The chess program that we have presented in parts 2 and 3 of this series (November 1978 and December 1978 BYTE, pages 162 and 140, respectively) represents a modern implementation of the basic type A strategy described by Shannon in 1950 (see references). If run on a powerful computer, this type of program can play a reasonably good game of chess. Its major weakness lies in its inability to engage in long-range planning. In many middle and end game positions, it will make seemingly aimless moves. Once it attains a position which optimizes the

if we have a clear idea of what we are looking for. To know what we are looking for, however, we must have more knowledge about chess.

So where do we go from here? The highly skilled players who are familiar with the chess programming literature (notably, Berliner, Botvinnik and Levy) are unanimous in their enthusiasm for a selective search strategy. Berliner (see references), for example, advocates a procedure in which very small (for a computer) look-ahead trees are generated, eg: 200 to 500 nodes. His idea is that the program should make an intensive analysis at each node "in order to ascertain the truth about each issue dealt with." Chess knowledge should play a primary role in directing the tree search. The search itself would discover additional relevant information and this would provide an even more knowledgeable focus for the search. This

Creating a Chess Player

Peter W Frey
Dept of Psychology
Northwestern University
Evanston IL 60201

Larry R Atkin
Health Information Services
542 Michigan Av
Evanston IL 60202

general heuristic goals of its evaluation function, it is faced with the prospect of finding a move which alters the position as little as possible. If the opponent is skillful in developing a long-range attack while not providing any immediate targets, the machine may simply shuffle its pieces back and forth until its position becomes hopeless. The absence of reasonable goal directed behavior is a common limitation of problem solving techniques which are based solely on forward search. The solution of this problem would have important implications for a wide variety of artificial intelligence tasks.

To play a strong game of chess, it is necessary to have a plan. To have a plan, however, the program must recognize specific patterns and relate them to appropriate goals. This, in turn, requires that the program have access to the detailed kind of chess knowledge which is characteristic of the skilled human player. Thus, we seem to have come round in a circle. In order to avoid selective searching, we have adopted a strategy which does not require very much chess knowledge. In examining the weaknesses of this approach, we discover that the forward search can only be truly successful

procedure is analogous to the progressive deepening technique which de Groot discovered in the human grandmaster and is the exact antithesis of the brute force (type A) strategy (see October 1978 BYTE, "Creating a Chess Player, An Essay on Human and Computer Chess Skill," page 182).

The efforts of the last decade have demonstrated that the selective search strategy is harder to implement than the full-width approach. In addition, full-width searching has consistently produced superior chess. Despite this, there is hardly anyone familiar with chess programming who does not believe that further progress depends on increasing the amount of chess knowledge in the program. The key question is not whether this should be done but how to do it. Since the selective search approach has not led to notable progress, perhaps it is time to consider a different approach.

We believe that a viable alternative exists which combines the proven virtues of the full-width procedure with the potential advantage of a goal-directed search. The central idea is the development of a unique evaluation function for each position. In addition to the general heuristics which are presently

employed, evaluations should consider features which are germane to appropriate goals.

According to this plan, move selection would involve two separate stages. In the first phase, a static analysis of the position would be made in an attempt to discover key patterns. This process would involve a hierarchical analysis in which the features of the position would be compared with a general set of library patterns. Highly specific features would be identified and relevant chess-specific knowledge would be accessed. This information, including appropriate short term and long term goals, would be used to construct a conditional evaluation function which would assess the usual general features (eg: material, mobility, King safety, etc) and also other features which are meaningful only in specific situations. Once the conditional evaluation function has been constructed, the second phase of analysis would begin, a conventional full-width tree search employing the special evaluation function.

The first phase of this process would rely heavily on domain specific knowledge (ie: information about chess). It would require a pattern recognition facility and an organizational plan for storing a vast amount of chess knowledge in a manner conducive to rapid retrieval. When this first phase was successful in identifying appropriate goals and producing relevant modifications in the evaluation function, the full-width search which followed would select a move which was thematic with the appropriate goal. If the first phase were unable to identify a key feature, the evaluation function would employ the same general heuristics which it presently uses. For this reason, the pattern recognition and information retrieval modules can be gradually implemented without a lengthy period in which serious blunders are frequent occurrences. This is a major advantage that the conditional evaluation function has in comparison to a selective search strategy.

Chess Structure

To implement a conditional evaluation function, it is necessary to develop a hierarchical descriptive structure for chess. At the top level, one can make the conventional distinctions between the opening, the middle game, and the end game. Within each of these three major divisions, there would be many specific subdivisions. Within each subdivision, there would be many specific variations.

The opening has three major themes: to develop a pawn structure which is favorable for you but unfavorable for your opponent; to increase the mobility of your minor pieces and limit the mobility of your opponent's minor pieces; and to castle as soon as possible and delay your opponent's opportunity to castle. These general goals provide a framework for evaluating specific variations. They do not provide a specific prescription for selecting a move because a sequence of moves which is thematic with these goals may have a tactical refutation. An apparently good move may not work because it loses material. For this reason, general principles are best applied at the terminal points of a look-ahead search rather than being used as a checklist for selecting the most thematic move as advocated by Church and Church in *Chess Skill in Man and Machine* (see references).

Part 4

Strategy in Computer Chess

The tournament player who knows opening theory as well as many specific move variations will have a clear advantage over an opponent who knows the general principles but is not familiar with the specific variations. For this reason, tournament players and good chess programs rely on a library of memorized opening variations. The contestant who has carefully planned his opening variations can often gain an important advantage early in the game. To maximize the benefit of a well-prepared opening library, it is also necessary to continue the general theme of the opening once the predigested move sequences have been exhausted. At this stage it is necessary to have a conditional evaluation function. When the machine leaves the library and starts to use a look-ahead procedure to calculate its move, it should use an evaluation function that augments general opening principles with special goals which are thematic with that type of opening.

A portion of the work required to implement this proposal has already been started. Chess specialists have prepared highly detailed analyses of specific opening variations and have developed well-defined rules for categorizing different move

sequences into specific subdivisions. For example, a game which starts (1) P-K4, P-K3 is labeled as the French defense. If the game continues (2) P-Q4, P-Q4; (3) N-QB3, B-N5, it is called the Nimzovich (or Winawer) variation of the French defense. If it continues (2) P-Q4, P-Q4; (3) N-QB3, N-KB3, the game is labeled as the classical variation. A continuation of (2) P-Q4, P-Q4; (3) N-QB3, PxP is called either the Rubinstein variation or the Burn variation depending upon subsequent moves. A different approach develops from (2) P-Q4, P-Q4; (3) N-Q2, which is labeled as the Tarrasch variation. And there are many more. The important point, however, is that each of these variations can be objectively identified, and that for each there are well-developed strategical ideas and specific immediate goals. These ideas can be stored in the opening library and can be retrieved when the machine leaves the library. In addition to general opening heuristics, the evaluation function would reflect the specific theoretical ideas which are appropriate to the particular opening at hand. In principle, this idea can be implemented without difficulty. In practice, however, a tremendous amount of chess knowledge is needed and hours and hours of effort are required. To our knowledge no serious attempt has yet been made to implement this strategy. The information on opening theory is needed only once during a game and thus could be stored on disk, since rapid access is not critical.

Pattern Recognition and the Middle Game

From a conceptual point of view, the application of chess knowledge to the evaluation function in the middle game is much more challenging. In this case, pattern recognition becomes an important ingredient. In implementing a goal oriented move selection strategy, Church and Church limited their middle game strategy to either a Kingside attack, a Queenside attack, or concentration on a weak point (ie: a target). The Kingside or Queenside attack is triggered when the machine determines that it has superior forces on one side or the other. This determination can be based on who controls key squares. In calculating the power relationship of different pieces over given squares, it is important to note that less valuable pieces exert more control than valuable pieces. A pawn has greater control over territory than a Queen because it is harder to dislodge. If an attack on one side or the other is deemed appropriate, the evaluation function can be modified to give an extra bonus for moves which augment

the attack on that side and for moves which increase the pressure on critical squares.

Pattern analysis is also important in detecting an appropriate target. There are several well-known chess relations which provide obvious targets for attack. One is the *backward pawn* which is prevented from advancing by a pawn or a minor piece. Another natural target is the minor piece which is pinned to the King or Queen. The third is the overworked piece, a key element in the defense against two or more different attacks. If the latter is removed in an exchange, the pieces it is defending will be open for attack. A fourth natural target is a square which would permit a Knight to fork two major pieces (ie: Rook, Queen, King) or a Bishop to skewer two major pieces. If the machine threatens to control that square and to locate an appropriate piece there, the opponent will be forced to devise a defense. Once one of these targets has been detected, the evaluation function can be modified to give a bonus for moves directed at the target. In addition, a plan might be devised to encourage the use of a decoy (a pawn or minor piece which is sacrificed to bring an important piece to a particular square) or to capture a piece which is serving an important defensive function.

A Chess "Snapshot"

In the past, programmers have attempted to implement such plans by using a selective search (eg: Berliner, Zobrist and Carlson) or by using no search at all (eg: Church and Church). Zobrist and Carlson (see references) have developed an innovative technique in which "computer snapshots" are devised which summarize important piece relationships such as attacks, pins, skewers, forks, etc, which presently exist in the given position, or which could occur after one or two moves. Each snapshot is given a weight based on the relative values of the pieces involved and the location of the pieces in respect to the opposing King and the center of the board. The weighted snapshots are then used to select moves for inclusion in a Shannon type B tree search. This procedure provides considerable goal direction to the move selection process.

Although the Zobrist-Carlson snapshot procedure has much to offer (including a highly efficient bit map implementation strategy), it incorporates a common problem shared by all selective search techniques. Occasionally an important continuation is overlooked and this results in the selection of an inappropriate move which may be a gross blunder. By implementing the plans

derived from the computer snapshots in the form of a conditional evaluation function, instead, the program can benefit from goal directedness without risking the oversights which are characteristic of selective searching. In this way, the machine can retain the benefits of the full-width search and at the same time engage in strategic planning.

There is a special class of positions for which this approach is especially appropriate. In his thesis at Carnegie-Mellon University Berliner described a special problem, the horizon effect, which plagues the conventional look-ahead approach (see *Chess Skill in Man and Machine*, pages 73 thru 77). One version of this problem involves a piece which is trapped and cannot be saved. Forward searching programs often engage in a bit of foolishness by making forcing but poor moves (such as attacking pieces with pawns or sacrificing pawns for no advantage) which delay the capture of the trapped piece and push its eventual loss beyond the horizon of the tree search. By doing this, the program erroneously concludes that the piece is safe, when in reality the planned move sequence weakens a reasonable position and is still insufficient to save the piece. In this type of situation, the trapped piece should be given up for lost and the program should do its best to take advantage of the tempo required by the opponent to capture the piece. A piece whose time has come is sometimes referred to as a desperado. The only option available is to make the opponent pay as dearly as possible for the capture. If the desperado can be traded for a pawn or a piece of lesser value, this is preferable to being given up for nothing.

This strategy can be implemented with a conditional evaluation function by simply assuming that the trapped piece has a material value of zero. This change would cause the search process to trade the piece for the highest valued candidate that can be found. This is obviously better than having the program engage in useless sacrifices of position and material in a hopeless attempt to resurrect a lost piece. The key element to this implementation is the ability to determine when a piece is truly lost and can be labeled as a desperado. This is a very difficult problem even for a very sophisticated pattern analysis facility.

End Game Considerations

The most interesting application of the conditional evaluation function is in the end game. Because end game strategy is highly dependent on the specific characteristics of the position, a general purpose evaluation function is not very effective. It is necessary to understand what is required in a given position and then select moves which are clearly directed at an appropriate goal. Church and Church list three common goals in the end game: to mate the opponent's King, capture a weak pawn, or promote a pawn. In this case, pattern analysis is important. First the machine must be able to identify the position as one belonging to the end game. Then it has to determine whether a mate attempt is reasonable or whether a pawn can be captured or promoted. Church and Church (see *Chess Skill in Man and Machine*, pages 151 thru 154) describe a general strategy for identifying and capturing a weak pawn. Although their approach does not involve a forward tree search, the specific techniques which they describe can be adapted to the full-width search strategy. Let us consider several specific end game positions involving either a mate, a pawn capture, or a pawn promotion.

For a number of mating situations, a specific algorithm (step-by-step instructions) or a complete lookup table can be developed to produce mate in a minimum number of moves. Typical applications would be King and Queen versus King; King and Rook versus King; and King, Bishop, and Knight versus King. The mating algorithm for each case would include rules for assigning the potential piece relationships into a few general categories, and a prescription for an appropriate type of move for each category. This approach requires no search. A second approach involving a lookup table is even more explicit. An appropriate move is stored in a table for every possible piece configuration. To play the mate perfectly, the machine uses the position to determine an address in the table and then simply reads the correct move.

Both of these procedures are perfectly feasible and avoid many problems which can be encountered in the end game. The limitation of this approach is that there are a very large number of mating situations and a tremendous amount of work would be required to make a detailed analysis of each one. In addition, this strategy requires the storage of a great deal of information which would be used only infrequently.

A third approach, and one which is thematic with the idea of conditional evaluations, is to make a small modification in the evaluation function for each specific mating situation. The notion is that a shallow search combined with a few key ideas should suffice to produce a mate in a reasonable number of moves. With King and Queen or King and Rook versus King, it is sufficient for the

program to "know" that the defending King must be forced to the edge. To do this, the program simply needs to add bonus points to the evaluation function when the defending King is near the edge. The size of the bonus should be a linear function of closeness to the edge. This modification of the evaluation function causes the minimax search to select a pathway in the look-ahead tree which forces the defending King to the edge.

With King, Bishop, and Knight against King, the job is slightly more complicated. In this case it is important to know that the defending King must be forced to one of the two corners having the same color as the Bishop's squares. The trick is to add a large bonus when the defending King is on the appropriate corner squares and a smaller credit when it is near these corners. This modification will cause the minimax procedure to find a sequence of moves which forces the defending King into one of the appropriate corners. The general theme is that the full-width search is a powerful device by itself and that the addition of a small amount of chess knowledge is sufficient to produce the desired outcome.

Kings and Pawns in the End Game

Some of the most challenging positions in the end game involve only Kings and pawns. Many of these require an approach which is more sophisticated than those described previously. Consider, for example, the position diagrammed in figure 1. This is a modification of a position presented in Berliner's thesis which demonstrates one of the major weaknesses of a full-width forward search. White has a pawn on f6 which could advance and be promoted if the Black King were out of the way. *[Algebraic notation is used throughout this article to designate chessboard squares. The horizontal rows (ranks) are numbered from 1 to 8, starting at the bottom (White). The files are labeled a through h from left to right . . . . CM]* To win, White must do an end run with his King and bring it to the aid of the pawn. Since Black cannot attack White's pawns on c3 or g5 without leaving the passed pawn, he is helpless to stop White's maneuver. Although this analysis is obvious at a glance to an experienced player, a program that discovers truth by doing a full-width search is faced with a difficult problem. In order to determine that the King can force promotion of the pawn, White must complete a look-ahead search of approximately 35 plies. This is beyond the scope of even the most powerful computer. If the machine employs a general purpose algorithm which encourages the King to centralize its position during the end

game, it will search for a pathway which eventually places it on its present square (f4) or one of the neighboring squares (e3 or f3). Because of this, the correct sequence of moves would never be discovered.

In order for a full-width search to make progress in this type of position, the evaluation function must produce goal direction. One way to do this is to provide a bonus for moves which reduce the distance between the White King and the passed pawn. A secondary goal is to reduce the distance between the White King and any Black pawns which are not defended by another pawn. A tertiary goal is to centralize the White King. The first step in developing a specific implementation of this plan is to identify the territory which is denied to the White King. For this purpose, we wish to determine which squares are controlled by the pawns. The White King cannot move to a square occupied by one of its own pawns, nor can it move to a square attacked by an opposing pawn. Figure 2 presents a map of the position with each of the forbidden squares darkened. The location of these

Figure 1: Chess position which demonstrates a weakness of the full-width forward search. In this example, White has a pawn on square f6 which could advance and be promoted if the Black King were out of the way. To win, the White King must come to the aid of the pawn. Since Black cannot attack White's pawns on c3 or g5 without leaving the passed pawn, he is helpless to stop White's maneuver. Although this analysis is obvious to an experienced player, a program using a full-width search would have to search its decision tree to a depth of 35 plies (ie: 35 half moves; a ply is defined as a move by one side) in order to come to the same conclusion.

BLACK

WHITE

BLACK
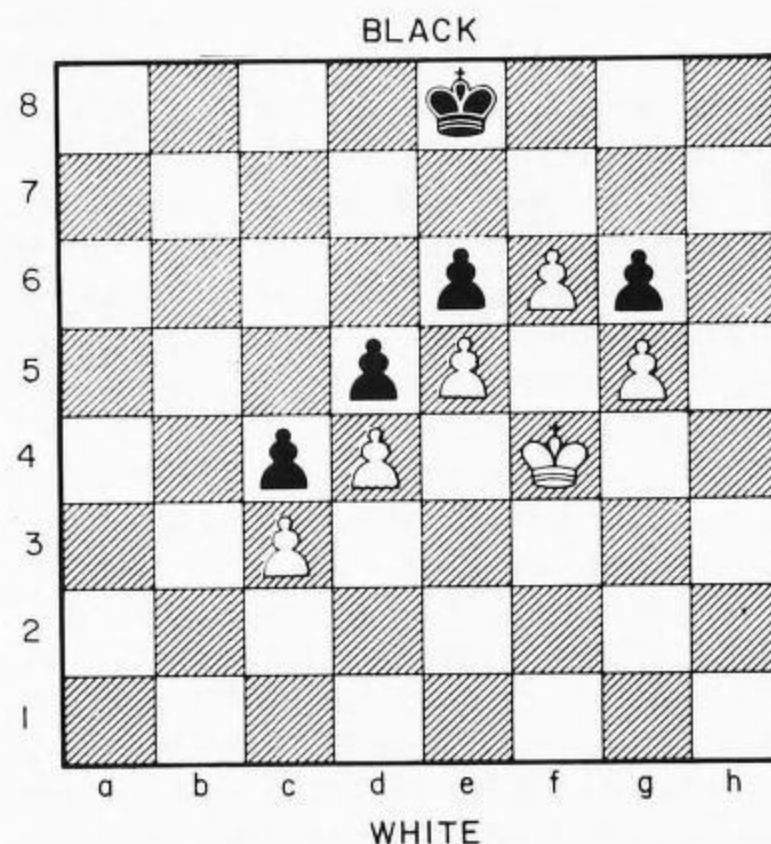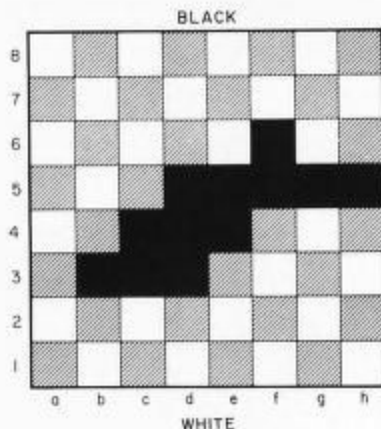
WHITE

Figure 2: Forbidden squares in the figure 1 position used to help White (the computer) evaluate the position more efficiently. The White King cannot move to a square occupied by one of its own pawns, nor can it move to a square attacked by an opposing pawn. All of these squares are darkened in the figure. This diagram is used in implementing the goal directed technique described by Church and Church (see figure 3).

"taboo" squares provides the defining boundaries for potential access routes to the desired goals. The second step in implementing this plan is to use a technique described by Church and Church. Starting at each goal object, work backward toward the attacking piece(s). In our case, we are interested in creating a reward gradient which encourages the White King to approach its own passed pawn and the target pawns. To do this, we consider one goal object at a time. All passed pawns are identified. In our example, only the White pawn at f6 qualifies. The two squares diagonally in front of it (e7 and g7) are each credited with 8 "points" each. All squares immediately adjacent to these squares (but not including squares inaccessible to the White King) are credited with 7 points. Next all squares adjacent to these squares (excluding inaccessible squares) are credited with 6 points. This process is continued until we run out of squares or until we have assigned all credits down to and including 1.

The next step in the process is to identify Black pawns which are not defended by other pawns (ie: targets). In this case, the pawns at e6 and g6 qualify. Credit these two squares and the adjacent ones with 5 points each, excluding darkened squares. Next,

credit squares adjacent to these with 4 points. Continue this process until all available squares have been exhausted or until the value of 1 has been assigned. This process is executed independently for each target pawn. The last step involves credit for centralization. The four most central squares (d4, d5, e4, e5) are credited with 3 points. The squares which surround these squares are credited with 2 points. The squares which surround those squares are credited with 1 point. Points are then removed from any square which is inaccessible to the White King. When this process has been completed, the credits are totaled for each square to provide a bonus map for the White King. This map is presented in

Figure 3: Bonus map for the White King in the position of figure 1, based on a technique described by Church and Church (see references). A goal is established for a particular attacking piece, in this case the White King, and an iterative numerical technique is used to implement it. The goal is to encourage the White King to approach its own passed pawn and the target pawns. (A target pawn is an enemy pawn not defended by other pawns.) Numerical figures of merit are assigned to strategic squares close to White's passed pawn and Black's undefended pawns. Points are also awarded or subtracted for positional characteristics such as centralization of squares, etc. A type of flow algorithm assigns lower and lower values to squares in direct proportion to their distances from the strategic squares, avoiding any forbidden squares. The resulting map of numbered squares enables the King to find the right pathway by constantly searching for ascending values of squares whenever possible.

BLACK

	a	b	c	d	e	f	g	h
8	6	9	12	14	15	15	15	14
7	6	10	13	16	18	18	18	15
6	6	10	14	17	18		17	15
5	6	10	14					
4	6	10				2	1	
3	6				2	2	1	
2	4	5	3	2	1	1	1	
1	2	2	2	1				

WHITE

BLACK

WHITE
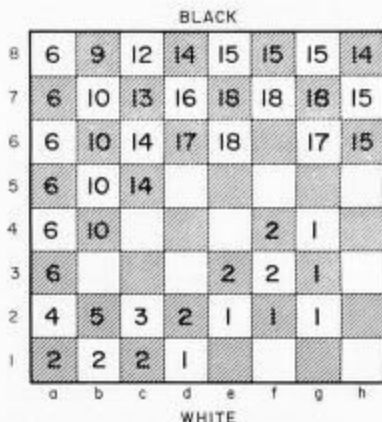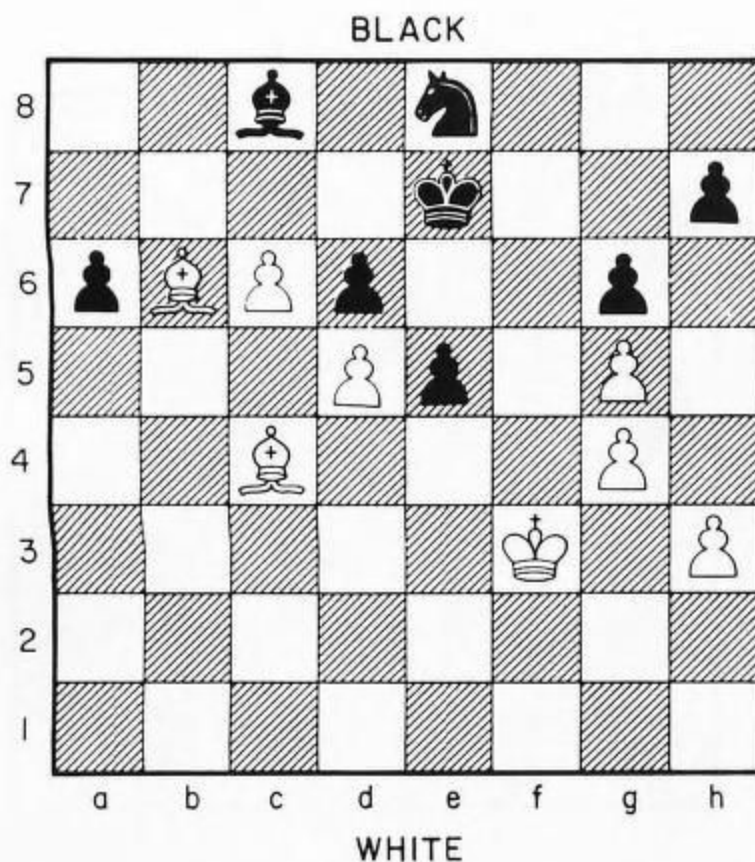
Figure 4: Another end game position, analyzed by the method of Church and Church in figures 5 and 6.

figure 3. By applying this bonus map to the terminal positions of the look-ahead search, the evaluation process will select a move sequence which causes the White King to gravitate in the proper direction. In fact, the correct sequence of moves will be selected even if White is restricted to a 5 ply search each time a move is selected. The bonus map, though simple in concept, has a tremendously beneficial effect.

There is an additional point which needs consideration. In our exposition, we have assumed that the pawns remained stationary. If a pawn were to move, the bonus map would have to be changed. This is not a major problem, however, since there are only a small number of positions that can result from pawn moves, and once the bonus map has been computed for a given configuration, it can be stored and used each time that configuration is encountered in the look-ahead tree. For this reason, the calculations which are required will not be particularly time consuming.

Another example of this strategy is based on the position presented in figure 4. This is a slight modification of figure 6.7 from the chapter of *Chess Skill in Man and Machine* by Church and Church. To apply our technique with respect to the bonus map for the White King it is necessary to determine which squares are not accessible to the White King by virtue of pawn control. As before, these include squares occupied by White pawns and squares attacked by Black pawns. The relevant squares are darkened in figure 5.

The next step is to locate passed pawns

for White. There is only one and it is located at c6. The two squares diagonally in front of this pawn (b7 and d7) are credited with 8 points. Squares adjacent to these squares which are not among the darkened squares in figure 5 are credited with 7 points. Squares adjacent to these receive 6 points. This process is continued until there are no more available squares or until the credit value of 1 has been assigned. The next step is to determine whether any Black pawns are potential targets. As before, a target pawn is defined as one which is not defended by a friendly pawn. In the present example, there are three candidates: the pawns at a6, d6 and h7. For each pawn, the value of 5 is credited to the pawn's square and the adjacent squares. Then the value of 4 is credited to each adjacent square. This process of establishing a gradient of decreasing values from 5 down to 1 as distance increases from the target is continued until the last values have been assigned. This is done for each target pawn and in each case, squares darkened in figure 5 are always excluded from the process. The last assignment process is conducted for centralization, with center squares (d4, d5, e4 and e5) receiving 3 credits each and neighboring squares receiving 2 credits. The squares one move in from the edge are assigned the value of 1 and then credits are removed from any square which has been darkened. The final step in developing a bonus map for the White King is to total the credits for each square.

The composite map is presented in figure 6. This set of bonus points will encourage the White King to move in the appropriate direction. Without this strategy

Figure 5: Forbidden squares for the position in figure 4.

BLACK

WHITE

Figure 7: A chess position which can be analyzed efficiently by means of the coordinate square concept proposed by Ken Church (see references). In this approach, the Black King must coordinate precisely with the White King in order to successfully defend its pawns. The technique is illustrated in table 1.

BLACK
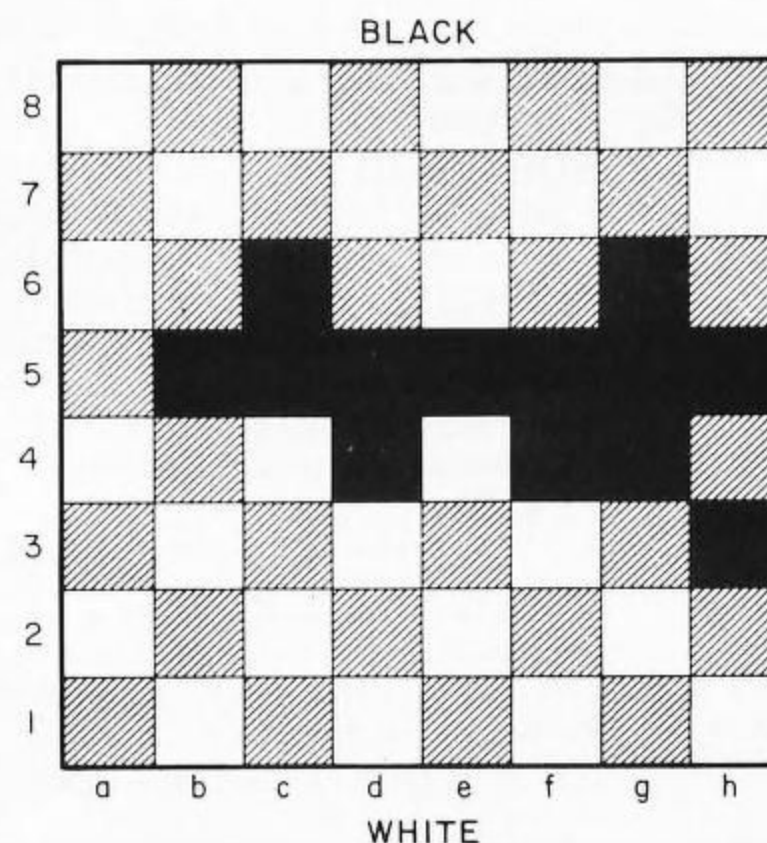
WHITE

Figure 6: Bonus map for the position of figure 4. Without this map, an 11 ply search would be required for the computer (White) to discover that the pawn at a6 can be captured. Using the map, only a 3 ply search is required.

BLACK

	a	b	c	d	e	f	g	h
8	14	15	16	16	16	15	13	11
7	15	18	18	19	18	16	14	11
6	15	17		19	19	17		11
5	14							
4	11	12	10		6			
3	8	9	10	7	5	3	1	
2	5	6	6	6	4	2	1	
1	3	3	3	3	3	1		
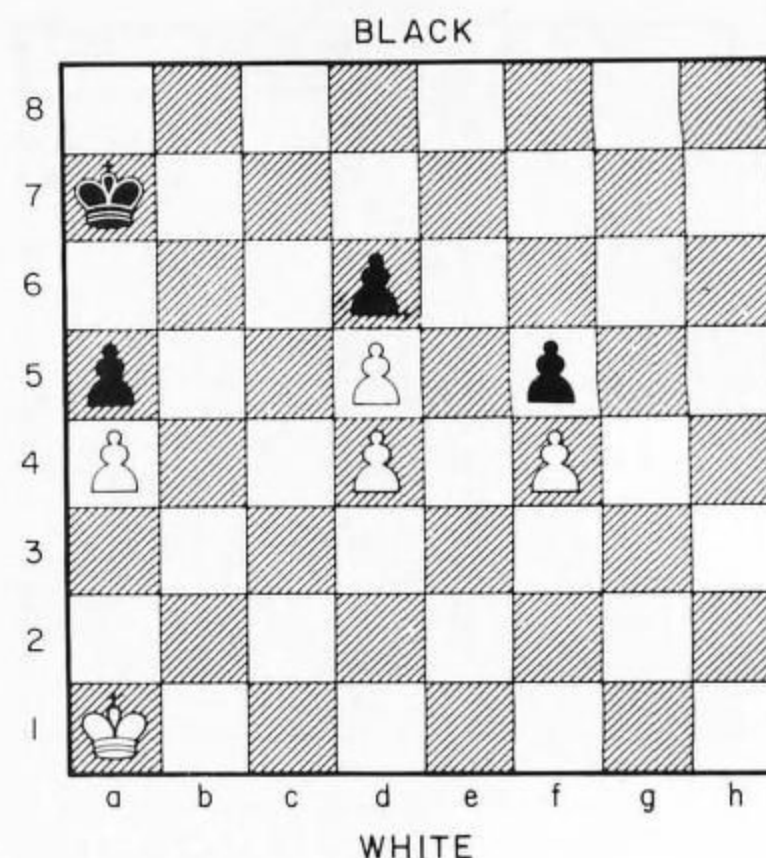
WHITE

an 11 ply search would be required for White to discover that the pawn at a6 can be captured. With the implementation of these attack gradients for the White King, however, the correct move can be selected with only a 3 ply search. As was the case in the previous example, the establishment of a plan within the evaluation function produces

a goal directed search without requiring an enormous look-ahead tree. This increase in efficiency is highly desirable.

Because the process is directed by the location of the pawns, changes in the map will occur infrequently and therefore only a relatively small number of bonus maps will be required for any one search. Once a map has been calculated for a particular pawn configuration, it can be stored and used later whenever it is needed. Although this strategy seems to work well in the examples we have presented, it is reasonable to ask whether this procedure will work in all end game situations. Unfortunately, the answer is no.

Consider the position presented in figure 7. This is a famous end game problem which appears as diagram 70 in Reuben Fine's classic chess book, *Basic Chess Endings* (see references). It was analyzed in 1975 by Monroe Newborn to determine if his special end game program, Peasant, could solve it. After several unsuccessful efforts, Newborn concluded that the problem would require about 25,000 hours of processor time before a solution could be found (see *Chess Skill in Man and Machine*, page 129). The problem is difficult, but not as impossible as Newborn suggests. Because Peasant does not have a transposition table, the program did not take advantage of the tremendous number of identical terminal positions which are encountered when an exhaustive search is made of this position. Because the pawns are locked, the only moves which are possible are King moves, and this greatly increases the potential number of transpositions.

The position was submitted to Northwestern's chess program Chess 4.5 running on the CYBER 176 system at Control Data headquarters in Minneapolis. David Cahlander discovered that Chess 4.5 could solve the problem after a 26 ply search! This required ten minutes of processor time on the powerful CYBER 176. Although it is interesting to know that the problem can be solved by a brute force search, this type of solution is not particularly elegant and it requires a level of hardware sophistication that is not likely to be available in the small system for a few years yet.

The Coordinate Squares Approach

What can be done to make this problem more manageable? Interestingly enough, there is a rather neat approach to problems of this type which has been examined in some detail by Ken Church in his under-

graduate thesis at MIT. Working with Richard Greenblatt as his advisor, Church applied the chess concept of coordinate squares to this position. The basic notion is that the Black King must coordinate precisely with the moves of the White King in order to successfully defend its pawns. For any particular square which the White King occupies, there are only a limited number of squares which the Black King can occupy and still hold his act together.

In his thesis, Ken Church presents a fairly extensive analysis of King and pawn end games. For our present purpose, we will limit our analysis to King and pawn end games in which the pawns are locked and we will modify Church's approach to suit our conditional evaluation strategy. The major difference is that Church attempts to discover a complete solution to the problem using the coordinate squares idea. We propose, instead, to use the coordinate squares approach to provide the evaluation function with additional chess knowledge. With this modification, a full-width search of reasonable depth can find the correct move.

Using figure 7 as an example, the first step in this process is to determine which squares are denied to each of the Kings by the existing pawn configuration. By noting that each King cannot move to a square that is occupied by its own pawn or that is attacked by an opponent's pawn, one can easily determine that squares a4, b4, c5, d4, d5, e4, e5, f4 and g4 are denied to the White King. Likewise, squares a5, b5, c5, c6, d6, e5, e6, f6 and g6 are denied to the Black King. Neither side has a passed pawn, but there are multiple targets, since none of the pawns are defended by friendly pawns.

By applying the strategy described earlier, it is possible to calculate a composite attack map for the White King on the basis of the target pawns at a5, d6, and f5 and taking into account the centralization subgoal. The resulting map for Fine's position is presented in figure 8. The squares without a number are the squares which are denied to the White King because of the pawn structure. Given the position of the White King (a1), a shallow search using this attack map as part of the evaluation function would encourage the White King to approach the target pawn at a5 (eg: b2, c3, c4, b5, a5). If the Black King were more than five moves from a5, this sequence of moves would lead to success. Given that the Black King is at a7, however, this plan is doomed to failure. In fact, the first move in the sequence, b2, is fatal and transforms a winning position into a draw. There are two important conclusions that follow from this discovery. The first is

BLACK

Figure 8: Bonus map for the position of figure 7, a composite attack map for White based on the target pawns at a5, d6 and f5, and taking into account the centralization subgoal.

that our simple goal-gradient approach does not always work. The second is that chess end games are much more difficult than a novice player might suppose.

Let us extend Ken Church's ideas and apply the concept of coordinate squares to this position. First, we wish to assign each of the squares to one of three categories: under the influence of the Black King, under the influence of the White King, or contested. To do this we compute the distance from each King to each square, given the constraints imposed by the existing pawn structure. This creates two distance maps, one for the White King and one for the Black King. For squares which are not accessible to one or both of the Kings, we assign a distance score based on the number of King moves required to reach that square by traveling across accessible squares. Next, each square which is closer in moves to the Black King than to the White King and is not denied to the Black King is assigned to Black. Each square which is closer to the White King than to the Black King and is not denied to the White King is assigned to White. The remaining squares are assigned to the contested category. The results of this procedure are summarized in figure 9. The squares assigned to Black are indicated by the letter B and the squares assigned to White are indicated by a W. The blank squares belong in the contested category.

If the territory under the influence of either King is adjacent to an opponent's pawn, the contest is essentially settled since

Figure 9: The square control concept applied to the position of figure 7. Each of the squares is assigned to one of three categories: under the influence of the Black King, under the influence of the White King, or contested. To do this, the distance from each King to each square is computed, given the constraints imposed by the existing pawn structure. Each square closer in moves to the Black King and not denied to the Black King is assigned to Black, and vice-versa. The remaining squares are labelled as contested. Through a complex series of manipulations and the use of so-called frontier squares (see text), White is actively directed to attack Black's pawns using the strategy of trying to prevent Black from moving onto strategic coordinate squares which are vital to Black's defense.

BLACK

	a	b	c	d	e	f	g	h
8	B	B	B	B	B	B	B	B
7	B	B	B	B	B	B	B	B
6	B	B				B	B	B
5								B
4			W					W
3	W	W	W	W	W	W	W	W
2	W	W	W	W	W	W	W	W
1	W	W	W	W	W	W	W	W

WHITE

that pawn would be open for capture. Since this is not the case for the present position, we wish to define a special category of squares called *frontier squares*. A frontier square is any square under your influence that is adjacent to an accessible contested square or is adjacent to an accessible square under the influence of the opponent.

For the position diagramed in figure 7, the frontier squares for White are c4 and h4. The next step is to determine, for each of these frontier squares, the set of squares under Black's influence which, if the Black King were located on that square, would prevent the White King from moving from the frontier square to any of the contested squares or to any of Black's squares. For the frontier square at c4, the Black King would have to be at either a6 or b6 to prevent the White King from penetrating to b5. For the frontier square at h4, the Black King would have to be at g6 or h6 to prevent penetration by the White King. (Note that the Black King could not legally be at h5 if the White King were at h4.) These defense squares for Black can be determined by the machine by placing the White King on the frontier square and conducting a shallow tree search with White to move first and determining

empirically which locations for the Black King successfully repel the invader.

The next step in this process is to determine the shortest distance between each pair of frontier squares. For the present position, there are only two frontier squares and thus one minimal distance. Five King moves are required to travel between the two frontier squares. If Black is to be successful in defending, the Black King must be able to move from a defense square for h4 to a defense square for c4 in the same number or in fewer moves than it takes the White King to travel between the two frontier squares.

For this reason, each square in Black's defense set for c4 must be five or fewer moves from one of the defense squares for h4. Also, each square in the defense set for h4 must be five or fewer moves from one of the defense squares for c4. This requirement places a further restriction on those squares which satisfy the necessary defense conditions. One will note that a6 is six moves from the nearest square in the defense set for h4. Also, h6 is six moves from the nearest square in the defense set for c4. Therefore, the true defense set for c4 contains only b6 (a6 will not suffice). The true defense set for h4 contains only g6 (h6 will not suffice). Thus, we have determined that when the White King is on c4 and has the move, there is one, and only one, coordinate square for the Black King (b6). If the White King is on h4 and has the move, there is one, and only one, coordinate square for the Black King (g6).

The next step is to generalize this analysis to squares in White's territory which are immediately adjacent to the frontier squares. In this case, squares b3, c3, d3, g3 and h3. The square at b3 is one King move from the frontier square at c4 and six moves from the frontier square at h4. If the White King is at b3, therefore, the Black King must be on a square which is simultaneously one move from b6 and six or fewer moves from g6. The squares which satisfy this condition (ie: the coordinate squares for b3) are a6, a7, b7, and c7. This same set of calculations can be made for the other adjacent squares. The coordinate squares for c3 are b7 and c7. For d3, there is only one coordinate square, namely c7. Since the White King can move directly from c3 to d3 and Black must move to c7, and only c7, to maintain his defense, it is not possible for him to be on c7 when the White King is on c3. If he were, he would not be able to move when White moved from c3 to d3 and still satisfy the defense requirements. For this reason, only square b7 is sufficient for Black when White is on c3. In addition, since b3 is adjacent to c3, the coordinate square for c3 is not avail-

able for b3. Thus the set for b3 is further restricted to a6, a7 and c7.

If we examine g3, we will discover that it is one move from the frontier square at h4 and four moves from the frontier square at c4. This implies that the Black King must be on a square which is one move from b6 and four or fewer moves from g6. There are only two squares which satisfy this requirement, namely, f6 and f7. Therefore we can conclude that no square other than f6 or f7 will serve as a coordinate for g3. When we examine h3, we will find that there are three potential coordinate squares: f6, f7 and g7. Since this set shares f6 and f7 with the defense squares for g3, further restrictions are implied. It is not possible for the same square to serve as a coordinate square for two adjacent squares since it is not possible for Black to pass when it is his turn to move. Therefore if f6 is assigned to h3, then f7 *must* be assigned to g3. If f7 is assigned to h3, then f6 *must* be assigned to g3.

The next step in this process is to determine the set of coordinate squares for each square on the minimum pathway(s) between the two frontier squares for which the coordinate squares have not yet been determined. The new squares are e2, e3, f2 and f3. By following the same analysis as before, we can determine that the coordinate squares for e2 and e3 are d7 and d8. The coordinate squares for f2 and f3 are e7 and e8. Because of the adjacency restrictions, the assignment of one of these values automatically restricts the other square to the remaining value.

The results of our coordinate square analysis are summarized in table 1. When it is Black's turn to move and White has moved to one of the squares listed in the table, Black must be able to move to a coordinate square. For this reason, the evaluation function for the machine can be modified to give

Square of the White King	Coordinate Squares for the Black King
b3	a6, a7, c7
c3	b7
c4	b6
d3	c7
e2	d7, d8
e3	d7, d8
f2	e7, e8
f3	e7, e8
g3	f6, f7
h3	f6, f7, g7
h4	g6

Table 1: Results of the coordinate square analysis for the position of figure 7. Shown are the potential squares for the Black King which defend against the White King's threats when it is White's turn to move.

a bonus of 20 points to White for any terminal position in the look-ahead tree where it is Black's turn to move and the Black King is more than one move from a necessary coordinate square. If it is White's turn to move, a 20 point bonus will be awarded to any terminal position in the look-ahead tree where Black is not located on a necessary coordinate square.

Let us consider how this in combination with the White King attack map (figure 8) will affect the outcome of the look-ahead search. The machine will try to find a pathway to squares c3 or d3 because their attack value of 7 is higher than any of the surrounding squares. Even better would be a pathway to c4, since its attack value of 10 is larger than 7. In each of these cases, the machine will also try to satisfy the condition that Black cannot be on a proper coordinate square when the White King reaches c3, d3, or c4 so that the additional 20 point bonus is also earned. In attempting to do this, it will find that if the White King moves from a1 to either a2 or b2 on his first move, the 20 point bonus will be lost forever. The reason is that either of these moves allows the Black King to coordinate and, because of the minimax strategy, the tree search will always assume replies for Black which maintain this coordination. If the White King's first move is to square b1, the Black King cannot coordinate and the 20 point bonus will *still* be available at some of the terminal positions in the tree. It is not surprising, therefore, to find Reuben Fine advising that K-N1 is the only move for White which preserves the win.

In order for the machine to find this move, assuming that both the attack map and the coordinate squares information are incorporated in the evaluation function, a search of nine plies is required. This is a tremendous improvement over the 26 ply search required by the unmodified program. In order to actually win a Black pawn, the White King must move to c3 or c4 with Black not in coordination and make a 13 ply look-ahead search. If the White King moves to d3 with Black not in coordination, an 11 ply search will suffice. In order to prevent a draw, White will avoid repeating identical positions and thus will eventually travel to e3. From this vantage point, the win of a pawn can be visualized with a 9 ply search. Therefore, the problem could be solved by the machine if it searched to a depth of nine plies for each move calculation. With a program such as Chess 4.5, a 9 ply search for this position can be conducted in less than two minutes on even a medium power computer.

The procedures which we have described

are applicable to a wide range of end game positions. The coordinate squares analysis demonstrates that even highly complex end game positions are manageable when the full-width search employs a sufficiently knowledgeable evaluation function. Although the examples we have discussed encompass only a few types of chess positions, we hope that the reader will envision the power which is potentially available when the evaluation function is modified to incorporate relevant chess knowledge. The implementation of this approach on a broad scale should eventually produce chess programs which can be run on medium power machines and still compete on equal terms with strong human players.

Quiescence

Another important area for the application of chess knowledge is the problem of *quiescence*. It is essential that the static evaluation function not be applied to a turbulent position. If the next move has the potential to produce a major perturbation of the situation, the evaluation which is rendered will not be accurate. For example, it makes little sense to apply a static evaluation function in the middle of a piece exchange or when one of the Kings is in check. In each case, the judgment which is rendered will not be reliable. For this reason Chess 4.5 presently goes beyond the predetermined search depth at "terminal" positions where a capture might be profitable for the side whose turn it is to move, where certain types of checking moves are possible, or where a pawn is on the seventh rank. This extended search facility is called the quiescence search, and its major objective is to produce reasonably static positions for which the evaluation function can provide accurate assessments.

A weakness of this present implementation is that the definition of a turbulent position is much too narrow. There are many situations in addition to capture threats, checks on the King, and pawn promotion threats which are clearly turbulent. Larry Harris has characterized some of these in chapter 7 of *Chess Skill in Man and Machine*. Harris includes in this category positions which involve a pawn lever, a back rank mate threat, or sacrifice potential. The interested reader can consult Harris' chapter for operational definitions of these patterns. It is essential to note that these and other important patterns are not easily detected. In each case, a fairly sophisticated pattern analysis capability is required. A reasonable

goal for improving the present forward search chess programs would be the development of an efficient procedure for detecting potential sources of turbulence. The central objective would be to use this information as one of the decision criteria for terminating search at a node. If the position is not quiescence in respect to a potential perturbation which has been detected, the look-ahead process should be continued.

For example, during the opening when the machine leaves its library with information that the control of a particular square is an important objective, the decisions about search termination can consider whether the position is quiescent in respect to perturbations which might influence control of the key square. Another example of this idea involves the end game. If the preliminary analysis indicates that a particular pawn should be an attack target, the decision for search termination should consider whether each position is quiescent with respect to this goal. Positions at the predetermined depth level will be evaluated only if all potential attackers are more than two moves away from the target. When one or more attackers are close to the goal, the search process will be continued to determine if capture is feasible. This modification of the search process introduces a goal directed selective search at the terminal positions of the full-width tree. The addition of several extra plies of search at relevant nodes in the tree can mean the difference between finding and just missing an important continuation. This type of facility is difficult to implement and difficult to control properly, but the potential gains are such that the effort is worthwhile.

Establishing Appropriate Goals

In order to implement this goal direction feature in the evaluation function and quiescence search, it is necessary to recognize that a goal which may be of paramount importance at the base node of the look-ahead tree may no longer be relevant at some of the terminal nodes. Intervening moves may accomplish the necessary goal or may alter the situation such that it is no longer possible. In these cases, the conditional evaluation function would be directed at an inappropriate goal. One way to deal with this problem would be to select goals which were both general and long range. In this case, they should continue to be relevant at the terminal nodes of the look-ahead tree. Unfortunately, this is a fairly severe limitation on the goal directed search and is

therefore not desirable. A second approach would be to apply pattern analysis at each terminal node instead of at the base node only. In this case, the goals which were selected would always be relevant to the position. This procedure would be very time-consuming, since feature analysis is a complex process. The essential aspect of the problem is a time relevance trade-off in which a guarantee that relevant goals are being pursued requires a heavy investment in additional computing time. The third and most reasonable approach would be to designate which features of the position are crucial to each particular goal and to incrementally update our goals (and thus the evaluation function and the decision rules for the quiescence search) whenever these features change. This is a highly sophisticated approach which would be difficult to implement.

Conclusion

Let us summarize our conclusions and relate them to the world of personal computing. We have attempted to argue that a full-width search strategy is feasible with a small computer, and that ultimately this approach will produce better chess than a selective search strategy. For this plan to be successful, it is necessary to employ software and hardware suited to the task. The software must incorporate recent improvements in tree searching strategy (ie: α-β pruning, the capture and killer heuristics, iterative searching, staged move generation, incremental updating, serial evaluation and transposition analysis) as well as other refinements such as conditional evaluations which provide goal direction to the search process.

On the hardware side, it is necessary to have a reasonably powerful system. Although there have been a number of recent efforts to program microprocessor systems to play chess, the games which have resulted have not been comparable to those played by established large system programs. Although it is quite an accomplishment to produce even rudimentary chess from a microprocessor system, the level of play to date is not very encouraging. An example of this type of game appeared recently in March 1978 BYTE, "Microchess 1.5 versus Dark Horse," page 166.

The type of chess program described in this article requires reasonably powerful hardware in order to provide an interesting game. Because of the many operations requiring bit map manipulation, a 16 bit processor is much more desirable than an 8 bit processor. It is more efficient to represent

a set of 64 squares with four 16 bit words than with eight 8 bit words. With a need for computing power in mind, one might select a microprocessor system based on one of the new high-speed 16 bit processors such as the Zilog Z-8000 or the Intel 8086. In addition, this type of program will require quite a bit of memory. The program itself will require about 20 K bytes and the transposition table, if implemented, will need at least another 20 K bytes. If the programmer plans to add chess knowledge for conditional evaluations, a total of 64 K bytes is desirable. An opening library which is sufficient to keep a skilled opponent on his toes requires disk storage.

These considerations may dampen the enthusiasm of many would-be chess programmers. On the other hand, a realistic orientation at the start could save a great deal of grief along the way. When implemented on fairly sophisticated hardware, our demonstration chess program will usually provide a reasonable chess move after two or three minutes of computation. If more time is available (eg: selecting a move for a postal chess game by letting the machine "think" for several hours), a fairly respectable level of play can be anticipated. With future hardware improvements, this type of program may soon become reasonably competitive at tournament time limits, even on a personal computing system.■

REFERENCES

1. Berliner, H, *Chess as Problem Solving: The Development of a Tactics Analyzer*, unpublished doctoral thesis, Carnegie-Mellon University, Pittsburgh, 1974.
2. Church, K W, "Coordinate Squares: A Solution to Many Chess Pawn Endgames," undergraduate thesis, Massachusetts Institute of Technology, June 1978.
3. Church, R M, and Church, K W, "Plans, Goals, and Search Strategies for the Selection of a Move in Chess," Frey, P W (ed), *Chess Skill in Man and Machine*, Springer-Verlag, New York, 1977.
4. Fine, Reuben, *Basic Chess Endings*, David McKay Company Inc, New York, 1941.
5. Harris, L R, "The Heuristic Search: An Alternative to the Alpha-Beta Minimax Procedure," Frey, P W (ed), *Chess Skill in Man and Machine*, Springer-Verlag, New York, 1977.
6. Newborn, M, "PEASANT: An Endgame Program for Kings and Pawns," *Chess Skill in Man and Machine*, Frey, P W (ed), Springer-Verlag, New York, 1977.
7. Shannon, C E, "Programming a Computer for Playing Chess," *Philosophical Magazine*, volume 41, 1950, pages 256 thru 275.
8. Zobrist, A L, and Carlson, F R Jr, "An Advice-Taking Chess Computer," *Scientific American*, volume 228, number 12, June 1973, pages 92 thru 105.