HOW TO MAKE A COMPUTER APPEAR INTELLIGENT

"Five-in-a-Row" offers no guarantees

by JOSEPH WEIZENBAUM, Computer Laboratory, General Electric Co., Mountain View, Calif.

There exists a continuum of opinions on what constitutes intelligence, hence on what constitutes artificial intelligence. Perhaps most workers in the fields of heuristic programming, artificial intelligence, et al, now agree that the pursuit of a definition in this area is, at least for the time being, a sterile activity. No operationally significant contributions can be expected from the abstract contemplation of this particular semantic navel. Minsky has suggested in a number of talks that an activity which produces results in a way which does not appear understandable to a particular observer will appear *to that observer* to be somehow intelligent, or at least intelligently motivated. When that observer finally begins to understand what has been going on, he often has a feeling of having been fooled a little. He then pronounces the heretofore "intelligent" behavior he has been observing as being "merely mechanical" or "algorithmic."

The author of an "artificially intelligent" program is, by the above reasoning, clearly setting out to fool some observers for some time. His success can be measured by the percentage of the exposed observers who have been fooled multiplied by the length of time they have failed to catch on. Programs which become so complex (either by themselves, e.g. learning programs, or by virtue of the author's poor documentation and debugging habits) that the author himself loses track, obviously have the highest IQ's.

The program described in this article plays the game Five-in-a-Row, also known as Go-MOKU. This game is, to the author's knowledge, undetermined in the sense that no strategy other than exhaustive look ahead is known for guaranteeing a win or even a tie. In this respect it differs from Nim and Tic-Tac-Toe for which guaranteed winning strategies are known and have been programmed. On the basis of the criterion given above, the program certainly appears (to many observers) to make the computer behave intelligently. The fact that it permits its opponent to develop a position while paying close attention to its own development for a time and only occasionally becomes obviously defensive in its play helps to create and maintain a wonderful illusion of spontaneity. In fact, however, the program represents a simple algorithm as is shown below.

It is perhaps ironic that the author has never succeeded in beating the program and also that the program is being used as an automatic opponent (i.e. teacher) to certain learning programs trying to master its game.

The game can be looked upon as a generalization of Tic-Tac-Toe which, from this perspective, should be called Three-in-a-Row. An additional generalization is that whereas Tic-Tac-Toe is played on a 3 x 3 board, Five-in-a-Row is traditionally played on a Go board of size 19 x 19. The game is won by that player who first succeeds in placing five of his symbols adjacently and in a straight row (in any direction) on this board.

The fundamental strategy is that after a certain value is assigned to every potential move both from the point of

the player (P) and that of the machine (M), the machine chooses its next move by a technique based on these values which has for its aim the invalidation to the greatest extent possible of the planning of the player while optimizing its own advantage. The trick is in coming up with good evaluation functions and optimizing techniques appropriate to these functions. Once this has been done, refinements suggest themselves rather easily. Obviously, a prior problem is that of determining a terminology and a notation within which evaluation functions and strategies can be unambiguously described.

definitions

A *chain* is defined to be any set of five consecutive squares in either the horizontal, vertical, or diagonal direction. Every point is said to *anchor* four chains. For the point (i, j)* these are:

$$(1)\quad \begin{aligned} W(i,j) &\equiv (i-s,\ j) \\ NW(i,j) &\equiv (i-s,\ j+s) \quad (s=0,1,\ldots,4) \\ N(i,j) &\equiv (i,\ j+s) \\ NE(i,j) &\equiv (i+s,\ j+s) \end{aligned}$$

Thus every possible chain has a unique name. Every chain has two *values*, a P value (PV) and a M value (MV). The chain value is simply the number of P or M marks respectively which are to be found in the chain. A chain which has a P value is said to be *uninteresting* from the M point of view and vice versa. Clearly some chains are uninteresting from both the P and M point of view. Chains which are not uninteresting are *interesting*.

Every point (i, j) is said to *belong* to twenty chains. These are the chains whose values are affected if a mark were to be made at that particular point. The names of these chains for the point (i,j) are:

$$(2)\quad \begin{aligned} &W(i+s,\ j) \\ &NW(i+s,\ j-s) \quad (s=0,1,\ldots,4) \\ &N(i,\ j-s) \\ &NE(i-s,\ j-s) \end{aligned}$$

In addition, every point (i, j) is said to be *associated with* 32 points, namely those points different from the point (i, j) which form the union of all points which make up the twenty chains to which the reference point belongs. For the point (i, j) this set consists of the points

$$(3)\quad \begin{aligned} &(i \pm t,\ j) \\ &(i,\ j \pm t) \quad (t=1,2,3,4) \\ &(i \pm t,\ j \pm t) \\ &(i \pm t,\ j \pm t) \end{aligned}$$

position values

It is now possible to define position values for both P and M. These will be called P position value (PPV) and M position values (MPV) respectively. Judgment enters into the determination of algorithms suitable for their computation. Before describing a specific algorithm, however, the general manner in which position values enter into the strategy is discussed.

The P position value of an unoccupied position (i, j), i.e. PPV(i, j) is supposed to be a numerical measure of the utility of that particular position to the developing plan of the player of P. It must therefore reflect the extent to which P-interesting chains would be enriched were that move to be made by P. The way this is done is to assign a specific weight w(PV) to the four non-trivial possible chain values, and then to compute PPV(i, j) by simply summing the weights of all P-interesting chains to which the position (i, j) belongs. The computation of PPV(i, j) is therefore given by

$$(4)\quad PPV = \sum_{s=1}^{r} w(PV)_s$$

where s ranges over the r P-interesting chains involved. Similarly for MPV(i, j).

The function (4) is still quite general in that the weight assignment procedure has not been specified. Indeed, experiments show that quality of machine play is strongly dependent on weight assignment rules. Such rules are easy to state in tabular form. The table given below is that used in the game exhibited in this paper.

Chain Weight Table

Chain Value	Weight
1	5
2	25
3	80
4	1000

the basic strategy

Assuming a specific chain weight table, machine strategy is now quite straightforward. When P makes his move all relevant chains are updated. New PPV's and MPV's for the 32 points associated with the chosen point are computed and stored. Every empty point associated with any point which has a mark in it is considered a candidate for the next machine move. For each candidate point the number

$$(6)\quad N = K_1 PPV + RK_2 MPV$$

is computed. K_1 and K_2 (small integers) are parameters within the strategy. R is computed as described below. Thus the strategy has essentially five free parameters, K_1/K_2, and four weights which determine the chain weight table. It may be seen that the ratio K_1/K_2 constitutes an aggression coefficient. That is, its value determines how offensive or defensive a game the machine will play. Observation reveals, however, that human players vary the aggressiveness with which they play from move to move. They make this determination on the basis of an evaluation of the danger they feel themselves to be in versus the strength of their own position. This evaluation is approximated in this strategy by having the machine (when it is its turn to move) compute the ratio

$$(7)\quad R = \max(MPV) \,/\, \max(PPV)$$

and then multiplying K_2 by R for each of its moves. This means that if the machine's strongest move is much stronger than that of the player's next possible one, the machine will play proportionately more aggressively, and vice versa.

The point for which N(i, j) is a maximum (if there is one) is the machine move. If several N's have the maximum value, then the final choice of the move is made from among the maximum candidates on the basis of a random number generator. After proper updating of the appropriate MVs, PPV's and MPV's the player may make his next move.

Experience has led to the conclusion that an initial aggressiveness coefficient of approximately 1 is proper. Obviously, if the machine game is too defensive, then the machine follows the player's moves like a puppy dog, permitting the observant player to lead the machine play into traps. Too offensive a coefficient, on the other hand, permits the player to build combinations which remain unchallenged until it is too late.

The weight table exhibited above can be justified on the basis of the judgments that

a) An interesting chain of value two should have a greater weight than two crossing chains of value one, and

b) An interesting chain of value three should have a

*For definiteness, it can be assumed that the first mark made determines the origin of the coordinate system.

greater weight than two crossing chains of value two each.

These lead to the inequalities

$$8\,w(1) < 2\,[3\,w(2) + w(1)]$$

(8) and

$$2\,w(3) > 5\,w(2) + w(1)$$

which are satisfied by the table.

The weight w(4) is really irrelevant in the sense that a simple test can be made to determine whether the machine or the player has a chain of value four. If the machine has such a chain, then it completes it and wins the game. Any player chain of value four must be stopped under all other conditions. The weight table could take care of these matters. But when the machine is playing defensively and it and the player have an interesting four chain each, the machine will block rather than win. A "four test" fixes this difficulty simply.

The program displays an initial aggressiveness which is a manifestation of an overly naive goal directed behavior. The job is to make five-in-a-row and the program proceeds right to the task. In the beginning game, the player has, of course, not yet developed any position against which a defense appears necessary. This form of initial behavior on the part of the program gives the player an important advantage. A way to avoid this is to give the machine a stack of starting patterns to develop. Two or three moves would probably be sufficient to overcome the difficulty under discussion. Similarly, a number of "pattern detectors" could easily be built in. These would catch certain patterns which are known to result in winning combinations if completed.

A program implementing the strategy here outlined has been written by R. C. Shepardson of the General Electric Computer Laboratory (Mountain View, California). This program has been matched against players of various strengths. In general it has beaten novices even when it plays second.

An interesting sidelight on the apparent intelligence of the strategy described, is that novices seem to play according to the same algorithm as that on which the program is based. They continue to do so in many cases for some time. After many games, the novice suddenly makes moves in an entirely different way. He then begins to beat the system rather regularly. Evidence for these assertions is gained from analysis of the print-out of games played. Among other data, the move the machine would have made had it been in the player's position is printed out together with its position value. In many instances, the novice player chooses exactly that move as his own. To some extent the machine win can therefore be attributed to the machine's infallible bookkeeping. The player is finally confused by the mass of data generated by a long game.

The above comment lends to a possible modification of the strategy which should improve machine play. The program can be made to notice whenever the player departs from orthodox (i.e. programmed) strategy. This condition can be detected on the basis of the fact that unorthodox moves have low position values. Whenever this takes place, the machine should switch its play to the purely defensive and furthermore consider only points associated with the player's unorthodox move as candidates for its own response. ■

REFERENCES:

1. C. E. Shannon, "Programming a Computer for Playing Chess," *Phil. Mag.*, Vol. 41, No. 314, March, 1950.
2. C. E. Shannon, "Game Playing Machines," *Journal of Franklin Inst.* Vol. 260, pp. 447-453, December, 1955.

WARSAW CONFERENCE ADOPTS ALGOL-60

Automatic Programming Committee established

From The Institute of Applied Mathematics, Polish Academy of Science, Warsaw

A working meeting on Automatic Programming Methods was held in Warsaw last Fall. About 60 participants took part in the Conference representing the Soviet Union, Czechoslovakia, East Germany, Hungary and Rumania.

The aim of the Conference was to consider theoretical and practical problems in the field of automatic programming, to get acquainted with progress in countries represented at the Conference, and to trace a common future action for this group.

Fourteen reports were presented and two panel discussions were held; one of which was concerned with problems connected with the international ALGOL language, and the other with address-free computers.

A demonstration of the SAKO automatic programming system which is similar to FORTRAN II, was organized at the Institute of Mathematical Machines in Warsaw. The demonstration was performed on the XYZ computer. Two mathematical problems were formulated by Conference participants during this demonstration. The first problem consisted in finding the root of a third degree polynomial. The second problem was to provide a tabulation of 100 values of a certain function in a given editorial format. Programs for solving these problems were written in SAKO, perforated on tape, translated into the resulting program and operated on by the XYZ computer. In both cases the performance time for all of these operations was under 45 minutes.

In the course of the discussion on ALGOL-60, many of its limitations were emphasized, as well as the complexity of its notations which suppress the popularization of the language among engineers, biologists, etc. In addition, numerous difficulties were indicated in producing an effective translator of the full ALGOL version. In spite of this, it was decided to accept ALGOL-60 as the basis for a standard mathematical language. Further development of this language was acknowledged as being necessary. It was pointed out that some work in this field has already been accomplished in Moscow and Nowosibirsk.

Besides the work connected with the realization of a possible full version of ALGOL, the application and propagation of simpler languages, already in use (for instance, the autocode used in Kiev, or the autocode SAKO) was also encouraged. The necessity of close cooperation of different scientific centers in countries represented at the Conference was emphasized.

For this purpose, a Committee of Automatic Programming was established, the members of which represent the various countries attending the Conference. The main task of this Committee will be to organize a mutual exchange of information and to establish contacts in the field of autocodes with other international institutions such as the International ALGOL Committee.

Finally, it was decided to arrange an annual Conference similar to the one held in Warsaw. It was suggested that the next Conference be held in the Soviet Union in autumn, 1962. ■