'heuristic' programs.

An heuristic program enables the computer to detect alterations in its opponent's strategy, and modify its algorithm accordingly. Such a program would have to keep a record of, let's say, the last 50 choices of both opponents, in an array. It constantly scans through this track record applying a statistical technique known as 'correlation'.

This involves the computer in making hundreds of comparisons between the player's choice and his previous choice, or the one before that, or the choice made five turns ago. The computer performs the same operation on its own choices. Let's consider the correlation between the player's move and his previous move, for example. We'll call Scissors — element 1, Paper — element 2, and Stone — element 3. First we must set up a three by three array, called say CORR1, because it represents our first correlation test. Now we must work through our game history, looking at the player's choices for the last 50 moves. Every time he followed Scissors (1) by Stone (3), we add one to the element CORR1(1,3); when Stone (3) is followed by Paper (2), one is added to element CORR1(3,2) and so on.

If the player is making truly random choices, then there should be approximately equal values in each element of CORR1 — but this is very unlikely to be the case. So, if the player chose Paper last, then the element in row 2 (Paper) of CORR1 with the largest value will give us the best guess as to what he will choose next. The greater the difference between the elements in any row, the better the correlation is, and the more reliable the prediction will be. However, it is possible that there will be little correlation between the player's choice and his previous choice, in which case we must also perform correlation calculations on the second to last choice, or between the player's choice and the computer's previous choice.

A problem arises if the various correlation routines all predict different results for the player's next move. The program has to decide which is the most reliable advice. In this simple game, all it needs to do is see which test has the most pronounced correlation. For example, the CORR1 array might predict the following probabilities: Scissors 51%, Paper 29%, Stone 20%; whereas CORR2 (which, say, compares the player's choice with the computer's last choice) might give: Scissors 24%, Paper 60%, Stone 16%. Clearly CORR2 has the better correlation, so its prediction should be selected. An intelligent games program will in fact frequently consist of a number of subroutines, each working on different strategies, and each advising the main routine of the best move. The playing routine can regard these subroutines as a 'committee', and act on a majority decision. But as the game proceeds, it can award marks to each routine according to whether its advice was good or not.

If there does turn out to be some correlation between the player's moves or choices and the previous moves of the computer, then it is possible to program in some kind of 'bluffing' factor that will deliberately mislead the player. This works best in gambling games, where the stakes increase as the game continues, and it is worthwhile losing the early rounds to win the later ones.

At the State University of New York at Buffalo (reported in *Scientific American*, July 1978) a collection of poker-playing programs (all of them with a learning capability) were set against each other for several thousand games. The overall winner was a program called the Adaptive Evaluator of Opponents (AEO), which made an initial judgement about the strength of its opponents' hands, and modified this estimate as each game proceeded. The SBI program, 'Sells and Buys Images', did surprisingly badly — its technique was to bluff in order to 'sell' a false image to its opponents, or effectively to 'buy' the playing style of others. The Bayesian Player (BP) tried to make inductive inferences, and improve its play by comparing the predicted consequences of its actions with the actual consequences. Finally, the Adaptive Aspiration Level (AAL) program attempted to mimic a feature believed to exist in human playing: adapting the level of aspiration (that is, the degree of risk it is prepared to take) according to its past record and current status.

No two chess programs or other artificially intelligent routines work in exactly the same way. But by experimenting with the techniques we've outlined here on progressively more complicated games, you may eventually be able to join the exclusive club of chess program writers.

Slow Learner

This program, based on the game Scissors — Paper — Stone, illustrates how a program can 'learn' as a game progresses. The computer selects from the numbers 1,2 and 3, compares its choice with the one you have typed in and adjusts the score. The GET statement has been used so that you can simply press the three number keys in rapid succession. If you attempt to make your sequence random, you should find that after a couple of hundred key-presses, the computer's score will pull ahead. It is possible to fool this program and hence continue to win, but more sophisticated routines can be added to it to prevent you from doing this

```
  5  CLS
 10  DIM C1(3,3),C2(3,3),C3(3,3)
 20  CR=0
 30  FOR I=1 TO 3
 40  IF C1(PL,I) >CR THEN BG=I:CR=C1(PL,I)
 50  IF C2(PP,I) >CR THEN BG=I:CR=C2(PP,I)
 60  IF C3(P3,I) >CR THEN BG=I:CR=C3(P3,I)
 70  NEXT I
 80  CT=BG-1
 90  IF BG=1 THEN CT=3
100  GET PT: IF PT=0 THEN 100
110  REM LINE 100 WAITS FOR A DIGIT TO
120  REM BE PRESSED.
130  IF CT=PT-1 THEN CS=CS+1
140  IF CT=PT-2 THEN PS=PS+1
150  IF CT=PT+1 THEN PS=PS+1
160  IF CT=PT+2 THEN CS=CS+1
170  CLS
180  PRINT "YOUR CHOICE: ";PT
190  PRINT "MY CHOICE: ";CT
200  PRINT "YOUR SCORE IS ";PS
210  PRINT "MY SCORE IS ";CS
220  C1(PL,PT)=C1(PL,PT)+1
230  C2(PP,PT)=C2(PP,PT)+1
240  C3(P3,PT)=C3(P3,PT)+1
250  P3=PP
260  PP=PL
270  PL=PT
280  GOTO 20
```