client area. In many applications they can be docked at any of the four sides of the client area, however, this is not practical and not allowed in PSoup. You may choose to dock them at the top, or leave them floating in the middle, or get

rid of them entirely. To get rid of them (temporarily), drag them to the middle, then click on the small 'Close' icon on their title bars. To get them back, however, you must exit PSoup and restart it.

PSoup offers a graduated user interface with three levels to serve the needs of users with an increased ability and interest in using the full range of features. For more about the graduated user interface see PSoup GUI. For each of the four toolbars, the following table indicates which buttons are available at each level:

	Novice	Intermediate	Advanced
Main Toolbar:			
Open	Yes	Yes	Yes
Save	Yes	Yes	Yes
Options Toolbar:			
Average Bug	No	Yes	Yes
Current Bug	Yes	Yes	Yes
Status Panel	Yes	Yes	Yes
Legend Panel'	Yes	Yes	Yes
Events Counts	No	No	Yes
Family Tree	Yes	Yes	Yes
Energy Chart	Yes	Yes	Yes
Life Functions	No	No	Yes
Population Chart	No	No	Yes
Allele Chart	No	No	Yes
Cluster Analysis	No	No	Yes
Evolution Toolbar:			
ReSeed	Yes	Yes	Yes
DOE	No	No	Yes
Pause/Stop	Yes	Yes	Yes
Go	No	Yes	Yes
Slow Motion	Yes	Yes	Yes
One Second	Yes	Yes	Yes
One Bug Yes	Yes	Yes	Yes
One Function	No	No	Yes
Recorder	No	No	Yes
Help Toolbar:			
Context Help (?)	Yes	Yes	Yes
About This Scenario	Yes	Yes	Yes
About PSoup	Yes	No	No
About Registration	Yes	No	No

The Main Toolbar (PSoup Toolbars)

The main toolbar is a radically reduced version of the standard Microsoft Windows toolbar for applications. It has only two buttons:

1st button	A file folder	Open
2nd button	A diskette	Save

The Options Toolbar (PSoup Toolbars)

The options toolbar has eleven buttons. Following is a list of the buttons, in order, from left to right, with a description of each button and the command it issues:

1st button	A bug with the letters Ave	Display Average Bug
2nd button	A bug with the letters Cur	Display Current Bug

Separator

3rd button	A panel with the word Status	Display PSoup Status
4rth button	A panel with the word Legend	Display Scenario Legend
5th button	A panel with the word Count	Display Event Counts

Separator

6th button	A tree with the word Tree	Display Family Tree
7th button	A lightning bolt with the letters Nrg	Display Energy Chart
8th button	A bug with the letters Fns	Display Life Function Panels
9th button	A graph with the letters PopP	Display Population Profiles

Separator

10th button	A tabular chart	Display Allele Chart
11th button	A dialog to perform cluster analysis	Display Cluster Analysis Report

The 'Novice' user interface set contains only buttons 1, 3, 4, 6 and 7.
The 'Intermediate' user interface set contains buttons 1, 2, 3, 4, 6 and 7.
The 'Advanced' user interface set contains all buttons.

The Evolution Toolbar (PSoup Toolbars)

The evolution toolbar has six buttons. Following is a list of the buttons, in order, from left to right, with a description of each button and the command it issues:

1st button	A bowl of PSoup with the word Seed	ReSeed

Separator

2nd button	A stop sign with the word Stop	Pause

Separator

3rd button	A green traffic light with the letter G	Go
4rth button	A right-pointing arrow with the letters 1Sec	Advance By One Second
5th button	A right-pointing arrow with the letters 1Sec	Advance By One Bug
6th button	A right-pointing arrow with the letters SloMo	Go Slow Mode
7th button	A right-pointing arrow with the letters 1Sec	Advance By One Function

The 'Novice' user interface set contains only buttons 1, 3, 4, and 5.
The 'Intermediate' user interface set contains buttons 1, 2, 3, 4, 6 and 6.
The 'Advanced' user interface set contains all buttons.

The Help Toolbar (PSoup Toolbars)

The help toolbar has two or four buttons. Following is a list of the buttons, in order, from left to right, with a description of each button and the command it issues:

1st button	A yellow question mark	Like the F1 key, invokes context-sensitive help
2nd button	A dialog box with the letters Sc	About This Scenario

Separator

3rd button	A dialog box with the letters PSoup	About PSoup
4th button	A dialog box with the letters Re	Register

All four buttons are presented in the 'Novice' user interface set. In the 'Intermediate' and 'Advanced' user interface sets, only the first two buttons are shown.

The PSoup Graphical User Interface (GUI)

PSoup has a relatively standard windows Graphical User Interface. Many of the standard features of such an interface are not documented here in detail. However, for the sake of those users who may be novices with Windows interfaces, here is an overview.

Most windows applications, including PSoup, are framed in a large window with standard doodads placed around the outside edge. Look for the following standard doodads:

- Title Bar - this is a thin monochrome bar across the very top of the window. It has the title of the application (in words) printed across it, and possibly the name of the current open file, or other such basic information. On the extreme left of the title bar is a small clickable icon (picture). If you click once on this icon, you will be presented with a drop-down system menu with the following items on it: (Restore, Move, Size, Minimize, Maximize, and Close). These commands control the size and position of the main window in which the application resides. To the far right of the title bar are three more clickable icons. One has a small bar at the bottom. It is used to minimize the window, the same effect as the Minimize command of the system menu. The next bar has two little overlapping windows in it. It is used to restore the window to visible size, the same effect as the Restore command of the system menu. The rightmost icon has an X on it. It is used to close the application, the same effect as the Close command of the system menu.

- Menu Bar - this is a thin monochrome bar immediately below the Title bar. It has a list of seven drop-down menus, each of which offers you several more commands or drop-down menus. In PSoup the menu options are:
  - File
  - Scenarios
  - Evolution
  - Options
  - Tinker
  - Tests
  - Recorder
  - and Help

- Status Bar - in fact, you will see two status bars at the bottom of your screen. The bottom-most status bar belongs to the operating system and is there all of the time. It is possible to make this status bar 0 pixels tall by grabbing the top of it (using the mouse) and dragging the top down to the bottom. The status bar above the system status bar belongs to PSoup. Normally it says 'For HELP press F1'. F1 is the 'function key' located in the top left corner of your keyboard. As the mouse moves over various features of the GUI this message will change. For example, if you slowly move the mouse over the toolbar buttons and watch the status bar, you will see a brief description of the purpose of each button appear in the status bar.

- Toolbars - PSoup has three toolbars for which the purpose is documented in some detail in the article on Toolbars. In this section we cover merely how the toolbars work. When PSoup opens, there are three toolbars resting side-by-side in a thin monochrome bar immediately below the menu bar. Each toolbar has a small handle on its left-hand side. This handle can be recognized as a vertically dimpled edge. If you grab the toolbar by its handle (i.e. point the mouse to the dimpled handle, press down on the left mouse button, and drag the toolbar to the middle of PSoup, then let the left-hand mouse button up again) and drag it to the middle of the screen it becomes a floating toolbar. Normally, it is a docked toolbar. In PSoup you can dock the three toolbars on any of the four sides of the main window, or leave them floating in the middle, or get rid of them entirely. To get rid of them (temporarily), drag them to the middle, then click on the small 'Close' icon on their title bars. To get them back, however, you must exit PSoup and restart it.

- Client Area. - The largish area in the center of the PSoup application is where the action happens. This is called the client area. It is divided up into a number of special-purpose panels which are described in detail elsewhere. These panels are NOT implemented as windows, in the technical sense, and so cannot be dragged, dropped or sized. If you choose a screen-size or resolution which is incompatible with the panel design, you will have a less than satisfactory experience. See About PSoup for a description of best viewing settings.

- Tooltips - PSoup has implemented tooltips. A tooltip is a small scrap of pop-up advice which appears whenever the mouse pointer hovers for a while over the top of an appropriate GUI feature. In particular, the buttons on the toolbars each have a tooltip associated with them.

- Selectable commands - PSoup is controlled via commands selected by the user. A command can be triggered either directly or after an extended dialog between PSoup and the user. A dialog is a single-purpose window which pops up and covers PSoup, and collects information from the user, or gives instruction to the user. Commands can be selected in the following ways:

- Mouse clicks. All commands, menus, and buttons in PSoup are activated by a single click. There are no double-click items. Point the mouse to the menu item or toolbar button and left-click once.

- Alt-arrow-enter. All menu items are selectable via Alt-arrow combinations. If you press the Alt key once and release it, you will notice that the 'File' menu item becomes depressed, as if selected by the mouse, but the drop-down menu does not appear. If you then use the arrow keys, you can select any menu item. The left and right arrows move you through the main menu items (including the system menu) and the up and down arrows move you through the sub-menu items. When you are pointing at the menu item you wish to invoke, press enter. The toolbar buttons are not accessible via this method.

- Alt-Key sequences. The above approach can be speeded up by holding the Alt key down and alternately pressing the underscored letter for each menu item you wish to move to. For example, if you hold down the Alt key and press 's', then '1', then 'b' you will select Scenarios, Level 1, Scenario 1B. Be careful, this is not as safe as the Alt-arrow-enter approach. As soon as a non-ambiguous selection is identified, it will activate. The enter key is required only if there is no non-ambiguous selection.

- Acceleration keys. Some commands are represented by a single special keystroke. If you enter that keystroke, the command will activate. Those menu items which can be activated by an acceleration keystroke have the appropriate keystroke identified to the far right of the menu name. For example, the menu item 'Options', 'Display Average Bug' is activated by F4.

- Toolbar buttons. Some commands are represented by a toolbar button. A single left-click of the mouse on the toolbar button will activate the command.

- Graduated user interface. PSoup offers three levels of complexity in the user interface. Under the Options menu there is an item labeled the Control Panel. In the bottom left corner of the control panel a user can choose one of the three levels of complexity for the user interface: novice, intermediate or advanced. When you first start PSoup, the novice user interface is activated. As you progress through the scenarios in levels 1 and 2 the user is introduced to additional functions, and the scenarios invoke the appropriate type of interface. At each higher level of complexity of the user interface, menu commands and toolbar buttons are added to the interface until, at the advanced level, all commands and buttons are available. Note that, at all three levels of complexity, if a particular command is inappropriate to the current situation, then that command (and any associated toolbar button) is grayed and disabled. To re-enable these visible items, you must establish the appropriate situation. For example, advancing PSoup by one bug causes the File Save command to be grayed. You cannot save a file with a partial second processed. You must complete the processing of the second to re-enable this command (button).

Drop-Down Menu

A drop-down menu is a menu of selectable commands, dialogs, or sub-menus which a user is able to invoke by various means, including mouse clicks, arrow keys followed by an 'enter' key, alt-key keystrokes, or accelerator key keystrokes. A drop-down menu appears when its menu name is selected by the user.

PSoup Graphic Panels (PSoup GUI)

In PSoup, the client area of the main window is divided up into a number of informational panels which are optionally displayed. Each panel has a specific purpose and is located in a specific region of the screen. Many panels have similar purpose and are displayed in the same location, and controlled by the same menu commands, toolbar buttons, Alt-keys or function keys. As you run the PSoup program and focus on different aspects of an unfolding scenario, you will want to enable different panels appropriate to the circumstances. If you wish to go for coffee and have PSoup's denizens rapidly evolve in your absence, you may wish to suppress the display of all such panels. As you progress in your understanding of PSoup and how it works, and achieve a higher plane of understanding (take that to mean you have passed one or more Level Achievement Tests or LATs) new panels will become available. There are nine types of panel which can be displayed, as follows:

- Title Panel
- Bowl of PSoup panel
- Display Average Bug panel
- Display Current Bug panel
- Display PSoup Status panel
- Display Scenario Legend panel
- Display Event Counts panel
- Display Family Tree panel
- Display Energy Chart panel
- Display Life Functions panel
- Display Population Profiles panel

The Title Panel.

This panel is not optional. It contains the name of the current scenario. When you use the 'Scenario' menu item to choose a new scenario, the title will be updated with the name of the new scenario. If you use the 'Tinker With The Environment' command of the 'Tinker' menu, you will have the opportunity to modify the scenario title. By default, any scenario which has been tinkered with will have the letters 'UD' appended to the front of the scenario title. This stands for 'User Determined'. This tag can, of course, be over-ridden by the user.

The Bowl of PSoup.

The rectangular area immediately below the title panel, and of the same width, is your bowl of primordial soup. It is divided up into square cells, each of which contain some or all of nutritive mud, algae, and one or two bugs. A bug can be identified by the directional line from the center of the cell running to the edge of the cell. The concept of 'primordial soup' is only the first of several analogies that will be used to describe this panel. Other analogies will be 'Desert/Oasis' and 'Terraced Paddies'. We call it the bowl of primordial soup reserving the right to use whatever analogy is appropriate to understand the evolutionary forces at play.

The Bowl of PSoup is exactly 450 units by 180 units. The minimum size of a cell is 2 units by 2 units. The maximum size of a cell is 18 units by 18 units. There are a range of cell sizes between these two limits which allow a tidy rectangular display. Each scenario has an appropriate sizing of cells to best demonstrate the point of the scenario. Scenarios with many cells can accommodate many bugs, and evolve slowly, but

have the greatest potential to display complex interactions. Scenarios with few cells in the bowl unfold very quickly.

You can use the 'Tinker' menu item 'Tinker With The Environment' to alter many of the settings for the bowl of PSoup. In particular, it is possible to change the number of cells, edge-wrap, terrain types, algae distribution mode, energy level, and more.

The Status Panel.

Again, immediately under the bowl of PSoup is the 'Status Panel', which you will recognize by the word 'Status' in the top left corner. This panel is the same width as the bowl of PSoup. It is updated once each tick of the PSoup clock, i.e. once each PSoup second. It displays a few basic things about the current Bowl of PSoup, as follows:

- Number of Bugs - this is the number of bugs currently alive and active in your bowl of PSoup;
- Last BugID used - each bug has a unique serial number assigned to it when it is first spawned. The first bug is assigned the number one. The second bug is assigned the number two. And so on... This number is the last serial number that was assigned to a bug, and therefore tells you how many bugs, in total, have been spawned in this scenario since the beginning. In each scenario, the numbering starts again at one.
- Age - this is displayed in days, hours, minutes and PSoup seconds. This is, very simply, a representation of the number of ticks of the PSoup clock since the scenario started. A PSoup second is one tick of the clock. A tick of the clock is NOT a pre-determined length of real time. Rather it is a logical tick of the clock. During one tick of the clock each and every bug gets to take a turn. The tick of the clock starts when the first bug starts its move, and the tick of the clock is completed when the last bug has finished its move. If the bowl of PSoup contains only a single bug, a tick is completed very quickly. If the bowl contains many hundreds of bugs a tick of the clock may take a while. For each tick of the clock, the bowl of PSoup ages by one PSoup second. For convenience, there are 60 PSoup seconds per minute, 60 PSoup minutes per hour, and 24 PSoup hours per day. PSoup can count up to 32000 days before it cycles back to 0d, 0h, 0m and 1 PSoup second again.
- Energy - PSoup is a closed energy system. All energy starts in the nutritive mud. The energy in the mud is not localized to one or several cells, but is considered to be ubiquitous. When 40 or more units of PSoup energy become available in the mud, then the computer attempts to place a colony of algae somewhere in the bowl of soup, according to the rules governing the scenario. When a bug eats the algae, the energy is then transferred to the bug. As a bug moves and dies, the energy is transferred back into the mud. This panel tells you how much energy is in each of the three stages of the energy cycle. The total energy in the ecosystem (i.e. in the bowl of PSoup) does not change.

The status panel uses computer cycles in each tick of the PSoup clock. If you want a scenario to develop at top speed, toggle the status panel off. This can be done in several ways, as follows:

- Via the menu bar, use the mouse to click on 'Options', then on 'Display PSoup Status'. The check mark will disappear, and so will the status panel.
- Hold down the 'Alt' key and press 'o', followed by 's'. These are the underlined letters in the menu bar items. 'O' and 's' are called the accelerator keys for these menu items, and they allow you to run PSoup without a mouse, if you prefer keyboard actions. All menu items can be executed via accelerator keys in place of mouse clicks.
- Press 'F6'.
- Use the mouse to click on the 'Status' button in the 'Options' toolbar.

In higher levels (after you have passed one or more LATs) you will be introduced to other panels which occupy the same real estate as the status panel. Specifically, these are the 'Scenario Legend' panels and the 'Event Counts' panel. If you toggle any one of these alternate panels on, the status panel will automatically toggle itself off.

The Family Tree Panel

This panel stretches across the bottom of your screen from side to side under the status panel. This panel displays the family tree of all currently active bugs. The tree is updated and pruned on a regular basis, so it disappears and redisplays itself often. To understand this panel, you need to understand some of the mechanics behind the implementation of PSoup. All living bugs in a bowl of PSoup are chained together in a logical chain, from first to last. In each tick of the PSoup clock each bug takes its turn, in order, from first to last. The relative position of two bugs never changes.

Each living bug can come to an end in one of four ways. It can die without issue, in which case it is deleted from the chain. It can have one daughter (by birth) and continue to live, in which case the daughter is inserted into the chain ahead of the mother. Following this, it may have a second child (it dies in childbirth of the second child) at which time it dies and is deleted from the chain, to be replaced by the second daughter. Or, if the reproduction mode is by fission rather than by birth, it splits, is deleted from the chain, and its two daughters are inserted into the chain in its place.

The family tree panel keeps track of these changes and displays the relevant connections between bugs on a historical basis. Each bug which is currently alive is represented by a vertical bar at the top of the panel. With each tick of the PSoup clock, the view in the panel slides downwards by one tick, and the bars lengthen, giving a graphic representation of the aging of the bugs. You will note that when a scenario is not yet started, there are two labels to the left, one of which says ‘00000 secs’ and the other of which says ‘-00126 secs’. The top counter keeps track of the current age of the PSoup. The time indicated here matches the age, in PSoup seconds, of the bowl of PSoup as shown in the status panel. The lower label tells you the oldest tick that the chart can represent. There is room for exactly 126 ticks of the PSoup clock in this chart, hence the time of -126. You will note that, during the first two minutes of any scenario, this Family Tree panel is updated every PSoup second. This gives you the sense that all bugs are aging and growing older, as their bars stretch in length. You will also note that the two labels change as time passes.

When the scenario reaches an age of exactly two PSoup minutes, it changes mode. Time becomes compressed, and the Family Tree panel is updated only once every four PSoup minutes. When this transition occurs, you will note that the upper label changes to say ‘00002 mins’ and the lower label changes to say ‘-00499 mins’. After the first two minutes the changes in the panel happen much less often, but are substantially more interesting. As each bug reproduces, its daughter bugs are inserted into the family tree. The bar representing the mother bug is terminated when she dies, and the bars representing the daughter bugs are started. A horizontal line is drawn from the mother to the daughter(s). If a scenario is allowed to run (and succeeds in running) for 500 PSoup minutes, then the family tree panel will be filled from top to bottom with the heritage information of all currently existing bugs. Records older than 500 PSoup minutes are discarded.

Each time the Family Tree panel is updated, the tree is pruned. All branches of the tree which do not currently have living descendants in the bowl of PSoup are removed from the tree.

Sometimes, there are too many nodes in the family tree chart to represent in the space available. In this case, the nodes on the right are not displayed and are not accessible to the user. Sorry!

The Family Tree panel uses computer cycles in each tick of the PSoup clock, whether it is displayed or not. Once the original two minutes have elapsed, it is updated every four PSoup minutes, as stated above. This represents a very small relative additional draw on computer cycles, so it is reasonable to leave the Family Tree panel active all the time, even during a prolonged run of a scenario. However, if you want a scenario to develop at absolute top speed, toggle the family tree panel off. This can be done in several ways, as follows:

- Via the menu bar, use the mouse to click on ‘Options’, then on ‘Display family Tree’. The check mark will disappear, and so will the family tree panel.

- Hold down the 'Alt' key and press 'o', followed by 't'. These are the underlined letters in the menu bar items. 'O' and 't' are called the accelerator keys for these menu items, and they allow you to run PSoup without a mouse, if you prefer keyboard actions. All menu items can be executed via accelerator keys in place of mouse clicks.

- Press 'F[X]'.

- Use the mouse to click on the 'Tree' button in the 'Options' toolbar.

In higher levels (after you have passed one or more LATs) you will be introduced to other panels which occupy the same real estate as the family tree panel. Specifically, these are the 'Life Function' panels, the 'Event Counts' panel, and the 'Population Distribution' panels. If you toggle any one of these alternate panels on, the family tree panel will automatically toggle itself off.

The Current Bug Genetic Profile panel

This panel can be recognized as the tall rectangular panel at the right, abutting against the Title, PSoup and Status panels. It is entitled 'C1 gene profile of xxxxxyy. For short, call this the CurBug panel. For a detailed description of the type 1 (C1) genes, see 'About This Scenario' for "Scenario 1A - The Basics". This can be accessed by loading Scenario 1A into the bowl of PSoup, then clicking on the 'Sc' button on the options toolbar, or, via the menu, clicking on 'Help', and then 'About this scenario...'. Xxxxxyy is the full name of the current bug. During a tick of the PSoup clock, each bug gets to take a turn, i.e. it gets to be the current bug for a brief period of PSoup time; a mere fraction of a PSoup second. When a bug is the current bug, it is highlighted in the CurBug panel.

The CurBug panel has three different sub-panels which correspond to the four different chromosome types.:

- Type 1 (C1) genes control a bug's movements. The C1 genes are also referred to by several other names, depending on the circumstances. Because they control a bug's movements, they are called the transport genes. Because they were invented conceptually by DR Michael Palmiter, they are called the Palmiter genes. Because the authors have a love of acronyms, they are called the C1 genes. Is that clear?

- Type 2 (C2) genes control a bug's life functions. The C2 genes are also referred to as the Regulatory genes, because they regulate the bug's life functions.

- Type 3 and Type 4 genes endow a bug with higher functions. The C3 & C4 genes are also called Sensory+ genes, because they endow the bugs with five senses plus four other higher functions.

Each type of chromosome is represented by its own graphic sub-panel.

The Current Bug C1 Genetic Profile sub-panel

This panel gives us a graphic representation of the strengths of each of the Palmiter genes (or type 1 chromosome or Transport genes). The CurBug C1sub-panel displays each of the following features:

- The Bug's full name - each bug is given a name consisting of five or fewer letters. The name is generated randomly from a list of fifty names. If a scenario has ten bugs in it, there is a reasonable possibility that two bugs will have the same name. If a scenario starts with 100 bugs, it is certain that many bugs will have the same name. The name is therefore qualified with the Bug's unique ID number which is appended to the end of its name. You will therefore see names such as Fred01 or Benny10. When a bug reproduces, its offspring inherit its name, so its daughter(s) will have the same name but a different Bug ID.

- The Palmiter clock - This is a circle with the letters F, FR, etc. arranged around the outside and up to eight 'hands' on the clock face. Each bug has four primary groups of genes called chromosomes. The first type of chromosome, of nine genes, prosaically called the 'type 1' (C1) genes, are also called the 'Palmiter' genes. The eight hands represent the eight genes which control the bug's motive mechanisms. A ninth C1 gene causes a bug to stand still for a turn. DR Michael Palmiter first wrote a program called 'Simulated Evolution' in 1985 in which he introduced the concepts implemented in the type 1 (C1) genes. PSoup has implemented a slightly modified version of his concepts in the C1 genes. Each Palmiter gene

has a strength which has meaning only on the basis of it's proportionate size when compared to the other eight Palmiter genes. For a given turn a bug MUST select one of nine possible moves, the nine adjacent cells into which it may step (including the one it currently occupies). The relative probability that it will go in a given direction is determined by the relative strength of the appropriate gene. All Palmiter genes have a positive value, so at least one has a maximum value. The longest hand represents the gene with the maximum value. All other hands are scaled appropriately to show the relative probability of expression of the associated gene. The color of the hands is determined by the color of the bug.

- A numeric table - there are three columns in this table. The leftmost column lists the short names of the nine Palmiter genes: those being Forward (F), Forward Right (FR), Right (R), Back Right (BR), Back (B), Back Left (BL), Left (L), Forward Left (FL) and Stand Still (SS). At level 1 the SS gene is not activated and is not shown in the table. At higher levels, it will appear at the bottom. The middle column displays the strength of each Palmiter gene. Each Palmiter gene has three components to it; a base, an exponent, and a strength. For Palmiter genes at level 1 the base is always 2, the exponent is always a positive or negative integer, and the strength is the result when the base number is raised to the power of the exponent. Therefore, the strength of a level 1 Palmiter gene is always a power of 2. The default starting value is 2 raised to the power 0, or 1. The rightmost column displays the relative strength of each Palmiter gene when compared with the other Palmiter genes. Relative strengths are presented as percentages. You can view the second column as the C1 genotype of the bug, and the third column as the C1 phenotype of the bug.

Each bug has an internal orientation which has one of eight values, representing each of the eight neighboring cells in the PSoup. Each and every second, the bug steps forward into the neighboring cell and it eats any algae found there. However, just before it steps forward, it pivots a random amount to face one of the eight optional cells. Each bug has eight genes which control its pivoting behavior prior to stepping forward. A gene with a larger value, relative to the other genes, has a proportionately higher probability of expressing itself, and causing the bug to turn that amount. Only one gene can express itself on any one step, and determine the amount of pivoting for that step.

In the display, each gene for the current bug is labeled, and the relative strength presented both as a line on the face of a clock-like diagram, and as a number below the clock.

The following chart describes the labels used in the display, the interpretation, and the pivoting effect on the bug when that gene is expressed.

F: Forward – the bug does not turn, but just steps forward.
FR: Forward Right – the bug turns forty five degrees to the right.
FL: Forward Left – the bug turns forty five degrees to the left.
R: Right – the bug turns ninety degrees to the right.
L: Left – the bug turns ninety degrees to the left.
BR: Backward Right – the but turns one hundred thirty five degrees to the right.
BL: Backward Left – the bug turns one hundred thirty five degrees to the left.
B: Backward – the bug turns one hundred eighty degrees and steps back into the cell it came out of on the previous step.

C1 Penalty. At the very bottom of the C1 genetic profile is a small one-sided bar graph labeled the C1P:. This stands for C1 Penalty. When we allow the nine type 1 (C1) genes to mutate under evolutionary pressure for very long runs, they have a tendency to grow to VERY large values. This is not dissimilar to what happens in biological systems. For example, the human genome is known to consist of roughly three trillion nucleotides, more than 90% of which is non-useful material. It is presumed that this number is still growing as time goes by. However, such large numbers are a problem for computers. Eventually, numbers that grow too large cause arithmetic overflow problems and the computer becomes confused. It is therefore useful to avoid excessively large numbers. There are two ways to do this. We can (a) simply disallow any numbers greater than a certain size. This is the mode of operation in the lower levels of PSoup. Base values cannot be larger than 50 and exponents cannot be larger than +-150. This introduces an unseen but

ugly and unwanted ‘edge effect’ which, in some rare circumstances, distorts the results. However, even with this approach, very large numbers are not well represented. Or we can (b) introduce genetically-determined counter balancing pressures which will keep these numbers within reasonable bounds. In PSoup’s higher levels we have implemented this approach. The User is able to activate something called the C1 Penalty. The formula for the C1 Penalty is (MaxStrength - MinStrength) / Scaling Factor. The scaling factor is user-determined. The C1 Penalty is added to the Energy Per Move (EPM) on each turn, and that much additional energy is consumed by the bug on each and every turn. If the scaling factor has a value of ‘1’, any and every mutation is quickly fatal. If the value of the scaling factor is very large, then it will have little or no effect.

The CurBug C1 sub-panel uses computer cycles in each turn of each bug during each tick of the PSoup clock, but only when displayed. If there are 200 bugs, this profile is updated 200 time per PSoup tick. If you want a PSoup scenario to develop at the slowest possible speed, toggle the CurBug Panel on and toggle the ‘Life Functions’ panel on as well. However, if you want a scenario to develop at top speed, toggle the CurBug panel off. This can be done in several ways, as follows:

- Via the menu bar, use the mouse to click on ‘Options’, then on ‘Display Current Bug’. The check mark will disappear, and so will the CurBug panel.
- Hold down the ‘Alt’ key and press ‘o’, followed by ‘c’. These are the underlined letters in the menu bar items. ‘O’ and ‘c’ are called the accelerator keys for these menu items, and they allow you to run PSoup without a mouse, if you prefer keyboard actions. All menu items can be executed via accelerator keys in place of mouse clicks.
- Press ‘F5’.
- Use the mouse to click on the ‘Cur’ button in the ‘Options’ toolbar.

In higher levels (after you have passed one or more LATs) you will be introduced to other sub-panels which occupy the same real estate as the CurBug C1 gene profile sub-panel. If you toggle any one of these alternate panels on, the CurBug C1 profile sub-panel will automatically toggle itself off.

The Current Bug C2 Genetic Profile sub-panel

This panel gives us a graphic representation of the strengths of each of the Regulatory (or type 2) genes. The CurBug C2 sub-panel displays each of the following features:

- The Bug’s full name again - see the description of the C1 Genetic profile for a detailed description of this.
- Six One Sided Bar Graphs - There is one for each gene as follows: DAT, DET, RAT, RET, EPM, and EPB. Each bar graph has two bars. The black bar shows the default strength. The red bar shows you the current strength of the gene. All values are either zero or a positive number. Zero is on the left. Remember, the strength of the gene is calculated as the base of the gene raised to the power of the exponent. The strength of the gene is the effective number for the expression of the gene. All six of these bars are scaled to fill the available width. Either the black bar, or the red bar, whichever represents the larger number, will fill the width. The other bar, the shorter one, is scaled to show its relative size. By this technique, you can quickly scan the profile and determine whether the gene has mutated upwards or downwards from the default strength. To the left of each bar is the gene acronym. To the right of each bar is a number indicating the strength of the largest of the two quantities, default and actual.
- Next is a separator, a bi-colored bar running from left to right without any label. This bar separates the gene values from the derived values. The six bars above this line represent the values of actual genes. The four values below the line are computed as a combination of the strengths of the six genes. You may consider that the items below the line are purely phenotypic, while the items above the line are not purely phenotypic.
- Immediately below the separator is a group of three two-sided bar graphs labeled ‘Sta’, ‘Fec’, and ‘Agi’. These labels stand for ‘Stamina’, ‘Fecundity’ and ‘Agility’ respectively.
- Stamina is a derived value. It is computed as (A * DeltaDAT) + (B * DeltaDET) where A = .00625, B = -0.36667, DeltaDAT = (DAT(current) - DAT(default)) and DeltaDET = (DET(current) - DET(default). There is a similar formula for each of Fecundity and Agility as well. Note a couple of

things about this formula. First, when DAT(current) is equal to DAT(default) and DET(current) is equal to DET(default) then Stamina is zero. This is intended to be so. As the DAT or DET genes mutate and migrate away from their default strengths, the value of Stamina will rise or fall. Second, A and B are scaling factors such that a mutation of either DAT or DET either upwards or downwards by one increment will change the value of the Stamina by roughly one. In other words Stamina is designed to be equally sensitive to a change in either gene in either direction. We can therefore carefully use this derived value to put evolutionary pressures on the genes from which Stamina is derived. In Level 3 Stamina is used with the other two derived characteristics to apply such pressures. See below under C2 Penalty.

- Fecundity is a derived value. It is computed as (A * DeltaRAT) + (B * DeltaRET) where A = .-0.01375, B = -0.011, DeltaRAT = (RAT(current) - RAT(default)) and DeltaRET = (RET(current) - RET(default). There is a similar formula for each of Stamina and Agility as well. In a similar mode to Stamina, note a couple of things about this formula. First, when RAT(current) is equal to RAT(default) and RET(current) is equal to RET(default) then fecundity is zero. This is intended to be so. As the RAT or RET genes mutate and migrate away from their default strengths, the value of Fecundity will rise or fall. Second, A and B are scaling factors such that a mutation of either RAT or RET either upwards or downwards by one increment will change the value of the Fecundity by roughly one. In other words Fecundity is designed to be equally sensitive to a change in either gene in either direction. We can therefore carefully use this derived value to put evolutionary pressures on the genes from which Fecundity is derived. In Level 3 Fecundity is used with the other two derived characteristics to apply such pressures. See below under C2 Penalty.

- Agility is a derived value. It is computed as (A * DeltaEPM) + (B * DeltaEPB) where A = .-2.20, B = 0.006667, DeltaEPM = (EPM(current) - EPM(default)) and DeltaEPB = (EPB(current) - EPB(default). There is a similar formula for each of Stamina and Fecundity as well. In a similar mode to Stamina, note a couple of things about this formula. First, when EPM(current) is equal to EPM(default) and EPB(current) is equal to EPB(default) then Agility is zero. This is intended to be so. As the EPM or EPB genes mutate and migrate away from their default strengths, the value of Agility will rise or fall. Second, A and B are scaling factors such that a mutation of either EPM or EPB either upwards or downwards by one increment will change the value of the Agility by roughly one. In other words Agility is designed to be equally sensitive to a change in either gene in either direction. We can therefore carefully use this derived value to put evolutionary pressures on the genes from which Agility is derived. In Level 3 Agility is used with the other two derived characteristics to apply such pressures. See below under C2 Penalty.

- The next item only appears for levels 3 and higher.

- C2 Penalty. When we allow the six type 2 (C2) genes (the regulatory genes called DAT, DET, RAT, RET, EPM and EPB) to mutate under evolutionary pressure, they have a tendency to grow to VERY large values. This is not dissimilar to what happens in biological systems. For example, the human genome is known to consist of roughly three trillion nucleotides, more than 90% of which is non-useful material. It is presumed that this number is still growing as time goes by. However, such large numbers are a problem for computers. Eventually, numbers that grow too large cause arithmetic overflow problems and the computer becomes confused. It is therefore useful to avoid excessively large numbers. There are two ways to do this. We can (a) simply disallow any numbers greater than a certain size. This is the mode of operation in the lower levels of PSoup. Base values cannot be larger than 50 and exponents cannot be larger than +-150. This introduces an unseen but ugly and unwanted 'edge effect' which, in some rare circumstances, distorts the results. However, even with this approach, very large numbers are not well represented. Or we can (b) introduce genetically-determined counter balancing pressures which will keep these numbers within reasonable bounds. In PSoup's higher levels we have implemented this approach. The User is able to activate something called the C2 Penalty. The formula for the C2 Penalty is Abs(Stamina + Fecundity + Agility) / Scaling Factor. The scaling factor is user-determined. The C2 Penalty is added to the Energy Per Move (EPM) on each turn, and that much additional energy is consumed by the bug on each and every turn. If the scaling factor has a value of '1', any and every mutation is quickly fatal. If the value of the scaling factor is very large, then it will have little or no effect.

The CurBug C2 sub-panel uses computer cycles in each turn of each bug during each tick of the PSoup clock, but only when displayed. If there are 200 bugs, this profile is updated 200 time per PSoup tick. If

you want a PSoup scenario to develop at the slowest possible speed, toggle the CurBug Panel on and toggle the 'Life Functions' panel on as well. However, if you want a scenario to develop at top speed, toggle the CurBug panel off. This can be done in several ways, as follows:

- Via the menu bar, use the mouse to click on 'Options', then on 'Display Current Bug'. The check mark will disappear, and so will the CurBug panel.
- Hold down the 'Alt' key and press 'o', followed by 'c'. These are the underlined letters in the menu bar items. 'O' and 'c' are called the accelerator keys for these menu items, and they allow you to run PSoup without a mouse, if you prefer keyboard actions. All menu items can be executed via accelerator keys in place of mouse clicks.
- Press 'F5'.
- Use the mouse to click on the 'Cur' button in the 'Options' toolbar.

In higher levels (after you have passed one or more LATs) you will be introduced to other sub-panels which occupy the same real estate as the CurBug C2 gene profile sub-panel. If you toggle any one of these alternate panels on, the CurBug C2 profile sub-panel will automatically toggle itself off. To clear all panels in this section of the screen (this is something you will want to do for fast or long runs) you will have to toggle the panel several times, but only for levels 2 and up.

The Current Bug C3 Genetic Profile sub-panel

This panel gives us a graphic representation of the strengths of each of the senses and higher functions that can be evolved by the bugs of PSoup. These are controlled by the Sensory+ (or type 3, or Capability) genes. The CurBug C3/C4 sub-panel displays each of the following features:

- The Bug's full name again - See the description of the C1 genetic profile for a detailed description of this.
- Eighteen One Sided Bar Graphs - These appear in pairs, the left one being for the 'Fight' (C3) gene, the right one being for the 'Flight' (C4) gene. At level four, ten of these bars appear, at level five, another four bars appear, and at level 6 the remaining four become visible. There is one pair of bar graphs for each capability as follows: Sight, Smell, Hear, Taste, Touch, XOver, Ovule, Herd and Think.
- The first five pair, those which appear at level 4, address the five senses. Each bar graph has two bars. The black bar shows the default strength. The red bar shows you the current strength of the gene. All values are either zero or a positive number. Zero is on the left. Remember, the strength of the gene is calculated as the base of the gene raised to the power of the exponent. The strength of the gene is the effective number for the expression of the gene. All six of these bars are scaled to fill the available width. Either the black bar, or the red bar, whichever represents the larger number, will fill the width. The other bar, the shorter one, is scaled to show its relative size. By this technique, you can quickly scan the profile and determine whether the gene has mutated upwards or downwards from the default strength. To the left of each pair of bars is the capability name. To the right of each bar is a number indicating the strength of the largest of the two quantities, default and actual.
- The next two pair of bar charts only appear for levels 5 and 6. Immediately below the separator is a group of four one-sided bar graphs labeled 'XOver', and 'Ovule'. When the XOver capability is activated for any bug, it is able to steal a copy of genetic code from any suitable 'mate' and use it to augment it's own code at the time of its next fission. When the Ovule capability is activated for any bug, it is able to reproduce sexually. If both 'XOver' and 'Ovule' are enabled, the cross-over is between the maternal and paternal genes of the diploid mother, rather than between the genes of the haploid mother and another haploid donor.
- The next two pair of bar charts only appear for level 6. Immediately below the separator is a group of four one-sided bar charts labeled 'Herd' and 'Think'. When Herd is activated for any bug, it is able to associate with other bugs of the same size, and has a tendency to group together into a loose herd. When Think is activated for any bug it has the ability to plan the best approach or escape route when it has encountered another bug.

The CurBug C3/C4 sub-panel uses computer cycles in each turn of each bug during each tick of the PSoup clock, but only when displayed. If there are 200 bugs, this profile is updated 200 time per PSoup tick. If you want a PSoup scenario to develop at the slowest possible speed, toggle the CurBug Panel on and toggle the 'Life Functions' panel on as well. However, if you want a scenario to develop at top speed, toggle the CurBug panel off. This can be done in several ways, as follows:

- Via the menu bar, use the mouse to click on 'Options', then on 'Display Current Bug'. The check mark will disappear, and so will the CurBug panel.
- Hold down the 'Alt' key and press 'o', followed by 'c'. These are the underlined letters in the menu bar items. 'O' and 'c' are called the accelerator keys for these menu items, and they allow you to run PSoup without a mouse, if you prefer keyboard actions. All menu items can be executed via accelerator keys in place of mouse clicks.
- Press 'F5'.
- Use the mouse to click on the 'Cur' button in the 'Options' toolbar.

In lower levels you were introduced to other sub-panels which occupy the same real estate as the CurBug C3/C4 gene profile sub-panel. If you toggle any one of these alternate panels on, the CurBug C3/C4 profile sub-panel will automatically toggle itself off. If you wish to clear the area, you will have to invoke the toggle several times until all alternate panels have been displayed and the blank panel is displayed.

The Average Bug Genetic Profile panels

These panels occupy the tall rectangular space at the left, abutting against the Title, PSoup and Status panels. The first of the three panels is entitled 'C1 gene profile of the "average" bug'. For short, call these the AveBug panels, as opposed to the CurBug panels. These panels are identical in appearance to the CurBug C1, C2 & C3/C4 gene profile panels, but their purpose is different. For a description of the visual aspects of these panels, see the section entitled The Current Bug Genetic Profile panels.

As for the CurBug panels, there are three AveBug sub-panels, one for each chromosome type:

- Type 1 (C1)
- Type 2 (C2)
- and Type 3/4 (C3/C4)

The Average Bug C1 Genetic Profile sub-panel

For a detailed discussion of the layout of this sub-panel see The Current Bug C1 Genetic Profile sub-panel.

This panel differs from the CurBug C1 sub-panel in the following ways:

- Because it represents all bugs, and not just one bug, no name is shown.
- It is updated only once per PSoup second, rather than when each bug takes a turn.
- The column of the table labeled 'Value' is computed as the average value of the strength of that gene across all active bugs in the bowl of PSoup. For example, the value of the 'F' gene is determined as the sum of the strength of all F genes in all active bugs, divided by the number of active bugs. The column labeled 'Pcnt' shows the relative strength of each average gene.

The AveBug panel uses computer cycles in each tick of the PSoup clock, but only when displayed, or when bugs change life status (i.e. are spawned or die). The draw on computer cycles is not excessive, and you may choose to leave this panel in full display during an extended run. However, if you want a scenario to develop at absolute top speed, toggle the AveBug panel off. This can be done in several ways, as follows:

- Via the menu bar, use the mouse to click on 'Options', then on 'Display Average Bug'. The check mark will disappear, and so will the AveBug panel.
- Hold down the 'Alt' key and press 'o', followed by 'a'. These are the underlined letters in the menu bar items. 'O' and 'a' are called the accelerator keys for these menu items, and they allow you to run

PSoup without a mouse, if you prefer keyboard actions. All menu items can be executed via accelerator keys in place of mouse clicks.

- Press 'F4'.
- Use the mouse to click on the 'Ave' button in the 'Options' toolbar.

In higher levels (after you have passed one or more LATs) you will be introduced to other panels which occupy the same real estate as the AveBug C1 gene profile. If you toggle any one of these alternate panels on, the AveBug C1 profile panel will automatically toggle itself off.

The Average Bug C2 Genetic Profile sub-panel

For a detailed discussion of the layout of this sub-panel see The Current Bug C2 Genetic Profile sub-panel.

This panel differs from the CurBug C2 sub-panel in the following ways:

- Because it represents all bugs, and not just one bug, no name is shown.
- It is updated only once per PSoup second, rather than when each bug takes a turn.
- The bar charts are computed as the average value of the strength for that gene across all active bugs in the bowl of PSoup. For example, the value of the 'DAT' gene is determined as the sum of the strength of all DAT genes in all active bugs, divided by the number of active bugs.

The AveBug panel uses computer cycles in each tick of the PSoup clock, but only when displayed, or when bugs change life status (i.e. are spawned or die). The draw on computer cycles is not excessive, and you may choose to leave this panel in full display during an extended run. However, if you want a scenario to develop at absolute top speed, toggle the AveBug panel off. This can be done in several ways, as follows:

- Via the menu bar, use the mouse to click on 'Options', then on 'Display Average Bug'. The check mark will disappear, and so will the AveBug panel.
- Hold down the 'Alt' key and press 'o', followed by 'a'. These are the underlined letters in the menu bar items. 'O' and 'a' are called the accelerator keys for these menu items, and they allow you to run PSoup without a mouse, if you prefer keyboard actions. All menu items can be executed via accelerator keys in place of mouse clicks.
- Press 'F4'.
- Use the mouse to click on the 'Ave' button in the 'Options' toolbar.

In higher levels (after you have passed more LATs) you will be introduced to yet another panel which occupies the same real estate as the AveBug C1 & C2 gene profiles. If you toggle any one of these alternate panels on, the AveBug C2 profile panel will automatically toggle itself off.

The Average Bug C3/C4 Genetic Profile sub-panel

For a detailed discussion of the layout of this sub-panel see The Current Bug C3/C4 Genetic Profile sub-panel.

This panel differs from the CurBug C3/C4 sub-panel in the following ways:

- Because it represents all bugs, and not just one bug, no name is shown.
- It is updated only once per PSoup second, rather than when each bug takes a turn.
- The column of the table labeled 'Value' is computed as the average value of the strength of that gene across all active bugs in the bowl of PSoup. For example, the value of the 'F' gene is determined as the sum of the strength of all F genes in all active bugs, divided by the number of active bugs. The column labeled 'Pcnt' shows the relative strength of each average gene.

The AveBug panel uses computer cycles in each tick of the PSoup clock, but only when displayed, or when bugs change life status (i.e. are spawned or die). The draw on computer cycles is not excessive, and you

may choose to leave this panel in full display during an extended run. However, if you want a scenario to develop at absolute top speed, toggle the AveBug panel off. This can be done in several ways, as follows:

- Via the menu bar, use the mouse to click on 'Options', then on 'Display Average Bug'. The check mark will disappear, and so will the AveBug panel.
- Hold down the 'Alt' key and press 'o', followed by 'a'. These are the underlined letters in the menu bar items. 'O' and 'a' are called the accelerator keys for these menu items, and they allow you to run PSoup without a mouse, if you prefer keyboard actions. All menu items can be executed via accelerator keys in place of mouse clicks.
- Press 'F4'.
- Use the mouse to click on the 'Ave' button in the 'Options' toolbar.

In higher levels (after you have passed one or more LATs) you will be introduced to other panels which occupy the same real estate as the AveBug C1 gene profile. If you toggle any one of these alternate panels on, the AveBug C1 profile panel will automatically toggle itself off.

The Event Counts panel (Options menu)

The Event Counts panel displays statistics on a variety of events, as listed below. When each event occurs, a counter is incremented, and, optionally, an identifying sound is issued. Each type of event has its own counter, and its own sound. There are essentially three types of events: (1) Instinctual urges of bugs based on the gene-derived abilities of the bugs to respond to circumstances in their environment, (2) interactions between the bugs and their environment that result in changes to the circumstances of the bug, or changes to the environment, and (3) events related to the implementation of PSoup that introduce hidden edge effects that may affect the course of the evolutionary development of the bugs.

When you open the Event Counts panel at the beginning of a scenario, no counts will be visible. Only those counts which rise above 0 are shown. The panel is self-spacing, so counts appear in the order given below. If a count first appears later, it will nevertheless appear in the proper place in the list, and the rest of the list will re-design itself.

Each count is identified by a three (or four) letter acronym.

The counts are:

Instinctual urges:

- IFPr - Instinctual urge to flee from a predator.
- IFPa - Instinctual urge to flee from a parasite.
- ISM - Instinctual urge to seek a mate.
- ISX - Instinctual urge to seek a cross-over of genetic material.
- ISP - Instinctual urge to seek for prey.
- ISH - Instinctual urge to seek for a host.
- ISA - Instinctual urge to seek algae.
- IHe - Instinctual urge to seek for a member of a herd.
- IEd - Instinctual urge to avoid an out of bounds edge cell.
- ITh - Instinctual urge to think before moving, by steering towards empty cells.

The words 'instinctual urge' has a special meaning here in PSoup. Of course, a bug cannot feel an urge, but its behavior must be modifiable by its genes if those genes are to have any outward expression. The ability of a bug to identify potential gene-based responses to environmental stimuli is the means by which its phenotype is expressed (for the most part).

All of the above instinctual urges are expressed as modifications to the strength of the first eight Palmiter genes (the genes found in the type 1 (C1) chromosome). For example, if the bug senses a predator in a cell a distance of three cells away (i.e. at its limit of detection) it may ignore it (no 'urge' is recorded) or it may

have an urge to flee from it (an urge is recorded). In general, a bug can only act upon one such instinctual urge per turn, so it must choose between the nine possible responses, ignore eight of them, and focus on one. PSoup has a pre-defined, unalterable order of precedence, which pre-determines which type of instinctual urge a bug will act on. This means the bug is driven by its genetic makeup and by its circumstances. It has no local control.

The order of priority is:

Seek mates (XOver or Ovule);
. Seek prey;
Seek Algae;
Flee predators and parasites;
Seek hosts;
Seek herd members; and
Avoid Edges.

The urge to 'think' before moving can be applied in any circumstance in association with any other instinctual urge.

When a bug has an urge to (e.g.) flee predators, it identifies all predators within its scan range (7 cells by 7 cells), computes a distance-reduced premium for each, and adds that value to the strength of the Palmiter gene which would lead the bug directly away from that predator. If a bug is surrounded by predators, all Palmiter genes may be so enhanced and it will not help the bug. It is nevertheless called an 'instinctual urge'. As another example, if a bug has an urge to Seek Algae, it notes all of the algae colonies within scan range, computes a premium for each, and adds the premium to the strength of the Palmiter gene which will lead the bug directly towards the algae, one such addition for each algae. Again, if it is surrounded by algae, it's instincts will tend to cancel each other and have less impact on the action of the bug. However, in the former case, if the bug also has an instinct to think, it will instinctively head for an opening in the wall of predators, rather than bumping into one of them, and so may enhance its chances of escape.

Numerable interactions - Instinctual urges merely modify the effective strengths of genes which have not yet been expressed. In each turn (i.e. each PSoup second), one and only one of the Palmiter genes will express itself just once (i.e. it will momentarily control the turning and stepping actions of the bug). Then the bug will engage in feeding, fighting, and procreational activities, in which it may die. These activities are not all in the mind of the bug, but rather alter the bug and its circumstances, either by the exchange of energy, or otherwise. These interactions between the bug and its environment are called 'numerable interactions'.

The numerable interactions are:

- NAE - Number of algae eaten;
- NHB - Number of hosts bitten;
- NPrB - Number of prey bitten;
- NPaE - Number of parasites eaten;
- NXO - Number of successful encounters in which genes were stolen from a target 'donor';
- NMa - Number of successful mating encounters in which genes were cooperatively donated.
- NFi - Number of times fission has occurred;
- NXF - Number of times XO-fission has occurred;
- NBi - Number of times birth has occurred;
- NND - Number of deaths due to natural causes (age > DAT or Nrg < DET);
- NPD - Number of deaths due to predation (predator eats prey, or host eats parasite);
- NHD - Number of deaths of a host bug due to parasitic attack.

The **overflow events** are:
If any of the following indicators appear in the counts panel your bowl of PSoup has encountered an overflow condition (an edge effect) in which the system has come up against one of the design limitations

of the program and the computer in which it is running. These conditions seriously interfere with the course of evolution of your bugs and may introduce large but irrelevant perturbations. When such events occur in large numbers, stop the bowl of PSoup, discard it, and start again. You may wish to first use the tinker panel to adjust some parameters which would help you avoid a re-occurrence of the overflow conditions. The overflow events are:

- OBu - Number of times a reproduction event has been delayed due to lack of room in the list of currently active bugs. PSoup supports a maximum of 500 bugs active at once. This limit is due largely to the huge array of combat profiles needed (500 * 500 = 250,000) required to support 500 bugs in a speedy scenario. More bugs could be supported in another implementation if we give up speed of processing per PSoup second. Reduction of the size of the bowl, or the energy factor, may reduce the likelihood of re-occurrence of this overflow condition.

- OTN - Number of times a reproduction event has been delayed due to lack of room in the list of tree nodes in the family tree. A scenario with many bugs, all with very short but productive lives, will cause a bottom-heavy family tree in which this event would start to occur. In other instances, a lack of competition for resources (ineffective natural selection) will cause a forest of tall skinny trees to replace a single family tree. OTN overflow will occur in this case.

- ONr - Number of times the Nrg chart is unable to display a bar due to lack of nodes in its tree. This should never happen in this implementation. Old bars are discarded ten at a time when the tree fills up.

- OBa - Number of time the base value for a simple gene could not mutate upwards due to the limit of a maximum value of 50. This may occur regularly in various scenarios if the C1, C2 and C3/C4 penalties are not activated and you undertake very long runs. These penalties serve to penalize extremely high strengths in any gene by exacting a price in terms of Nrg per move. The effect is to modify the natural evolutionary pressures causing selection against known problem configurations. However, the effect is very strong, and the likelihood of producing many interesting evolutionary possibilities is greatly reduced. If you think of the PSoup action as occurring on a gently sloped but bumpy incline, the penalties warp that incline to become a steep-walled pocket. The edges are no longer attainable (the goal of the penalty) but the course of evolution as it rushes to the bottom of the pocket is much more predictable and less interesting. It is suggested that user-constructed scenarios be run first without the penalties, and that the penalties be applied judiciously if and when needed. The effect of each penalty should be scaled to minimize its impact. Some scenarios cause exponential increases of the strength of genes in some chromosome types only. Note the type of chromosome in which the overflow is occurring, activate the penalty in that type of chromosome, and retry.

- OEx - Number of times the exponent value for a simple gene could not mutate upwards or downwards due to the limiting values of +150 and -150 respectively. As per the OBa event, these will occur regularly, and are controlled by the C1, C2 and C3/4 penalties.

In summary:

- during each PSoup second, a bug senses its surroundings and records an Instinctual Urge for each and every colony of algae, bug, edge cell or empty cell detected. A bug may record many instinctual urges in any turn. Each such ‘urge’ is counted.
- during each PSoup second, after a bug has moved, it will feed, mate, reproduce, or die. Each such event is counted.
- during each PSoup second, edge effects which arise from computer program or hardware design may prevent a bug from doing something that the simulation calls for. Each such artificial intervention due to edge effects is counted.

The Life Functions panels (Options menu)

There are five life function panels, each of which displays two types of information, as follows:

. - Sensing Function - this panel shows the scan area around the bug (i.e. a 7 by 7 area of cells) showing the as is and as sensed views. The ‘as is’ view shows all mud, bugs and algae in their natural colors as shown in the bowl of PSoup. The ‘as sensed’ view shows only those things which the bug is able

to sense, and therefore those things which might enter into decisions. (However, edges are shown whether or not the bug can sense them.) There is a legend which can be used to interpret the 'as sensed' view.

- Moving Function - this panel shows the scan area before and after the bug's move, showing an 'as is' view. The 'pre move' view from the moving function is identical to the 'as is' view from the sensing function panel. The 'post move' view is identical to the 'pre move' view, except that this current bug has moved (or not moved) as a result of the bug's movement life function. When the sensing function is being performed, the C1 Current Bug genetic profile (top right corner of the screen) shows the Palmiter gene strengths unmodified by any premiums. After the bug has sensed the scan area, in the order of precedence, it determines if any decisions are required, and adds premiums to the Palmiter genes. These premiums are reflected in the C1 Current Bug genetic profile. The moving function panel displays the direction in which the bug is pointing pre and post move, (as well as the Palmiter gene expressed which effected the move), and the change in Nrg of the bug. You may note that the value of EPM is displayed. If the standstill gene is expressed, the actual energy expended by a bug in a move is 1 + (EPM/2) + C1P + C2P + C3PP. If any other Palmiter gene is expressed, the actual energy expended by a bug in a move is 1 + EPM + C1P + C2P + C3PP. Of course, the chromosome penalties are only applied if they are activated.

- Feeding Function - this panel displays data about the bug and about any potential target prey pre and post feeding. The change in Nrg is displayed. The identity of the target bug, if any, is displayed.

- Repro Function - this panel displays data about mating activity (for XOver and Ovule enabled bugs) and reproductive events (fission, XO-fission, birth). For mating activities, the identity and stats of the target partner are displayed.

- Dying Function - if a bug dies of natural causes (old age or hunger) then its death will appear here. Bugs that die due to predation or parasitic attack are not the 'current bug' and so their deaths do not appear here. Their death is noted in the Feeding Function panel.

The Energy Chart panel (Options menu)

The Energy Chart in fact is a chart of two things, the relative energy in the different parts of the energy system, and the current number of bugs alive in the bowl of PSoup.

Number of Bugs

There are two sets of labels on the left edge of the chart. The labels in square brackets (e.g. [20]) refer to the number of bugs. Only the top and bottom of the chart are labeled, just to provide a sense of scale. The bottom always says [0], reasonably enough. The top is always a multiple of 20 just higher than the largest number of bugs currently being displayed. [20] is the smallest such maximum the chart will display. This chart is self-scaling, so when the number of bugs rises or drops, the scale changes.

The number of bugs is marked on the bar graph by a small black square.

Energy

In PSoup energy can exist in the nutritive mud (as ubiquitous non-localized energy), in algae colonies (as green cells) and in bugs. Each bar in the graph represents all three kinds of energy as colored bars. A legend identifying the colors is at the left of the panel. The color of the bars for 'energy in the bugs' changes as the color of the average bug changes, i.e. it changes as the average phenotype for the Palmiter genes changes.

There are two labels at the left. The bottom is always 0. The top label is a multiple of 1000 above the highest total energy in the system, as currently displayed in the chart. If the system is closed, this number will not change. If the system is open, the chart is self-scaling.

During the first two minutes of any scenario, the chart updates itself every PSoup second. Often there are critical changes during these first few seconds that a user may want to understand. After two minutes, the

changes come more slowly, and a new bar is added only once per PSoup minute. If the chart becomes full, the left-most ten bars are discarded and the chart shifted left.

The Population Profiles panels (Options menu)

The population profile panels are used to monitor the changing distribution of alleles within a population of bugs. Each type of chromosome has a series of small panels as follows:

- Palmiter genes (C1) - Nine mini-panels, one per gene;
- Regulatory genes (C2) - Six mini-panels, one per gene;
- Sensory+ genes (C3) - Twenty-six panels, one per gene type (GType).
- Sensory+ genes (C4) - Twenty-six panels, one per gene type (GType).
- Sensory+ genes (C3) - Nine mini-panels, one per capability type (CType);
- Sensory+ genes (C4) - Nine mini-panels, one per capability type (CType);

Each mini panel is a bar chart. The X axis shows either the gene strength (if mutation of the base is allowed) or the exponent value (if mutation of the base is not allowed). The Y axis shows the percentage of bugs in the current population which carry the allele represented by each bar.

Note that, for chromosome types 1 and 2 (C1 and C2) all bugs carry all genes, so the corresponding bars in the bar chart will add up to 100% for all such genes. For types 3 and 4 (C3 and C4) a gene may be presented by few or no bugs. All mini-panels are self-scaling. A maximum of 11 bars are available per chart, so if a gene has more than 11 alleles extant, or if two alleles have very close values (closer than 1/11th of the range) then they will be lumped together in a single bar. When mutation of the base is allowed, the bars tend to clump characteristically around 0.25, with odd alleles having values much higher. When mutation of the base in not allowed, and the exponents are plotted, the distribution often approaches a normal curve.

The range of the X axis is determined by the number of values plotted. If all alleles have a single common value, a single bar appears in the middle, so labeled, and an arbitrary range around that is displayed. If the alleles occur in two values, they define the top and bottom of the range and two bars at the extremities are displayed. If the alleles have three or more values, then the maximum and minimum values define the range.

The Legend panels (Options menu)

There are two panels in this group. The first panel provides specific details about the current bowl of PSoup and the defining parameters. This information is useful to know if you wish to enter the tinker panels and build your own scenario.

The second panel provides information about the colors of the bugs. If you have chosen to color the bugs by root, then all bugs belonging to any specific root will be of the same color. The Legend panel identifies the root colors being used.

Additionally, either legend panel will label the paddies if the bowl of PSoup has either cascading or independent paddies. This information is useful if you are moving bugs about in the bowl using the tinker capabilities.

PSoup Technical Glossary

PSoup Second

One 'tick' of the PSoup clock. Within one PSoup second the following things happen:

- Certain counters are initialized and the panel for the average bug genetic profile is displayed, if activated.
- PSoup loops through a list of all of the bugs giving each bug a turn consisting of seven steps, which are:
    - Setup for the bug, which includes refreshing the panel for the current bug genetic profile, if activated;
    - Completion of the Sensing function;
    - Completion of the Moving function;
    - Completion of the Feeding function;
    - Completion of the Reproduction functions which include XOver, Mating, Fission and Birth;
    - Completion of the Death function;
    - Cleanup, which includes removal of the body if there is one.
- When all bugs have had their turn, PSoup displays all of the other panels which are activated, and any energy which has accumulated in the nutritive mud is distributed as algae, if possible.

Panel Area

In PSoup, a panel area is a rectangular area of the screen set aside for displaying specific panels. There are six panel areas, each named for the default panel which appears within it, as follows:

- Title panel area - The title panel is the only panel which appears in this panel area and it is always displayed;
- PSoup Bowl panel area - The bowl of PSoup is the only panel which appears in this panel area and it is always displayed;
- Status panel area - Three panels are displayed in this area: the Status panel, the Legend panel and the Event Counts panel. Or it may be blanked.
- Tree panel area - This is the most active panel area. Up to eight panels may be displayed here, and some of these have up to nine sub-panels. The eight main panels using this area are Family Tree, Energy Chart, Life function panels (5 sub panels) and Population Profile panels (6 panels, 50 sub-panels).
- Average Bug panel area - Three panels are displayed here, showing the genetic profile of each of the three groups of genes.
- Current Bug panel area - Three panels are displayed here, showing the genetic profile of each of the three groups of genes.

Panel

In PSoup, a panel is a standard rectangular display of information about the current contents of the bowl of PSoup. A panel may be divided into rectangular sub-panels. A panel may be fixed (e.g. the title panel) or may be turned on or off. Each panel has an assigned panel area, and, if activated, will always appear in its assigned panel area. Several panels may be assigned to the same panel area, in which case only one will appear and the others will be toggled off.

Type 1 Chromosomes (C1)

In PSoup, genetic material is organized into genes and chromosomes. Each bug carries its genes in four major chromosomes. Type 1 chromosomes (labeled C1) consist of nine genes which are the primary determinants in the way a bug transports itself around the bowl of PSoup. They are also called the Transport genes, or the Palmiter genes, named after Dr. Michael Palmiter.

Type 2 Chromosomes (C2)

In PSoup, genetic material is organized as genes and chromosomes. Type 2 chromosomes (labeled C2) consist of six genes which are the primary determinants in the regulation of the bug's five life functions. These genes are also called the 'Regulatory genes'.

Type 3/4 Chromosomes (C3 & C4)

In PSoup, genetic material is organized as genes and chromosomes. Each bug's gene are contained in four types of chromosomes. Type 3 and type 4 chromosomes consist of a variable number of genes, with a limit of 26 genes per chromosome. There may be anywhere from 0 genes up to 26 genes in this chromosome. The number of genes is called the 'Complexity' of the chromosome. Complexity is a 'size' indicator. The complexity of all of the type 3 and type four chromosomes in a bug determines the overall complexity of the bug. The complexity of the bug is used in a difference function to determine relative species differences between two bugs. Each gene in a C3 or C4 chromosome has a type corresponding to a letter of the English alphabet. If a bug has a collection of C3 genes which spell part of the name of a capability, then the bug is enabled with that capability. For example, if a bug has two C3 genes of the types S, I, G, H and T, then the bug is visually capable. The C3 genes are called the fight genes, as they enable aggressive capabilities. The C4 genes are called the flight genes, as they enable defensive capabilities. The sum of the strengths of the 'fight' genes for a capability determine its ability to detect other bugs. The sum of the strengths of the 'flight' genes for a capability determine its ability to avoid detection by other bugs.

Capabilities

Capabilities are enabled via capability genes located in type 3 (C3) chromosomes within a bug's genetic material. The nine possible capabilities are listed in the C3/C4 genetic profile. A capability is enabled when a bug has sufficient genes of the correct type to spell part of the name of the capability. For example, suppose a bug has the following types of C3 genes: A, E, R, V & Z. With these letters you can spell part of H**EA**R, XO**VER**, O**V**UL**E** and H**ER**D, so those four capabilities are enabled for this bug. Note that only two letters of the set are needed to enable the capability. One letter is insufficient. This will be a diploid bug (OVULE-enabled) which is herbivorous (HEAR-enabled) and travels in herds (HERD-enabled) when not stressed. It will also benefit from genetic cross-over between its maternal and paternal chromosomes from time to time, or, rather, its offspring will (XOVER-enabled).

SIGHT

SIGHT is one of the nine capabilities which can be enabled via the C3 and C4 genes. When SIGHT is enabled in C3 a bug can see algae, other bugs, empty spaces, edges (if any). It is able to see all this within the scan area of the bug. The SIGHT capability has a fight component (C3) and a flight component (C4), each of which must be activated independently. The fight component enables detection of other bugs. The flight component enables the avoidance of detection by SIGHT by other bugs.

SMELL

SMELL is one of the nine capabilities which can be enabled via the C3 and C4 genes. When SMELL is enabled in C3, a bug can smell algae. It is able to smell only those within the scan area of the bug. The SMELL capability has a fight component (C3) and a flight component (C4). The fight component enables detection of other bugs by SMELL. It also enables the consumption of foul-tasting bugs, but only if the bug is carnivorous due to another capability (SIGHT, HEAR, TOUCH). The flight component enables the avoidance of detection by SMELL by other bugs. The fight component also enables a foul smell which causes attackers to spit out 80% of the bug, and to eventually leave the quivering remains alone, and alive.

HEAR

HEAR is one of the nine capabilities which can be enabled via the C3 and C4 genes. When HEAR is enabled in C3, a bug can hear other bugs. It is able to hear only those within the scan area of the bug. The HEAR capability has a fight component (C3) and a flight component (C4). The fight component enables detection of other bugs by HEAR. The flight component enables the avoidance of detection by HEAR by other bugs.

TASTE

TASTE is one of the nine capabilities which can be enabled via the C3 and C4 genes. When TASTE is enabled in C3, a bug can detect algae. It is able to taste only neighboring cells. The TASTE capability has a fight component (C3) and a flight component (C4). The fight component enables consumption of foul-tasting foods, but only if the bug is carnivorous due to another capability (SIGHT, HEAR, TOUCH). The flight component enables the production of foul-tasting enzymes unpleasant to an attacker. Attackers will spit out 80% of a foul-tasting bug, eventually leaving the sorry remains, still alive.

TOUCH

TOUCH is one of the nine capabilities which can be enabled via the C3 and C4 genes. When TOUCH is enabled, a bug can detect by touch algae, other bugs, empty spaces, edges (if any). It is able to detect this only in neighboring cells. The TOUCH capability has a fight component (C3) and a flight component (C4). The fight component enables weapons of attack such as claws and teeth. The flight component enables means of defense such as spines and shells. These are implemented as aggressive or defensive premiums during melee. This is an important capability.

XOVER

XOVER is one of the nine capabilities which can be enabled via the C3 and C4 genes. When XOVER is enabled in C3, a bug can obtain a copy of the genetic material of any other suitable bug to be used when it is able to undergo fission. (See XO-Fission). An XOver-enabled bug can detect a mate even if other senses are not enabled. Cross-over of the genetic material happens when the bug attempts to move into the cell occupied by the target bug.

OVULE

OVULE is one of the nine capabilities which can be enabled via the C3 and C4 genes. When OVULE is enabled in C3, a bug can detect and mate with suitable other bugs. An OVULE-enabled bug can sense a suitable mate, even though other senses are not enabled. Suitable mates must be of the same species (i.e. similar complexity or feeding habits), sexually enabled (i.e. OVULE enabled) and physically mature (old enough).

HERD

HERD is one of the nine capabilities which can be enabled via the C3 and C4 genes. When HERD is enabled in C3, a bug can exhibit a tendency to stay close to other bugs of the same species (i.e. of similar complexity). HERD is a higher function. A bug must be able to sense other bugs (SIGHT, HEAR, TOUCH) in order to benefit from HERD.

THINK

THINK is one of the nine capabilities which can be enabled via the C3 genes. When THINK is enabled in C3, a bug can plan to make better use of empty cells when approaching or avoiding other bugs. THINK is a higher function. A bug must be able to sense other bugs (SIGHT, HEAR, TOUCH) in order to benefit from THINK. THINK can be enabled in C4, but it is phenotypically inert.

Complexity

Every bug has its genetic material organized into four types of chromosomes. Type 3/4 chromosomes contain the 'Capability' genes, from 0 up to 104 genes (up to 26 per chromosome, up to 4 type 3/4 chromosomes) which can enable sensory capabilities and higher functions. The number of capability genes is called the complexity. A bug's complexity can therefore be between 0 and 104. Complexity is used to determine the size relationship between bugs. If a bug can sense other bugs (via SIGHT, HEAR, TOUCH, XOVER, OVULE), then it determines the complexity of the other bug and its species classification. The other bug can be one of:

- Parasite - All bugs with complexity less than 1/4 of the complexity of this bug.
- Prey - All bugs with complexity between 1/4 and 1/2 of the complexity of this bug.

- Same species - All bugs with complexity between 1/2 and 2 times the complexity of this bug.
- Predators - All bugs with complexity between 2 and 4 times the complexity of this bug.
- Hosts - All bugs with complexity greater than 4 times the complexity of this bug.

Note that these determinations are reciprocal. If this bug determines that another bug is prey, the other bug sees this bug as a predator.

The smallest bug that can exhibit parasitic behavior has HEAR, SIGHT or TOUCH enabled and has a complexity of 2.

Bugs of the same sub-species may exhibit genetic cross-over, sexual mating, and herding behavior.

Parasite

This is a sub-species classification. Any other bug which has a complexity less than 1/4 of the complexity of the current bug is classified (by the current bug) as a parasite.

Prey

This is a sub-species classification. Any other bug which has a complexity between 1/4 and 1/2 of the complexity of the current bug is classified (by the current bug) as a prey.

Predator

This is a sub-species classification. Any other bug which has a complexity between 1/2 and 2 times the complexity of the current bug is classified (by the current bug) as a predator.

Host

This is a sub-species classification. Any other bug which has a complexity less than 4 times the complexity of the current bug is classified (by the current bug) as a host.

Fight

'Fight' has several manifestations within PSoup but they all derive from its role in a C3 chromosome. The Fight gene is any active gene found in a C3 chromosome. The strength of the Fight gene is summed with the strength of other active fight genes, for each capability, to give an overall fight strength for each capability. Specifically, the strength of the fight component of capability type (CType) SIGHT is computed as the sum of the fight genes for all of the GTypes of which SIGHT is constituted. The fight component of any CType is phenotypic and determines the relative aggressive abilities of the bug, when compared with the similar C4 (defensive, or flight) component of the genetic makeup of any opposing bug.

Flight

'Flight' has several manifestations within PSoup but they all derive from its role in a C4 chromosome. The Flight gene is any active gene found in a C4 chromosome. The strength of the Flight gene is summed with the strength of other active flight genes, for each capability, to give an overall flight strength for each capability. Specifically, the strength of the flight component of capability type (CType) SIGHT is computed as the sum of the flight genes for all of the GTypes of which SIGHT is constituted. The flight component of any CType is phenotypic and determines the relative defensive abilities of the bug, when compared with the similar C3 (offensive, or fight) component of the genetic makeup of any opposing bug.

Scan Area

When a bug in PSoup has one of the senses enabled, it has the ability to scan a square area of cells in its immediate vicinity. This area is called the scan area. If wrap mode is turned on, the scan area includes the appropriate cells on the other side of the bowl of PSoup. If wrap mode is turned off, the scan area is limited by the edge of the bowl (of course). For the senses of SIGHT, SMELL and HEAR the scan area is

a square of 7 cells by 7 cells, 49 cells in all. For the senses of TASTE and TOUCH the scan area is a square of 3 cells by 3 cells, 9 squares in all.

If you enable the Life Function panels, and advance PSoup either by one bug, or by one life function, then you will see the scan area displayed in the Sensing and Moving function panels, together with the bug's interpretation of the material scanned.

Fission

This is one of three means by which bugs reproduce. In fission, a bug splits into two bugs and the parent disappears. All bugs can reproduce by fission by default. If a bug has XOVER or OVULE enabled, then it can no longer reproduce by fission. When a bug fissions, it produces two daughters of equal energy, of age zero, and with almost identical genetic makeup. The genetic material is a duplicate of the mother's genetic material, but mutated. Mutation only occurs upon reproduction. In higher levels it is possible, but not likely, that a daughter's genetic makeup would be identical either to the other daughter's, or to the mother's. Both daughters occupy the same cell in which the mother was previously. A daughter in such a double-occupancy cell is forced to move (even if the standstill gene expresses itself and says not to move) into any nearby cell with fewer than two bugs. By this means, sessile bugs are forced to spread out over the mud.

XO-Fission

XO-Fission is a variation on fission used by bugs which have the XOVER capability enabled. Such a bug will search for any other bug of the same sub-species and will appropriate a copy of its genetic material and store it in it's body in a special code called the cross-over code. When the bug matures and is able to reproduce (i.e. it meets the conditions mediated by the RAT and RET genes) it mixes the cross-over with its own code, then copies its (possibly modified) genetic code into each of its daughters, then mutation occurs. Each daughter then has genetic code which is a mixture of (a) its mothers code (b) the cross-over code, and (c) the results of mutation, if any. In other ways, XO-Fission is like Fission. The donor bug for the cross-over code may be any haploid bug of suitable complexity.

Birth

Birth is one of three means of reproduction. The other two are Fission and XO-Fission. A bug may give birth to up to two daughters, but one at a time. Upon the birth of the first daughter, the mother continues to live. Upon the birth of the second daughter, the mother dies. In order to give birth, a bug must have OVULE enabled, and it must detect, chase and mate with another OVULE-enabled bug of the same species classification.

Upon mating, a haploid gamete formed from the other bug's genetic code is placed in a special code called the spermatic material. The sperm is formed by randomly selecting one of each of the four types of chromosomes from among the maternal and paternal chromosomes of the donor bug.

A mated bug is said to be pregnant. Upon successful exhibition of the conditions of birth, as mediated by the RAT and RET genes, the bug gives birth to one daughter, as follows. If the bug is also XOVER-enabled, then the maternal and paternal genes of the mother cross-over, forming two haploid crossed gametes, which are used to produce two haploid ova. One is discarded. If the bug is not XOver-enabled, then the bug produces two non-crossed gametes, which are then used to produce two haploid ova. Again, one is discarded. In either case, the haploid ovum is copied into the daughter as maternal chromosomes. The haploid sperm is copied into the daughter as the paternal chromosomes. Then the daughter's genetic code is mutated. The mother is then flushed of all spermatic material and she must mate again to produce a second daughter.

Gene Type (GType)

All genes are typed by their location in the chromosome.

For C1 chromosomes, the GType (gene type) is the ordinal number of their location. The first gene is at position zero and is called gene0. The last one, the ninth one, is at position 8 and is called gene8. These genes also have names based on their phenotypic effect. The gene at position causes a bug not to turn, but to go straight forward, and so is called the Forward gene, or the F gene. The GTypes of the C1 genes are:

GType	Acronym	Name
gene0	F	Forward
gene1	FR	Forward right
gene2	R	Right
gene3	BR	Back right
gene4	B	Back
gene5	BL	Back left
gene6	L	Left
gene7	FL	Forward left
gene8	SS	Stand still

For C2 chromosomes, the genes are each named for the trait they represent. The six genes, in order, are:

GType	Acronym	Name
gene0	DAT	Death age threshold
gene1	DET	Death energy threshold
gene2	RAT	Reproductive age threshold
gene3	RET	Reproductive energy threshold
gene4	EPM	Energy per move
gene5	EPB	Maximum energy per bug

For C3 and C4 chromosomes there are 26 GTypes ranging from gene0 (the first) to gene25 (the last). These are also given names which are letters from the English alphabet from A to Z, respectively.

Capability Type (CType)

There are nine capability types called SIGHT, SMELL, HEAR, TASTE, TOUCH, XOVER, OVULE, HERD and THINK. Each of the nine capabilities can be enabled in either fight mode (via the C3 chromosome) or in flight mode (via the C4 chromosome) For any of these capabilities to be enabled, a bug must harbor at least two activating genes of the component gene types required to spell the capability. For example, to be able to hear, a bug must have at least two C3 genes out of the list of types H, E, A and R. The first five capabilities are senses. The next two are means of reproduction. The last two are higher functions. In general, it is expected that bugs will evolve the senses first, the higher means of reproduction next, and the higher functions last. Of course, the user can build bugs with higher functions and inject them into a bowl of PSoup to see how well they manage.

PSoup/Biological Terminology Comparisons

autotrophic

In Nature
Self nourishing; said of plants that make their own food by photosynthesis, and of bacteria that can grow without organic carbon and nitrogen compounds: distinguished from heterotrophic. Synonym - holophytic.
In PSoup
Able to self nourish by photosynthesis, drawing sustenance directly from the nutritional mud.

heterotrophic

In Nature
Obtaining nourishment chiefly, or entirely, from complex organic substances: distinguished from autotrophic. Synonym - holozoic.
In PSoup
PSoup has avoided the use of the words heterotrophic or holozoic. PSoup uses a variety of analogies for the feeding habits of different types of its denizens. PSoup has made up a new word algivore to describe organisms which live entirely on the algae which grows on the nutritive mud in a bowl of PSoup. Algivorous organisms are analogous to heterotrophic organisms in nature.

alga (algae)

In Nature
Any plant of a sub-division (Algae) of thallophytes, consisting of primitive, chlorophyll-bearing plants widely distributed in fresh and salt water and moist lands, including the seaweeds, kelps, diatoms, pond scums, and stone worts.
In PSoup
The greenish immobile edible material that is placed randomly about the bowl of PSoup to provide energy to the denizens of the bowl of PSoup. While there is, obviously, intended to be an analogous relationship between algae in the natural world and the algae of PSoup, one should not push the analogy too hard. In other words, if you want to learn more about algae, don't use PSoup as your source. The generic word for a denizen of a bowl of PSoup is a 'bug'. Some bugs are variously called cruisers, bushes, peas, lynx, rabbits, or any number of other names. In each case, there is an intended analogy with nature respecting only some aspects of the behavior of the bugs. The same type of loose analogy is intended with algae. In some scenarios, algae may be analogous to grass, humus in the soil, phytoplankton, or protozoans. In all cases, the algae of PSoup is analogous to the bottom link of the food chain. For more about the 'PSoup Analogies' see the article in the PSoup Help System under PSoup Theory and Practice.

lichen

In Nature

Any of various flowerless plants (class or order of lichenes) composed of fungi and algae in symbiotic union, commonly growing in flat greenish fray, brown, yellow or blackish patches on rocks, trees, etc.
In PSoup
Synonymous with algae.

algivore

In Nature
This is not a real word, and cannot be found in any dictionary.
In PSoup
This is a made-up word meaning bugs that eat algae, and are otherwise next in the food chain, being suitable as prey for carnivorous or omnivorous bugs. All bugs which are haploid and unable to see, hear, smell, taste or touch are algivorous. It includes some bugs which reproduce via fission, and some bugs which reproduce via cross-over fission.

Algivorous bugs at Levels 4-6 may have complexity, and may therefor see other bugs as 'prey', but they are unable to eat them.

carnivore

In Nature
Carnivore - Any of an order (Carnivora) of flesh-eating mammals, including cats, dogs, bears, seals, etc.
Carnivorous
1. Eating or living on flesh.
2. Of or pertaining to Carnivora.
In PSoup
Carnivore - Any bug which has the ability to hear or touch (inclusively) but does not have the ability to see, or smell or taste, as enabled by the third type of chromosome (the C3 chromosome). Such bugs are unable to eat algae. They must find, pursue, capture and eat other bugs to survive. Prey must be of lesser complexity, having a complexity between 1/4 and 1/2 of the complexity of the carnivore.

herbivore

In Nature
1. An animal which feeds on vegetable matter; plant eating.
2. An animal belonging to a group or division of mammals (now called ungulata) that feed mainly on herbage, as cows, horses, camels, etc.
In PSoup
Any bug which has the ability to smell or taste (inclusively) but does not have the ability to see, or hear or touch, as enabled by the third type of chromosome (the C3 chromosome). Such bugs are unable to eat other bugs. They must sense, move towards, and eat algae to survive. They can be distinguished from algivores as follows:

Algivores	**Herbivores**
Unable to sense anything	Able to sense algae
Eat only algae	Eat only algae

Always haploid (asexual) Some are diploid (sexual)

omnivore

In Nature
1. Eating both animal and vegetable food.
2. Eating food of all kinds indescriminantly.

In PSoup
Any bug that is able to see, or which is otherwise both a carnivore (able to hear or touch) and a herbivore (able to smell or taste). Omnivores may be haploid (asexual) or diploid (sexual). As for carnivores, prey must be of lesser complexity, having a complexity between 1/4 and 1/2 of the complexity of the carnivore.

insectivore

In Nature
Any of an order (Insectivora) of insect-eating mammals as shrews, moles, hedgehogs, etc.
In PSoup
Synonymous with carnivore.

fungus

In Nature
Any of a subdivision (Fungi) of thallophytes, comprising non-flowering plants that have no chlorophyll, usually reproduce asexually, and grow on dead organic matter, or live parasitically, and including mushrooms, molds, or mildews.
In PSoup
Synonymous with algae.

parasite

In Nature
An animal or plant that lives in or on another organism, the host, at whose expense it obtains nourishment and shelter.
In PSoup
A temporary classification of a remote bug by the current bug (the bug which is currently completing its move under the transition function). To be considered a parasite, the remote bug must be of lesser complexity, less than 1/8th the size of the current bug. To pose any threat, the parasite must be carnivorous or omnivorous. Algivorous and herbivorous parasites pose no threat. Bugs do not flee from parasites, but THINKing bugs can dodge around parasites in the pursuit of other goals.

host

In Nature
Any living plant or animal from which a parasite obtains nourishment and/or protection.
In PSoup
A temporary classification of a remote bug by the current bug (the bug which is currently completing its move under the transition function). To be considered a host, the remote

bug must be of greater complexity, greater than 8 times the complexity of the current bug. To be of interest, the current bug must be carnivorous or omnivorous. Parasites will pursue host bugs and bite them repeatedly.

predator

In Nature
Living by preying on others, as a beast or bird; raptorial.
In PSoup
A temporary classification of a remote bug by the current bug (the bug which is currently completing its move under the transition function). To be considered a predator, the remote bug must be of greater complexity, greater than 2 times the complexity of the current bug, but less than four times the complexity of the current bug. To be a threat, the remote bug must be carnivorous or omnivorous. Predators will pursue prey bugs and eat them, or bite them repeatedly.

prey

In Nature
Any animal seized by another for food.
In PSoup
A temporary classification of a remote bug by the current bug (the bug which is currently completing its move under the transition function). To be considered as prey, the remote bug must be of lesser complexity, less than 1/2 the complexity of the current bug, but greater than 1/4 of the complexity of the current bug. To be of interest, the current bug must be carnivorous or omnivorous. Predators will pursue prey and eat them or bite them repeatedly.

gamete

In Nature
Either of two mature reproductive cells, an ovum or sperm, that uniting produce a zygote.
In PSoup
An ordered set of up to four chromosomes of four different types being transport, regulatory, fight and flight (C1, C2, C3 and C4 respectively). Gametes are produced in a variety of circumstances to complete reproductive processes.

(a) Fission - a haploid bug with one (C1 only), two (C1 and C2 only) or four (C1, C2, C3 and C4) chromosomes will produce two gametes, each containing an exact and complete copy of the mother's genes, and these two gametes will then mutate and each becomes a daughter of the mother.

(b) Fission with cross-over - a haploid bug with four (C1, C2, C3 and C4) chromosomes (being XOver-enabled in the third chromosome) will steal a gamete, a complete copy of another haploid bugs genes which is called the cross-over code. It will then execute a cross-over function in which part of its own genetic code may be exchanged with genetic code from the cross-over gamete. At this point the (possibly modified) cross-over gamete is discarded. The bug, now in possession of (possibly modified genetic code of its own)

proceeds with normal fission. That is, it will produce two gametes, each containing an exact and complete copy of the mother's (new) genes, and these two gametes will then mutate and each becomes a daughter of the mother.

(c) Birth - a diploid bug (being OVULE-enabled in the third chromosome) with eight chromosomes in four homologous pairs (C1, C2, C3 and C4) will mate with another OVULE-enabled bug of suitable complexity and status, obtaining a male gamete (sperm). The male gamete will be formed by the donor bug by random selection of four chromosomes from among the four pairs of homologous chromosomes. In a similar fashion, the current bug forms a female gamete (ovum) using her own pairs of homologous chromosomes as source. The male gamete is a copy of a subset of the donor bugs genetic code. The female gamete is a copy of a subset of the mother bugs genetic code. These gametes undergo mutation, then are united to form a single genetic unit with eight chromosomes in four homologous pairs. The compete set of genes is inserted in a daughter and a daughter is born.

(d) Birth with cross-over -- a process identical to the process for birth without cross-over, as described above except (1) the mother is XOver-enabled in the third chromosome; and as the mother produces the ovum, there is an extra step in which two proto-gametes are formed, one containing her maternal chromosomes, and one containing her paternal chromosomes, and these proto gametes are crossed over prior to the production of an ovum. Not only is the ovum a mix of maternal and paternal genetic material from the mother, but each chromosome within the gamete may contain a combination of both maternal and paternal genetic material from within the mothers 8 chromosomes. Once the ovum is formed, normal birth proceeds as described above.

gene

In Nature
A specific sequence of nucleotides in DNA or RNA that is located in the germ plasm usually on a chromosome and that is the functional unit of inheritance controlling the transmission and expression of one or more traits by specifying the structure of a particular polypeptide and especially a protein or controlling the function of other genetic material
In PSoup
A sequence of numbers located in the genetic store of a bug that is the functional unit of inheritance controlling the transmission and expression of one or more traits by specifying the nature and strength of a particular gene loci, or controlling the function of other genetic material.

Gene expression

In Nature
The circumstance in which a specific gene plays a role in the development and/or functioning of an organism. A gene which is not expressed is suppressed and plays no role.
In PSoup
In PSoup, gene expression can have two meanings:

(1) The circumstance in which a specific gene plays a role in the development and/or functioning of a bug. In PSoup some genes are suppressed in some circumstances as follows:
(i) - the standstill gene is suppressed in levels 1 and 2, and in upper levels if the toggle is switched to 'inert' in the ‘Tinker with the environment’ panel.
(ii) - the type 2 (C2) chromosomes genes do not act as genes, but rather play a role similar to the ‘rules of chemistry’ or ‘rules of physics’ in Level 1. They are not suppressed. They are simply not genes. In levels 2-6 they act as genes, and they are never suppressed.
(iii) - the type 3 (C3) and type 4 (C4) chromosomes genes are not genes levels 1-3. They are genes in Levels 4-6. However, some of them are inactive 'trash' genes. That means that they occupy real estate in the chromosome, but are not active in any way. They must be activated by a complexity mutation. Some C3 and C4 genes have a greater role than others in higher levels, but none are suppressed. E.g. all play a role in determining complexity.
(iv) - in a diploid bug in levels 5 and 6, some genes may be dominant, and some genes recessive. The dominant genes are expressed. The recessive genes are not expressed, but do contribute to the complexity of the bug.
(2) A gene expresses itself by determining the actions taken by a bug as it executes its five life functions when it has a turn, or the actions taken to defend itself when attacked by another bug. See the Role of Chance.

gene flow

In Nature
The passage and establishment of genes typical of one breeding population into the gene pool of another by hybridization and backcrossing
In PSoup
The passage and establishment of genes typical of one breeding population (as may be found in one paddy) into the gene pool of another breeding population by hybridization or by cross-over.

In biological terms, gene flow defines species. This is the most strict definition of species. All organisms which participate in a gene flow are considered to be in the same species. By this rule, PSoup exhibits true species only when diploid bugs appear and have variant feeding habits. For example, diploid carnivores and diploid herbivores do not mate, and there is no gene flow between them.

gene frequency

In Nature
The ratio of the number of a specified allele in a population to the total of all alleles at its genetic locus
In PSoup
The ratio of the number of chromosomes carrying a specified allele in a population over the total number of all chromosomes in the population (a population, in this case, being all living chromosomes of all bugs in the bowl of PSoup). A bug carries one allele in its

maternal genes and one allele in its paternal genes, if it has paternal genes (diploid bugs only).

gene pool

In Nature
The collection of genes of all the individuals in an interbreeding population
In PSoup
The collection of genes of all the bugs in an interbreeding population. Interbreeding populations may be defined by the bowl of PSoup as a whole, or by independent paddies within the bowl of PSoup.

gene therapy

In Nature
The insertion of normal or genetically altered genes into cells usually to replace defective genes especially in the treatment of genetic disorders
In PSoup
The use of the TinkerBug Wizard to alter the genetic makeup of a bug to improve it's chances of survival.

lethal gene

In Nature
A gene that in some (as homozygous) conditions may prevent development or cause the death of an organism or its germ cells
In PSoup
A gene that in some conditions (primarily determined by competition with contemporaneous bugs) may prevent the gathering of energy, or cause the early death of a bug. For example, a C2.EPM allele with a high strength is usually lethal.

regulatory gene

In Nature
A gene that regulates the expression of one or more structural genes by controlling the production of a protein (as a genetic repressor) which regulates their rate of transcription
In PSoup
1 A gene in a type 2 chromosome (C2) which controls life functions, of which there are six:

- Death Age Threshold (DAT)
- Death Energy Threshold (DET)
- Reproduction Age Threshold (RAT)
- Reproduction Energy Threshold (RET)
- Energy Per Move (EPM)
- Maximum Energy Per Bug (EPB)

2 A gene in a type 3 or type 4 chromosome (C3 or C4) which, in combination with other C3 or C4 genes, alters the execution of a bug's life functions. Some C3/C4 gene types are inert, and play no regulatory role (e.g. GType = Z). Other C3/C4 gene types are exceptionally active, and have an effect on many capabilities (e.g. GType = S affects sight, smell, taste while GType = T affects sight, taste, touch and think). Some

capabilities which are derived from C3/C4 genes (e.g. SIGHT) enable other capabilities (E.g. Herd or Think) which would not operate if the enabling capability were not active. While C3/C4 genes clearly have a regulatory role, they are not normally referred to as regulatory genes in PSoup.

structural gene

In Nature
A gene that codes for the amino acid sequence of a protein (as an enzyme) or for a ribosomal RNA or transfer RNA
In PSoup
A gene that plays no regulatory role but nevertheless plays a strong role in determining a bug's behavior. The Palmiter genes are structural genes. Some C3/C4 genes are structural (e.g. GType = S) while others are inert (e.g. GType = Z). It is interesting to watch the effects of evolutionary pressures on structural and regulatory genes as compared to inert genes.

allele

In Nature
1 Any of the alternative forms of a gene that may occur at a given locus
2 Either of a pair of alternative Mendelian characters (as smooth and wrinkled seed in the pea)
In PSoup
1 Any of the alternative forms of a gene having a common genetic locus, but differing dominance type or strength.
2 Either of a pair of contemporaneous phenotypes.

multiple allele

In Nature
An allele of a genetic locus having more than two allelic forms within a population
In PSoup
An allele of a genetic locus having more than two allelic forms (strengths) within a bowl of PSoup.

heterozygous

In Nature
Having the two alleles at corresponding loci on homologous chromosomes different for one or more loci
In PSoup
PSoup In a diploid bug, when the two genes at corresponding loci on homologous chromosomes differ in strength or dominance.

chromosome

In Nature
One of the linear or sometimes circular DNA-containing bodies of viruses, prokaryotic organisms, and the cell nucleus of eukaryotic organisms that contain most or all of the genes of the individual

In PSoup
One of the four main groups of genes. The C1 (Palmiter) genes form one chromosome. The C2 (Regulatory) genes form a second chromosome. The C3 and C4 genes (Capability genes, or Sensory+ genes) form the third and fourth chromosomes.

euchromatin

In Nature
The genetically active portion of chromatin that is largely composed of genes
In PSoup
The non-inert genes within a bug. This includes all C1 (Palmiter) genes except for the Stand Still gene when it is de-activated. It includes all C2 (Regulatory) genes. It includes all C3 and C4 (Sensory+) genes which have been activated, but excludes the inactive trash genes which fill the chromosome awaiting activation.

eukaryote

In Nature
An organism composed of one or more cells containing visibly evident nuclei and organelles
In PSoup
A bug. In contrast, algae is prokaryotic.

prokaryote

In Nature
A cellular organism (as a bacterium or a blue-green alga) that does not have a distinct nucleus
In PSoup
A colony of algae. In contrast, a bug is eukaryotic.

mitosis

In Nature
A process that takes place in the nucleus of a dividing cell, involves typically a series of steps consisting of prophase, metaphase, anaphase, and telophase, and results in the formation of two new nuclei each having the same number of chromosomes as the parent nucleus
In PSoup
Mitosis occurs with diploid organisms. In PSoup, bugs are haploid.

Nevertheless, similar processes exist in PSoup.

Fission - It involves two phases (1) fission by which the code is copied and inserted into the two daughters, and (2) mutation by which the genes of each daughter are independently modified according to the chemical rules which govern such mutations.

XO-Fission - It involves three phases (1) cross-over of genetic material between the mother's own genetic code and the genetic code contained in the cross-over material donated by (stolen from) the father, according to the extant rules of chemistry; (2) fission,

as described above, but with the crossed genetic material undergoing fission, and (3) mutation by which the genes of each daughter are independently modified according to the chemical rules which govern such mutations.

Birth - It involves four phases, being (1) if the XOver capability is enabled, the cross-over of genetic material between the mother's own homologous chromosomes, according to the extant rules of chemistry; (2) fertilization of the mother's genetic material by the father's genetic material, by which some chromosomes are replaced by the father's chromosomes, according to the rules of chemistry extant, (3) insertion of one copy of the resulting genetic material into one daughter, and (4) mutation of the daughter's genes.

In both XO-Fission and Birth, if a bug comes of age and is ready to give birth but no suitable donor (source) of genetic material has been made available, the bug invokes parthenogenesis and takes a copy of its own genetic material to use in place of the required donor material. The machinery continues as for XO-Fission or Birth, but the genetic outcome is identical to Fission.

cell

In Nature
A small usually microscopic mass of protoplasm bounded externally by a semi-permeable membrane, usually including one or more nuclei and various other organelles with their products, capable alone or interacting with other cells of performing all the fundamental functions of life, and forming the smallest structural unit of living matter capable of functioning independently [cell illustration]
In PSoup
1 A small rectangular location within a bowl of PSoup which may contain none, one or all of nutritive mud, a colony of algae, and one or two bugs.
2 A unit of genetic material containing four chromosomes, also called a bug.

mutation

In Nature
1 A relatively permanent change in hereditary material involving either a physical change in chromosome relations or a biochemical change in the codons that make up genes.
2 The process of producing a mutation
3 An individual strain or trait resulting from mutation
In PSoup
1 A permanent change in one of the numbers that make up a gene, causing the gene to activate (in the case of C3 and C4 genes) or to change its strength.
2 The process of producing a mutation.
3 A trait resulting from mutation.