Greenblatt Program Info

```
ALG=0 INPUT
ALLT DEBUG
AWKS=-1 TOURN
BCS=1 PARAM
BD OUTPUT
BOOK=1 PARAM
CASSW=0 PARAM
CBOOK INPUT
CLEAR INPUT
CLKKLU=-1 TIME
CSQSW=1 ?
DBDIS=1 DISPLAY
DFDSW=0 PARAM
DRAW MOVE
EGSW=1 PARAM
FANCY=-2 DISPLAY
FASTER PARAM
FCSW=1 PARAM
FILE OUTPUT
HASH=1 PARAM
HBPT DEBUG
HOPEN DEBUG
HPV OUTPUT
HRCSW=1 INTERN
HREAD DEBUG
HSSW=1 DEBUG
ICLOCK=0 TIME
ICSW=-2 DEBUG
KINGSA=1 PARAM
LIST OUTPUT
M MOVE
MVAN=1 ?
MVNSW=1 ?
PARCSW=1 ?
PARSW=0 TOURN
PB CONTROL
PCGSW=2 PARAM
PG OUTPUT
PMGDB=0 DEBUG
PMSASW=1 PARAM
PN CONTROL
PNT=0 OUTPUT
PS CONTROL
PSVD=0 DISPLAY
PUT INPUT
PW CONTROL
RBOOK INPUT
RBOOKB INPUT
RENDG INPUT
REPLAY DEBUG
RESET INPUT
RHDSK DEBUG
RPOS INPUT
RS INPUT
SASW=0 PARAM
SAVE OUTPUT
SDS=0 DEBUG
SE=1 TOURN
SETC=2 PARAM
SETCL TIME
```

```
SETD=4 PARAM
SETEST DEBUG
SETF=2 PARAM
SETFD=2 PARAM
SETP=6 6 3 3 2 PARAM
SETPVD=6 PARAM
SETSD=2 PARAM
SETSSD=0 PARAM
SETW=6 PARAM
SFS=1 PARAM
SIDE INPUT
SKIP INPUT
SLASW=0 PARAM
SLOWER PARAM
SLS=0 DEBUG
SMSS=1 PARAM
SMVD=0 DEBUG
SPARM TOURN-PARAM
SPOT INPUT
SPVS=0 DEBUG
SQCTSW PARAM
SSSW=1 PARAM
STOP TIME
STOPIN ?
SVHDSK DEBUG
T DEBUG
TP DEBUG
TPC DEBUG
TPDV DEBUG
TRACE DEBUG
TSW=1 OUTPUT
TTIME TIME
TWOGTS=0 TOURN-CONTROL
U MOVE
UREAD INPUT
UWRITE OUTPUT
WALLP=-1 OUTPUT
WBOOKB OUTPUT
WHDSK DEBUG
WPOS OUTPUT

        ?
CSQSW=1
MVAN=1
MVNSW=1
PARCSW=1
STOPIN
```

CONTROL

π Quit. Returns control to the top level and does a PN.

¬ Same as M, but needs no delimiter.

∃ Same as U, but needs no delimiter.

M This command causes the program to make a move.

U This command takes back the last move. It includes a PN (play neither) to prevent the program from trying to make a move it just took back. Therefore, after using this instruction it is necessary to feed in another PW or PB if you wish the program to continue playing.

PW Play White. The program makes moves for white, awaits input when black is to move.

PB Play Black. The program makes moves for black, awaits input when white is to move.

PS Play Self. The program moves for both sides.
PN Play Neither. The program awaits input.
TWOGTS=0 If positive, Think While Other Guy Thinks.
While in this mode, commands must be preceded by a colon.
Before a move has been typed in,
a rubout will cause all buffered commands to be aborted so that
a move can immediately be typed in. If the move is terminated by
"DRAW?" instead of a carriage return, the program will eventually
type its move followed by "ACCEPT" or "DECLINE". If the move
is terminated by ":", the program will eventually wait for a command
before continuing to TWOGT.
After a move is typed in (the input format for normal moves is descriptive:
piece/square-square or piece/square*piece/square,
but anything redundant may be omitted), the move will be typed
back in full format followed by a question mark. If it is correct,
type a period to have it inputted. Any other character will cause
the program to request a move again. Illegal moves will be so
commented. Ambiguous moves will cause the program
to make its best guess of what you meant, and will type
this move in full format together
with the warning "AMBIGUOUS". Special moves:
O-O or O O for king-side castling;
O-O-O or O O O for queen-side castling;
PxP EP or PxG etc. for en passant capture;
P-K8=Q etc. for promotion.

DEBUG
ALLT Traces all plausible moves.
HBPT Hash BreakPoinT. Sets a breakpoint which breaks
if current position is reached in search.
HOPEN Takes file specification as argument. Opens file for HREAD.
HREAD Searches forward in HOPEN'ed file for hash table
relevant to current position and reads it in.
HSSW=1 Hash Save SWitch.
-1 Opens file CHASHT > for writing. Save hash table in this file after each move
0 Don't save hash table.
1 Save hash table in core after each move by program.
ICSW=-2 Incremental Cat (calculate attack) SWitch.
-2 Spot check.
-1 Always check.
0 Don't use IC.
1 Use IC.
PMGDB=0 If positive, display pins, threats and constraints
after move typed in.
REPLAY Takes file specification as argument. Types "PLAY WHITE?" Answer
"Y" or "N". Then asks "PLAY BLACK?" Again answer
"Y" or "N". If not playing side to move, program reads a move
from the file. If playing side to move, program makes a move to
compare with move from file, but uses move from file. Normally
this feature is used with WALLP=-1.
RHDSK Takes file specification as argument. Reads hash table from
specified file, saving it in core.
SDS=0 Set Display Switch.)
SLS=0 Set Look Switch.)
0,0 Normal mode of operation
1,0 Give PMG display for current position when given a
make move command (no move is made).
1,1 Do a normal search but make display and pause at each node.
(Type a space to continue to the next node
or type an integer to continue to the next node at that ply.)

2,0 Do a normal search and display feedover conditions.
(Type a space to continue.)

SETEST Takes numeric argument. Sets estimate of value of position.

SMVD=0 If positive, give analysis of principal variations stemming from each plausible move.

SPVS=0 If positive, print static evaluator and development values after move typed in.

SVHDSK Takes file specification as argument. Saves hash table as the specified file.

T Trace. With no argument, traces static board evaluator evaluation of current position. With a move for argument, traces PMG evaluation of that move.

TP Trace Positional.

TPC Types plausible captures and their values.

TPDV Types development values.

TRACE Same as "T".

WHDSK Takes file specification as argument. Writes hash table saved in core as the specified file.

^ Enter DDT.

DISPLAY

DBDIS=1

FANCY=-2

-2	No display
-1	Whose move
0	Board position in characters
1	Board
2	Board with game so far
3	Display on Knight console

ϵ Does FANCY -2, but at interrupt level.

← Turns off display until input is requested by program. (Done at interrupt level.)

PSVD=0 If positive, display principal variation and its value after making a move.

INPUT

ALG=0

1	Sets preferred input to algebraic notation. Output is also algebraic.
0	Sets preferred input to descriptive notation. Output is also descriptive.
-1	Output is in both notations.

CBOOK Continues reading book after an error during an RBOOK.

CLEAR Removes all pieces from board.

PUT piece square Puts specified piece (same format as is outputted by BD) on specified square (same format as is outputted in a move by the program, but always with respect to white).

RBOOK Reads book of openings. (See description of book format.)

RBOOKB Takes file specification as argument. Reads a file outputted by the WBOOKB command.

RENDG Reads endgame program. (See CHEG description.)

RESET Restores initial position. Does a PN. RESET n restores to move n, e.g. RESET 5 backs up the game to the point at which white is about to make his 6th move.

RPOS Takes next eight lines to be a board as outputted by the BD command, and sets up the specified position.

RS Takes file specification as argument. Reads in commands from file. (Note that moves are commands, so this command can be used to restore SAVE'd games.)

SIDE=W Side to move.

SKIP Takes numeric argument. Skips to the next occurrence of "[" followed by its argument in the file being read.

SPOT Gives handicap, e.g. SPOT QVBR removes white's queen and black's queen bishop and queen rook, also moving black's queen rook pawn to R3. In general, "P", "N", "B", "R", and "Q" removes the KBP, the QN, the QB, the QR, and the Q respectively for the appropiate side (the side is initially white, and is switched by "V"); in addition, removal of the QR moves the QRP to R3.

UREAD Takes file specification as argument. Selects auxiliary device input.

⊃ Initiates read-in from auxiliary device.

INTERN

HRCSW=1

OUTPUT

⊗ Turns on teletype output. (Done at interrupt level.)

↔ Turns off teletype output. (Done at interrupt level.)

ALG=0 See under input.

DRAW Asks if machine wants a draw.

BD Prints out the board.

PG Prints out the game.

FILE Closes output file.

HPV Prints out principal variation from current position as gotten from hash table.

LIST Lists program commands and parameters with their current values.

PNT=0 1 Output to printer.
0 Output to TTY.
-1 Output to secondary output device.

SAVE Takes file specification as argument. Writes file consisting of the moves made, similar to the output of a PG command.

TSW=1 If positive, type lines after typing out move.

UWRITE Takes file specification as argument. Opens auxiliary output file.

WALLP=-1 1 Send debugging information (wallpaper) to printer.
0 Don't output wallpaper.
-1 Send wallpaper to file WALLP >.

WBOOKB Takes file specification as argument. Writes binary file specifying the book of openings. This file can be read in by the RBOOKB command.

WPOS Takes file specification as argument. Writes file consisting of "RPOS", followed by output of BD, followed by "SIDE W" or "SIDE B" depending on whether white or black is to move.

PARAM

BCS=1 If positive, use board control option in static evaluator.

BOOK=1 If positive, use opening book.

CASSW=0

DFDSW=0 Delayed FeeDover SWitch.

EGSW=1 End Game SWitch. If positive, use endgame program.

FASTER Goes to next lower (faster) parameter set.

FCSW=1 Forward Cutoff SWitch. If positive, use forward cutoff heurisitic.

HASH=1 -1 Don't use hash feature.
0 Use hash for draw detection only.
1 Use hash for detecting duplication of search. Obtain value from table instead of continuing search.

KINGSA=1 KING SAfety. If positive, use king safety term in static board evaluator.

PCGSW=2 Additional depth (after SETD) for PMG in PCG mode.

PMSASW=1 Poor Man's Surprise Analysis SWitch. (# of ply)

SASW=0 Surprise Analysis SWitch.

SETC=2 Number of ply to look at all captures.

SETD=2 Basic search depth for PMG.

```
SETF=2          Maximum number of ply to look for feedovers.
SETFD=2         Maximum number of extra ply for feedovers
        (only those due to possibility of value being worse for
        side to move are allowed).
SETP=6 6 3 3 2  Maximum number of positional (only) moves to look at
        at each ply.  [See SETW.]  [Not implemented.]
SETPVD=6        Using new PCG, depth to look at good moves.
SETSD=2         Set Secondary Search Depth.
SETSSD=0        Set Secondary Search Starting Depth.
SETW=6          Basic search width.  Takes a list of numbers separated by spaces,
        applying to plies 1,2,3,...; the last number applies to all higher plies.
SFS=1           Set Feedover Stop.  If positive,
        stop feedover in case a feedover is obviously not called for (i.e.
        in resulting position one side has lost a piece and is not
        threatening anything of equal value).
SLASW=0         Set Line Analysis SWitch.
SLOWER          Goes to next higher (slower) parameter set.
SMSS=1          Set Multiple Search Switch.
SQCTSW
SSSW=1          Search Strategy SWitch.

        TIME
CLKKLU=-1       -1      Run time according to side to move.
                0       Don't run time.
                1       Run time according to clock kludge.
ICLOCK=0        1 inverts sense of clock kludge.
SETCL   "B" or "W" followed by time to set clock.
STOP    Stops clock.
TTIME   Types out times used by white and black.

        TOURN
AWKS=-1 If non-negative, send all TTY output to T!AWKS (e.g. T27 if AWKS=23)
        as well as to TTY.
PARSW=0 If positive, change parameters automatically.
SPARM   Takes numeric argument between 0 and 6.
        Stores current parameters as specified parameter set.
SE=1    Sets an echo to cut off local people
        if tournament console being used.
```

Section 0 Introduction

The Greenblatt Chess Program is currently the best chess-playing program in existence. It was written beginning in 1966 and has undergone extensive modification since the 1967 paper describing it [FJCC]. Its current USCF rating is 15xx, but it is in fact much stronger than this would indicate.

We shall describe the workings of the program in some detail, but first we shall give a brief glossary of chess terminology and a basic description of standard game-playing techniques.

Section 1 Chess terminology

A ∃move is a change in the configuration of the pieces on the board. A move is ∃legal if it conforms to chess rules. When no ambiguity can arise, we use "move" to denote "legal move". A move is ∃possible if it would be legal but for the prohibition of moving into or through check. If a piece disappears during a possible move, the move is called a capture, and that piece is said to be captured. If only one piece changes position in a possible move, that piece is said to have made the move. If in addition a piece is captured, the moving piece is said to have captured that piece, or less specifically, to have made a capture.

A position is a triple consisting of the side to move, the configuration of the pieces on the board (assuming all pieces that look the same, e.g. all white pawns, are indistinguishable), and the castling privileges of the two sides (4 bits: WK, WQ, BK, BQ; where XY=1 iff neither the X king nor the XY rook have moved or been captured). Note that castling privileges as thus defined do not indicate that castling is legal in the position, but merely that it might be in some subsequent position. Chess rules state that if a position as thus defined is repeated three times (and so declared by one of the players), the game is a draw. [This is a mild lie, but will probably eventually be the truth.]

A piece is ∃constrained if it is unable to make certain possible moves without immediate loss of material or game.

A piece is ∃pinned if some of its possible moves expose mating attacks or attac on friendly pieces.

A piece is ∃trapped if its immediate capture is unavoidable.

A piece ∃defends another (friendly) piece if it would have a possible capture of an enemy piece if that piece captured the friendly piece.

Pawn structure

A pawn is ∃isolated if there are no friendly pawns in adjacent files.

A pawn is ∃backward if it is not defended by a pawn but there are friendly pawns in adjacent files which could defend it were it advanced far eno but it cannot be defended by a pawn in one move (even allowing movement through friendly pieces and movement of other pawns through enemy pieces) except possibly by its own double advance exposing it to en passant capture.

A pawn is ∃doubled if there is a friendly pawn in the same file.

A pawn is ∃tripled if there are two friendly pawns in the same file.

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

A piece is ∃en ∃prise if it is subject to capture without adequate material compensation.

Section 2 Game-playing

We consider a game played by two persons (called ∃white and ∃black).

Starting at an initial state (∃position), the players alternately make plays (∃moves); first white and then black. Thus the sequence of moves would be move 1 for white, move 1 for black, move 2 for white, move 2 for black, etc. We will call a single move by a single player a ∃ply, so that two ply will constitute a move for each side. Thus after 3 ply, black would be about to make his second move.

Eventually, the sequence of moves terminates in a ∃final ∃position, one for which the outcome of the game is determined by the rules. Note that it may be the entire sequence of moves that determines the outcome. Such a sequence of moves is called a ∃complete ∃game.

We can think of a complete game as a path in the ∃game ∃tree. Each node in the game tree represents a position, and each branch emanating from that node a possible move in that position. The root node would then be the initial position, and leaf nodes would be final positions. Note that the same position may appear many times in the game tree.

Since all complete games are finite, if there are only finitely many possible moves in each position, then, by the fan lemma, the entire game tree is finite. We can assign a value to each node, starting at the leaf nodes and working back toward the root as follows:

Assign to each leaf node +1 if it is a winning position for white
0 if it is a drawn position
-1 if it is a winning position for black.

[If we wished to be more discriminating, we could assign a real number to each leaf node, with (algebraically) larger values being better for white.] To each node all of whose children have been evaluated, assign the best of the children's values (the max if white to move, the min if black to move). This is called ∃mimimax∞ing. Inductively all nodes can be evaluated. The value of any node is then +1 if white can force a win, 0 if neither side can force a win, and -1 if black can force a win.

We can describe an algorithm, called ∃depth-first ∃search, for evaluating a given node. If the node is a leaf node, evaluate it using the game's rules. Otherwise, order the node's children from left to right, and evaluate each of them in turn by applying the algorithm recursively. Return the best value so gotten. The path from the node found by following down the leftmost best moves is called the ∃principal ∃variation from that node. Note that the algorithm can be easily modified to also return the sequence of moves of the principal variation. Of course, all the nodes on the principal variation have the same value as the top node, since the principal variation is a path of best moves. Thus we define the value of the principal variation to be the value of its top node.

In order to evaluate a single node, it is not really necessary to evaluate all its descendants, as depth-first search does. If we know t the side to move, say white, can achieve a certain value, say alpha, then we need not evaluate completely every black node below it: for any descendant node at which black can achieve less than alpha cannot be relevant to the final value, because white would not have allowed black to be in such a position. Similarly, if black can achieve beta at some node, then any descendant node at which white can achieve more than beta is irrelevant. We can modify the depth-first search algorithm to maintain alpha and beta, which are the best that black might do and the best that white might do at the current node or one of its ancestors. Then if a white node is found to have a value greater than or equal to beta, we need not evaluate any more of its descendents and can simply return that v Similarly, if a black node is found to have a value less than or equal to alpha, we can immediately return that value. This is called the ∃alpha-beta ∃algorithm. (alpha,beta) may be initialized to (-inf,+inf), or may be set to some smaller range initially. In the latter case, if the actual value of the node lies outside the initial interval, the value returned for the n

will also lie outside the interval but may not be the correct value.
The alpha-beta algorithm will investigate
fewer nodes than depth-first search, and it can be seen that the closer the left-to-right
sequence of moves is to best-to-worst, the fewer nodes will be investigated.
ϵ[This is not completely true, since it may take fewer node investigations
to evaluate a move from a given node which is not best but good enough to have a value
outside (alpha,beta) on the good side. Also, in many cases the
ordering of moves from a given node is irrelevant since all of their values
lie outside (alpha,beta) on the bad side.]
The alpha-beta algorithm may easily be modified to produce the principal variation,
but it will only do so for nodes which have been completely evaluated. Thus if the value retur
for a node lies outside the initial (alpha,beta) for that node, then the principal variation
returned may be incorrect. In particular, the first move of the principal variation
will not necessarily be the best move.

An obvious improvement to any node-evaluation algorithm is to use information
already discovered to avoid reevaluating an already evaluated (or partially evaluated)
position. To do this, we store a table containing for each position investigated the
results of the investigation (and a flag to indicate if investigation
is in progress). For the alpha-beta
algorithm this information includes the range of possible values for the position
(either (-inf,x] where x .le. alpha, [x,inf) where x .ge. beta,
possibly [x,y] for a position partially evaluated more than once,
or [x,x] for a completely evaluated position) and the best investigated move from the position.
[The principal variation may then be reconstructed recursively by playing the
best move and looking up the resulting position in the table.] If a position
is to be evaluated it is first looked up in the table. If it is found, the information
in the table is used: If investigation is in progress, a 0 (draw) value is returned.
ϵ[This assumes that a repeating position is a draw (as in chess).
Other game rules might require different action in this case.]
If the table contains a precise value for the position, that value is returned.
If the range of values for the position is outside the current (alpha,beta), that
value range is returned. Otherwise, the position is evaluated normally, except that
alpha and/or beta may be changed for the evaluation to reflect
the range information obtained from the table; and the new range stored
in the table can be the intersection of the old range with the returned range.
Note that what is returned is not really a value range but a single value
which is either considered to be exact if within (alpha,beta)
or to represent the ray emanating from that value which does not
intersect (alpha,beta). Thus if a range [x,y] were supposed to be
returned, only the endpoint closest to (alpha,beta) would be returned.

Another improvement to a node-evaluation algorithm can be made if it can
be determined beforehand that certain moves from a given position cannot be the best
move. These moves need never be investigated. This technique is known as Ǝforward
Ǝcutoff.

Up to now we have been assuming that we could proceed to the leaf nodes to
evaluate a position. For a game such as chess this is clearly impractical. What
we instead do is perform a partial depth-first search.
Given a position to evaluate, we list all moves from that position, making
an estimate (called the Ǝplausibility) of how good each is. We order these moves according to
this estimate from best to worst. This procedure is called Ǝplausible Ǝmove
Ǝgeneration. We then investigate the best of these moves by applying the
algorithm recursively to the resulting positions, and return the best value so gotten.
During the recursion, we keep track
of our depth of search (i.e. the number of ply from the original position)
in order to help decide which deeper moves to investigate. When we reach a position
from which we wish to do no further investigation, we estimate its value by
Ǝstatic Ǝevaluation; and the value so estimated is returned.

Our previous discussion on precise node-evaluation algorithms applies to this inexact algorithm as well. ¢{There is one exception, which is that the table of evaluated positions should store information on the certainty of its value, so that an unsure value will not be used when a more certain value is required. This might otherwise happen if a position evaluated during a search occurs again later in the search but at a node nearer the root node of the search.}

The inexact algorithm just described has not been completely specified. There remain the problems of computing the plausibility of moves and the static value of positions, as well as of deciding which moves to investigate. The algorithms used for these determine the nature of the game player. For the Greenblatt chess program, these algorithms are described in the next section.

One technique which can be applied to the inexact algorithm in order to improve its evaluation is called secondary search. Whenever a move at the top level is determined to be a candidate for the best move (by virtue of its receiving the best value so far) it is reassessed by following its principal variation to the end and then performing an additional short depth-first search from that node. The value returned is then used to replace the original value of the top-level move if it is worse for the moving side. This technique tends to prevent the algorithm from misevaluating moves which delay an inevitable worsening of the static value.

Section 3 The Program
- A. Flow of Control
- B. Major Sections
  - PMG
  - SBE, PCG
  - book
  - CHEG
- C. Debugging Features
- D. Tournament Features

Acknowledgements: RWG, AGB

References: Greenblatt et al, Knuth