A CHESS PLAYING PROGRAM FOR THE IBM 7090 COMPUTER

by

Alan Kotok

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
BACHELOR OF SCIENCE

at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June, 1962

Signature of Author . . . . . .
Department of Electrical Engineering

Certified by . . . . . .
Thesis Supervisor

Accepted by . . . . . .
Chairman, Departmental Senior Thesis Committee

ABSTRACT

This paper covers the development of a chess playing program. The preliminary planning led to the decision to use a variable depth search, terminating at either an arbitrary maximum, or at a stable position. Two schemes of controlling material balance are discussed. Of major significance is the use of the "alpha-beta" heuristic, a method of pruning the tree of moves. This heuristic makes use of values obtained at previous branches in the tree to eliminate the necessity to search obviously worse branches later.

The program has played four long game fragments in which it played chess comparable to an amateur with about 100 games experience.

ACKNOWLEDGMENT

I wish to thank Michael Lieberman, Charles Niessen, and Robert Wagner, the current members of the MIT chess group, for their invaluable assistance in this project. I also wish to express my appreciation to Elwyn Berlekamp, B. F. Wells and Paul Abrahams, who were previously associated with this project.

Special thanks go to Prof. John McCarthy who has guided the chess program through good days and bad. I wish to acknowledge the cooperation of the MIT Computation Center for providing the computation facilities necessary for this project.

Lastly, I wish to thank Robert Saunders for his help with the programming, and Milton Garber and Robert Fiorenza for giving their time to play against the machine.

TABLE of CONTENTS

INTRODUCTION

This thesis describes a chess playing program for the IBM 7090 computer. Although chess programs have been previously written, none of these played what could be considered "good chess".

Before commencing work on our chess program, we studied the report published by Newell, Shaw and Simon covering previous attempts, such as the Los Alamos program, and Bernstein's program at IBM.

PRELIMINARY INVESTIGATION

The chess group, consisting of Messrs. Berlekamp, Niessen, Lieberman and Kotok, inherited routines for generating and making legal moves. With these as a basis, we decided to write a three move mate solving program for the purpose of familiarizing ourselves with the existing routines, and to come in contact with many of the problems we would later face in the actual general playing program. The three move mate program was completed in the spring of 1960. It was given problems from actual games, and successfully solved many of them. The three move mate program was written for the IBM 704, which was removed from the MIT Computation Center in the summer of 1960. Due to incompatibility with the incoming 709, the project was dropped at the end of the spring term of 1960.

In the fall of 1960 the chess group, without Mr. Berlekamp, began planning for the general chess program.

It was decided to retain the original McCarthy-Abrahams move routines, and to continue coding in FORTRAN and FAP. The program was to be a variable depth search with a "stable position" termination. An evaluation was to be made at the terminal points of the move tree. This evaluation would be a weighted sum of such criteria as material balance, center control, pawn structure, "tempo" advantage, and development.

Moves on each level were to be proposed by "plausible move generators" which would propose moves to fulfill various goals. As the tree was searched, a backing up process would take place, in which the move declared best at each level by the evaluation would have its value brought up to the next higher level.

This procedure, also called mini-max, leads to a "principal variation" which is that set of moves which the machine considers most likely to happen. The evaluation always assumes that a player will always make the best move available to him a given time.

It was, of course, recognized that any evaluation could not be perfect, since chess is a game in which the only way a position can be perfectly evaluated is to look to the end of the game, and see whether it leads to a win, draw, or loss. The only sound basis for an evaluation is that chess masters have, over the years, accumulated knowledge concerning the play of the game. For instance, a position in which a piece is "en prise" is considered

bad, while having rooks on open files is considered good, even though the rules do not state anything about such things.

Since none of the members of the chess group are more than amateurs, we consulted books by masters to find out how much better it is to control the center than to have a strong pawn structure. These books are amazingly elusive on such details. Although many tips were given concerning the play of the game, relative importance of various strategies was uncertain.

We therefore considered having the program play for a while, and adjust the weights of the evaluation criteria to optimize its position. Although such a scheme seemed desirable, it was decided not to include any "learning" in the program due to the unavailability of suitably large amounts of computer time.

ORGANIZATION OF THE CHESS PROGRAM

Work on the chess program itself began in the spring term of 1961. The program is written in subroutine form, using the Fortran Monitor System of linkage. Where possible, programs are written in FORTRAN, and where it becomes too clumsy, or inefficient, FAP is used.

The actual implementation of the above mentioned "plausible move generators" has never been accomplished. Instead, we have a program, called REPLYS, which scans the legal move table, updates, evaluates, and reverts each move and orders them according to a single ply evaluation. (A ply is a half-move, i.e. a move by only one side.) The number of moves actually chosen is a function of the current depth in the tree.

Evaluation functions were written for material balance, center control, and development, since we intended to concentrate our efforts on openings until the program was thoroughly debugged.

The coordinating routine written in the spring of 1961, called TREE, employed the above mentioned mini-max scheme. REPLYS was set to cut the search at a depth of eight plys, or whenever the situation was stable, whichever came first.

The program was tested late in the spring of 1961. The 709 took about 5 to 20 minutes per move, depending on the complexity of the situation. Although the machine did not do too badly, we noted that it was looking at many

irrelevant positions. We thefore attempted to find a method of pruning the move tree, without discarding good as well as bad moves.

Prof. McCarthy proposed a heuristic for this purpose, called "alpha-beta". It operates as follows: Alpha is a number representing the value of the best position which white can reach, using a pessimistic evaluation. Beta represents the best position white can reach, using an optimistic evaluation, due to the fact that black can hold him to this position. Under normal circumstances, alpha starts at -infinity, and beta at +infinity.
At each level, optimistic and pessimistic evaluations are made, and compared to alpha and beta in the following way. If a white move is optimistically less than alpha, it is discarded, since a better alternative exists elsewhere. Likewise, if a white move pessimistically is better than beta, it too is discarded, since black had a better alternative previously; furthermore we revert two levels since no other white moves are worth considering at that position. The reverse strategy is applied for black.

The "alpha-beta" version of TREE was written during the summer of 1961, and was first put to use during the fall of that year. Also, we were joined by Mr. Wagner in the fall term of 1961.

After testing in the fall of 1961, it was decided that the material balance programs were insufficient. We therefore decided to replace the scheme then in use with

a new, updated scheme. The programs then in use, and, as it happens, in use now, completely re-generate the material balance function at each position.

The material balance evaluator consists of two subroutines, SWAP and LTRADE. SWAP's function is to list all attacks and defences on each occupied square. Secondary attackers which reside behind primary attackers (or defenders) are included. The pieces are listed in the order in which they would be played. Lowest valued pieces come first, unless the order is disturbed by the necessity of a higher valued piece to move first due to position. Pieces pinned to the king and queen were not recognized, leading to embarrassing evaluations. Likewise, discovered attacks were not considered.

LTRADE then simulates trade-off of all attacked pieces, and chooses the line most profitable for the side to move. The opponent is given the option of having a given piece taken, or moving the piece away. After all possible trades have been made, the program computes whether it is to the advantage of the machine to initiate an exchange, and if so, what the probable gain would be.

This scheme is both time consuming, and occasionally inaccurate. It was therefore decided to write a new evaluator for the material balance, which kept an updated set of tables, in a list structure format, from which the outcome of a given exchange could be found at a glance.

After a few months of planning and programming, the new list structure program was found to be impractical, due to excessive complication in the update procedure. Furthermore, the values which were to be included in the list were found to be no more accurate than the ones which the above scheme produced. The project was therefore abandoned.

DESCRIPTION OF COMPONENT SUB-PROGRAMS

The chess program is organized into a non-recursive hierarchy of sub-programs. Listings are to be found in appendix 1.

ADMINISTRATIVE ROUTINES

(MAIN) This is the highest level program. The on-line main program has the job of handling input-output, and timing. It determines the opponent's move by looking at the console keys, and picks the appropriate move from the legal move table. It then calls TREE which actually makes the move, after which (MAIN) prints out the machine's reply.

TREE Tree is the second level of control. Tree has the responsibility of constructing the tree of legal moves. It calls REPLYS to generate a list of plausible moves, and enters these in the LISP table, which is the actual tree. The moves are then chosen in order of decreasing value, and

updated. A new list of plausible moves is then generated for the opponent. The optimistic and pessimistic evaluators are called, and the alpha-beta tests are made, as described above. In the event that no replies are generated, due to stability, or excessive depth, a static evaluation is made and assigned to the position. The last move is then reverted, and the search proceeds down the next most likely branch of the tree. When all desired positions have been examined, the "best" move is returned as the answer.

PLAUSIBLE MOVE GENERATION

REPLYS This program supplies lists of plausible moves to TREE. It updates each of the legal moves, evaluates the position and reverts. The number of moves presented is a function of the present ply. Current values in order of increasing ply are: 4 3 2 2 1 1 1 1 0 0. These are input parameters to the program.

EVALUATION ROUTINES

EVAL Eval is the static evaluation program. Its function is to call all the subsidiary evaluation programs and to apply suitable multipliers, and hence form a weighted sum. Material values are: pawn 1, knight and bishop 3, rook 5, queen 9, and king 1000. These values are normally multiplied by 60 when combined with the other functions. Should one side be ahead at least 4 points, the material multipliers are adjusted to make trading

off advantageous.

LTRADE This program, described in more detail above, provides the projected material gain, considering all attacks and defenses.

ICENTR The center control evaluator gives points for controlling the 16 center squares. Looking from either side, these values are:

8	8	4	4
4	8	8	4
2	4	4	2
1	1	1	1

The center control points are each worth 1/60 of a pawn. After the game passes the twentieth full move, the center control function is decreased in importance until the 30th move, when it is discarded.

IDVLOP The development function, gives points for each developed piece. These range from 1 point per pawn, to 3 or 4 points for other pieces. Development points are weighted 1/15 of material points. This function is also eliminated as the game progresses.

JPAWNS The pawn structure function, considers the following situations, with aproximate point values:

open file	+8

isolated pawn	-1
backward pawn	-5
doubled pawn	-3
passed pawn	+10

These points are weighted 1/20 of material points.

SERVICE ROUTINES

UPDATE Updates any legal move, and records all relevant information on a push-down list. It then generates all legal replies available to the other side, using the general purpose move routines UPREV and PUTCH.

REVERT Takes back the last updated move. This is actually another option of the the updating routine UPREV.

PUTCH A lower level routine used in making moves. It keeps tables of almost legal moves and piece bearings updated. This table does not include castling, and "en passant" moves.

SWAP Generates the list of all attacks and defenses on occupied squares, listed in the order in which the pieces would be played.

PINS Generates the list of all pieces pinned to Kings and Queens. Includes the pinning direction, so that SWAP will only consider a pinned piece as an attacker or defend-

er along the line of the pin.

INPUT-OUTPUT ROUTINES

PRINT The major output routine. It handles most of the printing, both on and off line. It, and its subroutines, print the chess board, legal move table, principal variation, move tree and log of all moves tried, plus other information useful in debugging.

INITIA Reads in any chess board position. Its input language is as follows:

The chess board is scanned, from left to right, starting at white's Queen Rook 1. Digits represent numbers of unoccupied squares. Pieces are represented by the normal chess notation, in its most explicit form; e.g. KBP for King Bishop Pawn. Black pieces are preceded by asterisks. After exactly 64 squares are specified, the character"." (period) signifies the end of the specification and that white is to move. "*." indicates black to move. Additional features include the ability to indicate promoted pawns, by stating the type of piece, followed by the name of the pawn from which it promoted, in parentheses; e.g. Q(KNP). Also, it is possible to indicate that a piece has previously moved (for rooks, kings and pawns) by suffixing (M) to the piece name. Comments must begin and end with slashes.

The input is on IBM cards, punched in columns 1

through 72, taking as many cards as necessary. In case of errors found by INITIA, a comment will be printed, the remaining part of the problem will be skipped, and the next problem will be used.

All tables are initialized, and the program is set to commence with the legal move table generated for the side indicated. An example of an INITIA input will be found in Appendix 2.

RESULTS

As of this date, the machine has not completed any chess games. We have, however, played 4 lenghty fragments of games, and also have investigated many individual positions.

For our first long machine run, we chose an undergraduate student, Milton Garber, who held second place in his dormitory chess tournament. A record of this, and other game fragments is to be found in Appendix 3.

The second game was also played against Mr. Garber. In the record of this game a column indicating the principal variation is included. These are the moves the machine considers most likely to happen in succeeding plays, based on the evaluation and minimax process.

In seventeen moves, the machine guessed correctly only thrice, including only one case where it predicted correctly more than one move ahead.

Figure 1 consists of a set of representative

FIGURE 1

Representative output

```
SET OF TABLES NUMBER  4  MOVE IS *QP - Q4
                  BLACK
                  -----
*****************************************
* QR * QN * QB *  Q *  K * KB * KN * KR *
* ---* ---* ---* ---* ---* ---* ---* ---*
*****************************************
* QRP* QNP* QBP*    * KP * KBP* KNP* KRP*
* ---* ---* ---*    * ---* ---* ---* ---*
*****************************************
*    *    *    *    *    *    *    *    *
*    *    *    *    *    *    *    *    *
*****************************************
*    *    *    * QP *    *    *    *    *
*    *    *    * ---*    *    *    *    *
*****************************************
*    *    *    * QP *    *    *    *    *
*    *    *    *    *    *    *    *    *
*****************************************
*    *    *    *    *    *    *    *    *
*    *    *    *    *    *    *    *    *
*****************************************
* QRP* QNP* QBP*    * KP * KBP* KNP* KRP*
*    *    *    *    *    *    *    *    *
*****************************************
* QR * QN * QB *  Q *  K * KB * KN * KR *
*    *    *    *    *    *    *    *    *
*****************************************
                  WHITE
MAVAIL
```

K - Q2	QRP-QR3	QRP-QR4	QNP-QN3	QNP-QN4	QBP-QB3	QBP-QB4	KP - K3	KP - K4	KBP-KB3
KBP-KB4	KNP-KN3	KNP-KN4	KRP-KR3	KRP-KR4	QN - Q2	QN -QB3	QN -QR3	KN -KR3	KN -KB3
QB - Q2	QB - K3	QB -KB4	QB -KN5	QB -KR6	Q - Q2	Q - Q3			

```
PRINCIPAL VARIATION
VALUE=     27 EFFORT=    1449
*QP - Q4     KN -KB3
```

Figure 1 (cont)

THE MOVE TREE

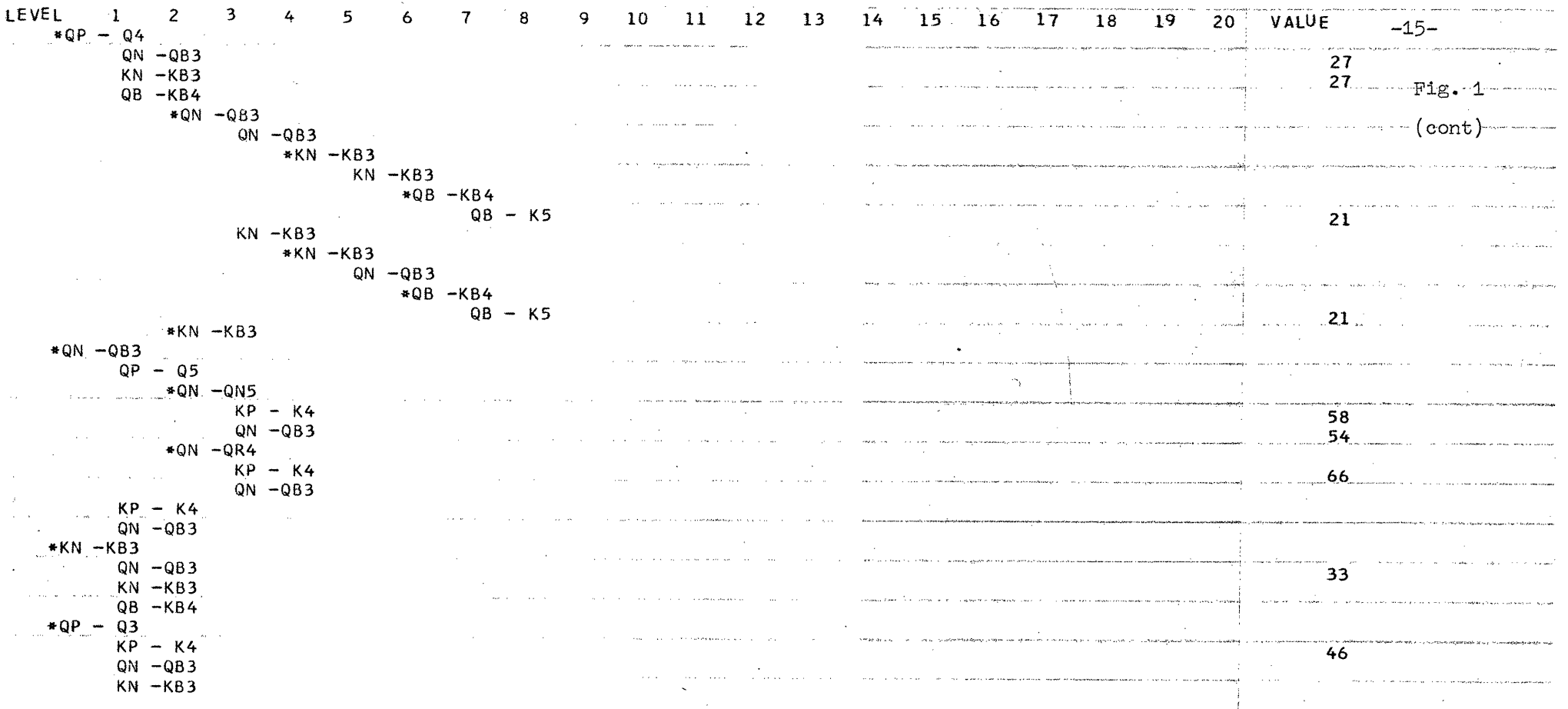

Fig. 1

(cont)

output for a single move. The first page is a printout of the chess board, and a list of the opponents legal replies, labeled MAVAIL. The second page contains the principal variation, beginning with the value of this variation, and the number of positions examined at the approximate rate of 1100 positions per minute. The principal variation itself commences with the machine's move.

The following pages contain the actual move tree. The moves listed therein are moves which were considered plausible by the reply generator. Moves were considered in the order top to bottom, however all moves on level one were generated simultaneously, and all level two replies to each level one move are generated together, etc. The "value" column contains a value on each terminating position. Values of $\pm$131071 indicate positions discarded for alpha-beta cutoff. Terminating positions which have no values have not even been examined, since the alpha-beta heuristic found previous moves on that level to be either too good, or too bad.

A third game fragment was played against an amateur with little chess experience; in particular, he knew the game, and had played some before he came to MIT. The game progressed 34 moves before time expired, with the result that the machine was ahead 1 rook, 2 knights and 2 bishops.

From our analysis of the results, we have found that in its present state, the program is comparable to

an amatuer with about 100 games experience.

Most of the machine's moves are neither brilliant nor stupid. It must be admitted that it occasionally blunders. These blunders can often be traced to wrong multiplyers in the evaluation, and occasionally to situations where discovered attacks, forks, etc. cause confusion. It is rare, however, not to find the correct move in the list of plausible moves.

This study is far from complete, but we feel that our efforts are proving fruitful. Hopefully this work will be continued.