Computing Bouts of the Prisoner's Dilemma

Cooperating with others and exploiting them for personal gain are the two main ways members of a society can interact. To gain further insight into the social dynamics of competing individuals, researchers have formalized the choices within a mathematical framework known as game theory. They have devised various strategies about when to cooperate and have pitted those schemes against one another to determine the most successful. Such analyses indicate that cooperation often emerges naturally in simple societies; moreover, members can often fend off cutthroat exploiters from the outside [see "The Arithmetics of Mutual Help," by Martin A. Nowak, Robert M. May and Karl Sigmund, page 76].

Many of these notions derive from the classic game called the Prisoner's Dilemma. A player can cooperate with opponents or try to cheat them (called defecting). The opponent, of course, faces the same choice. With this game and some programming, a reader can explore the concept of mutual help, without going to prison.

I describe a spatial setup, where players inhabit the squares of an oversize chessboard and spend their time repeatedly playing rounds of the Prisoner's Dilemma against their neighbors. To simplify the programming, I ignored corners and edges of the chessboard and instead considered the squares to wrap back around on themselves. All the games in each round are played at the same time.

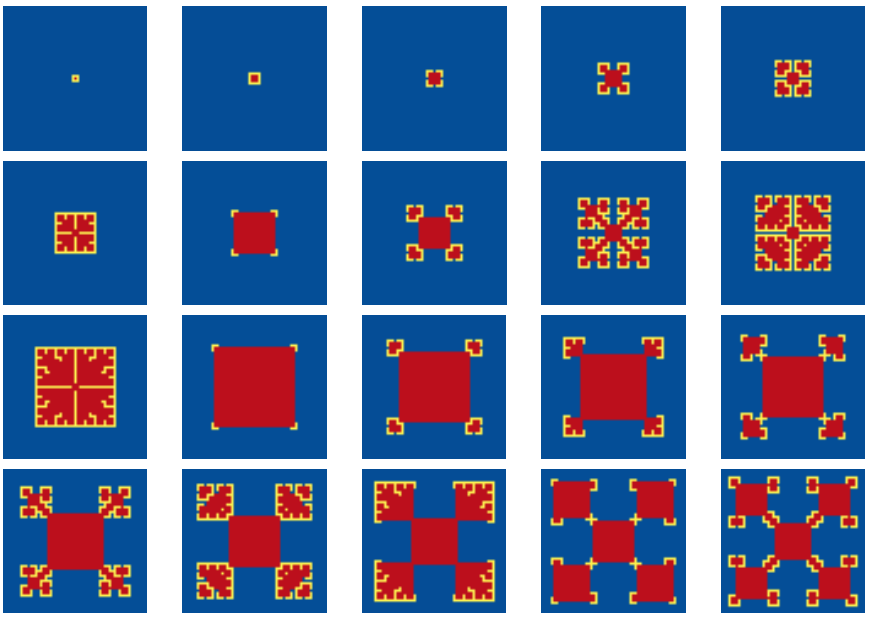
Within each round, every player takes on, one at a time, its eight nearest neighbors and itself (this self-interaction is included to make the computer program simpler). The players earn points depending on the strategy they and their opponents play. Each one gets a point if both cooperate; none if both defect. The player receives nothing if it cooperates and the opponent defects. The highest score, which I have labeled *b,* is for cheating (a player defects as the opponent cooperates). The value of *b* will ultimately control the outcome of the game. Just pick a value greater than 1; I used 1.85. A table known as a payoff matrix summarizes the scoring; it lists the rewards for the four different possible combinations of strategies [*see illustration on page 112*].

The nine payoffs resulting from the play are added up to give each player's score in that round. Each player then looks to see if any of its eight neighbors earned a score higher than it did. If so, the player will adopt the more successful strategy for the next round. For instance, if the opponent with the highest score in the player's neighborhood cooperated, the player will cooperate in the next round.

If you play the game once, against just one other player, your best choice is to defect. Betrayal maximizes your score regardless of what your opponent does. In the spatial game, however, the outcome is harder to predict because the strategy each player adopts in the next round depends on the scores of its eight neighbors as well as its own. In turn, each neighbor's score depends on its nearest neighbors, which means that the configuration of the nearest 24 players affects the outcome at each square of the chessboard.

I wrote the program in a dialect of the BASIC language called QBASIC, which comes with recent versions of MS-DOS for PC-compatible computers. It will work with Microsoft's QuickBASIC on the PC or the Macintosh and can easily be converted to run with other computer languages. The code is listed on page 114, and the explanation of its function appears in the box on page 112.

Once the program is working, you can adjust several parameters to influence the outcomes. As I mentioned, the game turns largely on the value of *b,* the advantage for hoodwinking. As *b* increases, defectors do better when they come across cooperators. Surrounding defectors always exploit lone cooperators, which always do worse. But groups of cooperators can help one another to flourish. One example is a square of four cooperators in a sea of defectors. Each cooperator will score four points (one from playing with each of its three



ALUN L. LLOYD

DEFECTORS* (red *and* yellow) *gradually invade a world of cooperators* (blue). *The game began with one defector at the center, with the score for cheating,* b, *set to 1.85.

ALUN L. LLOYD is a graduate student in the department of zoology at the University of Oxford. His interests include mathematical biology and nonlinear dynamics (that is, chaos). His research is supported by the Wellcome Trust.

neighbors and one from playing against itself), but the defectors neighboring the cooperators will score at most $2b$ because they are next to, at most, two cooperators. If b is not too large, cooperators will score higher than defectors. Lone defectors will do well because they are surrounded by cooperators.

The outcome of the competition between such different circumstances is that clusters of defectors and cooperators are seen to grow and shrink, to collide and tear one another apart. Often a dynamic equilibrium results. From a distance, an overall pattern is discernible, but individual squares are constantly changing.

Because each square's payoff is some multiple of b plus some multiple of 1, there is only a certain set of b values for which the behavior changes. The first experiment you could carry out is to run the program for different b values between 1 and 3. Try, for instance, 1.15, 1.35, 1.55, 1.77, 1.9 and 2.01. For the smaller b values, the defectors tend to be isolated, but for larger ones they form connected structures.

Once you have seen some of the different behaviors, you may want to describe them in more quantitative terms. Count the numbers of cooperators and defectors (identified by a total of four colors, depending on the strategies played over two rounds) in each round. How do these quantities vary as the contest continues? For some b values, these frequencies get progressively closer to some fixed value; for others, they vary periodically (for instance, taking one of two values, depending on whether the round is even or odd) or even in an un-

BASIC Ideas of Cooperating and Defecting

Getting the most out of the Prisoner's Dilemma games means understanding the construction of the program. The program keeps track of the players and their scores by organizing them into arrays.

Each square on the chessboard is labeled by its row and column number, written as a pair (i,j). I chose the number of squares, N, on the chessboard to be 60. The strategies are labeled by numbers: 1 for cooperation and 2 for defection. The schemes adopted by each player can then be recorded as an array of numbers, $s(i,j)$. The payoffs are recorded in another array, $pm(x,y)$—which stands for payoff matrix—where x is the strategy a player adopts and y the strategy the opponent plays. Hence, a player adopting a strategy of defecting (2) against an opponent cooperating (1) will receive $pm(2,1)$, which equals b, the score for cheating.

Before play begins, the program decides which strategy each player will use in the first round. For every square, the computer picks a number at random between 0 and 1. If it is less than a certain value—say, 0.1—then the program places a defector on that square; otherwise, a cooperator goes there. The cutoff number roughly indicates the proportion of players who will defect in the first round.

The program goes through every square (i,j) on the board, calculating each player's payoff and recording it as $payoff(i,j)$. The payoff is worked out by adding the scores from the games with the nine players, whose positions relative to the player on (i,j) are given by $(i+k, j+l)$, where k and l take the values -1, 0 or 1.

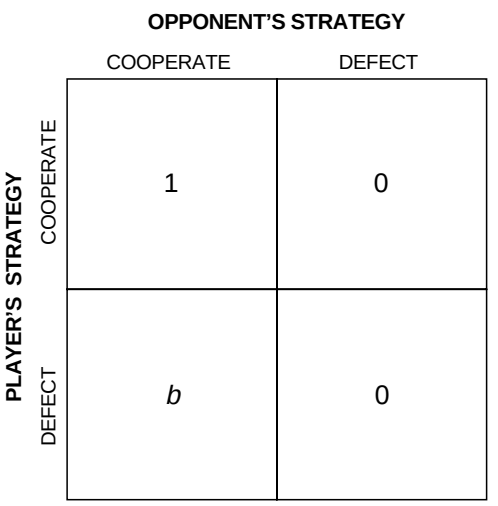
Notice in the program there is a slight complication because the chessboard wraps around on itself. If j equals 1, we are looking at the first column; its neighbors to the "left" actually lie in the last column. Similar troubles plague the last column and the first and last rows. To solve the problem, I included an array, $bc(m)$—for boundary conditions—which redirects the computer in these cases.

Now that we know the position of a player (i,j) and each neighbor $(bc(i+k),bc(j+l))$, the array $s(i,j)$ indicates the strategies each will play. The payoff from this single game can be calculated by looking in the array containing the payoff matrix: $pm(s(i,j), s(bc(i+k),bc(j+l)))$. Nine such payoffs are summed before moving to the next square on the board.

Once the payoff for every square has been calculated, the program finds the most successful strategy in each neighborhood. Then it updates the array of strategies accordingly, storing these new strategies in the array $sn(i,j)$. The program goes through every square, recording its payoff in the variable hp, for highest payoff, and its strategy in $sn(i,j)$. The program then looks at the neighboring squares in turn. If one of the neighbor's payoff is greater than hp, hp is set equal to that payoff, and the neighbor's strategy is recorded in $sn(i,j)$. Once all the neighbors have been examined, the variable hp contains the highest value of the payoff in the neighborhood, and $sn(i,j)$ has the strategy used by that player.

After all the new strategies, $sn(i,j)$, have been decided, they are copied into the array $s(i,j)$, and the next round begins. The progress of the game is followed by coloring a square grid of points on the screen. The coloring depends on the strategy each player adopts in the current and previous rounds. So there are four colors: blue (is cooperating, did cooperate), red (is defecting, did defect), green (is cooperating, did defect) and yellow (is defecting, did cooperate). Using this scheme, we can see not only which players are cooperators (blue and green) or defectors (red and yellow) but also which players' strategies are changing (green and yellow) and which are not changing (red and blue). In the program, the array $c(x,y)$ tells the computer which colors represent the previous and current strategies (x and y, respectively).

If the program runs too slowly, reduce the size of the board. Doubling the number of squares means that the program will take approximately four times as long to play each round. If you do wish to increase the board size, you may have to increase the sizes of the arrays s, sn and bc accordingly.



The payoff matrix

SCIENTIFIC AMERICAN

CORRESPONDENCE

Reprints are available; to order, write Reprint Department, Scientific American, 415 Madison Avenue, New York, NY 10017-1111, or fax inquiries to (212) 355-0408.

Back issues: $8.95 each ($9.95 outside U.S.) prepaid. Most numbers available. Credit card (Mastercard/Visa) orders for two or more issues accepted. To order, fax (212) 355-0408.

Index of articles since 1948 available in electronic format. Write SciDex®, Scientific American, 415 Madison Avenue, New York, NY 10017-1111, fax (212) 980-8175 or call (800) 777-0444.

***Scientific American*-branded products** available. For free catalogue, write Scientific American Selections, P.O. Box 11314, Des Moines, IA 50340-1314, or call (800) 777-0444.

Editorial correspondence should be addressed to The Editors, Scientific American, 415 Madison Avenue, New York, NY 10017-1111. Manuscripts are submitted at the authors' risk and will not be returned unless accompanied by postage.

Advertising correspondence should be addressed to Advertising Manager, Scientific American, 415 Madison Avenue, New York, NY 10017-1111, or fax (212) 754-1138.

Subscription correspondence should be addressed to Subscription Manager, Scientific American, P.O. Box 3187, Harlan, IA 51537. The date of the last issue of your subscription appears on each month's mailing label. For change of address notify us at least four weeks in advance. Please send your old address (mailing label, if possible) and your new address. E-mail: SCAinquiry@aol.com.

The Program for the Prisoner's Dilemma

```
DEFINT C, I-N, S                        define variables, array sizes
DEFSNG B, H, P
DIM s(120, 120), sn(120, 120)
DIM bc(121), c(2, 2)
DIM payoff(120, 120)

LET b = 1.85                            advantage for cheating
LET N = 60                              size of board
LET p = 0.1                             proportion of defectors
LET pm(1, 1) = 1                        set up payoff matrix
LET pm(1, 2) = 0
LET pm(2, 1) = b
LET pm(2, 2) = 0
                                        set up colors
LET c(1, 1) = 1                         = 409 on Mac
LET c(2, 2) = 4                         = 205 on Mac
LET c(1, 2) = 2                         = 341 on Mac
LET c(2, 1) = 14                        = 69 on Mac

RANDOMIZE TIMER                         initialize board
FOR i = 1 TO N
  FOR j = 1 TO N
    LET s(i, j) = 1
    IF (RND < p) THEN LET s(i, j) = 2
NEXT j, i

FOR i = 1 TO N                          set up boundary conditions
  LET bc(i) = i                         no problem if i between 1 and N
NEXT i
LET bc(0) = N                           redirect neighbors of edges
LET bc(N + 1) = 1

SCREEN 12                               not needed on Mac

FOR M = 1 TO 1000                       begin playing game
  FOR i = 1 TO N                        calculate payoffs for each player
    FOR j = 1 TO N
      LET pa = 0
      FOR k = -1 TO 1
        FOR l = -1 TO 1
          LET pa = pa + pm(s(i, j), s(bc(i + k), bc(j + l)))
      NEXT l, k
      LET payoff(i, j) = pa
  NEXT j, i

  FOR i = 1 TO N                        find largest payoff in each
    FOR j = 1 TO N                      neighborhood and calculate
      LET hp = payoff(i, j)             new strategies
      LET sn(i, j) = s(i, j)
        FOR k = -1 TO 1
          FOR l = -1 TO 1
            IF payoff(bc(i + k), bc(j + l)) > hp THEN
              LET hp = payoff(bc(i + k), bc(j + l))
              LET sn(i, j) = s(bc(i + k), bc(j + l))
            END IF
        NEXT l, k
  NEXT j, i

  FOR i = 1 TO N                        display strategies
    FOR j = 1 TO N
      COLOR (c(sn(i, j), s(i, j)))      ForeColor on Mac
      PSET (i, j)
      LET s(i, j) = sn(i, j)
    NEXT j, i
NEXT M

END
```

predictable fashion. Calculate the fraction of players that change their strategies in each round. If it equals 0, you have reached a static equilibrium. Graphically, the situation shows up as all squares being two colors. (Such a pattern occurs when b is set to 2.01.)

Is the initial proportion of defectors to cooperators important for these results? Try running the program several times with different ratios. The clustering of defectors and cooperators is also crucial. If the number of cooperators is small, on some occasions they may be clustered and can flourish. On others, they may be isolated and are doomed.

This program can generate some pretty patterns if the initial configuration is symmetrical. It is easy to modify the code so that every square at first is a cooperator. Remove the line in the program that decides whether a square should start off as a defector. Now set just one square, near the center, to be a defector. (Insert the line $s(30,30) = 2$, for instance, just after the loop that now sets all squares to be cooperators.) Choose a b value between 1.8 and 2 and start the program. You should see that the single defector can invade the world of cooperators [*see illustration on page 110*].

Another easy modification is to play the game only against the eight neighbors. To do this, do not add on the payoff when the square plays against itself (when $k = 0$ and $l = 0$ in the program). Similar behaviors will be seen but with slightly different b values.

Or try different boundary conditions—for instance, unwrap the chessboard and let the edge players compete against their five neighbors and the corner players their three. This scenario can be messier to program, as edge and corner squares must be treated differently. Another alternative would be to insist that all corner and edge squares always cooperate or always defect. An interesting change would be to play the game against only the four nearest neighbors (up, down, left and right). For those of you who like a challenge, try setting up the game on a honeycomb-pattern board, so that each square has six neighbors.

Further explorations could include altering more than one entry in the payoff matrix. Small changes, such as setting the entry at the bottom right in the payoff matrix (both player and opponent defecting) to a tiny positive value such as 0.01, do not alter overall behavior very much. More radical changes will have a noticeable effect. The number of possible manipulations of the program is almost limitless, as will be the patterns produced.