MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PROJECT MAC

Artificial Intelligence
Memo. No. 174

April 1969

The Greenblatt Chess Program

Richard D. Greenblatt
Donald E. Eastlake, III
Stephen D. Crocker

The Greenblatt chess program*

by *RICHARD D. GREENBLATT,*

DONALD E. EASTLAKE, III,

and

STEPHEN D. CROCKER

Massachussetts Institute of Technology
Cambridge, Massachusetts

INTRODUCTION

Since mid-November 1966 a chess program has been under development at the Artificial Intelligence Laboratory of Project MAC at M.I.T. This paper describes the state of the program as of August 1967 and gives some of the details of the heuristics and algorithms employed.

Development of the program

The first step we took was to produce a simulated chess set, whereby the computer would display the current board and accept moves in standard chess notation through a teletype. Routines to evaluate the board, generate legal moves, and perform a minimax search of a game tree were quickly added, and with further development the program played in its first tournament in February of 1967. It played in local tournaments again in March, April and May. The improvement it has shown is due to additional programming and debugging, not learning.

Table 1 summarizes the program's performance in tournaments. For comparison, the mean of all U. S. tournament players is about 1800, while the mean of all chess players is in the 800 to 1000 range. The program wins about 80% of its games against non-tournament players.

Table 1

	Won	Lost	Drew	Rating	Performance Rating
Feb	0	4	1	1243	1243
Mar	1	4	0	1330	1360
Apr	2	0	2	1450	1640
May	0	4	0	1400	(weakest opponent was 1680)

The program is an honorary member of the United States Chess Federation and the Massachusetts Chess Association, under the name Mac Hack Six. In the April amateur (non master) tournament the program won the class D trophy.

A short history of chess playing programs

The first important paper dealing with methods for programming chess playing programs was written by Shannon in 1949 (1). In his paper the concept of minimax tree search is used. In 1950, Turing described a hand simulation of a chess program (2). In Turing's paper the concept of a dead position is introduced. A dead position is one in which neither side can immediately gain by making a capture. These papers and two programs known as the Los Alamos program and the Bernstein program (8) are described in a paper by Newell, Shaw and Simon (3). Their paper describes a chess program which deviates from the analysis done in the previous programs in that it employs explicit plans and goals in making its moves. A more recent program in the Newell, Shaw and Simon tradition is the MATER program of Simon and Baylor (4). This program, however, deals only with mating combinations of a few moves. The program which is most similar to our program is described in a Bachelor's thesis by Alan Kotok (5). A variant

*The program was written primarily by the first author who was assisted by the second author. Work reported herein was supported in part by Project MAC, and M.I.T. research program sponsored by the Advanced Research Project Agency, Department of Defense, under Office of Naval Research Contract Number Nonr-4102(01). Reproduction in whole or in part is permitted for any purpose of the United States Government.

of Kotok's program was used by John Mccarthy in a chess match with a Russian program (6). It is fair to say that our program is stronger than any of these programs in across the board play.

Approach and environment

The approach we have taken in writing the chess program has been quite pragmatic. We did not pretend to be writing a general problem solving system, but addressed ourselves directly to the problems of chess. The goal of being able to play complete games under tournament conditions has meant that most of the effort so far has gone into building an efficient and effective tactical base. Therefore, consideration of learning mechanisms, strategic planning mechanisms and special case treatment of opening and end play were forestalled. Book openings were recently added, although it turned out that the computer played much better in the openings without them than was expected.

The environment in which this program has been developed is, we feel, more advantageous than for any previous chess program. The machine used is the Digital Equipment Corporation PDP-6 in the Artificial Intelligence Laboratory of Project MAC. This machine is equipped with a 256K Fabritek memory, a DEC 340 graphic display, a model 35 teletype, a line printer, and four Dectape drives.

The machine was originally used on-line by one person at a time and the teletype and graphic display provided a high degree of interaction between the user and the program.

The software provides for the editing, assembling and debugging of programs and makes full use of the interactive facilities. The mass memory and a time sharing system were added after most of the initial work on the program was done.

The mass memory has proved very useful in later versions of the program, but it should be noted that at the time of the first two tournaments the machine had only a 16K memory.

The program was written entirely in MIDAS, a PDP-6 macro assembly language (7). MIDAS was chosen for this program because of the ease of constructing and debugging in it the complex data and control manipulations involved in writing a high performance chess program. Large economies of time and memory are also effected by writing in assembly language. The order code of the PDP-6 computer is exceptionally well suited to assembly language coding.

The program has been edited and reassembled over 200 times and has played several hundred complete games; consequently, those portions of the code which have been in use for a while are extremely reliable and the program's performance has yielded many ideas for improvement.

Debugging aids

The chess program contains several powerful interaction debugging aids. These are briefly listed below:

1) scope display of the board and game history
2) acceptance of standard chess notation input (e.g., P-K4)
3) scope display of evaluation at any selected node in the game tree
4) tracing of specific move in plausible move generator, displaying all factors that went into plausibility and a comment about each. (e.g., 10 points for unblocking the white queen bishop so that it now attacks QN5)
5) printed record of plausibility of all moves at top level and main variation from each top level move investigated
6) statistics on how long the computation took, how many plausible move generations, feedovers and static evaluations occurred, etc. (These terms are desicribed below.)

An outline of the program

We begin this section with a definition of some of the important chess terms and then describe the major components and the flow of control.

Chess terms

Ply—one play by one side. Two plies equal one complete move.

Pinned—a piece is pinned if moving it exposes (discovers) an attack on another piece, rendering that piece *en prise* (see below). If an attack on the king is thereby discovered, the original move is illegal.

Safe move—a legal move for a piece that does not render it immediately *en prise*.

Trapped—a piece is trapped if it has no safe moves.

Isolated, backward, doubled, tripled—various pawn structure defects. (See section on static board evaluator for further discussion.)

Development value—refers to a piece's range over the board (number of squares and importance of those squares) in a particular position.

Principal variation—the sequence of moves the computer thinks most likely in a position.

Game tree—the set of all positions considered by the program in a search, visualized in the form of a tree. This tree is diagrammed with the ancestor positions near the top of the page.

Game tree node—a node in the game tree represents a position. The line leading to the node represents the move which lead to that position. The lines down

from the node represent moves leading to successor positions.

En prise

A piece is *en prise* when it is under attack and is inadequately defended. An example is a knight under attack by a pawn and defended by a pawn (or any other piece). Clearly it is in the opponent's interest to take the knight even though he would lose the pawn. A more complex case is where a knight is under attack by a bishop and rook and defended by a pawn. In this case, it is the existence of a second attacker (the rook) which makes the knight *en prise*. The situation is further complicated when some of the attackers or defenders are pinned. Often a complete check for whether a piece is *en prise* can be quite complex, so in the program only partial checks are made at various stages. A typical determination is made by considering the value of the piece attacked, the number of attackers, the number of defenders and the values of the least valuable attacker and defender. *En prise* checks are made to determine whether or not the board is stable (in a dead state) and are also made at several places in the plausible move generator.

Description of a simplified minimax search

The program is organized around a minimax search of a game tree. The branches of the tree correspond to alternative moves and the nodes correspond to positions. Beginning with the actual position in which it is the machine's turn to move, a routine known as the plausible move generator lists each legal move and assigns a plausibility value to each move. The moves are then ordered according to their plausibility score and a subset of the moves is selected for further consideration. The first move of this subset is then postulated and the resulting position calculated. This process is repeated recursively until a certain depth is reached, at which point the position is evaluated using another routine known as the position evaluator. The position evaluator makes use of a function called the static board evaluator to compute a numerical value for the position. This numerical value has the significance that a positive value represents an advantage for white (the larger the number, the greater the advantage), a negative number represents an advantage for black, and zero represents an even game.

After a position is evaluated, the value is returned to the level above and it becomes the "best value so far" for that position. Each other move of the subset selected by the plausible move generator is treated in the same manner, and when a value is obtained for the move, the value is compared to the best value found so far. If the value associated with the move just considered is better for the side to move than the best value so far, the new move is remembered and its value becomes the new best value so far. "Better" is synonomous with "algebraically greater" if white is the side to move and "algebraically less" if black is the side to move. If two moves lead to the same value, it is presumed that the first is slightly better because it received a higher plausibility score. After all of the selected moves at a position have been considered, the best value so far and the move associated with that value are returned to the level above. The process is continued until a value for the actual current position is determined. The sequence of moves which are the best moves is called the principal variation.

Since it not feasible to consider either all moves at any level or an indefinite number of levels, some severe constraints are placed on the search. The basic search (just described) starts from the current game position and proceeds a fixed number of plies. A position evaluator is applied to each of the end positions of the basic search tree. This routine tests a condition known as the feedover condition (see below) of the position. If this condition is true, then the plausible move generator is reapplied (up to certain limits) and the position evaluator called at the resulting nodes. If the feedover condition is false, a value for the position is developed by calling the static board evaluator and by exploring all plausible captures. Plausible captures are generated in a manner similar to regular plausible moves, but they must appear to lead to relative gain of material, either through an actual capture or a pawn promotion. Positions resulting from plausible captures are turned over to the position evaluator. The program will explore sequences of favorable captures or pawn promotions without a depth or width limit. This is necessary because otherwise pieces might be left *en prise* and this would result in blunders of the first magnitude.

Should the program at any depth reach a checkmate, stalemate, or draw by repetition of the position, it will immediately return to the previous level with an appropriate mate or draw value. Also the alpha-beta algorithm may provide an exit at any level in the tree except the topmost level.

The details of the static board evaluator, the plausible move generator, the feedover conditions and the determination of the width of the search are all given In the next section. The alpha-beta algorithm is described in a later section.

The plausible move generator has three basic goals.

The plausible move generation

1) To select a subset of legal moves for inclusion in the move tree.

2) To order these moves so as to optimize the advantage the program receives from the alpha-beta tree-pruning algorithm.

3) To calculate the positional and developmental values that will decide the program's move if several moves lead to the same static value.

The analysis done in the plausible move generator is done on a per move basis rather than a per position basis—that is, for example, "this move is bad because it blocks my bishop" rather than "the position resulting after this move is bad because the bishop is blocked." To determine the latter fact starting just with the board position would require considerably more processing and analysis of irrelevent details.

Numerous heuristics are available for the plausible move generator. As is frequently the case with heuristics, they may not be valid in particular situations, therefore a program organization is required which allows for the interaction of the heuristics to determine which of them most nearly applies in the current situation.

Very generally speaking, two types of heurstic interaction are used in the chess program. One type of interaction involves enumeration of all combinations of facts. Such an enumeraion leads to the familiar tree structure with the nodes of the tree corresponding to subdecisions. Each node is dependent upon only one fact. The size of this tree grows exponentially with the number of facts involved, severely limiting the usefulness of this technique.

The second type of interaction uses weighted sums. A value is assigned to each fact proportional to its average importance, and each move is scored as the sum of the weights of the attributes which apply to the move. In the simplest case, the move with the highest score is chosen. The complexity of this process grows linearly with the number of facts; not exponentially. Also there is opportunity for a large number of small factors to add up and sway the final decision in a way hard to achieve with the enumerative process. While it is true that any linear weighting process can be simulated by an appropriate enumerative process, for large numbers of facts the size of the enumerative process becomes absolutely unmanageable. So for practical purposes the techniques are distinct. Linear weighting methods have been used before in game playing programs; nevertheless they have a weakness in that they basically fail to take into account the relationships that may exist between the facts. To put it another way, the importance of a fact may vary depending on the position. Non-linear techniques have been proposed to solve this problem, but chess is a game where the relationships are so complicated and numerous that it is unlikely much additional headway could be made by making the weighting nonlinear.

The solution incorporated in the current chess program is a nested combination of the two methods. The top level decision process is enumerative; that is a game tree is searched. However, selection of moves for the game tree is controlled by a weighted decision process, the plausible move generator. Many of the "facts" going into the plausible move score are themselves enumeratively determined using such criteria as whether the move is a capture or not, whether various pieces are *en prise* or not, etc. These predicates (or in some cases weights) are themselves decisions which are made by enumerative or weighted sum decision processes and so forth. The net result is that the program is frequently able to grasp the effect of particular features of the position that make some otherwise insignificant factor more important.

Details of the major components

The major reason for the quality of the program's play is that considerable chess knowledge has been programmed in. In this section much of the detail is presented. To some extent, these details are volatile, so what follows is more representative than definitive.

The plausible move generator

About 50 identifiable heuristics are used in computing the plausibility. Many, though, apply only in special cases such as captures, moves with certain pieces, or certain stages of the game.

Each square is assigned a importance during each plausible move computation, corresponding roughly to the estimated worth of having an additional piece bearing on the square or the cost of taking away a piece presently bearing on the square. The principal criteria used for assigning these values include the closeness of the square to the center of the board, its proximity to the opponent's king, and its occupation by one of our pieces which is *en prise*. Small values are given for occupation of the square by one of our pieces and for its closeness to opponent's side of the board.

The current developmental value of a piece is the sum of the values of all the squares it attacks (can move to in one move) plus values accumulated for actual attacks on enemy pieces. The new developmental value is similarily computed assuming the piece is in its proposed new location. The difference between these is used as a factor in the plausibility, encouraging developing moves and discouraging apositional moves. Gains or losses in development resulting from blocking or unblocking the opponent's or our pieces are also considered in the developmental value. Of course,

putting opponent's pieces *en prise* is plausible. Furthermore, factors are added to encourage certain types of attacks on probable weak spots (weak pawns, pinned pieces, pieces defending other pieces, etc.). When a capture is made, the capturing move receives the developmental value of the piece captured. Some very specialized heuristics also are employed, such as, "it is bad to move pieces in front of center pawns on their original squares, thereby tending to block your own center."

Several weaknesses were noticed in the early play of the program and measures were taken to eliminate them. For example, sometimes an apositional move would receive a high value because it was an attacking move. If this leads to gain, all is well and good; but if the opponent can simply move away then the move is a pointless waste of time. So, moves are scored separately on their positionality and if this is bad these moves are rejected if there is some other move which leads to an equal terminal score.

Evaluation of the board

The value of the board is given by

$$S = B + R + P + K + C, \text{ where}$$

B is a material balance term,
R is a piece ratio change term,
P is a pawn structure term,
K is a king safety term, and
C is a center control term.

The material balance term makes use of the evaluation shown in table 2.

Table 2

Piece	Value	Value Relative to Pawn
Pawn	128	1.
Knight	416	3.25
Bishop	448	3.50
Rook	640	5
Queen	1248	9.75
King	1536	12

The value of B is the sum of the values of the white pieces on the board minus the sum of the values of the black pieces on the board.

The piece ratio change term is aimed at promoting even or near even trades when ahead and avoiding them when behind. The ratio of white pieces to black pieces at the current node is compared to that ratio at the top of the tree. If the side to move is three pawns ahead, for example, a trade of a bishop for a knight will receive a positive piece ratio term.

$R = \{N/(T\text{-}1)\}*¼*M$, where

N is the ratio of white material to black material at the node being evaluated,
T is the ratio of white material to black material at the top node of the tree, and
M is the material for one side at the beginning of the game.

The ratios are evaluated using the table above, except that the king is valued at 1 instead of 1536.

It has been pointed out that the piece ratio change is slightly asymmetric with respect to color, but this is of little consequence since this term only has effect when one side is very significantly ahead.

The pawn structure term depends upon four subterms, which score positively for each of the following: tripling up of opponent's pawns (doubling only if isolated), the isolation of opponent's pawns, our own passed pawns, and the opponent's backward pawns. Backward pawns are considered weaker if they occur on an open file or if the opponent has rooks or queens on the board.

A pawn is isolated if there are no friendly pawns on an adjacent file.

A pawn is passed if there are no enemy pawns in front of it in the same file or an adjacent file.

A pawn is backward according to the following criteria:

If it is defended by a pawn, it is not backward.

If it can be defended by a pawn in one move, (assuming moves through friendly pieces are permitted), it is not backward unless it is on the second rank and the only pawn move which would defend it is a double advance which would then subject it to *en passant* capture.

If there is a defending pawn move blocked by an enemy piece, if the pawn is blocked, the pawn is backward. If an adjacent pawn is blocked, the pawn is not backward.

Otherwise, if there are friendly pawns in adjacent files such that the pawn would become defended if advanced far enough, the pawn is backward. Otherwise, the pawn is not backward (i.e., it's probably isolated).

The king safety term applies only if queens are on the board. The king safety term (K) is eight times the rank of the black king minus eight times the rank of the white king.

The center control term (C) is +1 if there is at least one white pawn in the center four squares and no black pawn, —1 if there is at least one black pawn in the center four squares and no white pawn, and zero otherwise.

Feedover conditions

The feedover condition is true if:

1) the side to move has a piece *en prise* and one of the following:
   A) the side to move is in check.

B) the *en prise piece* is trapped or pinned.

2) The side to move has two or more pieces *en prise*.

3) Both sides have exactly one piece *en prise* and the piece of the side not to move is trapped or pinned, while the piece of the side to move is not.

The reasoning behind the first two of these conditions is that while the side to move could undoubtedly save a piece that was simply *en prise* he might not be able to save two pieces, both *en prise*, or one if it is trapped or pinned or if the side to move is also constrained to escape a check. Thus the side to move is forced to try his plausible moves and give the opponent an opportunity to try to capture the *en prise* material.

The reasoning behind the third condition is that the side to move may be able to save his piece instead of capturing the opponent's piece. Then the opponent will try to save his piece, which he may not be able to do since it is trapped or pinned.

The width of the search

Like the depth, the number of moves considered at each level is a constant tempered by some heuristics. The constants (a different one for each level) are usually all 6 for normal play, and are increased to 15, 15, 9, 9, 7 for tournament play, which means that the basic width at the top two levels is 15, while the basic width at levels three and four is 9, and the width is 7 for all succeeding levels.

The heuristics involved all have the effect of extending the width beyond the basic setting, so the only way that the program can fail to consider the indicated number of moves is either that the requisite number of moves simply do not exist or the tree-pruning algorithm provides an exit from the current level.

The heuristics are:

1) All safe checks are investigated.

2) At the first or second level, all captures are investigated.

3) An attempt is made to investigate moves of a certain minimum number of distinct pieces. This minimum is either half the basic width or the number of pieces with safe moves, whichever is less. This heuristic covers the case where all the moves of a single piece are highly plausible (say the queen, because it's *en prise*) and the rest of the board is not looked at.

4) Moves which lead to mate against the side to move are ignored and not tallied against the basic width. This guarantees that when a principal variation shows a mate, that mate is forced.

Additional features

Two algorithms for speeding up the search and three heuristic components for improving the reliability of the search comprise this section. The algorithms do not affect the quality of the program's play.

The alpha-beta tree-pruning algorithm

The alpha-beta algorithm (sometimes misnomered heuristic) has been a standard component of every modern game playing program. It was apparently first used by Newell, Simon, and Shaw (3).

In the search as described above, a move is discarded if it leads to a value which is worse for the side to move than some already considered move. If we look, however, at two levels of the tree, say moves by white, followed by replies by black, we notice the following: As moves by black are being explored, the value which is going to be returned back up to the white level (black's best value so far) cannot be getting any better for white and may be getting worse and worse. If a white move has already been evaluated, it is possible to check a black move not only to see if it is worse for black than some alternative, but also to see if it is so good for black that white would never make the move leading to that choice for black, or in other words, whether the move is "too good" for black. If the move is "too good," it is useless to consider any more moves for black from that position and the white move leading to that position may be discarded immediately. Thus, only one refutation is required to a proposed move and once it is found further search may be discontinued. The probability of alpha-beta cutoffs is increased by the fact that moves are investigated in order of decreasing plausibility, and a move is refued if it is equally good as the best so far at the previous level.

Such a consideration leads to a tremendous speedup of the search, especially if what turns out to be the best move at each position is considered first. One of the attributes of the plausible move generator is that it usually assigns the highest plausibility score to the best move, so almost maximal advantages is gained. (Rough calculation shows that the workload of the search is reduced by a factor of about one hundred.)

The name "alpha-beta" is derived from the fact that in the classic implementation of the algorithm, two recursive variables are kept: alpha, the best value so far for white, and beta, the best value so far for black.

Hash coding

One obvious way to speed up the searching process is to avoid considering the same position twice (as could happen through a transposition of moves). To

this end, the program incorporates a hash table into which an entry is made for each position considered. The entry records not only the results of the search but also a measure of how deep the search was which yielded the value. If the position is reached again, and the search in progress will not penetrate any deeper than the stored entry, then the results are immediately obtaind from the hash table. Due to the tree pruning algorithm, it is not always known exactly what the value of a node is, but only that it is greater or less than a certain value. Provision is made for storing this information in the hash table. On retrieval, the value is compared with alpha or beta (the tree prune variables) and a determination is made if further investigation is needed. Presently, the program uses a hash table of 32,000 entries with two machine registers per entry. An additional bonus of the hash table feature is that it enables the program to detect draws by repetition conveniently.

Modifications to the value returned by the search

If two moves are found by the search to lead to the same static value, the move which has the higher plausibility score is preferred. However, in some situations, this move is not the most desirable one to make. In order to take such cases into account, two types of small modifications may be made to the value returned from lower levels in the process of move tree searching.

The first modification subtracts a few points if the current move being investigated was marked as being developmentally poor by the plausible move generator.

The second type of modification occurs only if the principal variation that is returned is two or more plies long. If so, and it is found that the same piece was moved two plies down as is being moved in the current move, various small amounts are subtracted, depending on whether the piece is moved back to the square it came from or took two moves to accomplish a translation possible in one legal move or the position occurs during the first eight moves of the game (moves which are almost always devoted to rapid development). This second type of modification was introduced to give the program some sense of tempo and to counter its early tendency to make senseless attacking moves that were easily forced back.

Secondary search

A feature called secondary search was recently introduced. This was done in an attempt to obtain improved search depth at low cost. By increasing the depth of the search one can prevent the program from walking into traps which would not be recognized with a search conducted up to the normal depth. Moreover, one can discourage the tendency of the program to make delaying moves which force inevitable losses to occur beyond its normal lookahead. A secondary search is employed when the normal search results in a new candidate for the best move at the top level. What is done is to move down the principal variation for that move as far as this variation was computed by the plausible move generator, and then to conduct an additional search. The depth of this search is usually limited to two plies, although capture and feedover conditions can increase this number. The value produced by the secondary search is then used in place of the value first found for the principal variation if it is worse for the side to move.

This feature seems to improve the program's evaluation of many moves even though it is somewhat probabilistic in nature, since it looks at only a small subset of the positions that may be reached if the particular top level move is made. It seems to cause greatest improvement at tournament width settings when the principal variation is more reliable.

Book openings

The program incorporates a table of opening positions and selected replies. This "book" was compiled by two M.I.T. students, Larry Kaufman, a chess master and the top rated U. S. Junior player, and Alan Baisley, a chess expert.

The lines in the book have been selected to suit the computer's "style." The book contains over 5000 moves; however, actual games rarely follow book for more than approximately 10 plies. The book aids most the computer in avoiding "book traps" when playing against experienced players.

SUMMARY

Tournament play

The computer enters the tournament under the same rules as a human contestant. Moves are transmitted from the tournament site directly into the PDP-6 by teletype. A human operator is at the tournament who observes the opponent's move, types it in using standard chess notation, receives the machine's reply, plays it on the board and operates the clock. Of the two hours allotted to the machine for making the first fifty moves, about 7 minutes are normally lost in these operations.

The machine never offers a draw, but if the opponent offers one, the operator types in "draw?". The machine replies either "accept" or "decline." If the machine becomes hopelessly lost, human operators resign for it.

Results

The program is estimated to have played in excess

of 300 games in over the board competition with human players. It has played 18 tournament games. We will quote several tournament games. These games were played under rules calling for a minimum of 50 moves in two hours or an average of 2.4 minutes per move. Actually, the program played most of its recent games at about twice that rate. A single plausible move generation takes about 80 milliseconds for a typical position. In the early tournaments the computer did not keep track of the time used for each move, although this information is included with the game from a later tournament. The time quoted is actual computer time and does not include the operator overhead. This later tournament also saw the introduction of the book opening feature, which is not present in any of the other games quoted. (To convert the times given to machine operations, multiply by the PDP-6's approximate speed of 200,000 operations per second.)

Computer Tournament Chess Games

Tournament 1 the Winter Amateur Tournament of the Massachusetts State Chess Association Jan 21-22 1967

First Tournament Game Played By a Computer
White—rating 2190 Black Mac Hack VI

1	P-KN3	P-K4
2	N-KB3	P-K5
3	N-Q4	B-QB4
4	N-QN3	B-QN3
5	B-KN2	N-KB3
6	P-QB4	P-Q3
7	N-QB3	B-K3
8	P-Q3	PXP
9	BXP	N-Q2
10	PXP	R-QN1
11	B-KN2	O-O
12	O-O	B-KN5
13	Q-QB2	R-K1
14	P-Q4	P-QB4
15	B-K3	PXP
16	NXP	N-K4
17	P-KR3	B-Q2
18	P-QN3	B-QB4
19	QR-Q1	Q-QB1
20	K-KR2	N-KN3
21	B-KN5	R-K4
22	BXN	PXB
23	N-K4	P-KB4
24	N-KB6ch	K-KN2
25	NXB	QXN
26	N-QB6	QR-K1
27	NXR	RXN
28	Q-QB3	P-KB3
29	R-Q3	R-K7
30	R-Q2	RXR
31	QXR	N-K4
32	R-Q1	Q-QB2
33	B-Q5	K-KN3
34	P-QN4	B-QN3
35	Q-QB2	N-QB3
36	B-K6	N-Q5
37	RXN	BXR
38	QXPch	K-KN2
39	Q-KN4	K-KR3
40	QXB	Q-K2
41	Q-R4ch	K-KN3
42	B-KB5	K-KN2
43	QXRPch	K-KB1
44	Q-QR8ch	K-KB2
45	Q-QR8	Q-QB2
46	Q-Q5	K-N2
47	K-N2	Q-K2
48	P-KR4	K-R3
49	P-N4	K-N2
50	P-R5	Q-K7
51	P-R6	K-KB1
52	P-R7	QXKBP
53	KXQ	K-K2
54	P-R8=Q	P-QR3
55	Q-K6 MATE	

First Non-Loss By Computer in Tournament Play
Game 3 Tournament 1
White—1410 Black—Mac Hack VI

1	P-K4	P-K4
2	N-KB3	N-QB3
3	B-B4	N-KB3
4	N-N5	P-Q4
5	PXP	N-QR4
6	B-N5ch	P-B3
7	PXP	PXP
8	Q-B3	O-Q4
9	QXQ	NXQ
10	B-K2	B-KB4
11	P-Q3	B-QN5ch
12	B-Q2	BXB
13	NXB	O-O
14	P-QR3	P-KB3
15	KN-B3	QR-QN1
16	P-QN4	N-QN2
17	O-O	N-QB6
18	KR-K1	NXB
19	RXN	N-Q3
20	N-K4	NXN
21	PXN	B-K3
22	R-Q1	B-QB5
23	R/K2-Q2	R-QN2

24	R-Q8	RXR
25	RXRch	K-B2
26	N-R4	N-KN4
27	N-B5	R-QB2
28	P-N4	K-KN3
29	R-Q6	B-K7
30	R-Q8	BXP
31	R-KN8ch	K-KR4
32	N-N7ch	K-KR3
33	N-B5ch	K-KR4

etc. and drawn by repetition

First Game Won by Computer in Tournament Competition, Game 3 Tournament 2, Massachusetts State Championship 1967

White Mac Hack VI Black—1510

1	P-K4	P-QB4
2	P-Q4	PXP
3	QXP	N-QB3
4	Q-Q3	N-B3
5	N-QB3	P-KN3
6	N-KB3	P-Q3
7	B-KB4	P-K4
8	B-KN3	P-QR3
9	O-O-O	P-QN4
10	P-QR4	B-R3ch
11	K-QN1	P-N5
12	QXP/Q6	B-Q2
13	B-KR4	B-N2
14	N-Q5	NXKP
15	N-QB7ch	QXN
16	QXQ	N-B4
17	Q-Q6	B-KB1
18	Q-Q5	R-B1
19	NXKP	B-K3
20	QXN!	RXQ
21	R-Q8 MATE	

A More Recent Game With Times For Computer Moves, Game 2, Tournament 3 Massachusetts Spring Amateur

White Mac Hac VI Computer Time in sec

Black Unrated

1	P-K4	BOOK	P-K4
2	N-KB3	BOOK	N-QB3
3	B-QN5	BOOK	P-QR3
4	BXN	BOOK	QPXB
5	O-O	BOOK	B-Q3
6	P-Q4	BOOK	B-KN5
7	PXP	BOOK	BXN
8	QXB	BOOK	BXP
9	P-QB3	BOOK	Q-R5
10	P-KN3	18.3	Q-K2
11	R-K1	44.9	P-KR 4
12	P-KR4	111.7	O-O-O
13	B-KN5	78.5	P-B3
14	B-KB4	74.5	P-KN4
15	BXB	45.3	QXB
16	PXP	41.5	PXP
17	Q-KB5ch	60.5	QXQ
18	PXQ	29.1	N-B3
19	P-QB4	33.8	P-R5
20	N-QB3	77.6	R-Q7
21	P-QN3	88.0	P-R6
22	N-K4	56.0	NXN
23	RXN	43.3	K-Q2
24	P-KB6	19.0	R-Q3
25	P-KB7	19.0	R-B3
26	R-Q1ch	25.8	R-Q3
27	RXRch	22.8	PXR
28	K-KR2	68.25	R-KB1
29	KXP	76.6	RXP
30	R-K2	22.6	P-N4
31	K-N4	30.5	R-N2
32	R-K4	28.3	P-Q4
33	PXQP	19.4	PXP
34	R-K5	23.2	K-Q3
35	RXNP	14.0	RXR
36	KXR	4.5	K-K4
37	P-KB4ch	6.0	K-K5
38	P-KB5	8.5	P-Q5
39	P-B6	4.9	P-Q6
40	P-KB7	5.1	P-Q7
41	P-B8=Q	12.1	P-Q8=Q
42	Q-QB5ch	27.7	K-K6
43	Q-K6	29.2	K-B7
44	QXRP	42.8	Q-Q4ch
45	K-B4	20.45	Q-Q5ch
46	K-B5	14.9	Q-Q4ch
47	K-KN4	21.6	Q-KB6ch
48	K-KR4	16.3	QXKNPch
49	K-KR5	4.7	Q-K4
50	K-R6	20.8	Q-R1

etc. finally drawn by repetition

ACKNOWLEDGMENT

Many thanks go to the people at Project MAC who have written various routines, assisted in debugging the program by playing it, and served as operators at tournaments.

REFERENCES

1 C E SHANNON
Programming a digital computer for playing chess
Philosphy Magazine vol 41 March 1950 pp 356-375

2 A M TURING
Faster than thought B V Bowder (Ed)
Putman, London 1953 pp 288-295

3 A NEWELL J C SHAW H SIMON
Chess playing programs and the problem of complexity
IBM Journal of Research and Development vol 2
October 1958 pp 320-335

4 G W BAYLOR H A SIMON
A chess mating combinations program
Proc. Spring Joint Computer Conference vol 28 April 1966 pp 431-447

5 A KOTOK
A chess playing program for the IBM 7090
Bachelor's Thesis, Department of Electrical Engineering MIT 1962

6 G M ADELSON-VELSKY V L ARLASAROV A G USKOV
Programme playing chess
Report on Symposium on Theory and Computing Methods in the Upper Mantle Problem

7 P SAMSON
MIDAS
Artificial Intelligence Project Memo 90 MIT October 1965

8 A BERNSTEIN M DE V ROBERTS T ARBUEKLE M A BELSKY
A chess playing program for the IBM-704 computer
Proc. 1958 Western Joint Computer Conference Los Angeles Calif pp 157-159

9 J KISTER P STEIN S ULAM W WALDEN M WELLS
Experiments in chess
Journal of the ACM vol 4 April 1957 pp 174-177