Home * Engines * **Chess 0.5**

BYTE, Vol. 3, No. 10 [6]

Chess 0.5,
a program by Larry Atkin and Peter W. Frey for didactic purposes, written in the ETH Zurich dialect of Pascal for the CDC 6600 series of mainframes, published 1978/1979 in Byte Magazine, and re-published on-line in 2005, available from Scott A. Moore's [1] sites, unfortunately with some OCR-errors such as syntactical correct replacement of '*' by '+' with score factor by side (XTMV, ±1) in some places [2] . Chess 0.5 may be compiled with GNU Pascal for recent hardware. Due to non-local goto statements over procedure or function boundaries compiling with Free Pascal is not possible [3] .

Larry Atkin is co-author of the famous Northwestern University Chess line of programs. At the time of the article, Chess was at about version 4.6 [4] . Chess 0.5 is based on his intimate knowledge of that program, but is a seperate program designed for pedagogical purposes to play chess vaguely well [5] using Chess 4.x like structures of a Shannon Type A approach with alpha-beta and quiescence search, bitboards and incremental updated attack tables.

Table of Contents

Blueprint

As encouraged by Frey and Atkin in their article "The reader with an interest in chess and programming should use this listing as starting point for developing a program" [7] , Chess 0.5 was origin of at least three other competing programs, the Austrian Merlin by Hermann Kaindl, Helmut Horacek, Marcus Wagner and Roland Schreier, further developed in Pascal except some time critical, often called routines, which were re-written in CDC assembly by Wagner [8] [9] , the Dutch Chess 0.5X by Wim Elsenaar, a PDP-11 assembly port, and according to postings in rgcc [10] , the Finnish Shy by Juha Kasanen, Mika Korhonen and Timo Saari, written in Algol.

Description

Board Representation

Chess 0.5 keeps an 8x8 board either indexed by square or rank and file, ...

```
CONST
  AX = 0;       ZX = 31;                (* SUBSETS OF SQUARES *)
  AS = 0;       ZS = 63;                (* BOARD VECTOR LIMITS *)
TYPE
  TF = (F1,F2,F3,F4,F5,F6,F7,F8);       (* FILES *)
  TR = (R1,R2,R3,R4,R5,R6,R7,R8);       (* RANKS *)
  TI = INTEGER;                         (* NUMBERS *)
  TS = AS..ZS;                          (* SQUARES *)
  TX = AX..ZX;                          (* SOME SQUARES *)
  TY = AY..ZY;                          (* NUMBER OF TX'S IN A BOARD *)
  TM = (LITE,DARK,NONE);                (* SIDES *)
  TP = (LP,LR,LN,LB,LQ,LK,DP,DR,DN,DB,DQ,DK,MT);
(* PIECES: LIGHT PAWN,... , DARK KING, EMPTY SQUARE *)
  SX = SET OF TX;                       (* SET OF SOME SQUARES *)

  RB = RECORD                           (*   BOARDS   *)
    RBTM : TM;                          (*  SIDE  TO MOVE   *)
    RBTS : TT;                          (*  ENPASSANT  SQUARE    *)
    RBTI : TI;                          (*  MOVE   NUMBER    *)
    RBSQ : SQ;                          (*  CASTLE  FLAGS    *)
    CASE INTEGER OF
      0: ( RBIS  : ARRAY [TS] OF TP);   (* INDEXED BY SQUARE *)
      1: ( RBIRF : ARRAY [TR,TF] OF TP);(* INDEXED BY RANK AND FILE *)
    END;
```

... and further relies on bitboards, defined as union of arrays of sets and integers:

```
  RS = RECORD                           (* BIT BOARDS *)
    CASE INTEGER OF
      0: (RSSS: ARRAY [TY] OF SX);      (* ARRAY OF SETS *)
      1: (RSTI: ARRAY [TY] OF TI);      (* ARRAY OF INTEGERS *)
    END;
```

Beside the mailbox arrays (BOARD, MBORD) and bitboard board definition (TPLOC, TMLOC), Chess 0.5 maintains incremental updated attack tables, two 8x8 arrays ATKTO and ATKFR, the first for every square a attack bitboard **to** other squares of the piece (if any) residing on that square, the second for each square a bitboard of all white and black man attacking this square.

```
  BOARD : RB;                           (* THE BOARD *)
  MBORD : ARRAY [TS] OF TP;             (* LOOK-AHEAD BOARD *)
  ATKFR : ARRAY [TS] OF RS;             (* ATTACKS FROM A SQUARE *)
  ATKTO : ARRAY [TS] OF RS;             (* ATTACKS TO A SQUARE *)
```

```
  ALATK : ARRAY [TM] OF RS;             (* ATTACKS BY EACH COLOR *)
  TPLOC : ARRAY [TP] OF RS;
(* LOCATIONS OF PIECE BY TYPE *)
  TMLOC : ARRAY [TM] OF RS;
(* LOCATIONS OF PIECE BY COLOR *)
```

Bitboard Infrastructure

Setwise Operations

Bitboard intersection, union and relative complement are implemented with set-wise operations "*, +, -", complement (not) by relative complement from the universe ([AX..ZX]):

```
PROCEDURE ANDRS
(* INTERSECTION OF TWO BITBOARDS *)
  (VAR C:RS;                            (* RESULT *)
   A, B:RS);                            (* OPERANDS *)
VAR
  INTY : TY;                            (* BIT BOARD WORD INDEX *)
BEGIN
  FOR INTY := AY TO ZY DO
    C.RSSS[INTY] := A.RSSS[INTY] * B.RSSS[INTY];
END;  (* ANDRS *)

PROCEDURE IORRS                         (* UNION OF TWO BIT BOARDS *)
 (VAR C:RS;                             (* RESULT *)
  A, B:RS);                             (* OPERANDS *)
VAR
  INTY : TY;                            (* BIT BOARD WORD INDEX *)
BEGIN
  FOR   INTY :=  AY TO ZY DO
    C.RSSS[INTY] := A.RSSS[INTY] + B.RSSS[INTY];
END;  (* IORRS *)

PROCEDURE NOTRS                         (* COMPLEMENT OF A BIT BOARD *)
 (VAR C:RS;                             (* RESULT *)
  A:RS);                                (* OPERAND *)
VAR
  INTY : TY;                            (* BIT BOARD WORD INDEX *)
BEGIN
  FOR INTY := AY TO ZY DO
    C.RSSS[INTY] := [AX..ZX] - A.RSSS[INTY];
END;  (* NOTRS *)
```

BitScan

BitScan reverse with reset is implemented with machine dependent CDC 6600 float conversion (omitted here) - the machine independent code inefficiently loops over up to 2*32 bits of the set and cries for improvement:

```
FUNCTION NXTTS                          (* NEXT ELEMENT IN BIT BOARD *)
 (VAR A:RS;
(* BIT BOARD TO LOCATE FIRST SQUARE, AND THEN REMOVE *)
  VAR B:TS
(* SQUARE NUMBER OF FIRST SQUARE IN BIT BOARD *)
 ): TB;

(* TRUE IFF ANY SQUARES WERE SET INITIALLY *)
LABEL
  11;                                   (* RETURN *)
VAR
  INTX : TX;                            (* BIT BOARD BIT INDEX *)
  INTY : TY;                            (* BIT BOARD WORD INDEX *)
  X : RK;                               (* KLUDGE WORD *)
BEGIN
  FOR INTY := ZY DOWNTO AY DO           (* LOOP THRU BIT BOARD WORDS *)
    IF A.RSTI[INTY] <> 0 THEN
    BEGIN
      FOR INTX := ZX DOWNTO AX DO
(* LOOP THROUGH BITS IN WORD OF SET *)
        IF INTX IN  A.RSSS[INTY] THEN
        BEGIN
          B := INTX+INTY*(ZX+1);        (* RETURN SQUARE NUMBER *)
          A.RSSS[INTY] := A.RSSS[INTY] - [INTX];
(* REMOVE  BIT FROM WORD *)
          NXTTS := TRUE;                (* RETURN A BIT SET *)
          GOTO 11;                      (* RETURN *)
        END;
    END;
  NXTTS := FALSE;                       (* ELSE RETURN NC BITS SET *)
11:                                     (* RETURN *)
END;  (* NXTTS *)
```

Popcount

The machine independent population count relies on the above bitscan with reset!

```
FUNCTION CNTRS
(* COUNT NENBERS OF A BIT BOARD *)
(A:RS): TS;                              (* BIT BOARD TO COUNT *)
VAR
  INTS : TS;                             (* TEMPORARY *)
  INRS : RS;                             (* SCRATCH *)
  IMTS : TS;                             (* SCRATCH *)
BEGIN
  INTS := 0;
  CPYRS(INRS,A);
  WHILE NXTTS(INRS,IMTS) DO
    INTS := INTS+1;                      (* COUNT SQUARES *)
  CNTRS := INTS;                         (* RETURN SUM *)
END;  (* CNTRS *)
```

Evaluation

Chess 0.5's evaluation routine **EVALU8** implements a lazy evaluation only for too worse scores. If the incremental updated material balance plus the maximum positional score is still less or equal than alpha (best value two plies up BSTVL[JNTK-2]), only the material balance is assigned without further positional analysis. Otherwise **EVALU8** consides piece mobility (counting attacks), pawn structure, king safety and some rook terms such as doubled rooks and rook on 7th rank. Differences of light minus dark positional terms are adjusted by appropriate feature weights. To make white point of view scores relative to the side to move, they are multiplied by a score factor (+1, -1) indexed by side.

```
PROCEDURE EVALU8;                        (* EVALUATE CURRENT POSITION *)
  FUNCTION EVKING                        (* EVALUATE KING *)
  FUNCTION EVMOBL                        (* EVALUATE MOBILITY *)
  FUNCTION EVPAWN                        (* EVALUATE PAWNS *)
  FUNCTION EVROOK                        (* EVALUATE ROOKS *)
BEGIN
  IF XTMV[JNTM]*MBVAL[JNTK] + MAXPS <= BSTVL[JNTK-2] THEN
(* !!! OCR  *)
                                         (* MOVE WILL PRUNE ANYWAY *)
    INTV := XTMV[JNTM] * MBVAL[JNTK]
  ELSE
  BEGIN
    INTV := (  FWPAWN*(EVPAWN(TPLOC[LP],S2,R2)-EVPAWN(TPLOC[DP],S4,
R7))
             + FWMINM*(EVMOBL(LB,LN)          -EVMOBL(DB,DN)
          )
             + FWMAJM*(EVMOBL(LR,LQ)          -EVMOBL(DR,DQ)
```

```
          )
             + FWROOK*(EVROOK(TPLOC[LR],XRRS[R7])
                      -EVROOK(TPLOC[DR],XRRS[R2])
                     )
             + FWKING*(EVKING(TPLOC[LK],TPLOC[LP])
                      -EVKING(TPLOC[DK],TPLOC[DP])
                     )
           ) DIV 64;
    MAXPS := MAX(MAXPS,ABS(INTV));
    INTV := XTMV[JNTM] * (MBVAL[JNTK] + INTV);
  END;
  IF SWTR THEN
    BEGIN
      WRITE(" EVALU8",JNTK,JNTW,INDEX[JNTK],INTV);
      PRIMOV(MOVES[INDEX[JNTK]]);
    END;
  VALUE[INDEX[JNTK]] := INTV;          (* RETURN SCORE *)
END;  (* EVALU8 *)
```

Search

The search is implemented spaghetti like non-recursively as finite-state machine with goto labels using explicit stacks aka ply indexed arrays, and features iterative deepening with aspiration and alpha-beta pruning. The value array of best moves indexed by current ply is the current alpha of a newly entered node, initialized by grand parent's best value (ply-2) - best value of the parent node (ply-1) is minus beta in the negamax sense. The nested function SELECT implements a staged move generation, considering root search (ply 0), full-width and quiescence, generating captures in MVV/LVA order, killers, and remaining moves. While the ply indices range from 0 to 16, the best value array needs three additional sentinels due to ply-2, ply-1, and ply+1 accesses. The outline of the search routine with jump labels (:) is given below, most lines omitted:

```
  BSTVL : ARRAY [AKM2..ZKP1] OF TV;
(* VALUE OF BEST MOVE [-2..17] *)

FUNCTION SEARCH                        (* SEARCH LOOK-AHEAD TREE *)
: TW;                                  (* RETURNS THE BEST MOVE *)
  PROCEDURE NEWBST
(* SAVE BEST MOVE INFORMATION *)
    (A:TK);                            (* PLY OF BEST MOVE *)

  FUNCTION MINMAX                      (* PERFORM MINIMAX OPERATION *)
    (A:TK)                             (* PLY TO MINIMAX AT *)
```

```
     : TB;                              (* TRUE IF REFUTATION *)
    MINMAX := FALSE;                    (* DEFAULT IS NO PRUNING *)
    IF -BSTVL[A+1] > BSTVL[A] THEN      (* Score > Alpha ? *)
    BEGIN
      BSTVL[A] := -BSTVL[A+1];          (* Alpha = Score  *)
      NEWBST(A);                        (* SAVE BEST MOVE *)
      MINMAX := BSTVL[A+1] <= BSTVL[A-1];
(* -Score < -Beta => Score > Beta *)
    END;
  END;  (* MINMAX *)

  FUNCTION SELECT
(* SELECT NEXT MOVE TO SEARCH *)
     : TB;                              (* TRUE IF MOVE RETURNED *)
  BEGIN
  21:  (* NEW SEARCH MOOE *)
    CASE SRCHM[JNTK] OF
      H0:  (* INITIALIZE FOR NEW MOVE *)
      H1:  (* INITIALIZE AT  NEW DEPTH *)
      H2:  (* CAPTURE SEARCH *)
      H3:  (* FULL WIDTH SEARCH - CAPTURES *)
      H4:
(* INITIALIZE SCAN OF CASTLE MOVES AND OTHER MOVES BY KILLER PIECE *)
      H5:
(* FULL WIDTH SEARCH - CASTLES AND OTHER MOVES BY KILLER PIECE *)
      H6:  (* FULL WIDTH SEARCH - REMAINING MOVES *)
      H7:  (* RESEARCH FIRST PLY *)
  22:  (* SELECT EXIT *)
    SELECT := INTB;                     (* RETURN VALUE *)
  END;  (* SELECT *)

BEGIN  (* SEARCH *)
  EVALU8;                               (* INITIAL GUESS AT SCORE *)
  BSTVL[AK-2] :=   VALUE[AW] - WINDOW; (* INITIALIZE ALPHA-
BETA WINDON *)
  BSTVL[AK-1] := -(VALUE[AW] + WINDOW);
  JMTK := AK+1;
  WHILE (NODES < FNODEL) AND (JNTK < MAX(ZK DIV 2, ZK-8)) DO
  BEGIN
11:  (* START NEW PLY *)
12:  (* DIFFERENT  FIRST MOVE  *)
13:  (* FLOAT   VALUE   BACK  *)
      IF   MINMAX(JNTK)   THEN
        GOTO 15;                        (* PRUNE *)
14:  (* FIND ANOTHER MOVE AT THIS PLY *)
15:  (* BACK UP A PLY *)
```

```
16:  (* EXIT SEARCH *)
  SEARCH := BSTMV[AK];                    (* RETURN BEST MOVE *)
END;  (* SEARCH *)
```

See also

- Chess
- Chess 0.5X
- Chess 7.0
- CookieCat
- Merlin
- Shy

Publications

- Peter W. Frey, Larry Atkin (**1978**). Creating a Chess Player. An Essay on Human and Computer Chess Skill, BYTE, Vol. 3, No. 10, pp. 182-191. pdf from The Computer History Museum
- Peter W. Frey, Larry Atkin (**1978**). Creating a Chess Player, Part 2: Chess 0.5. BYTE, Vol. 3, No. 11
- Peter W. Frey, Larry Atkin (**1978**). Creating a Chess Player, Part 3: Chess 0.5 (continued). BYTE, Vol. 3, No. 12
- Peter W. Frey, Larry Atkin (**1979**). Creating a Chess-Player, Part 4: Thoughts on Strategy. In Blaise W. Liffick (ed.), The Byte Book of Pascal, pp. 143-155. Byte Publications, also BYTE, Vol. 4, No. 1

Forum Posts

- Byte Chess 0.5 finally available. From Byte Magazine 1978 by I Forget, rgcc, June 01, 2005
- Update for Byte Chess 0.5 by I Forget, comp.lang.pascal.misc, June 12, 2005
- Need a little help from a Pascal guru (Chess05 Frey/Atkin) by Jim Ablett, CCC, January 09, 2012

External Links

- Chess 0.5, Release 1 - 2005-05-30 by Scott A. Moore
  Byte Chess 0.5 source code hosted by Scott A. Moore
- Classic Computer Chess - ... The programs of yesteryear by Carey, hosted by the Internet Archive
- Computer-Schach by Andre Adrian (German)
  Chess05GNU.pas

References

1. ^ Scott A. Moore's site
2. ^ Byte Chess 0.5 source code (OCR-Errors!)
3. ^ Re: Need a little help from a Pascal guru (Chess05 Frey/Atkin) by Steven Edwards, CCC, January 09, 2012
4. ^ Chess 4.6 source code, gift of David Slate, from The Computer History Museum, pdf
5. ^ Chess 0.5, Release 1 - 2005-05-30
6. ^ Byte Vol. 3, No. 10, October 1978 from the Internet Archive
7. ^ Peter W. Frey, Larry Atkin (**1978**). Creating a Chess Player, Part 3: Chess 0.5 (continued). BYTE, Vol. 3, No. 12
8. ^ Werner DePauli-Schimanovich (**2006**). Europolis 6. Informatik für Spiele und Verkehr. Extension der Mengenlehre, Herausgeber: Franz Pichler, Universitätsverlag Rudolf Trauner, ISBN 978-3-85487-946-6, (SG7) Merlin (ein ComputerChess-Programm) s. 171 (German), Google Books
9. ^ Helmut Horacek, Marcus Wagner (**1981**). *Das Schachprogramm Merlin, Verbesserung von Laufzeit-Effizient, Eröffnungsbibliothek und Bewertungsfunktion.* 4. Tagung "Berichte aus Informatik-Instituten" (German)
10. ^ Re: Byte Chess 0.5 finally available. From Byte Magazine 1978 by I Forget, rgcc, June 02, 2005

What links here?

Page	Date Edited
BitScan	Sep 10, 2017
Byte Magazine	Nov 20, 2016
CDC 6600	Oct 8, 2014
Chess	Jan 21, 2018
Chess (Program)	Dec 22, 2017
Chess 0.5	Nov 20, 2016
Chess 0.5X	Nov 5, 2015
Chess 2013	Nov 6, 2013
Chess 7.0	Jul 13, 2015
CookieCat	Nov 15, 2016
Engines	Mar 10, 2018
Larry Atkin	Jan 7, 2016
Marcus Wagner	Jan 7, 2016
Merlin	Jan 20, 2018
Northwestern University	Sep 5, 2017
Pascal	Nov 28, 2016
Peter W. Frey	Dec 25, 2017
Shy	Jul 19, 2016

Up one Level