An Upgrade for KIM MICROCHESS 1.0  
Garold R. Stone  
P.O. Box 153  
Annapolis Junction, MD. 20701

If you have Peter Jennings' MICROCHESS program for the KIM-1 microcomputer you can teach it to play a significantly better game of chess without adding a single byte of expansion memory. This article describes a "patch." I have written for MICROCHESS which gives the computer a more flexible opening game and two new strategies for the middle and end game. Just load your copy of MICROCHESS, enter my code from the accompanying program listing along with the chess opening sample from table one, and play chess. There are no changes in the way you run the program. (For a description of the MICROCHESS program see KB, August 1978, page 74). For clarity I will use the term MICROCHESS only to refer to the original program as written by Peter Jennings. I will say "patch" to refer to the changes I am describing here.

Off the Shelf

The MICROCHESS I bought from Micro-ware Ltd. opens the game by playing from a pre-selected list of moves for a user chosen chess opening (Roy Lopez, French Defence, etc.). That opening list also contains one anticipated opponent move for each computer move. Things go well as long as the opponent makes the anticipated replies. But a human opponent seldom does that -- at least I don't. As soon as I make a novel move MICROCHESS permanently abandons the opening list. Whenever MICROCHESS is forced to quit the opening list too early, coherent development of pieces stops, the queen usually comes out too early, an ill-prepared attack is launched, and the computer loses its ability to castle (because castling is only possible from the opening list).

Compromises in 1.1K

Mr. Jennings points to these problems in his excellent documentation manual:

"A major problem in the analysis is that there is only one strategy which is used for the opening, the middle game and the end game. This involves a considerable compromise of three different types of play."

The single strategy used by MICROCHESS is best suited for the middle game, where the capture of pieces dominates. In order to add a dynamic opening strategy which would emphasize the development and positioning of pieces, I had to settle for my own set of compromises, as you'll see. I should point out that Mr. Jennings seems to have surmounted this problem in the other versions of MICROCHESS he has written for microcomputers with more memory, such as the PET, TRS-80, and the APPLE.

The Opening

Table 1 shows my data format for eight opening development moves. Unlike in MICROCHESS, anticipated opponent replies are not listed. On each turn the patched program evaluates all of the computer's available moves. The available move which comes out with the highest evaluation is compared with the evaluation for the next legal move in my opening list and the higher of the two is selected as the computer's move for that turn. The development move is usually selected because its evaluation is always boosted by a threshold factor. I set the threshold factor high enough so that only moves with a significantly higher evaluation can override the development move. The higher the threshold, the more likely it is that the development move will be selected for that turn. Thus, the computer follows an opening game plan, responds to significant attack threats or capture opportunities, and then continues to carry out the opening game plan on the next turn by consulting the opening list again.

Books on chess openings and opening game strategy can serve as guides in writing new lists of development moves. Choose openings which are general in nature and do not depend on specific moves by the opponent. Specify each development move by giving the piece (variable DEVP), the square of origin (FROM), and the destination (TO), using the same notation as in MICROCHESS (see tables 2 and 3). Openings for white and black will require separate notation. Fill all unused locations in the opening list with the magic number 1F (hexadecimal), which causes those locations to be skipped because they are off the board.

Castling

As in MICROCHESS the computer's castling move must be completed for it by moving its rook after the computer signals castling by moving its King the necessary two squares. My added programming will prevent castling if the computer's King is off its starting square or if it would end up in check. The other rules for castling are not checked, however. If the computer castles illegally, then the move must be refereed. The simplest way is to use the "touch-move" rule -- once a player touches a piece it
must be moved. Thus, the computer would have to move it. If there are no legal moves left for the King, then the computer must resign.

This situation seldom comes up because I write openings which castle early enough to avoid the risk and annoyance of an illegal attempt.

Program Flow

What follows is a description of how the patched program works. MICROCHESS command loop CHESS calls my version of subroutine GO (see 03A2 in the program listing). MICROCHESS uses the value of a variable called STATE to keep track of what’s doing. State 4 guides the generation and evaluation of the computer’s available moves. There are other states for generating potential opponent replies, etc.

MICROCHESS subroutine GNMX (see 03AA) initializes some variables called “counts” for evaluating moves and then generates all moves available to the computer on that turn. GNMX calls MICROCHESS subroutine JANUS to calculate and evaluate the counts for each trial move. Based on the value in STATE, JANUS decides what to do next -- generate potential opponent replies for evaluation, calculate exchanges of pieces, etc.

Table 1

<table>
<thead>
<tr>
<th>ADDR</th>
<th>VARIABLE</th>
<th>MOVE</th>
<th>WHITE</th>
<th>BLACK</th>
<th>COMMENT</th>
<th>THRESHOLD</th>
</tr>
</thead>
<tbody>
<tr>
<td>00C3</td>
<td>.FACTOR</td>
<td>03</td>
<td>05</td>
<td></td>
<td></td>
<td>FACTOR</td>
</tr>
<tr>
<td>00C6</td>
<td>.TO</td>
<td>22</td>
<td>26</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00C7</td>
<td>.PN</td>
<td>0A</td>
<td></td>
<td>0A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00C8</td>
<td>.FROM</td>
<td>11</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00C9</td>
<td>.TO</td>
<td>21</td>
<td>26</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00CA</td>
<td>.PN</td>
<td>04</td>
<td>04</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00CB</td>
<td>.FROM</td>
<td>02</td>
<td>05</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00CC</td>
<td>.TO</td>
<td>11</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00CD</td>
<td>.PN</td>
<td>0F</td>
<td>0F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00CE</td>
<td>.FROM</td>
<td>13</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00CF</td>
<td>.TO</td>
<td>23</td>
<td>24</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00D0</td>
<td>.PN</td>
<td>00</td>
<td>00</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

See Tables 2 and 3 for coding of Pieces and Squares

Table 2

<table>
<thead>
<tr>
<th>CODE</th>
<th>PIECE</th>
<th>MEMORY LOCATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>KING</td>
<td>0000</td>
</tr>
<tr>
<td>01</td>
<td>QUEEN</td>
<td>0001</td>
</tr>
<tr>
<td>02</td>
<td>ROOK</td>
<td>0002</td>
</tr>
<tr>
<td>03</td>
<td>KNIGHT</td>
<td>0003</td>
</tr>
<tr>
<td>04</td>
<td>BISHOP</td>
<td>0004</td>
</tr>
<tr>
<td>05</td>
<td>KNIGHT</td>
<td>0005</td>
</tr>
<tr>
<td>06</td>
<td>PAWN</td>
<td>0006</td>
</tr>
<tr>
<td>07</td>
<td>QUEEN</td>
<td>0007</td>
</tr>
<tr>
<td>08</td>
<td>ROOK</td>
<td>0008</td>
</tr>
<tr>
<td>09</td>
<td>KNIGHT</td>
<td>0009</td>
</tr>
<tr>
<td>0A</td>
<td>PAWN</td>
<td>000A</td>
</tr>
<tr>
<td>0B</td>
<td>PAWN</td>
<td>000B</td>
</tr>
<tr>
<td>0C</td>
<td>PAWN</td>
<td>000C</td>
</tr>
<tr>
<td>0D</td>
<td>PAWN</td>
<td>000D</td>
</tr>
<tr>
<td>0E</td>
<td>PAWN</td>
<td>000E</td>
</tr>
<tr>
<td>0F</td>
<td>PAWN</td>
<td>000F</td>
</tr>
</tbody>
</table>

Table 3

<table>
<thead>
<tr>
<th>Board Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer</td>
</tr>
<tr>
<td>00 01 02 03 04 05 06 07</td>
</tr>
<tr>
<td>10 11 12 13 14 15 16 17</td>
</tr>
<tr>
<td>20 21 22 23 24 25 26 27</td>
</tr>
<tr>
<td>30 31 32 33 34 35 36 37</td>
</tr>
<tr>
<td>40 41 42 43 44 45 46 47</td>
</tr>
<tr>
<td>50 51 52 53 54 55 56 57</td>
</tr>
<tr>
<td>60 61 62 63 64 65 66 67</td>
</tr>
<tr>
<td>70 71 72 73 74 75 76 77</td>
</tr>
</tbody>
</table>

OPPONENT

Note: Whether playing White or Black, the computer’s starting squares are always 00 through 17. Be sure to orient the playing board so that the lower left corner is black. The White Queen should be on a white square and the Black Queen should be on a black square.

Table 4

<table>
<thead>
<tr>
<th>ADDR</th>
<th>VARIABLE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>00C2</td>
<td>.FACTOR</td>
<td>Threshold factor for opening moves</td>
</tr>
<tr>
<td>00DC</td>
<td>.GMOVE</td>
<td>MICROCHESS opening move flag</td>
</tr>
<tr>
<td>00EF</td>
<td>.BKNM8</td>
<td>Number of legal moves for Opponent King</td>
</tr>
<tr>
<td>00F0</td>
<td>.BP8</td>
<td>Receiver threshold factor for legal list move</td>
</tr>
</tbody>
</table>

JANUS and portions of GNMX call each other recursively, again and again, until all of the computer’s available moves have been evaluated in the light of all possible opponent replies. By the time program control returns from that very first call to subroutine GNMX, one move has emerged with an evaluation higher than all the others.

Then my patch searches the opening move list from the beginning to find the first piece (variable DEVPI) which is still where it is supposed to be (FROM) (see 03B1). The move by this piece to its destination (TO) is checked for legality by a call into the middle of MICROCHESS subroutine GMOVE.
If the list move is legal, then the threshold factor (FACTOR) is stored in the variable BIAS for later use (see 03D8). MICROCHESS subroutine JANUS is called to do the counts for this list move and for the opponent’s potential replies.

To evaluate these counts JANUS calls up my version of subroutine STRATEGY (see 1780-17C1). This is where the evaluation of the list move is boosted by adding the threshold factor which was stored earlier in the variable BIAS. Actually, this same subroutine STRATEGY is used by JANUS to evaluate any trial move but BIAS is always zero except for legal list moves. If the selected list move is not legal, then JANUS is not called to evaluate it, and no more list moves will be tried for that turn. This ensures that moves from the opening list are made in the order you wrote them. After the last list move has actually been moved, the variable OMOVE is set to zero and the opening list is ignored for the rest of the game (see 03AF).

As you exit subroutine STRATEGY you enter that portion of MICROCHESS which compares the evaluation of the current trial move with that of the best move so far, saving the better of the two as the new best move so far. This is also where MICROCHESS tests for check or checkmate before returning to JANUS. Control then passes to the MICROCHESS subroutine which takes the best trial move and actually moves it (see 03E3). The computer’s move is flashed on the KIM display and the program returns to the MICROCHESS command loop, ready for the opponent to enter his move.

Middle and End Game

MICROCHESS sees only one and a half moves ahead. With this limited horizon it has trouble finding and closing in on the opposing King. To compensate for this I give a bonus of two points for moves inside a zone which surrounds the opposing King and moves along with it. The computer’s Pawns and King do not get the bonus (see 179D).

Another strategy encourages moves which hem in the opposing King, in preparation for checkmate. The value of any trial move is increased by the number of safe moves it leaves for the opposing King. This is the same as adding a point for each square denied to the opposing King. Since MICROCHESS calls subroutine JANUS to evaluate only legal moves, it was easy enough to put a subroutine call inside JANUS which would increment a mobility count (BKMOV) for each legal move found for the opponent King when the computer is checking for opponent replies moves during state zero (see 0112, 17D9, 179A).

Both strategies come into play only after the opening list has been emptied, so as not to interfere with the development of pieces during the opening game (see 1796).

Evaluation

I approached move evaluation in much the same way as in MICROCHESS -- adding and subtracting weighted counts representing captures, position, and mobility for both sides. I did not use some of the counts generated by MICROCHESS and I created the new ones I described above. Given the severe memory restrictions, my goal was an evaluation formula which emphasizes immediate and tangible factors, such as position and the values of pieces captureable during the current turn. Less immediate factors, such as overall attack strengths, are given fractional weighting. These become influential only after more significant factors have cancelled each other out.

For now I’ve had to be satisfied with just breaking MICROCHESS of its habit of throwing away its pieces by occasionally making bad decisions about captures where pieces are exchanged. In my patch any piece the computer wants to capture must be greater than or equal to the most valuable piece the computer would lose by making that move (variable BMAXG). Only trial moves which pass this admittedly simplistic test are given an extra 20 hex points (see 17B1). There is more that could be done, like making better use of the MICROCHESS counts for exchanges involving up to three captures per side.

I hope I’ve made my point. All you need is a shoe horn and you can slip just about any changes you want into the 1.1K KIM MICROCHESS. You may pinch a few toes in the process, but the result is a KIM that plays better chess. By trying to “upgrade” MICROCHESS I really learned to appreciate what an excellent piece of work it is.

MICROCHESS is available in KIM cassette with documentation manual from Micro-Ware Ltd., 456 Albert St., Suite 7, Waterloo, Ontario, Canada, N2L 3V4

Abbreviated Instructions for Loading and Running MICROCHESS 1.0 UPGRADE

Load:
Enter (RS) to reset KIM
Enter (AD) 081 (DA) 00 to reset decimal flag
Enter (AD) 17F9 (DA) C1 to enter tape 1D for program segment
Enter (AD) 1873 (GO) to start read routine of KIM
Press “Play” on cassette player
STOP recorder when display shows: 0000
Enter (RS) (AD) 1873 (GO) to read second program segment (same label “C1”)
STOP recorder when display shows: 0000
Enter (RS) (GO) to start program execution

Playing:
Enter (C) on KIM keypad keyboard to reset program for new game
Enter (PC) (for “play chess”) because KIM plays first
After KIM gives its move, enter your move as
FROM:TO according to the board notations in table 3 of the article. Keep typing until your move shows correctly, then enter (F) (PC).
compute ii.

03A2- A2 04 0120 GO LDX $504 ; RESET BEST EVALUATION
03A4- B8 FA 0130 LDY #$F4 ; SO PAR
03A6- 6F B5 0140 STY *STATE ; STATE = 4; TRAIL MOVES
03A8- A2 12 0150 STX *BESTV ; ZERO COUNTERS & BIAS
03AA- 28 02 02 0160 JSR GMNX ; GENERATE TRAIL MOVES
03AD- A4 DC 0170 LDY *GMOVE ; OPENING LIST DONE?
03AF- 10 32 0180 BPL NODEVP ; - YES, MID-GAME
03B1- A0 E6 0190 LDY #$56 ; - NO, NEXT DEVP
03B3- C8 0200 NEXT INY
03B4- C8 0210 INY ; INDEX OF DEVP
03B5- 84 DC 0220 STY *OMOVE ; OPENING LIST EMPTY?
03B7- 10 2A 0230 BPL NODEVP ; - YES, MID-GAME
03B9- B6 DC 0240 LDX *DEVP,Y ; - NO, NEXT DEVP
03BD- 86 B0 0250 STX *PIECE
03BE- 55 50 0260 LDA *BOARD,X ; DEVP LOCATION
03BF- C8 0270 INY ; INDEX OF FROM
03C0- 48 0280 PHA ; (SAVE DEVP LOCATION)
03C1- 90 0290 TYA ; TRANSFER INDEX OF
03C2- AA 0300 TAX ; FROM INTO X
03C3- 68 0310 PLA ; DEVP LOCATION IN ACCUM
03C4- D5 DC 0320 CMP *FROM,X ; DEVP AT ORIGIN?
03C6- DB EB 0330 BNE NEXT ; - NO, GET NEW DEVP
03C8- EB 0340 INX ; INDEX OF TO
03C9- 65 DC 0350 LDA *TO,X ; CHECK LEGALITY OF DEVP
03CB- 28 D1 02 0360 JSR CMP ; MOVE FROM .PROM TO .TO
03CE- 30 13 0370 BMI NODEVP ; NEQ = ILLEGAL MOVE
03D0- A6 B0 0380 LDX *PIECE ; - LEGAL MOVE
03D2- B0 08 0390 CPX #$88 ; IS PIECE A PAWN
03D4- 30 02 0400 BMI LEGAL ; NEG = NOT PAWN
03D6- 7B 0B 0410 BVS NODEVP ; SET = ILLEGAL PAWN CAPTURE
03DA- A6 C3 0420 LEGAL LDX *FACTOR ; LEGAL OPENING MOVE!!
03DB- 35 0430 STX *BIAS ; SET BIAS TO FACTOR
03DC- A2 04 0440 LDX #$B4 ; EVALUATE OPENING MOVE
03DE- B9 B5 0450 STX *STATE ; AND PUT IT IN BESTV
03E0- 28 00 01 0460 JSR JANUS ; IF ITS THE BEST MOVE
03E3- A6 FA 0470 NODEVP LDX *BESTV ; SO PAR
03E5- 8F 0F 0480 CPX #$8F ; RESIGN OR STALEMATE IF
03E7- 4C C2 17 0490 JMP CONT ; BESTV TOO LOW
0500 ;
0510 ;
17C2- 90 12 0520 CONT BCC MATE ; (ORIGINAL MICROCHESS
17C4- A6 FB 0530 MV2 LDX *BESTP ; CODING)
17C6- B5 50 0540 LDA *BOARD,X ; MOVE AND DISPLAY THE
17C8- 85 FA 0550 STA *BESTV ; BEST MOVE
17CA- 86 B0 0560 STX *PIECE
17CC- A5 F9 0570 LDA *BESTM
17CE- 85 B1 0580 STA *SQUARE
17D0- 2B 4B 03 0590 JSR MOVE
17D3- 4C 00 00 0600 JMP CHESS ; END COMPUTER'S TURN
17D4- A9 F0 0610 MATE LDA @$FF ; RESIGN OR
17DB- 60 0620 RTS
0630 ;
0640 ;
1780- A9 80 0650 STRATEGY LDX #$80 ; EVALUATION = 80 + OR - SCORE
1782- 18 0660 CLC
1783- 65 EB 0670 ADC *WMOB ; COMPUTERS'S MOBILITY
1785- 4A 0680 LSR A
1787- 10 0690 CLC
1787- 89 40 0700 ADC #$40 ; RESET EVAL TO 80 + OR- SCORE
1789- 65 ED 0710 ADC *WCC ; COMPUTER'S ATTACK STRENGTH