Int J Comput Math Learning (2007) 12:167–171
DOI 10.1007/s10758-007-9118-y

Computational Diversions: Circus Maximus

Michael Eisenberg

Published online: 18 July 2007

One of the tricky aspects of a "computational diversions" column is determining just how much programming should, or plausibly can, be included in a particular example. On the one hand, it is certainly possible to suggest complex research-level projects that require pages and pages of procedures; but in that case, we may be out of the realm of computational diversions and into the (admittedly nearby) realm of computational obsessions. On the other hand, very simple programs with very interesting behavior seem hard to come by.

Or so I thought. Actually, this column describes just such a simple-but-interesting algorithm taken from the book *How Nature Work*s, by the late Per Bak (1996) The book is accessible in style, but highly provocative and challenging in content—a difficult combination to pull off. In the book, Bak, a physicist, focuses his attention on "self-organized criticality": a state toward which some complex systems seem to evolve, and in which the system can be perturbed—sometimes to large and unexpected effect—by small external influences. Bak applies this idea to topics as diverse as earthquakes, mass extinctions, and free-market economies.

One of the especially wonderful aspects of *How Nature Works* is the way in which it weaves surprisingly straightforward computational models into the text: it's the sort of book that could convince even a computerphobic reader to take up programming, just on a lark. The algorithm that we'll look at here is arguably the simplest in the book; in fact, Bak describes it as "probably simpler than any model that anybody had ever [before] written for anything," and he may be right. I can hardly hope to improve on Bak's prose summary of the model, quoted below:

> *Random numbers are arranged in a circle. At each time step, the lowest number, and numbers at its two neighbors, are each replaced by new random numbers. That's all! This step is repeated again and again.* [p. 138; emphasis in the original]

That really is just about all that needs to be said, though a few additional details should be added. First, we have to make a choice of how many numbers to arrange in a circle; Bak's experiments used a circle of 300 numbers, and my own experiments have run pretty well

M. Eisenberg (✉)
University of Colorado, Boulder, CO, USA
e-mail: ijcml-diversions@ccl.northwestern.edu

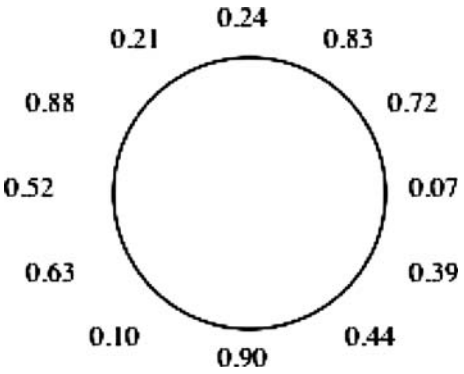
with 200. In any event, it should be intuitively plausible that if you choose a very large set of numbers to arrange in a circle (say, 1,000), then the algorithm will run more slowly: in the most straightforward version of the algorithm, you have to look through all 1,000 numbers at every iteration to find the lowest number in the circle.

We also have to be specific about what we mean by "random numbers." Bak's experiments use numbers between 0 and 1; he isn't more specific than that in the text, but in my own experiments I've used sets of 1,000 numbers for quick-and-dirty experiments (namely, the values 0, 0.001, 0.002... 0.999) and 10,000 numbers for finer-grained experiments (the values 0, 0.0001, 0.0002, ... 0.9999). In Scheme (my preferred language), the natural way of choosing a new random number in this fashion can be expressed:

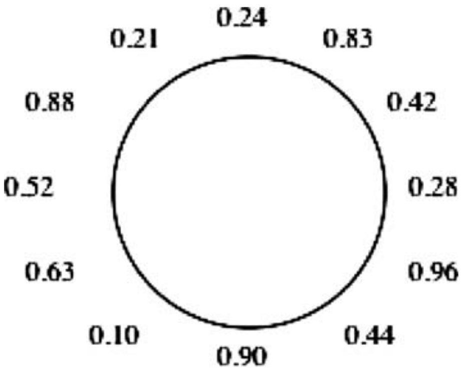
```
(define (choose-random-value grain)
   (/ (random grain) grain)
```

Calling this procedure with a grain parameter of 1,000 will yield elements of the "quick-and-dirty" set of values mentioned above.

Before proceeding any further, it may be worth illustrating Bak's algorithm with a tiny "toy" example in which twelve random numbers, all between 0 and 1, are arranged around a circle:



The smallest of these values is 0.07 (at approximately the three-o'-clock position); so it, and its two immediate neighbors, are replaced with three new randomly chosen numbers. After this change, the circle of numbers now appears as follows:



At the next iteration of the algorithm, we again find the smallest number (now it's 0.10, at approximately the seven-o'-clock position), and we replace it and its two immediate neighbors; and so forth.

The reader might now pause to reflect: suppose we iterate this procedure many times. What will eventually happen to the distribution of numbers in the circle? On the one hand, it seems clear that we are "weeding out," at each step, the lowest value in the set; thus, if we think of the numbers as something like "fitness values" (Bak's analogy), we can see that there is an element of natural selection about this process. At every step, the lowest element ("weakest link," if you want to think of it that way) is removed. But of course, it is replaced at random: so it just *may* be replaced by an even lower value. And its neighbors, who may have high values, are likewise vulnerable to replacement (consider the two-o'-clock entry of 0.72 in the initial circle above). So we wouldn't expect the values in the circle to steadily increase forever (i.e., to converge toward a value of 1).

My own name for this process is the *Circus Maximus*, named after the ancient Roman stadium in which gladiators fought; the name is intended to suggest the circular arrangement and (the *Maximus* part) a certain hostility toward minima. Like its Roman counterpart, our circle of numbers is a dangerous place to be for a weak link; but (again like the ancient site) it's not a particularly safe place for the strong, either, and no one can be expected to last forever. After all, consider the value of 0.90 at six-o'-clock in the second figure: it's almost the highest number present, but that won't save it from being replaced at the next iteration because of its unlucky proximity to the lowest value.

As it happens, you can think of the iterated Circus Maximus process as taking place in "waves," or "cascades." Pick the lowest number value v, replace it and its neighbors; quite possibly, one of the new replacement values will now itself be the lowest number in the circle, and thus replaced (along with its neighbors) at the next iteration. Continue with this process until the lowest value in the circle is once again v or higher; this constitutes a single "cascade." (An alternative definition—not quite the same thing—would be to continue with this process until the new lowest value to be replaced is not among the set of values that has been replaced since we first spotted v as the lowest value. This alternative definition is actually a bit more careful than the first, though it requires a bit more effort in programming as well.) If you start out with a random set of numbers, the initial cascade will likely begin with a low value of v; the next cascade with a higher value; and so forth. As it turns out, the value of v for successive cascades works its way upward toward a critical value of approximately 0.667; once all the values in the circle are at or above 0.667, the system has achieved the state of self-organized criticality.

There are many, many projects and questions that could now be posed from this starting point: I hope the reader will be intrigued enough to experiment. Consider some of the following possible avenues for exploration:

a. Choose some likely parameters for a careful Circus Maximus experiment (e.g., a circle of 300 numbers, with random numbers chosen at 0.001 intervals between 0 and 1). Initialize the circle with random values. Now graph the value of v as it inches upward in successive cascades. (You should see v moving toward, but not beyond, the critical value.)
b. Initialize the set of numbers so that they are all well above the experimentally determined critical value. Now run the Circus Maximus process again, and watch the pattern of cascades; in this case, rather than using the earlier definition for delimiting a cascade, it may make most sense to declare a cascade ended once all values in the circle are (once again) above the critical value. Call the *size* of the cascade the number

of sites affected during the course of the cascade: size will range from a minimum of 3—the original replaced value and its neighbors—to, conceivably, the entire circle. (Note, by the way, that if a cascade really does impact the entire circle, it has reached a maximal size.) Repeat this process many times and make a histogram of cascade sizes: that is, we want to know how many cascades of size 3 there are, how many of size 4, size 5, and so forth. What we would like to know is the relationship between the number of cascades of a given size S and the value of S: intuitively, large cascades are rarer than small ones, but just how much rarer? (As it turns out a graph of the logarithm of the number of cascades at size S versus the logarithm of S will be most helpful.) Pursuing this topic, it might also be worth exploring this same process but using the second definition of "cascade" mentioned earlier: i.e., a cascade ends when the lowest value in the circle is one that hasn't been created at any time during the current cascade.

c. In the Circus Maximus, weak numbers get killed off quickly, but strong numbers eventually get killed off as well. One question that Bak does not discuss in his book is whether there is any reward for an exceptionally high number: will a value of 0.99, once placed in the circle, have any greater longevity on average than a value of (say) 0.7? Here, both values are "high"—both are above the critical value—but is there any benefit to being *extremely* high? This leads to the notion of "value longevity" in general: let's define a number's longevity in the Circus as the number of replacement cycles that it survives. What is the relationship between a number's value and its longevity in the Circus (this question might have different answers for a Circus in its initial phase, as it evolves toward self-organized criticality, as opposed to a Circus that is in the critical state where all values are above 0.667)?
d. All of the questions posed above might be described as a matter of mathematical experimentation. But perhaps the most fertile ground for experimenting with the Circus Maximus is to think of it as a project for mathematical visualization and creative representation design. It isn't necessarily appealing to visualize a circle of rational numbers, after all. Suppose, for the sake of argument, we imagine a circle of 360 numbers; each number will be placed on the circle at its own angle value *theta*, ranging from 0 to 359. Now, we'll represent each random number as a line segment extending from the point (cos *theta*, sin *theta*, 0) to (cos *theta*, sin *theta*, *x*) where *x* is the value of the random number. The net effect of this would be to represent our ring as a set of line segments arranged in a circle (or, if you like, as something close to a piece of cylinder with a very irregular upper edge). With the ring of numbers represented this way, we could watch the algorithm proceed by seeing the heights of line segments shift over time; and we could watch "low values" move during particular cascades from segments to neighboring segments. This might be a good deal more informative than trying to look at lists of numbers. Alternatively we could use color to represent the numeric value in a particular location within the Circus: for example, we could represent our ring of numbers as a multicolored circular line in which color varies from (say) blue (near 0) to red (near 1). In this case, shifting color patterns would represent the evolution of the numbers in the ring over time.

Readers are encouraged to experiment with the Circus Maximus and send in any ideas for visualization (or animation) techniques, experimental results, baffling questions, variations, and so forth. As explained in the first column: correspondence should be sent to the email address given above. Please put "IJCML Puzzle Column" in the subject line, followed by a word or two indicating what type of correspondence this is (e.g., “IJCML Puzzle Column Experimental Result,” “IJCML Puzzle Column Visualization Idea,” and so forth).

References

Bak, P. (1996). *How nature works*. New York: Springer-Verlag.