Computer Chess Compendium

To Alastair and Katherine

David Levy, Editor

# Computer Chess Compendium

*Springer Science+Business Media, LLC*

First published 1988
© David Levy 1988
Originally published by Springer-Verlag New York, Inc. in 1988
Softcover reprint of the hardcover 1st edition 1988

Typeset by
W. Turner & Son, Ltd
Halifax

ISBN 978-1-4757-1970-3 ISBN 978-1-4757-1968-0 (eBook)
DOI 10.1007/978-1-4757-1968-0

Library of Congress Cataloging-in-Publications Data

Levy, David N.L.
Computer Chess Compendium

Bibliography:p.
Includes index.
I. Chess--Data processing I. Title.
GV 1449.3.L47 1988 794-1'7 88-10423
ISBN 978-1-4757-1970-3

# Contents


| | | |
|---|---|---|
| | **Preface** | vii |
| **1** | **Introduction** | 1 |
| 1.1 | Programming a Computer for Playing Chess – C. E. Shannon | 2 |
| 1.2 | Chess – A. Turing | 14 |
| 1.3 | The Chess Machine: An Example of Dealing with a Complex Task by Adaptation – A. Newell | 18 |
| **2** | **Famous Chess Programs** | 28 |
| 2.1 | Chess Playing Programs and the Problem of Complexity (excerpt) – A. Newell, J. C. Shaw and H. A. Simon | 29 |
| 2.2 | Computer v Chess Player – A. Bernstein and M. de V. Roberts | 43 |
| 2.3 | A Chess Playing Program for the IBM 7090 – A. Kotok | 48 |
| 2.4 | The Greenblatt Chess Program – R. D. Greenblatt, D. E. Eastlake IIIrd and S. D. Crocker | 56 |
| 2.5 | The Technology Chess Program – J. J. Gillogly | 67 |
| 2.6 | CHESS 4.5 – The Northwestern University Chess Program – L. Atkin and D. Slate | 80 |
| 2.7 | Cray Blitz – Robert A. Hyatt, Albert E. Gower and Harry L. Nelson | 104 |
| **3** | **Position Evaluation** | 112 |
| 3.1 | Statistics for the Chess Computer and the Factor of Mobility – E. T. O. Slater | 113 |
| 3.2 | A Five-Year Plan for Automatic Chess (excerpt) – I. J. Good | 118 |
| **4** | **Tree Searching Techniques** | 122 |
| 4.1 | Tree-Searching and Tree-Pruning Techniques – J. A. Birmingham and P. Kent | 123 |
| 4.2 | Some Methods of Controlling the Tree Search in Chess Programs – G. M. Adelson-Velskiy, V. L. Arlazarov and M. V. Donskoy | 129 |
| 4.3 | The Heuristic Search and the Game of Chess. A Study of Quiescence, Sacrifices and Plan Oriented Play – L. R. Harris | 136 |
| 4.4 | A Theory of Evaluative Comments in Chess with a Note on Minimaxing – D. Michie | 143 |
| **5** | **Analysis** | 156 |
| 5.1 | The Sequence of Phases – A. D. de Groot | 157 |
| 5.2 | Skill in Chess – H. A. Simon and W. G. Chase | 175 |

| | | |
|---|---|---|
| 5.3 | Decision Making and Computers – M. M. Botvinnik | 189 |
| 5.4 | A Chess Mating Combinations Program – G. W. Baylor and H. A. Simon | 196 |
| 5.5 | Robot Chess – D. G. Prinz | 213 |
| **6** | **Writing a Chess Program** | 220 |
| 6.1 | A Computer Chess Tutorial – N. D. Whaland | 221 |
| 6.2 | Using Patterns and Plans in Chess – D. Wilkins | 233 |
| 6.3 | Mate at a Glance – J. Birmingham and P. Kent | 258 |
| **7** | **Special Purpose Software and Hardware** | 266 |
| 7.1 | Some Ideas for a Chess Compiler – M. R. B. Clarke | 267 |
| 7.2 | Robots – H. Vigneron | 273 |
| 7.3 | CHEOPS: A Chess-orientated Processing System – J. Moussouris, J. Holloway and R. D. Greenblatt | 279 |
| 7.4 | BELLE: Chess Hardware – J. H. Condon and K. Thompson | 286 |
| **8** | **The Endgame** | 293 |
| 8.1 | Co-Ordinate Squares: A Solution to Many Chess Pawn Endgames – K. W. Church | 295 |
| 8.2 | Goal-Directed Search in Chess Endgames – H. J. van den Herik | 316 |
| 8.3 | Computer Analysis of a Rook End-Game V. L. Arlazarov and A. L. Futer | 330 |
| **9** | **Games Played by Chess Programs** | 337 |
| 9.1 | Games from the ACM Tournaments | 338 |
| 9.2 | Games from the World Computer Championships | 382 |
| 9.3 | Blitz Games between Computers and Human Players | 408 |
| 9.4 | Simultaneous Games between Computer Programs and Human Players | 410 |
| 9.5 | Games Played by Computer Programs in Human Tournaments | 411 |
| 9.6 | Miscellaneous Games played by Computer Programs | 418 |
| 9.7 | The World Microcomputer Championships | 419 |
| | **Bibliography** | 421 |

# Preface

For many years I have been interested in computer chess and have collected almost every learned paper and article on the subject that I could find. My files are now quite large, and a considerable amount of time, effort and expense has been required to build up this collection.

I have often thought how difficult it must be for many computer chess enthusiasts to acquire copies of articles that they see referenced in some other work. Unless one has access to a good reference library, the task is almost impossible. I therefore decided to try to make available, in one volume, as many as possible of the most interesting and important articles and papers ever written on the subject. Such a selection is naturally somewhat subjective, and I hope that I will not offend authors whose works have been excluded. In particular I have decided to exclude any material which has appeared in the *Journal of the International Computer Chess Association* (ICCA), or in its precursor, the *ICCA Newsletter.* The reason is simply that the ICCA itself is in the process of compiling a compendium containing the most important material published in those sources. For further information on ICCA membership and publications the reader is invited to contact:

| | | |
|---|---|---|
| Professor H. J. van den Herik,<br>University of Limburg,<br>Department of Computer Science<br>6200 MD Maastricht<br>Netherlands | or | Dr Jonathan Schaeffer<br>Computing Science Department,<br>University of Alberta,<br>Edmonton<br>Alberta, Canada T6G 2H1. |

In compiling a volume of this nature, the editor must naturally seek permission from the copyright owners of the original publications. In every case the permission that I requested was granted, and I have acknowledged the source and the copyright owner on the first page of each article. In general I have been as faithful to the original as the typesetters and proof-readers would permit, but in the interest of consistency I have adopted a uniform policy in two areas. Firstly, instead of retaining the bibliographies that appeared at the end of many of the papers in this collection, I have ensured that each and every one of the references contained in those bibliographies has been included in the bibliography at the end of this volume. This saves space and allowed me to make the style of reference uniform throughout the book.

Secondly, where games or fragments of games appeared in descriptive notation in the original, I have translated them into algebraic. Chess programs employ algebraic notation for move entry and display, and most of my readers will be more familiar with the algebraic form than descriptive. For anyone who has never come across algebraic notation before, it is sufficient to know that the square in White's bottom left-hand corner is a1, his bottom right hand corner is h1, his top left hand corner is a8 and the remaining corner h8. A chess move in algebraic notation may appear as: ♕d1-h5 (naming the piece, the "from" square and the "to" square), or merely ♕h5. In pawn moves the pawn's symbol is often omitted. Checks are indicated by +, and the rest is easy.

In a book of this size some typographical errors are inevitable, and I will always be pleased to hear, via the publishers, of any mistakes discovered by the reader.

At the start of each chapter I have written a brief introduction. Apart from these meagre contributions, the credit for this volume goes to the original authors. Their work has been the inspiration of chess programmers everywhere, and anyone who is interested in the subject ought to find most, if not all of the material in this book, absorbing and valuable.

David Levy
London, September 1988

"Chess is *the* intellectual game par excellence. Without a chance device to obscure the contest it pits two intellects against each other in a situation so complex that neither can hope to understand it completely, but sufficiently amenable to analysis that each can hope to out-think his opponent. The game is sufficiently deep and subtle in its implications to have supported the rise of professional players, and to have allowed a deepening analysis through 200 years of intensive study and play without becoming exhausted or barren. Such characteristics mark chess as a natural arena for attempts at mechanization. If one could devise a successful chess machine, one would seem to have penetrated to the core of human intellectual endeavour."

A. Newell, J. C. Shaw and H. A. Simon

# 1. Introduction

This chapter serves as a historical introduction to the remainder of the volume. The very first paper, Shannon's seminal work dating back to 1949, was first presented as a lecture on March 9th of that year to the National Convention of the Institute of Radio Engineers in New York. Shannon pioneered computer chess as we know it today, and his ideas have been employed in almost every chess program ever written. He was not the first to write on computer chess: Konrad Zuse (1945) wrote a less interesting and considerably less useful theoretical contribution, and Henri Vigneron reported as long ago as 1914 on a special purpose chess machine that played the ending of king and rook v king (see section 7.2), but it was Shannon who really showed the way.

Turing's section is included because of the author's renown as a computer scientist from the earliest era. Although Turing was a keen chess player he was, by all accounts, not a very strong one. His paper includes a description of a "program" simulated by hand, and serves as a brief state-of-the-art survey for the early 1950s.

Alan Newell was one of the first to become involved in writing a real computer program that could play chess. He and his colleagues Shaw and Simon made a significant contribution to computer chess during its first decade. Newell's paper is about the early work on the NSS chess program, a later version of which is described in more detail in section 2.1

# 1.1 Programming a Computer for Playing Chess

**Claude E. Shannon**

First published in *Philosophical Magazine,* volume 41 (7th series), 1950, pp. 256-275.
Reprinted by permission of Taylor & Francis Ltd., and of the author.

## 1. Introduction

This paper is concerned with the problem of constructing a computing routine or "program" for a modern general purpose computer which will enable it to play chess. Although perhaps of no practical importance, the question is of theoretical interest, and it is hoped that a satisfactory solution of this problem will act as a wedge in attacking other problems of a similar nature and of greater significance. Some possibilities in this direction are :-

(1) Machines for designing filters, equalizers, etc.
(2) Machines for designing relay and switching circuits.
(3) Machines which will handle routing of telephone calls based on the individual circumstances rather than by fixed patterns.
(4) Machines for performing symbolic (non-numerical) mathematical operations.
(5) Machines capable of translating from one language to another.
(6) Machines for making strategic decisions in simplified military operations.
(7) Machines capable of orchestrating a melody.
(8) Machines capable of logical deduction.

It is believed that all of these and many other devices of a similar nature are possible developments in the immediate future. The techniques developed for modern electronic and relay type computers make them not only theoretical possibilities, but in several cases worthy of serious consideration from the economic point of view.

Machines of this general type are an extension over the ordinary use of numerical computers in several ways. First, the entities dealt with are not primarily numbers, but rather chess positions, circuits, mathematical expressions, words, etc. Second, the proper procedure involves general principles, something of the nature of judgement, and considerable trial and error, rather than a strict, unalterable computing process. Finally, the solutions of these problems are not merely right or wrong but have a continuous range of "quality" from the best down to the worst. We might be satisfied with a machine that designed good filters even though they were not always the best possible.

The chess machine is an ideal one to start with, since: (1) the problem is sharply defined both in allowed operations (the moves) and in the ultimate goal (checkmate); (2) it is neither so simple as to be trivial nor too difficult for satisfactory solution; (3) chess is generally considered to require "thinking" for skilful play; a solution of this problem will force us either to admit the possibility of mechanized thinking or to further restrict our concept of "thinking"; (4) the discrete structure of chess fits well into the digital nature of modern computers.

There is already a considerable literature on the subject of chess-playing machines. During the late 18th and early 19th centuries, the Maelzel Chess Automaton, a device invented by von Kempelen, was exhibited widely as a chess-playing machine. A number of papers appeared at the time, including an analytical essay by Edgar Allan Poe (entitled Maelzel's Chess Player) purporting to explain its operation. Most of these writers concluded, quite correctly, that the Automaton was operated by a concealed human chess-master; the arguments leading to this conclusion, however, were frequently fallacious. Poe assumes, for example, that it is as easy to design a machine which will invariably win as one which wins occasionally, and argues that since the Automaton was not invincible it was therefore operated by a human, a clear *non sequitur.*

For a complete account of the history and method of operation of the Automaton, the reader is referred to a series of articles by Harkness and Battell in *Chess Review*, 1947.

A more honest attempt to design a chess-playing machine was made in 1914 by Torrés y Quévedo, who constructed a device which played an end game of king and rook against king (Vigneron, 1914). The machine played the side with king and rook and would force checkmate in a few moves however its human opponent played. Since an explicit set of rules can be given for making satisfactory moves in such an end game, the problem is relatively simple, but the idea was quite advanced for that period.

The thesis we will develop is that modern general purpose computers can be used to play a tolerably good game of chess by the use of a suitable computing routine or "program". While the approach given here is believed fundamentally sound, it will be evident that much further experimental and theoretical work remains to be done.

## 2. General Considerations

A chess "position" may be defined to include the following data:-

(1) A statement of the positions of all pieces on the board.

(2) A statement of which side, White or Black, has the move.

(3) A statement as to whether the kings and rooks have moved. This is important since by moving a rook, for example, the right to castle on that side is forfeited.

(4) A statement of, say, the last move. This will determine whether a possible *en passant* capture is legal, since this privilege is forfeited after one move.

(5) A statement of the number of moves made since the last pawn move or capture. This is important because of the 50 move drawing rule. For simplicity, we will ignore the rule of draw after three repetitions of a position.

In chess there is no chance element apart from the original choice of which player has the first move. This is in contrast with card games, backgammon, etc. Furthermore, in chess each of the two opponents has "perfect information" at each move as to all previous moves (in contrast with Kriegspiel, for example). These two facts imply (von Neumann and Morgenstern, 1944) that any given position of the chess pieces must be either:-

(1) A won position for White. That is, White can force a win, however Black defends.

(2) A draw position. White can force at least a draw, however Black plays, and likewise Black can force at least a draw, however White plays. If both sides play correctly the game will end in a draw.

(3) A won position for Black. Black can force a win, however White plays.

This is, for practical purposes, of the nature of an existence theorem. No practical method is known for determining to which of the three categories a general position belongs. If there were chess would lose most of its interest as a game. One could determine whether the initial position is won, drawn, or lost for White and the outcome of a game between opponents knowing the method would be fully determined at the choice of the first move. Supposing the initial position a draw (as suggested by empirical evidence from master games*) every game would end in a draw.

It is interesting that a slight change in the rules of chess gives a game for which it is provable that White has at least a draw in the initial position. Suppose the rules the same as those of chess except that a player is not forced to move a piece at his turn to play, but may, if he chooses, "pass". Then we can prove as a theorem that White can at least draw by proper play. For in the initial position either he has a winning move or not. If so, let him make this move. If not, let him pass. Black is now faced with essentially the same position that White had before, because of the mirror symmetry of the initial position †. Since White had no winning move before, Black has none now. Hence, Black at best can draw. Therefore, in either case White can at least draw.

In some games there is a simple *evaluation function* $f(P)$ which can be applied to a position P and whose value

---

* The world championship match between Capablanca and Alekhine ended with the score Alekhine 6, Capablanca 3, drawn 25.

† The fact that the number of moves remaining before a draw is called by the 50-move rule has decreased does not affect this argument.

determines to which category (won, lost, etc.) the position P belongs. In the game of Nim (Hardy and Wright, 1938) for example, this can be determined by writing the number of matches in each pile in binary notation. These numbers are arranged in a column (as though to add them). If the number of ones in each column is even, the position is lost for the player to move, otherwise won.

If such an evaluation function $f(P)$ can be found for a game it is easy to design a machine capable of perfect play. It would never lose or draw a won position and never lose a draw position and if the opponent ever made a mistake the machine would capitalize on it. This could be done as follows: Suppose

$f(P) = 1$ for a won position,
$f(P) = 0$ for a drawn position,
$f(P) = -1$ for a lost position.

At the machine's turn to move it calculates $f(P)$ for the various positions obtained from the present position by each possible move that can be made. It chooses that move (or one of the set) giving the maximum value to $f$. In the case of Nim where such a function $f(P)$ is known, a machine has actually been constructed which plays a perfect game*.

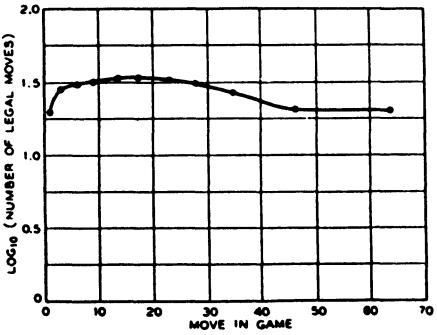
With chess it is possible, *in principle*, to play a perfect game or construct a machine to do so as follows: One considers in a given position all possible moves, then all moves for the opponent, etc., to the end of the game (in each variation). The end must occur, by the rules of the game after a finite number of moves † (remembering the 50 move drawing rule). Each of these variations ends in win, loss or draw. By working backward from the end one can determine whether there is a forced win, the position is a draw or is lost. It is easy to show, however, that even with the high computing speeds available in electronic calculators this computation is impractical. In typical chess positions there will be of the order of 30 legal moves. The number holds fairly constant until the game is nearly finished as shown in Fig.1. This graph was constructed from data given by De Groot, who averaged the number of legal moves in a large number of master games (De Groot, 1946, pp.17-18). Thus a move for White and then one for Black gives about $10^3$ possibilities. A typical game lasts about 40 moves to resignation of one party. This is conservative for our calculation since the machine should calculate out to checkmate, not resignation. However, even at this figure there will be $10^{120}$ variations to be calculated from the initial position. A machine operating at the rate of one variation per micro-microsecond would require over $10^{90}$ years to calculate its first move!

Another (equally impractical) method is to have a "dictionary" of all possible positions of the chess pieces. For each possible position there is an entry giving the correct move (either calculated by the above process or supplied by a chess master). At the machine's turn to move it merely looks up the position and makes the indicated move. The number of possible positions, of the general order of $64! / 32!\, 8!^2\, 2!^6$, or roughly $10^{43}$, naturally makes such a design unfeasible.

It is clear then that the problem is not that of designing a machine to play perfect chess (which is quite impractical) nor one which merely plays legal chess (which is trivial). We would like it to play a skilful game, perhaps comparable to that of a good human player.

A *strategy* for chess may be described as a process for choosing a move in any given position. If the process always chooses the same move in any given position. If the process always chooses the same move in the same position the strategy is known in

Fig. 1




* Condon, Tawney and Derr (1940). The "Nimotron" based on this patent was built and exhibited by Westinghouse at the 1938 New York World's Fair.

† The longest possible chess game is 6350 moves, allowing 50 moves between each pawn move or capture. The longest tournament game on record between masters lasted 168 moves, and the shortest four moves. (Chernev, 1937.)

the theory of games as a "pure" strategy. If the process involves statistical elements and does not always result in the same choice it is a "mixed" strategy. the following are simple examples of strategies:-

(1) Number the possible legal moves in the position P, according to some standard procedure. Choose the first on the list. This is a pure strategy.

(2) Number the legal moves and choose one at random from the list. This is a mixed strategy.

Both of these, of course, are extremely poor strategies, making no attempt to select good moves. Our problem is to develop a tolerably good strategy for selecting the move to be made.

### 3. Approximate Evaluating Functions

Although in chess there is no known simple and exact evaluating function $f(P)$, and probably never will be because of the arbitrary and complicated nature of the rules of the game, it is still possible to perform an approximate evaluation of a position. Any good chess player must, in fact, be able to perform such a position evaluation. Evaluations are based on the general structure of the position, the number and kind of Black and White pieces, pawn formation, mobility, etc. These evaluations are not perfect, but the stronger the player the better his evaluations. Most of the maxims and principles of correct play are really assertions about evaluating positions, for example:–

(1) The relative values of queen, rook, bishop, knight and pawn are about 9, 5, 3, 3, 1, respectively. Thus other things being equal (!) if we add the numbers of pieces for the two sides with these coefficients, the side with the largest total has the better position.

(2) Rooks should be placed on open files. This is part of a more general principle that the side with the greater mobility, other things equal, has the better game.

(3) Backward, isolated and doubled pawns are weak.

(4) An exposed king is a weakness (until the end game)

These and similar principles are only generalizations from empirical evidence of numerous games, and only have a kind of statistical validity. Probably any chess principle can be contradicted by particular counter examples. However, from these principles one can construct a crude evaluation function. The following is an example:–

$$f(P) = 200(K-K') + 9(Q-Q') + 5(R-R') + 3(B-B' + N-N') + (P-P') - \cdot 5(D-D' + S-S' + I-I') + \cdot 1(M-M') + \ldots$$

in which:–

K, Q, R, B, N, P are the number of White kings, queens, rooks, bishops, knights and pawns on the board.

D, S, I are doubled, backward and isolated White pawns.

M = White mobility (measured, say, as the number of legal moves available to White).

Primed letters are the similar quantities for Black.

The coefficients ·5 and ·1 are merely the writer's rough estimate. Furthermore, there are many other terms that should be included (see Appendix). The formula is given only for illustrative purposes. Checkmate has been artificially included here by giving the king the large value 200 (anything greater than the maximum of all other terms would do).

It may be noted that this approximate evaluation $f(P)$ has a more or less continuous range of possible values, while with an exact evaluation there are only three possible values. This is as it should be. In practical play a position may be an "easy win" if a player is, for example, a queen ahead, or a very difficult win with only a pawn advantage. In the former case there are many ways to win while in the latter exact play is required, and a single mistake often destroys the advantage. The unlimited intellects assumed in the theory of games, on the other hand, never make a mistake and the smallest winning advantage is as good as mate in one. A game between two such mental giants, Mr. A and Mr. B, would proceed as follows. They sit down at the chessboard, draw for colours, and then survey the pieces for a moment. Then either

(1) Mr. A says, "I resign" or

(2) Mr. B says, "I resign" or

(3) Mr. A says, "I offer a draw," and Mr. B replies, "I accept."

### 4. Strategy Based on an Evaluation Function

A very important point about the simple

type of evaluation function given above (and general principles of chess) is that they can only be applied in relatively quiescent positions. For example, in an exchange of queens White plays, say Q×Q (×=captures) and Black will reply P×Q. It would be absurd to calculate the function $f(P)$ after Q×Q while White is, for a moment, a queen ahead, since Black will immediately recover it. More generally it is meaningless to calculate an evaluation of the general type given above during the course of a combination or a series of exchanges.

More terms could be added to $f(P)$ to account for exchanges in progress, but it appears that combinations, and forced variations in general, are better accounted for by examination of specific variations. This is, in fact, the way chess players calculate. A certain number of variations are investigated move by move until a more or less quiescent position is reached and at this point something of the nature of an evaluation is applied to the resulting position. The player chooses the variation leading to the highest evaluation for him when the opponent is assumed to be playing to reduce this evaluation.

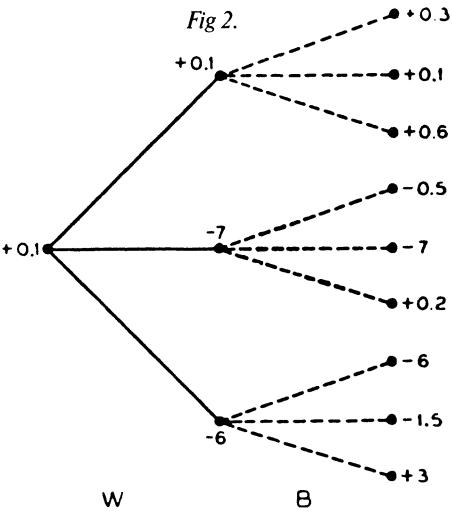
The process can be described mathematically. We omit at first the fact that $f(P)$ should only be applied in quiescent positions. A strategy of play based on $f(P)$ and operating one move deep is the following. Let $M_1$, $M_2$, $M_3$, ..., $M_s$ be the moves that can be made in position P and let $M_1P$, $M_2P$, etc. denote symbolically the resulting positions when $M_1$, $M_2$, etc. are applied to P. Then one chooses the $M_m$ which maximizes $f(M_mP)$.

A deeper strategy would consider the opponent's replies. Let $M_{i1}$, $M_{i2}$, ..., $M_{is}$ be the possible answers by Black, if White chooses move $M_i$. Black should play to *minimize* $f(P)$. Furthermore, his choice occurs *after* White's move. Thus, if White plays $M_i$ Black may be assumed to play the $M_{ij}$ such that $f(M_{ij}M_iP)$ is a *minimum*. White should play his first move such that $f$ is a maximum after Black chooses his best reply. Therefore, White should play to maximize on $M_i$ the quantity

$$\min_{M_{ij}} f(M_{ij}M_iP).$$

The mathematical process involved is shown for a simple case in Fig. 2. The point at the left represents the position being considered. It is assumed that there are three possible moves for White, indicated by the three solid lines, and if any of these is made there are three possible moves for Black, indicated by the dashed lines. The possible positions after a White and Black move are then the nine points on the right, and the numbers are the evaluations for these positions. Minimizing on the upper three gives +·1 which is the resulting value if White chooses the upper variation and Black replies with his best move. Similarly, the second and third moves lead to values of −7 and −6. Maximizing on White's move, we obtain +·1 with the upper move as White's best choice.

*Fig 2.*




In a similar way a two-move strategy (based on considering all variations out to 2 moves) is given by

$$\max_{M_i} \min_{M_{ij}} \max_{M_{ijk}} \min_{M_{ijkl}} f(M_{ijkl}\ M_{ijk}\ M_{ij}\ M_iP) \qquad \ldots (1)$$

The order of maximizing and minimizing this function is important. It derives from the fact that the choices of moves occur in a definite order.

A machine operating on this strategy at the two-move level would first calculate all variations out to two moves (for each side) and the resulting positions. The evaluations $f(P)$ are calculated for each of these positions. Fixing all but the last Black move, this last is varied and the move chosen which minimizes $f$. This is Black's assumed last move in the variation in question. Another move for White's second move is chosen and the process repeated for

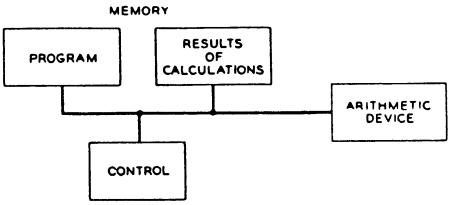
Black's second move. This is done for each second White move and the one chosen giving the largest final $f$ (after Black's best assumed reply in each case). In this way White's second move in each variation is determined. Continuing in this way the machine works back to the present position and the best first White move. This move is then played. This process generalizes in the obvious way for any number of moves.

A strategy of this sort, in which all variations are considered out to a definite number of moves and the move then determined from a formula such as (1) will be called a type A strategy. The type A strategy has certain basic weaknesses, which we will discuss later, but is conceptually simple, and we will first show how a computer can be programmed for such a strategy.

## 5. Programming a General Purpose Computer for a Type A Strategy

We assume a large-scale digital computer, indicated schematically in Fig.3, with the following properties :-

*Fig.3*




(1) There is a large internal memory for storing numbers. The memory is divided into a number of boxes each capable of holding, say, a ten-digit number. Each box is assigned a "box number".

(2) There is an arithmetic organ which can perform the elementary operations of addition, multiplication, etc.

(3) The computer operates under the control of a "program". The program consists of a sequence of elementary "orders". A typical order is A372, 451, 133. This means, extract the contents of box 372 and of box 451, add these numbers, and put the sum in box 133. Another type of order involves a decision, for example, C291, 118, 345. This tells the machine to compare the contents of box 291 and 118. If the first is larger the machine goes on to the next order in the program. If not, it takes its next order from box 345. This type of order enables the machine to choose from alternative procedures, depending on the results of previous calculations. It is assumed that orders are available for transferring numbers, the arithmetic operations, and decisions.

Our problem is to represent chess as numbers and operations on numbers, and to reduce the strategy decided upon to a sequence of computer orders. We will not carry this out in detail but only outline the programs. As a colleague puts it, the final program for a computer must be written in words of one microsyllable.

*Fig.4*

BLACK

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

WHITE

CODE FOR PIECES

| | P | N | B | R | Q | K |
|---|---|---|---|---|---|---|
| WHITE | 1 | 2 | 3 | 4 | 5 | 6 |
| BLACK | -1 | -2 | -3 | -4 | -5 | -6 |

0 = EMPTY SQUARE

CODE FOR MOVE

(OLD SQUARE, NEW SQUARE, NEW PIECE (IF PROMOTION))

e2 - e4 → (14, 34, -)
e7 - e8(Q) → (64, 74, 5)

The rather Procrustean tactics of forcing chess into an arithmetic computer are dictated by economic considerations. Ideally, we would like to design a special computer for chess containing, in place of

the arithmetic organ, a "chess organ" specifically designed to perform the simple chess calculations. Although a large improvement in speed of operation would undoubtedly result, the initial cost of computers seems to prohibit such a possibility. It is planned, however, to experiment with a simple strategy on one of the numerical computers now being constructed.

A game of chess can be divided into three phases, the opening, the middle game, and the end game. Different principles of play apply in the different phases. In the opening, which generally lasts for about ten moves, development of the pieces to good positions is the main objective. During the middle game tactics and combinations are predominant. This phase lasts till most of the pieces are exchanged, leaving only kings, pawns, and perhaps one or two pieces on each side. The end game is mainly concerned with pawn promotion. Exact timing and such possibilities as "Zugzwang", stalemate, etc. become important.

Due to the difference in strategic aims, different programs should be used for the different phases of the game. We will be chiefly concerned with the middle game and will not consider the end game at all. There seems no reason, however, why an end game strategy cannot be designed and programmed equally well.

A square on a chessboard can be occupied in 13 different ways: either it is empty (0) or occupied by one of the six possible kinds of White pieces (♙=1, ♘=2, ♗=3, ♖=4, ♕=5, ♔=6) or one of the six possible Black pieces (♙=−1, ♘=−2, ..., ♔=−6). The 64 squares can be numbered according to a co-ordinate system as shown in Fig.4. The position of all pieces is then given by a sequence of 64 numbers each lying between −6 and +6. A total of 256 bits (binary digits) is sufficient memory in this representation. Although not the most efficient encoding, it is a convenient one for calculation. One further number $\lambda$ will be +1 or −1 according as it is White's or Black's move. A few more should be added for data relating to castling privileges (whether the White or Black kings and rooks have moved), and *en passant* captures (*e.g.*, a statement of the last move). We will neglect these, however. In this notation the starting chess position is given by :-

4, 2, 3, 5, 6, 3, 2, 4; 1, 1, 1, 1, 1, 1, 1, 1;
0, 0, 0, 0, 0, 0, 0, 0; 0, 0, 0, 0, 0, 0, 0, 0;
0, 0, 0, 0, 0, 0, 0, 0; 0, 0, 0, 0, 0, 0, 0, 0;
−1,−1,−1,−1,−1,−1,−1,−1;
−4,−2,−3,−5,−6,−3,−2,−4;+1(=$\lambda$).

A move (apart from castling and pawn promotion) can be specified by giving the original and final squares occupied by the moved piece. Each of these squares is a choice from 64, thus 6 binary digits each is sufficient, a total of 12 for the move. Thus the initial move e2–e4 would be represented by 1,4; 3,4. To represent pawn promotion a set of three binary digits can be added specifying the piece that the pawn becomes. Castling is described by giving the king move (this being the only way the king can move two squares). Thus, a move is represented by (*a, b, c*) where *a* and *b* are squares and *c* specifies a piece in case of promotion.

The complete program for a type A strategy consists of nine sub-programs which we designate $T_0$, $_1$, ..., $T_8$ and a master program $T_9$. The basic functions of these programs are as follows :-

$T_0$ – Makes move (*a, b, c*) in position P to obtain the resulting position.

$T_1$ – Makes a list of the possible moves of a pawn at square (*x, y*) in position P.

$T_2$, ..., $T_6$ – Similarly for other types of pieces: knight, bishop, rook, queen and king.

$T_7$ – Makes list of all possible moves in a given position.

$T_8$ – Calculates the evaluating function $f$(P) for a given position P.

$T_9$ – Master program; performs maximizing and minimizing calculation to determine proper move.

With a given position P and a move (*a, b, c*) in the internal memory of the machine it can make the move and obtain the resulting position by the following program $T_0$.

(1) The square corresponding to number *a* in the position is located in the position memory.

(2) The number in this square *x* is extracted and replaced by 0 (empty).

(3) (*a*) If $x$=1, and the first co-ordinate of *a* is 6 (White pawn being promoted) or if $x$=−1, and the first co-ordinate of *a* is 1 (Black pawn being promoted) the number *c* is placed in square *b* (replacing whatever was there).

(*b*) If $x$=6 and $a-b$=2 (White castles, king

side) 0 is placed in squares 04 and 07 and 6 and 4 in squares 06 and 05 respectively. Similarly for the cases $x=6$, $b-a=2$ (White castles, queen side) and $x=-6$, $a-b=\pm 2$ (Black castles, king or queen side).
(*c*) In all other cases, $x$ is placed in square $b$.
(4) The sign of $\lambda$ is changed.

For each type of piece there is a program for determining its possible moves. As a typical example the bishop program, $T_3$, is briefly as follows. Let $(x, y)$ be the co-ordinates of the square occupied by the bishop.
(1) Construct $(x+1, y+1)$ and read the contents $u$ of this square in the position P.
(2) If $u=0$ (empty) list the move $(x, y)$, $(x+1, y+1)$ and start over with $(x+2, y+2)$ instead of $(x+1, y+1)$.

If $\lambda u$ is positive (own piece in the square) continue to 3.

If $\lambda u$ is negative (opponent's piece in the square) list the move and continue to 3.

If the square does not exist continue to 3.
(3) Construct $(x+1, y-1)$ and perform similar calculations.
(4) Similarly with $(x-1, y+1)$.
(5) Similarly with $(x-1, y-1)$.

By this program a list is constructed of the possible moves of a bishop in a given position P. Similar programs would list the moves of any other piece. There is considerable scope for opportunism in simplifying these programs: *e.g.*, the queen program, $T_5$, can be a combination of the bishop and rook programs, $T_3$ and $T_4$.

Using the piece programs $T_1 \ldots T_6$ and a controlling program $T_7$ the machine can construct a list of *all* possible moves in any given position P. The controlling program $T_7$ is briefly as follows (omitting details):-
(1) Start at square 1,1 and extract contents $x$.
(2) If $\lambda x$ is positive start corresponding piece program $Tx$ and when complete return to (1) adding 1 to square number. If $\lambda x$ is zero or negative, return to 1 adding 1 to square number.
(3) Test each of the listed moves for legality and discard those which are illegal. This is done by making each of the moves in the position P (by program $T_0$) and examining whether it leaves the king in check.

With the programs $T_0 \ldots T_7$ it is possible for the machine to play legal chess, merely making a randomly chosen legal move at each turn to move. The level of play with such a strategy is unbelievably bad.* The writer played a few games against this random strategy and was able to checkmate generally in four or five moves (by fool's mate, etc.). The following game will illustrate the utter purposelessness of random play:-

| | White (Random) | Black |
|---|---|---|
| 1 | g2-g3 | e7-e5 |
| 2 | d2-d3 | ♗f8-c5 |
| 3 | ♗c1-d2 | ♕d8-f6 |
| 4 | ♘b1-c3 | ♕f6×f2 mate |

We now return to the strategy based on an evaluation $f(P)$. The program $T_8$ performs the function of evaluating a position according to the agreed-upon $f(P)$. This can be done by the obvious means of scanning the squares and adding the terms involved. It is not difficult to include terms such as doubled pawns, etc.

The final master program $T_9$ is needed to select the move according to the maximizing and minimizing process indicated above. On the basis of one move (for each side) $T_9$ works as follows:-
(1) List the legal moves (by $T_7$) possible in the present position.
(2) Take the first in the list and make this move by $T_0$, giving position $M_1P$.
(3) List the Black moves in $M_1P$.
(4) Apply the first one giving $M_{11}M_1P$, and evaluate by $T_8$.
(5) Apply the second Black move $M_{12}$ and evaluate.
(6) Compare, and reject the move with the smaller evaluation.
(7) Continue with the third Black move and compare with the retained value, etc.
(8) When the Black moves are exhausted, one will be retained together with its evaluation. The process is now repeated with the second White move.
(9) The final evaluation from these two computations are compared and the maximum retained.
(10) This is continued with all White moves until the best is selected (*i.e.* the one remaining after all are tried). This is the move to be made.

These programs are, of course, highly iterative. For that reason they should not

* Although there is a finite probability, of the order of $10^{-75}$, that random play would win a game from Botvinnik. Bad as random play is, there are even worse strategies which choose moves which actually *aid* the opponent. For example, White's strategy in the following game: 1 f2-f3 e7-e5 2 g2-g4 ♕d8-h4 mate.

require a great deal of program memory if efficiently worked out.

The internal memory for positions and temporary results of calculations when playing three moves deep can be estimated. Three positions should probably be remembered: the initial position, the next to the last, and the last position (now being evaluated). This requires some 800 bits. Furthermore, there are five lists of moves each requiring about 30×12=360 bits, a total of 1800. Finally, about 200 bits would cover the selections and evaluations up to the present calculations. Thus, some 3000 bits should suffice.

**6. Improvements in the Strategy.**

Unfortunately a machine operating according to this type A strategy would be both slow and a weak player. It would be slow since even if each position were evaluated in one microsecond (very optimistic) there are about $10^9$ evaluations to be made after three moves (for each side). Thus, more than 16 minutes would be required for a move, or 10 hours for its half of a 40-move game.

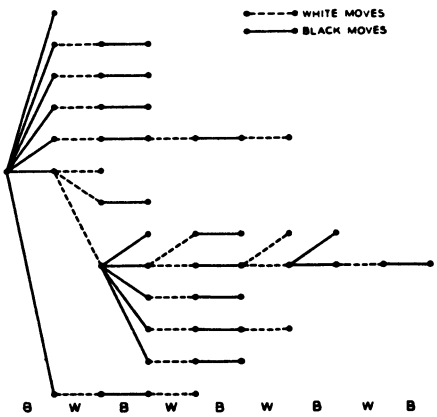
It would be weak in playing skill because it is only seeing three moves deep and because we have not included any conditions about quiescent positions for evaluation. The machine is operating in an extremely inefficient fashion – it computes *all* variations to *exactly* three moves and then stops (even though it or the opponent be in check). A good human player examines only a few selected variations and carries these out to a reasonable stopping-point.

A world champion can construct (at best) combinations say, 15 or 20 moves deep. Some variations given by Alekhine ("My Best Games of Chess 1924-1937") are of this length. Of course, only a few variations are explored to any such depth. In amateur play variations are seldom examined more deeply than six or eight moves, and this only when the moves are of a highly forcing nature (with very limited possible replies). More generally, when there are few threats and forceful moves, most calculations are not deeper than one or two moves, with perhaps half-a-dozen forcing variations explored to three, four or five moves.

On this point a quotation from Reuben Fine (Fine, 1942), a leading American master, is interesting: "Very often people have the idea that masters forsee everything or nearly everything; that when they played h2-h3 on the thirteenth move they foresaw that this would be needed to provide a loophole for the king after the complications twenty moves later, or even that when they play 1 e2-e4 they do it with the idea of preventing ... ♘f6-d5 on Black's twelfth turn, or they feel that everything is mathematically calculated down to the smirk when the a-pawn queens one move ahead of the opponent's g-pawn. All this is, of course, pure fantasy. The best course to follow is to note the major consequences for two moves, but try to work out forced variations as they go."

The amount of selection exercised by chess masters in examining possible variations has been studied experimentally by De Groot (1946, p.207). He showed various typical positions to chess masters and asked them to decide on the best move, describing aloud their analyses of the positions as they thought them through. In this manner the number and depth of the variations examined could be determined. Fig.5 shows the result of one such experiment. In this case the chess master examined sixteen variations, ranging in depth from $\frac{1}{2}$ (one Black move) to $4\frac{1}{2}$ (five Black and four White) moves. The total number of positions considered was 44.

*Fig.5*




From these remarks it appears that to improve the speed and strength of play the machine must:-

(1) Examine forceful variations out as far as

possible and evaluate only at reasonable positions, where some quasi-stability has been established.

(2) Select the variations to be explored by some process so that the machine does not waste its time in totally pointless variations.

A strategy with these two improvements will be called a type B strategy. It is not difficult to construct programs incorporating these features. For the first we define a function $g(P)$ of a position which determines whether approximate stability exists (no pieces *en prise*, etc.). A crude definition might be:-

$$g(P)=\begin{cases} 1 \text{ if any piece is attacked by a piece of lower value, or by more pieces than defences or if any check exists on a square controlled by opponent;} \\ 0 \text{ otherwise.} \end{cases}$$

Using this function, variations could be explored until $g(P)=0$, always, however, going at least two moves and never more than say, 10.

The second improvement would require a function $h(P,M)$ to decide whether a move M in position P is worth exploring. It is important that this preliminary screening should *not* eliminate moves which merely look bad at first sight, for example, a move which puts a piece *en prise*; frequently such moves are actually very strong since the piece cannot be safely taken.

"Always give check, it may be mate" is tongue-in-cheek advice given to beginners aimed at their predilection for useless checks. "Always investigate a check, it may lead to mate" is sound advice for any player. A check is the most forceful type of move. The opponent's replies are highly limited – he can never answer by counter attack, for example. This means that a variation starting with a check can be more readily calculated than any other. Similarly captures, attacks on major pieces, threats of mate, etc. limit the opponent's replies and should be calculated whether the move looks good at first sight or not. Hence. $h(P,M)$ should be given large values for all forceful moves (checks, captures and attacking moves), for developing moves, medium values for defensive moves, and low values for other moves. In exploring a variation $h(P,M)$ would be calculated as the machine computes and would be used to select the variations considered. As it gets further into the variation the requirements on $h$ are set higher so that fewer and fewer subvariations are examined. Thus, it would start considering every first move for itself, only the more forceful replies, etc. By this process its computing efficiency would be greatly improved.

It is believed that an electronic computer incorporating these two improvements in the program would play a fairly strong game, at speeds comparable to human speeds. It may be noted that a machine has several advantages over humans:-

(1) High-speed operation in individual calculations.

(2) Freedom from errors. The only errors will be due to deficiencies of the program while human players are continually guilty of very simple and obvious blunders.

(3) Freedom from laziness. It is all too easy for a human player to make instinctive moves without proper analysis of the position.

(4) Freedom from "nerves". Human players are prone to blunder due to overconfidence in "won" positions or defeatism and self-recrimination in "lost" positions.

These must be balanced against the flexibility, imagination and inductive and learning capacities of the human mind.

Incidentally, the person who designs the program can calculate the move that the machine will choose in any position, and thus in a sense can play an equally good game. In actual fact, however, the calculation would be impractical because of the time required. On a fair basis of comparison, giving the machine and the designer equal time to decide on a move, the machine might well play a stronger game.

## 7. Variations in Play and in Style

As described so far the machine once designed would always make the same move in the same position. If the opponent made the same moves this would always lead to the same game. It is desirable to avoid this, since if the opponent wins one game he could play the same variation and win continously, due perhaps to some particular position arising in the variation where the machine chooses a very weak move.

One way to prevent this is to have a statistical element in the machine. Whenever there are two or more moves which are of nearly equal value according

to the machine's calculations it chooses from them at random. In the same position a second time it may then choose another in the set.

The opening is another place where statistical variation can be introduced. It would seem desirable to have a number of the standard openings stored in a slow-speed memory in the machine. Perhaps a few hundred would be satisfactory. For the first few moves (until either the opponent deviates from the "book" or the end of the stored variation is reached) the machine plays by memory. This is hardly "cheating" since that is the way chess masters play the opening.

It is interesting that the "style" of play of the machine can be changed very easily by altering some of the coefficients and numerical factors involved in the evaluation function and the other programs. By placing high values on positional weaknesses, etc. a positional-type player results. By more intensive examination of forced variations it becomes a combination player. Furthermore, the strength of the play can be easily adjusted by changing the depth of calculation and by omitting or adding terms to the evaluation function.

Finally we may note that a machine of this type will play "brilliantly" up to its limits. It will readily sacrifice a queen or other piece in order to gain more material later or to give checkmate provided the completion of the combination occurs within its computing limits.

The chief weakness is that the machine will not learn by its mistakes. The only way to improve its play is by improving the program. Some thought has been given to designing a program which is self-improving but, although it appears to be possible, the methods thought of so far do not seem to be very practical. One possibility is to have a higher level program which changes the terms and coefficients involved in the evaluation function depending on the results of games the machine has played. Small variations might be introduced in these terms and the values selected to give the greatest percentage of "wins".

## 8. Another Type of Strategy

The strategies described above do not, of course, exhaust the possibilities. In fact there are undoubtedly others which are far more efficient in the use of available computing time on the machine. Even with the improvements we have discussed the above strategy gives an impression of relying too much on "brute force" calculations rather than on logical analysis of a position. It plays something like a beginner at chess who has been told some of the principles and is possessed of tremendous energy and accuracy for calculation but has no experience with the game. A chess master, on the other hand, has available knowledge of hundreds or perhaps thousands of standard situations, stock combinations, amd common manoeuvres which occur time and again in the game. There are, for example, the typical sacrifices of a knight at f7 or a bishop at h7, the standard mates such as the "Philidor Legacy", manœuvres based on pins, forks, discoveries, promotion, etc. In a given position he recognizes some similarity to a familiar situation and this directs his mental calculations along lines with greater probability of success.

There is no reason why a program based on such "type positions" could not be constructed. This would require, however, a rather formidable analysis of the game. Although there are various books analysing combination play and the middle game, they are written for human comsumption, not for computing machines. It is possible to give a person one or two specific examples of a general situation and have him understand and apply the general principle involved. With a computer an exact and completely explicit characterization of the situation must be given with all limitations, special cases, etc. taken into account. We are inclined to believe, however, that if this were done a much more efficient program would result.

To program such a strategy we might suppose that any position in the machine is accompanied by a rather elaborate analysis of the tactical structure of the position suitably encoded. This analytical data will state that, for example, the Black knight at f6 is pinned by a bishop, that the White rook at e1 cannot leave the back rank because of a threatened mate on c1, that a White knight at h4 has no move, etc.; in short, all the facts to which a chess player would ascribe importance in analysing tactical possibilities. These data would be supplied by a program and would be continually changed and kept up-to-date as the

game progressed. The analytical data would be used to trigger various other programs depending on the particular nature of the position. A pinned piece should be attacked. If a rook must guard the back rank it cannot guard the pawn in front of it, etc. The machine obtains in this manner suggestions of plausible moves to investigate.

It is not being suggested that we should design the strategy in our own image. Rather it should be matched to the capacities and weaknesses of the computer. The computer is strong in speed and accuracy and weak in analytical ability and recognition. Hence, it should make more use of brutal calculations than humans, but with possible variations increasing by a factor of $10^3$ every move, a little selection goes a long way toward improving blind trial and error.

**Acknowledgment**
The writer is indebted to E. G. Andrews, L. N. Enequist and H. E. Singleton for a number of suggestions that have been incorporated in the paper.

October 8, 1948.

## APPENDIX.

**The Evaluation Function for Chess.**

The evaluation function $f(P)$ should take into account the "long term" advantages and disadvantages of a position, *i.e.* effects which may be expected to persist over a number of moves longer than individual variations are calculated. Thus the evaluation is mainly concerned with positional or strategic considerations rather than combinational or tactical ones. Of course there is no sharp line of division; many features of a position are on the borderline. It appears, however, that the following might properly be included in $f(P)$:-

(1) Material advantage (difference in total material).
(2) Pawn formation;
  (a) Backward, isolated and doubled pawns.
  (b) Relative control of centre (pawns at d4, e4, f4).
  (c) Weakness of pawns near king (*e.g.* advanced g-pawn).
  (d) Pawns on opposite colour squares from bishop.
  (e) Passed pawns.
(3) Positions of pieces:
  (a) Advanced knight (at d5, e5, f5, d6, e6, f6), especially if protected by pawn and free from pawn attack.
  (b) Rook on open file, or semi-open file.
  (c) Rook on seventh rank.
  (d) Doubled rooks.
(4) Commitments, attacks and options:
  (a) Pieces which are required for guarding functions and, therefore, committed and with limited mobility.
  (b) Attacks on pieces which give one player an option of exchanging.
  (c) Attacks on squares adjacent to king.
  (d) Pins. We mean here immobilizing pins where the pinned piece is of value not greater than the pinning piece; for example, a knight pinned by a bishop.
(5) Mobility.

These factors will apply in the middle game: during the opening and end game different principles must be used. The relative values to be given each of the above quantities is open to considerable debate, and numerous other factors which may well be worth inclusion. The more violent tactical weapons, such as discovered checks, forks and pins by a piece of lower value are omitted since they are best accounted for by the examination of specific variations.

# 1.2 Chess

**Alan M. Turing**

First published in *Faster Than Thought* (B. V. Bowden, Editor), pp. 286-295. London, Pitman (1953).

When one is asked, "Could one make a machine to play chess?" there are several possible meanings which might be given to the words. Here are a few—

(a) Could one make a machine which would obey the rules of chess, i.e. one which would play random legal moves, or which could tell one whether a given move is a legal one?

(b) Could one make a machine which would solve chess problems, e.g. tell one whether, in a given position, white has a forced mate in three?

(c) Could one make a machine which would play a reasonably good game of chess, i.e. which,confronted with an ordinary (that is, not particularly unusual) chess position, would after two or three minutes of calculation, indicate a passably good legal move?

(d) Could one make a machine to play chess, and to improve its play, game by game, profiting from its experience?

To these we may add two further questions, unconnected with chess, which are likely to be on the tip of the reader's tongue.

(e) Could one make a machine which would answer questions put to it, in such a way that it would not be possible to distinguish its answers from those of a man?

(f) Could one make a machine which would have feelings as you and I have?

The problem to be considered here is (c), but to put this problem into perspective with the others I shall give the very briefest of answers to each of them.

To (a) and (b) I should say, "This certainly can be done. If it has not been done already it is merely because there is something better to do."

Question (c) we are to consider in greater detail, but the short answer is, "Yes, but the better the standard of play required, the more complex will the machine be, and the more ingenious perhaps the designer."

To (d) and (e) I should answer, "I believe so. I know of no really convincing argument to support this belief, and certainly of none to disprove it."

To (f) I should say, "I shall never know, any more than I shall ever be quite certain that *you* feel as I do."

In each of these problems except possibly the last, the phrase, "Could one make a machine to" ... might equally well be replaced by, "Could one programme an electronic computer to" ... Clearly the electronic computer so programmed would itself constitute a machine. And on the other hand if some other machine had been constructed to do the job we could use an electronic computer (of sufficient storage capacity), suitably programmed, to calculate what this machine would do, and in particular what answer it would give.

After these preliminaries let us give our minds to the problem of making a machine, or of programming a computer, to play a tolerable game of chess. In this short discussion it is of course out of the question to provide actual programmes, but this does not really matter on account of the following principle –

*If one can explain quite unambiguously in English, with the aid of mathematical symbols if required, how a calculation is to be done, then it is always possible to programme any digital computer to do that calculation, provided the storage capacity is adequate.*

This is not the sort of thing that admits of clear-cut proof, but amongst workers in the field it is regarded as being clear as day. Accepting this principle, our problem is reduced to explaining unambiguously in English the rules by which the machine is to choose its move in each position. For definiteness we will suppose the machine is playing white.

If the machine could calculate at an infinite speed, and also had an unlimited

storage capacity, a comparatively simple rule would suffice, and would give a result that in a sense could not be improved on. This rule could be stated:

"Consider every possible continuation of the game from the given position. There is only a finite number of them (at any rate if the fifty-move rule makes a draw obligatory, not merely permissive). Work back from the end of these continuations, marking a position with white to play as 'win' if there is a move which turns it into a position previously marked as 'win.' If this does not occur, but there is a move which leads to a position marked 'draw,' then mark the position 'draw.' Failing this mark it 'lose.' Mark a position with black to play by a similar rule with 'win' and 'lose' interchanged. If after this process has been completed it is found that there are moves which lead to a position marked 'win,' one of these should be chosen. If there is none marked 'win' choose one marked 'draw' if such exists. If all moves lead to a position marked 'lose,' any move may be chosen."

Such a rule is practically applicable in the game of noughts and crosses, but in chess is of merely academic interest. Even when the rule can be applied it is not very appropriate for use against a weak opponent, who may make mistakes which ought to be exploited.

In spite of the impracticability of this rule it bears some resemblance to what one really does when playing chess. One does not follow all the continuations of play, but one follows some of them. One does not follow them until the end of the game, but one follows them a move or two, perhaps more. Eventually a position seems, rightly or wrongly, too bad to be worth further consideration, or (less frequently) too good to hesitate longer over. The further a position is from the one on the board the less likely it is to occur, and therefore the shorter is the time which can be assigned for its consideration. Following this idea we might have a rule something like this—

"Consider all continuations of the game consisting of a move by white, a reply by black, and another move and reply. The value of the position at the end of each of these sequences of moves is estimated according to some suitable rule. The values at earlier positions are then calculated by working backwards move by move as in the theoretical rule given before. The move to be chosen is that which leads to the position with the greatest value."

It is possible to arrange that no two positions have the same value. The rule is then unambiguous. A very simple form of values, but one not having this property, is an evaluation of material, e.g. on the basis—

| | | |
|---|---|---|
| ♙ | = | 1 |
| ♘ | = | 3 |
| ♗ | = | $3\frac{1}{2}$ |
| ♖ | = | 5 |
| ♕ | = | 10 |
| Checkmate | = | 1000 |

If $B$ is black's total and $W$ is white's, then $W/B$ is quite a good measure of value. This is better than $W - B$ as the latter does not encourage exchanges when one has the advantage. Some small extra arbitrary function of position may be added to ensure definiteness in the result.

The weakness of this rule is that it follows all combinations equally far. It would be much better if the more profitable moves were considered in greater detail than the less. It would also be desirable to take into account more than mere "value of material".

After this introduction I shall describe a particular set of rules, which could without difficulty be made into a machine programme. It is understood that the machine is white and that white is next to play. The current position is called the *position on the board,* and the positions arising from it by later moves *positions in the analysis.*

### "Considerable" Moves

"Considerable" here is taken to mean moves which will be "considered" in the analysis by the machine.

Every possibility for white's next move and for black's reply is "considerable." If a capture is considerable then any recapture is considerable. The capture of an undefended piece or the capture of a piece of higher value by one of lower value is always considerable. A move giving checkmate is considerable.

### Dead Position

A position in the analysis is dead if there are no considerable moves in that position, i.e. if it is more than two moves ahead of

the present position, and no capture or recapture or mate can be made in the next move.

**Value of Position**

The value of a dead position is obtained by adding up the piece values as above, and forming the ratio *W/B* of white's total to black's. In other positions with white to play the value is the greatest value of (a) the positions obtained by considerable moves, or (b) the position itself evaluated as if a dead position. The latter alternative is to be omitted if all moves are considerable. The same process is to be undertaken for one of black's moves, but the machine will then choose the least value.

**Position-Play Value**

Each white piece has a certain position-play contribution and so has the black king. These must all be added up to give the position-play value.

For a ♕, ♖, ♗, or ♘, count –

(a) The square root of the number of moves the piece can make from the position, counting a capture as two moves, and not forgetting that the king must not be left in check.

(b) (If not a ♕) 1·0 if it is defended, and an additional 0·5 if twice defended.

For a ♔, count –

(c) For moves other than castling as (a) above.

(d) It is then necessary to make some allowance for the vulnerability of the ♔. This can be done by assuming it to be replaced by a friendly ♕ on the same square, estimating as in (a), but subtracting instead of adding.

(e) Count 1·0 for the possibility of castling later not being lost by moves of ♔ or rooks, a further 1·0 if castling could take place on the next move, and yet another 1·0 for the actual performance of castling.

For a ♙, count –

(f) 0·2 for each rank advanced.

(g) 0·3 for being defended by at least one piece (not ♙).

For the black ♔, count –

(h) 1·0 for the threat of checkmate.

(i) 0·5 for check.

We can now state the rule for play as follows. The move chosen must have the greatest possible value, and, consistent with this, the greatest possible position-play value. If this condition admits of several solutions a choice may be made at random, or according to an arbitrary additional condition.

Note that no "analysis" is involved in position-play evaluation. This is to reduce the amount of work done on deciding the move.

The game below was played between this machine and a weak player who did not know the system. To simplify the calculations the square roots were rounded off to one decimal place, i.e. this table was used.

| Number | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Square Root | 0 | 1 | 1·4 | 1·7 | 2·0 | 2·2 | 2·4 | 2·6 |

| Number | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|
| Square Root | 2·8 | 3·0 | 3·2 | 3·3 | 3·5 | 3·6 |

No random choices actually arose in this game. The increase of position-play value is given after white's move if relevant. An asterisk indicates that every other move had a lower position-play value.

| | *White:* | Machine | *Black:* | Human |
|---|---|---|---|---|
| 1 | e2–e4 | 4·2* | e7–e5 | |
| 2 | ♘b1–c3 | 3·1* | ♘g8–f6 | |
| 3 | d2–d4 | 2·6* | ♗f8–b4 | |
| 4 | ♘g1–f3 | 2·0 | | |

If 4 ♗c1–d2 3·7* then 4 . . . e5×d4 is foreseen.

| | | | | |
|---|---|---|---|---|
| 4 | . . . | | d7–d6 | |
| 5 | ♗c1–d2 | 3·5* | ♘b8–c6 | |
| 6 | d4–d5 | 0·2 | ♘c6–d4 | |
| 7 | h2–h4 | 1·1* | ♗c8–g4 | |
| 8 | a2–a4 | 1·0* | | |

White's last two moves are most inappropriate.

| | | | | |
|---|---|---|---|---|
| 8 | . . . | | ♘d4×f3+ | |
| 9 | g2×f3 | | ♗g4–h5 | |
| 10 | ♗f1-b5+ | 2·4* | c7–c6 | |
| 11 | d5×c6 | | 0–0 | |
| 12 | c6×b7 | | ♖a8–b8 | |
| 13 | ♗b5–a6 | –1·5 | ♕d8–a5 | |
| 14 | ♕d1–e2 | 0·6 | ♘f6–d7 | |
| 15 | ♖h1–g1 | 1·2* | | |

If 15 0-0-0 ♗b4×c3 16 ♗d2×c3 ♕a5×a4.

| | | | | |
|---|---|---|---|---|
| 15 | . . . | | ♘d7–c5 | |

The fork is unforeseen at White's last move.

| | | | | |
|---|---|---|---|---|
| 16 | ♖g1–g5 | | | |

Heads in the sand!

| | | | | |
|---|---|---|---|---|
| 16 | . . . | | ♗h5–g6 | |
| 17 | ♗a6-b5 | 0·4 | ♘c5×b7 | |

| | | | |
|---|---|---|---|
| 18 | 0-0-0 | 3·2* | ♘b7–c5 |
| 19 | ♗b5–c6 | | ♖f8–c8 |
| 20 | ♗c6–d5 | | ♗b4×c3 |
| 21 | ♗d2×c3 | | ♕a5×a4 |
| 22 | ♔c1–d2 | | ♘c5–e6 |
| 23 | ♖g5–g4 | –0·3 | ♘e6–d4 |
| 24 | ♕e2–d3 | | ♘d4–b5 |
| 25 | ♗d5–b3 | | ♕a4–a6 |
| 26 | ♗b3–c4 | | ♗g6–h5 |
| 27 | ♖g4–g3 | | ♕a6–a4 |
| 28 | ♗c4×b5 | | ♕a4×b5 |
| 29 | ♕d3×d6 | | |

Fiddling while Rome burns!

| | | |
|---|---|---|
| 29 | ... | ♖c8–d8 |
| 30 | Resigns | |

On the advice of his trainer.

Numerous criticisms of the machine's play may be made. It is quite defenceless against forks, although it may be able to see certain other kinds of combination. It is of course not difficult to devise improvements of the programme so that these simple forks are foreseen. The reader may be able to think of some such improvements for himself. Since no claim is made that the above rule is particularly good, I have been content to leave this flaw without remedy; clearly a line has to be drawn between the flaws which one will attempt to eliminate and those which must be accepted as a risk. Another criticism is that the scheme proposed, although reasonable in the middle game, is futile in the end game. The change-over from the middle game to the end-game is usually sufficiently clear-cut for it to be possible to have an entirely different system for the endgame. This should of course include quite definite programmes for the standard situations, such as mate with rook and king, or king and pawn against king. There is no intention to discuss the end-game further here.

If I were to sum up the weakness of the above system in a few words I would describe it as a caricature of my own play. It was in fact based on an introspective analysis of my thought processes when playing, with considerable simplifications. It makes oversights which are very similar to those which I make myself, and which may in both cases be ascribed to the considerable moves being inappropriately chosen. This fact might be regarded as supporting the glib view which is often expressed, to the effect that "one cannot programme a machine to play a better game than one plays oneself". This statement should I think be compared with another of rather similar form. "No animal can swallow an animal heavier than himself". Both statements are, as far as I know, untrue. They are also both of a kind that one is easily bluffed into accepting, partly because one thinks that there ought to be some slick way of demonstrating them, and one does not like to admit that one does not see what this argument is. They are also both supported by normal experience, and need exceptional cases to falsify them. The statement about chess programming may be falsified quite simply by the speed of the machine, which might make it feasible to carry the analysis a move farther than a man could do in the same time. This effect is less than might be supposed. Although electronic computers are very fast where conventional computing is concerned, their advantage is much reduced where enumeration of cases, etc., is involved on a large scale. Take for instance the problem of counting the possible moves from a given position in chess. If the number is 30 a man might do it in 45 seconds and the machine in 1 second. The machine has still an advantage, but it is much less overwhelming than it would be for instance when calculating cosines.

In connexion with the question of the ability of a chess-machine to profit from experience, one can see that it would be quite possible to programme the machine to try out variations in its method of play (e.g. variations in piece value) and adopt the one giving the most satisfactory results. This could certainly be described as learning, though it is not quite representative of learning as we know it. It might also be possible to programme the machine to search for new types of combination in chess. If this project produced results which were quite new, and also interesting to the programmer, who should have the credit? Compare this with the situation where a Defence Minister gives orders for research to be done to find a counter to the bow and arrow. Should the inventor of the shield have the credit, or should the Defence Minister?

# 1.3 The Chess Machine: An Example of Dealing with a Complex Task by Adaption

**Allan Newell**

First published in *Proceedings of the Western Joint Computer Conference,* 1955, pp. 101-108. Reprinted by permission of American Federation of Information Processing Societies Inc.

The modern general-purpose computer can be characterized as the embodiment of a three-point philosophy: (1) There shall exist a way of computing anything computable; (2) The computer shall be so fast that it does not matter how complicated the way is; and (3) Man shall be so intelligent that he will be able to discern the way and instruct the computer.

Sufficient experience with the large machine has accumulated to reveal the peculiar difficulties associated with these points. There has been a growing concern over problems which violate them, and instead satisfy the condition that (1) The relevant information is inexhaustible; (2) The set of potential solutions is neither enumerable nor simply representable; (3) The processing to be accomplished is unknown until other processing is done; and (4) An acceptable solution is required within a limited time.

Most design problems, including programming a computer, are of this nature; so are the very complex information processing tasks like translating languages or abstracting scientific articles. The current arguments about thinking machines and general-purpose robots also revolve about whether computers can deal with problems of this general nature.

The problem of playing good chess certainly falls into this class of ultracomplicated problems. It is a useful type case for general discussion because the nature of the task and the complexities surrounding it are common knowledge. Further, it already has something of a history. (Shannon, 1950; Weinberg, 1951; Richards, 1952; Strachey, 1952)

The aim of this effort, then, is to program a current computer to learn to play good chess. This is the means to understanding more about the kinds of computers, mechanisms, and programs that are necessary to handle ultracomplicated problems. The limitation to current computers provides the constant reminder that the heart of the problem lies in the limitation of resources, both memory and time. This aim would be somewhat ambitious even if all the details were available. The paper will actually be limited to presenting an over-all schema which appears feasible, and relating it to some of the critical problems which must be solved. The reference to learning expresses the conviction that the only way a machine will play good chess is to learn how. Although learning considerations have been prominent in the thinking and motivation behind the machine, attention will have to be restricted to the performance system; that is, to those features which are necessary in order to play t he game. However, some of the learning potentialities implicit in the performance system will be discussed.

The work presented here represents the early phases of an attempt to actually construct such a program for the JOHNNIAC, one of Rand's high speed computers.

Before starting it is desirable to give some additional conditions of the problem. From now on the computer with program will be called the machine, and the various parts of what it does will be called mechanisms. The problem is not to construct a machine which can induct the rules of chess by playing; it will be instructed concerning the legalities. Finally, the machine is only to do the job of one man; it will require an outside opponent, human or otherwise.

## Problems

As everyone knows (von Neumann et al., 1947), it is possible in principle to determine the optimal action for a given chess position. First compute out all

continuations to the bitter end. See whether they win, lose, or draw; and then work backwards on the assumption that the opponent will always do what is best for him and you will do what is best for you.

The difficulty, of course, is that this in principle solution requires a rather unnerving amount of computing power, and doesn't give any clues about what to do if you don't have it. It will provide us, however, with a short checklist of problems which must be solved if the computing requirements are ever to shrink to a reasonable size.

The most striking feature of the in principle solution is the tremendous number of continuations. This is accounted for by both the number of new consequences that appear at each additional move into the future, and the large number of moves required to explore to the end. This provides two problems:

1. The consequences problem, or which of the possibilities that follow from a given proposed action should be examined:

2. The horizon problem, or how far ahead to explore.

The possibility that one might stop looking at some intermediate position only raises a third problem:

3. The evaluation problem, or how to recognise a good position when you see one.

Another feature of the in principle solution is the identical examination of all the possible alternative actions. Despite the similarity in describing both present and future moves, it is worth while to keep distinct the actions that are actually available at a move, and from which a choice must be made; and the future consequences of these actions, which may include, among other things, limitations on the alternatives available in the future. Hence, we have;

4. The alternatives problem, or which actions are worth considering.

These four problems, consequences, horizon, evaluation, and alternatives, will be sufficient to keep us aware of the difficulties as we search for a set of mechanisms to play chess. Solutions must be found to all of them if the machine is to play good chess with reasonable resources.

## Overview

There is a common pattern to the solutions to be described here. In all of them the machine uses very partial and approximate methods. It is as if the machine consisted of a vast collection of rules of thumb. Each rule is a much oversimplified expression of how the machine should behave with respect to some particular aspect of the problem. The rules are of all kinds: chess principles to follow, measurements to make, what to do next, how to interpret rules of thumb, and so on. These are so organized that they form the necessary qualifications and additional specifications for each other. Each rule is essentially a machine program. At any particular instant the machine is under the control of some such rule, or shunting between rules under the control of a master program.

The main effort of the paper is devoted to describing how such a set of rules can be defined and organized to achieve solutions to the four problems, and thus provide a schema for a machine which puts all these pieces together to play chess. Only minor effort is devoted to indicating the detailed structure of these programs at the level of machine code.

One aspect of the underlying coding does require attention, and is dealt with at the end of the paper. The large number of rules, their complexity, and the necessity for adding new ones and modifying old ones, implies the use of a fairly extensive general-purpose language. That is, all these rules are to be given in this language or pseudo-code, as it might also be called. Hence, each use of a rule must be preceded by an interpretive step. However, a few programs suffice for using any and all of the rules that might be required in the machine.

## Preliminaries

Let us start by providing the machine with a few basic facilities. Each chessman and each square of the chessboard needs a name, suitably coded into binary bits. A fixed set of addresses are set aside to hold the current position. This can be given as a list of the men with the squares they occupy, including a "zero" square if the man is off the board. The machine can accept an opponent's action by reading a punched card. This can be given as a list of the men which have been moved, along with their new locations. Thus an action

involving a capture has two terms: one giving the new location of the man that captured, and a second giving the location of the captured man as the zero square, that is, off the board. The machine obtains the new position by substituting in the old one, which is already stored. It can also output its own action on a punched card. The machine must also be equipped to answer an array of elementary questions that recur constantly, such as, "Can a given man move to a given square?" or, "What man is blocking the Queen Pawn?" Each of these questions can be answered by a straightforward and not-too-lengthy investigation of the current position. The number of actual programs needed is within reasonable bounds since the more complicated questions are combinations of the more elementary ones.

Assume, then, that the machine has these basic capabilities. They have solved none of the four problems.

## Goals and Tactics

Suppose, in the midst of a game, the machine (which is White) has stored in a suitable place the following expression:

*att(BKB, WKR).*

The machine interprets this as: Attack the Black King-Bishop with the White King-Rook. Attack will be considered to mean a successful attack, which in turn means that the attacking man (here, the Rook) is capable of capturing the object of the attack (here, the Bishop) with a relative gain in material after the smoke of engagement has cleared. Thus, for a given position, the attack is either successful or not; and the machine can determine this by a sequence of those elementary programs given it earlier. Call such an expression a goal, which is either achieved or not achieved for any given position.

Now, suppose the machine were to start tracing out continuations. It could determine at each new position it arrived at whether or not the goal was achieved. If it is, the machine could stop searching. This might provide a solution to the horizon problem: have a goal and only explore continuations until a position is reached where the goal is achieved.

The machine can represent such a set of continuations as a branching net, or tree, of actions. Each action is linked by some kind of indexing to the immediately preceding action, and to each action is associated by this indexing a number of other actions that might possibly follow. Such a tree is a tactic for a given goal if it always terminates in positions that achieve that goal. Further, a tactic will always indicate a single action for the machine to take, and many actions for the opponent.

But if the machine has a goal like attack the Black King, it is right back with the original difficulty of searching for continuations which end in checkmate. This merely indicates that in general one can hardly expect to find a tactic for a given goal. But on the other hand, one sometimes will: men do get captured and games do get won.

By some means let the machine acquire in memory a second goal, which is indexed to the first:

*blk(BQ, BKB) → att (BKB, WKR).*

The machine interprets this as: Block the Black Queen's ability to recapture where the Black King-Bishop is. If this ability to recapture was in fact one of the deterrents to taking the Bishop with the Rook, then achievement of this second goal could be considered as an intermediate aid to achievement of the attack. Such a goal will be called a subgoal. It may itself have subgoals, and thus the machine may acquire rather large networks of goals.

Now, instead of trying to construct a tactic for the attack goal, the machine can search for a tactic for the subgoal of blocking. Perhaps it will find one. If it can not, then it needs either other subgoals for the attack or some subgoals for the block.

The single large search for winning positions has been replaced by two searches, each terminating the other. The machine searches for intermediate goals; it stops this search when it finds subgoals it can achieve. It determines the question of achievement by a search for particular sequences of action. This search terminates with a completed tactic, that is, when the sequences end in goal achievement.

But when to stop if the tactic search continues to be unsuccessful? The mechanism of subgoal formation operates only if the signal has been given to terminate further search for a tactic. Before discussing this, it is appropriate to consider another aspect of

the total problem, which, after a twist or two, will lead back to this same decision problem.

**Likelihoods**

The mechanisms introduced so far have only yielded a solution to the horizon problem. For instance, they do not provide a solution for the consequences problem. It would still seem the machine would examine all branches at any given future move it arrived at. Thus, tactics might be short, but they would have very large spreads.

Suppose the machine had some other goals in its memory labeled "opponent's goals." For example, the machine might have:

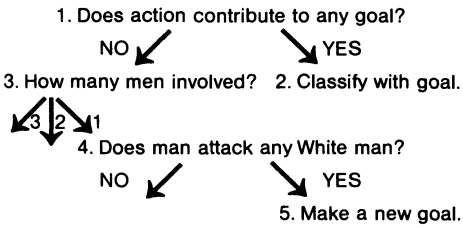
*ctl(d5).*

The machine interprets this as, "Control the d5 square," and takes as a definition of achievement for the opponent that he could win an engagement if White tried to occupy the square or capture a Black man who did occupy it. If the machine considered this goal to be one of the opponent's important goals, then clearly, any opponent's action which achieved this goal would be relatively likely to occur. Or would it? It certainly depends to what degree the machine's representations of the opponent's goals correspond to the opponent's real goals or whatever other mechanisms guide his actions.

The machine must now be equipped with all the apparatus for collecting evidence, forming hypotheses, verifying them, and inferences from them. At least in some rudimentary form these will all be required to develop and maintain a running model of the opponent's intentions.

If such a set of mechanisms were available to the machine, then, within certain limits of accuracy, it could assign likelihoods to the various alternatives by inference from the model. These likelihoods provide a solution to half the problem of consequences: how to know which consequences to examine at any position that is the opponent's move. Examine those alternatives which seem most likely.

A rather large number of mechanisms are involved. First, each action that the opponent makes must be classified as stemming from some goal or goals. Somewhere in memory let there be stored some expressions like:




These expressions are coded in the language spoken of earlier. The machine starts with expression 1 and interprets it. This instructs the machine to test each of the opponent's goals to see if the action contributes to it. This term is used in a narrow sense: only men who can capture contribute to an attack; only men who command the square contribute to its control. Thus, in the example, "control d5", the machine would determine if the action had resulted in any new man commanding the d5 square. If the answer is yes, the machine proceeds to expression 2. When interpreted, this instructs the machine to classify the action as stemming from the goal to which it contributes. This ends the decision process. But as usual, the answer could also be no; the first rule does not suffice to classify the action. The machine then proceeds to expression 3, interprets it, and counts the number of men who changed in the current action. Finding the number to be one, which means that nothing complicated like a capture, promotion, or castle occurred, it proceeds to expression 4. The machine is now instructed to see if the man which moved can be considered as attacking some new man, and if so to read expression 5, which instructs the machine to create a new goal. Notice that this alternative is considered only if the action has failed to be consistent with any of the goals already hypothesized as accounting for the opponent's action.

So far only a mechanism for classification and possible introduction of new goals has been described. It should be remarked about this mechanism that the content of the expressions is much less important at this juncture than the general scheme. Other expressions and more of them could just as well be used. Instead of describing the rest of the mechanisms needed to complete this component, it is necessary to move on to other key pieces. The undescribed elements will be constructed in the

same vein. For instance, there will be expressions for obtaining likelihoods like, "How often has a goal been contributed to recently," and "High activity implies continued activity is likely."

### Utility

Even admitting the machine can assign likelihoods to the opponent's alternatives, this solves only half of the problem of consequences associated with opponent's moves. There remains the other half: how to know which of the machine's possibilities to explore when searching for a tactic.

Consistent with the design philosophy emerging here, the machine will have a section of memory devoted to expressions in the language that are relevant to this problem. These will be organised into a decision net in the same fashion as the expressions for classifying opponent's actions. The object of such a net will be to assign a utility to future machine actions that could be explored in trying to complete tactics. This utility would be expected to reflect a complex of factors. The feature to note is not just the possibility of such nets, but the kinds of relevant information that now are available to be used.

The most important kind, I think, is that every man has associated with it the functions that it is performing. These associations occur by means of the goal structure. Suppose there is a subgoal that the King-Knight supports the King-Bishop; that is, can retaliate by recapturing any man that captures the Bishop. This serves as a statement that the Knight has a function to perform. Any goal whose tactic depends on moving the Knight into a new position must compete with the goal which already has the Knight supporting the Bishop. The machine can determine this by a simple search of its goals. Therefore a reasonable decision net would recommend exploration of the Knight's moves only as a last resort.

The vision of a large collection of small reasons why various moves should have low utility for exploration raises a caution. We do not want the machine to spend all its time examining the future actions of committed men; yet if it were never to do this, it could overlook real opportunities and might even gradually paralyze itself by the accumulation of commitments. The solution, of course, is the random element. The difficulty with random search in a complex environment is that in reasonable time scales, nothing will ever be discovered. Therefore randomness should be introduced into the decision nets only in small and well controlled amounts. The machine should rarely search for combinations which sacrifice a Queen and Rook to no apparent gain only to develop a mate eight moves later.

### Level of Aspiration

By now we have finally returned to the decision problem stated earlier. There are devices for stopping the search for tactics if successful. There are devices for differentiating the various directions of search. These are likelihoods for the opponent's moves, and utilities for the machine's. But there exists no stop rule in the face of continued difficulties.

To find a solution we turn to a phenomenon well known in the human sciences: levels of aspiration. The fundamental reason why the machine must terminate the search is limitation of resources, in this case mostly computing time. By a level-of-aspiration type solution is meant the introduction into the machine of the limitation of resources as an expicit mechanism. That is, various expressions will exist in the machine which measure and utilize the information about the amount of resources available. Thus, the limitations will no longer function as absolute constraints; the way in which they affect the machine is partially within the machine's control.

For tactics a suitable measure, correlated with computing time but more appropriate to the task, is the likelihood of a position. As the machine explores into the future, selecting one possible action after another, the likelihoods compound like probabilities so that the ultimate likelihood diminishes as actions get further into the future, or are reached through lower-likelihood actions.

The machine sets out on a tactic search with a predetermined level of final likelihood. It gradually extends the tree of actions until each terminal position has fallen below this level; then it stops. If now, the aggregate likelihood of reaching a position of goal achievement lies above another predetermined aspiration level, the machine considers the tactic adequate. If not, it returns to the goal structure and

develops some additional goals, as described earlier.

### Transformation Rules

Nothing has yet been said about how all these subgoals are generated. The structure of the mechanism is clear; it will consist of more rules of thumb. There will exist a set of expressions in the language which function like the rules of inference in logic: they allow us to derive new expressions from old ones. They are of the form, "For a goal of type A, try a subgoal of type B." For example, "For att (*x,y*) try def (*y*)." This expresses the fact that there is some merit to defending the men who are involved in carrying out an attack. If this transformation rule, as it is called, were applied prior to the opponent's actual attack on the attackers, this would constitute a fine example of anticipatory response.

The machine must examine its goal structure to determine the types of goals for which it requires new subgoals. It selects out those transformation rules which might be relevant; that is, where the structure of the goals in the net fits the specifications of the first part, or premise, of the transformation rule. Several possible rules may be obtained. It seems appropriate to introduce a random element to guide the final choice, since the inference expressed by the rules is rather a loose one. The weights assigned to each rule will be functions of experience, so that rules which work get chosen relatively often.

We have again arrived at the place where the important question is the actual content of these expressions. Certainly, if the rules are poor the machine will play terrible chess. Its operation will bear little relation to the objective game situations. Again, other problems are more important. No solution has yet been described for either the evaluation or the alternatives problem. It will be appropriate to consider evaluations next.

### Evaluations

It may seem that somehow the evaluation problem has an implicit solution in the array of mechanisms postulated so far. The problem is to relate a position to the winning of the game. Since the machine uses intermediate goals, the problem now is to relate positions to these subgoals. It seems that the tactics do provide this relation. If a sequence of actions in a tactic results, sometime, in the given position, then the likelihood of achieving that tactic's goal from the position is known. The machine spent time computing it when generating the tactic.

The immediate difficulty with this is that it relates a position to a goal only if it occurs in that goal's tactics.This provides no evaluation for the other goals. Hence, it is only a partial solution since the problem of evaluation is assessing the relation to the ultimate goal, which is winning. In order to make playing possible, the machine has replaced the single remote goal with a large number of more immediate ones. Thus, for the machine the problem is: for any given goal, determine the likelihood of achievement from any given position.*

The solution exists for the special case where the position occurs in the tactic of the goal. Consider two other special cases. First, suppose the goal were achieved for the given position. It would receive a likelihood of one if we think of likelihoods as similar to probabilities. Second, suppose a goal were not achieved, had no tactic, and no subgoals. It would receive a likelihood of zero, since its achievement rests solely with coincidence or the opponent, both equally bad bets.

These special cases prvide the basis for a solution of the evaluation problem. If the machine generates goals as described, then at any instant all the terminal elements belong to one of the three special cases. This is so since the situations not covered by the special cases all involve a goal having subgoals, which means they cannot be terminal elements. The terminal evaluations are a set of boundary conditions, from which, step by step, evaluations can be assigned throughout the goal structure.

In the general case, the problem will be to compute the likelihood of a goal, given the likelihoods of all its subgoals. It is here that the relation of subgoal is given

*Throughout the paper I have very carefully used the term "likelihood," instead of "probability." The logical status of the entities referred to is not at all clear since the machine uses the measured likelihoods to determine action which in turn determines whether the move or position will ever occur; what the likelihood "really" is.

operational significance. The machine operates as if the following principle were true: if $g_1$ is a subgoal of $g_2$, then an increase in the likelihood of achievement of $g_1$ produces an increase in the likelihood of achievement of $g_2$. It interprets its experience in this light, so contrary instances are treated as implying that the subgoal is a poor one.

The laws of combination of likelihoods must have several simple properties, mostly those implied by the principle above. Other than this, it may not make too much difference; simple linear combination may do admirably.

Although some of the details have not been enumerated, the machine now has the mechanisms for evaluation. the result is a large set of numbers, one for each goal. The reason why more combination is not required, or perhaps not even desirable, leads to the problem of alternatives.

**Alternatives**

Clearly, the machine is interested only in alternatives that further its goals. Now, goals are general statements, and the tactic is the bridge provided to pass between the goals and the very particular current situation. Tactics provide actions that lead to goal achievement. At any moment, then, there are a certain number of tactics and these yield an equal number of alternatives (not necessarily all distinct), all of which are worth considering since they each further some part of the machine's goal structure.

This would be a solution to the alternatives problem if this set were always adequate. It can hardly be expected to be so at all times. For instance, right after a very unexpected move by the opponent there may be no tactics left at all.

The solution lies in more levels of aspiration, partial efforts and iteration. For each alternative, the machine computes the goal evaluations for the new position that would follow if the alternative were chosen. This yields an evaluation of each alternative's effect throughout the goal structure. Although this evaluation appers to be only one more move deep, it is fundamentally grounded in the tactics, which extend much further into the future.

The decision is now made whether to make the available set suffice, or whether to return and work some more: to add and modify the goals and tactics. This is a level-of-aspiration type of decision, which will depend not only on whether the alternatives are good enough, but also on how much time remains, and whether the move is crucial. Only if the decision is made not to explore and expand further, is the best alternative picked from the limited set and punched into the card as the machine's actual move.

The term "best alternative", is used in a very casual way. The evaluations consist of many numbers, at least one for each goal. It is clear that if a single alternative dominates all others, it should be chosen. It is also fairly clear that an alternative which achieves a very important subgoal is to be preferred over one which only increases the likelihood of a few very subordinate ones. But basically this is a multiple value situation, and in general no such simple rules can be expected to indicate a single best action. The problem for the machine is not to somehow obtain a magic formula to solve the unsolvable but to make a reasonable choice with least effort and proceed with more productive work. There are other ways to deal with the problem; for instance, include conflict as a fundamental consideration in the decision to explore further. Thus, at each move the machine can be expected to iterate several times until it achieves an alternative that it likes, or until it runs out of time and thus loses the game by not being smart enough or lucky enough.

**Performance Schema**

The pieces now exist to give an overall schema for the performance system of the chess learning machine. This is a set of mechanisms which is sufficient to enable the machine to play chess. There is no learning in this system; it will play no better next time because it played this time. If the content of all the expressions required is appropriate, it will play good chess; if they are not, it will play very poor chess.

This performance system is highly adaptive. A goal structure peculiar to each play of the game is generated during the course of play. Tactics reflect the minute detail of the current situation. This short-run adaptability is not to be confused with learning which would permanently affect the way the machine would play in the future.

Fig.1 gives the scheme of operation. Rather than present as systematic and complete a representation as possible, attention has been given to relating the elements discussed so far. The rectangles represent the major kinds of information in the system. These may be viewed as memories. The arrows indicate processes that operate on one kind of information to produce another. The small writing by these arrows relates these processes to key words used earlier. Some of the main decisions are put in circles, since it makes the diagram easier to follow. The programs for carrying out most of these processes are the various nets, like the classification net. For the sake of clarity, these are not shown as expicit kinds of information, although they certainly occupy a large part of the computer's memory.

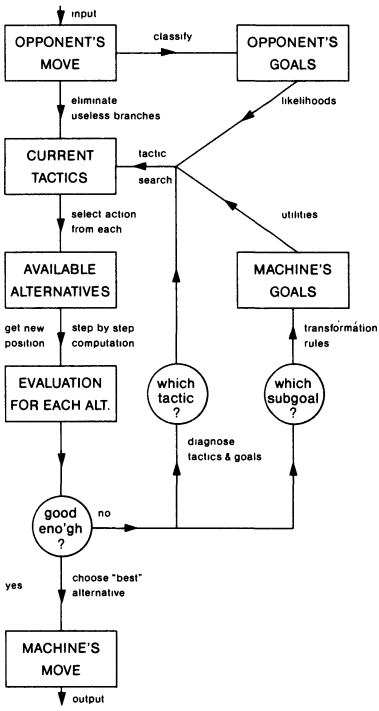
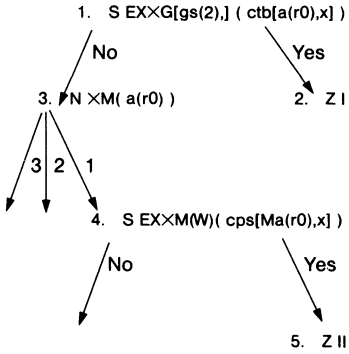



*Fig. 1—Schematic flow diagram for performance system.*

Each sequence always starts with an opponent's move being received (at the top). The process continues (downward) by a series of straightforward computations until the question is reached whether the situation is "good enough." This is the fundamental question. If the answer is yes, the machine has only to choose from among the available alternatives and play, thus ending the sequence (down). So far the effort spent is nominal. If, however, the answer is no, the machine proceeds to the modification and extension of the goals and tactics (to the right and up). This part is of indeterminate duration and effort and utilizes all of the complex apparatus that has been built up. Following this, the machine again attempts to produce a move (downward again). This is the fundamental cycle: to try to decide on a move with little effort, to modify the basis of decision, and to try again. Finally, of course, a move is made and the sequence stops.

## Language

It is necessary to add a few comments about the general-purpose language required to make a machine of this nature feasible. An essential feature of this language is its ability to refer to itself as well as to the "external" chess situation. The only languages of this power of expression that have been formalized at all are those used in symbolic logic such as the calculus of propositions (Carnap, 1937). The one illustrated below is an adaptation for computer use.




Each symbol has a binary code, although, for efficiency's sake, not all have different codes. An expression, then, is a long sequence of zeros and ones. The expression is decodable by proceeding from left to right: knowing which symbols have been found so far gives sufficient information to determine how many bits to consider next and what class of symbols it belongs to.

The following illustrates this language. It is the translation of the classification net exhibited earlier.

Consider expression 1 as an example. *In toto* it means, "Does the action contribute to

any goal?" A question in the symbolic language is a sentence without its truth value. The machine interprets the missing value as an instruction to determine it; that is, to obtain the answer to the question. Expression 1, then, starts with "S" to identify itself as a sentence. (Expression 3 has an "N," and its value would be a number.) "EX $x$ G" means, "there exists an $x$ which is a goal." The "gs(2)" in parentheses gives additional specifications on the range of $x$. The machine will only search gs(2) which is the opponent's goal structure. The last parentheses contain the predicate, "contribute" symbolized by "ctb". Its arguments are first, an action, "a(r0)," the "r0" standing for relative time zero so the machine will always look at the current action; and then a goal, "$x$," which is the variable over which the machine will search.

The machine interpretation and execution of such an expression runs along lines very similar to those used in algebraic coding schemes. The machine has programs for determining the truth or falsity of each individual predicate, here only the "ctb." Truth values are combined, with due regard for brackets, by the laws of "and," "or," and "not," depending on which occur in the expression. The machine interprets "EX" as making the sentence true if for any of the goals considered it obtains "true" for ctb.

## Learning

The section is limited to some remarks on the requirements for learning and the potentialities for it that exist in the machine.

The large number of highly interrelated mechanisms involved in the performance of the machine indicates a major difficulty. Consider the hundreds of expressions which each determine a highly specific rule of action. How is it possible to have a set that actually fits together to produce effective chess play? The assumption that man can discern which sets will work seems rather untenable; there are too many effects and hidden consequences. Learning seems to offer a solution.

At any given time the machine has a set of expressions, which work tolerably well over some restricted range of environments. The machine generates a few extensions or modifications to its current set. These are incorporated if their use provides an improvement in performance. The features to note about this learning process are, first, that it gradually extends the set of expressions, and second, that expressions get admitted only if they "fit in" with the others already there. Thus, given that an appropriate training sequence of games are played, the machine will grow itself a set of expressions of sufficient complexity to play good chess.

Many of the mechanisms necessary to provide this learning are already in the machine. First, it is necessary to measure the effects of one mechanism on another. The machine is already able to perform any test or measurement given by a network of expressions. All that is required is that the expressions refer to the appropriate internal parts of the machine and ask the right questions. The language is powerful enough to do this. These nets also allow diagnosis of which parts of a complex mechanism are causing the undesirable effects. Memory is needed to keep performance records, and mechanisms are needed to collect and classify these records; all of this is already possible. The possibilities for new mechanisms are limited only by the modes of expression of the language, since the specific behavior of the performance system is determined by the content of expressions. Finally, the search for new mechanisms is quite similar to obtaining subgoals for goals. That is, there would be sets of transformation rules to give the types of modifications that might prove useful for a given type of expression.

The problem of sample-size requires mention: How large a sample of experience is necessary to obtain learning? Or better: How much information about the effects of behavior is necessary to successfully modify the behavior? Chess affords a good example of this problem. It is extremely doubtful whether there is enough information in "win, lose, or draw" when referred to the whole play of the game to permit any learning at all over available time scales. There is too much behavior. For learning to take place, each play of the game must yield much more information. This is exactly what is achieved by breaking the problem into components. The unit of success is the goal. If a goal is achieved, its subgoals are reinforced; if not, they are inhibited. (Actually, what is reinforced is the transformation rule that provided the

subgoal). This is so whether the game is ultimately won or lost. Each play gives learning information about each goal that is generated. This also is true of the other kinds of structure: every tactic that is created provides information about the success or failure of tactic search rules; every opponent's action provides information about success or failure of likelihood inference; and so on. The amount of information relevant to learning increases directly with the number of mechanisms in the chess playing machine.

## Conclusion

"As every design engineer knows, the only difference between a good design and the actual machine is time and effort." This adage is a byword in the field of robots and thinking machines. The scheme presented here is not far from a "good design." One can estimate the man-hours necessary to draw up the detailed flow diagrams from which machine coding follows as routine chore. But this is not sufficient. These mechanisms are so complicated that it is impossible to predict whether they will work. The justification for the present article is the intent to see if *in fact* an organized collection of rules of thumb can "pull itself up by its bootstraps" and learn to play good chess.

# 2. Famous Chess Programs

There are seven papers in this chapter, describing the most famous programs in the history of computer chess. The first paper, by Newell, Shaw and Simon, describes a program to play chess on a 6×6 board (omitting the bishops), which was written at the Los Alamos Scientific Laboratories in New Mexico, birthplace of the hydrogen bomb! The authors then discuss the Bernstein program, more details of which are found in the next section, before providing an updated account of their own work whose earlier performance is reported in section 1.3.

The Bernstein program represents the first efforts in computer chess to come to the attention of the general public. In addition to the paper presented in section 2.2, there was a report on the program in *Scientific American,* and in various newspapers and magazines as well as on radio and television. The program played at the level of a weak amateur, and was only slightly less able than the program written by Alan Kotok for his B.Sc. thesis at M.I.T. Here (section 2.3) we are reproducing he text of Kotok's thesis, but omitting the 95 pages of his program code. The Kotok program was later moved to Stanford University from where it competed in a match against a Soviet program in 1966. The Russians won the match convincingly (see p.55), thereby establishing their dominance in computer chess for the first time.

The Greenblatt program (section 2.4) was primarily the work of Richard Greenblatt, an M.I.T. undergraduate who dropped his studies to devote more time to computer chess. The program was the first to compete in human chess tournaments and made a respectable score at its first attempts. For perhaps three years or thereabouts the Greenblatt program was the strongest, but during the period 1968-1975 little was done to improve it, until the CHEOPS special purpose hardware was developed (see section 7.3).

Jim Gillogly's program TECH (section 2.5) was written as a kind of benchmark for chess programmers. The author's philosophy was that the program's strategic evaluation should all be done at the first ply of the game tree, and that all further search effort should be devoted to tactics (specifically to material gains and losses). Gillogly's idea was that TECH should serve as a minimal performance program, beyond which any sophisticated programming strategy ought to be able to reach. Although his program's chess knowledge was minimal, Gillogly scored reasonably well in various computer chess tournaments before withdrawing from the computer chess arena in 1974.

The next section in this chapter is by David Slate and Larry Atkin, who are possibly the most famous chess programmers of all time. Their earliest efforts at computer chess, in collaboration with Keith Gorlen (a fellow student at Northwestern University), gained them first place at the 1970 North American Computer Championships in New York. This was the first event of its kind, and the Northwestern program returned to take first prize time and again. The program described here, CHESS 4.5, was the 1977 Minnesota Open Champion!

The Northwestern program dominated computer chess almost completely during the 1970's, but in 1974, in Stockholm, the Soviet entry KAISSA won the first World Computer Chess Championship, pushing CHESS 4.0 into second place. Some of the tree searching techniques used by the Soviet programmers are described in their paper which forms section 4.2. When the Northwestern reign finally ended, in 1980, a new leader emerged in the form of a special purpose chess machine named BELLE. This is described by its authors in section 7.4. At the time of publication of this compendium, the computer chess world is at the end of yet another era, that of the CRAY BLITZ program which executes on the world's fastest commercially available computers. CRAY BLITZ won the 1983 and 1986 World Championship tournaments.

# 2.1 Chess Playing Programs and the Problem of Complexity

**Allen Newell, Cliff Shaw and Herbert Simon**

The paper from which this is an excerpt was first published in the IBM *Journal of Research and Development,* volume 2, October 1958, pp. 320-335. Reprinted by permission of IBM.
Copyright 1958 by International Business Machines Corporation.

### The Los Alamos Program

In 1956 a group at Los Alamos programmed MANIAC-I to play chess (Kister et al, 1957). The Los Alamos program is an almost perfect example of the type of system specified by Shannon. As shown in the Table, all alternatives were considered: all continuations were explored to a depth of two moves (i.e., two moves for Black and two for White): the static evaluation function consisted of a sum of material and mobility measures; the values were integrated by a minimax procedure,* and the best alternative in terms of the effective value was chosen for the move.

In order to carry out the computation within reasonable time limits, a major concession was required. Instead of the normal chess board of eight squares by eight squares, they used a reduced board, six squares by six squares. They eliminated the Bishops and all special chess moves: castling, two-square Pawn moves in the opening, and *en passant* captures.

The result? Again the program is a weak player, but now one that is capable of beating a weak human player, as the machine demonstrated in one of its three games. It is capable of serious blunders, a common characteristic, also, of weak human play.

Since this is our first example of actual play on a computer, it is worth looking a bit at the programming and machine problems. In a normal 8 × 8 game of chess there are about 30 legal alternatives at each move, on the average, thus looking two moves ahead brings $30^4$ continuations, about 800,000, into consideration. In the reduced 6 × 6 game, the designers estimate the average number of alternatives at about 20, giving a total of about 160,000 continuations per move. Even with this reduction of five to one, there are still a lot of positions to be looked at. By comparison, the best evidence suggests that a human player considers considerably less than 100 positions in the analysis of a move (De Groot, 1946). The Los Alamos program was able to make a move in about 12 minutes on the average. To do this the code had to be very simple and straightforward. This can be seen by the size of the program – only 600 words. In a sense, the machine barely glanced at each position it evaluated. The two measures in the evaluation function are obtained directly from the process of looking at continuations; changes in material are noticed if the moves are captures, and the mobility score for a position is equal to the number of new positions to which it leads – hence is computed almost without effort when exploring all continuations.

The Los Alamos program tests the limits of si mplification in the direction of minimizing the amount of information required for each position evaluated, just as Turing's program tests the limits in the direction of minimizing the amount of exploration of continuations. These programs, especially the Los Alamos one, provide real anchor points. They show that, with very little in the way of complexity, we have at least entered the arena of human play – we can beat a beginner.

### Bernstein's Program

Over the last two years Alex Bernstein, a chess player and programmer at IBM, has constructed a chess-playing program for the IBM 704 (for the full 8 × 8 board)

* The minimax procedure was a slight modification of the one described earlier, in that the mobility score for each of the intermediate positions was added in.

(Bernstein et al, 1958a; Bernstein et al, 1958b). This program has been in partial operation for the last six months, and has now played one full game plus a number of shorter sequences. It, too, is in the Shannon tradition, but it takes an extremely important step in the direction of greater sophistication: only a fraction of the legal alternatives and continuations are considered. There is a series of subroutines, which we can call plausible move generators, that propose the moves to be considered. Each of these generators is related to some feature of the game: King safety, development, defending own men, attacking opponent's men, and so on. The programme considers at most seven alternatives, which are obtained by operating the generators in priority order, the most important first, until the seven are accumulated.

The program explores continuations two moves ahead, just as the Los Alamos program did. However, it uses the plausible move generators at each stage, so that, at most, 7 direct continuations are considered from any given position. For its evaluation function it uses the ratio of two sums, one for White and one for Black. Each sum consists of four weighted factors; material, King defence, area control, and mobility. The program minimaxes and chooses the alternative with the greatest effective value.

The program's play is uneven. Blind spots occur that are very striking; on the other hand it sometimes plays very well for a series of moves. It has never beaten anyone, as far as we know; in one full game it played it was beaten by a good player, (Bernstein et al, 1958a) and it has never been pitted against weak players to establish how good it is.

Bernstein's program gives us our first information about radical selectivity, in move generation and analysis. At 7 moves per position, it examines only 2,500 final positions two moves deep, out of about 800,000 legal continuations. That it still plays at all tolerably with a reduction in search by a factor of 300 implies that the selection mechanism is fairly effective. Of course, the selections follow the common and tested lore of the chess world; so that the significance of the reduction lies in showing that this lore is being successfully captured in mechanism. On the other hand, such radical selection should give the program a strong proclivity to overlook moves and consequences. The selective mechanisms in Bernstein's program have none of the checks and balances that exist in human selection on the chess board. And this is what we find. For example, in one situation a Bishop was successively attacked by three Pawns, each time retreating one square to a post where the next Pawn could attack it. The program remained oblivious to this possibility since the successive Pawn pushes that attacked the Bishop were never proposed as plausible moves by the generators. But this is nothing to be unhappy about. Any particular difficulty is removable: in the case of the Bishop, by adding another move generator responsive to another feature of the board. This kind of error correction is precisely how the body of practical knowledge about chess programs and chess play will accumulate, gradually teaching us the right kinds of selectivity.

Every increase in sophistication of performance is paid for by an increase in the complexity of the program. The move generators and the components of the static evaluation require varied and diverse information about each position. This implies both more program and more computing time per position than with the Los Alamos program. From Table 1, we observe that Bernstein's program takes 7,000 words, the Los Alamos program only 600 words; a factor of about 10. As for time per position, both programs take about the same time to produce a move – 8 and 12 minutes respectively. Since the increase in problem size of the 8 × 8 board over the 6 × 6 board (about 5 to 1) is approximately cancelled by the increase in speed of the IBM 704 over the MANIAC (also about 5 to 1, counting the increased power of the 704 order code), we can say they would both produce moves in the same 8 × 8 game in the same time. Hence the increase in amount of processing per move in Bernstein's program approximately cancels the gain of 300 to 1 in selectivity that this more complex processing achieves. This is so, even though Bernstein's program is coded to attain maximum speed by the use of fixed tables, direct machine coding, and so on.

We have introduced the comparison in order to focus on computing speed versus selectivity as sources of improvement in complex programs. It is not possible,

*Table 1.* **Comparison of current chess programs.**

| | TURING | LOS ALAMOS<br>Kister, Stein, Ulam, Walden, Wells | BERNSTEIN<br>Roberts, Arbuckle, Belsky | NSS<br>Newell, Shaw, Simon |
|---|---|---|---|---|
| **● Vital Statistics** | | | | |
| *Date* | 1951 | 1956 | 1957 | 1958 |
| *Board* | 8 × 8 | 6 × 6 | 8 × 8 | 8 × 8 |
| *Computer* | Hand Simulation | MANIAC-I<br>11,000 ops/sec | IBM 704<br>42,000 ops/sec | RAND JOHNNIAC<br>20,000 ops/sec |
| **● Chess program** | | | | |
| *Alternatives* | All moves | All moves | 7 plausible moves<br>Sequence of move generators | Variable<br>Sequence of move generators |
| *Depth of analysis* | Until dead (exchanges only) | All moves<br>2 moves deep | 7 plausible moves<br>2 moves deep | Until dead<br>Each goal generates moves |
| *Static evaluation* | Numerical<br>Many factors | Numerical<br>Material, Mobility | Numerical<br>Material, Mobility, Area control, King defense | Non-numerical<br>Vector of values<br>Acceptance by goals |
| *Integration of values* | Minimax | Minimax (modified) | Minimax | Minimax |
| *Final choice* | Material dominates<br>Otherwise, best value | Best value | Best value | 1. First acceptable<br>2. Double function |
| **● Programming** | | | | |
| *Language* | | Machine code | Machine code | IPL-IV, interpretive |
| *Data scheme* | | Single board<br>No records | Single board<br>Centralized tables<br>Recompute | Single board<br>Decentralized<br>List structure<br>Recompute |
| *Time* | Minutes | 12 min/move | 8 min/move | 1-10 hrs/move (est.) |
| *Space* | | 600 words | 7,000 words | Now 6,000 words, est 16,000 |
| **● Results** | | | | |
| *Experience* | 1 game | 3 games<br>(no longer exists) | 2 games | 0 games<br>Some hand simulation |
| *Description* | Loses to weak player<br>Aimless<br>Subtleties of evaluation lost | Beats weak player<br>Equivalent to human with 20 games experience | Passable amateur<br>Blind spots<br>Positional | Good in spots (opening)<br>No aggressive goals yet |

unfortunately, to compare the two programs in performance level except very crudely. We should compare an 8 × 8 version of the Los Alamos program with the Bernstein program, and we also need more games with each to provide reliable estimates of performance. Since the 8 × 8 version of the Los Alamos program will be better than the 6 × 6, compared to human play, let us assume for purposes of argument that the Los Alamos and Bernstein programs are roughly comparable in performance. To a rough approximation, then, we have two programs that achieve the same quality of performance with the same total effort by two different routes; the Los Alamos program by using no selectivity and being very fast, and the Bernstein program by using a large amount of selectivity and taking much more effort per position examined in order to make a selection.

The point we wish to make is that this equality is an accident; that selectivity is a very powerful device and speed a very weak device for improving the performance of complex programs. For instance, suppose both the Los Alamos and the Bernstein programs were to explore three moves deep instead of two as they now do.Then the Los Alamos program would take about 1,000 times ($30^2$) as long as now to make a move, whereas Bernstein's program would take about 50 times as long ($7^2$), the latter gaining a factor of 20 in the total computing effort required per move. The significant feature of chess is the exponential growth of positions to be considered with depth of analysis. As analysis deepens, greater computing effort per position soon pays for itself, since it slows the growth in number of positions to be considered. The comparison of the two programs at a greater depth is relevant since the natural mode of improvement of the Los Alamos program is to increase the speed enough to allow explorations three moves deep. Furthermore, attempts to introduce selectivity in the Los Alamos program will be extremely costly relative to the cost of additional selectivity in the Bernstein program.

One more calculation might be useful to emphasize the value of heuristics that eliminate branches to be explored. Suppose we had a branching tree in which our program was exploring $n$ moves deep, and let this tree have four branches at each node. If we could double the speed of the program – that is, consider twice as many positions for the same total effort – then this improvement would let us look half a move deeper ($n+\frac{1}{2}$).

If, on the other hand, we could double the selectivity – that is, only consider two of the four branches at each node, then we could look twice as deep ($2n$). It is clear that we could afford to pay an apparently high computing cost per position to achieve this selectivity.

To summarize. Bernstein's program introduces both sophistication and complication to the chess program. Although in some respects – e.g., depth of analysis – it still uses simple uniform rules, in selecting moves to be considered it introduces a set of powerful heuristics which are taken from successful chess practice, and drastically reduce the number of moves considered at each position.

## Newell, Shaw, and Simon Program

Although our own work on chess started in 1955 (Newell, 1955), it took a prolonged vacation during a period in which we were developing programs that discover proofs for theorems in symbolic logic. (Newell et al., 1957; Newell et al., 1956). In a fundamental sense,proving theorems and playing chess involve the same problem; reasoning with heuristics that select fruitful paths of exploration in a space of possibilities that grows exponentially. The same dilemmas of speed versus selection and uniformity versus sophistication exist in both problem domains. Likewise, programming costs attendant upon complexity seem similar for both. So we have recently returned to the chess programming problem equipped with ideas derived from the work on logic.

The historical antecedents of our own work are somewhat different from those of the other investigators we have mentioned. We have been primarily concerned with describing and understanding human thinking and decision processes (Newell et al., 1958a). However, both for chess players and for chess programmers, the structure of the task dictates in considerable part the approach taken, and our current program can be described in the same terms we have used for the others. Most of the positive features of the earlier programs are clearly discernible: The basic factorization

| | GOAL SPECIFICATION | MOVE GENERATOR | STATIC EVALUATION | ANALYSIS GENERATOR |
|---|---|---|---|---|
| GOALS | KING SAFETY | | | |
| | MATERIAL BALANCE | | | |
| | CENTER CONTROL | | | |
| | DEVELOPMENT | | | |
| | KING-SIDE ATTACK | | | |
| | PROMOTION | | | |

*Figure 3 Basic organisation of NSS chess program.*

introduced by Shannon; Turing's concept of a dead position; and the move generators, associated with features of the chess situation, used by Bernstein. Perhaps the only common characteristic of the other programs that is strikingly absent from ours – and from human thinking also, we believe – is the use of numerical additive evaluation functions to compare alternatives.

*Basic organization*

Figure 3 shows the two-way classification in terms of which the program is organized. There is a set of goals, each of which corresponds to some feature of the chess situation – King safety, material balance, center control, and so on. Each goal has associated with it a collection of processes, corresponding to the categories outlined by Shannon: a move generator, a static evaluation routine, and a move generator for analysis. The routine for integrating the static evaluations into an effective value for a proposed move, and the final choice procedure are both common routines for the whole program, and therefore are not present in each separate component.

*Goals*

The goals form a basic set of modules out of which the program is constructed. The goals are independent; any of them can be added to the program or removed without affecting the feasibility of the remaining goals. At the beginning of each move a preliminary analysis establishes that a given chess situation (a "state") obtains, and this chess situation evokes a set of goals appropriate to it. The goal specification routines shown for each goal in Fig.3 provide information that is used in this initial selection of goals. The goals are put on a list with the most crucial ones first. This goal then controls the remainder of the processing: the selection of alternatives, the continuations to be explored, the static evaluation, and the final choice procedure.

What kind of game the program will play clearly depends on what goals are available to it and chosen by it for any particular move. One purpose of this modular construction is to provide flexibility over the course of the game in the kinds of considerations the program spends its effort upon. For example, the goal of denying stalemate to the opponent is relevant only in certain endgame situations where the opponent is on the defensive and the King is in a constrained position. Another purpose of the modular construction is to give us a flexible tool for investigating chess programs – so that entirely new considerations can be added to an already complex but operational program.

*Move generation*

The move generator associated with each goal proposes alternative moves relevant to that goal. These move generators carry the burden of finding positive reasons for doing things. Thus, only the center-control generator will propose d2-d4 as a good move in the opening; only the material-balance generator will propose moving out of danger a piece that is *en prise*. These move generators correspond to the move generators in Bernstein's program, except that here they are used exclusively to generate alternative moves and are not used to generate the continuations that are explored in the course of analysing a move. In Bernstein's program – and *a fortiori* in the Los Alamos program – identical generators are used both to find a set of alternative moves from which the final choice of next move is made, and also to find the continuations that must be explored to assess the consequences of reaching a given position. In our program the latter function is performed by a separate set of analysis generators.

*Evaluation*

Each move proposed by a move generator is assigned a value by an analysis procedure. We said above that the move generators have the responsibility for finding positive reasons for making moves. Correspondingly, the analysis procedure is concerned only with the acceptability of a move once it has been generated. A generator proposes; the analysis procedure disposes.

The value assigned to a move is obtained from a series of evaluations, one for each goal. The value is a vector, if you like to think of it that way, except that it does not necessarily have the same components throughout the chess game, since the components derive from the basic list of goals that is constructed from the position at the beginning of each move. Each component expresses acceptability or unacceptability of a position from the viewpoint of the goal corresponding to that component. Thus, the material-balance goal would assess only the loss or gain of material; the development goal, the relative gain or loss of *tempi;* the Pawn structure goal, the doubling and isolation of Pawns; and so on. The value for a component is in some cases a number – e.g., in the material-balance goal where we use conventional piece values: 9 for a Queen, 5 for a Rook, and so on. In other cases the component value is dichotomous, simply designating the presence or absence of some property, like the blocking of a move or the doubling of a Pawn.

As in the other chess programs, our analysis procedure consists of three parts; exploring continuations to some depth, forming static evaluations and integrating these to establish an effective value for the move. By a process that we will describe later, the analysis move generators associated with the goals determine what branches will be explored from each position reached. At the final position of each continuation, a value is assigned using the static evaluation routines of each goal to provide the component values. The effective value for a proposed move is obtained by minimaxing on these final static values. Minimaxing seems especially appropriate for an analysis procedure that is inherently conservative, such as an acceptance test.

To be able to minimax, it must be possible to compare any two values and decide which is preferable, or whether they are of equal value. For values of the kind we are using, there must be a complete ordering on the vectors that determine them. Further, this ordering must allow variation in the size and composition of the goal list. We use a lexicographic ordering: Each component value is completely ordered within itself; and higher priority values completely dominate lower priority values, as determined by the order of goals on the goal list. To compare two values, then, the first components are compared. If one of these is preferable to the other, this determines the preference for the entire value. If the two components are equal, then the second pair of components is compared. If these are unequal in value, they determine the preference for the entire value, otherwise the next components are compared, and so on.

*Final choice*

It is still necessary to select the move to be played from the alternative moves, given the values assigned to them by the analysis procedure. In the other programs the final choice procedure was simply an extension of the minimax; choose the one with the highest value. Its obviousness rests on the assumption that the set of alternatives to be considered is a fixed set. If this assumption is relaxed, by generating alternatives sequentially, then other procedures are possible. The simplest, and the one we are currently using, is to set an acceptance level as final criterion and simply take the first acceptable move. The executive routine proceeds down the goal list, activating the move generators of the goals in order of priority, so that important moves are considered first. The executive saves the best move that has been found up to any given moment, and if no moves reach the specified level of acceptability, it makes the best move that was found.

Another possible final choice procedure is to search for an acceptable move that has a double function – that is, a move that is proposed by more than one generator as having a positive effect. With this plan, the executive proceeds down the list of goals in order of priority. After finding an acceptable move, it actuates the rest of the generators to see if the move will be proposed a second time. If not, it works from the list of unevaluated moves just obtained

to see if any move proposed twice is acceptable. If not, it takes the first acceptable move or the best if none has proved acceptable. This type of executive has considerable plausibility, since the concept of multiple-function plays an important role in the chess literature.

Yet a third variation in the final choice procedure is to divide the goals into two lists. The first list contains all the features that should normally be attended to; the second list contains features that are rare in occurrence but either very good or very bad if they do occur. On this second list would be goals that relate to sacrificial combinations, hidden forks or pins that are two moves away, and so on. The executive finds an acceptable move with the first, normal list. Then the rest of the available time is spent looking for various rare consequences derived from the second list.

*Analysis*

In describing the basic organization of the program we skipped over the detailed mechanism for exploring continuations, simply assuming that certain continuations were explored, the static values computed, and the effective value obtained by minimaxing. But it is clear that the exact mechanisms are very important. The analysis move generators are the main agents of selectivity in the program: They determine for each position arrived at in the analysis just which further branches must be explored, hence the average number of branches in the exploration tree and its average depth. The move generators for the alternatives and the final choice procedure also affect the amount of exploration by determining what moves are considered. But their selection operates only once per move. Whereas the selectivity of the analysis generators varies geometrically with the average depth of analysis.

The exploration of continuations is based on a generalization of Turing's concept of a dead position. Recall that Turing applied this notion to exchanges, arguing that it made sense to count material on the board until all exchanges that were to take place had been carried out.

We apply the same notion to each feature of the board: The static evaluation of a goal is meaningful only if the position being evaluated is "dead" with respect to the feature associated with that goal — that is, only if no moves are likely to be made that could radically alter that component static value. The analysis-move generators for each goal determine for any position they are applied to whether the position is dead with respect to their goal; if not, they generate the moves that are both plausible and might seriously affect the static value of the goal. Thus the selection of continuations to be explored is dictated by the search for a position that is dead with respect to all the goals, so that, finally, a static evaluation can be made. Both the number of branches from each position and the depth of the exploration are controlled in this way. Placid situations will produce search trees containing only a handful of positions; complicated middle game situations will produce much larger ones.

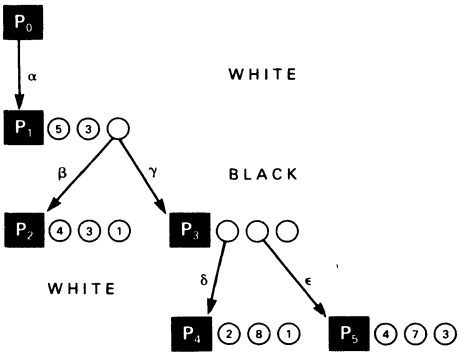



*Figure 4* **Analysis**

To make the mechanics of the analysis clearer, Fig.4 gives a schematic example of a situation. $P_0$ is the initial position from which White, the machine, must make a move. The arrow, $\alpha$, leading to $P_1$ represents an alternative proposed by some move generator. The move is made internally (i.e., "considered"), yielding position $P_1$, and the analysis procedure must then obtain the value of $P_1$, which will become the value imputed to the proposed alternative, $\alpha$. Taking each goal from the goal list in turn, an attempt is made to produce a static evaluation. For $P_1$ this attempt is successful for the first and second components, yielding values of 5 and 3 respectively. (Numbers are used for values throughout this example to keep the picture simple; in reality, various sets of ordered symbols are used, their exact structure depending on the nature of the

computation.) However, the third component does not find the position dead, and generates two moves $\beta$ and $\gamma$. The first, $\beta$, is considered, leading to $P_2$, and an attempt is made to produce a static evaluation of it. This proceeds just as with $P_1$, except that this time all components find the position dead and the static value (4, 3, 1) is obtained. Then the second move, $\gamma$, from $P_1$ is considered, leading to $P_3$. The attempt to produce a static value for $P_3$ runs into difficulties with the first component, which generates one move, $\delta$, to resolve the instability of $P_3$ with respect to its feature. This move leads to $P_4$ which is evaluable, having the value (2, 8, 1). However, the second component also finds $P_3$ not dead and generates a single move, $\varepsilon$, leading to $P_5$. This is also evaluable, having the value (4, 7, 3). The third component finds $P_3$ dead and therefore contributes no additional moves. Thus the exploration comes to an end with all terminal positions yielding complete static values. Since it is White's move at $P_3$, White will choose the move with the highest value. This is $\varepsilon$, the move to $P_5$, with a value of (4, 7, 3) (the first component dominates). The value of this move is the effective value assigned to $P_3$. Black now has a choice between the move, $\beta$, to $P_2$, yielding (4, 3, 1) and the move, $\gamma$, to $P_3$, yielding (4, 7, 3). Since Black is minimizing, he will choose $\beta$. This yields (4, 3, 1) as the effective value of the alternative, $\alpha$, that leads to $P_1$, and the end of the analysis.

The minimaxing operation is conducted concurrently with the generation of branches. Thus if $P_5$, which has a value of (4, 7, 3), had been generated prior to $P_4$ no further moves would have been generated from $P_3$, since it is already apparent that Black will prefer $P_2$ to $P_3$. The value of $P_3$ is at least as great as the value of $P_5$, since it is White's move and he will maximize.

This analysis procedure is not a simple one, either conceptually or technically. There are a number of possible ways to terminate search and reach an effective evaluation. There is no built-in rule that guarantees that the search will converge; the success depends heavily on the ability to evaluate statistically. The more numerous the situations that can be recognized as having a certain value without having to generate continuations, the more rapidly the search will terminate. The number of plausible moves that affect the value is also of consequence, as we discussed in connection with Bernstein's program, but there are limits beyond which this cannot be reduced. For example, suppose that a position is not dead with respect to Material Balance and that one of the machine's pieces is attacked. Then it can try to (a) take the attacker, (b) add a defender, (c) move the attacked piece, (d) pin the defender, (e) interpose a man between the attacker and the attacked, or (f) launch a counter-attack. Alternatives of each of these types must be sought and tried – they are all plausible and may radically affect the material balance.

As an example of the heuristics involved in achieving a static evaluation, imagine that the above situation occurred after several moves of an exploration, and that the machine was already a Pawn down from the early part of the continuation. Then, being on the defensive implies a very remote chance of recovering the Pawn. Consequently, a negative value of at least a Pawn can be assigned to the position statically. This is usually enough in connection with concurrent minimaxing to eliminate the continuation from further consideration.

*Summary*

Let us summarize our entire program. It is organized in terms of a set of goals: these are conceptual units of chess – King safety, passed Pawns, and so on. Each goal has several routines associated with it:

1. A routine that specifies the goal in terms of the given position;
2. A move generator that finds moves positively related to carrying out the goal;
3. A procedure for making a static evaluation of any position with respect to the goal, which essentially measures acceptability;
4. An analysis move generator that finds the continuations required to resolve a situation into dead positions.

The alternative moves come from the move generators, considered in the order of priority of their respective goals. Each move, when it is generated, is subjected to an analysis. This analysis generates an exploration of the continuations following from the move until dead positions are reached and static evaluations computed for them. The static evaluations are

compared, using minimax as an inference procedure, so that an effective value is eventually produced for each alternative. The final choice procedure can rest on any of several criteria: for instance, choosing the first move generated that has an effective value greater than a given norm.

*Examples of goals*

In this section we will give two examples of goals and their various components to illustrate the type of program we are constructing. The first example is the center-control goal:

**Center control**

*Specification*

Goal is always operative unless there are no more center Pawns to be moved to the fourth rank.

*Move generator*

1. Move d2-d4 e2-e4 (primary moves).
2. Prevent the opponent from making his primary moves.
3. Prepare your own primary moves:
   a) Add a defender to d4 or e4 square.
   b) Eliminate a block to moving d♙ or e♙

*Static evaluation*

Count the number of blocks to making the primary moves.

*Analysis move generators*

None; static evaluation is always possible.

To interpret this a little: Goals are proposed in terms of the general situation – e.g. for the opening game. The list of goals is made up for a position by applying, in turn, the speciification of each of the potential goals. Whether any particular goal is declared relevant or irrelevant to the position depends on whether or not the position meets its specification. For Center Control, no special information need be gathered, but the goal is declared irrelevant if the center Pawns have already been moved to the fourth rank or beyond.

The most important part of the center-control program is its move generator. The generator is concerned with two primary moves: d2-d4 and e2-e4. It will propose these moves, if they are legal, and it is the responsibility of the analysis procedures for all the goals) to reject the moves if there is anything wrong with them – e.g. if the Pawns will be taken when moved. So, after 1 d2-d4 d7-d5, the center-control move generator will propose 2 e2-e4, but (as we shall see) the evaluation routine of the material balance goal will reject this move because of the loss of material that would result from 2.....d5×e4. The center-control generator will have nothing to do with tracing out these consequences.

If the primary moves cannot be made, the center-control move generator has two choices: to prepare them, or to prevent the opponent from making his primary moves. The program's style of play will depend very much on whether prevention has priority over preparation (as it does in our description of the generator above), or vice versa. The ordering we have proposed, which puts prevention first, probably produces more aggressive and slightly better opening play than the reverse ordering. Similarly, the style of play depends on whether the d Pawn or the e Pawn is considered first.

The move generator approaches the subgoal of preventing the opponent's primary moves (whenever this subgoal is evoked) in the following way. It first determines whether the opponent can make one of these moves by trying the move and then obtaining an evaluation of it from the opponent's viewpoint. If one or both of the primary moves are not rejected, preventative moves will serve some purpose. Under these conditions, the center-control move generator will generate them by finding moves that bring another attacker to bear on the opponent's e5 and d5 squares or that pin a defender of one of these squares. Among the moves this generator will normally propose are ♘g1-f3 and f2-f4.

The move generator approaches the subgoal of preparing its own primary moves by first determining why the moves cannot be made without preparation – that is, whether the Pawn is blocked from moving by a friendly piece, or whether the fourth rank square is unsafe for the Pawn. In the former case, the generator proposes moves for the blocking piece; in the latter case, it finds moves that will add defenders to the

fourth rank square, drive away or pin attackers, and so on.

So much for the center-control generators. The task of the evaluation routine for the center-control goal is essentially negative – to assure that moves, proposed by some other goal, will not be made that jeopardize control of the centre. The possibility is simply ignored that a move generator for some other goal will inadvertently provide a move that contributes to center control. Hence, the static evaluation for Center Control is only concerned that moves not be made that interfere with e2-e4 and d2-d4. A typical example of a move that the center-control evaluation routine is prepared to reject is ♗f1-d3 or ♗c1-e3 before the respective center Pawns have been moved.

The second example of a goal is Material Balance. This is a much more extensive and complicated goal than Center Control, and handles all questions about gain and loss of material in the immediate situation. It does not consider threats like pins and forks, where the actual exchange is still a move away; other goals must take care of these. Both the negative and positive aspects of material must be included in a single goal, since they compensate directly for each other, and material must often be spent to gain material.

**Material balance**

*Specification*

A list of exchanges on squares occupied by own men, and a list of exchanges on squares occupied by opponent's men. For each exchange square there is listed the target man, the list of attackers, and the list of defenders (including e.g., both Rooks if they are doubled on the appropriate rank or file). For each exchange square a static exchange value is computed by playing out the exchange with all the attackers and defenders assuming no indirect consequences like pins, discovered attacks, etc. Exchange squares are listed in order of static exchange value, largest negative value first. Squares with positive values for the defender are dropped from the list. At the same time a list of all pinned men is generated.

*Move generator*

Starting with the exchange squares at the top of the list, appropriate moves are generated. If the most important exchange square is occupied by the opponent, captures by attacking pieces are proposed, the least valuable attacker being tried first. If the move is rejected because the attacker is pinned, the next attacker is tried. If the move is rejected for another reason, the possibility of exchange on this square is abandoned, and the next exchange square examined.

If the exchange square under examination is occupied by the program's own piece, a whole series of possible moves is generated:

a) Try "no move" to see if attack is damaging.
b) Capture the attacker.
c) Add a defender not employed in another defence.
d) Move the attacked piece.
e) Interpose a man between the attacker and the target; but not a man employed elsewhere, and not if the interposer will be captured.
f) Pin the attacker with a man not employed elsewhere and not capturable by the attacker.

*Static evaluation*

For each exchange square, add the values of own men and subtract the values of opponent's men. Use conventional values: Q-9, R-5, B-N-3, P-1.

*Move generators toward dead positions*

A position is dead for this goal only if there are no exchanges – that is, if the specification list defined above is empty. Then a static evaluation can be made. Otherwise, the various kinds of moves defined under the move generator are made to resolve the exchanges. However, various additional qualifications are introduced to reduce the number of continuations examined. For example, if in a particular exchange material has already been lost and a man is still under attack, the position is treated as dead, since it is unlikely that the loss will be recovered. When a dead position is reached, the static evaluation is used to find a value for the position.

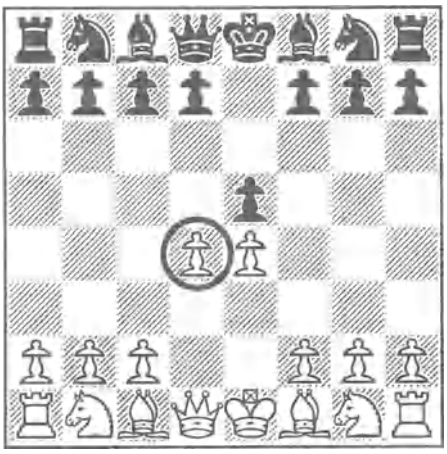
It is impossible to provide here more than a sketchy picture of the heuristics contained in this one goal. It should be obvious

from this brief description that there are a lot of them, and they incorporate a number of implicit assumptions about what is important and what isn't, on the chess board.

*Performance of the program*

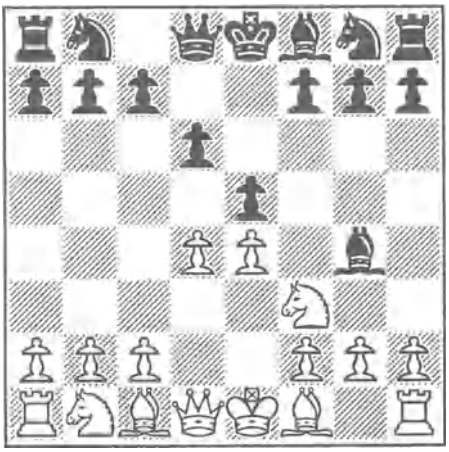
We cannot say very much about the behaviour of the program. It was coded this spring and is not yet fully debugged. Only two goals have been coded: Material Balance and Center Control. Development is fully defined as well as a Pawn structure goal sufficient for the opening, where it is primarily to prevent undesirable structures like doubled Pawns. These four goals – Material Balance, Center Control, Development and Pawn Structure – in this order seem an appropriate set for the first phase of the opening game. Several others – King safety, Serious Threats, and Gambits – need to be added for full opening play. The serious threats goal could be limited initially to forks and pins.

*Figure 5*



We have done considerable hand simulation with the program in typical positions. Two examples will show how the goals interact. In Fig.5 the machine is White and the play has been 1 e2-e4, e7-e5. Assuming the goal list mentioned above, the material balance move generator will not propose any moves since there are no exchanges on the board. The center-control generator will propose d2-d4 which is the circled move in the figure. (In the illustration, we assume the center-control move generator has the order of the primary moves reversed from the order described earlier.) This move is rejected – as it should be – and it is instructive to see why. The move is proposed for analysis. Material Balance does not find the position dead, since there is an exchange, and generates Black's move, 2 ....., e5×d4. The resulting position is still not dead, and 3 ♕d1×d4 is generated. The position is now dead for Material Balance, with no gain or loss in material. The first component of the static evaluation is "even". There are obviously no blocks to Pawn moves, so that the center control static value is acceptable. However, the third component, Development, finds the position not dead because there is now an exposed piece, the Queen. It generates replies that both attack the piece and develop – i.e., add a tempo. The move 3 ....., ♘b8-c6 is generated. This forces a Queen move, resulting in loss of a tempo for White. Hence Development rejects the move, 2 d2-d4. (The move 3 ....., ♗f8-c5 would not have sufficed for rejection by Development, since the Bishop could be taken.)

*Figure 6*



The second example, shown in Fig.6, is from a famous game of Morphy against Duke Karl of Brunswick and Count Isouard. Play had proceeded 1 e2-e4, e7-e5; 2 ♘g1-f3, d7-d6; 3 d2-d4. Suppose the machine is Black in this position. The move 3 ....., ♗c8-g4 is proposed by Material Balance to deal with the exchange that threatens Black with the loss of a Pawn.

This is the move made by the Duke and Count. The analysis proceeds by 4 d4×e5, d6×e5. This opens up a new exchange possibility with the Queens, which is tried: 5 ♕d1×d8+, ♔e8×d8; 6 ♘f3×e5. Thus the Pawn is lost in this continuation. Hence, alternative moves are considered at Black's nearest option, which is move 4, since there are no alternative ways of recapturing the Queen at move 5. The capture of White's Knight is possible, so we get: 4 ...., ♗g4×f3; 5 g2×f3, d6×e5; 6 ♕d1×d8+, ♔e8×d8. This position is rejected by Development since the forced King move loses Black his castling privilege, and this loss affects the tempo count. This is a sufficient reason to reject the move 3 ...., ♗c8-g4, without even examining the stronger continuation, 5 ♕d1×f3, that Morphy as White chose. In our program, 5 g2×f3 is generated before 5 ♕d1×f3. Either reply shows that 3 .... ♗c8-g4 is unsound.

One purpose of these examples is to illustrate a heuristic for constructing chess programs that we incline to rather strongly. We wish not only to have the program make good moves, but to do so for the right reasons. The chess commmentary above is not untypical of human analysis. It also represents rather closely the analysis made by the program. We think this is sound design philosophy in constructing complex programs. To take another example: the four-goal opening program will not make sacrifices, and conversely, will always accept gambits. The existing program is unable to balance material against positioned advantage. The way to make the program take account of sacrifices is to introduce an additional goal having to do with them explicitly. The corresponding heuristic for a human chess player is: don't make sacrifices until you understand what a sacrifice is. Stated in still another way, part of the success of human play depends on the emergence of the concept of the center and the notion of what it means to control the center. One should not expect the equivalent of such a concept simply to emerge from computation based on quite other features of the position.

*Programming*

The program we have been describing is extremely complicated. Almost all elements of the original framework put forward by Shannon, which were handled initially by simply uniform rules, have been made variable, and dependent on rather complicated considerations. Many special and highly particular heuristics are used to select moves and decide on evaluations. The program can be expected to be much larger, more intricate, and to require much more processing per position considered than even the Bernstein program.

In the introduction to this paper we remarked on the close connection between complexity and communication. Processes as complex as the Los Alamos program are unthinkable without languages like current machine codes in which to specify them. The Bernstein program is already a very complicated program in machine code; it involved a great deal of coding effort and parts of it required very sophisticated coding techniques. Our own program is already beyond the reach of direct machine coding: it requires a more powerful language.

In connection with the work on theorem-proving programs we have been developing a series of languages, called information processing languages (IPL's). (Newell and Shaw 1957) The current chess program is coded in one of them, IPL-IV. An information processing language is an interpretative pseudo-code – that is, there exists a program in JOHNNIAC machine code that is capable of interpreting a program in IPL and executing it. When operating, JOHNNIAC contains both the machine code and the IPL code.

It is not possible to give in this paper a description of IPL-IV or of the programming techniques involved in constructing the chess program. Basically IPL is designed to manipulate lists, and to allow extemely complicated structures of lists to be built up during the execution of a program without incurring intolerable problems of memory assignment and program planning. It allows unlimited hierarchies of sub-routines to be easily defined, and permits recursive definitions of routines. As it stands – that is, prior to coding a particular problem – it is independent of subject matter (although biased towards list manipulation in the same sense that algebraic compilers are biased towards numerical evaluation of algebraic expressions). To code chess, a complete "chess vocabulary" is built up from definitions in IPL. This vocabulary consists of a set of processes for expressing

basic concepts in chess: tests of whether a man bears on another man, or whether two men are on the same diagonal; processes for finding the direction between two men, or the first man in a given direction from another; and processes that express iterations over all men of a given type, or over all squares of a given rank. There are about 100 terms in this basic process vocabulary. The final chess program, as we have been describing it in this paper, is largely coded in terms of the chess vocabulary. Thus there are four language "levels" in the chess program: JOHNNIAC machine code, general IPL, basic chess vocabulary, and finally the chess program itself.

We can now make a rough assessment of the size and complexity of this program in comparison with the other programs. The table indicates that the program now consists of 6000 words and will probably increase to 16000. The upper bound is dictated by the size of the JOHNNIAC drum and the fact that JOHNNIAC has no tapes. In terms of the pyramiding structure described above, this program is already much larger than Bernstein's, although it is difficult to estimate the "expansion" factor involved in converting IPL to machine code. (For one thing, it is not clear how an "equivalent" machine coded program would be organized.) However, only about 1000 words of our program are in machine code, and 3000 words are IPL programs, some of which are as many as ten definitional steps removed from machine code. Further, all 12000 words on the drum will be IPL program; no additional data or machine code are planned.

The estimated time per move, as shown in Table 1, is from one to ten hours, although moves in very placid situations like the opening will take only a few minutes. Even taking into account the differences in speed between the 704 and JOHNNIAC, our program still appears to be at least ten times slower than Bernstein's. This gap reflects partly the mismatch between current computers and computers constructed to do efficiently the kind of information processing required in chess. (Shaw et al, 1958). To use an interpretive code, such as IPL, is in essence to simulate an "IPL computer" with a current computer. A large price has to be paid in computing effort for this simulation over and above the computing effort for the chess program itself. However, this gap also reflects the difficulty of specifying complex processes; we have not been able to write these programs and attend closely to the efficiency issue at the same time.

On both counts we have felt it important to explore the kind of languages and programming techniques appropriate to the task of specifying complex programs, and to ignore for the time being the costs we were incurring.

## Conclusion

We have now completed our survey of attempts to program computers to play chess. There is clearly evident in this succession of efforts a steady development toward the use of more complex programs and more and more selective heuristics; and toward the use of principles of play similar to those used by human players. Partly, this trend represents – at least in our case – a deliberate attempt to simulate human thought processes. In even larger part, however, it reflects the constraints that the task itself imposes upon any information processing system that undertakes to perform it. We believe that any information processing system – a human, a computer, or any other – that plays chess successfully will use heuristics generically similar to those used by humans.

We are not unmindful of the radical differences between men and machines at the level of componentry. Rather, we are arguing that for tasks that could not be performed at all without very great selectivity – and chess is certainly one of these – the main goal of the program must go to achieve this selection. The higher-level programs involved in accomplishing this will look very much the same whatever processes are going on at more microscopic levels. Nor are we saying that programs will not be adapted to the powerful features of the computing systems that are used – e.g., the high speed and precision of current digital computers, which seems to favour exploring substantial numbers of continuations. However none of the differences known to us – in speed, memory, and so on affect the essential nature of the task; search in a space of exponentially growing possibilities. Hence the adaptations to the idiosyncrasies of particular computers will

all be secondary in importance, although they will certainly exist and may be worth while.

The complexity of heuristic programs requires a more powerful language for communicating with the computer than the language of elementary machine instructions. We have seen that this necessity has already mothered the creation of new information processing languages. But even with these powerful interpretive languages, communication with the machine is difficult and cumbersome. The next step that must be taken is to write programs that will give computers a problem-solving ability in understanding and interpreting instructions that is commensurable with their problem-solving ability in playing chess and solving theorems.

The interpreter that will transform the machine into an adequate student for a human instructor will not be a passive algorithmic translator – as even the most advanced interpreters and compilers are today – but an active, complex, heuristic problem-solving program. As our explorations of heuristic programs for chess playing and other tasks teach us how to build such an interpreter, they will at last enable us to make the transition from the low-level equilibrium at which man-machine communication now rests to the high-level equilibrium that is certainly attainable.

# 2.2 Computer v Chess-Player

**Alex Bernstein and Michael de V. Roberts**

First published in *Scientific American,* volume 198, pp. 96-105. Copyright 1958 by Scientific American, Inc. All rights reserved. Reprinted by permission of W. H. Freeman and Company.

Chess is not only one of the most engaging but also one of the most sophisticated of human activities. The game is so old that we cannot say when or where it was invented; millions of games have been played and thousands of books have been written about it; yet the play is still fresh and forever new. Simple arithmetic tells why. On the average, each move in chess offers a choice of about 30 possibilities, and the average length of a full game is about 40 moves. By this reckoning there are at least $10^{120}$ possible games. To get some idea of what that number means, let us suppose that we had a superfast computing machine which could play a million games a second (a ridiculous supposition). It would take the machine about $10^{108}$ years to play all the possible games!

So no conceivable machine could play a perfect game of chess, examining all possible moves. This is what makes the problem of programming a computer to play chess so intriguing. A present day computing machine, with all its speed of calculation, is about as limited as a human being, on any reasonable time scale, in exploring the likely consequences of a chess move. Since it cannot study all the possibilities, the machine must play the game in human terms – that is, it must detect the strategy and anticipate the judgments of its human opponent. In other words, lacking the omniscience that would enable it to win no matter what its opponent does, it must try to outwit the opponent.

Needless to say, devising a program which would give a machine this property – what amounts to the capacity to think – has proved a very difficult job. The late A.M.Turing, the ingenious British theoretician on thinking machines, was one of the first to try his hand at designing a chess-playing program for a computer, but his machine (MADAM) played a very weak game, made stupid blunders and usually had to resign after a few moves. The problem has interested a number of computer experts in the U. S. (see "A Chess Playing Machine" by Claude E. Shannon; *Scientific American,* February, 1950), and several groups are currently working on chess programs. We want to report here what we believe is the first satisfactory program – one with which the machine plays a game sophisticated enough so that its opponent has to be something more than a novice to beat it. The program was written by four collaborators – the authors of this article, who work for the International Business Machines Corporation, and Timothy Arbuckle and M. A. Belsky of the Service Bureau Corporation. It is designed for the IBM 704, the very rapid digital computer which has performed as many as one billion calculations in a single day in computing the orbit of an artificial satellite.

The program is a set of explicit instructions to the computer on how it must act in each of the specific situations with which it may be confronted. The instructions are given to the machine on a reel of magnetic tape. The operation of the computer is itself fascinating to watch. You sit at the console of the machine with a chessboard in front of you and press the start button. Within four seconds a panel light labelled "Program Stop" lights up on the console, and you now make your choice of black or white: to choose black you flip a switch on the console; if you want white, you simply leave the switch as it is. Suppose you have picked black. To begin the game you press the start button again. The machine now "thinks" about its first move. There is nothing spectacular about this. Some lights flash on the console, but the computer is working so swiftly that it is impossible to say just what these flashes mean. After about eight minutes, the computer prints out its move on

a sheet of paper.

Let us say the machine's (White's) first move is king's pawn to the king's fourth square. The print-out then is W1 P-K4. The machine proceeds to print the chessboard with the positions of the pieces, designating its own by the letter M and its opponent's by the letter O.

Now the "Program Stop" light goes on again and the computer waits for its opponent to reply. You punch your replying move on an IBM card and put this card in a section of the machine which reads it. To signal that it is the machine's turn you press the start button again. The machine prints your move and the new board position and then goes to calculate its second move. If you have made an illegal move, the computer will refuse to accept it, printing out "PLEASE CHECK LAST MOVE." So the game proceeds. At the end of the game, after a mating move or a resignation, the machine prints the score of the game, and to its opponent: "THANK YOU FOR AN INTERESTING GAME."

In explaining the program of instructions to the machine it will be helpful if we start by contrasting it with an ordinary job performed by a computer – say calculating John Doe's pay check. The machine in the latter case simply takes the data – so many dollars for a 44-hour week, so much for overtime at a certain rate, so much deducted for social security and income tax – and quickly computes what the check has to be. There is one, and only one, correct answer. But in a chess game there are only two questions to which absolutely definite and unavoidable answers can be given: "Is this move legal?" and "Is the game over?" To all other questions there are various possible answers, though some may be more acceptable than others. The problem is to equip the machine with a system of evaluating the merits of the alternatives. This, as we have remarked, is what makes the task interesting. If cut-and-dried answers to all possible situations could be worked out by a computer, chess would immediately lose its fascination.

Obviously the machine's first job is to size up the board. The instructions therefore direct it to start by examining the state of the squares. The computer painstakingly and single-mindedly considers square by square, giving the same minute attention to squares of little interest as to those of key importance. It asks about each square whether it is occupied by whose man, whether it is attacked, whether it is defended, whether it can be occupied. The information is summed up in tables compiled by the machine. All this takes about one tenth of a second, which is a long time by computer standards. The computer then proceeds to consider its best move.

Here we reach the most difficult and controversial part of the program, for to find a workable basis for the machine's decisions we must make some hypotheses about how a human being plays chess. To begin with, we have to decide on what basis a human player (or the machine) will select the moves that are to be given serious consideration (full consideration of all possible moves being out of the question). There are two distinct philosophies about this. One is that the player concentrates on the moves that look most plausible in the immediate situation. The other is that the player's approach to the selection is dictated by a grand strategy, and as far as he can he looks for moves which will further his plan. We built our program on the second hypothesis.

Of the various possible moves it might make (usually about 30) the machine selects seven for detailed analysis. It picks these on the basis of eight questions, which it asks in the following order:

1. Am I in check, and if so, can I capture the checking piece, interpose a piece or move away?
2. Are any exchanges possible, and if so, can I gain material by entering upon the exchange, or should I move my man away?
3. If I have not castled, can I do so now?
4. Can I develop a minor piece?
5. Can I occupy an open file?
6. Do I have any men that I can put on the critical squares created by pawn chains?
7. Can I make a pawn move?
8. Can I make a piece move?

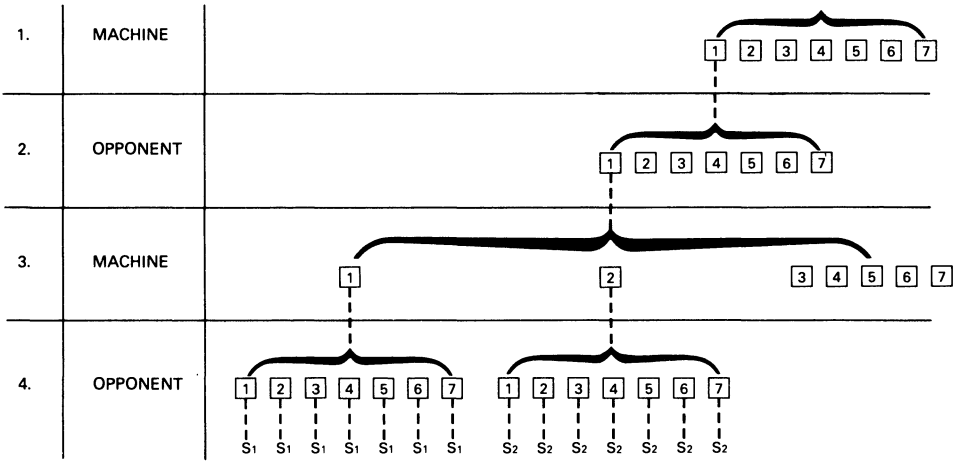
Let us take the opening move for illustration. Examining the initial setup of the board, the machine finds that questions 1, 2 and 3 must be answered "No." The answer to question 4 is "Yes"; the machine notes that it can move either knight and has four possible knight moves ♘g1-h3, ♘b1-a3, ♘g1-f3, ♘b1-c3. To questions 5 and 6 the answer is "No." It is "Yes" to question 7. Any of the eight pawns may be moved, but the instructions tell the machine to give

priority to e2-e4, e2-e3 and d2-d4. These three pawn moves, with the four knight moves, provide the machine with seven moves for study.

It now proceeds to test each of the seven in turn through four moves ahead, considering its opponent's possible replies and its own counter-responses in each case. The machine starts with one of the seven moves and asks itself what it might reply were it the opponent. It generates seven possible replies, on the basis of the questions listed above, and now it takes the first of these and considers its own possible responses. After generating seven plausible responses, it again takes the first of these and in turn generates seven plausible replies by the opponent to this move.

The machine has reached the fourth level: initial move, reply, counter-reply and now the opponent's seven potential responses to its counter-reply *(see diagram below)*. It goes on to examine each of these seven moves to see which one would net the highest value for its opponent. The value, or score, is measured by four considerations: (1) gain of material (a pawn counting as one unit, a knight or bishop three, a rook five and the queen nine); (2) defense of the king; (3) mobility of the pieces (4) control of important squares. After the machine has determined the score for the opponent's best move in level 4, it carries this back as the score for its own move 1 in level 3.

In this manner the machine investigates all the possible sequences of plays, taking each of the seven moves at every level of the "tree," and arrives at scores for all the outcomes at the fourth level. In all, it examines 2,800 possible positions. After this examination, the machine then




The machine makes a move by the procedure suggested in this diagram. First, the machine selects, on the basis of eight questions, its seven most logical moves *(row 1)*. Second, the machine selects its opponent's seven most logical responses to the first of these seven moves *(row 2)*. Third, the machine selects its seven most logical counter-responses to the first of its opponent's responses *(row 3)*. Fourth, the machine selects its opponent's seven most logical responses to the first of its seven counter-responses *(row 4)*. Fifth, the machine scores its opponent's seven responses to the first of its seven counter-responses. ($S_1$). Sixth, the machine selects its opponent's seven most logical responses to the second of its seven counter-responses. Seventh, the machine scores its opponent's responses to the second of its seven counter-responses ($S_2$). The machine continues in this manner until it has examined all moves.

chooses as its first move the one that will lead to the highest score both for itself and for its opponent. It acts, in other words, as if its opponent will make his best possible moves within the limits it is programmed to explore.

These limits – four half-moves ahead with seven choices at each step – are dictated by the time factor. It takes the machine close to eight minutes to decide on each move in most cases. If it had to weigh eight plausible moves instead of seven at each level, it would take about 15 minutes for a move. If it carried the examination through to one more level ahead, a single move would take some six and a half hours. So the present program is considered about the limit for a machine operating at the speed of the IBM 704.

How does the machine make out with this program? In the first place, the machine is never absent-minded. It makes no blatant blunders such as letting a piece be caught *en prise,* as every chess master has done at some time or other. When its opponent is careless enough to expose a piece, the machine takes instant advantage of the opportunity to capture it. Secondly, in its choice of individual moves the machine often plays like a master, making what an expert would consider the only satisfactory move. Thirdly, the machine is certainly not in the master class in the play of a complete game.

Opponent

Machine

A masterly move was made out of this position by the machine. The move was ♕d1-e2. Experts would consider this the only satisfactory move.

A typical game played by the machine against a skillful opponent is illustrated here. We have deliberately chosen a game which the machine lost, because we want to emphasize the point that a machine is not infallible and also because it is more instructive to watch the computer lose than to watch it win. The machine's opening moves in this game are quite acceptable. But by middle game the machine betrays its chief weakness: namely, a heavy bias toward moving attacked pieces rather than defending them (a weakness which could be corrected only by increasing the time for considering moves). At the tenth move White (the machine) makes a weak move which puts Black in a strong position; by the thirteenth move White's position is clearly hopeless, and 10 moves later, seeing the inevitability of a forced mate, the machine resigns.

Our contests with the machine show that anyone good enough to construct a three-move trap can beat it. Knowing how it selects its moves for consideration, you can often think of moves which you can be confident the machine will not consider. The machine will invariably accept a "sacrifice" (but then, so did the grand master José Capablanca). It will offer a sacrifice only to avoid being mated or if it can see an almost immediate mate of its opponent.

Yet notwithstanding its weaknesses, the IBM 704 plays a respectable and not-too-obvious game of chess – a game about which one can ask such questions as "Why did it make that move?" and "What does it have in mind?" We can even say frequently that "It made an excellent move at this point," or "At this stage it had a good position."

Undoubtedly our chess player is only a prototype for far more skillful players to be built in the future. Probably they will not go much farther in depth of planning: even with much faster computers than any now in existence it will be impracticable to consider more than about six half-moves ahead, investigating eight possible moves at each stage. A more promising line of attack is to program the computer to learn from experience. As things stand now, after losing a game the machine quite happily makes the same moves again and loses

again in exactly the same way. But there are some glimmerings of ideas about how to program a machine to avoid repeating its mistakes, and some day – not overnight – we may have machines which will improve their game as they gain experience in play against their human opponents.

Actual game between computer and human opponent is described in conventional chess terminology. The comments of the human opponent have been interpolated.

| | Bernstein Program (White) | Opponent (Black) |
|---|---|---|
| 1. | e2–e4 | e7–e5 |
| 2. | ♗f1–c4 | b7–b6 |
| 3. | d2–d3 | ♘g8–f6 |
| 4. | ♗c1–g5 | ♗c8–b7 |

Black is preparing for a direct attack on the centre, via d7–d5.

| | | |
|---|---|---|
| 5. | ♗g5×f6 | ♕d8×f6 |
| 6. | ♘g1–f3 | c7–c6 |
| 7. | 0-0 | d7–d5 |
| 8. | e4×d5 | c6×d5 |
| 9. | ♗c4–b5+ | ♘b8–c6 |
| 10. | c2–c4? | d5×c4 |

White 10. ♘f3×e5 is better because if Black replies ♕f6×e5 then ♖f1–e1. Since the pawn is defended by the queen ♘f3×e5 seemingly loses material, and the move is discarded.

| | | |
|---|---|---|
| 11. | ♗b5×c6+ | ♕f6×c6 |
| 12. | d3×c4? | e5–e4 |

White 12 is bad. ♖f1–e1 is better.

| | | |
|---|---|---|
| 13. | ♘f3–g5 | ♕c6–g6 |
| 14. | ♘g5–h3 | e4–e3 |
| 15. | f2–f3 | ♗f8–c5 |
| 16. | ♖f1–e1 | 0-0 |
| 17. | ♘b1–c3 | |

Fiddling while Rome burns.

| | | |
|---|---|---|
| 17. | .... | e3–e2+ |
| 18. | ♘h3–f2 | ♗b7×f3 |
| 19. | g2–g3 | e2×d1=♕ |
| 20. | ♘c3×d1 | ♕g6–c2 |
| 21. | b2–b3 | ♖f8–d8 |
| 22. | h2–h4 | ♖d8×d1 |
| 23. | Resigns | |

# 2.3 A Chess Playing Program for the IBM 7090 Computer

**Alan Kotok**

First published as a thesis submitted in partial fulfilment of the requirements for the degree of Bachelor of Science at the Massachusetts Institute of Technology, June 1962. Reprinted with permission of Massachusetts Institute of Technology, and of the author.

**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.

**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 fulfil 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 at 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 optimise 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.

## Organisation 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 therefore 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 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 heirarchy 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 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" is returned as the answer.

### Plausible move generation

REPLIES 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

* Appendix 1, the program listing, is not included here (DNLL).

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 $\frac{1}{60}$ 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 one point per pawn, to 3 or 4 points for other pieces. Development points are weighted $\frac{1}{15}$ of material points. This function is also eliminated as the game progresses.

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

| | |
|---|---|
| Open | file +8 |
| isolated | pawn −1 |
| backward | pawn −5 |
| doubled | pawn −3 |
| passed | pawn+10 |

These points are weighted $\frac{1}{20}$ 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 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 defender 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 a1. 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 column 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 lengthy fragments of games, and 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 output for a single move. The first page [Omitted here – Editor] is a printout of the chess board, and a list of the opponents legal replies, labelled 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. [This page has also been omitted. The principal variation was 1 d2-d4 ♘g8-f6. 1449 positions were examined, and the value of the principal variation was 27. – Editor]

Below is the actual move tree. The moves listed therein are moves which were

Figure 1 – Move Tree
[This tree is for the position after 1 d2-d4 – editor]

| Level | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | Value |
|---|---|---|---|---|---|---|---|---|---|
| | *d7-d5 | | | | | | | | |
| | | ♘b1-c3 | | | | | | | 27 |
| | | ♘g1-f3 | | | | | | | 27 |
| | | ♗c1-f4 | | | | | | | |
| | | | *♘b8-c6 | | | | | | |
| | | | | ♘b1-c3 | | | | | |
| | | | | | *♘g8-f6 | | | | |
| | | | | | | ♘g1-f3 | | | |
| | | | | | | | *♗c8-f5 | | |
| | | | | | | | | ♗f4-e5 | 21 |
| | | | | ♘g1-f3 | | | | | |
| | | | | | *♘g8-f6 | | | | |
| | | | | | | ♘b1-c3 | | | |
| | | | | | | | *♗c8-f5 | | |
| | | | | | | | | ♗f4-e5 | 21 |
| | | | *♘g8-f6 | | | | | | |
| | *♘b8-c6 | | | | | | | | |
| | | d4-d5 | | | | | | | |
| | | | *♘c6-b4 | | | | | | |
| | | | | e2-e4 | | | | | 58 |
| | | | | ♘b1-c3 | | | | | 54 |
| | | | *♘c6-a5 | | | | | | |
| | | | | e2-e4 | | | | | 66 |
| | | | | ♘b1-c3 | | | | | |
| | | e2-e4 | | | | | | | |
| | | ♘b1-c3 | | | | | | | |
| | *♘g8-f6 | | | | | | | | |
| | | ♘b1-c3 | | | | | | | 33 |
| | | ♘g1-f3 | | | | | | | |
| | | ♗c1-f4 | | | | | | | |
| | *d7-d6 | | | | | | | | |
| | | e2-e4 | | | | | | | 46 |
| | | ♘b1-c3 | | | | | | | |
| | | ♘g1-f3 | | | | | | | |

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 ±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 amateur 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 multipliers 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.

## Appendix 3

Record of game played 2/24/62

| | Machine - white | M. Garber - black | |
|---|---|---|---|
| move | White | Black | Time (mins) |
| 1. | e2-e4 | e7-e5 | 1.5 |
| 2. | ♘b1-c3 | ♘g8-f6 | 1.8 |
| 3. | ♘g1-f3 | ♘b8-c6 | 2.2 |
| 4. | d2-d4 | e5×d4 | 4.8 |
| 5. | ♘f3×d4 | ♗f8-c5 | 0.8 |
| 6. | ♘d4×c6 | b7×c6 | 3.3 |
| 7. | e4-e5 | ♕d8-e7 | 4.4 |
| 8. | ♗c1-f4 | d7-d6 | 2.2 |
| 9. | ♕d1-e2 | ♘f6-g4 | |
| 10. | e5×d6 | ♗c5×f2+ | 1.2 |
| 11. | ♔e1-d2 | ♕e7×e2+ | 0.9 |
| 12. | ♗f1×e2 | c7×d6 | 1.5 |
| 13. | ♗f4×d6 | ♗f2-e3+ | 2.4 |
| 14. | ♔d2-d3 | ♗c8-a6+ | 1.1 |
| 15. | c2-c4 (ILLEGAL) | 0-0-0 | 1.0 |
| 16. | ♗e2×g4+ | ♔c8-b7 | 0.4 |
| 17. | ♘c3-e4 | ♗e3-f4 | 0.4 |
| 18. | ♘e4-c5+ | ♔b7-b6 | 0.9 |
| 19. | ♘c5-d7+ | ♖d8×d7 | 0.9 |
| 20. | ♗g4×d7 | ♗f4×d6 | 0.3 |
| 21. | ♖a1-f1 | f7-f6 | 3.4 |
| 22. | ♖f1-f5 | ♖h8-d8 | 3.3 |
| 23. | ♗d7-e6 | g7-g6 | 0.6 |
| 24. | ♖f5-f1 | f6-f5 | 1.5 |
| 25. | ♔d3-d4 | ♗d6-c5+ | 2.2 |
| 26. | ♔d4-c3 | ♗c5-d4+ | 0.8 |
| 27. | ♔c3-b3 | ♔b6-c5 | 0.8 |

Average time = 1.8 min./move

---

[In the game against Fiorenza the reader can see the "principal variation" which was printed out by Kotok's program. This is the variation expected by the program in reply to the computer's move. Thus, when the computer played 1 e2-e4 it expected the reply 1 . . . e7-e5. When the computer played 3 ♘g1-f3 it expected the reply 3 . . . ♗f8-c5, when it intended to play 4 ♗f1-b5 (though when Fiorenza did reply 3 . . . ♗f8-c5 the program changed its mind).

Similarly, the next game (against Garber) also includes the moves of the principal variation, this time to a depth of up to 7-ply. For example when playing 3 . . . e7-e5, the program expected the continuation 4 d2-d3 ♘g8-f6 5 0-0 e5-e4 6 d3×e4 d5×e4 7 ♘f3-g5 — Editor.]

Record of game played 4/21/62. Machine — white. R. Fiorenza — black

| Move | White | Black | Principal Variation | | Time (mins) |
|---|---|---|---|---|---|
| 1. | e2-e4 | e7-e5 | 1. ... e7-e5 | | 1.7 |
| 2. | ♘b1-c3 | ♘b8-c6 | 2. ... ♘b8-c6 | | 2.8 |
| 3. | ♘g1-f3 | ♗f8-c5 | 3. ... ♗f8-c5 | 4. ♗f1-b5 | 2.4 |
| 4. | ♗f1-c4 | ♘g8-f6 | 4. ... d7-d6 | 5. d2-d3 | 5.4 |
| 5. | ♗c4-d5 | 0-0 | 5. ... d7-d6 | 6. 0-0 | 6.2 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 6. | 0-0 | d7-d6 | 6.... | d7-d6 | 7. ♗d5×c6 | 5.8 |
| 7. | ♗d5×c6 | b7×c6 | 7.... | b7×c6 | 8. d2-d3 | 3.2 |
| 8. | d2-d3 | ♘f6-g4 | 8.... | ♗c8-g4 | 9. ♗c1-g5 | 5.5 |
| 9. | ♗c1-g5 | f7-f6 | 9.... | f7-f6 | 10. ♗g5-h4 | 4.0 |
| 10. | ♗g5-h4 | d6-d5 | 10.... | ♗c8-a6 | 11. ♖f1-e1 | 3.8 |
| 11. | e4×d5 | c6×d5 | 11.... | c6×d5 | 12. ♖f1-e1 | 4.9 |
| 12. | d3-d4 | e5×d4 | 12.... | e5×d4 | 13. ♘f3×d4 | 5.1 |
| 13. | ♘f3×d4 | ♗c5-d6 | 13.... | ♗c5×d4 | 14. ♕d1×d4 | 1.5 |
| 14. | f2-f4 | ♗c8-f5 | 14.... | ♗d6-c5 | | 3.7 |
| 15. | ♘d4×f5 | g7-g6 | 15.... | ♘g4-h6 | 16. ♘f5×h6 | 1.9 |
| 16. | ♕d1×g4 | d5-d4 | 16.... | ♗d6-c5 | 17. ♔g1-h1 | 0.7 |
| 17. | ♘c3-e4 | ♔g8-h8 | 17.... | ♗d6-b4 | 18. ♖a1-d1 | 3.6 |
| 18. | ♘f5×d4 | ♕d8-e8 | 18.... | ♗d6-e7 | 19. ♕g4-d1 | 4.8 |
| 19. | ♕g4-f3 | ♖a8-b8 | 19.... | ♕e8-d8 | 20. ♘e4×d6 | 4.3 |
| 20. | ♗h4×f6+ | ♔h8-g8 | 20.... | ♔h8-g8 | 21. ♖a1-b1 | 2.4 |
| 21. | ♖a1-b1 | ♖b8-b4 | 21.... | ♖b8-b4 | 22. ♖f1-e1 | 5.7 |
| 22. | c2-c3 | ♖b4-c4 | 22.... | ♖b4-c4 | 23. ♖b1-d1 | 8.0 |
| 23. | ♖b1-e1 | ♕e8-d7 | 23.... | ♕e8-a8 | 24. ♕f3-d1 | 5.8 |
| 24. | ♕f3-d1 | ♖c4-c5 | 24.... | ♗d6-e7 | 25. ♗f6×e7 | 3.9 |
| 25. | ♘d4-f3 | ♖f8-d8 | 25.... | ♕d7-e6 | 26. ♗f6-e5 | 8.0 |
| 26. | ♘e4×c5 | ♗d6×c5 | 26.... | ♗d6×c5 | 27. ♔g1-h1 | 4.1. |
| 27. | ♘f3-d4 | ♗c5-e7 | 27.... | ♖d8-f8 | 28. ♗f6-e5 | 3.3 |
| 28. | ♖e1×e7 | ♕d7-d6 | 28.... | ♕d7-c8 | | 3.2 |
| 29. | ♗f6-g5 | ♖d8-f8 | 29.... | ♖d8-d7 | 30. ♖e7×d7 | 2.3 |
| 30. | c3-c4 | c7-c5 | 30.... | ♕d6-d8 | | 3.5 |
| 31. | ♘d4-e6 | ♕d6×d1 | 31.... | ♕d6-a6 | 32. ♘e6×f8 | 2.0 |
| 32. | ♖f1×d1 | ♖f8-c8 | 32.... | ♖f8-f5 | 33. ♖d1-d7 | 1.3 |
| 33. | ♖e7-g7+ | ♖g8-h8 | 33.... | ♔g8-h8 | 34. ♖d1-d8 | 1.0 |
| 34. | ♖g7-c7 | | 34.... | ♖c8-a8 | 35. ♖d1-d7 | 3.0 |

Level
(i.e. move
number)

| M. Garber Machine | Principal variation | | | | |
|---|---|---|---|---|---|
| 1. ♘g1-f3 d7-d5 | 2. ♘f3-e5 | | | | |
| 2. g2-g3 ♘b8-c6 | 3. d2-d3 | | | | |
| 3. ♗f1-g2 e7-e5 | 4. d2-d3 ♘g8-f6 | 5. 0-0 e5-e4 | 6. d3×e4 d5×e4 | 7. ♘f3-g5 | |
| 4. d2-d3 ♗c8-f5 | 5. 0-0 ♘g8-f6 | 6. ♘b1-c3 e5-e4 | 7. d3×e4 d5×e4 | 8. ♘f3-g5 | |
| 5. ♘b1-d2 ♘g8-f6 | 6. 0-0 ♗f8-d6 | 7. c2-c4 e5-e4 | 8. d3×e4 d5×e4 | 9. ♘f3-g5 | |
| 6. c2-c4 ♗f8-d6 | 7. ♘f3-h4 ♗f5-e6 | 8. ♗g2×d5 ♗e6×d5 | 9. c4×d5 ♘f6×d5 | 10. e2-e4 | |
| 7. 0-0 0-0 | 8. ♘f3-h4 ♗f5-e6 | 9. ♗g2×d5 ♗×d5 | 10. c4×d5 ♘f6×d5 | 11. e2-e4 | |
| 8. c4×d5 ♘c6-b4 | 9. e2-e4 ♗f5-g4 | 10. ♘d2-c4 ♗g4×f3 | 11. ♗g2×f3 | | |
| 9. ♘d2-c4 e5-e4 | 10. d3×e4 ♗f5×e4 | 11. ♗c1-f4 ♗d6×f4 | 12. g3×f4 ♘f6×d5 | 13. e2-e3 | |
| 10. ♘f3-h4 g7-g6 | 11. ♗g2×e4 ♗f5×e4 | 12. d3×e4 ♘f6×e4 | 13. e2-e3 | | |
| 11. ♗c1-g5 ♖f8-e8 | 12. ♘h4×f5 g6×f5 | 13. d3×e4 f5×e4 | 14. ♗g5×f6 ♕d8×f6 | 15. e3-e3 | |
| 12. ♘h4×f5 g6×f5 | 13. d3×e4 f5×e4 | 14. ♗g5×f6 ♕d8×f6 | 15. e2-e3 | | |
| 13. d3×e4 f5×e4 | 14. ♗g5×f6 ♕d8×f6 | 15. e2-e3 | | | |
| 14. ♘c4-d2 ♘b4×d5 | 15. ♗g5×f6 ♘d5×f6 | 16. ♘d2-c4 ♗d6-e5 | 17. e2-e3 | | |
| 15. ♘d2×e4 ♖e8×e4 | 16. ♗g2×e4 ♗d6-f4 | 17. ♗g5×f6 ♘d5×f6 | 18. ♕d1×d8+ ♖a8×d8 | 19. ♗e4×b7 | |
| 16. ♗g2×e4 ♗d6-e5 | 17. ♕d1-c2 ♖a8-c8 | 18. ♗e4×d5 ♕d8×d5 | 19. ♗g5×f6 ♗e5×f6 | 20. e2-e4 | |
| 17. ♔g1-h1 | [Author's typographical error requires omission of principal variation at move 17 - Editor] | | | | |

## Editorial Appendix

After graduating from MIT Kotok's interest in computer chess died but his program remained alive. When McCarthy left MIT to take charge of the Artificial Intelligence Laboratory at Stanford University he took Kotok's program with him and improved its tree searching mechanism.

At the end of 1966 a four game match began between the Kotok/McCarthy program, running on Stanford University's IBM 7090 computer, and a program developed at the Institute of Theoretical and Experimental Physics in Moscow which used a Soviet M-20 computer. The Soviet program was written by Arlazarov, Adelson-Velsky, Bitman (a Soviet Master), Uskov and Zhivtovsky.

In two of the games both programs used a basic three-ply search, in the other two they searched to a depth of five ply. The result of the match was an outstanding success for the Soviet program, even though it had been written along the lines of Shannon's primitive type-A strategy and run on a slower machine. The American program, as we have already shown, used Shannon's type-B strategy.

The Soviet program won two games and the other two were agreed drawn when one of them reached move 40. (This agreement, made before the start of the match, was prompted by the abysmal end- game play of both programs.) In the two unfinished games the Soviet program had advantages which would certainly have proved decisive in human master play.

There follows the game records of all four encounters.

1. Moscow (3 ply) v Stanford 1 e4 e5 2 ♘c3 ♘c6 3 ♘f3 ♗c5 4 ♗c4 ♘f6 5 0-0 0-0 6 d3 d6 7 ♗e3 ♗g4 8 h3 ♗h5 9 ♗d5 ♗d4 10 g4 ♗×c3 11 bc ♗g6 12 ♗g5 ♖e8 13 ♖b1 ♖b8 14 ♕e2 ♔h8 15 d4 ♔g8 16 ♕c4 ♘a5 17 ♗×f6 ♕×f6 18 ♕d3 c6 19 de de 20 ♗b3 ♖bd8 21 ♕e3 b6 22 ♖fd1 ♖d6 23 g5 ♕e7 24 ♖d3 ♖×d3 25 cd ♖d8 26 ♖a1 ♕d6 27 d4 ed 28 cd ♘×b3 29 ab a5 30 ♖a4 ♕e6 31 ♘e5 ♕e8 32 f4 ♖d6 33 f5 ♗h5 34 ♘c4 ♖d8 35 ♘×b6 ♖b8 36 ♘c4 ♗d1 37 ♖a3 ♗c2 ½-½

2. Stanford v Moscow (3 ply) 1 e4 ♘f6 2 e5 ♘d5 3 ♘f3 e6 4 ♗b5 a6 5 ♗a4 b5 6 ♗b3 ♗b4 7 ♘c3 ♘f4 8 0-0 ♗b7 9 d4 ♗×c3 10 bc ♘d5 11 ♗×d5 ♗×d5 12 ♗a3 d6 13 ed cd 14 ♖e1 ♘c6 15 ♖e3 0-0 16 ♕e2 ♗c4 17 ♕e1 ♕c7 18 ♗b4 a5 19 ♗a3 ♔h8 20 ♘g5 h6 21 ♘e4 ♖fd8 22 ♘×d6 ♖×d6 23 ♗×d6 ♕×d6 24 a3 ♘e7 25 ♖e5 ♘c6 26 ♖c5 e5 27 ♕e4 ♖a6 28 ♖d1 g6 29 ♖d2 g5 30 ♖d1 a4 31 ♖d2 f6 32 ♕e3 ed 33 cd ♘e7 34 ♕g3 ♕×g3 35 hg ♘d5 36 ♖c8+ ♔h7 37 ♖f8 b4 38 ab ♘×b4 39 c3 ♘d5 40 ♖c8 ½-½

3. Moscow (5 ply) v Stanford 1 e4 e5 2 ♘f3 ♘c6 3 ♘c3 ♗c5 4 ♘×e5 ♘×e5 5 d4 ♗d6 6 de ♗×e5 7 f4 ♗×c3+ 8 bc ♘f6 9 e5 ♘e4 10 ♕d3 ♘c5 11 ♕d5 ♘e6 12 f5 ♘g5 13 h4 f6 14 hg fg 15 ♖×h7 ♖f8 16 ♖×g7 c6 17 ♕d6 ♖×f5 18 ♖g8+ ♖f8 19 ♕×f8 mate

4. Stanford v Moscow (5 ply) 1 e4 ♘f6 2 e5 ♘d5 3 ♘f3 ♘b4 4 ♗b5 c6 5 ♗a4 d6 6 d4 ♕a5 7 c4 ♘c2+ 8 ♔f1 ♘×a1 9 ♘c3 ♕b4 10 ♕e2 de 11 de ♗e6 12 ♕d1 ♗×c4+ 13 ♘e2 b5 14 ♗c2 ♘×c2 15 ♕×c2 ♗×a2 16 ♘ed4 ♕c4+ 17 ♔g1 c5 18 ♕d2 cd 19 ♘×d4 e6 20 ♘f3 ♘c6 21 ♕g5 ♖d8 22 ♗d2 ♕c1+ 23 ♗e1 ♕×b2 24 ♕f4 ♗d5 25 ♕g3 ♕e2 26 ♗c3 b4 27 ♗e1 ♗×f3 28 gf ♕×e1+ 29 ♔g2 ♕×e5 30 ♕h4 a5 31 ♖c1 ♘d4 32 ♖f1 ♘×f3! 33 ♕h3 ♖d3 34 ♕g3 ♘e1+ 35 ♖×e1 ♖×g3+ 36 ♔f1 ♕b5+ 37 ♖e2 ♖a3 38 ♔e1 ♖a1+ 39 ♔d2 ♕d5+ 40 ♔e3 ♖a3+ 41 ♔f4 ♕f5 mate.

# 2.4 The Greenblatt Chess Program

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

First published in the proceedings of the Fall Joint Computer Conference, 1967, pp. 801-810. Reprinted by permission of American Federation of Information Processing Societies Inc.

### 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 (Shannon, 1950a). In his paper the concept of minimax tree search is used. In 1950, Turing described a hand simulation of a chess program (1953). 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 others known as the Los Alamos program and the Bernstein program (Bernstein et al., 1958b) are described in a paper by Newell, Shaw and Simon (1958b). Their paper describes a chess program which deviates from the analysis done in 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 (1966). 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 (1962). A variant of Kotok's program was used by John McCarthy in a

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, 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.

chess match with a Russian program (Adelson-Velskii, et al., 1966) 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 (Samson, 1965). 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., e2-e4)
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 b5)
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 described 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 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, double, tripled –

various pawn structure defects. (See section on static board evaluator for further discussion.)

Development value – refers to a 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 leads 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 the position is evaluated, the value is returned to the level above and it becomes the "best value so far" for that position. Each 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 synonymous 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 is 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 a 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 favourable 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 generator**

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 irrelevant 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 hueristic interaction are used in the chess program. One type of interaction involves enumeration of all combinations of facts. Such an enumeration 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 enough 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 makes 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 an 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 similarly 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 development 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 specialised 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 recieve 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.

$$R = \{N/(T-1)\} \times \tfrac{1}{8} \times M, \text{ where}$$

N is the ratio of white material to black material at the mode 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 sub-terms, 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 centre 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 widths 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. The 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 (1958b).

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 from 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 refuted if it is equally good as the best so far at the previous level.

Such a consideration leads to a tremendous speed-up of the search, especially if what turns out to be the best move at each position is considered first. One of three attributes of the plausible move generator is that it usually assigns the highest plausibility score to the best move, so almost maximal advantage 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 obtained 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 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 recognised 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 look-ahead. 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 the 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 greater 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 MIT 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 – MacHack VI |
|---|---|---|
| 1. | g2-g3 | e7-e5 |
| 2. | ♘g1-f3 | e5-e4 |
| 3. | ♘f3-d4 | ♗f8-c5 |
| 4. | ♘d4-b3 | ♗c5-b6 |
| 5. | ♗f1-g2 | ♘g8-f6 |
| 6. | c2-c4 | d7-d6 |
| 7. | ♘b1-c3 | ♗c8-e6 |
| 8. | d2-d3 | e4×d3 |
| 9. | ♗g2×b7 | ♘b8-d7 |
| 10. | e2×d3 | ♖a8-b8 |
| 11. | ♗b7-g2 | 0-0 |
| 12. | 0-0 | ♗e6-g4 |
| 13. | ♕d1-c2 | ♖f8-e8 |
| 14. | d2-d4 | c7-c5 |
| 15. | ♗c1-e3 | c5×d4 |
| 16. | ♘b3×d4 | ♘d7-e5 |
| 17. | h2-h3 | ♗g4-d7 |
| 18. | b2-b3 | ♗b6-c5 |
| 19. | ♖a1-d1 | ♕d8-c8 |
| 20. | ♔g1-h2 | ♘e5-g6 |
| 21. | ♗e3-g5 | ♖e8-e5 |
| 22. | ♗g5×f6 | g7×f6 |
| 23. | ♘c3-e4 | f6-f5 |
| 24. | ♘e4-f6+ | ♔g8-g7 |
| 25. | ♘f6×d7 | ♕c8×d7 |
| 26. | ♘d4-c6 | ♖b8-e8 |
| 27. | ♘c6×e5 | ♖e8×e5 |
| 28. | ♕c2-c3 | f7-f6 |
| 29. | ♖d1-d3 | ♖e5-e2 |
| 30. | ♖d3-d2 | ♖e2×d2 |
| 31. | ♕c2×d2 | ♘g6-e5 |
| 32. | ♖f1-d1 | ♕d7-c7 |
| 33. | ♗g2-d5 | ♔g7-g6 |
| 34. | b3-b4 | ♗c5-b6 |
| 35. | ♕d2-c2 | ♘e5-c6 |
| 36. | ♗d5-e6 | ♘c6-d4 |
| 37. | ♖d1×d4 | ♗b6×d4 |
| 38. | ♕c2×f5+ | ♔g6-g7 |
| 39. | ♕f5-g4+ | ♔g7-h6 |
| 40. | ♕g4×d4 | ♕c7-e7 |
| 41. | ♕d4-h4+ | ♔h6-g6 |
| 42. | ♗e6-f5+ | ♔g6-g7 |
| 43. | ♕h4×h7+ | ♔g7-f8 |
| 44. | ♕h7-h8+ | ♔f8-f7 |
| 45. | ♕h8-a8 | ♕e7-c7 |
| 46. | ♕a8-d5+ | ♔f7-g7 |
| 47. | ♔h2-g2 | ♕c7-e7 |
| 48. | h3-h4 | ♔g7-h6 |
| 49. | g3-g4 | ♔h6-g7 |
| 50. | h4-h5 | ♕e7-e2 |
| 51. | h5-h6+ | ♔g7-f8 |
| 52. | h6-h7 | ♕e2×f2+ |
| 53. | ♔g2×f2 | ♔f8-e7 |
| 54. | h7-h8=♕ | a7-a6 |
| 55. | ♕d5-e6 | mate |

First Non-Loss by Computer in Tournament Play.
Game 3 Tournament 1.

| | White – 1410 | Black – MacHack VI |
|---|---|---|
| 1. | e2-e4 | e7-e5 |
| 2. | ♘g1-f3 | ♘b8-c6 |
| 3. | ♗f1-c4 | ♘g8-f6 |
| 4. | ♘f3-g5 | d7-d5 |
| 5. | e4-d5 | ♘c6-a5 |
| 6. | ♗c4-b5+ | c7-c6 |
| 7. | d5×c6 | b7×c6 |
| 8. | ♕d1-f3 | ♕d8-d5 |

| | | |
|---|---|---|
| 9. | ♕f3×d5 | ♘f6×d5 |
| 10. | ♗b5-e2 | ♗c8-f5 |
| 11. | d2-d3 | ♗f8-b4+ |
| 12. | ♗c1-d2 | ♗b4×d2+ |
| 13. | ♘b1×d2 | 0-0 |
| 14. | a2-a3 | f7-f6 |
| 15. | ♘g5-f3 | ♖a8-b8 |
| 16. | b2-b4 | ♘a5-b7 |
| 17. | 0-0 | ♘d5-c3 |
| 18. | ♖f1-e1 | ♘c3×e2+ |
| 19. | ♖e1×e2 | ♘b7-d6 |
| 20. | ♘d2-e4 | ♘d6×e4 |
| 21. | d3×e4 | ♗f5-e6 |
| 22. | ♖a1-d1 | ♗e6-c4 |
| 23. | ♖e2-d2 | ♖b8-b7 |
| 24. | ♖d2-d8 | ♖f8×d8 |
| 25. | ♖d1×d8+ | ♔g8-f7 |
| 26. | ♘f3-h4 | g7-g5 |
| 27. | ♘h4-f5 | ♖b7-c7 |
| 28. | g2-g4 | ♔f7-g6 |
| 29. | ♖d8-d6 | ♗c4-e2 |
| 30. | ♖d6-d8 | ♗e2×g4 |
| 31. | ♖d8-g8+ | ♔g6-h5 |
| 32. | ♘f5-g7+ | ♔h5-h6 |
| 33. | ♘g7-f5+ | ♔h6-h5 |

etc. and drawn by repetition.

First game Won By Computer in Tournament Competition.
Game 3 Tournament 2, Massachusetts State Championship 1967.

| White – MacHack VI | | Black – 1510 |
|---|---|---|
| 1. | e2-e4 | c7-c5 |
| 2. | d2-d4 | c5×d4 |
| 3. | ♕d1×d4 | ♘b8-c6 |
| 4. | ♕d4-d3 | ♘g8-f6 |
| 5. | ♘b1-c3 | g7-g6 |
| 6. | ♘g1-f3 | d7-d6 |
| 7. | ♗c1-f4 | e7-e5 |
| 8. | ♗f4-g3 | a7-a6 |
| 9. | 0-0-0 | b7-b5 |
| 10. | a2-a4 | ♗f8-h6+ |
| 11. | ♔c1-b1 | b5-b4 |
| 12. | ♕d3×d6 | ♗c8-d7 |
| 13. | ♗g3-h4 | ♗h6-g7 |
| 14. | ♘c3-d5 | ♘f6×e4 |
| 15. | ♘d5-c7+ | ♕d8×c7 |
| 16. | ♕d6×c7 | ♘e4-c5 |
| 17. | ♕c7-d6 | ♗g7-f8 |
| 18. | ♕d6-d5 | ♖a8-c8 |
| 19. | ♘f3×e5 | ♗d7-e6 |
| 20. | ♕d5×c6+ | ♖c8×c6 |
| 21. | ♖d1-d8 mate | |

A More Recent Game with Times for Computer Moves. Game 2, Tournament 3, Massachusetts Spring Amateur.

| White – MacHack VI | | | Black – unrated |
|---|---|---|---|
| 1. | e2-e4 | Book | e7-e5 |
| 2. | ♘g1-f3 | Book | ♘b8-c6 |
| 3. | ♗f1-b5 | Book | a5-a6 |
| 4. | ♗b5×c6 | Book | d7×c6 |
| 5. | 0-0 | Book | ♗f8-d6 |
| 6. | d2-d4 | Book | ♗c8-g4 |
| 7. | d4×e5 | Book | ♗g4×f3 |
| 8. | ♕d1×f3 | Book | ♗d6×e5 |
| 9. | c2-c3 | Book | ♕d8-h4 |
| 10. | g2-g3 | 18.3 | ♕h4-e7 |
| 11. | ♖f1-e1 | 44.9 | h7-h5 |
| 12. | h2-h4 | 111.7 | 0-0-0 |
| 13. | ♗c1-g5 | 78.5 | f7-f6 |
| 14. | ♗g5-f4 | 74.5 | g7-g5 |
| 15. | ♗f4×e5 | 45.3 | ♕e7×e5 |
| 16. | h4×g5 | 41.5 | f6×g5 |
| 17. | ♕f3-f5+ | 60.5 | ♕e5×f5 |
| 18. | e4×f5 | 29.1 | ♘g8-f6 |
| 19. | c3-c4 | 33.8 | h5-h4 |
| 20. | ♘b1-c3 | 77.6 | ♖d8-d2 |
| 21. | b2-b3 | 88.0 | h4-h3 |
| 22. | ♘c3-e4 | 56.0 | ♘f6×e4 |
| 23. | ♖e1×e4 | 43.3 | ♔c8-d7 |
| 24. | f5-f6 | 19.0 | ♖d2-d6 |
| 25. | f6-f7 | 19.0 | ♖d6-f6 |
| 26. | ♖a1-d1+ | 25.8 | ♖f6-d6 |
| 27. | ♖d1×d6 | 22.8 | c7×d6 |
| 28. | ♔g1-h2 | 68.25 | ♖h8-f8 |
| 29. | ♔h2×h3 | 76.6 | ♖f8×f7 |
| 30. | ♖e4-e2 | 22.6 | b7-b5 |
| 31. | ♔h3-g4 | 30.5 | ♖f7-g7 |
| 32. | ♖e2-e4 | 28.3 | d6-d5 |
| 33. | c4×d5 | 19.4 | c6×d5 |
| 34. | ♖e4-e5 | 23.2 | ♔d7-d6 |
| 35. | ♖e5×g5 | 14.0 | ♖g7×g5+ |
| 36. | ♔g4×g5 | 4.5 | ♔d6-e5 |
| 37. | f2-f4+ | 6.0 | ♔e5-e4 |
| 38. | f4-f5 | 8.5 | d5-d4 |
| 39. | f5-f6 | 4.9 | d4-d3 |
| 40. | f6-f7 | 5.1 | d3-d2 |
| 41. | f7-f8=♕ | 12.1 | d2-d1=♕ |
| 42. | ♕f8-f5+ | 27.7 | ♔e4-e3 |
| 43. | ♕f5-e6+ | 29.2 | ♔e3-f2 |
| 44. | ♕e6×a6 | 42.8 | ♕d1-d5+ |
| 45. | ♔g5-f4 | 20.45 | ♕d5-d4+ |
| 46. | ♔f4-f5 | 14.9 | ♕d4-d5+ |
| 47. | ♔f5-g4 | 21.6 | ♕d5-f3+ |
| 48. | ♔g4-h4 | 16.3 | ♕f3×g3+ |
| 49. | ♔h4-h5 | 4.7 | ♕g3-e5+ |
| 50. | ♔h5-h6 | 20.8 | ♕e5-h8+ |

etc. finally drawn by repetition.

## ACKNOWLEDGEMENT

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

# 2.5 The Technology Chess Program

**James J. Gillogly**

First published in *Artificial Intelligence,* volume 3, 1972, pp. 145-163. Reprinted by permission of North-Holland Publishing Company, and of the author.

**Abstract**

*A chess program has been developed which plays good chess (for a program) using a very simple structure. It is based on a brute force search of the move tree with no forward pruning, using material as the only terminal evaluation function, and using a limited positional analysis at the top level for a tiebreak between moves which are materially equal. Because of the transparent structure, this program is proposed as a technological benchmark for chess programs which will continue to improve as computer technology increases.*

## 1. Chess and Technology

Until recently the main effort in chess programming has been to develop programs which selectively (and hopefully "intelligently") examine a small subset of the legal moves in any position. The surprising performance of the Varian minicomputer (programmed by K. King and C. Daly) in the First Annual Computer Chess Championship (New York 1970), although due primarily to good luck in the pairings, led to increased speculation about the possibility of playing respectable chess with an unselective "brute force" program.

We were led to reconsider again programs that would simply generate all legal moves to a fixed depth, then evaluate the final position only with respect to material. Such programs would be very small and would have a transparent control structure, so it would be easy to reprogram them for faster computers as they become available. This type of program would need about a factor of 7 in computing speed to go each additional ply deeper in the same time. Since this speed increase is not unreasonable for (say) a decade of computer technology, a "technology program" would provide an increasing baseline for chess programming against which more sophisticated programs could be compared; that is, in order to justify the effort of bringing up a complex program, that program must be able to beat a technology program.

Initial experiments indicated that this basic design does not produce a useful baseline, since the standard of play is low for any reasonable depth of search. In practice, the problem is that before anything tactical (i.e. discoverable by changes in material) happens, the program has a hopeless positional disadvantage in terms of future opportunities. This type of program also makes tactical blunders due to evaluating at non-quiescent positions, although this effect decreases with increasing depth. (An example of the play of this type of program is given in the appendix, game 4. White is played by a technology-type program which uses no positional analysis but which has a limited knowledge of quiescence, and Black is played by the full program described in this paper.)

Consequently a more useful definition of a Technology program was sought. To be a useful benchmark the program should take only a few man-months to implement; therefore it must have a simple structure. The search should depend primarily on brute force, rather than on chess-specific heuristics; hence it probably could not afford forward pruning, which requires substantial analysis at each position. A reasonable limitation might be 5% or less CPU time spent on chess-specific heuristics.

* This work was supported in part by a RAND Corporation Fellowship and in part by the Advanced Research Projects Agency of the Office of the Secretary of Defense (F44620-70-C0107) and is monitored by the Air Force Office of Scientific Research. This document has been approved for public release and sale; its distribution is unlimited.

A program, TECH, has been developed which conforms to this less restrictive definition of a technology program. TECH was conceived in October 1970 and played its first complete game in November 1970. In March 1971 TECH beat a 1968 version of the Greenblatt program* with White and drew with Black. TECH joined the U.S. Chess Federation in May 1971, and has since played 26 USCF rated games (see Table 1), in August 1971 TECH won the second place trophy in the ACM-sponsored Second Annual Computer Chess Championship, placing behind Chess 3.5 (the defending champion, written by Slate, Atkin and Gorlen) and ahead of six other programs.

TECH is written in BLISS (Wulf et. al., 1971), a system implementation language developed at Carnegie-Mellon University. BLISS was chosen because (1) the language was designed to yield efficient object code, and (2) it is a higher-level language, and thus more legible than assembly language. TECH currently runs on a PDP-10.

This paper is an attempt to describe a technology program (TECH) in sufficient detail for others to build a similar program and compare its performance with TECH's. Several sample games are appended to demonstrate the level of play that can be achieved.

## 2. Design

The move production mechanism consists of two main parts: positional and tactical analysis. The portions of the tree affected by these components are shown in Fig.1. The positional analysis routine pre-sorts the moves at the top level (ply = 1) so that the move which has the best superficial positional score is considered first. No tactical considerations are included in the positional analysis. The tactical analyzer is a brute force tree search which investigates all moves to a fixed depth, applies a simple quiescence scheme, then evaluates the final position using material as its only criterion. Alpha-beta will accept the first of a group of materially equal best alternatives; the positional pre-sort ensures that the first of these alternatives will also have the highest superficial positional value.

In addition to invoking the move production routine, the supervisor controls utilization of the time when the opponent is planning his move (TECH is the only program so far which does this). TECH uses a two ply search to guess its opponent's move, then begins calculating its move on the basis of that assumption. If the opponent makes the expected move, TECH will print its reply immediately if finished, or continue the computations. The supervisor also gathers statistics on timing and on the size and shape of the tree, and controls the interface between the program and the operator, e.g. by interpreting and typing standard English notation, considering draw offers, setting up problem positions, and saving the board, score, and tree shaping variables.

## 3. Tactical Analysis

The most important part of the Technology Program (in terms of playing ability) is the brute force search or tactical analysis component. All moves are searched to a fixed depth (usually five ply in the middlegame), after which quiescence is approximated by

* Several versions of the Greenblatt program are available through DEC and others. The version used is somewhat identified by the fact that teletype input is in line mode and the prompting character is an asterisk. The tournament setting was used: SET 15 15997.

*Table 1. TECH's USCF-rated events.*

| Date | Event | Rounds | Points |
|---|---|---|---|
| May 1971 | Golden Triangle Open | 5 | 1.5 |
| June 1971 | Fred Thompson Memorial | 5 | 2.0 |
| Fall 1971 | Walled Knights Open | 4 | 1.5 |
| Sept 1971 | Pittsburgh Industrial League | 2 | 1.0 |
| Oct 1971 | Gateway Open | 5 | 2.0 |
| Nov 1971 | Carnegie-Mellon Open | 5 | 2.0 |

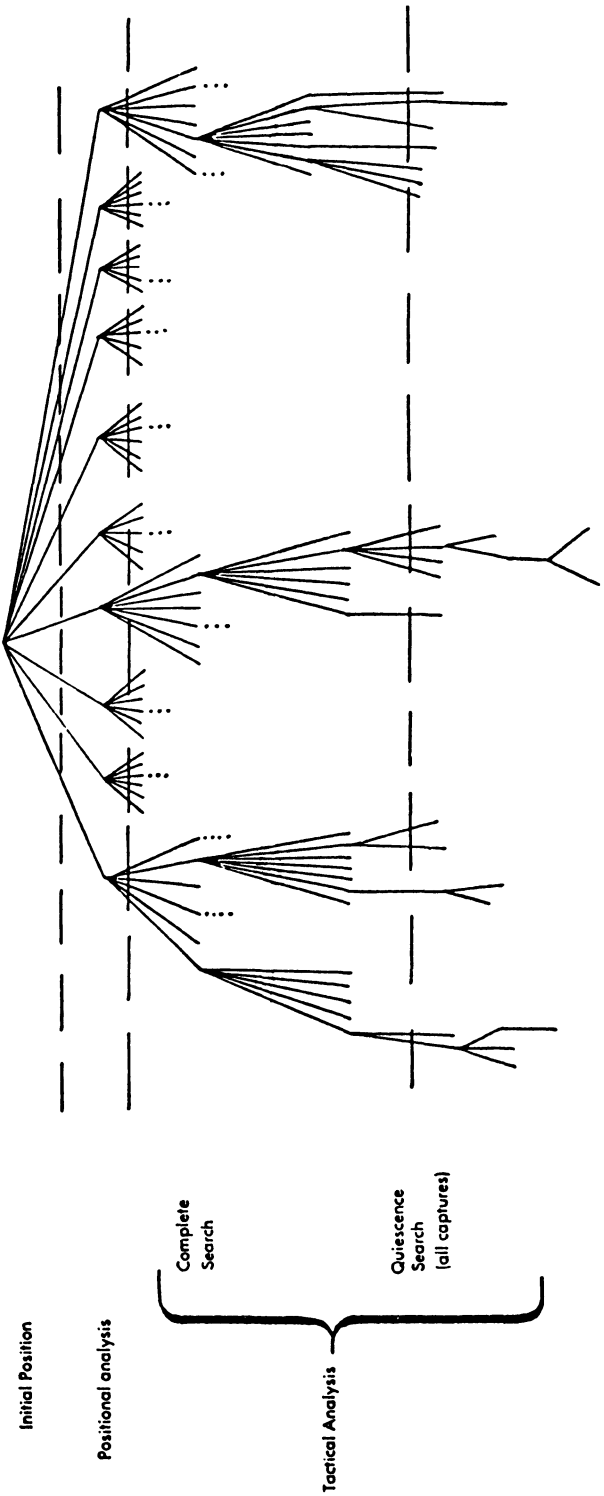



*Fig. 1. Move production for a 3 ply search with quiescence.*

investigating all sequences of captures.

Even using alpha-beta, this search strategy can result in as many as 500,000 bottom positions in a move analysis tree under tournament conditions, which means that TECH looks at many more positions than other programs in a given amount of time. By way of comparison, the Raymond/Ridley program which took third place at the Second computer Chess Championships looked at less than 100 bottom positions on each move.

This speed is achieved by using a simple terminal evaluation functions and efficient move generation. Most chess programs us a terminal evaluation function which includes material, king safety, pawn structure, development, and many other terms; TECH's terminal evaluation function is simply the value of TECH's material minus the opponent's material.

Potential legal moves are generated in the usual way, i.e. by adding offsets to a piece's location on the board, which is represented as a 120-word vector (see Fig.2). For example, the potentially legal moves for a knight on a3 are obtained by adding the offsets 8, 12, 19, 21, −8, −12, −19 and −21 to square 41 and testing the contents of those cells. If any of the resulting moves is illegal (moving into check, moving a pinned piece) they will be eliminated by the tree search when it notices that a king can be captured. Since this is a relatively rare occurrence, a considerable saving of time is achieved by not checking absolute legality of potentially legal moves.

The tree is represented as a stack of single words which contain the information necessary to travel up and down the tree

| | 110 | 111 | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 110 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 119 |
| 100 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 109 |
| 90 | 7 | -4 | -2 | -3 | -5 | -6 | -3 | -2 | -4 | 7 | 99 |
| 80 | 7 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | 7 | 89 |
| 70 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 79 |
| 60 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 69 |
| 50 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 59 |
| 40 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 49 |
| 30 | 7 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 7 | 39 |
| 20 | 7 | 4 | 2 | 3 | 5 | 6 | 3 | 2 | 4 | 7 | 29 |
| 10 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 19 |
| 0 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 9 |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | |

*Fig.2. Initial board representation.*

(see Fig.3). Two types of positions are represented by each word in the tree, distinguished by bit 35: those positions whose successors have been investigated, and those whose successors have not (see Fig.4). The stack is initialized by pushing the positionally sorted top-level moves onto the tree so that the best move will be popped first. This move is marked in bit 35, the move is executed, and its successors are generated and pushed onto the tree. When a bottom position is reached, it is evaluated and the value is backed up by minimaxing until the next unevaluated position is reached. The successors of that node are then evaluated.

For the sake of efficiency moves are retracted while traversing the tree, rather than recopying the previous position at each point. The value of material is incrementally updated only when a capture is encountered, so that the terminal evaluation function consists of the single BLISS assignment statement "AB[.PLY]←.MATERIAL;".

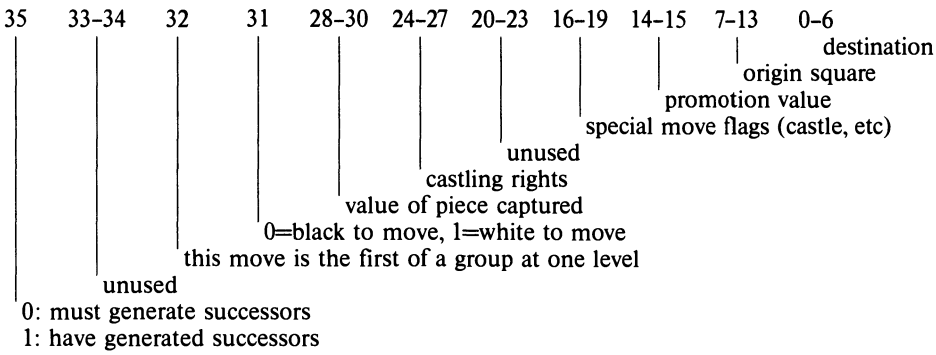



*Fig.3. Tree word bit allocations.*

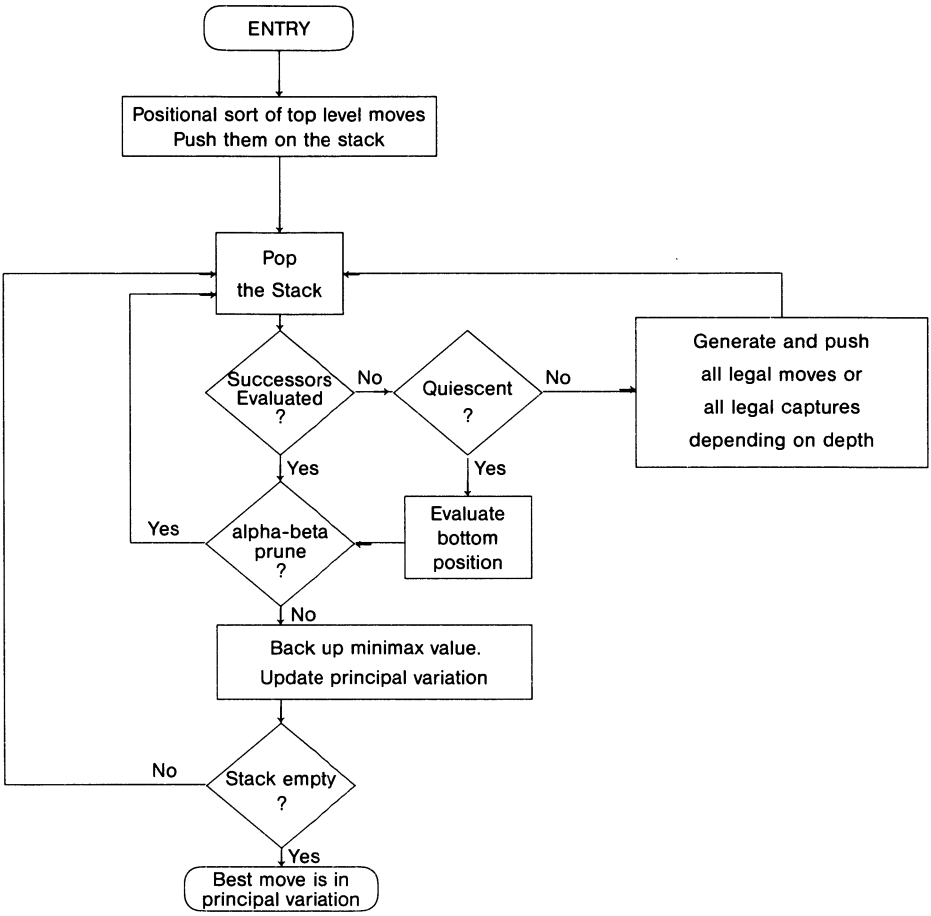



*Fig.4. Tree Discipline.*

(AB is a vector which contains the best value so far at each level. AB stands for Alpha-Beta.) The basic piece values are ♙ = 100, ♘ = ♗ = 330, ♖ = 500, ♕ = 900, and ♔ = 15,000. These values can be modified for one side by the positional analysis.

The efficiency of alpha-beta is greatly increased if the best moves are considered first (Slagle et. al., 1969). Since the refutation of a bad move is often a capture, all captures are considered first in the tree, starting with the highest valued piece captured. This is an inexpensive process, since captures can be recognized and sorted during move generation. The killer heuristic is also used; if a move is a refutation for one line, it may also refute another line, so it should be considered first if it appears in the list of legal moves.

## 4. Positional Analysis

The positional pre-sorting routine provides a tie-break for all moves with equal tactical values. It contains nearly all the chess-specific heuristics used by TECH. When used with the tactical search, it almost always can achieve a strong opening, even against players and programs which use a book. Although TECH does not use a book of openings, it follows standard opening play very closely (see Appendix, game C). TECH recognizes five phases: opening, middle game, endgame with pawns, general end-

game, and endgames where TECH must mate with pieces only. Remember that all the heuristics described in this section are applied only to the moves at the top level of the tree.

In each phase TECH will readjust the piece values given in the previous section if necessary. Specifically, if TECH is ahead by 200 points (2 pawns) or more, the new value of a piece (not a pawn) is computed by the formula new value ← old value * max (.6, opponent's material/TECH's material).

This encourages TECH to exchange pieces when ahead by making its pieces worth as little as 60% of its opponent's pieces.

After the opening phase TECH will also adjust its maximum depth for the next tactical analysis by comparing the average amount of time per move available before the next time control with the average amount of time used on the last five moves. If it has taken too much time, it decreases the depth. If it has taken enough time less than the allotted amount it increases the depth. It needs a factor of 3 to go up from an even depth to an odd one, and a factor of about 7 to go from an odd depth to an even one.

### *4.1. Opening*

The opening is defined to be the first eight moves. The most important heuristic in the opening evaluation is occupation of the center. Each square on the board is weighted with a desirability value ranging from 0 points for the corners to 8 points for the center (Fig.5). Each move represents a net gain or loss of centrality. For example ♘g1-f3 would yield a gain of 5 points in centrality.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 3 | 2 | 1 | 0 |
| 1 | 3 | 4 | 5 | 5 | 4 | 3 | 1 |
| 2 | 4 | 6 | 7 | 7 | 6 | 4 | 2 |
| 3 | 5 | 7 | 8 | 8 | 7 | 5 | 3 |
| 3 | 5 | 7 | 8 | 8 | 7 | 5 | 3 |
| 2 | 4 | 6 | 7 | 7 | 6 | 4 | 2 |
| 1 | 3 | 4 | 5 | 5 | 4 | 3 | 1 |
| 0 | 1 | 2 | 3 | 3 | 2 | 1 | 0 |

*Fig.5. Center control array.*

This is multiplied by the priority factor for the piece to move: ♙=1, ♘=4, ♗=3, ♖=2, ♕=1, and ♔= −1. Thus ♘g1-f3 would have a final score of 20 points for centrality. Notice that the king is encouraged to move away from the center in the opening, since its center-tropism factor is negative. This heuristic alone dictates a very reasonable opening with rapid development.

Each move is given a final positional score of the centrality term plus the value of each of the following heuristics which applies to it:

Pawn from e2 to e4 : 30 points
Pawn from e3 to e4 : 2 points
Pawn from d2 to d4 : 20
Pawn from d3 to d4 : 2
0-0 : 30
0-0-0 : 10
♘b1-a3,♘g1-h3 : −15
Piece to e3 or d3 blocking a pawn : −50
Piece moving from king side : 2
Playing Petroff defence : −50
Capture with pawn toward center : 5
Capture with pawn away from center : −5
Pawn capture leading to multipled isolated pawns : −10
Wing pawn advance : −10
Capture isolated center pawn : 50
Capture supported center pawn : −15

### *4.2. Middle Game*

The middle game begins with the ninth move and continues until one side has less than 1950 points worth of material, excluding the king (each side has 4420 in the initial position). The centre control heuristic is still used, but the priority factors are slightly altered: ♙ = 3, ♘ = 4, ♗ = 3, ♖ = 2, ♕ = 1, and ♔ = 1. Since most pieces have found their best squares by the middle game, this factor has less influence than in the opening. Each move is credited with a mobility term, which is the number of potentially legal moves available after the move is made. Movement of a piece into the opponent's king field (see Fig.6) is rewarded in the same way as the centre control heuristic, and the net gain is again multiplied by the priority for that piece. This heuristic occasionally results in a king-side attack.

| | | | | | | |
|---|---|---|---|---|---|---|
| 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2 | 8 | 8 | 8 | 8 | 8 | 2 |
| 2 | 8 | 10 | 10 | 10 | 8 | 2 |
| 2 | 8 | 10 | ♔ | 10 | 8 | 2 |
| 2 | 8 | 10 | 10 | 10 | 8 | 2 |
| 2 | 8 | 8 | 8 | 8 | 8 | 2 |
| 2 | 2 | 2 | 2 | 2 | 2 | 2 |

*Fig.6. Middle game king field.*

The pawn heuristics are the same as in the opening, except that advances of wing pawns get −5 instead of −10. Castling values are the same as in the opening. If TECH is ahead in material, piece captures get 10 points more. Moving a piece which blocks the KBP or QBP is rewarded with 5 points.

*4.3. Endgame with pawns*

The most important goals in pawn endgames are advancing and blocking passed pawns.

| | | | | |
|---|---|---|---|---|
| 8 | 12 | ♙ | 12 | 8 |
| 3 | 5 | 4/8 | 5 | 3 |
| 4 | 6 | 5/7 | 6 | 4 |
| 1 | 2 | 1/5 | 2 | 1 |
| | | 0/1 | | |

own pawn/opponent's pawn

*Fig.7. Pawn endgame passed pawn field.*

Each move is credited with the net gain in the passed pawn field shown in Fig.7. This allows TECH to escort the pawn (if its own) or block it (if the opponent's). The king field (Fig. 8) and center control arrays are used only for king moves.

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 3 | 2 | 1 | 1 |
| 1 | 3 | 4 | 5 | 4 | 3 | 1 |
| 2 | 4 | 6 | 6 | 6 | 4 | 2 |
| 3 | 5 | 6 | ♔ | 6 | 5 | 3 |
| 2 | 4 | 6 | 6 | 6 | 4 | 2 |
| 1 | 3 | 4 | 5 | 4 | 3 | 1 |
| 1 | 1 | 2 | 3 | 2 | 1 | 1 |

*Fig.8. Pawn endgame king field.*

Pawn moves are weighted by the rank of their destination and by whether they are opposed:

| Rank | Opposed | Unopposed |
|---|---|---|
| 3 | 2 | 3 |
| 4 | 1 | 5 |
| 5 | 3 | 10 |
| 6 | 4 | 13 |
| 7 | – | 23 |
| 8 | – | 80 |

If TECH has more than one pawn on a file, only the first is given this bonus; the other pawns lose 10 points.

*4.4. General Endgame*

As in the pawn endgame, TECH's main goal is to promote. The pawns are given the same weights for advancing as in the preceding section. The material value of a pawn is raised from 100 to 120; if TECH has 2 or less pawns, they are worth 190 each. A move which places a rook behind a passed pawn of either color is rewarded with 15 points. The center control term uses priorities of ♙ = 0, ♘ = 4, ♗ = 3, ♖ = 1, ♕ = 1, and ♔ = 4. This encourages the king to centralize. TECH also uses the king field mask (Fig.9) to minimize the distance between kings. As in the middle game, the mobility is added to the score for a move.

| | | | | | | |
|---|---|---|---|---|---|---|
| 4 | 4 | 5 | 6 | 5 | 4 | 4 |
| 4 | 8 | 10 | 10 | 10 | 8 | 4 |
| 5 | 10 | 10 | 10 | 10 | 10 | 5 |
| 6 | 10 | 10 | ♔ | 10 | 10 | 6 |
| 5 | 10 | 10 | 10 | 10 | 10 | 5 |
| 4 | 8 | 10 | 10 | 10 | 8 | 4 |
| 4 | 4 | 5 | 6 | 5 | 4 | 4 |

*Fig.9. General endgame king field.*

*4.5. Endgame with Pieces*

Unlike other forms of endgame, TECH's goal on the endgame with pieces is to drive its opponent's king to the edge in order to deliver mate. This is achieved by doing a small (2 ply) tree search and using as an evaluation function:

(1) −32*opponent's king location on the center control field (Fig.5)

(2) 2*opponent's king location in TECH's king field (Fig.10)

(3) TECH's king location on the center control field, and

(4) the sum of TECH's piece locations in TECH's king field divided by the number of TECH's pieces (to keep pieces near the king as a tiebreak).

| | | | | | | |
|---|---|---|---|---|---|---|
| 4 | 4 | 5 | 6 | 5 | 4 | 4 |
| 4 | 8 | 10 | 9 | 10 | 8 | 4 |
| 5 | 10 | 10 | 10 | 10 | 10 | 5 |
| 6 | 9 | 10 | ♔ | 10 | 9 | 6 |
| 5 | 10 | 10 | 10 | 10 | 10 | 5 |
| 4 | 8 | 10 | 9 | 10 | 8 | 4 |
| 4 | 4 | 5 | 6 | 5 | 4 | 4 |

*Fig.10. Piece endgame king field.*

This method of forcing the king to the side of the board is due in part to Slate, Atkin, and Gorlen (authors of CHESS 3.5).

## 5. Statistics

In order to assess present and future implementations of technology programs, it is

necessary to determine how much effort is required to search one ply deeper, and how much better an $n$+1-ply program is than an $n$-ply program. Statistics are presented in this section to shed light on the former question, but the latter question has not yet been settled. To get the statistics, every fifth move of each of TECH's tournament games was analyzed at depths of 2, 3, 4, and 5 ply.

One measure of search effort used frequently in the literature is the number of bottom positions (NBP) in the tree shown in Fig.11. As Slagle and Dixon's results indicate (1969), the factor required to go from even to odd ply (about 8 in this sample) is larger than that required to go from odd to even (about 4). However, NBP is not a good

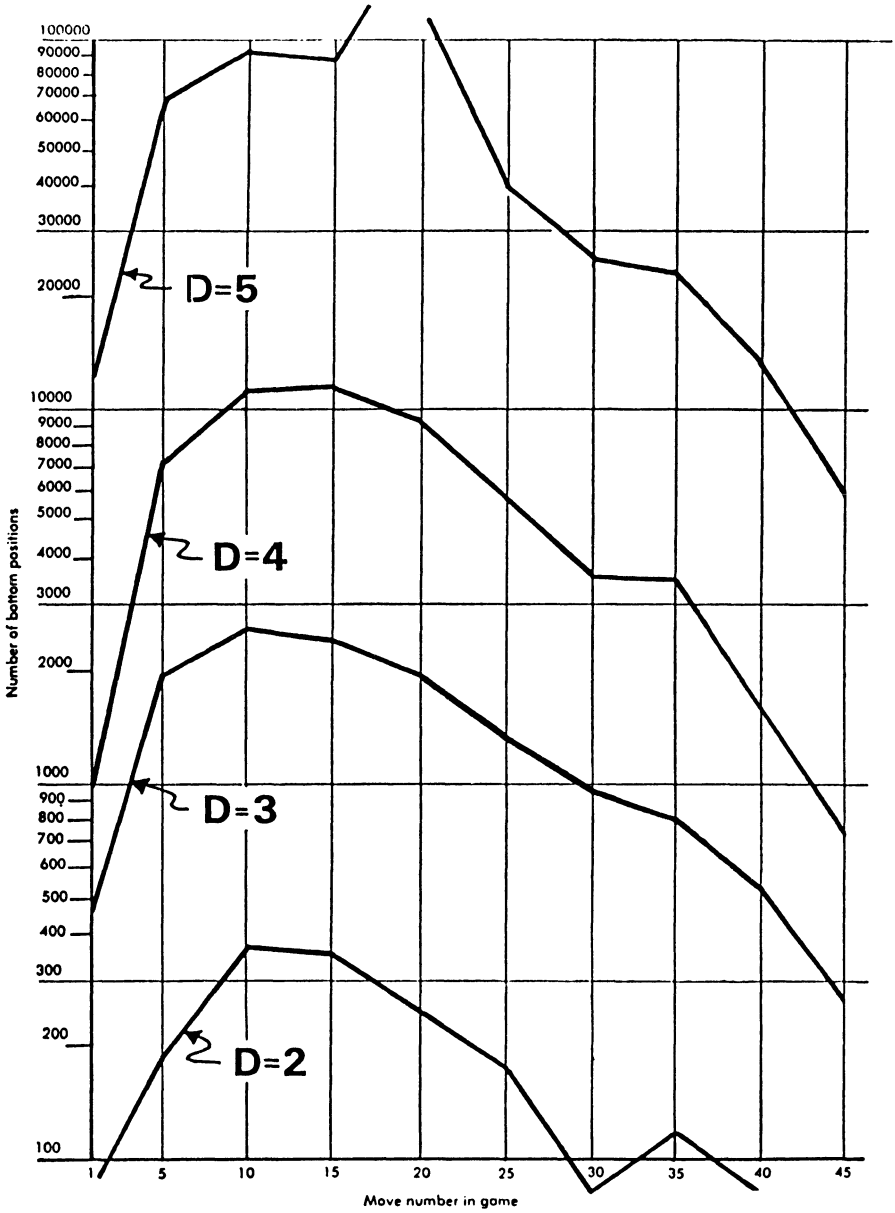



*Fig.11. Number of bottom positions vs. move number.*

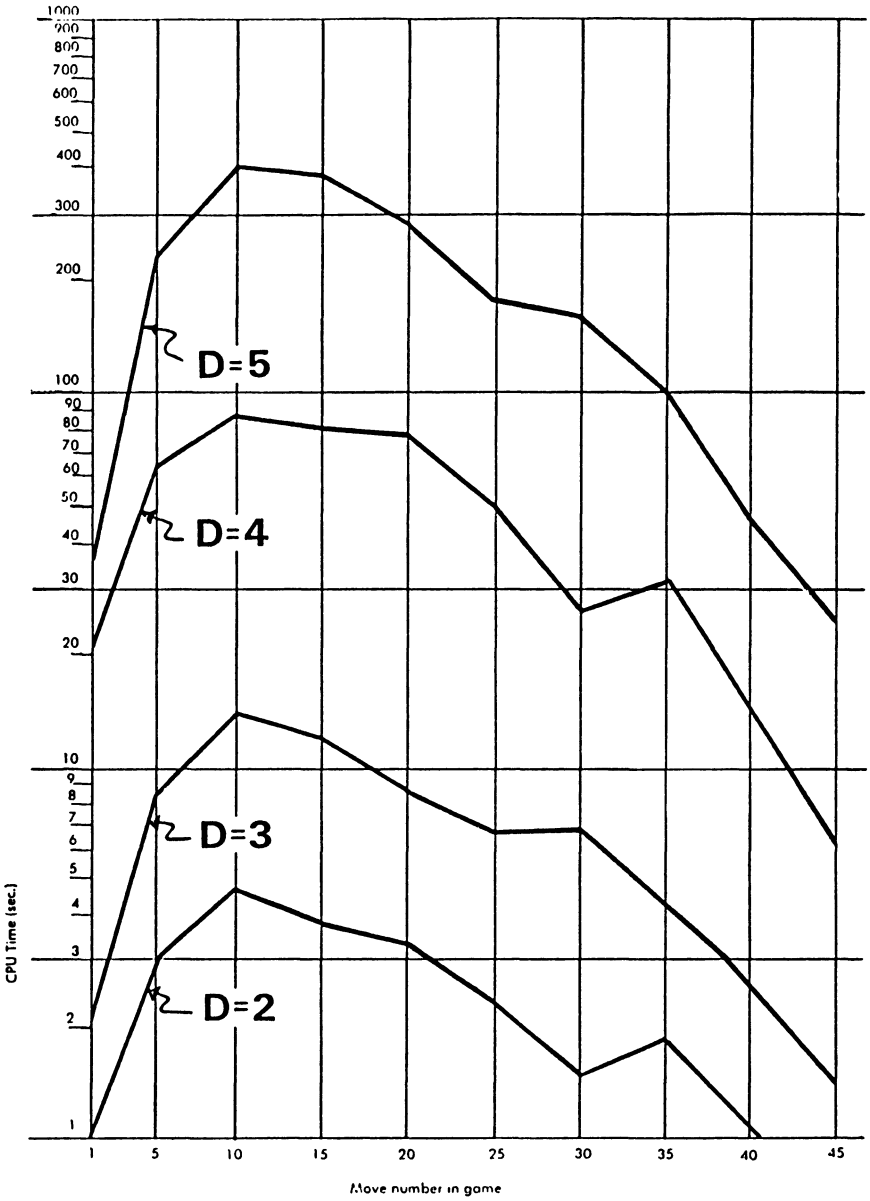



*Fig.12. CPU time vs. move number.*

measure of effort for TECH, since the amount of time spent processing bottom positions is negligible compared to the cost of move generation to arrive at those bottom positions. The CPU time (Fig.12) and the number of move generations (Fig.13) each show less effort to go from even to odd ply (about 3.5) than from odd to even (about 7). All the CPU times are inflated by about 10% due to the overhead of gathering statistics.

The following figures will be useful in comparing different technology programs:

PDP-10 cycle speed: 1 microsecond

TECH legal move generation (all potentially legal moves): 8,000 microseconds

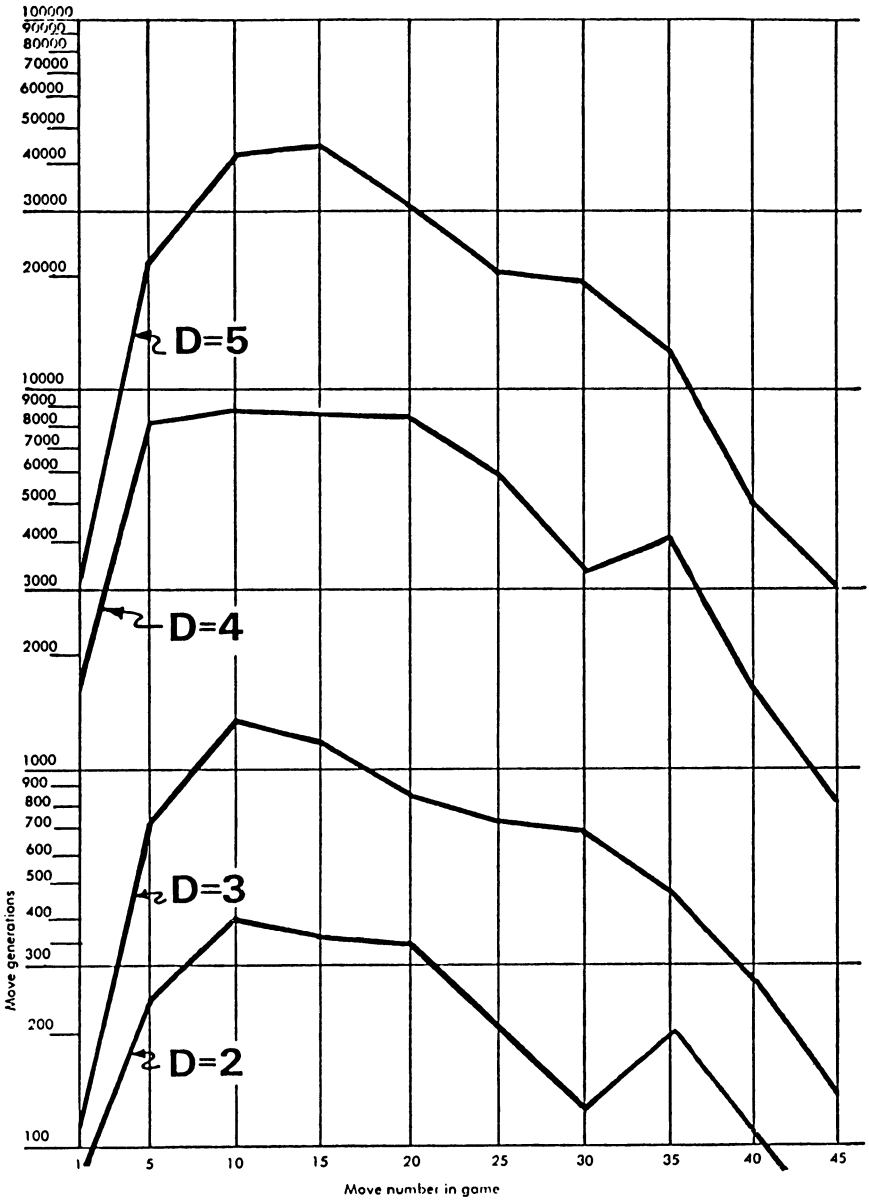



*Fig. 13. Number of move generations vs. move number*

Board updating (making or retracting moves): 1,300 microseconds

TECH spends its time as follows:

Legal move generation: 77%
Tree management: 7%
Board updating: 6%
Positional analysis: 1%
Everything else: 9%

## 6. The Future of Technology Programming

The experience gained from TECH suggests other types of programs within the "Technology" framework. For example, it would be possible to expand greatly the positional module, specifically including the basic endgames and allowing a deeper positional search. Some effects of the positional search

could also be included directly in the tactical search, so that without additional computation the program could sacrifice a small amount of material for significant positional advantages.

TECH is now close to its final form. Some of the planned minor modifications are implementation of Greenblatt's secondary search scheme (1967), variable individual piece values (to distinguish between "good bishops" and "bad bishops", for example), and a method of doing some incremental positional updating within the tree. A major goal is to get a firm USCF rating for the current configuration, so the technology baseline would be established. TECH's current rating in 1286 (Class D) based on about fifteen tournament games.

The future of chess programming lies with programs that (unlike TECH) have some understanding of chess, and which do a restricted, goal-oriented search. TECH's main limitations (no planning ability, no deep positional knowledge) seem to be inherent in the concept of a technology program. However, the goal of TECH is to be a useful benchmark for chess programming, rather than to become the world's best chess player. In this respect TECH seems to be quite successful.

**Appendix: Annotations by Hans Berliner**

Game 1: COKO III vs. TECH, Second Computer Chess Championship.

| | COKO III | TECH |
|---|---|---|
| 1. | e2-e4 | e7-e5 |
| 2. | ♘g1-f3 | ♘b8-c6 |
| 3. | ♗f1-c4 | ♘g8-f6 |
| 4. | d2-d3(A) | d7-d5 |
| 5. | ♗c4×d5?(B) | ♘f6×d5 |
| 6. | e4×d5 | ♕d8×d5 |
| 7. | ♘b1-c3 | ♗f8-b4 |
| 8. | 0-0 | ♗b4×c3 |
| 9. | b2×c3 | 0-0 |
| 10. | ♘f3-g5 | ♗c8-f5 |
| 11. | ♖a1-b1 | f7-f6 (C) |
| 12. | c2-c4 | ♕d5-c5 |
| 13. | ♘g5-h3 (D) | ♗f5×h3 (E) |
| 14. | ♗c1-e3 | ♘c6-d4! |
| 15. | g2×h3 | ♕c5-c6!! (F) |
| 16. | c2-c3? | ♘d4-f3+ |
| 17. | ♔g1-h1 | ♘f3-d2+! |
| 18. | f2-f3 | ♘d2×f1 |
| 19. | ♕d1×f1 | f6-f5 (G) |
| 20. | ♖b1-b5 | f5-f4 |
| 21. | ♖b5-c5 | ♕c6-e6 |
| 22. | ♗e3-c1 | c7-c6 |
| 23. | d3-d4 | ♖a8-e8 (H) |
| 24. | ♖c5×e5 | ♕e6-g6 |
| 25. | ♖e5×e8 | ♕g6×e8 |
| 26. | ♕f1-f2 | ♕e8-e6 |
| 27. | ♕f2-f1 | ♖f8-f5 |
| 28. | h3-h4 | c6-c5! (I) |
| 29. | d4-d5 | ♕e6-d6 |
| 30. | ♕f1-h3 | ♕d6-e5 |
| 31. | ♕h3-f1 | ♕e5×c3 |
| 32. | d5-d6 | ♕c3-d4! (J) |
| 33. | ♕f1-e2 | ♕d4×d6 |
| 34. | ♕e2-e8+ | ♖f5-f8 |
| 35. | ♕e8-a4 | ♖f8-f5 |
| 36. | ♕a4-e8+ | ♖f5-f8 |
| 37. | ♕e8-a4 | ♕d6-e6 (K) |
| 38. | ♕a4-b3 | ♕e6-e2!! (L) |
| 39. | h2-h3 | ♖f8-d8 (M) |
| 40. | ♗c1×f4 | ♖d8-d1+ |
| | Resigns | |

(A) A passive move which gives up any hope of advantage.

(B) e4×d5 was necessary. Now Black obtains a dominant position.

(C) Black has attained a fine position by making only simple classical developing moves.

(D) White should continue with ♗c1-c3 and then bring the knight back to f3. The text move results in a terrible weakening.

(E) A stroke of fortune. TECH does not recognise doubled pawns in its evaluating function, but selects this move because of the king proximity.

(F) Again serendipity. This is computed to be the best square for the queen (it is necessary to move it to avoid material loss due to c2-c3) and the threats on the diagonal are not part of the evaluation. Now ♗e3×d4 or f2-f3 had to be played.

(G) Correctly pursuing a policy of gaining space, but on the next move e5-e4 is better.

(H) This loss of a pawn was later found to be the result of a program bug, and makes things difficult. Simply e5×d4 wins.

(I) It is interesting that this important strategic break comes as a result of the positional heuristics, since there is no material gain involved.

(J) Now White is clearly lost. It is only a matter of time.

(K) The position repetition mechanism asserts itself in order to replace the favoured ♖f8-f5 by a previously untried move.

(L) A truly great move — instead of the "obvious" (to humans) ♕e6-e1+ which wins the bishop, TECH recognises the mate in one

possibility and prefers ♕e6-e2 to ♕e6-e1+, since it is more centralizing and the threat to win the bishop does not go away.

(M) Now TECH sees it can win the queen and does so, which leaves it in a situation that it can readily convert to mate. Actually, a mate was possible by 39...♕e2-f1+ 40 ♔h1-h2 ♖f8-e8 threatening both ...♖e8-e2 mate and ♕f1-f2+ followed by ♖e8-e1 mate, both of which cannot be answered; but this is out- side TECH's horizon.

Game 2: TECH (depth 4) vs David Levy, International Master, played 29 July 1971 at Carnegie-Mellon University.

| | TECH | Levy |
|---|---|---|
| 1. | e2-e4 | d7-d6 |
| 2. | d2-d4 | ♘g8-f6 |
| 3. | ♘b1-c3 | g7-g6 |
| 4. | ♘g1-f3 | ♗f8-g7 |
| 5. | ♗f1-d3 | 0-0 |
| 6. | 0-0 | c7-c6 |
| 7. | ♗c1-e3 | ♘b8-d7 |
| 8. | ♕d1-d2 | b7-b5 |
| 9. | ♗e3-h6 | ♘d7-b6 |
| 10. | ♗h6×g7 | ♔g8×g7 |
| 11. | ♘f3-g5 | ♘b6-c4? (A) |
| 12. | ♗d3×c4 | b5×c4 |
| 13. | f2-f4! | ♖a8-b8 |
| 14. | e4-e5! (B) | ♘f6-e8 |
| 15. | e5×d6? (C) | ♘e8×d6! |
| 16. | ♖a1-b1 | ♗c8-f5 |
| 17. | d4-d5 | c6-c5! |
| 18. | ♖f1-f3 | h7-h6 |
| 19. | ♘g5-h3 | ♕d8-a5 |
| 20. | ♘h3-f2 | ♘d6-b5 |
| 21. | ♖f3-e3 | ♘b5-d4 |
| 22. | ♘f2-e4 | ♖f8-d8 |
| 23. | ♔g1-h1 (D) | ♗f5×e4 (E) |
| 24. | ♖e3×e4 | ♖d8×d5 |
| 25. | ♖e4×e7 | ♖d5-d6 |
| 26. | ♘c3-e4 (F) | ♕a5×d2 |
| 27. | ♘e4×d2 | ♘d4×c2 |
| 28. | ♘d2×c4 | ♖d6-d4 (G) |
| 29. | ♘c4-e5 | ♖d4×f4 |
| 30. | ♖e7×a7 | ♖b8×b2 |
| 31. | ♖a7×f7+(H) | ♖f4×f7 |
| 32. | ♖b1-d1 | ♘c2-e3 |
| 33. | ♖d1-c1 | ♖f7-f2 |
| 34. | ♘e5-d3 | ♖b2-c2 |
| 35. | ♖c1-a1 | ♖f2×g2 |
| 36. | ♖a1-b1 | ♖g2×h2+ |
| 37. | ♔h1-g1 | ♖c2-g2 mate |

(A) Better is 11 ... h7-h6

(B) White has built up a formidable position by very simple means. It is significant that White has avoided playing e4-e5 prematurely when it would have resulted in a weak pawn, but has waited for the moment of maximum effect. Credit the heuristics which encourage full deployment of pieces before undertaking anything adventurous.

(C) A bad mistake due to a programming misunderstanding. The simple ♖a1-b1 leaves White in command. Now Black's game has been freed and White caves in to his opponent's superior ability.

(D) White sidesteps the dangerous ♘d4-f3+ in one variation, but his game is positionally hopeless.

(E) 23 ... ♖b8-b7 first is better and would win a pawn.

(F) ♕d2-e3 could have given rise to a dangerous counter-attack. But this involved more judgement than the program is capable of.

(G) Finally Black wins a pawn and for practical purposes ends the game.

(H) This error is due to the quiescence mechanism which only investigates captures but not checks. The program thought the main line was 31 ... ♖f4×f7 32 ♖b1×b2 and saw no danger in the move ♖f7-f1 (mate).

Game 3 GENIE vs TECH, Second Annual Computer Chess Championship.

| | GENIE | TECH |
|---|---|---|
| 1. | e2-e4 | e7-e5 |
| 2. | ♘g1-f3 | ♘b8-c6 |
| 3. | ♗f1-b5 | ♘g8-f6 |
| 4. | 0-0 | ♗f8-c5 |
| 5. | ♘b1-c3 | d7-d6 |
| 6. | d2-d4 | e5×d4 |
| 7. | ♘f3×d4 | ♗c8-d7 |
| 8. | ♘d4-f5 | 0-0 |
| 9. | ♗c1-g5 | ♘c6-e5 (A) |
| 10. | ♘c3-a4 | ♗d7×b5 |
| 11. | ♘a4×c5 | ♗b5×f1 |

and TECH eventually won.

(A) White has achieved a significantly superior position. However, this was accomplished by following a pre-stored "book" variation. TECH has stayed in the "book" by making moves selected by its opening heuristics package. It is significant to note that this heuristics package is good enough to generate a "standard" line of play, and that a program that relies on pre-stored variations of this type frequently (as is the case here) makes a mistake as soon as the game

departs from its pre-stored knowledge.

Game 4: TECH without positional analysis vs TECH (both with 4 ply search plus quiescence.)

| | WITHOUT | WITH |
|---|---|---|
| 1. | h2-h3 | e7-e5 |
| 2. | ♘b1-a3 | d7-d5 |
| 3. | ♘g1-f3 | ♘b8-c6 |
| 4. | e2-e3 | ♘g8-f6 |
| 5. | g2-g3 | ♗f8-c5 |
| 6. | ♘a3-b5 | 0-0 |
| 7. | a2-a4 | ♗c8-f5 |
| 8. | ♗f1-g2 | ♘f6-e4 |
| 9. | h3-h4 | ♖f8-e8 |
| 10. | ♖h1-h2 | ♖e8-e6 |
| 11. | ♖a1-b1 | ♘c6-b4 (A) |
| 12. | ♗g2-f1 | ♘e4×f2! |
| 13. | ♖h2×f2 | ♗f5×c2 |
| 14. | ♘f3-g5 | ♗c2×d1 |
| 15. | ♘g5×e6 | f7×e6 |
| 16. | ♔e1×d1 | ♕d8-e7 |
| 17. | ♗f1-e2 | ♖a8-f8 |
| 18. | ♖f2×f8+ | ♔g8×f8 |
| 19. | h4-h5 | e5-e4 |
| 20. | g3-g4 | ♕e7-h4 |
| 21. | d2-d3 | e4×d3! |
| 22. | ♗e2-f1 | ♕h4-f2 |
| 23. | ♗f1×d3 | ♘b4×d3 |
| 24. | ♗c1-d2 | c7-c6 |
| 25. | ♘b5-a3 | ♗c5×e3 |
| 26. | ♗d2×e3 | ♕f2×e3 |
| 27. | ♔d1-c2 | ♔f8-e7 |
| 28. | ♔c2-c3 | d5-d4+ |
| 29. | ♔c3-c4 | ♘d3×b2+! |
| 30. | ♔c4-c5 | ♕e3-c3+ |
| 31. | ♘a3-c4 | ♕c3×c4 mate (B) |

(A) Preparing a rather clever combination which could however have been repelled by 12 d2-d3.

(B) The advantage of the heuristics package which presorts the moves at the top of the tree can be very clearly seen from this example.

**Acknowledgements**

I am indebted to Hans Berliner, World Correspondence Chess Champion, who developed the elegant techniques used in the positional analysis, and whose patient discussion helped to clarify many of the conceptual problems. The basic idea of the Technology Program is due in large part to Allen Newell.

# 2.6 Chess 4.5: The Northwestern University Chess Program

**David Slate and Larry Atkin**

First published in *Chess Skill in Man and Machine* (P. W. Frey, Editor), Springer-Verlag, New York, 1977, pp. 82-118. Reprinted by Permission of Springer-Verlag and of the authors and editor.

CHESS 4.5 is the latest version of the Northwestern University chess program. CHESS 4.5 and its predecessors have won the U.S. Computer Chess Championships in 1970, 1971, 1972, 1973 and 1975, placing second in the 1974 U.S. Tourney and also in the first World tournament held the same year. This chapter will describe the structure of the program, focusing on the practical considerations that motivated the implementation of its various features. An understanding of not only what CHESS 4.5 is, but also why it turned out that way, is necessary if one is to appreciate its role in the present and future development of chess programming.

In the spring of 1968, engineering students Larry Atkin and Keith Gorlen launched Northwestern University's computer chess program in their spare time. Later in the year, then physics graduate student David Slate began his own effort. By mid 1969, the two groups had joined forces and produced their first successful program, CHESS 2.0. Between 1969 and 1972, CHESS 2.0 was gradually refined into CHESS 3.6. In that period, Atkin, Gorlen, and Slate all became professionals in the computing field. Gorlen left Northwestern in 1970 but continued for some time to contribute ideas to the program, whose development continued under Atkin and Slate. The story of that first era of chess programming at Northwestern is told in detail in Larry Atkin's masters thesis (1975). This chapter will concentrate on the "modern" era of our program, comprising CHESS 4.0 and the minor improvements that have led to CHESS 4.5.

## Background

In the spring of 1973 we (Larry Atkin and David Slate) faced the bleak prospect of yet another imminent computer chess tournament. By this time we had gotten over the initial thrill of winning the U.S. Championships. In fact, the annual ritual had turned into an annual chore – we could think of better things to do with our time than going through the motions of preparing for another tournament. In April we could no longer push the problem of ACM 1973 out of our minds. There seemed to be four alternatives:

1. Do not enter at all
2. Enter last year's program (CHESS 3.6)
3. Make modifications to CHESS 3.6
4. Write a whole new program

The possibility of not entering at all we finally discarded partly out of cowardice. We knew that our absence would be noticed, and we were afraid to answer the embarrassing questions that would inevitably greet our sudden and premature retirement from competition. Besides, we enjoyed getting together with other chess programmers, and perhaps we secretly enjoyed the competition at least as much as we disliked it.

We quickly ruled out entering CHESS 3.6. If there is anything more useless than yesterday's newspaper, it is last year's chess program. Our interest in the tournament lay in the chance to test something new and different, not to find out whether the other programs had improved enough to smash our old program. We knew, despite our unbeaten record and a well-developed myth about the "solidity" of our program, that our luck must soon give out. The bubble would burst, and the gross weaknesses of CHESS 3.6 would suddenly pour out in a series of ridiculous, humiliating blunders. For CHESS 3.6 was the latest in a series of evolutionary changes to our original chess program, written in 1968-1969, and it faithfully carried most of the

original design deficiences. CHESS 3.6 was, like the dinosaur, a species about to become extinct. Basically a Shannon type-B program, it had a depth-first, $\alpha$-$\beta$, more-or-less fixed depth tree search. A primitive position evaluation function scored the end-points and also doubled as a plausible move generator earlier in the tree by selecting the "best-*n*" moves for further exploration. Rudimentary as they were, CHESS 3.6's evaluation and tree search were just adequate to make "reasonable-looking" moves most of the time and not hang pieces to one- or two- move threats. Apparently this was enough to play low class C chess and, for a while, to beat other programs.

We could have tried to improve CHESS 3.6 by yet another notch. But that would have required expending much effort for rather little return. The design deficiences of CHESS 3.6 went much deeper than the mere fact that it was an old-fashioned program that depended on the searching of large trees filled with unlikely positions. It was poorly documented and not very modular. The instructions that formed the evaluation function were nearly unreadable. Figuring out how our old program worked was like deciphering hieroglyphics, and adding anything to it was like writing in ancient Sanskrit. One glance at the listing told us that the program had outlived its usefulness. It had taught us how not to write a chess program. It had sharpened our skills, brought us into contact with other chess-programmers, and broadened our perspective to the point where we could see CHESS 3.6 as the hopelessly archaic creature that it was. No, it had become too painful even to look at CHESS 3.6 anymore, let alone work on it.

Thus, by a process of elimination, we were forced to write a whole new program, despite the limited amount of time (four months) available for doing so. Fortunately, one's second chess program grows much more quickly and smoothly than the first.

## The development of CHESS 4.0

We had realized by April, 1973 that our view of a strong program – one that would play master-level chess – would be very different from CHESS 3.6. It would conduct very smart specialized, narrow, deep searches of the game tree. It would evaluate positions in a subtle, thorough fashion. The criteria for searching and evaluating would come from a large data base of chess knowledge to which chess players could contribute in an easy-to-use language that spoke of squares, pieces, and moves, rather than binary bits. In time, one would be able to instruct the program in methods of modifying its "knowledge pool" based on its own playing experience. Knowing all this, of course, didn't help much as far as the 1973 ACM tournament was concerned. We had to settle for something much less ambitious in the dwindling weeks that remained. So we chose instead to write a program which in terms of chess-playing methods was not very different from CHESS 3.6, but which in its architecture was more in the spirit of our vision of a strong program.

Some features of CHESS 4.0 were designed before we started writing code. Others developed as we went along, and the finishing touches went in just hours before the first game of the tournament. The criteria for adding something to the program were that it be simple to write, simple to understand, and easy to test. There was little time for experimentation or testing or the fine tuning of parameters. The description which follows details the essential features of this program and compares its design to that of its predecessor.

### *Program structure*

In terms of its organization, CHESS 3.6 was a mess. Not only was it difficult to modify the evaluation function – it was difficult even to find it in the listing of the program. So in 4.0 we decided to combine the tree searching and evaluation decisions and put them into one routine, called "EVALU8." The rest of the program performs utility functions in an efficient and straightforward manner. These include input-output of the moves and the processing of various commands that the user can type in to adjust parameters, debug the program, set up certain positions, etc. Other utility routines are concerned more directly with playing chess. They can generate moves, manage the tree search, and perform other tasks when called upon to do so by EVALU8. Much of the programming effort has gone into the design and construction of the

support routines rather than the chess-playing heuristics themselves. To illustrate the technical problems involved, some of these routines will be examined in detail later.

EVALU8 is responsible for a node in the tree. It returns a score and/or a best move for each node. It may call for tree searching to be done outward from that node to aid in its decision making, but control returns to it when each of these searches is complete. Since these searches invoke EVALU8 again at greater depths of the tree, EVALU8 is recursive and reentrant. When EVALU8 calls for such a search, all the data relevant to the current node are saved, and the board and other data are updated to reflect the move that is being searched. When searching of the move is complete, the data for the parent node is restored and control is returned to EVALU8 to continue processing that node. The purposes of such a structure is to eliminate, at least in principle, the artificial separation that exists in many programs (including CHESS 3.6) between node evaluation and tree searching.

CHESS 3.6 was a mixture of FORTRAN and assembly language (for CDC 6000/CYBER machines). Although we would have liked to use a high-level language, we felt that neither FORTRAN nor other languages available at the time offered the right combination of efficiency and power of expression. In writing CHESS 4.0, we used assembly language so we could have complete control over the instructions that were generated. For EVALU8, which contains all of the "chess decisions," we used high-level assembly language macros, which give the "illusion" of a higher level language.

## Data base

For each position in the look-ahead tree, CHESS 3.6 maintained certain data related to that position that were useful for evaluation and move generation. The largest part of this was a table specifying which pieces attacked which squares. This table suffered from two major problems: it had to be regenerated from scratch for each position in the tree, and it had no clean, simple format that might have more general utility. In 4.5, we chose a simple data format which could be used for many purposes, including the construction of attack tables which could be incrementally updated as the tree search stepped from node to node. Incremental updating means altering only those aspects of positional information that actually change due to a move on the board. The attack tables after (1) e2-e4, for example differ only slightly from those before, and much time is saved (although more code is required) if the changes are computed rather than the entire tables. The basic data element in CHESS 4.5 is the bit board.

The bit board is a string of 64 binary bits specifying some subset of the 64 squares on the chess board. Each bit that is set to one represents a square that is a member of the set.

Examples of such sets are "all the squares attacked by the White queen," "all the Black rooks that are on the seventh rank," etc. Some bit boards are permanent (incrementally updated from node to node), and some are temporary (generated and discarded as needed by EVALU8). Bit boards, which were described by Adelson-Velskii *et al.* (1966), are simple, yet can express a variety of notions concisely. Machine instructions exist on the CDC 6400 and most other computers to efficiently perform often desired operations on bit boards, like set functions "and," "or," etc., and such things as counting the elements of a set and finding the next member of a set. The incrementally updated attack tables, from which most move generation is done, are called ATKFR and ATKTO. ATKFR is a set of 64 bit boards which give, for each square, all the squares attacked by the piece, if any, that resides on the square. ATKTO is the transpose of ATKFR, giving, for each square, the locations of all pieces that attack that square.

Some data in forms other than bit boards are also maintained incrementally during the tree search. Assorted variables, like the current level in the tree, the current values of $\alpha$ and $\beta$, and the move being searched are needed for book-keeping purposes.

### *Hash tables*

In addition to the raw data kept in the data base, there are some functions that are computed for a large number of position in the tree. Some of these functions are time consuming, so we would like to avoid these calculations whenever possible. One scheme for saving time is to save the value of every function computed in a table and

look up its value (rather than recomputing it) whenever possible. In order to be advantageous, the table look-up scheme must be faster than the function it is replacing. CHESS 4.5 uses three hash tables to store previously computed results.

Material balance is the most important term in the evaluation of a position. It is a fairly complicated function of only the number of each type of piece on the board. The exact function is described in more detail later. For now, we will just describe how the values are kept in the hash table.

One word in the data base (the material balance signature word) contains the number of each type of piece for each side:

| w score | wq | wr | wb | wn | wp | b score | bq | br | bb | bn | bp |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 10 | 4 | 4 | 4 | 4 | 4 | 10 | 4 | 4 | 4 | 4 | 4 |

The left half of the word contains the counts of the White pieces, and the right half holds the Black pieces. The White score (w score) is the value of the material balance function for this count of White and Black pieces; the Black score (b score) is the negation of the White score. The numbers indicate the number of bits in each field. Each time that a move affecting the material balance (i.e., a capture or pawn promotion) occurs, the appropriate counts are updated. The program must now find the value of the material balance function. It first computes a very simple function of the material balance signature word and uses that value as an index into the material balance hash table. If that entry in the table matches (i.e., has the same number of each type of piece), the score is immediately available and is saved in the material balance signature word. If, however, the hash table entry is different, the value of the function must be computed, and the value and signature are placed into the table.

The hash function used to probe the table must have a number of characteristics. First, it must be fast. In particular, it must be faster than the function it is replacing. Second, it must distribute the positions that are likely to occur in a single search. Thus, the removal of a single piece from the material balance signature should always change the value of the hash function. Finally, if possible, the function should be symmetric with respect to Black and White. That is, a material balance signature should hash to the same location if the Black and White piece counts are reversed. For positions where both sides are about even, this doubles the utility of the table.

The hash function actually used is the sum of the White side of the signature (taken as a 20-bit number) and the Black side of the signature exclusive "or"ed with the absolute value of their difference, folded over (with an exclusive "or") to get a 7-bit number. The reader may verify that this function does indeed have the three desired characteristics.

The pawn structure hash table operates in a similar manner. The pawn structure on a file is dependent only on the locations of pawns on that file and the two adjacent files. The pawn structure signature has 18 bits (6 ranks by 3 files) for each side, indicating the presence or absence of a pawn on each of those squares.

The trans/ref table was added in the summer of 1975 and is the main difference between CHESS 4.5 and 4.2. It is a hash table of chess positions kept in auxiliary core memory (ECS) and is used for two purposes: to store previously computed evaluations and to store best moves. A similar hash table was mentioned by Greenblatt (1967). Our table has a user-adjustable capacity of from 256 to 65536 positions. In tournament play we have used from 8192 to 32768 entries depending on the availability of storage. The utility of the table is based on the graph of move sequences that is grown not being strictly a tree, because different branches may lead to the same position or node of the graph. Consider the position of Figure 1. The two sequences of moves: 1 ♘g1-f3 ♘b8-c6 2 ♗f1-c4 and 1 ♗f1-c4 ♘b8-c6 2 ♗g1-f3 both lead to the same position because White has transposed his first and second moves. If a score has been saved from the first occurence of that position, it can be used for the second, and we may avoid the whole subtree that would otherwise have been grown from that node. Another example is that of a piece that moves from one square to another in two steps, stopping at any one of several intermediate squares. For example, in Figure 1, the two sequences 1 ♘g1-f3 ♘b8-c6 2 ♘f3-g5 and 1 ♘g1-h3 ♘b8-c6 2 ♘h3-g5 are equivalent. There are situations in which the same position may recur several times. In Figure 1, the white bishop may stop at any of e2, d3, c4 or a6 on its way to b5,

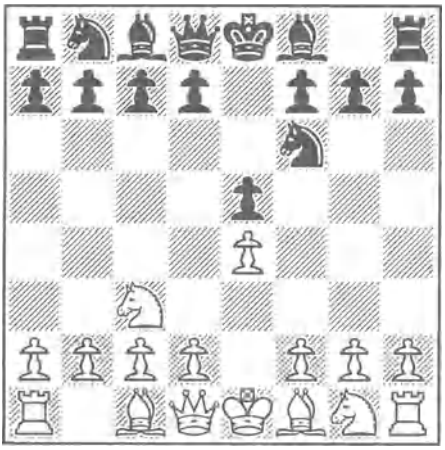


*Figure 1 A position in the opening with White to move.*

resulting in the same position after 3 ply providing Black's intervening move is the same each time. Of course it may not make sense for the bishop to take two moves to get to b5, but this makes little difference to a full-width search. Because of $\alpha$-$\beta$ cutoffs, not all such equivalent move sequences occur even in a full width search, but the killer heuristic, described later, helps maximize the occurence of equivalent branches. Although the above examples are all 3-ply long, there are many varieties of transpositions and multiple pathways creating equivalent branches of length greater than 3-ply.

Strictly speaking, positions reached via different branches are rarely truly identical, because the 50-move and three-time repetition draw rules make the identity of a position dependent on the history of moves leading to the position. This effect is small, and therefore we decided to ignore it. Differences in positions due to *en passant* pawns or castling status are correctly handled by the program.

The table contains the computed evaluation of the position as well as the predicted best move from the position. Since, in general, not all positions in the tree can fit in the table, priority scheme decides which are worthy. The main criterion is the ply depth at which the position occurred, with preference given to shallower positions.

## Move generation

Earlier versions of our chess program had a very simple mechanism for generating moves. The board was scanned, square by square, until a piece that belonged to the side on the move was encountered. The moves for this piece were then generated, and these new positions were scored.

With the introduction of the data base indicating which squares are attacked by each piece, move generation became more efficient. With the exception of pawns, a piece can move to any square that it attacks, unless that square is occupied by a friendly piece. Thus, to generate all of the moves of the knight from f4, all that the program must do is to intersect (i.e., logical "and") the bit board representing all attacks from f4 (i.e., ATKFR f4) with the complement (i.e., logical "not") of the bit board representing the location of all friendly pieces. The squares that are left represent the valid destination squares for the knight, and it is a simple matter to translate this bit board into a list of moves.

A similar procedure can be used to determine which pieces (if any) can move to a particular square. All moves that move a nonpawn piece to a specific square can be generated by examining the bit board representing all squares that contain pieces that attack that square, interesected with the set of squares occupied by the side to move.

Generating pawn moves is not so simple, but here, too, there is an advantage to using a bit board representation. It is possible to generate the moves of all the pawns simultaneously. The bit board representing the location of all pawns belonging to the side to move is shifted one rank, producing the set of all squares that are directly in front of a pawn. That, in turn, is intersected with all the empty squares on the board. The bits which remain represent the destination squares of all pawn moves where the pawn advances one square. This bit board can then be intersected with all squares on the third rank, shifted forward one more rank, and again intersected with the empty squares, producing all moves where a pawn moves forward two squares. Similarly, pawn captures are generated by shifting the pawns forward one rank, to the side one file, and intersecting with the location of all enemy pieces. Even *en passant* captures can be generated by including the "*en passant* square" (that square that a pawn passed over on the last move, if any) as the location of an enemy piece.

The move generator in CHESS 4.5 has two routines. One is GENTSL, which generates all moves from a set of squares. These routines have been constructed in a way which prevents the generation of the same move twice at a given node in the tree. This is accomplished at practically no expense by a very simple trick. Upon initial entry to a node, two bit boards are produced: the set of valid originating squares, GENFR, is initialized to be the set of all squares from which pieces can move (i.e., the location of all pieces belonging to the side on the move), and the set of valid destination squares, GENTO, is initialized as all other squares. Each time GENTSL is called, it intersects its parameter with GENTO (so that it does not generate any moves to squares that it has already generated moves to) and removes those squares from GENTO. Similarly, GENFSL intersects its parameter with GENFR and removes those squares from GENFR.

Now comes the real trick. Rather than using the "complement of the bit board representing the location of all friendly pieces" to determine if a square attacked by a piece is a valid move, GENFSL uses GENTO, thereby generating only those moves that GENTSL has not generated. Correspondingly, GENTSL uses GENFR instead of "the set of squares occupied by the side to move." In effect, neither routine will generate a move unless both:

1. The from-square was not a parameter to an earlier call to GENFSL,

   and

2. The to-square was not a parameter to an earlier call to GENTSL.

The move generator produces pseudolegal moves, moves that would be legal if leaving or moving a king into check were legal. A number of obvious schemes for generating only legal moves have been suggested, but they have weaknesses that require an excess of special-case code and are therefore difficult to implement and debug. Consider the problem of generating only strictly legal moves from the class of positions similar to Figure 2. The pseudolegal moves for White include five king moves and two pawn moves. Of these, the only illegal move is e5×d6 *en passant*. It is illegal because it removes the *two* pieces protecting the White king from being in check by the Black rook. In fact, most of the moves generated by the program are legal. Rather than spending time trying to determine if a generated move (which may never be searched anyway) is legal CHESS 4.5 defers this test until a more convenient time. If the move is actually searched, the data base will be updated, and it is then simple to determine that the king is sitting on a square that is attacked by an enemy piece.

*Figure 2 Position after Black plays ... d7-d5*

**Tree-searching strategy**

As stated earlier, CHESS 3.6 had a plausible move generator based on its evaluation function. Moves were ordered on the basis of preliminary scores produced by that function, and best *n*, plus some captures, checks, and other "threatening" moves, were searched from each position in the tree to a fixed depth limit. Other moves, except for certain special cases, were discarded as being unworthy of search. At first, we were going to implement a similar scheme in CHESS 4.0. However, with only a month or two remaining before the tournament, we changed our minds. Although our plausible-move generator sounded plausible enough, and differed not very much from methods employed in several other chess programs, we had built up profound dissatisfactions with it over the years. A suggestion by Peter Frey triggered some thoughts on the matter, and as a result we dumped selective searching in favor of full-width searching, ostensibly a more primitive algorithm. In CHESS 4.5 all legal moves are searched to the same depth. Beyond

that depth, only a limited "quiescence" search of captures and some checks is conducted. One may think of the quiescence search as a tool employed by EVALU8 to refine the tactical accuracy of the scores it produces for the positions at the horizon of the full-width search. At each node in the capture search, the side to move is given the choice of accepting the evaluation score or of trying to better it by trying capture moves.

The principal motivation for switching to full-width searching was a desire for simplicity. Simplicity was important to ease the testing and debugging that had to be crammed into a short period of time. The easiest way to avoid all the complexities of generating plausible moves is to do away with the plausible-move generator. The trouble with plausible-move generators is that they have to be very clever to avoid discarding, at ply 1, good moves whose merit even a meager 5-ply search would discover. This is true for both tactical and strategic moves. Thus a move that appears "quiet" (to a naive plausible-move generator) at ply 1 may pose a threat at ply 3 and win outright at ply 5. With CHESS 3.6 and other Shannon type-B programs, whether the right move is played often depends on whether, by sheer accident, that move is inadvertently included in the "best *n*," *n* being about 8 or so moves, at the base of the tree. This problem is discussed in some detail in Chapter 3 (Frey, 1977).

Some programs try to alleviate this problem by setting the "width" (number of moves looked at) higher near the base of the tree than at deeper ply levels. This works up to a point, but there are still too many moves whose consequences look good at 5-ply but are ranked no better than mediocre (20th or 30th or even worse) at the first ply where they are capriciously discarded by a well meaning, but stupid, plausible-move generator. The main cause of this phenomenon is that most plausible-move generators and evaluation functions are so near-sighted that they are apt to become suddenly aware of a feature of the position at, say, 5-ply that totally eluded them at 4-ply. This characteristic, which will become clear in the description of our evaluation function, gives rise to the "horizon effect" described in detail by Berliner (1974).

The proper way to conduct a selective search lies in such techniques as long-range planning, piece pathway analysis of causality of events in the tree search (see Berliner), and other devices that are not perfected in a period of a few months. So, instead, we took the coward's way out and made all of the moves "plausible." We made the pleasant discovery that the elimination of the move-ranking process, which required applying the evaluation function to each move, saved a great deal of time. It saved so much, in fact, that together with the 300 percent speedup of CHESS 4.5 over CHESS 3.6 due to greater efficiency, CHESS 4.5 is able to look at all the moves in the same time that CHESS 3.6 took to search a fraction of them. The lesson learned was not that selective searching doesn't work, but that unless it is done cleverly it is no better and may be worse than full-width brute-force searching.

The implementation of full-width searching had immediate beneficial results. At last we had a program whose behaviour we could explain simply. When it searched to 5 ply it found everything within that range, including both tactical combinations and positional maneuvers, some of which were obscure and ingenious. Although the endpoints of the tree were still subject to the gross defects of the primitive heuristics of the evaluation function, we had freed the search leading up to them from those defects, and thus unencumbered, the search was able to make up for some of them.

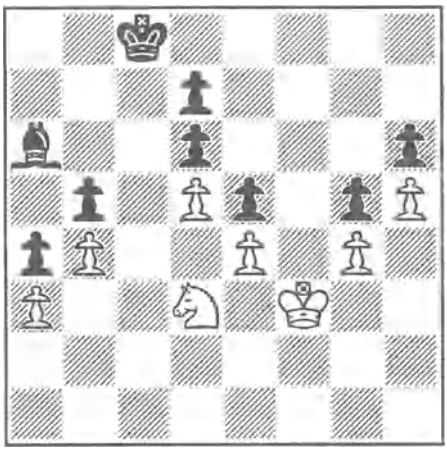


*Figure 3. An endgame position with White to move.*

In Figure 3 White may win by maneuvering the knight to f5. A smart plausible-move

generator or evaluation function would indentify the Black pawns at d6 and h6 as targets and construct the pathway that the knight should take to attack them. But, in this position, the evaluator of CHESS 4.5 is just lucid enough to know that eating the Black pawns is good for White. It also has a vague desire to make passed pawns and to centralize pieces like Knights. None of these factors justify evaluating the move ♘d3-f2 as good for White. But an 8-ply (plus capture sequences) search of all the moves makes the surprising (to the program) discovery that a Black pawn is won by force in 5 moves: ♘d3-f2, ♘f2-d1, ♘d1-e3, ♘e3-f5, ♘f5 × h6.

Besides simplicity, a full-width search rewards its creators with "peace of mind." The following daydream (or nightmare) illustrates the psychological hazards associated with the standard "best *n*" tree-search approach.

Imagine yourself at a computer chess tournament. In a complicated position your program has the opportunity to shine by finding the right continuation or to embarass you by making a blunder. You are reduced to a mere agent of the machine – communicating moves between it and its opponent and reporting the time on request. While anxiously waiting for the machine's decision, you speculate about what move it will make. It is difficult to infer the program's thinking processes. By combining an estimate of the machine's ability with an analysis of the structural features of the position, you decide that:

1. The program will very likely make the right move.
2. The program will very likely make the wrong move.
3. The machine's move will depend on seemingly irrelevant factors that are difficult to estimate.

In Cases 1 and 2 the suspense is relieved – one has peace of mind. Case 3, however, is hard on the nerves. Often one likes to be surprised by one's program, but not in positions where there is something straightforward to be done. A full-width search sharply reduces the number of incidents of Case 3, by eliminating the "seemingly irrelevant factor" of whether a tactically crucial move at a low ply level happens to lie just with the best-*n* group or just outside of it.

By now the reader is surely wondering what is the nature of an evaluation function which requires deep tree searching to enable it to recognize phenonema that appear obvious to most human players. So we will now describe the evaluator and return later to the tree search to actually show how we have made some small headway in overcoming the enormous branching factor that hangs like an albatross about the neck of brute-force searching.

## The evaluation function

The CHESS 4.5 evaluation function was derived from that of CHESS 3.6, with the addition of perhaps 30 percent more chess knowledge, but is similar in spirit. The code was completely rewritten in assembly language macros that combined greater clarity with greater efficiency. The whole function was produced in a few weeks shortly before ACM 73 and has remained almost unchanged in the two and a half years since.

Basically, the function adds several easily computable factors together. The dominating term is material. The remaining terms are rules of thumb designed to give the program a vague positional sense. Rather than expressing some theoretically rigorous chess principles, these factors merely improve somewhat over "aimless" moves. The design motive for the evaluator appears to be a self-fulfilling prophecy: we knew that it was going to be primitive and would thus depend heavily on tree-searching. To do such deep tree searching would require that the evaluator do it work quickly, and so it wouldn't have the time to do anything clever, and thus would have to be primitive.

### *Material balance*

Material constitutes by far the biggest term in the function. The total of the non-material terms does not usually exceed the value of about a pawn and a half. The theory of a materially dominated evaluation function is that whether and how quickly one is winning correlates well with having a favorable material balance, where material balance is dependent only on what pieces are on the board and not where those pieces are. CHESS 4.5 gives the pieces

"standard" values: pawn = 100, knight = 325, bishop = 350, rook = 500, queen = 900. It modifies this sum by a "trade-down" bonus that encourages the winning side to trade pieces but not pawns. The total function, expressed from the winning side's point of view is: Total material advantage = minimum (3100, MS), where: MS = minimum (2400, MD) + MD*PA* (8000– MT)/ [6400*(PA+1)]. MD is the material difference computed by adding the standard values of each piece and subtracting the total of the side with the least material from the side with the most (the winning side). PA is the number of pawns possessed by the winning side. MT is the total material on the board, i.e. the sum of the material of both sides.

As an example of how the function works, consider a material configuration in which each side has all its original pieces, except that Black has lost a bishop. Then MD = 350 and MT = 2*4050–350 = 7750. Also PA = 8 since White has all his pawns. Then compute: minimum (3100, MS), where MS = minimum (2400, 350) + 350*8 (8000–7750)/ [6400*(8+1)] = 362. Minimum (3100, 362) = 362, which is, therefore, the total advantage. Suppose a pair of knights are now traded off. The MT changes to 8100–350–2*325 = 7100, but MD and PA remain the same, and the advantage comes out to 394. Therefore, the trade-down bonus for exchanging knights is 394–362 = 32 or about one-third of a pawn. Actually, the material advantage is truncated to the next lower multiple of a quarter of a pawn by the program, so this difference would be less accurately represented. This function is, of course, so simplistic that it fails badly in many situations. In particular, it gives the wrong answers for such trivially drawn endgames as king and bishop against king and king and knight against king. To take care of a few of these situations, a table of exceptions is maintained which is searched before the usual computation is performed. Also, the trade-down bonus has the probably unfortunate property of decreasing the importance of positional terms relative to material terms as the material advantage, and hence the bonus, increases.

*Hung pieces*

It is important not to overlook pieces that are subject to immediate capture by the side to move. In CHESS 4.5, these pieces are not considered by the evaluation function but are instead relegated to the quiescence/ capture search, described in detail later. Pieces that are subject to capture by the side not on move are presumed likely to escape. Although the responsibility for an accurate prognosis of the future of these pieces is shifted to the capture search, that search, as we shall see later, fails to accept this responsibility. A term in the positional score helps to correct this oversight.

*Positional evaluation*

The nonmaterial terms depend on which of two phases the game is in: one set is used for most positions, while a simpler set serves in "mop-up" situations – endgames in which there are few pieces and one side has an overwhelming material advantage. The regular positional evaluation is the sum of the attacked piece term, which depends on which side is on move, and several piece position terms, which do not. These piece position terms are computed for both sides. Terms for White pieces are added to the total score, and those for Black are substracted from it.

*Attacked-piece term*

A piece subject to profitable capture by the side not on the move is penalized a small amount, not because its capture is considered likely, but because the tempo presumably used up moving to safety detracts from the quality of the position. A threatened piece is defined here simply as a nonpawn piece attacked by a smaller piece, or one attacked by an equal or larger piece and not defended. The definition of attack and defense here does not distinguish situations in which the attacker or defender is pinned or otherwise overloaded. If there is only one threatened piece, the penalty is 16. If there are more than one, the situation is more complicated. Prior to 1975 and CHESS 4.5, an extra penalty of 22 was assessed in addition to the 16. We had noticed, however, that CHESS 4.0 and 4.2 would sometimes judge favorably positions, reached in the tree, in which the computer was on move but suffered from multiple hung pieces. Of course the question of whether a piece must really be lost should be decided either by a very clever static analysis or by the quiescence search, which, unfortunately,

was at ease only in searching captures by the side on move, and not defenses or retreats. Often, the best defense to such a double threat, especially in the strange positions reached in the tree, is itself a capture which will be found by the capture search. So one could "play it safe" by penalizing a much larger amount than just 38. However, that would play havoc with the program's dependence on the fact that nonmaterial evaluation terms are always small, a fact that the program makes considerable use of. So instead, for CHESS 4.5, we made a last minute change to penalize the doubly threatened side by setting its total positional score to the worse value that side has yet experienced in the tree. This is usually worse than the pre-4.5 penalty. The change seems to have a beneficial effect in some positions.

*Pawns*

Earlier versions of our program have always involved an evaluation of pawn structures and CHESS 4.5 follows in that tradition. The basic idea is that doubled backward and isolated pawns are bad, and passed pawns and some advanced pawns are good. Doubled pawns are 2 or more pawns of the same color on the same file. Isolated pawns are those for which there are no friendly pawns on either of the adjacent files (or single file for rook pawns). Passed pawns are those whose journey to the 8th rank is no longer impeded by blocks or attacks by enemy pawns. The penalties for doubled and isolated pawns are fixed at 8 and 20 respectively, independent of the stage of the game and independent of surrounding nonpawn pieces. Two types of backward pawns are recognized: a "very backward" pawn is currently undefended by other pawns, could not be defended by advancing other pawns, and, if advanced one rank itself, would be immediately confronted by attacks by enemy pawns on both adjacent files without any compensating defenses by friendly pawns. A "lightly backward" pawn is similar to a very backward pawn except that if advanced one rank the number of enemy pawn attacks it would encounter would exceed the number of pawn defenses by 1, rather than 2. Penalties for very and lightly backward pawns are 8 and 4. Passed pawn bonuses are computed by multiplying the square (2nd power) of the rank number (2 to 7) by a factor whose basic value is 2.3 but which is modified by the status of the square in front of the pawn. If this square is blocked by an enemy piece, the factor is reduced by 0.7. If attacked by an enemy piece, it is reduced by 0.4, and if defended by a friendly piece, it is increased by 0.3. Thus if a passed pawn is on the fifth rank and the square on the sixth rank ahead of it is defended by a friendly piece but blocked by an enemy piece, the bonus would be $(2.3 - 0.7 + 0.3) *5*5 = 48$, or about half the material value of the pawn itself. If a pawn is part of a 3-file majority (the files including and adjacent to the file the pawn is on have more friendly than enemy pawns on them), then a bonus is awarded equal to 1.6 times how far the pawn has advanced (rank number minus 2). All pawns in general receive additional bonuses for advancing depending on what file they are on: multiply the rank number minus 2 by one of the factors (0, 0, 3.9, 5.4, 7.0, 2.3, 0, 0) corresponding to the pawn's file reckoned left to right from queen rook to king rook. Note that only the four center files currently have nonzero values.

Each of the different pawn bonuses or penalties is awarded independently. For example, a pawn can be both passed, double, and isolated. However, only one pawn is counted for each file, so that, for example, the doubled pawn penalty is exacted only once for White pawns on e4 and e5. A few other evaluation terms involve pawns in combination with other pieces and are described elsewhere.

*Rooks*

Rooks are given bonuses for square control, king tropism, doubling, open and semiopen files, and seventh-rank occupation. Square control means in this case the number of squares attacked by the rook including squares occupied by enemy or friendly pieces. Each square is worth 1.6.

The king tropism is a measure of nearness to the enemy king. The factor $-1.6$ is multiplied by the minimum of the vertical and horizontal distances between the rook and the king. The vertical (or horizontal) distance is the absolute difference between the rank (or file) numbers of the rook and the king. This term, therefore, is a penalty for distance rather than a bonus for closeness.

A doubled rook is a rook on the same rank or file as another friendly rook, without intervening pieces, and is awarded 8. Since both rooks are counted separately, a pair of doubled rooks is credited 16 altogether.

A rook is awarded 8 for standing on an open file, which is one which contains no pawns, and 3 for standing on a semiopen file, which is a file on which there are no friendly pawns but at least one enemy pawn that is not defended by other enemy pawns and which cannot move forward a single square without being attacked by one or more pawns.

The bonus for a rook on the seventh rank is 22.

*Knights*

Knights are evaluated for king and center tropism and development. Center tropism is a peculiar measure of the closeness of a knight to the center of the board and is computed as 6 minus twice the sum of the rank and file distances from the center of the board to the square the knight is on. This value is then multiplied by 1.6. Example: consider a knight on c3. Numbering files and ranks from 1 to 8, the knight's coordinates are 3, 3. The center of the board is 4.5, 4.5, so the center closeness is $[6 - 2^* (4.5 - 3 + 4.5 - 3)] = 0$. A knight on h5, however has closeness $[6 - 2^*(8 - 4.5 + 5 - 4.5)] = -2$, and so would be credited $(-2)^*(1.6) = -3.2$ (hence penalized 3.2, since the credit is negative).

King tropism, for knights, is computed differently from that of rooks. We compute 5 minus the sum of the rank and file distances between the knight and enemy king and multiply the result by 1.2. Example: a knight on g5 and an enemy king on h8 are 1 file and 3 ranks apart, so the tro is bonus is $[5 - (3 + 1)]^*1.2 = 1.2$. By contrast, if the knight were still on f3, the bonus would be $[5 - (5 + 2)]^*1.2 = -2.4$. Thus the knight would gain 3.6 points of king tropism by moving from f3 to g5. Like bishops, development of knights means not being on the back rank, the penalty being 9.4.

*Bishops*

Bishops are evaluated for square control and development. Bishop square control is computed differently from that of either rooks or queens. The value 7 is subtracted from the number of squares that the bishop directly attacks (or defends), not counting squares occupied by friendly pawns. The result is multiplied by 2.4. Development is defined, as for knights, simply as whether or not the bishop is on the back rank. A penalty of 11 is exacted if it is.

*Queens*

Queens are evaluated for square control and king tropism. The queen square-control term is counted differently from that of rooks. Only squares not attacked by enemy pieces or pawns are counted. Each square is worth 0.8. Like rooks, queens are penalized in proportion to the minimum of the horizontal and vertical distances to the enemy king. The number of files or ranks is multiplied by –0.8, or half as much as for rooks.

*Kings*

Kings are evaluated for safety from attack and for center and pawn tropism (if endgame). King safety is a primitive measure that is designed only to impart a vague tendency to protect the king, such as a reluctance to wander into the center of the board during the opening. Like other positional terms, it is not sufficiently large to offset the value of a sacrificed piece. Basically, the king-safety term is the product of two factors: one, a measure of how important it is for the king to be guarded at this stage of the game, and the other a measure of how well guarded the king is. The importance measure is $(NP + QB - 2)$, where NP is the number of nonpawn enemy pieces, and QB is 2 if the enemy has a queen, else 0. If the importance comes out zero or below, then king safety is ignored. The effect of the importance factor is to encourage the side whose king is exposed to trade pieces, especially queens, and to encourage his opponent to avoid such trades. In the opening position, the importance is 8. Trading queens reduces it to 5. The guardedness measure itself is the sum of several terms. These are computed as penalties for vulnerability rather than bonuses for safety.

The program assumes that a king is safer on one of the corner squares h1, h2, g1, g2, a1, a2, b1, b2, than elsewhere. A king is

punished 3.1 for not sitting on one of these "sanctuary" squares. Actually, the program is lazy and considers all 16 corner squares alike, so that a king that has somehow wandered to his opponent's sanctuary also avoids the penalty. Such a situation rarely comes up, however. Note that in the opening board position, the contribution of the sanctuary factor to the total positional score is 8*3.1 = 24.8, which means, roughly, that a king receives about a $\frac{1}{4}$ pawn bonus for castling short.

A king on one of the first-rank sanctuary squares (a1, b1, h1, g1) is penalized 0.8 for being "cramped," which means not having at least one square next to it, on the second rank, which is not attacked by any enemy pieces and not occupied by a friendly pawn. This term, which is one of the very few additions to the evaluation function since 1973, was designed to help overcome a tendency toward vulnerability to back-rank mates that we observed in tournament play (see Harris's comments on this theme, 1977). Although CHESS 4.0 did not actually get mated, its failure to push one of its king-side pawns made it vulnerable to mate threats that cramped its game. The success of this evaluation term has not been determined.

The king-safety term also determines whether the king is "sheltered" and whether opponent pieces are attacking the squares adjacent to the king. To help prevent the king from sitting on a barren region of the board devoid of the protection of friendly pawns and pieces, the king is penalized if there are not at least two friendly pieces or pawns on squares adjacent (within one king move) to it. If there is only one such piece, the penalty is 1.6, and if none (a bare king), 3.1. If no friendly pawn stands on the same file as the king, a fee of 4.1 is charged. If no friendly pawn stands on at least one of the files adjacent to the king, the king is docked 3.6. For example, if the f, g and h pawns are all missing, a king on g1 is penalized for lack of both direct and side pawn cover: 4.1 + 3.6 = 7.7. Before any piece trades, when importance = 8, the positional score contributions would be 7.7*8 = 61.6. Fines are levied if at least 2 squares adjacent to the king are attacked by enemy pieces. The penalty is 1.2*(NAT − 1), where NAT is the number of such squares attacked. This concludes the description of the king-safety term.

When a sufficient amount of material has been traded, the king acquires a desire to come out of hiding and roam towards the center of the board and toward whatever pawns lie on the board. The threshold is reached when enemy material falls below 1500 in value (e.g., a rook, bishop, knight, and 3 pawns). When the material is that low or less, the king is suddenly penalized for distance from the center. The penalty is 2.3 times twice the sum of the rank and file distances from the king to the center of the board. See the section on knights for a similar measure.

The king in the endgame is also penalized for wandering away from any remaining pawns. The penalty is 7.8*(ATPD − 6). where ATPD is the average of the "taxicab" distances of the king to each pawn, friendly or enemy. The taxicab distance is the sum of the rank and file distances, as if the king could move only horizontally or vertically to approach the pawn. This term was inspired by observing ours and other chess programs reach an ending with kings and locked pawns and then draw by repetition because the kings moved aimlessly back and forth.

*Mop-up evaluation*

The mop-up evaluator is assigned positions in which one side has a material advantage of at least 400 points (4 pawns) and the other side has no more than 700 points worth of material. In addition, the winning side must have at least one queen or rook, or must have no pawns and at least minimum mating material in minor pieces (♗ + ♘, ♗ + ♗, ♘ + ♘ + ♘, or more). All positions in the tree not satisfying these criteria are assigned to the regular position evaluator. The no-pawns criterion was based on the notion that it is easier to promote pawns than to checkmate with minor pieces alone.

The mop-up evaluation consists of two phases, the stalemate guard and the checkmate drive. The stalemate guard is a quick check for stalemate done only in the case of a lone king. If the king has no moves but is not in check, the draw score is set and EVALU8 exits. The checkmate drive is the sum of several terms:

1. The king getting mated has a great desire to avoid the edges and corners of the board. The mating side is credited 4.7*CD, where CD is twice the sum of the rank and file distances from the king to the centre of the board (See the section on knights and kings in the regular positional evaluation for a similar

measure.) Example: for a king on f3: 4.7*2*(4.5 − 3 + 4.5 − 3) = 28. If the king is driven to g2, then we have 4.7*2*(4.5 − 2 + 4.5 − 2) = 47, for an increase of 19.

2. A few plies of search are enough to show effects on the center tropism term by moves of the sliding pieces (rooks, bishops, queens) that restrict the enemy king. However, the king and his knight(s) must get close to the target king before having an effect, and so a term is added to encourage this approach: for each king and knight on the winning side add 14 minus the sum of the rank and file distances from the king or knight to the enemy king. Multiply the total by 1.6.

3. A minor factor is also added which seems to help position the winning king properly in the final stages of difficult checkmates, such as ♔ + ♘ + ♘ vs. ♔. If the winning king is on the third or sixth rank or the c or f file, it is awarded 2.

*Special evaluations*

Checkmate and stalemate are recognized during tree searching when no legal moves are found. Except for the mop-up evaluation, stalemate is not recognized by the evaluation function itself at the endpoints of the tree. Repetition of position is detected by comparing the position to be evaluated with a table of the fifty most recent positions in both the tree search leading up to the current node and the recent history of the actual game. The "fifty-move" rule is incorporated in spirit by pulling scores towards the draw value as the number of moves since the last pawn push or piece capture approaches 50. The draw score itself is not set to 0 but is modified by a "contempt" factor that depends on estimated ratings of the program and its opponent, which are set prior to the game. Another parameter pretends to know how negative the evaluation can get and still have the "better" player play for a win. These factors are usually set (for the computer tournaments) so that CHESS 4.5 will still play for a win, avoiding repetitions, etc., even though it is a pawn down. This is done so that the program will not become panic-stricken and seek a draw after having played a gambit from its opening book.

## Tree Searching in CHESS 4.5

Here we look at CHESS 4.5's tree search in detail. In summary, it looks simple (and simple-minded) enough: the search consists of a series of iterations terminated when a preset time threshold is exceeded. Each iteration is a complete $\alpha$-$\beta$ depth-first search of all the legal move sequences (except those eliminated by $\alpha$-$\beta$ cutoffs) out to the nominal depth of the iteration, with extensions beyond that depth only for sequences of capture moves (of almost unlimited depth) and also for checking moves one or two ply beyond the nominal depth. The depth of the first iteration is 2 and each subsequent iteration is a ply deeper than the preceding one. To gain efficiency, an iteration uses move-ordering and score information saved from previous iterations.

The key phrase in the above summary is "all the legal move sequences." Textbook accounts of computer-chess algorithms usually touch on the futility and foolishness of such brute-force searches, drawing analogies to the number of grains of sand in the Sahara Desert or perhaps the number of atoms in the universe. In fact, it does require maneuvering to compute a positional score at each endpoint and still achieve depths of 4–6 ply in the middle game and up to 9 or more in the endgame. It is important to order the moves so as to get many $\alpha$-$\beta$ cut-offs. In a full-width search the difference between poorly-ordered and well-ordered searching is large, and so CHESS 4.5 devotes pages of code to move ordering. It is ironic that the lazy programmer, hoping to escape the brain-racking task of selective searching (forward pruning) by searching all the moves, is forced to devote his energies to move selection anyway so as to get the near-optimal ordering necessary to make the full-width scheme feasible. The difference is that mistakes in move selection are not as fatal to full-width searches, since they only affect efficiency, whereas they can doom a forward-pruning program by completely eliminating a crucial move from consideration. However in CHESS 4.5 we succeed in cheating nature a little by using almost no chess knowledge in our move ordering. This way we have kept true to our principles, one of which is that "chess knowledge" is something that belongs in our "next" chess program. Actually, as you shall see in what follows, there is nothing magical in our ordering heuristics. In fact, at least as much of our programming effort

has been devoted to another principle of full-width searching, which is to spend as little time as possible at any given node in the tree. It is at this dubious endeavour that our program excels. On a CDC 6400 CHESS 4.5 processes from 270 to 600 nodes (calls to EVALU8) per second.

### *The BASE module*

The BASE module processes the base of the tree (the position on the actual board), whose ply level we define as zero. First, BASE looks this position up in the openings library. If found, the stored move is made immediately. Usually, it is not found. Then all the legal moves from this position are generated and tried, with instructions that the resulting ply-1 positions be processed by the PREL module, which assigns preliminary scores by using the evaluation function. The capture search is not done, but a very brief analysis of hung pieces is performed and the result is folded into a score. BASE also tells the search to omit $\alpha$-$\beta$ minimax calculations that are usually performed. The moves are then ordered on the basis of these preliminary scores.

The iterations come next, starting with a 2-ply search. BASE tries out each move in order of preliminary scores from best to worst, with instructions that the resulting ply-1 positions be processed by the FULL module and that the $\alpha$-$\beta$ minimax procedure be employed. This means that the score expectation variables $\alpha$ and $\beta$ will be maintained at each level of the tree. Before each iteration starts, $\alpha$ and $\beta$ are not set to $-(\infty)$ and $+(\infty)$ as one might expect, but to a window only a few pawns wide, centered roughly on the final score from the previous iteration (or previous move in the case of the first iteration). This setting of "high hopes" increases the number of $\alpha$-$\beta$ cutoffs. If the final score does not lie within this window, however, all of the moves must be researched with lower expectations. Each time that a BASE move comes back with a better score than any of those preceding, that move is placed at the head of the list so that it will be searched earlier in subsequent iterations. Under certain conditions BASE will terminate an iteration prematurely and use the best move found so far. This may happen, for example, if the time consumed by the current iteration is judged excessive by the time-control monitor. The time is checked before each new move is tried out by BASE.

Another way an iteration can terminate early is in the case of an "obvious" move. Since the program continues to try new iterations until a preset time threshold is exceeded, there is usually little correlation between the difficulty of a position and the time spent pondering it. In 1974, the "obvious-move" feature was added to spare us further embarassment at having to wait a minute and a half for CHESS 4.0 to respond pawn-takes-queen (assuming only one way to capture), which any of the spectators could have played after a moments thought. Now, when CHESS 4.5 finds a move which was judged clearly superior by preliminary scores (i.e., at least 156 points better than any of the others) and has maintained its first place thru every iteration up to the last one, then the last iteration is terminated after examining only this move to ensure that it hasn't worsened by more than 156 points. Now pawn-takes-queen takes only perhaps 35 seconds and CHESS 4.5 prints out "that was easy" to the amusement of all.

### *The FULL module*

The FULL module handles nodes at depths 1 thru $n-1$, where n is the iteration depth. Since end-point evaluation is done at depth $n$ or beyond, FULL is concerned mostly with more searching. The order of operations is very important in maximizing efficiency. FULL tries first to avoid any searching at all. Then it tries to search moves in an order likely to produce a quick $\alpha$-$\beta$ cutoff. Here is the detailed flow of the FULL module, divided into phases:

A. *Repetition check.* The position is matched against previous positions in this branch and also against up to 50 previous positions in the actual game. A match indicates a position repetition, which is scored as a draw, even if it has only occurred once. FULL then exits immediately. To speed up the comparison, the number of moves since the last irreversible one (piece capture or pawn move) is stored, so that no comparisons need be done beyond that point.

B. *Transpositions/refutations table look up.* Here we are interested in the evaluation stored in the trans/ref table that was described earlier. The table is probed, and if a match with a table entry is found, the stored evaluation is retrieved. Depending

on the $\alpha$-$\beta$ values in effect at the time the position was stored, the value may be a true score, an upper bound, or a lower bound. The $\alpha$-$\beta$ values at the time of retrieval are also important in determining the applicability of the stored value to the present position. There emerge three cases:

1. The value is completely valid, in which case the current node is assigned this value, and FULL returns without further tree searching or evaluation.
2. The value is partially useable, in which case processing continues, but with a narrowed $\alpha$-$\beta$ window that will increase efficiency of searches done in the subtree grown from this node.
3. The value is useless, in which case processing of this node must continue as if no match had been found.

Values in the table are good only for the current search iteration. The utility of the table increases with the depth of the iteration and the simplicity of the position. The effect of increasing depth is significant because of combinatorial effects on the number of transpositions and multiple pathways. Thus a faster machine makes better use of the table than a slower one, provided that enough space is available to store the table. On a CDC 6400 in complicated middle game positions, the table reduces search times by from zero to about 50 percent. In simple endgames the benefit is much greater, showing gains of a factor of 5 or more, with a resulting gain in search depth of up to two or three additional plies.

C. *Check evasion.* If the side to move is in check, and the depth is $n$-1, the program branches to a special routine to efficiently generate moves to get out of check, and the remaining phases are skipped. See phase C of the MINI module for the details of the algorithm. FULL phase C is skipped during the first (2-ply) iteration.

D. *Main variation.* The main predicted sequence of moves from the previous iteration was saved, and, if we are still following that sequence, we search the predicted move. Since the BASE module also searches the main variation first, this sequence is the first one searched in each iteration.

Unlike some other programs which generate all of the moves from a given position before searching any of them, CHESS 4.5. tries to generate only a few at a time and thereby save time if one of the early moves turns out to be a refutation. In phase D, all moves of one piece are generated and saved for use by other phases, and the particular move found in the main variation is selected from the generated list and searched. For bookkeeping reasons, the move from the main variation cannot be passed directly to the tree search but must be generated by one of a standard set of move generation routines. The tree search is instructed to send the resulting position to the FULL module if it occurs at depth $n$–1 or below and to MINI if at depth $n$. This is true of subsequent phases as well.

E. *Transpositions/refutations table.* The look up of the current node in the trans/ref table described previously may have retrieved a move that had been found – probably in a previous iteration – to be the best (or a refutation) in this position. This move is now searched. Most, but not all of the main variation moves also show up in the trans/ref table. Each move in the move list is flagged when searched, so that a move is not searched redundantly by different phases.

The principle of phases D and E is that a move found to be good for a particular position in a previous iteration is also likely to be good for this same position in the current iteration in which scores are computed one ply deeper. This principle has not proven to be as valid as we had originally thought, especially at the deeper ply levels close to depth n of the iteration. In fact, recent tests suggest that phase E might be more effective if it followed the captures (phase F) and the killer moves (phase G) rather than preceding them, since its skill in selecting moves, while significant, appears to be less than that of these other phases. Phases D and E are skipped during the first iteration because there are no previous iterations from which moves could have been saved.

F. *Captures.* All of the capture moves are generated by this phase. There is a routine named GENFSL that generates all moves from a given set of squares and also a routine GENTSL that generates all moves to a set of squares. Captures (except for *en passant* pawns) are generated simply by calling GENTSL with the set of squares containing enemy pieces. Then they are tried out, in order of decreasing estimated gain, down to an estimated gain of zero. The estimated gain defines a loose ordering of the captures and is simply the value of the

captured piece if that piece is undefended, or its value minus the value of the capturing piece. If defended.

Examining the captures first has a large effect on search efficiency. First, the full-width search involves lots of crazy moves and thus contains many weird positions in which pieces are left hanging. A capture is a refutation move more often than would be the case in positions reached in a normal chess game. Second, at a node at which any one of several moves would serve as a refutation, it is better to choose a capture than a noncapture, because the subtree grown from it is smaller. Not only does the capture simplify the board, reducing the number of legal moves, but it reduces the number of potential exchanges on the board, thus cutting down the capture/quiescence search done at the endpoints. A capture also reduces the opponent's reasonable options for one ply, thus improving $\alpha$-$\beta$ cutoffs.

The importance of searching capture moves early became more apparent in 1974, when a predecessor of phase E was implemented. Since phase E passes refutation moves from one iteration to the next, the refutations chosen in the first iteration greatly influence the rest of the search. In CHESS 4.0, the killer heuristic (phase G) took precedence over captures. In 1974 we put the captures before the killers and observed a two-to-one speed-up in many positions. Rarely did the change produce a slower search.

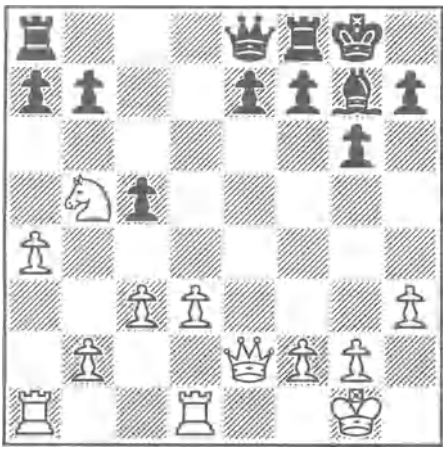


*Fig 4 Black to move must defend against the knight fork.*

G. *Killer moves.* The killer heuristic has been used by other chess programs and is basically an inexpensive attempt, based on superstition to find a quick refutation move. Consider the position in Figure 4. White is threatening ♘b5-c7, forking the Black queen and rook and winning the exchange. Most of the Black moves do not avoid the fork. Once the program discovers (by accident, as usual) that (1) ... e7-e5 is answered sharply by (2) ♘b5-c7, it would do well to remember ♘b5-c7 and try it early against the other Black moves from the diagram. This principle goes by the name of the "killer heuristic." and ♘b5-c7 is the killer move. Phase G concentrates on the idea that the immediate offspring of a given parent node are often similar enough to be susceptible to the same killer move. So, in CHESS 4.5, from and to squares of the last refuting or best move at each ply level are saved in the "killer" table so as to be retrieved by phase G the next time the same ply level is reached, which is when a new move is tried from the same parent position, unless all those moves have been exhausted. Of course the killer move may not be legal at some of these nodes, in which cases it is ignored. Generation and listing of the move is accomplished as in phase D, once it is determined that a piece of the proper side stands on the from-square. If the killer move is best or causes a refutation, then of course it will be saved to be used again for the next offspring. However, if a different move proves best, then it will replace the killer table entry for that ply level and will be used the next time. A move that captures the piece the opponent just moved is an exception. It is not put into the table since it most likely refutes only that one move.

In 1975, for CHESS 4.5, an enhancement was made to the killer heuristic that is related to the transpositions/refutations table. For phase B to be most effective, the killer heuristic should be used as often as possible, since the use of the same move over and over increases the number of transpositions. Suppose that, most, but not all, of the moves from a given parent node can be refuted by a single move. Then this move will be used over and over until one of the offspring positions is reached for which it does not work. For that node a different move will be found to be best and

will be stored in the killer table. Then perhaps follows a series of off-spring nodes for which the original killer move would once again serve as a refutation. However, knowledge of it has been lost since its place in the table has been taken. It might be rediscovered and thus reappear in the table, but it is likely that some other move would take its place, and would serve as a competent refuter, except for its negative effect on the frequency of transpositions. So for CHESS 4.5 the table was expanded to hold two entries per ply level – first and second most "popular" killers, together with popularity counts for each one. Both killers are tried in phase G, the most popular one first. When a FULL node is exited, the best or refuting move (if any) is compared to those in the table. If already in the table, its popularity is upped by one unit. If not, it replaces the lower entry and acquires a popularity of one. Thus a long-standing killer move, with many quick refutations to its credit, will not be hastily lost.

H. *Killer of 2-plies lower.* This phase tries out the most popular killer move of 2 plies shallower in the tree. This move is of no use, of course, if it already forms part of the current branch, since it can't be played twice in succession. This phase sometimes produces a refutation when the previous attempts did not.

I. *The rest of the moves.* The previous phases attempted to find a quick refutation move, or, if not that, then at least a good move that would set a respectable $\alpha$-$\beta$ expectation value. If there was no refutation found, then phase I must set about the business of searching the remainder of the legal moves. Of course any of these may still cause a refutation and terminate the search. First the rest of the moves are generated and added to the end of the moves list, which already contains some moves generated but not searched by previous phases. To impose some admittedly very weak chess sense on the order of these moves, they are generated in three steps:

1. Moves of hung (attacked and undefended) pieces
2. Moves of other attacked pieces
3. All other moves

Then the complete list is searched from beginning to end in two phases. Pass 1 covers only moves of pieces to squares not attacked by enemy pieces, and pass 2 includes all remaining moves.

In 1974, a time-saving feature was put into this phase, for nodes at ply level $n$–1, to take advantage of the evaluation function being dominated by the material balance. Consider a move $m$ at ply $n$–1 (i.e., a move that results in a position at ply $n$ and therefore is eligible for terminal evaluation), and suppose that $m$ is neither a capture nor a check. Then the material score after the move $m$ is the same as that before $m$. The positional score after $m$ is not known before $m$ is searched but is presumed, as we shall see in the description of the MINI module, to lie within the relatively narrow bounds of the minimum and maximum positional scores so far encountered in the whole search. Therefore, the total score after $m$ can be predicted to lie in a certain range. The capture search conducted beyond move $m$ can alter the score, but only to the advantage of the opponent of the side playing $m$. So already at ply $n$–1 a maximum value can be placed on the move $m$. If this value is less than the minimum expectation specified by current value of $\alpha$-$\beta$ , then the move $m$ is discarded without ever being searched. This algorithm has its greatest utility in branches in which an early move gives up much material for no good reason, so that most of the moves at ply $n$–1 are discarded as unable to win it back. In some positions the saving approaches 50 per cent. The requirement that $m$ not be a check stems from the effect of a check being unpredictable at level $n$–1. A few other criteria for the ineffectuality of move $m$ are investigated before $m$ is discarded – these involve the 50-move and repetition-draw rules.

J. *Cleaning up.* When all the phases are complete, FULL confirms that at least one legal move was found. If none were, then a draw is scored if the side to move is not in check, and a checkmate is scored otherwise. This checkmate score is larger the smaller the ply number so as to prefer fast checkmates over slow ones. If there was no checkmate or stalemate, then the best move (if any) is saved for the trans/ref and killer tables. If the ply level is one, the lack of a best move may be due to the rejection of all the moves by the artificially narrow $\alpha$-$\beta$ window that was imposed at the start of the search. In that case, the window is enlarged to relax the opponent's expectations, and phase I is reexecuted to try all the moves again. If this re-searching is not necessary,

then FULL is done, the position is abandoned, and control is returned to the parent node one ply level back.

*The MINI module*

The MINI module completes the tree search. MINI processes all nodes at or beyond depth $n$, the terminal depth of the current iteration.

A. *Repetition check.* The check for repetition of position is done the same way as in the FULL module.

B. *Transpositions table look up.* The table is inspected as in the FULL module. Even at the "terminal" depth of the iteration, an evaluation from the table can still save considerable work — notably, the computation of the positional score and the capture and check- quiescence searches.

C. *Check evasion.* If the king is in check a branch is made to a special routine that generates moves that specifically try to get out of check. All of these moves are searched, so the effective terminal depth in this case is $n + 1$. Phase C of the FULL module also may branch here if the king is in check at level $n - 1$. The use of this special routine saves time because the move generation usually done in the FULL module does not screen out moves that leave the king in check. These moves are rejected only after tree-search control has updated the board and data base, a time-consuming process (1.5 millisecond or so on a CDC 6400), and then finds the resulting position illegal. In positions in which the king is already in check, most of the moves are illegal. This phase avoids generating most of these moves by considering only the following plausible alternatives:

1. Try to capture the checking piece
2. Try to move the king to an unattacked square
3. Try interpositions
4. Try an *en passant* capture

Checkmate is scored if all these remedies fail.

D. *End-point evaluation.* This phase provides a score using either the regular or mop-up function described earlier. The regular evaluation is done in two steps. First, the material score is computed. This is done very quickly, because values are saved in a hash table and used over and over. All positions which contain the same number of pieces of each type have the same score. The number of pieces of each type form the "material signature," which is incrementally updated during the search, and is the key to the material balance hash table.

Now comes a time-saving trick. Our positional score is crude and is therefore weighted lightly. It would be nice if we could avoid calculating it at all. This can be done if the positional score is presumed to lie within certain bounds and these bounds bear a certain relation to the current values of $\alpha$ and $\beta$. Rather than compute analytically the extremes of values taken on by the evaluation function, we assume, somewhat erroneously, that the positional score will lie within certain limits.

Consider the preliminary scores given out by the PREL module for the purpose of sorting the moves at the base node. When the first iteration begins, positional scores are presumed to lie within a window whose limits are 75 points above and below the average value of the positional components of all these preliminary scores. This range is approximately one and a half pawns wide. Then each time a positional score is found outside these limits, the range is stretched just enough to include it. Usually the range of scores encountered during the search somewhat exceeds 150 points, so the limits become the minimum and maximum positional scores encountered so far in the search.

To make use of this information, we note that the $\alpha$-$\beta$ algorithm treats all scores less than or equal to $\alpha$ alike, and all scores greater than or equal to $\beta$ alike. If a score is less (greater) than $\alpha$ ($\beta$), it doesn't matter by how much. Now the total score is the sum of the material and positional terms. If the material term is so low that the addition of even the assumed maximum positional score could not pull it up to $\alpha$, then the positional score need not be computed at all. This is similarly true if the material term is so high that even the minimum positional score could not pull it down below $\beta$.

If the base position is relatively quiet, perhaps 50 percent of the end-point positions in the tree actually require positional scores. If the base position is tactically alive, and there are many forcing moves, this fraction may drop to as low as 10 percent. It seems to take about three times as long to process a node with full positional

scoring as to process one without. For some reason that is lost in antiquity, we do not use this same trick in mop-up positions. It is not as important here, however, since the mop-up evaluation function is much faster.

The material and positional scores are added together and tapering, based on the 50-move rule, is folded in. The resulting score is not the final judgment on the position, however, but is considered only one of several alternatives, available to the side on the move, which are all minimaxed together. The side to move has the choice of accepting the computed score (standing pat) or of trying captures (and perhaps checking moves) which might improve it. Of course it is possible that the just-computed score is enough to cause an $\alpha$-$\beta$ refutation and so eliminate the need for further searching from this node.

E. *Capture search.* The capture search tries out capture moves in order of decreasing estimated gain, as in phase F of the FULL module, except that even moves with negative estimated gain may be tried. There is a criteria for rejecting some capture moves *a priori.* The material score (assuming a successful capture) is computed. If this score, plus the most favorable possible positional score, is not enough to reach the current $\alpha$-$\beta$ expectation, then there is no sense trying it, since in the position resulting from the capture the other side will have the option of "standing pat." Positions resulting from the capture search are fed back into the MINI module, except that the transpositions table look up is bypassed. Because the look up is bypassed, the storing of new information in the table for this position also does not take place. As a result, scores and best moves for positions within the capture search are not saved for use by the MINI module in subsequent occurrences of these positions. Also, the check-evasion code of phase C is bypassed for nodes more than 2-plies beyond the nominal depth $n$ of the current iteration. In 1975 pawn promotions were added to the capture search on the grounds that they also immediately change the material balance.

So the main business of the capture search is to explore sequences of capture moves, with each side having the option at alternate levels of accepting the existing material balance or of trying more captures. The depth of these branches is limited only by the maximum capacity (30) of certain arrays. In practice, branches of 10 to 15 ply are not uncommon, although their utility is doubtful. Theoretically, the search could explode and hangup indefinitely looking at unlikely piece exchanges, but it usually fades out because of its self-limiting nature: each capture removes a piece from the board and so reduces the potential for further captures. Typically, the capture search constitutes about 20 to 70 percent of the total search. On rare occasions the search does blow up and this fraction reaches as high as 95 percent. It is quite likely that when the capture search exceeds 50 percent or so of the total, it is no longer cost effective in comparison to using just the primitive static exchange analysis such as is done in the PREL module. This hypothesis has not been tested, however.

The capture search is a special case of a tactical "quiescence" search – a search whose purpose is to reach "quiet" positions in which the static evaluator will in theory make more reliable decisions. However, the presence of capturable pieces on the board is not the only feature that makes a position tactically nonquiescent, nor is a search of capture moves alone a sufficient remedy. That our quiescence search is top-heavy with captures is due mostly to its simple, self-limiting nature and our laziness. A balanced quiescence search containing the most promising capture, threat, and threat-evasion moves, together with sophisticated time-control to prevent explosions, could yield much more accurate tactical analysis. Harris discusses this in some detail (1977).

F. *Check-quiescence search.* For nodes satisfying certain strict criteria, a more general quiescence search replaces the simple capture search of phase E. The *raison d'etre* of the quick-quiescence search is fear of embarrassment – the embarrassment of falling into a three-move mate after having spent a minute-and-a-half doing a 4-ply search.

This phase averts the majority of such mates by essentially adding 2 additional plies onto branches in which the last 2 moves by one side were checks. Of course, these additional 2 plies would not prevent a 4-move mate (after a 4-ply iteration) or a 3-move mate in which not all of the three moves were checks. But we considered

these possibilities either less likely or less embarrassing. The exact criteria for check-quiescence searching are that:

1. The node be at depth n or n+1
2. The last 2 moves (in the current branch) of the side to move be checks
3. The side to move not be winning by more than 400 points (a little less than a rook) as measured by the current $\alpha$-$\beta$ expectation value.

If these criteria are met, all capture, pawn-promotion, and checking moves are searched, with the resulting nodes fed back to the MINI module. First, capture moves are searched in order of decreasing estimated gain. There is no special routine for generating only checking moves, so all the moves are generated and the checks selected from them. Check-quiescence finds mating combinations for both sides and also some checking sequences that lead to a win of material. It does not add more than a few percent of search time as averaged over a whole game. However, at the drop of a hat (or the slightest relaxation of the strict criteria for its application), check quiescence blows the search sky-high. This accounts for the severe limitations on it and illustrates the necessity for tight controls on a quiescence search.

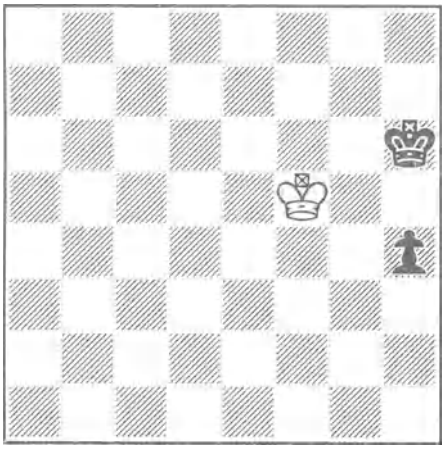


*Figure 5 White with the move can capture the Black pawn.*

This concludes the description of the MINI module and in fact the entire tree-search algorithm. We are now in a position to examine an example in detail. For simplicity, we will analyze the position depicted in Figure 5, in which each side has only a few legal moves. White has a king, Black a king and a pawn. It is clear that White's task is to stop the Black pawn from queening. The initial expansion of the base position indicates that White has six legal moves:

| Move | PREL score | Move | PREL score |
|---|---|---|---|
| ♔f5-e4 | −116 | ♔f5-e5 | −121 |
| ♔f5-f4 | −114 | ♔f5-e6 | −129 |
| ♔f5-g4 | −106 | ♔f5-f6 | −127 |

These are ♔f5-g4, ♔f5-f4, ♔f5-e4, ♔f5-e5, ♔f5-f6, ♔f5-e6 when ordered by score. ♔f5-g4 heads the list because the evaluation encourages the king to be near the pawn. Its preliminary score, however, does not differ very much from that of any of the other moves, since the evaluation function does not understand that the pawn will be captured two half-moves later.

Below are shown the actual trees of the first 2 iterations. In parentheses are the module that generated the move (B = BASE, F = FULL, M = MINI), and also, for MINI and FULL, the letter of the phase within that module.

**2-ply iteration**

| Ply 1 | Ply 2 | Ply 3 |
|---|---|---|
| ♔f5-g4 (B) | ♔h6-g6 (FI) | ♔g4×h4 (ME) |
| | ♔h6-g7 (FI) | ♔g4×h4 (ME) |
| | ♔h6-h7 (FI) | ♔g4×h4 (ME) |
| | h4-h3 (FI) | ♔g4×h3 (ME) |
| ♔f5-f4 (B) | ♔h6-g6 (FG) | |
| ♔f5-e4 (B) | ♔h6-g6 (FG) | |
| ♔f5-e5 (B) | ♔h6-g6 (FG) | |
| ♔f5-f6 (B) | ♔h6-h5 (FI) | |
| ♔f5-e6 (B) | ♔h6-g6 (FG) | |

(1) ♔f5-g4 is the first move searched at the base node because of its PREL score. FULL tries out all 4 responses: (1) ... ♔h6-g6, ♔h6-g7, ♔h6-h7, and h4-h3, and the MINI capture search finds they are all met by (2) ♔g4×h4 (or ♔g4×h3). The next 3 moves tried by BASE (1) ♔f5-f4, ♔f5-e4, and ♔f5-e5, are all refuted by the killer move (1) ... ♔h6-g6, since $\alpha$-$\beta$ now reflects White's expectation of winning the pawn, something which MINI did not see as possible after these moves. The BASE move (1) ♔f5-f6 is refuted by (1) ... ♔h6-h5 the killer ♔h6-g6 having been tried and found illegal. But (1) ... ♔h6-g6 comes out of hiding to refute the last BASE move, (1) ♔f5-e6. Thus (1) ♔f5-g4 is best and so retains its position at the head of the moves list.

**3 ply iteration**

| | | |
|---|---|---|
| ♔f5-g4 (B) | ♔h6-g6 (FD) | ♔g4×h4(FD) |
| | ♔h6-g7 (FI) | ♔g4×h4 (FF) |
| | ♔h6-h7 (FI) | ♔g4×h4 (FF) |
| | h4-h3 (FI) | ♔g4×h3 (FF) |
| ♔f5-f4 (B) | ♔h6-g6 (FE) | ♔f4-e3 (FI) |
| ♔f5-e4 (B) | ♔h6-g6 (FE) | ♔e4-d3 (FI) |
| ♔f5-e5 (B) | ♔h6-g6 (FE) | ♔e5-d4 (FI) |
| ♔f5-f6 (B) | ♔h6-h5 (FE) | ♔f6-e5 (FI) |
| ♔f5-e6 (B) | ♔h6-g6 (FE) | ♔e6-d5 (FI) |

Once again, (1) ♔f5-g4 is tried first. The FULL responses (1) ... ♔h6-g6, ♔h6-g7, ♔h6-h7, and h4-h3, are still answered by (2) ♔g4×h4 (or h3), which, however, is found by FULL instead of MINI, since it lies within the 3-ply horizon. In response to (1) ... ♔h6-g6 (2) ♔g4×h4 is a main variation move. All other moves besides ♔g4×h4 are discarded by the time saving feature of phase I, since they cannot (within the horizon) meet the established expectation of winning a pawn. In response to (1) ... ♔h6-g7, ♔h6-h7 and h4-h3 (2) ♔g4×h4 (or h3) comes from the FULL capture phase and is a simple $\alpha$-$\beta$ refutation. The next three BASE moves (1) ♔f5-f4, ♔f5-e4 and ♔f5-e5, are refuted, as in the 2-ply iteration, by (1) ... ♔h6-g6, which is recalled from the trans/ref table. Notice that in response to (1) ... ♔h6-g6, only one move is tried in each of the three cases. Once it is confirmed to be legal (to avoid getting into the checkmate or stalemate code), the others are discarded by the phase I time-saving feature. Finally, the last 2 moves at BASE, (1) ♔f5-f6 and ♔f5-e6, are refuted similarly to the previous ones. So the 3-ply iteration discovers the same simple truth that the 2-ply iteration did, that (1) ♔f5-g4 is best because it quickly wins the Black pawn. The predicted moves are (1) ♔f5-g4, ♔h6-g6; (2) ♔g4×h4.

Although the base position is trivial, CHESS 4.5's search exhibits many of the features seen in much larger trees in more complicated positions. Note that because of $\alpha$-$\beta$ cutoffs and the time-saving feature, the tree does not appear to be "full width." The whole 3-ply search here was completed in about 100 milliseconds. In a tournament game the search would have gone out to perhaps 12-ply to get the same result, since the program lacks the sense to see that since White can force a position in which all material is gone, the game is necessarily drawn.

**Program performance**

Experience shows that the features of the present evaluation function coupled with a brute-force search of 3 to 10 plies (about 5 in the middle game) are sufficient to motivate CHESS 4.5 to do something "constructive" in most positions. Of course, what is constructive to the program may be a blunder to a Master, but at least the program rarely lapses into shuffling pieces back and forth aimlessly. In a typical game against a weaker opponent, the program keeps busy by maneuvering, with varying degrees of success, to optimize its notions of pawn structure, piece mobility, and general motion of pieces towards the enemy king, while it waits for a tactical blunder by its opponent. Once the program obtains a substantial material advantage, it can usually promote pawns and win more material until its mop-up algorithm drives the enemy king to a mate. Sometimes it wins in the middle game with a direct assault on the king without having to promote pawns.

**Conclusions and perspective**

What have we learned from the experience of creating CHESS 4.5 and its predecessors? Although CHESS 4.5 is more-or-less a Shannon type-A program, it is only one arbitrary example of a large number of possible programs that would fit into that same category. So one should be careful not to draw too many conclusions about the prospects of Shannon-type programs in general. The features of its evaluation function and tree search were determined often by the whim of the moment. Certainly they were not the result of a comprehensive, steady, and coherent research plan designed to exhaust the potential of brute-force tree searching. Sometimes we have been mistakenly pictured as hard-working, government-financed full-time exponents of the "big tree." In reality, the program has been a part-time effort, with long stretches of inactivy punctuated by occasional panicky spurts of attention.

Now, having related CHESS 4.5's unsystematic mode of development, we can reflect on what it is we have really constructed. CHESS 4.5 can be approximated by the following theoretical model: Imagine that we have an evaluation function $f$ that — when applied to any chess position — yields a numerical value, dependent on that position alone, which in some sense measures how well

White is doing. A positive value means White has the superior winning chances, and the larger the value, the greater is the likelihood and imminence of victory. Now imagine a chess program which generates the moves from the base node and applies $f$ to each of the ply-1 positions reached by those moves. Then, clearly $f$ can be used to order the moves and in particular to select a "best" one from among them, so that we have a full-width 1-ply search. Here $f$ is understood to consist of not only a static score but also the results of any quiescence search applied. Now imagine a program which applies that same function $f$ to all of the ply-2 nodes of a full-width $\alpha$-$\beta$ 2-ply search. Heuristics may be used to order the second ply moves so as to produce more cut-offs, but these affect only the speed and not the result of the search and are not of interest here. This 2-ply program also picks a best move from the base node. If one chose to disable $\alpha$-$\beta$, the program would produce a score for each of the base moves, just as the 1-ply search did. In fact we could pretend that the 2-ply search was really a 1-ply search, but with a different evaluation function applied at the ply-1 nodes. Now imagine an *n-ply search with the same function* $f$ applied at the ply-$n$ nodes. Once again we can pretend that the backed-up values at ply 1 were determined by some imaginary evaluation function applied directly at ply 1. This imaginary function embodies the real nature of the program, since it picks the move the machine would actually make. In mathematical terms, it is as if we had a depth operator that, when applied to an evaluation function $f$, yielded an imaginary "smarter" function $f_n$ that, applied merely at ply 1, would simulate the results of a full-width $n$-ply search that applied $f$ to its end-points. What is the relation of $f_n$ *to* $f$ itself? The answer to this question should give us an understanding of the full-width search technique. Obviously $f_1 = f$ and $f_2$ has similarities to $f$ but also some important differences. For example, if $f$ understands (because of a capture search or static analysis or whatever) that it is bad to make a move that leaves one of one's own pieces attacked and inadequately defended, then $f_2$ understands that it is good to make a move that forks two enemy pieces, since at 2 ply at least one of the forked pieces is still capturable. And $f_3$ understands additionally that it is good to make a move that prevents a forking maneuver by the opponent. The point is that if $f$ contains a particular chess concept such as material gain, then the depth operator in a peculiar way transforms, enriches, and extends the concept, by expressing the result of projecting this concept out to the end-points of an $n$-ply search. This notion holds for positional ideas as well. If $f$ likes knights in the center of the board, then $f_5$ finds 3-move pathways to get them there. So the full-width search program deals not just in the ideas imparted to it by its authors but also in subtle extensions of them coaxed from the program by a costly expenditure of computing time. The basic axiom of full-width searching is that given a reasonable $f$, $f_{n+1}$ is better than $f_n$, so that given enough time, we might as well do another iteration. But what is a reasonable $f$? Is it enough that the values of $f$ correlate positively with the game-theoretic values (win, lose, draw)? Tests that pit CHESS 4.5 against itself at time odds tend to confirm the basic axiom fairly strongly. However, the $f$ of CHESS 4.5 is quite rudimentary, and the search depths achieved are necessarily low. There are many interesting aspects, beyond the scope of this discussion, to the question of what happens to $f_n$ as one increases $n$ while keeping $f$ fixed. One example is the questionable validity of the assumption made by our search model that the opponent will use the same algorithm as the computer in choosing its moves. Thus as $n$ increases, the program "sees" more, but it also assumes erroneously that its opponent will necessarily see the same things. (See Michie (1976) for a discussion of this and related topics). If we do assume that $f_n$ in general improves with $n$, then do different $f$'s (or different aspects of the same $f$) improve at different rates? We conjecture that the better $f$ already is, the more it will benefit from a deeper search. If this is true then we should work on improving the quality of evaluation functions, rather than on small increments in search efficiency. This brings us to the next topic of discussion: what might one do to improve CHESS 4.5?

There is controversy over how to write better chess programs. Some think that gradual changes to a program like CHESS 4.5 are the way to go. Others think that CHESS 4.5 is heading in the wrong direction and an entirely different approach is needed. It is our belief that both of these positions are

valid, and that, in fact, one could arrive at a radically different and more powerful program even by "gradual" changes to a brute-force tree-searcher like CHESS 4.5. The weakest parts of CHESS 4.5 are its evaluation function and quiescence search – its *f*, the brains of the program. Our present *f* is blind to the simplest phenomena. The evaluator gladly accepts a position in which the computer is a knight ahead although its king is out in the center of the board surrounded by hostile enemy queens and rooks. In a king-and-pawns endgame, that a passed pawn is outside the square of the king and so queens by force is discovered only by a grueling 9-ply search. The list goes on and on. But why is the program so devoid of chess knowledge? It is not because we are deliberately trading it for speed of execution. For example, in a position in which the king is surrounded by enemy pieces, the program is still wasting time worrying about doubled pawns. In these situations more knowledge would probably save time in the evaluation. Similarly, some sensible heuristics in the quiescence search would restrict it to a balanced selection of tactical tries rather than letting it get lost in a forest of capture moves. It is not quite true to say that we don't know how to feed additional knowledge to the program. Certainly we could teach it something about the square of the king. Our problem is that the programming tools we are presently using are not adequate to the task. Too much work is needed to encode the sizeable body of knowledge needed to significantly improve the quality of play. The macro language developed for EVALU8 is hopelessly inadequate. If we had put in a few hundred man hours writing statements in this language, we could have substantially improved the evaluator between CHESS 4.0 AND CHESS 4.5. Instinctively we have recoiled from such an effort. Intuition tells us that "there ought to be an easier way" and that our time would be better spent finding it than in hacking away at make-shift solutions, especially if we are aiming our sights at the shimmering mirage of master-level play.

This lack of programming tools has plagued the whole field of computer chess. With the proper tool one might accomplish in a day a job that had been put off for years. Although serious efforts are underway to overcome this deficiency, the number of people working on this is surprisingly small, and their research is still in its infancy. When this research begins to produce results, the domination of the computer-chess tournaments by old-fashioned programs like CHESS 4.5 will come to an end.

These programming tools could be developed at several levels. Each successive step would be more sophisticated and should take computer chess close to the realm of artificial intelligence. The first step for our group might be just a modest structured programming language with records, sets, lists, other user-defined data types, macros, and user-defined operators. Within this language one would embed definitions of data elements and operators which are suitable for chess programming. Modularity, simplicity, and elegance would be stressed. For the last few years we have given some thought to the problem of what language to use as a base and what kinds of chess-specific data type and operations to embed in it. Completing this task will probably take more time than actually implementing a chess program within the finished design. This design will not look very impressive to a computer scientist, but it should make chess programming a lot easier for us. Still, this design will go only a limited distance in solving what we see as two major problems in chess programming: the difficulty of specifying chess knowledge to the program and of representing this knowledge internally. It will be awkward to express chess notions in the form of statements of a conventional programming language, even if the data-types and operators have been custom designed. The next logical step therefore would be to develop a language for expressing statements about chess which would not require an understanding of computer programming. The program would preprocess these statements and convert them into an internal form (perhaps linked lists or some other form of associative data base) for use during actual execution. The TYRO program of Zobrist and Carlson (1973) is an experiment in this direction. The language would be tailored for ease of use by the person writing it, perhaps resembling a subset of English. Eventually one would like to be able to introduce strategic concepts of the kind strong players use. A hypothetical example might be: "In positions of the classical variation of the

King's Indian Defense, Black should try to establish control of c5 before developing his queen's knight."

What is the role of the quiescence search in a full-width brute force program? Presumably, as more chess knowledge is introduced into the program, the quiescence procedure will share in this bounty and become cleverer in the way it selects moves to be searched. Although the quiescence search as a whole may, like the evaluation function, depend only on the terminal position at which it is conducted, the processing of nodes within the quiescence subtree need not have this restriction. In fact, information would ideally be passed along the branches of the subtree as in Berliner's tactical analyzer (1974.) A smart quiescence procedure should have all the characteristics of a selective (Shannon type-B) search. But if we pretend to be able to do selective searching, what is the sense of grafting such a search onto a full-width procedure? Why shouldn't selective searching be used throughout? One might ask: "What is a smart search like you doing in a dumb program like this?" The answer is that as the quiescence becomes more sophisticated, it can be entrusted with greater responsibility. In the limit, the quiescence procedure would take over and become the whole search as the number of full-width plies is reduced to one. And so the transition from an old-style program to a modern one would be effected. But does this method of transition make sense, with an itermediate stage in which full-width searching and smart quiescence searching co-exist? A plausible justification is that it provides a method of handling time control in a "production" situation such as a tournament chess game. The problem of time control is very important in a chess program. When the program spends time searching a subtree or evaluating a position it implies a decision as to the cost-effectiveness of that search or evaluation. Only a few percent of the chess positions encountered in tournament play permit a complete solution by a program in a reasonable amount of time. It will be difficult to build time-control decisions into every nook and cranny of the program's thinking processes and yet this capability is essential for a selective search. For the present it may be possible to isolate the time-control process in a simple way by using iterated searching and by restricting the quiescence and evaluation procedures so that they cannot blow up. In a position in which the program's knowledge is small, so that its complete search take only – say – 5 seconds, the safest way (if we believe in the basic axiom mentioned above) to utilize additional time is to add another 1-ply layer of full-width searching to the base of the tree. As we have pointed out earlier, leaving out moves early in the tree is dangerous. The deeper the tree extends, the more intelligent must be the criteria for pruning at shallow nodes.

The speculative model of chess-program evolution sketched above is only one of several possible routes to stronger programs. We believe that the common element of any such efforts is that they involve the addition of more intelligence to chess programs. This intelligence may take many different forms. The Russian group of Adelson-Velskiy, Arlazarov, and Donskoy (1975) have expressed strong belief in brute-force full-width searching. But they believe they can greatly improve the depth of search by the intelligent use of knowledge of the rules of chess to prune away branches of the tree. The program would do this by rigorously proving that these branches lead to positions not necessarily indentical to others already seen, but similar enough so as to have the same evaluations. On the other hand, some groups believe, perhaps rightly, that an advanced chess program will need to do very little tree searching. In any case, the whole effort to build more intelligent chess programs is just beginning, and I believe we can look forward to some very interesting results over the next several years, even including programs that are able to monitor their own performances and use the information gathered to alter their algorithms and thus learn on their own.

# 2.7 Cray Blitz

**Robert A. Hyatt, Albert E. Gower and Harry L. Nelson**

First published in Advances in Computer Chess 4, pp. 8-18. Reprinted by permission of Robert Hyatt.

**Abstract**
The computer chess program Cray Blitz is currently the World Computer Chess Champion and the North American Computer Chess Champion. In human competition, Cray Blitz has obtained master level performance ratings. Cray Blitz runs on the most powerful computer system currently available, the Cray X-MP multiprocessor machine.

**Introduction**

Cray Blitz was developed in late 1975 as an experiment in artificial intelligence using computer chess. It was improved over the next year and was entered in the 1976 ACM Computer Chess Championship in Houston, Texas. It finished in a three-way tie for second place. In each ACM tournament since then, Blitz (as it was then called) finished in the middle of the field, running on relatively slow computer systems.

After another mediocre finish in 1979, it became apparent that more computing power was necessary to be competitive with the CDC Cyber 176 that Chess 4.9 was running on and to be competitive with the special-purpose hardware which had been developed for Belle. The management at Cray Research was asked to support Blitz with computer time and they agreed, realizing that the publicity and public relations would more than offset the cost of the machine time required for testing and for tournament competition. Cray Blitz was born and began its climb to the top of the computer chess world.

After adapting Blitz to the Cray-1 computer, which, in 1980, was the fastest machine available, its level of play was dramatically improved. During the first year technical problems prevented Cray Blitz from finishing in the top two in the annual ACM tournament, but it was rapidly improving. In fact, it won the first ACM computer speed chess championship that year by defeating all programs in a round-robin tournament.

As the 1981 ACM tournament approached, Harry Nelson of the Lawrence Livermore National Laboratory joined the team of Robert Hyatt and Albert Gower and became a major contributor to the success the program would soon have. He brought with him a techinical understanding of Cray-1 Assembly Language (CAL) and the peculiar timing considerations necessary to develop high speed code for the Cray-1 computer system. At this time, Cray Blitz was examining approximately 3,000 nodes per second and consistently doing six ply searches. Before the 1981 ACM tournament, Cray Blitz was entered in the Mississippi State Closed Championship in September. It surprised everyone by finishing in first place with a perfect 5-0 score including a win over a USCF rated Master. Furthermore, it earned a provisional rating of 2258 for its performance. In the 1981 ACM tournament, Cray Blitz finished in a tie for second place.

During the next year, Nelson's influence became much more evident as additional assembly langauge was developed. As the 1982 tournament approached, Cray Blitz had been improved to the point that it was examining 5,000 to 8,000 nodes per second and was performing seven ply searches. Cray Blitz finished in a tie for first place with Belle, NUchess and Chaos.

The next major advance in performance came again during 1983, as preparations for the World Computer Chess Championship in New York were underway. Again, Nelson made major improvements in speed with CAL, to the extent that Cray Blitz was examining 10,000 to 12,000 nodes per second. Cray Research offered the use of the new Cray X-MP computer, a faster

machine with multiprocessing capability, which was quickly accepted. The basic machine is 30% faster than a regular Cray-1 and additionally has two processors with software features that allow user programs to take advantage of both processors simultaneously. Nelson modified the CAL routines to allow multiprocessing, and Hyatt modified the tree search algorithm to take advantage of both processors in a parallel search manner (to be described later). As the 1983 ACM event approached, Cray Blitz was searching 20,000 and 30,000 nodes per second, and was usually searching to a depth of eight plies. By defeating Belle in the final round, Cray Blitz finished in first place and won the 4th World Computer Chess Championship.

Since the 1983 tournament, additional work has been done on the CAL routines, along with searching strategy modifications, so that Cray Blitz is now searching 40,000 to 50,000 nodes per second. A new Cray X-MP with four processors will soon be available for a potential two-fold increase in computing speed. By the next ACM tournament, Cray Blitz should be searching over 100,000 nodes per second.

**Tree searching overview**

Cray Blitz is a full width (or brute force) tree search program which uses the iterative deepening algorithms for timing control. It also has the usual "killer" heuristic and uses the alpha-beta backward pruning algorithm. The program searches to a fixed depth, extends this with the quiescence search, and then evaluates the resulting terminal position. There are several improvements in this basic algorithm that will be described in detail.

When performing a search to depth "N", Cray Blitz does not count a ply if the side on move is in check. The reason for this is that since there are usually very few legal moves in such a case, the extra time spent in additional analysis on these variations is offset by the reduced number of moves to consider at this ply. One enhancement to this idea is to consider captures on squares adjacent to the king as "checks" also. A position was reached in a test game where the program would have found a mate if the first move (♗×h7) had been a check. Since the king was at h8, ♗×h7 was not a check, and the search was not carried deeper to discover the consequences of this capture. Moving the king to g8 caused the program to quickly find the mate since ♗×h7 was a check.

After reaching the quiescence search depth limit, Cray Blitz uses a selective quiescence search to reach a terminal position. While most programs analyze all possible captures and recaptures until no more are available, Cray Blitz has a routine which eliminates most of the captures as futile. This eliminates a large portion of the potential quiescence search tree. Cray Blitz saves a considerable amount of time by analyzing the captures with a simple exchange evaluator before deciding which captures to include in the quiescence search.

Another search enhancement relative to examining checks is the inclusion of some checking moves in the quiescence search. While searching to some fixed depth, Cray Blitz will also include some checking moves in the quiescence search in addition the capturing moves usually examined. Up to two additional checking plies can be included in the quiescence search if there were checks in the basic search depth. This would extend the depth of tactical branches further to find/avoid mates and losses of material brought on by a direct attack on the king. These two checking plies must be part of a consecutive series of checks that are part of the quiescence search. If each move in the quiescence search is not a check, then the opposing side could "stand pat" and avoid everything by doing nothing. Two important points here are: 1) in the quiescence search, if the side on move is in check, all moves MUST be tried so that the search can be sure that it is not mate; and, 2) these two checking plies do not count checking moves that are also captures.

To avoid terminating the search prematurely, the program evaluates positions with passed pawns more thoroughly under certain circumstances. The program includes pawn moves to the seventh rank as quiescence moves, even though they are not captures or checks. These moves threaten to change the material balance and need to be carefully analyzed. In order to better evaluate the effect of moving a pawn to the seventh rank in the quiescence search, these moves are treated as checks so that the opposing side is forced to stop the pawn, capture it, or allow it to promote. Stopping

the search with a pawn on the seventh rank could result in severe evaluation errors.

## Hash Tables

Since Cray Blitz is an exhaustive search program, it could spend considerable time evaluating the same things over and over because different branches of the search tree have certain positional attributes in common. To avoid wasted work, three hash tables are used.

The first table is the transposition table used by most exhaustive search programs to avoid re-searching identical sub-trees. This table is the largest one in the chess program and is used to store a position, the value of the position, and a suggested best move from the position. Unfortunately, the value of the position is sometimes not known, only that the position is better or worse than some other position. This case is also considered, and is the reason for storing the suggested move. If the value (or upper/lower bound) is not good/bad enough to cause a cutoff or stop the search at this point, the suggested move can be retrieved and made, possibly causing an alpha-beta cutoff and saving a move generation. In normal positions, the move generator is called only about 35% of the time. The other 65% is handled by the transposition table moves.

The transposition table entries occupy 96 bits of memory. On a two million word Cray X-MP, about 1.5 million words are reserved for a one million entry table. Since the program regularly searches more than ten million positions in three to five minutes, the table is not nearly large enough. Much work still remains to be done concerning which positions are kept and which are overwritten when the table is too small.

The pawn hash table is used to store pawn structure information. Since a fair percentage of the positions in the tree search are reached by moving pieces, but not pawns, the program has to evaluate the same pawn structure repeadedly. The pawn structure is reduced to a 32-bit key and is stored along with the pawn structure evaluation. Anytime this pawn structure is found, the resulting score is immediately available, saving the evaluation time. The pawn structure evaluation routine is complex enough to take ten to twenty percent of the total search time without this table. The table searches usually find the correct score 98 to 99 percent of the time, reducing the pawn scoring to a negligible fraction of the total search time.

The king safety hash table is used to store the king safety information, since a great deal of analysis is done by the evaluation procedures to determine how safe/exposed the kings are. Again, most of the positions examined in the search share the same king safety position since the king and associated pawns rarely move. In order to help the program understand king safety, attack and defense, these routines are quite time consuming, and are comparable to the pawn evaluations in terms of percentage of the total search. Again, the king safety hash table provides a score 90 to 95 percent of the time, which reduces king safety to a negligible fraction of the total search time.

These tables are very important to program design, as they make it possible to do any type of evaluation deemed necessary without being overly concerned about efficiency. For a given position, once the evaluation is done, the program will not have to keep recomputing the value over and over, but will retrieve it from the table each time after the evaluation is done and stored. In general, anything can be added to the evaluation of king safety and pawn structure without causing any additional search time to be used.

## Multiprocessing

The Cray X-MP system is currently available as a 2-processor computer system configured as a tightly coupled multiprocessor with two to eight million words of memory. The first 4-processor configuration will soon be available with four to sixteen million words of memory. Software on the machine allows a use to easily implement multi-tasking algorithms if a given task can be divided into two or more independent subtasks.

The version of Cray Blitz that won the 4th World Computer Chess Championship in New York was converted to multi-task-over a period of approximately two months. This time included development, coding, debugging, testing and performance evaluation, with the actual time from the first turn on the Cray X-MP until the

tournament covering a period of one month.

The world championship algorithm was a hasty attempt to utilize multiprocessing. The tree searching code was duplicated to yield two identical sub-tasks for multiprocessing. When the search started, each processor would select a move from the shared ply-1 move list and examine it completely. When finished, the processor would take the next untried move on the ply-1 move list and examine it. This approach was taken as the most expedient way of utilizing the second processor, rather than because it was thought to be the best technique.

The main problem area in this technique is the shared ply-1 data that must be protected while either processor is using it. The Cray X-MP software has a set of FORTRAN callable procedures that implement this quite easily, allowing one processor to "lock" a resource while it is using it, which prevents access by the second processor until the first one has finished using it.

To determine which ply-1 moves have been examined, Cray Blitz uses an array of logical variables, TRIED(N). If TRIED(J) is .TRUE., then MOVE(J) has already been examined. While a processor is scanning the TRIED array looking for .FALSE., the array is locked so that the other processor must wait until the current one releases the list. This serves to implement a self-scheduling algorithm where each processor selects another move to examine as soon as it finishes with the current one. This eliminates problems associated with either processor being given a list of moves and one finishing long before the other. In this case, either one processor will remain idle, or else it will have to ask the other for additional moves.

After a processor finishes examining a ply-1 move, one of two cases must be handled. The first case is trivial, and occurs when the move results in an alpha cutoff. This move is worse than the best already found, and all relevant information about it is discarded. The second case occurs when the move is better than the best one found so far. The move must be saved as the best one found so far so that when the search is complete, it will be made; and the score must be shared with the other processor so that it will now obtain more alpha cutoffs in the search since the lower search bound has been upgraded to the new best score.

A final problem that surfaced immediately prior to the World Championship was timing related. It is possible that after finishing an iteration and starting the next, the first (best) move from the previous iteration will take a long time to evaluate. The second processor will examine the second, third, etc. moves on the list and get better and better backed-up scores. If time runs out, making the best move found so far could be a tragic mistake, since the BEST move from the previous iteration has not yet been examined to determine how good it is now. To avoid this, it is considered worthwhile to spend extra time to force this move to be examined, before deciding which move to make. Simply stated, the first move must be completely examined before any other move can be chosen, In certain rare occasions, it is possible for the first move to suddenly become bad, which results in the program making a poor move rather than a good one. Due to the program's timing algorithm (Hyatt, 1984), this is indeed a RARE occurrence and has not happened in testing or actual tournament play.

The most obvious problem with the above approach is that the first move each processor tries is examined with a large alpha-beta window. In most chess positions, the first move takes a major portion of the search time. Once the first move is finished, the rest are dismissed in rapid fashion unless a better move is found farther down the list. Two distinct possibilities exist based on the correctness of the ply-1 move list ordering: 1) the first move is best and will stay best through the current iteration: and 2) the first move is not best and will be replaced by one or more moves as the current iteration progresses.

Testing several sample positions has clearly shown that the current Cray Blitz search algorithm works well for case two. Each time the program searches a new best move, that move takes far longer to examine than moves that yield quick cutoffs. In this case, the second processor will start searching one of the better moves before the first processor completes analysis of a move that was sorted higher in the move list. Testing with favorable positions has yielded results showing that two processors are almost exactly twice as fast as one! The two examples in table 1 give search time on

a dedicated Cray X-MP with one processor, and then with two processors. The times are wall clock times and clearly show that for case two, the program is very fast. However, as the first example shows, if the program has one clearly best move that does not change, then the multiprocessing code reduces the search time by only 35%.

In table 1, the depth, represented as a number followed by a letter is the exhaustive search depth or iteration depth. The letter identifies which processor examined the move. The time is the elapsed time (wall clock) on a dedicated Cray X-MP computer in minutes and seconds. The evaluation is in units of one pawn. On the summary line, the time is the total time required to finish the search, followed by the percentage of cpu utilization. In multiprocessing examples, the number can go over 100% since there are two processors available. "Nodes" represents the total number of nodes examined for the given search. The numbers following the "H" represent hashing statistics. The first number identifies what percentage of the searched positions were found in the transposition table, the second percentage refers to the pawn structure hash table, and the third percentage refers to the king safety hash table. It should be noted that the percentages drop slightly in multiprocessing mode since two processors fill up the tables faster. The final number, NPS, is the number of nodes per second analyzed.

Since the current algorithm is not always twice as fast, and additional processors will greatly magnify the problem, a new algorithm is currently under development. All processors will work on the same ply-1 move and will move down into the tree and divide up the

**Table 1 Multiprocessing example times**

**case 1 (no multiprocessing)**

| Depth | Time | Eval | Variation |
|---|---|---|---|
| 5a | 0:01 | 0.383 | ♗×f3 ♕×f3 0-0 0-0 c5 |
| 6a | 0:07 | 0.251 | ♗×f3 ♕×f3 0-0 ♘c3 c5 0-0 |
| 7a | 0:30 | 0.305 | ♗×f3 ♕×f3 0-0 ♘c3 c5 0-0 h6 |
| 8a | 2:46 | 0.221 | ♗×f3 ♕×f3 e5 ♗e2 0-0 0-0 ♘e4 f×e5 ♘×e5 |

Time: 3.25 100% Nodes: 4981731 H 31% 99% 94% NPS: 24242

**case 1 (with multiprocessing)**

| Depth | Time | Eval | Variation |
|---|---|---|---|
| 5a | 0:01 | 0.383 | ♗×f3 ♕×f3 0-0 0-0 c5 |
| 6a | 0:04 | 0.251 | ♗×f3 ♕×f3 0-0 ♘c3 c5 0-0 |
| 7b | 0:19 | 0.305 | ♗×f3 ♕×f3 0-0 ♘c3 c5 0-0 h6 |
| 8a | 1:48 | 0.221 | ♗×f3 ♕×f3 e5 ♗e2 0-0 0-0 ♘e4 f×e5 ♘×e5 |

Time: 2:13 199% Nodes: 6545639 H 25% 96% 92% NPS: 49123

Net improvement with two processors: 1.5

**case 2 (no multiprocessing)**

| Depth | Time | Eval | Variation |
|---|---|---|---|
| 6a | 0:01 | 0.504 | ♘d5 e4 f×e4 f×e4 ♘f6 ♖e1 |
| 6a | 0:01 | 1.160 | h6 ♘h3 ♖e6 ♘f4 ♖×d6 ♔e1 |
| 7a | 0:09 | 1.310 | h6 ♘h3 ♖e6 ♘f4 ♖×d6 ♔e1 ♖d8 |
| 8a | 0:44 | 1.120 | h6 ♘h3 ♖ac8 ♘f4 ♖c6 ♔e1 ♖×d6 ♖c1 |
| 8a | 1:43 | 1.237 | ♗c4 ♔e1 h6 ♘h3 .... |
| 8a | 3:07 | 1.280 | ♖ac8 a3 ♖c2 ♖c1 ♖a2 ♖c7 ♖a1+ ♗c1 ♖×e3 ♖×a7 |

Time: 3:20 100% Nodes: 3887306 H 40% 99% 96% NPS: 19606

**case 2 (with multiprocessing)**

| Depth | Time | Eval | Variation |
|---|---|---|---|
| 6a | 0:01 | 0.504 | ♘d5 e4 f×e4 f×e4 ♘f6 ♖e1 |
| 6b | 0:01 | 1.160 | h6 ♘h3 ♖e6 ♘f4 ♖×d6 ♔e1 |
| 7a | 0:04 | 1.310 | h6 ♘h3 ♖e6 ♘f4 ♖×d6 ♔e1 ♖d8 |
| 8a | 0.23 | 1.120 | h6 ♘h3 ♖ac8 ♘f4 ♖c6 ♔e1 ♖×d6 ♖c1 |
| 8b | 0:51 | 1.237 | ♗c4 ♔e1 h6 ♘h3 .... |
| 8a | 1:31 | 1.280 | ♖ac8 a3 ♖c2 ♖c1 ♖a2 ♖c7 ♖a1+ ♗c1 ♖×e3 ♖×a7 |

Time: 1.40 200% Nodes: 3987721 H 40% 99% 96% NPS: 39877

Net improvement with two processors: 2.0

work at some deeper level. This means that the first move will be evaluated much faster since all of the processors are concentrating on it. Then, after the value for the first move is known, each processor will take a different ply-1 move and search it independently. The final enhancement occurs when one processor is busy working on the last move, and the other(s) will assist the remaining processor by taking some of the moves at a deeper level in the tree and examining them.

Another special case occurs after the first move has been examined and the processors start working through the remaining ply-1 moves. If one of the processors examines a ply-1 move and gets a beta cutoff (indicating that this move is FAR better than any other found so far), all processors will stop searching and treat this move as the first one searched so far. This will get all of the processors involved in determining how good this move is, rather than having one examine it while the others search the remainder of the list and then sit idle while waiting on the last move to be finished.

It appears that the total search time required for a multiprocessing parallel search can be computed as follows:

$$\text{time} = \frac{\text{(total time required)}}{\text{(number of processors)}} + \text{(idle time)}$$

The idle time is a number representing the time during which one or more processors are waiting for something to do. As an example, if the ply-4 move list were divided among the processors, when all but the last move has been tried, there is nothing for the remaining processors to do but wait. This idle time can be reduced by moving deeper into the tree to divide up the work, but then interprocessor conflicts become more frequent which introduces a different type of idle time during which one or more processors are waiting for access to the move list to get another move. These times are inversely proportional to each other and have a lower limit which is controlled by the multiprocessing software overhead, the multiprocessing hardware overhead, and the multiprocessing application.

## Special scoring

One particular area of the evaluation procedure that has resulted in much stronger play by the program is the special pawn scoring for endgames. In the early phases of development, Blitz frequently found itself a pawn ahead in an endgame that it did not understand. It lost many games that it should have won, and, in fact, opponents started playing for endgames even if they were material behind because the program could not search deeply enough to understand the pawn situation. After one particularly frustrating draw where the program was a piece ahead but let a pawn advance too far before realizing that it could not be stopped, the current pawn endgame algorithm was developed.

The first segment developed concerned playing with king and pawn against a lone king. The rules are clearly defined and the program now plays this endgame perfectly with a one ply search. This helped, but did not solve the real problem of pawn races, unstoppable pawns, etc.

The next development covered the "square of the pawn" and allowed the program to understand king play when many pawns were on the board. It simply "counts" the number of moves to promotion by either side and then determines if the king can stop the pawn. This algorithm is used only when one side has a passed pawn and the other side has no pieces. The program plays these positions correctly and does not lose because it does not search deeply enough. If the tree search includes the position, then the program will know what to do to handle it properly. This algorithm also includes special cases for a pawn that will promote on a square protected by its own king. In this case, the "square of the pawn" is not important since the opponent's king cannot capture or blockade the pawn.

When this algorithm was added, the program immediately advanced one USCF class in playing strength. The program was strong tactically, but its performance was hindered by the poor play it exhibited in endgames. Since the development of this algorithm, players have found it much more difficult to trade down into an endgame and swindle the program because of its inferior endgame understanding.

An additional enhancement was added later to handle the case where both sides have pawns that are free to promote. If one king is "in the square" of the opponent's pawn, but the other is not, it is a trivial case of a win. If both kings are "out of the

square", then the pawn that promotes in one of the following ways wins: 1) the pawn promotes two moves before the other, so that it has time to stop the other pawn from promoting: 2) the pawn that promotes first and with check wins as this is then a case of rule 1 ; 3) the pawn that promotes first and attacks the promoting square of the other wins unless the promoting square is also defended by the opponent's king. In this case it is usually a draw.

With these algorithms, it is no longer necessary to worry about the program entering a king and pawn endgame. In fact, with no pieces on the board, the program is probably capable of playing 200 USCF rating points ABOVE its real USCF rating because of its deep searching augmented by the endgame evaluation.

## Conclusion

The performance of the program, has been briefly described in the history of the development of Cray Blitz. It is currently examining about 50,000 nodes per second and searching to a depth of about eight plies in the middle game. In endgames, it regularly goes sixteen to twenty plies, and in king and pawn endgames has searched as deeply as forty-five plies.

The following two positions with corresponding analysis by the program illustrate the performance of the current 2-processor version of Cray Blitz which still uses the World-Championship algorithm of dividing the ply-1 move list.

The first position (diagram 1) is a position from a game between USSR grandmaster David Bronstein and the EVM M-20 Chess computer. Cray Blitz finds the mate in ten moves in under three minutes:

| Depth | Time | Eval | Variation |
|---|---|---|---|
| 3a | 0:00 | 1.820 | ♘×g5+ ♕×g5 ♘×f4+ ♔e7 ♘d5+ ♔e8 ♘×c7+ ♔d8 ♗×g5+ f6 ♗×f6+ ♗e7 ♕×b5 |
| 4a | 0:01 | 5.623 | ♘×g5+ ♕×g5 ♘×f4+ ♔e7 ♘d5+ ♔e8 ♘×c7+ ♔d8 ♗×g5+ ♗e7 ♗×e7+ ♔×e7 ♕c5+ ♔d8 ♘×a8 |
| 5a | 0:04 | 5.623 | ♘×g5+ ♕×g5 ♘×f4+ ♔e7 ♘d5+ ♔e8 ♘×c7+ ♔d8 ♗×g5+ ♗e7 ♗×e7+ ♔×e7 ♕c5+ ♔d8 ♘×a8 |
| 6b | 0:25 | 5.788 | ♘×g5+ ♕×g5 ♘×f4+ ♔e7 ♘d5+ ♔e6 ♘×c7+ ♔e7 ♘d5+ ♔e8 ♕×c8+ ♕d8 ♘c7+ ♔e7 ♗b4+ d6 ♗×d6+ ♕×d6 e×d6+ ♔×d6 ♘×a8 |
| 7b | 2:27 | 999.980 | ♘×g5+ ♕×g5 ♘×f4+ ♔e7 ♘d5+ ♔e6 ♘×c7+ ♔e7 ♘d5+ ♔e8 ♕×c8+ ♕d8 ♘c7+ ♔e7 ♗b4+ d6 ♗×d6+ ♕×d6 ♕e8++ (mate in 10) |

Time: 2:27 200% Nodes: 5653322 H 17% 95% 92% NPS: 38458

The second position (diagram 2) is from

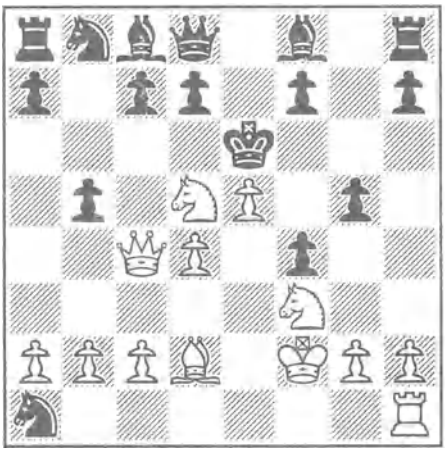


**Diagram 1**

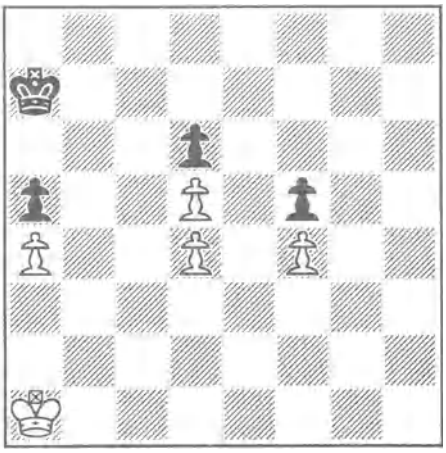


**Diagram 2**

**Basic Chess Endings** by Reuben Fine, and is problem No. 70. The search tree is quite narrow due to the locked pawns, and Cray Blitz displays its speed by quickly finding the correct (but not obvious) winning plan:

| Depth | Time | Eval | Variation |
|---|---|---|---|
| 18a | 0:00 | »»» | ♔b1!! |
| 18a | 0:00 | 1.868 | ♔b1 ♔a8 ♔b2 ♔a7 ♔b3 ♔c3 ♔c7 ♔d3 ..... |
| 19a | 0:00 | 1.868 | ♔b1 ♔a8 ♔b2 ♔a7 ♔b3 ♔b7 ♔c3 ♔c7 ♔d3 ..... |
| 20a | 0:00 | 1.868 | ♔b1 ♔a8 ♔b2 ♔a7 ♔b3 ♔b7 ♔c3 ♔c7 ♔d3 ♔b6 ♔e3 ♔c7 ♔f3 ♔d7 ♔g3 ♔d8 ♔h4 ♔e8 ♔g5 ♔e7 ♔×f5 ..... |
| 21a | 0:00 | 1.862 | ♔b1 ♔a8 ♔b2 ♔a7 ♔b3 ♔b7 ♔c3 ♔c7 ♔d3 ♔b6 ♔e3 ♔c7 ♔f3 ♔d7 ♔g3 ♔d8 ♔h4 ♔e8 ♔g5 ♔e7 ♔×f5 ..... |
| 22a | 0:01 | 1.862 | ♔b1 ♔a8 ♔b2 ♔a7 ♔b3 ♔b7 ♔c3 ♔c7 ♔d3 ♔b6 ♔e3 ♔c7 ♔f3 ♔d7 ♔g3 ♔d8 ♔h4 ♔e8 ♔g5 ♔f7 ♔×f5 ..... |
| 23a | 0:01 | 1.862 | ♔b1 ♔a8 ♔b2 ♔a7 ♔b3 ♔b7 ♔c3 ♔c7 ♔d3 ♔b6 ♔e3 ♔c7 ♔f3 ♔d7 ♔g3 ♔d8 ♔h4 ♔e8 ♔g5 ♔f7 ♔×f5 ♔e7 ♔g5 |
| 24a | 0:01 | 1.862 | ♔b1 ♔a8 ♔b2 ♔a7 ♔b3 ♔b7 ♔c3 ♔c7 ♔d3 ♔b6 ♔e3 ♔c7 ♔f3 ♔d7 ♔g3 ♔d8 ♔h4 ♔e8 ♔g5 ♔f7 ♔×f5 ♔e7 ♔g6 ♔d7 |
| 25a | 0:02 | 1.898 | ♔b1 ♔a8 ♔b2 ♔a7 ♔b3 ♔b7 ♔c3 ♔c7 ♔d3 ♔b6 ♔e3 ♔c7 ♔f3 ♔d7 ♔g3 ♔d8 ♔h4 ♔e8 ♔g5 ♔e7 ♔×f5 ♔f7 ♔g5 ♔g7 f5 ..... |
| 26a | 0:03 | 1.898 | ♔b1 ♔a8 ♔b2 ♔a7 ♔b3 ♔b7 ♔c3 ♔c7 ♔d3 ♔b6 ♔e3 ♔c7 ♔f3 ♔d7 ♔g3 ♔d8 ♔h4 ♔e8 ♔g5 ♔e7 ♔g6 ♔e8 ♔f6 ♔d8 ♔e6 ..... |
| 27a | 0:05 | 1.928 | ♔b1 ♔a8 ♔b2 ♔a7 ♔b3 ♔b7 ♔c3 ♔c7 ♔d3 ♔b6 ♔e3 ♔c7 ♔f3 ♔d7 ♔g3 ♔d8 ♔h4 ♔e8 ♔g5 ♔e7 ♔×f5 ♔f7 ♔g5 ♔g7 f5 ♔f7 ..... |
| 28a | 0:06 | 1.964 | ♔b1 ♔a8 ♔b2 ♔a7 ♔b3 ♔b7 ♔c3 ♔c7 ♔d3 ♔b6 ♔e3 ♔c7 ♔f3 ♔d7 ♔g3 ♔d8 ♔h4 ♔e8 ♔g5 ♔e7 ♔×f5 ♔f7 ♔g5 ♔g7 f5 ♔g8 ♔f6 ♔f8 |
| 29a | 0:10 | 2.008 | ♔b1 ♔a8 ♔b2 ♔a7 ♔b3 ♔b7 ♔c3 ♔c7 ♔d3 ♔b6 ♔e3 ♔c7 ♔f3 ♔d7 ♔g3 ♔d8 ♔h4 ♔e8 ♔g5 ♔e7 ♔×f5 ♔f7 ♔g5 ♔g7 f5 ♔g8 ♔f6 ♔f8 f6 |
| 30a | 0:13 | 1.994 | ♔b1 ♔b7 ♔c1 ♔c7 ♔d1 ♔b7 ♔e2 ♔c7 ..... |
| 31a | 0:15 | 1.998 | ♔b1 ♔b7 ♔c1 ♔c7 ♔d1 ♔b7 ♔d1 ♔b7 ♔e2 ♔c7 ♔d3 ♔b7 ♔c3 ♔c7 ♔b3 ♔b7 ..... |
| 32a | 0:44 | 1.994 | ♔b1 ♔b7 ♔c1 ♔c7 ♔d1 ♔b7 ♔e2 ♔c7 ♔d3 ♔b7 ♔e3 ..... |
| 33a | 1:02 | 1.994 | ♔b1 ♔b7 ♔c1 ♔c7 ♔d1 ♔b7 ♔e2 ♔c7 ♔d3 ♔b7 ♔e3 ♔c8 ♔f3 ♔d8 ♔g3 ♔e7 ♔h4 ♔f6 ♔h5 ♔f7 ♔g5 ..... |

Time: 1:05 200% Nodes: 4061005 H 72% 99% 90% NPS: 62477

Since the current program is searching to a depth of eight plies, and the new computing hardware will double the computing power, Cray Blitz is rapidly approaching a search depth of nine plies during the middle game. It is almost certain that by the World Computer Championship in 1986, the program will be performing nine ply searches due to faster computing hardware currently under development. How much that will affect the program's rating is unknown, but it will obviously become stronger as it searches faster and deeper.

In conclusion, while no computer grandmaster currently exists, several programs are closing in on this prestigious title. The master barrier has been broken by at least two programs, and senior master, international masters, and grandmasters are around the corner.

# 3 Position Evaluation

Every chess program contains an evaluation function (sometimes called a scoring function) to assess the merit of the positions that it encounters during its search for a move. Many of the features employed in these evaluation functions are described in the previous chapters, or in other parts of this volume. In the present chapter there are two brief contributions of a more detailed nature.

Slater's paper (section 3.1) was the first to attempt to quantify the significance of mobility in chess. He studied the relative mobilities of the winners and losers of a number of master games, and his data provides clear evidence as to the importance of employing mobility as a feature in position evaluation. I have included the transcript of a discussion which took place on Slater's paper, at the symposium where the paper was given, and the reader will see that both Jack Good and Alan Turing were present.

Good's work is rather more radical. He provides logical arguments to show why particular features might be quantified in a certain manner. He also introduces the notion of turbulence, which has never really been employed in chess programming even though it is a significant factor in human chess thinking. A human chess master might well say to himself: "In a position as sharp as this it would be dangerous to leave my king in the centre," whereas a computer program does not (typically) make its weighting of certain features dependent on the presence or absence of others. Perhaps non-linear evaluation functions will become popular at some future date, in which case some of Good's ideas will come into their own.

# 3.1 Statistics for the Chess Computer and the Factor of Mobility

**Eliot Slater**

First published in *Proceedings of the Symposium on Information Theory,* London 1950, Ministry of Supply, pp. 150-152. The discussion on this paper was first published in pp. 198-200 of the same volume.

Shannon (1950a) has argued that the problem of providing a programme for a chess-playing computer is of theoretical interest, and its use might lead to a wide range of practical developments. The problem is also interesting psychologically. If the human and the mechanical players are to play the same game, they will each have to be directed by concepts which have a certain equivalence. But the concepts used by the skilled human chess-player are both subtle and complex, and for the purpose of programming a computer they will have to be reduced to their simplest form. Chess-masters are, as a class, men of considerable general intellectual ability, and come from the ranks of professional men, mathematicians, scientists, lawyers, etc. They have in addition a special ability. Very few chess-masters, who began the game early, did not show unusual excellence at it at a very early age. The specific chess ability begins to show itself, given the opportunity, at about the age of eleven. Furthermore, there are few, if any, chess-masters who cannot play blindfold, and play many games at once, achievements which are entirely beyond the powers of the ordinary player. The order of intellectual activity which we are required to reduce to simple terms is therefore of a superior kind.

Shannon has suggested that one practical strategy for the chess computer would be for it to combine a deeper analysis of forcing variations with a two- or three-move analysis of development in more quiescent positions. Both are necessary if the machine is neither to fall into simple tactical traps, nor to fail to develop its pieces in an adequate way. The present investigation is confined to the latter aspect of play. Statistical data have been taken from the game played in the following tournaments: St. Petersburg 1914, New York 1916, London 1922, Hastings 1922, Kecskemet 1927, Capablanca-Alekhin match 1927, Alekhin-Euwe 1933.

It would simplify the problem of programming the computer if it could be provided with only a few parameters for the measurement of a given chess position. The total value of a position would then be the weighted sum of the value of these parameters, i.e. the directive for the machine would be a discriminant function which could be calculated by the statistical analysis of chess positions of known value. Shannon has given a list of fifteen features which should be evaluated by the machine. It is submitted that this list could be simplified. The three noteworthy parameters appear to be those of material, structure and function. Structure is inherent in the pawn formation, which changes only slowly during the course of a game. It is far from easy to provide any single scale of measurement for its evaluation. Material, on the other hand, is easily measurable. For practical purposes we can adopt Shannon's scale of 9 for ♕, 5 for ♖, 3 for ♗ or ♘, 1 for ♙. If the value of a game is +1 in the case of a win for White, 0 in the case of a draw, −1 in the case of a win for Black, then the value of a chess position of a defined kind, Shannon's $f(P)$, is $(w - b)/(w + d + b)$, where w, d and b are respectively the number of White wins, draws and Black wins which have been obtained from that kind of position. From the games analysed an advantage of one pawn has at the 20th move if positive, i.e. in favour of White, a value of +0.27, if negative, i.e. in favour of Black, a value of −0.21.

Material gains are therefore, as we could have guessed, highly decisive. They are, however, attained as a rule only after a considerable number of moves and with difficulty. Out of 350 master games, only in 98

cases had a material advantage been won by the 20th move. This factor by itself will be of value to the machine towards the end of the game in administering the coup-de-grace, and in ordinary situations in preventing it from falling into crass error. For move-to-move guidance, advantages measurable in smaller units will be evaluated. This, the strategic advantage, is involved in the functional factor.

This factor appears to be largely identical with mobility. Its value is recognised by chess experts, but it is often inaccurately judged. On the simplest view it is fundamental. The outstanding characteristic of a chess position is that it is a dynamic situation with a definite and limited number of degrees of freedom (the total number of legal moves). The game cannot be finally won without administering mate, when the opponent's degrees of freedom are reduced to zero. A series of checks, known by chess-players to be highly dangerous, is so because it reduces the opponent's mobility to the lowest possible level short of paralysis, and often eventually compels a disastrous move.

If the degrees of freedom available to each opponent are charted through the course of a game, a double curve is obtained whose rises and falls correspond closely with the strategic advantage. Means taken from 78 arbitrarily selected games which ended with a decision on or before the 40th move shows the following state of the parties:

| After move | Winners | Losers |
|---|---|---|
| 0 | 20.0 | 20.0 |
| 5 | 34.2 | 33.9 |
| 10 | 37.5 | 36.0 |
| 15 | 39.7 | 35.2 |
| 20 | 38.9 | 36.4 |
| 25 | 39.6 | 31.9 |
| 30 | 35.6 | 27.7 |
| 35 | 31.7 | 23.2 |

These figures show (1) a rise in the degrees of freedom as pieces are developed, followed by (2) a fall as pieces are exchanged, and (3) a consistent and increasing difference to the advantage of the winner.

Shannon's $f(\mathrm{P})$ has as its limits +1 and −1. We can define an advantage in mobility, M, as $(\mathrm{dF}_W - \mathrm{dF}_B) / (\mathrm{dF}_W + \mathrm{dF}_B)$, which has the same limits. It has been found from a total of 380 games from the tournaments mentioned above, i.e. including all those which proceeded as far as the 20th move without one or other player having won an advantage in material, that there is a close relation between $f(\mathrm{P})$ and M. This is shown in the following table:

| Value of M at 20th move | $f(\mathrm{P})$ |
|---|---|
| +·26 | +·46 |
| +·19 | +·35 |
| +·14 | +·27 |
| +·09 | +·17 |
| +·04 | +·09 |
| −·01 | −·05 |
| −·07 | −·06 |
| −·13 | −·03 |
| −·24 | −·43 |

If graphed, these points approximate to a straight line, and to the equation $f(\mathrm{P}) = +{\cdot}086 + 1.658\mathrm{M}$.

We may use as the value of M to predict the value of $f(\mathrm{P})$, and are then interested to know its efficiency. If wins for White, wins for Black and draws are in equal proportions (as is very nearly the case), a random classification of a series of games will result in a mis-classification rate of 89 points per hundred. If games are ranked by the value of M at the 20th move, the misclassification rate drops to 63 points per hundred, and if both the values of a material advantage and of M are combined by simple addition, the rate drops still further to 55 points per hundred. Efficiency is therefore rather low. It is rather doubtful whether it could be much improved without the intrusion of a subjective element. The players themselves "misclassify" their games, as it is far from infrequent for a player to lose from a theoretically drawn position, even in a fairly advanced stage of the game, or to draw or lose from a theoretically won position. The frequency with which this is actually observed depends on the care and skill of the subsequent analysis to which tournament games are usually subjected when play is over. Much depends, therefore, on the annotator. With a highly skilled and interested analyst like Alekhin, one finds that, in his judgement, the "misclassification" produced by errors in play approaches in the tournament games he has annotated to 40 points per 100.

To summarise the effect of these

arguments, it does seem possible that a chess computer which was programmed, beyond immediate tactical tasks, to maximise the mobility difference between itself and its opponent over a series of moves, might play a strategically tolerable game of chess. The mobility factor by itself might be a sufficient measure of all the factors (Rook on open file, etc.) listed by Shannon as requiring evaluation under his headings (3) and (5). Moreover the concept of mobility overlaps with other concepts much used by chess players. Control of space, when defined as the number of squares in the opponent's half of the board whose use can be denied him for at least one move, proves to be correlated with mobility by a coefficient of +0.83, i.e. the two concepts are nearly identical. "Development" is largely tantamount to the acquisition of increased mobility for all pieces together and for each piece separately. "Initiative" is nearly always in the hands of the player with the advantage in mobility, and "having the attack" seems to consist in having the opportunity for its aggressive application. "Combination", though knowhere very precisely defined by chess authors, seems to be the same thing as the exchange of one sort of advantage for another, e.g. when superior mobility enables a rapid concentration of attack on a piece which cannot be so quickly defended, or when a sacrifice in material buys an open file bearing on the opposing king.

The speculation may be offered that many other games, from draughts to war, may be found by appropriate analysis to involve the same concept, i.e. that advantage lies in creating a difference in mobility. For the more restricted problem of programming a chess computer, however, these preliminary investigations also suggest that a digital computer may prove an inefficient instrument, and that an analogue machine, or a combination of digital and analogue machines, will provide better results.

## Discussion on Dr. E. Slater's paper on "Statistics for The Chess Computer and The Factor of Mobility"

### Dr. D. K. C. MacDonald

In so far as a mental analysis of the feasibility of a machine to "play" Chess may assist us to decide more clearly what true constructive thought is and whether brains *in principle* are different from machines, I wholeheartedly support such work. I feel however that the actual construction of such a machine in all its relatively vast complexity – apart from personal satisfaction which it may provide – is of dubious value. To me, the purpose of Shannon's paper in *Phil. Mag.* was the analytical demonstration that a machine could simulate a process normally regarded as essentially "brainy" – not that one should therefore make one. I am reminded that Professor Simon once remarked to me that one could presumably make a machine that would smoke tobacco – *cui bono?*

### Dr. F. L. Stumpers

I agree with Dr. MacDonald that the construction of a chess-machine in itself is of doubtful value. I discussed Dr. Shannon's paper with Dr. Euwe, former world-champion and mathematician and we thought that it was doubtful whether even this complicated machine would play a really good game of chess (better than an average club-player). It is very difficult to see what inferences can be drawn from Dr. Slater's interesting statistical analysis. To stress the mobility too much would bring us back to the classical school in chess (Tarrasch). In the 1920s, e.g. Reti, Euwe, Alekhine have shown that mobility and space also give a great responsibility. The Indian games rose in popularity and Alekhine played his defence in which a knight moves over the board without any increase in mobility for his side. However, when many other things are taken into account, mobility is also of importance.

### Mr. A. M. Turing

I wish to make two points concerning Dr. Slater's paper. I was greatly interested by the statistics provided, but fear that some people might draw invalid conclusions from them. It might for instance be thought that a good way of playing is to maximise one's mobility at one's next move, or perhaps to minimise that of one's opponent at his next move but one. It is evidently not feasible to foresee mobilities many moves ahead. Although the immediate mobility is a useful measure of the relative advantage

of the players in normal play it by no means follows that it is wise to direct one's play to maximising such a measure. To do so would be like taking a statistical analysis of the laundry of men in various positions and deciding, from the data collected, that an infallible method of getting ahead in life was to send a large number of shirts to the wash each week.

I wish also to put forward a plea for a more experimental approach to chess machines. A more realistic attitude to these machines can be reached by making them than by talking about them. There are three forms which a chess machine might take.

(a) Special machines built to play chess and for no other purpose.

(b) A digital computer programmed as a chess machine.

(c) Paper machines.

The special machines I do not recommend. They would be too much trouble and Dr. MacDonald's comparison with a 'smoking machine' is very apt in connection with them. To programme a computer would be more rewarding, but not everyone has a computer. I wish however to recommend the 'paper chess machines' to the symposium. By making such a machine I mean laying down a definite set of rules governing play and then obeying them, or in other words 'programming oneself as a chess machine'.

Anyone can do this: the only apparatus required is board and men, and perhaps pencil and paper. Both Dr. Shannon and I have many others. I recommend it as easy, instructive and entertaining. Similar experiments can be made with 'paper learning machines'.

**Dr. I. J. Good**

I agree with Dr. Turing that mobility by itself is not enough. It seems not to take sufficiently into account the essential difference between strategy and tactics. I am reminded of the alleged motto of the Hampstead Chess Club: 'Never miss a check, – it might be mate'. Mobility may be sufficient evaluation of a position for strategic purposes, but it does not allow sufficiently for tactics, i.e. combinational play or forced variations. A good chess machine would have to analyse all forceful variations, with a suitable definition of a forceful variation. This fact has almost certainly been recognised by Dr. Shannon and others. The definition of the forcefulness of a move would have to depend partly on the difference between the strengths of an attacked piece and the attacking piece.

The analysis of the forceful variations in any given position would be a stochastic branching process, and the end-points of the corresponding tree would be quiescent positions which would need strategic evaluation (e.g. by means of a measure of mobility). The total time taken to cope with such a tree would be roughly proportional to the number of individuals in this tree, and it is known that the probability distribution of this number is extremely skew. (See, for example, *Proc. Cam. Phil. Soc.*, 45 (1949) 360-3.) The time taken to analyse a position might be anything from 0.1 seconds to 100 years. There would therefore be a danger of the machine getting into time-trouble. It would be advisable in fact for the machine to take into account the state of its chess clock. It would then modify its definition of the threshold forcefulness of a move. In other words the programme would contain a parameter, which would say how forceful a move must be in order to be examined as a sub-variation i.e. it determines whether a position is quiescent; and it would increase when the machine was short of time. Alternatively it might be an increasing function of the number of moves 'deep' in any analysis (i.e. a function of the generation number in the tree mentioned before).

A really clever machine would also take into account the state of its opponent's clock. The machine might move faster when its opponent was in time trouble in order to give its opponent less time to think.

**Dr. E. Slater (in reply)**

Dr. Good's remarks seemed to me important and very much to the point. I am sorry if I gave the impression that tactical consideration were of little significance; they would have to be taken full account of in programming the computer, and some method would have to be devised for making the machine sensitive to threats so that it would analyse forcing variations to a depth of five or six moves or more. The control of the time factor is a most interesting suggestion. Dr. Turing's argument by

analogy from what a naive laundry worker might conclude about ways of becoming rich really amounts to the suggestion that strategic advantage is the cause rather than the product of an advantage in mobility. I do not think that this can be accepted. An advantage in mobility usually appears in a game a number of moves before strategic advantage is detectable in other ways; it seems to be an essential aspect of what chess-players understand by "development"; and it supplies the decisive criterion of winning or losing. The relegation of mobility to a place of almost trifling importance by the modern school of players, following the work of Reti, to which Dr, Stumpers has referred, has, of course, been an interesting evolutionary change in the style of the game; but one does not have to regard it as final. There are such things as fashions in chess as in other fields into which an element of intuitive appreciation enters.

The final, or rather the first question, which is raised by Dr. MacDonald, namely whether this whole discussion is not so much beating of the air, is to some extent one to which there is no answer. In so far as a present answer can be given, that has already been done by Shannon in the introductory paragraphs of his paper. The real answer would appear only after such a hypothetical machine had been built. To the general implication of Professor Simon's witticism one might reply with Faraday's classic retort: "What is the good of a new-born baby?"

# 3.2 A Five Year Plan for Automatic Chess (excerpts)

**Jack Good**

First published in *Machine Intelligence 2* (E.Dale and D.Michie, editors), Oliver and Boyd, Edinburgh, 1968, pp. 110-115. Reprinted by permission of Edinburgh University Press, and of the author.

**Appendix F.**
**The Value of the Pieces and Squares**

A satisfactory theory of the value of positions should include a theory for the values of the pieces as a special case. So it is worth while to look for a theory that gives reasonable values to the pieces. Then we could try to generalise the theory.

The values of the pieces ♘, ♗, ♖ and ♕, have been found by experience to be approximately proportional to 1, 3, 3, 5 and 9, and a king is worth about 4 in the endgame. These values vary with the position and with the number of pieces on the board. For example, two knights are worth less than rook when the only other pieces on the board are two kings, in so far as two knights and a king cannot force mate against a lone king. But the values given are applicable rather widely. For example, two knights are about the equal of six pawns on the second rank even when the kings are removed, as I have found by experiment.

A theoretical attempt to evaluate the pieces was made by H.M. Taylor in 1876, reported by Coxeter (1940, pp 162-165). The value of a piece is taken as proportional to the average number of squares controlled, averaged over all 64 positions of the piece on the board. This argument leads to the relative values of ♘, ♗, ♖ and ♕ : 3, 5, 8 and 13. Coxeter (or Taylor) goes on to modify the argument by asking instead for the probability of 'safely' giving check, that is, without being *en prise* to the king, if the piece and king are both placed on the board at random. This gives the ratios 12, 14, 22 and 40. These values are good, but this modification of the argument is artificial. For the piece values seem to be valid even for chess without kings, as I have found from some limited experiments, such as the one just mentioned. In fact they are probably more valid when there are no kings. Consider, for example, a piece, which I shall call the Galactic Emperor, that controls all the squares on the board. When the kings are on, it is worth at least seven queens, but, measured by squares dominated, it is worth only about three queens, on an open board. Therefore I believe that a theory that allows for the presence of the kings is too difficult to start with, and that it is more sensible to try first to obtain a theory that arrives at the known values without reference to checking.

In about 1936 (unpublished) I tried to make some allowance for the number of pieces on the board:

Let $p$ be the probability that a square on the board is occupied. I assumed that the value of a piece is proportional to the expected number of squares dominated, averaged for all positions of the piece, and obtained the following values for the knight, bishop and rook.

$$E_N(p) = 5{\cdot}25 \text{ (independent of } p),$$
$$E_B(p) = (140-196p+182p^2-112p^3+44p^4-10p^5+p^6)/16,$$
$$E_R(p) = \frac{4}{p}-\frac{1}{2p}\left\{\frac{1-(1-p)^8}{p}\right\}$$

The expected number of squares dominated by the queen is $E_Q(p) = E_R(p)+E_B(p)$. But we should allow also for the fact that rook and bishop have overlapping fields of influence. For example, a rook on d5 and a bishop on c4, on an open board, dominate only 24 squares instead of 27, and so lose about one ninth of their value. This gives a partial explanation of why a queen is worth more than a rook plus a bishop. A similar argument explains why two bishops are worth more than a bishop and knight, or than two knights. It is usually better to control two squares than to control one square twice, not allowing for the different values of squares.

Table 1 gives the numerical values of $E_R(p)$, etc., for a few values of $p$.

Table 1
Values of $E_N(p)$, etc.

| $p$ | 1 | $\frac{1}{2}$ | $\frac{1}{3}$ | $\frac{1}{4}$ | 0 |
|---|---|---|---|---|---|
| ♘ | 5.25 | 5.25 | 5.25 | 5.25 | 5.25 |
| ♗ | 3.07 | 4.7 | 5.72 | 6.35 | 8.75 |
| ♖ | 3.5 | 6 | 7.5 | 9 | 14 |

The case $p=1$ cannot occur in ordinary chess, but it is certainly reasonable that a knight would be worth more than a rook if all 64 squares were occupied. The case $p=\frac{1}{2}$ is not expected to apply for the first several moves of a game, when we know roughly where the pieces stand. The initial position can be examined in its own right, as indeed can any given position. But it would be too short-sighted to say that rooks, bishops and queens have no value in the initial position, merely because they cannot move. It would be more reasonable to make some guess concerning the probability that each piece will occupy the various squares after any given number of moves, and then to use these probabilities as weights in a weighted average of the number of squares dominated. Moreover we should discount the future in some way, for example, by giving further weights of $\mu$, $\mu^2$, $\mu^3$,..., to positions 1, 2, 3,...moves ahead, where $\mu$ is some positive number less than unity.

The case $p=\frac{1}{3}$, which is typical in the middle-game, gives values that are generally regarded as good average values of the pieces. It would be interesting to try to verify the remainder of Table 1 experimentally.

To make the theory more accurate we may attach a value to each square, making the central squares more valuable than the others. To do this we can use a similar principle to the one used for pieces, that is, we can find the average number of squares dominated from the square when various pieces are placed there. These can be taken as 1♕, 2♖s, 1♗ (only one colour!), 2♘s, but not 8♙s since not every pawn has a reasonable chance of getting to the given square. It seems reasonable to assume $1\frac{1}{2}$ pawns. I found the following values for the squares by this method. The squares are labelled as in Fig.1.

| | | | | |
|---|---|---|---|---|
| D | E | F | I | 7 |
| B | C | E | H | 6 |
| A | B | D | G | 5 |
| e | f | g | h | |

*Fig. 1. Labelling of squares. The labelling is centrosymmetrical, apart from the edges.*

A: $87-84p+44p^2-8p^3$.

B: $83-82p+55p^2-20p^3+3p^4$.

C: $83-92p+80p^2-40p^3+8p^4$.

D: $75-90p+95p^2-50p^3+18p^4-3p^5$.

E: $75-100p+110p^2-70p^3+23p^4-15p^5$.

F: $71-120p+160p^2-120p^3+48p^4-8p^5$.

G: $65-108p+130p^2-110p^3+63p^4-21p^5+3p^6$.

H: $65-118p+155p^2-130p^3+86p^4-21p^5+3p^6$.

I: $63-138p+205p^2-180p^3+93p^4-26p^5+3p^6$.

We can use these values of the squares to improve the estimates for the values of the pieces. We could then use these improved estimates to improve the estimates for the squares and so on. There will always be an error because of the assumption that the density of $p$ is the same over the whole board, and also because we can *choose*, for example, not to keep knights at the edge of the board, as a general rule. The chance that a knight is on the edge, after the opening, is much less than 28/64, if the players are any good!

I did this algebra before there were any electronic computers. It will now be easy to do the arithmetic.

If we wish to allow for the existence of kings we should give extra weight to the squares in their vicinities, or in some other way. We might, for example, compute a 'checking value' for each piece as the probability of giving check without being *en prise* to the king, and then take a linear combination of the checking value and the square-domination value.

The values of the pawns require a different theory since (i) they can be promoted, and (ii) they move differently from the way they capture.

It should be considered whether the values of the pieces can be related to the probability of winning. Should the value be taken as the improvement in the log-odds of winning as against losing?

Theories of evaluation of pieces could be tested by inventing new pieces; we are not inevitably tied to the small sample of distinct kinds of piece that occur in ordinary chess. An example of this test, was given above in connection with the Galactic Emperor, although perhaps this piece is too extreme for most purposes.

## Appendix H.
## Quiescence, Turbulence, and Agitation.

Positions for which there is no move forceful or purposeful enough to be worth analysing were independently described as 'quiescent' by Shannon (1950) and by Good (1950b). Turing (1953) called such positions 'dead'. One objection to this term is that a 'dead draw' is a standard expression among chess players. Also it is useful to be able to refer to 'degrees of quiescence', whereas 'degrees of deadness' sounds facetious, in spite of the expression 'dead as a door-nail' and of the notorious variability of deadlines. A good name for the degree of quiescence is simply the 'quiescence' of the position, or inversely one might refer to the 'turbulence'. A quiescent position is one of low turbulence, a combinative position is one of high turbulence. It is possible to define turbulence with respect to goals other than material balance, such as control of the centre (cf. Newell *et al.* 1958). The position illustrated in Fig.2 is of high turbulence with respect to material balance.

Before considering a more precise definition of 'turbulence' we discuss an allied notion called 'agitation'.

If in the analysis of a position $\pi_0$ we reach a position $\pi$, the turbulence of $\pi$ is not an adequate criterion for deciding whether to suspend further analysis. If $\pi$ were obviously won we would suspend its analysis even if it were turbulent. We need some measure, called say 'agitation', that measures turbulence and also takes into account whether the position is obviously won, drawn or lost. If we can measure the agitation of $\pi$, then we would set a threshold on this agitation multiplied by $P(\pi \mid \pi_0)$, the probability of reaching $\pi$ from the current position $\pi_0$. (Notice that, when we estimate this probability, we are in part, estimating the probability of our own future actions, not just those of our opponent. In my opinion, the same thing happens in the definition of a 'decision'. See Good 1964.) If this threshold is not reached, we would almost suspend analysis of $\pi$. I say 'almost' because there are a few points to attend to before finally classifying $\pi$ as an endpoint on the analysis tree of $\pi_0$. We should first (i) check whether $\pi$ is a position known definitely to be won, drawn, or lost; (ii) if $\pi$ is favourable to us, we should see if we have a move that prevents check; (iii) if $\pi$ is unfavourable to us, we should see if we have a check; (iv) if $\pi$ seems to be an obvious draw, we should carry out both the tests (ii) and (iii).

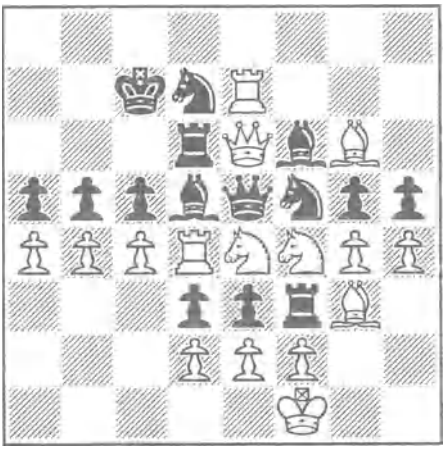


*Fig.2 A position of high turbulence!*

How then should agitation be measured? It will be recalled from Appendix C that the utility of position $\pi$, if we reach it, is of the form $P(W \mid \pi) - \lambda P(L \mid \pi)$, where $P(W \mid \pi)$ and $P(L \mid \pi)$ are, in some sense, the probabilities of a win or loss, given $\pi$. If we decide to treat $\pi$ as an endpoint, then these probabilities are 'superficial' and $P(\mathrm{W} \mid \pi) - \lambda P(\mathrm{L} \mid \pi)$ may reasonably be described as the 'superficial expectation of the utility' or, for short, as the 'superficial utility'. If we decide to suspend the analysis of $\pi$, that is, to regard it as an endpoint, it should be because we

believe that a deeper estimate of the utility of $\pi$, obtained say by spending a dollar on its analysis, is unlikely to make much difference. We must therefore have some method of guessing the expected value of $|U(\pi|\$)-U(\pi)|$, where $U(\pi)$ is the superficial utility of $\pi$, and $U(\pi|\$)$ is the expected utility of $\pi$ in the light of a dollar's worth of analysis, \$, or of course some other unit. The *agitation* $A(\pi)=A(\pi|\$)$, of $\pi$ is the guessed expected value of $|U(\pi|\$)-U(\pi)|$, and the decision whether to treat $\pi$ as an endpoint in the analysis of $\pi_0$ is to be made by setting a threshold on $A(\pi)P(\pi|\pi_0)$. Part of the five-year plan is to find an evaluation function that estimates $U(\pi)$, and another one that estimates $A(\pi|\$)$. We need evaluation functions even for turbulent positions.

A well-known issue in the theory of rationality is whether the distribution of a utility is relevant except in regard to its expectation. My own belief, shared by most 'rationalists', is that only the expectation is relevant, but this is on the assumption that all the relevant calculations have been completed. In complicated problems, like mathematics and chess, this assumption is unrealistic, and superficial probabilities and expectations have to be used (cf. Good 1962b). The guessed expected value of $|U(\pi|\$)-U(\pi)|$ is roughly proportional to the standard deviation of the superficial subjective distribution of the true utility of $\pi$. The true utility of $\pi$ is of course either 1 or $-\lambda$, or $\frac{1}{2}(1-\lambda)$, if both we and our opponent are perfectly rational.

I believe that this notion of the agitation of a position is applicable to almost every decision in practical life, since it is very rare that we can complete our reasoning in a practical problem. Instead of using the axiom of probability that $P(E|H)=P(F|H)$ when $E$ and $F$ are logically equivalent, we are forced to use the weaker one that these probabilities are equal when we have *proved* that $E$ is equivalent to $F$ (Good 1950a, p.49). Both humans and machines must often use imperfect logic.

It seems reasonable to define the turbulence, $\tau(\pi|G)$, of $\pi$ with respect to some goal $G$, as the superficial expectation, or guessed value, of $|L(G|\pi, \$)-L(G|\pi)|$, which equals $|W(G{:}\$|\pi)|$, where $L$ denotes log-odds, $W$ denotes 'weight of evidence', and the colon denotes 'provided by'. But it seems that decisions should depend more on agitation than on turbulence. The utilities should allow for all goals that we are prepared to consider. Agitation with respect to a goal can also be defined by interpreting $U$ appropriately.

If these ideas could be properly worked out, they would be applicable to the question of finding the value of storing previously analysed positions, as in Samuel's draughts program.

At the end of Section 1, we mentioned the problem of deciding what moves to discard 'without analysis'. I think the right approach to this question is to try the move, and estimate $|U(\pi')-U(\pi)|P(\pi|\pi_0)$ where $\pi'$ is the position reached after that move is played, and the utility $U(\pi')$ is estimated on the (false) assumption that the player retains the move. If this estimate is small enough, then the move can be treated as of no importance. In fact, if this estimate is small, then $P(\pi'|\pi)$ would be small, since a free move is usually quite advantageous. An exception is when the player is *Zugzwang*, and also in some end-game positions when no move is any good.

# 4 Tree Searching Techniques

Almost every chess program ever written uses some sort of look-ahead. Perhaps the most notable exception is Helmut Richter's program SCHACH (1976). Since the game tree for chess becomes enormous within only a few ply (half-moves) of look-ahead, a considerable amount of research effort has been devoted to the problems of tree searching. The contribution by Birmingham and Kent (section 4.1) provides a concise summary of the fundamentals of tree searching. The paper by the authors of KAISSA (section 4.2) is an account of their interesting "Method of Analogies", which aims to use information gained from one part of the search tree in analogous positions. In section 4.3, Larry Harris describes how the intelligence of heuristic search can be applied to chess programs, and how the method may be used to find sacrificial combinations or to defend against them.

Donald Michie's paper (section 4.4) breaks away from the classical traditions of game theory, and utilizes the fact that chess players are imperfect to show that it is not necessarily best to assume that one's opponent will always find the correct move. In my opinion Michie's contribution has been very much underestimated, and could be extended to enable a computer program to play "psychologically" – making moves which were thought to be stylistically unpleasant for a particular opponent.

# 4.1 Tree-Searching and Tree-Pruning Techniques

**John Birmingham and Peter Kent**

First published in *Advances in Computer Chess 1,* (M. R. B. Clarke, editor) Edinburgh University Press, Edinburgh 1977, pp. 89-97. Reprinted by permission of Edinburgh University Press.

This paper first discusses the now standard tree searching techniques of mini-max and alpha-beta and the recent improvements to alpha-beta, in particular the killer heuristic. It then describes a new and very powerful technique that we have termed 'razoring'. Unlike the alpha-beta technique, razoring cannot guarantee finding the optimum solution but it will, however, always find a good solution in a much shorter time. Typically a four-ply tree with a branching factor of 33 can be searched *an order of magnitude faster* than with alpha-beta. We also discuss 'forward marginal pruning' – a technique similar to razoring.

We then discuss 'chopper', a simple but usually overlooked device that avoids wasting time making a choice when no choice exists. Finally we consider a 'feed-over' technique to improve the effectiveness of alpha-beta and razoring.

### Mini-Max Algorithm

This technique (Shannon 1950, Turing 1953) is best described in terms of a game between Max and Min, where Max attempts to maximize the ultimate value of the game while Min attempts to minimize it. We will illustrate the technique by means of a simple game tree (figure 1), which has a branching factor of 3 and a depth of 2 ply. The square nodes indicate that it is Max's turn to move while the circles indicate that it is Min's turn. The nodes represent positions reached after a sequence of moves by Max and Min.

In the mini-max algorithm scores are only computed for the deepest level of the game tree, and are then 'backed up' to higher levels. No scores are computed for nodes at intermediate levels of the tree.

Min chooses the lowest score from her three choices at B, i.e. 4. She repeats this procedure for nodes C and D, obtaining values –4 and –8. Max then selects from the backed up values of B,C,D the node with the highest value; in this case B with a value of 4. The mini-max value of the tree is therefore 4 and Max's decision is to move to node B.

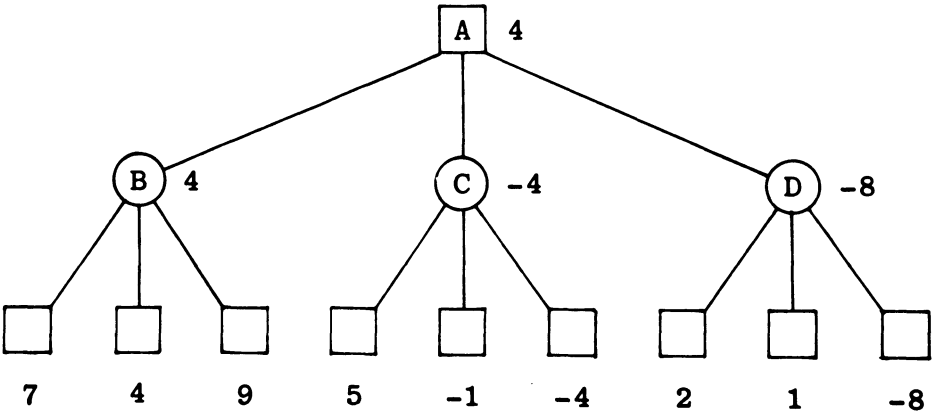



*Figure 1*

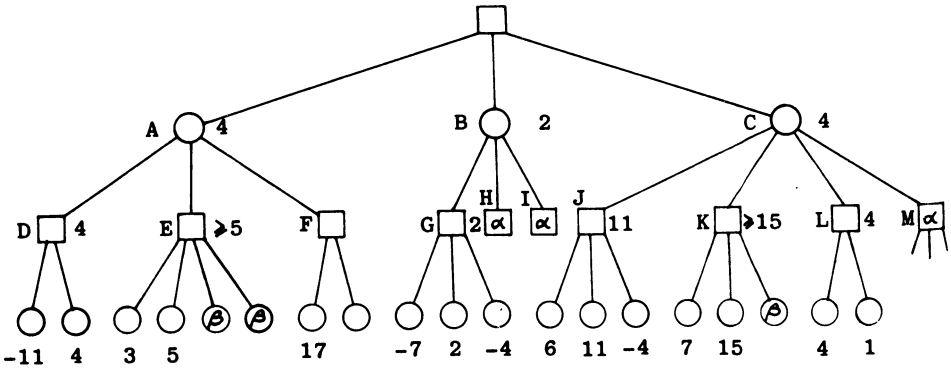



*Figure 2*

**Alpha-Beta Tree Pruning Algorithm**

The alpha-beta algorithm (Newell, Shaw and Simon 1959; Fuller, Gaschnig and Gillogly 1973) obtains exactly the same result as the mini-max algorithm, but is faster because it does not examine all the nodes of the tree.

We can again consider a game between Max and Min (figure 2). At node D it is Max's choice, so the value 4 is backed up to node D. The second node that Max finds from node E has value 5, the backed up value at node E is therefore $\geq 5$. As Min has a choice at node A of going to node D or node E, she will not select node E. This is an example of $\beta$-pruning.

$\alpha$-pruning is similar to $\beta$-pruning with the rules of minimizing and maximizing reversed. Thus Max has the initial choice of nodes A,B,C. He can obtain a score of 4 by going to node A. Min can limit Max to 2 or less at node B by selecting node G. Nodes H and I are therefore not worth examining and are cut by $\alpha$-pruning. Similarly Min can limit the value at node C to 4 or less by choosing node L. The value of the tree is therefore 4 and Max's decision is to move to node A.

The mini-max algorithm would have obtained the same result as alpha-beta but would have required the evaluation of 28 nodes instead of 15.

*Deep Alpha-Beta Pruning*

An alpha or beta cut-off value may be used to prune trees at any node which is an even number of levels below it (Fuller, Gaschnig and Gillogly 1973). This can be illustrated by the four-ply tree shown in figure 3.

In this example the value of $\alpha$ is obtained from the value backed up to node B, i.e. 7. As the first successor to node E has value 4 the remaining nodes are cut by deep $\alpha$- pruning. A new value of 9 is found for $\alpha$ at node F. As the first successor to node G has the value 8 the remaining nodes are cut by shallow $\alpha$-pruning. A similar technique may be used for deep $\beta$-pruning.

*Killer Heuristic or Refutation*

To analyze a four-ply tree with a branching factor of 33 the minimax algorithm will evaluate about one million bottom nodes. If the bottom nodes have a wide range of values and are randomly ordered the alpha-beta algorithm will reduce this number to about 25000.

A considerable improvement can be obtained by ordering the nodes at each level before applying the alpha-beta algorithm. Typically, mean branching factors can be reduced from 33 to 7 or 8 using the ordered alpha-beta technique.

A number of techniques have been used to perform this ordering, from fairly simple static ordering to elaborate dynamic systems. The simplest way is to make use of the same static evaluation function as used at the deepest level of the tree. One simply evaluates each node at a particular level and then sorts them into the appropriate order. However this technique cannot be used at the deepest level of the tree, as one has to evaluate all nodes before re-ordering, by which time it is too late to be of any use.

An alternative technique for ordering the moves is the killer heuristic (Gillogly 1972,

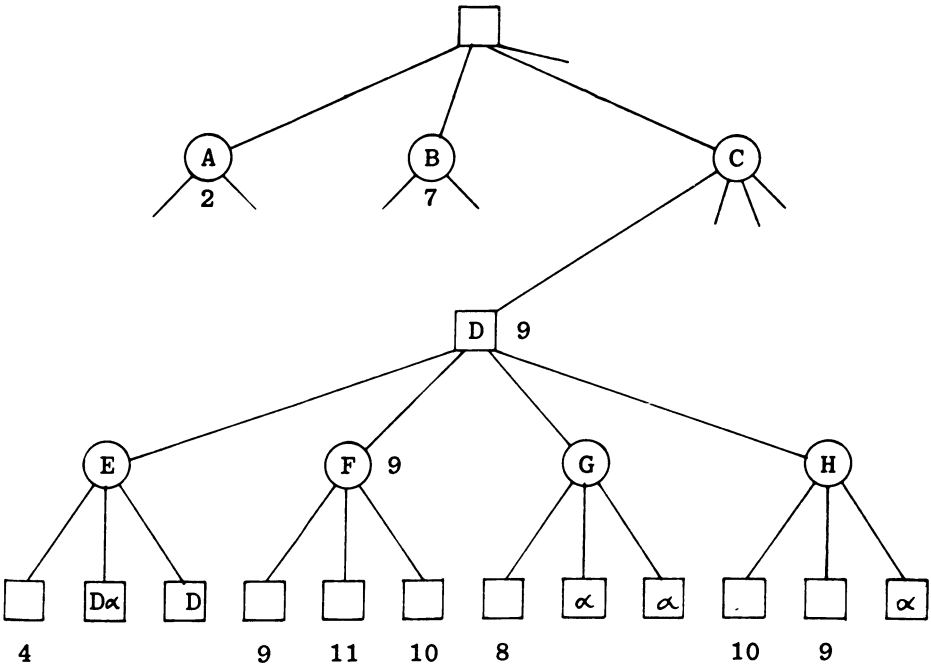



*Figure 3*

Cichelli 1973, Bell 1973, Adelson-Velskiy *et al.* 1963). It can be used at all levels of the tree but is particularly useful at the deepest level where other methods cannot be used. It works on the principle that a move that refutes *one* of Max's moves is likely to refute the rest of Max's moves.

The way in which this is implemented will depend on the particular application. Care has to be taken that the overheads involved in the re-ordering do not outweigh the reduction in node evaluations. In our chess program the score for the best move of each piece is kept and used to select the order in which each piece is examined. This complete ordering of the pieces is performed each time that an alpha cut-off does not occur at the deepest level of the tree. A further ordering takes place whenever an alpha cut-off does occur. As this is much more frequent we only move the 'killer' piece to the head of the list of pieces and just shift the intervening pieces down one place.

This produced approximately a 50 per cent reduction in node evaluations and a 40 per cent reduction in time.

**Razoring**

*Occam's Razor*: One should not make an assumption unless there is a good reason to do so.

*Inverse*: If there is a very good reason to make an assumption then make it (even if it is not *always* true).

This technique works on the assumption that from any given position my opponent will be able to find at least one move that improves his position. This assumption is invalid in the context of chess if a *zugzwang* situation occurs. In fact the program defines *zugzwang* precisely as a state in which every move available to one player creates a position having a lower value to him (in its own evaluation terms) than the present board position.

Razoring is a new and very powerful technique for speeding up mini-max searching. Unlike alpha-beta it will not guarantee finding the best decision but it will, however, find a very good move far faster than alpha-beta. In an $n$-ply tree alpha-beta will prune plies 2 to $n$, while razoring will prune plies 1 to $n-1$. Razoring has frequently been observed to cut a list of 40—50 choices

at ply 1 to a single node, while it once found the best choice in 28th position. Thus it is an effective way of automatically limiting the width of search at ply 1, and a considerable improvement on the rather arbitrary technique of picking the first *m* moves, a technique used in most current chess programs.

The program lists Max's three moves B, C, D (figure 4), makes each one in turn and computes a static evaluation for the resulting positions. The moves are ordered according to these values of 10, 8 and 2, and are then examined in order of decreasing value. The three replies to move B are valued 6, 12, 20. Min selects the lowest of these, so the backed up value at node B is 6.

Move C has static value 8 which is greater than 6, the value of Max at node A, and we therefore examine it further. The first reply has a value 4, and all further replies to move C are therefore cut by $\alpha$-pruning as the backed up value at node C must be $\leq 4$, which is $<6$ (the value of $\alpha$ at node B).

Move D has a static value of 2. This is less than the present value of Max and will be even worse after Min has found her best reply (on the above assumption). We do not, therefore, examine this further; it has been razored out.

*Deep Razoring*

The razoring technique can be extended to cut deeper trees by comparing values backed up from, say, ply 4 of a four-ply program, with the static values at ply 1. This is risky, however, as it assumes that an improvement cannot be found (for me) by the sequence of yourmove, mymove, yourmove. This is a weaker assumption than the normal one; that improvement (for me) cannot result from your move.

In order to keep the tempo of the evaluation correct, plies 1 and 3 should be cut using the value backed up from the static evaluation at ply 4, while ply 2 should be cut using the value backed up from the static evaluation at ply 3.

*Enhanced Razoring*

In normal razoring, a node at plies 1 or 3 is cut if its static value is less than or equal to Max, where Max is the value backed up from ply 4 to ply 1. A node is cut at ply 2 if its static value is less than or equal to Min (where Min is the value backed up from ply 3 to ply 1).

The degree of pruning can be modified by comparing the static node evaluations with Max or Min plus some value PRUNE. With a value of PRUNE=30 the chess program plays a sound but unimaginative game taking level swaps at every opportunity. All the more subtle moves are cut out at ply 1. It averages about 500 node evaluations per move, i.e. about two seconds per move. In order to keep the subtle moves it has been found necessary to have different levels of enhancement at each ply. The present values are PRUNE 1=0, PRUNE 2=20, PRUNE 3=40. This produces a similar standard of moves to the unenhanced razoring in about one-third of the time.

To put the prune values in context, a pawn is worth about 40 points.

**Marginal Forward Pruning**

This technique (Slagle 1971) is similar to

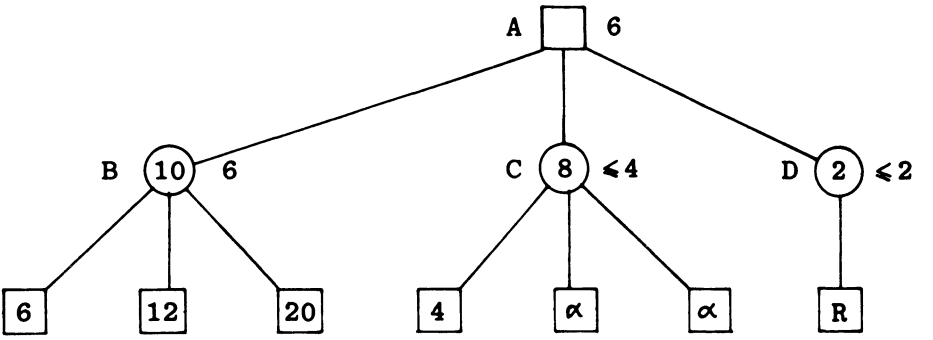



*Figure 4*

deep razoring with enhancement, and prunes by comparing the static value at any ply with the $\alpha$-$\beta$ value. One problem is that the tempo of the evaluation at the final ply is always wrong for one side or the other, but the effect of this can be reduced by putting an element of tempo in the evaluation function. For example, if a capture is made at the final ply of the search which leaves a piece *en prise*, then the value of the capture should be greatly reduced. For one side the basic assumption is the same as for deep razoring, i.e. I cannot improve my score by the sequence yourmove, mymove, your move. For the other side, however, the assumption is that you cannot improve your score by the sequence mymove, yourmove. This is an even weaker assumption than the one for deep razoring. It can be overcome by allowing for a margin of error, i.e. we examine all moves having a static value greater than $\alpha$-ERROR.

The second problem with this technique is found also in deep razoring. Most of the time the static value and the backed up value for any position are reasonably similar and the pruning behaves in a fairly stable manner. On occasion however the static and backed up values differ by considerably more than the ERROR margin. This causes one side to be pruned very lightly and the other side extremely heavily, with disastrous consequences.

### Chopper

When playing a game a chess playing program has to search a sequence of trees, and occasionally one of the trees in a given sequence will have a single node at ply 1. It is well worth while to test for such an occurrence, otherwise the program will search through plies 2, 3, 4, etc., only to decide that its only choice is also its best choice! So long as the value of the tree is not required, but only the decision about which move to select, one can chop the tree off at ply 1 and the only move available.

This situation often occurs in chess when the king has to move out of check.

### Feed-Over Technique

In a chess program much effort is put into sorting moves into the best order at the various levels of the tree and then selecting replies to each of the moves. This transforms the initial randomly organised tree into a well-ordered tree having all the best sequences of moves in one part of the tree.

Once a move has been selected the whole tree is lost and then rebuilt two plies further on at the next move. As this seems very wasteful we now save the most important parts of the tree and use them to speed up the searching process on the next move. The optimum sequence from each of my opponent's possible replies to my chosen move is saved. This sequence is then fed back in after my opponent has responded and examined first. This primes alpha-beta with a good value at the start, and this can produce a considerable reduction in the number of nodes examined. The overhead for feedover was found to be about 5 per cent, giving an average node reduction of approximately 30 per cent.

### Application to the Chess Program

Most of the better chess programs search to a 'Turing dead position'. This has been a standard technique for several years. We question this approach and instead attempt to compute a static evaluation for non-quiescent positions. In order to do this we have to have an element of lookahead in the evaluation function, achieved by giving values for threats to pieces as well as captures. We also include values for pins, x-rays, control of centre squares and squares near the kings. The value of threats by the player about to move is greater than that of the player who has just moved, producing an element of tempo in the evaluation function. By use of the tables we have also been able to give pieces different values in different positions on the board, for example knights are weak at the edges of the board and pawns increase in value as they advance.

The degree of pruning achieved is to some extent dependent upon the evaluation function. One can in fact use the node count to tune the evaluation, for if the node count goes very high on some move it is usually indicative that an important element is missing from the evaluation routine.

While our program could run into trouble evaluating long sequences of swaps, it gets compensation by examining the full width of moves, subject only to pruning by the various algorithms. This means that the

program will often select quite subtle moves, which would be lost if a fixed width of search were used.

The actual move selected at ply 1 has been as low as 28th in the list. On another occasion the program found a knight fork of a king and queen first in the list at ply 1, checked it out to ply 4 and then razored out the remaining 40 or so moves without examining any of them deeper than ply 1. The razor technique seems able to cut the tree ruthlessly when there is an obvious sequence of moves, but to search far wider when the position is uncertain.

Initial experiments indicate that a four ply deep chess program with alpha-beta takes about 40 seconds per move using the optimized PL1 compiler on an IBM 370/165. With razoring the time falls to about twelve seconds and with enhanced razoring about four seconds, while most of the moves selected remain the same.

## Conclusions

We have found that the razoring technique will cut about half the branches from each node left after alpha-beta pruning. We can therefore obtain *an order of magnitude improvement* in pruning a four-ply tree over ordered alpha-beta.

We have found no evidence to suggest that it is necessary to evaluate only at 'Turing dead positions'. In practice the program seems capable of handling quite turbulent positions.

The feed-over technique was found to be very useful and effective. It not only speeds up the search but also encourages the program to keep to a particular line instead of starting from scratch at each move.

The problem of the horizon effect has been observed only once in the last eight games played by the program. This problem occurs when a program throws away material to delay a strong move (e.g. a pawn queening) beyond the limit of its lookahead. The problem is obviously less likely to occur as the depth of search increases since the value of the material sacrificed will eventually be greater than the value of the pawn queening. The one case occurred in the game with R. Maybury at move 44 when the program sacrificed a knight to get a series of delaying checks. This put off a pawn queening to the 8th ply, i.e. beyond its horizon.

## Acknowledgements

Many people have helped in 'educating' MASTER. In particular we would like to acknowledge Jack Howlett for providing the machine time, Alex Bell for convincing many people that what we were doing with the time was sensible, Chris Osland for running the program whilst we were otherwise occupied and John Waldron, its tutor in chess. Many other people gave up their Sunday afternoons to play the program, some of these are recorded in appendix 2.

Finally, we thank all those in the Harwell and Atlas/Rutherford Computing Divisions (particularly the operators and shift leaders who had every right to throw us out but, surprisingly, never did).

# 4.2 Some Methods of Controlling the Tree Search in Chess Programs

**Grigori M. Adelson-Velskiy, Vladimir L. Arlazarov and Mikhail V. Donskoy**

First published in *Artificial Intelligence,* volume 6, 1975, pp, 361-371. Reprinted by permission of North-Holland Publishing Company, and of the authors.

**Abstract**

This paper describes several tree methods that have been implemented in our chess playing program (KAISSA) and some perspectives on this domain. The most general and most flexible method of program organization is exhaustive search with cut-offs. Therefore our program is based on searching exhaustively to a given depth, and permitting wide usage of cut-offs at any level of the search tree. Besides the program includes analysis of all exchanges up to their end. Experience with earlier programs has shown that disregard of exchanges would lead to gross errors.

## 1. A General Description of the Program

The program receives a position to choose a move from (initial position). Any method of such choice should be formulated in terms of cut-offs in the search tree. As a rule this is easy to do. The main parameter controlling a search is depth (or level) $n$ of a position under consideration in the game tree, or the number of plies (half-moves) which should be executed to obtain this position from the initial (for a given search) position. In the absolute schema, in all positions at depth $n \leq N$ (where $N$ is an integer chosen for a given search) all legal moves are considered (admitted for the search). If $n > N$, then the forcing schema is employed; namely only captures, checks and replies to checks are considered. Moreover, the computation of a static evaluation function is a forcing move and each side can decide whether it is more advantageous to compute the value of the evaluation function or to continue the forcing variation. The static evaluation function used by our program is of no particular interest and does not essentially differ from those used in other programs. It should be noted, however, that any change in the value of material is more important for the value of the function than any positional changes.

It is easy to see that the search described by these rules might have an unrestricted depth. Thus, a sequence of checks of unlimited length may occur during search when one side has a poor position, and the other side uses unsuccessful replies. However, because the algorithm that was implemented requires memorizing all positions on the branch from the initial one to the current one, an upper bound on the maximal depth is necessary. In the latest version of the program the maximal depth is 30. All positions at this depth are evaluated statically. When $N \leq 7$, the time to select a move under the above paradigm is acceptable. The depth of 30 is reached very seldom, owing to the application of numerous heuristics.

The value of terminal positions of the game tree is taken to be equal to the evaluation function. The value of non-terminal positions is taken to be the maximum or minimum (according to whether White or Black should make a move in the current position) of values of positions which are obtainable from the current one by an admissible move. The best move is the move leading to the position whose value is equal to that of the current position (it is possible that there exist a number of best moves).

To choose the best move in a given position it is necessary to find the value of this position. To this end, it suffices to find the value in some position obtainable by a move from the root node, and to prove that the remaining positions obtainable by other moves cannot give a better value. It is not necessary to exhaust all positions of the game tree to establish this proof. The cut-off procedure, known as the method of bounds and values, or $\alpha$-$\beta$ heuristics can be used. (Its essence can be easily illustrated

by the following example. If in a position a move is found which ensures the material balance, and after another move from the same position, a reply of the opponent wins a pawn, there is no need to search for a better reply.)

In addition to this method, another method was also used in our program. It is heuristic in the sense that it is not always valid. Namely, before the beginning of the search rather narrow limits are taken, between which the value of a given position is expected to lie. Captures and replies to checks are taken for forcing moves only in the case where they return the value of the static evaluation function to within the corresponding limits (upper for positions with Black's move and lower for those with White's move). This means that it may be necessary to repeat the search with other limits in the case where the value of a given position falls outside of the limits. The reduction of the search because of the narrow limits is larger than losses due to repetition of the search. But the main effect is obtained by the decrease in the number of moves in the forcing variations.

## 2. The Order of Move Consideration

When the method of bounds and values is used, the number of positions to be considered, i.e. the time of the move choice, depends essentially upon the order in which game tree positions are considered.

It is possible to show that this number is smallest when in all positions obtained by best moves the best reply is considered in the first place, and in all positions obtained by a worse move a so-called "closing move" is considered first. A "closing move" is a move which leads to a position whose value is beyond the adapted limits. (It may happen that after the strongest reply to a worse move, a rather extensive forcing variation can occur, and that closing by a weaker reply permits reducing the search.) To find such a reply, one must be able to solve the problem of move choice in all positions of the game tree that are actually considered. This is, of course, the main problem of conducting the search. However, in contrast to this main problem, an approximate solution involving the ordering of moves would suffice. It is highly important that an expected move be investigated first in the search, at least for most positions.

Various methods of preliminary estimation of the value of a move have been implemented in different versions of our and other programs. They are founded on some semantic analysis of these moves, i.e. on investigation of properties which are used to determine the quality of these moves. The preliminary estimation should be fast enough so as not to waste a considerable part of the time it saves. Therefore, the proposal that a static evaluation function of a position be taken as the preliminary estimate of the move which leads to this position was declined. To compute the evaluation function it is necessary to build a new position after the move and to determine in this position the values of all properties which are essential for computation of this function, i.e. to do much computation. It should be noted that the estimate provided by the evaluation function may be badly wrong, e.g. captures by major pieces of defended pawns may receive higher estimates than ordinary moves. To avoid the effect of bad estimates one must examine a forcing variation after the move. Such examination is utterly unacceptable in the absolute schema. Far more effective are ideas connected with the fact that the number of moves in chess play is far less then the number of positions. The former is no more than 10,000. Thus classification of moves is easier than classification of positions. The simple assumption that moves which were the best in many positions would most likely be good enough also in the current position underlies the so-called best-move-service. For each level, ten moves are stored which were most frequently the best or closing ones in positions of this level. To save storage this frequency is computed only approximately. When moves are ordered in a position of a level, these moves are placed at the head of the move-list of the position, and hence are considered earlier. The application of the best-move-service decreased the CPU-time of the absolute schema with $N=5$ tenfold. It should be noted, however, that further proposals of this kind, for instance, storage of the best replies, gave a very slight improvement.

A less trivial idea was that sometimes an extension of the game tree by introduction of dummy moves can lead to a reduction of the search tree. In positions with material

advantage (with respect to limits) it was permitted to try a so-called "blank" move in which ones own pieces are not moved. Certainly in positions with "Zugzwang" (the side-to-move must weaken his position) this may lead to errors. However, the standard of the program's play is still such that such errors do not essentially lower it. (We plan to develop in the future a procedure for recognizing "Zugzwang" positions, and to forbid a blank move in them.) The reduction of search takes place in this case, because the blank move proves to be a closing one and furthermore does not generate complex forcing variations.

The search for methods of search reduction different from the $\alpha$-$\beta$ heuristics and based only on general properties of two-person games with complete information has led us nowhere. One of the authors has proved that such procedures cannot reduce a search more than perfect application of the bounds and values method does. (The proof and some other questions discussed in this paper can also be found in Adelson-Velskiy et al., 1970.)

### 3. Strategies

Analyzing the performance of the absolute schema, the main attention should be given not to revealing the character of its mistakes but to determination of reasons for time waste due to the consideration of moves which are absurd from the human view-point. Indeed, it is usually easy to propose ways to eliminate a specific error, but every such elimination requires additional time. (The simplest way to strengthen the play of a program is simply to increase the depth of its search tree.) Therefore any strengthening of a program should be compensated by the introduction of more powerful methods of branch cut-offs. From this point of view the following two typical kinds of absurd variations were noticed during our study of the search tree. The first is caused by the fact that when a grossly erroneous move is made, even if it is refuted by the first reply of the adversary, the program attempts to save the situation by completely unnatural and absurd moves. The variations of the second kind are absurd for a more subtle reason. They cannot be rejected a priori, but they prove to have been already considered in a similar position. This reason is good enough for a human. Especially notable is the similarity of forcing variations considered in different positions. To avoid absurd moves of the first kind, numerous authors propose so-called strategies. (The notion does not coincide with the one of the chess theory. It is more akin to the notion of tactics, but yet more elementary.) In some languages possible intermediate goals of a game, and conditions which ensure that moves in a current position satisfy these goals, are formulated. Routines are constructed which determine moves which make sense in the strategies under consideration, and only these moves are taken as admissible. It should be noted that in this case the computer plays a game which differs from the absolute schema described above. Of course numerous absurd moves are cut off, but some delicate moves can also be cut off.

Thus a search reduction is achieved not by elimination of extensive search in undoubtedly poor positions, but merely by decreasing the number of positions to be considered; although, in general, it is achieved by decreasing the number of moves considered in worse positions, obtained after absurd moves.

Any program that will play according to the above method of strategies will waste much time considering variations of the second kind. This is so because a move satisfying some strategy in a position satisfies the same strategy in similar positions obtainable from the former position by one of two moves, and results of this move in both positions are almost the same. This assumption was tested for the so-called "active schema". In the active schema, in all positions at a depth less than $N$, forcing variations are examined for the side which is next to move. This means that this side is permitted to make two first moves successively, but the second and subsequent moves for each side should be forcing (checks, admissible captures and replies to checks). Those moves, after which a forcing variation results in material gain, are called active: they promise material advantages. All active moves and one passive, the best according to an estimate, are admissible for the search. (Since the forcing variation and computation of the evaluation function for position considered in it are used not only to reorder the move list, but chiefly to cut off moves, one may

spend additional time to examine this variation.) The active schema corresponds approximately to the method proposed by Botvinnik (1970) with horizon 2.

When $N=5$ the active schema program consumes $\frac{1}{5}$ to $\frac{1}{8}$ of the time of the absolute schema. This permits increasing the search depth by between one and two ply. The active schema was one of the poorest schemata of play among the various strategies that were tested. Thus the problem of search reduction by cut-offs of similar variations is very important.

## 4. Method of Analogies

Let us discuss first the search reduction by cut-offs of similar variations in the absolute schema. We will consider the whole tree, but consider absurd moves only once. We store these absurd moves, and will not consider searching them in other positions until circumstances appear which may essentially change the variation after this move. We believe that this is precisely how man plays chess, and that the standard of his play increases with the accuracy with which he determines the time when a move rejected earlier should be re-examined.

However, such a search reduction poses a number of problems:

(1) Which moves are absurd?

(2) Under which conditions should a move rejected earlier be re-admitted for the search?

In our program, those moves which lead to immediate loss of material are considered absurd. Below, an exact definition of an immediately losing move is given.

Let White's move $P$ be a losing move in position $U$. Then there exists a subtree $D(U,P)$, with root $U$. with the following properties. In the root $U$ there exists only the arc $P$. For the remaining vertices where White is to move, $D(U,P)$ contains all arcs which are associated with moves considered in the search. For each vertex, this set consists of all moves if the vertex is at depth $\leq N$; otherwise it consists only of forcing moves. In a vertex with Black's move, $D(U,P)$ contains only the arc associated with the best move of Black at this vertex. The subtree $D(U,P)$ will be from now on referred to as the variation of move $P$ in position $U$.

A move $P$ is said to be an immediately losing move if, in each vertex of $D(U,P)$ with White's move, the value of material is (from the view-point of White) less than at $U$. Therefore after a search with $\alpha$-$\beta$ cut-offs from the position $U'$ which is obtained from $U$ after the move $P$, one can easily

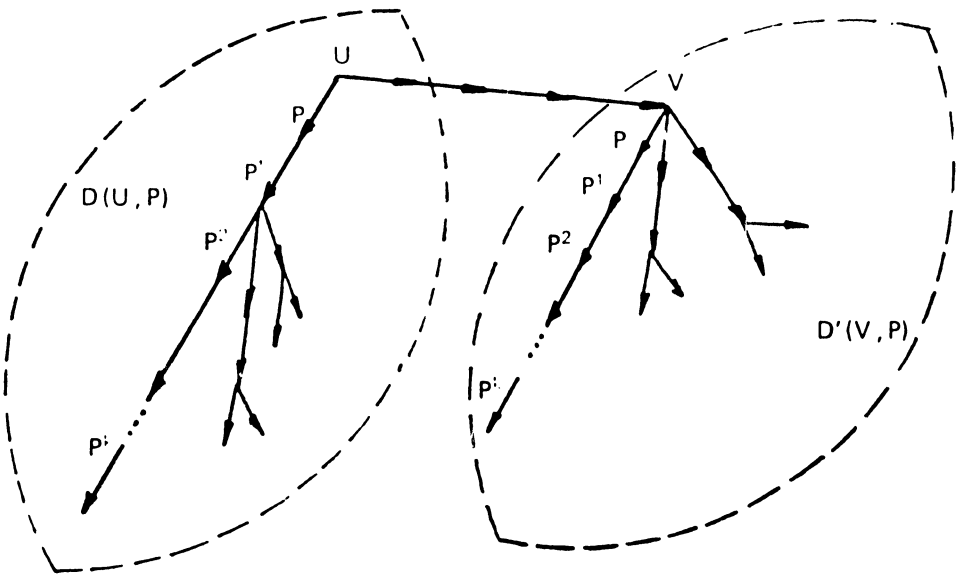



*Fig. 1*

establish whether this move loses immediately or not. The problem is how one may use the information from this search, in order to be able in other positions to establish that the move $P$ also loses immediately.

Suppose that in some position $V$ with the same material difference as in $U$, the move $P$ is not an immediately losing one. The variation $D'(V,P)$ of this move in this position is, by definition, the subtree with a root $V$ which contains all arcs beginning in vertices with Black's move and only one arc associated with the best move in vertices with White's move. (In the vertex $V$ there is only one arc, namely $P$, in this subtree.) A critical branch $P^1,...,P^k$ is defined as the intersection of $D'(V,P)$ and $D(U,P)$; the notion of a move which is used here to identify subtrees with different vertices is the class of arc equivalence, where arcs are associated with the same move, i.e. the same piece, the same square-from, the same square-to and the same captured piece (see Fig. 1).

The material difference along the critical branch in the variations $D(U,P)$ and $D'(V,P)$ is the same, since along this branch in both subtrees there are the same captures. In any position of this branch the material difference is certainly not in favour of the side whose turn it is to move, because this side could end (compute the evaluation function) also the variation which was lost.

Thus the best move on the critical branch is never the computation of the evaluation function! Hence a situation must arise where a move $P^{(k+1)}$, admissible in one of the variations $D(U,P)$ or $D'(V,P)$, is not admissible in the other (see Fig. 2).

Therefore the problem of, let us say, "geometrical" analysis of chess rules, arises. Namely, what changes in the position $V$ in comparison with the position $U$ can lead to a situation where an even level of $D'(V,P)$ a move exists which is inadmissible in the corresponding position of $D$(U,P), or at an odd level such a move exists in $D(U,P)$?

## 5. The Method of Geometrical Relations

The search reduction can be achieved by rather easy-to-check necessary conditions, although they might be insufficient. We succeeded in finding such conditions. Roughly speaking, they check whether the intersection of changes in the position $V$ in comparison with the position $U$ with the trace of $D(U,P)$ on the chess board is empty. To give a more precise formulation, a few definitions are necessary.

Let $A^0$ be the set of squares (on the chess board) which are squares-from for moves (Black's and White's) of $D(U,P)$; let $B^0$ be the set of squares on which in the position $V$ some piece stands that did not stand there in the position $U$.

Let $A^1$ be a set of squares-to of $D(U,P)$ and $B^1$ be the set of squares on which in the position $U$ some piece stood that does not stand there in the position $V$.

Let $A^2$ be the set of squares which have been attacked by White's pieces in some position of $D(U,P)$ with White's move. Namely, the square $s$ belongs to $A^2$ if in some position of $D(U,P)$ with White's move

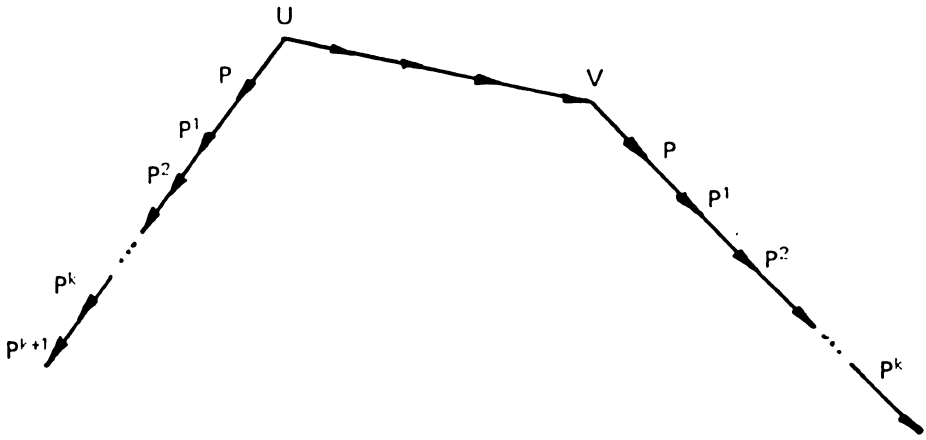



*Fig.2.*

it is under the attack of a White piece, but in the preceding position with White's move it is not attacked by the same piece. $B^2$ is defined analogously.

Let $A^3$ consist of squares of $D(U,P)$, where with White's move new possibilities are opened for checking the Black king. Analogously $B^3$ consists of those squares from which White may check the Black king in $V$, but could not do it in $U$ (all squares with these properties are taken, even those which cannot be reached according to the rules of chess).

Let $A^{4,0}$ consist of squares which are on paths along which Black's long-range pieces have moved in $D(U,P)$. $A^{4,1}$ consists of squares on the paths along which Black's long-range pieces have checked the White king in $D(U,P)$. $A^{4,2}$ consists of squares on the paths along which White's pieces were pinned (from the pinning long-range piece to the White King) in a position with White's move. Let $A^4 = A^{4,0} \cup A^{4,1} \cup A^{4,2}$.

Let $B^4$ coincide with set $B^{4,2}$ which is defined analogously to $A^{4,2}$.

The set of influence of changes in the position $V$ in comparison with the position $U$ on variation $D(U,P)$ is defined as the set

$$\begin{aligned}\Omega(D(U,P), V/U) = &[(A^0 \cup A^2) \cap (B^0 \cup B^1)] \\ &\cup [(A^0 \cup A^1) \cap B^4] \cup [(A^4 \cap (B^0 \cup B^2)] \\ &\cup [(A^0 \cup A^2) \cap B^3] \cup [A^3 \cap (B^0 \cup B^2)].\end{aligned}$$

The set of influence of changes in one variation on another is defined analogously. If $\Omega(D(U,P) = \emptyset$, one can say that changes in position $V$ in comparison with position $U$ do not influence variation $D(U,P)$. Influence between variation $D(U,P)$ and $D$ is defined analogously. The following theorem was proved in Donskoy (1974).

THEOREM 1. *Suppose that in position U all admissible moves were immediately losing, and all moves admissible in a position V were also admissible in U. If changes in position V in comparison with V do not influence variations D(U,P) for each move P, admissible in V, then all admissible moves are immediately losing in V.*

The theorem is valid for many model games which the chess program considers in place of chess. In these model games all positions are chess positions; all moves but blank ones are chess moves, but not all feasible moves are admissible. Our program widely uses the forcing variation as such a model game. The active schema is also an example of a model game. The conditions of the theorem are rather often satisfied for this game.

Two directions of further research can be mentioned. The first is to try to decrease the sets of influence described above, since they are very large. They are sometimes non-empty when there is no real influence, and when losing moves remain losing. It would be desirable to make the notion of influence more precise in such a manner that theorems on search reduction would remain valid, but the sets of influence would diminish. The second direction is extending the set of conditions for search reduction. In this direction, we have succeeded in obtaining interesting results and new methods were found.

## 6. The Final Model Game

In all positions but the initial one of the above variation $D(U,P)$ all moves admissible in a given model game are considered. To reduce the search one can use some specific model games with fewer moves.

Suppose that moves $P^1, ..., P^k$ are admissible in position $U$. Suppose, to avoid ambiguity, that in $U$ it is White's turn to move. In a "truncated model", $M^o$, all moves which are feasible according to chess rules are admissible in positions with Black's move. But in all positions except $U$ with White's move, the only moves which are admissible are those feasible ones which were inadmissible in $U$. In $U$ only $P^2, ..., P^k$ are admissible moves. For each move $P^i$ which loses immediately, one can construct the corresponding variation $D^o(U,P^i)$ of the truncated model $M^o$. The moves $P^i$ which do not lose immediately are eliminated from the set $P^1, ..., P^k$. Thus from now on it is assumed that every $P^i$ loses immediately in the models $M^r$, $r = 0, 1, ..., k$, we are to consider. If the truncated model $M^s$, $s \geq 0$, and variations $D^s(U,P^i)$ have already been constructed, then in the truncated model $M^{s+1}$ all moves are admissible in positions with Black's move, but in all positions except $U$ with White's move the only moves that are admissible are those (feasible ones) which were inadmissible in $U$ and also those $P^j$ for which $\Omega(D^s(U,P^i), D^s(U,P^j)) \neq \emptyset$, where $P^i$ is the first move on the branch from $U$ to the current position. If models $M^s$ and $M^{s+1}$ coincide, one denotes $\overline{M} = M^s$ and calls $\overline{M}$ a final model. Analogously $\overline{D}(U,P^i) = D^s(U,P^i)$ are called final

variations of immediately losing moves.

Let $V$ be a position with the same side to move as in $U$. Define model $\tilde{M}$ by the following rules. In all positions with the same move as $U$ in $\tilde{M}$ all feasible moves except $P^1, \ldots, P^k$ are admissible and in the remaining positions all feasible moves are admissible. Define next the variation $\tilde{D}(V)$ (called Brudno's variation) for model $\tilde{M}$ whose root is $V$. $\tilde{D}(V)$ consists of all moves which are considered in the exhaustive search with $\alpha$-$\beta$ cut-offs and has the top position $V$.

THEOREM 2. *If* $\Omega(\overline{D}(U, P^i), V/U) = \emptyset$ *and* $\Omega(\overline{D}(U,P^i), D(V)) = \emptyset$ *for all i, then* $P^1, \ldots, P^k$ *are immediately losing in the position* $V$ *(see Fig.* 3).

The proof of Theorems 1 and 2 will soon be published in a paper of one of the authors.

## 7. Some General Remarks

The construction of the final model $\overline{M}$ and of the final variations $\overline{D}(U,P^i)$ requires additional time. Therefore, the question arises in what positions these models should be constructed so that the total time is reduced. In the above truncated models, the only moves that are not considered are those which are feasible in some fixed position $U$. It is natural to pose the question under which conditions one need not consider various sets of moves in various positions of the part of game tree under consideration. The same search reduction is recursively used for variations defining these sets. It should finally be noted that in investigations which we have carried out, the influence of variations was related to material issues in the positions which were considered. But one should evidently include the positional estimate in a high-standard program. And this estimate should permit formulation (and proof) of positional analogs of the above theorems on influence. To this end it might be necessary not only to correct the evaluation function, but also to change the definition of sets of influence.

We wish to note in conclusion that all the above methods use some classification of moves. This direction of classification of moves and variants, and heuristics based on it seems to be very promising for the following reasons.

(1) As was already stated, the number of moves in a chess game (and also in most other games) is essentially lower than the number of positions. This is especially true for the number of moves in a specific search tree (owing to the closeness of its positions). (2) The relation of influence (which can be introduced for many games) and other notions which are used to formulate heuristic and accurate methods, are based on classification of moves and may be formalized. This permits avoiding inclusion of semantic notions and of notions such as "similarity" all of which we are not yet able to cope with, even in far simpler cases.

(3) It is possible that this classification underlies man's learning to play chess. In any case this presumption does not contradict today's biological knowledge and does not need the existence of still unknown mechanisms.

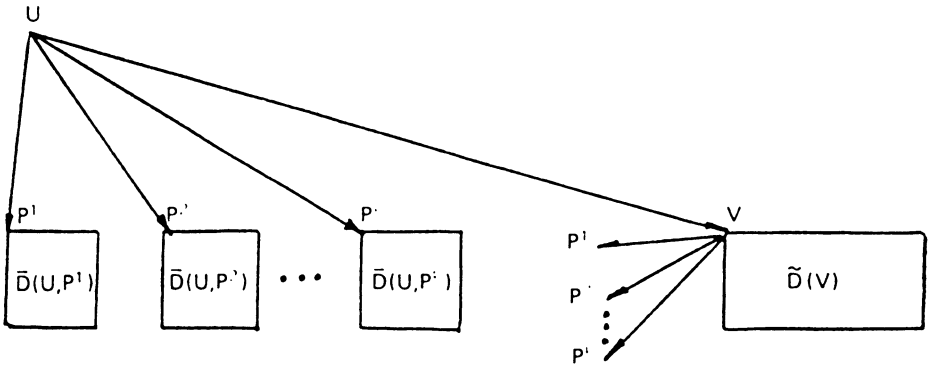



*Fig.3*

# 4.3 The Heuristic Search and The Game of Chess. A Study of Quiescence, Sacrifices, and Plan Oriented Play

**Larry Harris**

First published in *Proceedings of International Joint Conference on Artificial Intelligence,* Tbilisi, 1975, pp. 334-339. Reprinted by permission of the author.

**Abstract**

This paper describes the results of applying the formal heuristic search algorithm to the game of chess, and the impact of this work on the theory of heuristic search. It is not that the application of the heuristic search can by itself solve the problems at the heart of computer chess, but that representing these problems within the formalism of the heuristic search will further their common solution. A separate search heuristic is proposed that does offer a common solution to the problem of quiescence, sacrifices, and plan-oriented play.

**Introduction**

The purpose of this paper is to describe the results of applying the formal heuristic search algorithm to the game of chess, and discuss the impact of this work on the theory of heuristic search. We hope to demonstrate that a symbiotic relationship exists between the two areas in that the problems encountered when playing computer chess can be better solved within the formalism of the heuristic search and that the theory of heuristic search can be furthered by gaining insight from this very complex application.

Computer chess has been dominated by programs using the alpha-beta minimax search (Greenblatt, 1967) and more recently by programs using an exhaustive search (Northwestern, Kaissa, Tech II). The trend of successful programs has thus been to a more brute force approach rather than developing more formal solutions to the difficult problems that arise. Approaches using more sophisticated representation and utilization of chess knowledge such as Zobrist (Zobrist et. al., 1973) and Berliner (1974) have been unable to perform competitively with programs using the alpha-beta technique. The actual assessment of new approaches is hampered by the fact that a chess program hangs by its weakest link, and poor play may not be the fault of the new approach. However, approaches that attempt to use more sophisticated heuristics in preference to a general tree search are more susceptible to occasional serious errors in play. It is by no means clear that in chess the tradeoff of search effort for accuracy in evaluation can be consistently made without significant loss of precision. Thus the appeal of the heuristic search, an attempt to combine a sufficiently powerful search mechanism with a knowledgable heuristic.

The heuristic search approach has not proven itself competitively superior to the alpha-beta technique either – winning, drawing, and losing games with Columbia, North-western, and Tech II respectively. This substantiates the current feeling in the AI community that search strategy itself plays a lower order role than other modules of the system.

However, certain aspects of this approach are worthy of study since they may prove to provide a better mechanism for dealing with the problems at the heart of the computer chess dilemma; namely, quiescence checking, sacrifices, and plan oriented play. We will specify the means by which the heuristic search can express these problems, and propose a common solution, whereas the current minimax paradigm seems roadblocked.

**Definitions**

The heuristic search makes use of the following functions defined over the set of legal board configurations referred to as nodes.

g(n) = the number of moves from the

root to the node n.

h(n) = the minimum number of moves from node n to a goal.

f(n) = h(n)+g(n). The minimum number of moves from the root to a goal via node n.

Since some of these functions are not known during the actual search we must estimate them. The estimating function is denoted by a "
function of h(n) is denoted
typically referred to as the "heuristic" or the board evaluation function. A goal is a winning position.

At each step of the heuristic search the most promising line of play, the one with the best
the heuristic

The heuristic search for a game tree proceeds as follows.

1) Initialize the search tree to the current board referred to as the root.
2) Select a node for expansion by following

terminal node.
3) Expand p, linking all sons into the search tree.
4) Calculate
to calculate
5) Back up the values of the "best" son, setting the
the path to p.
6) Go to (2)

The algorithm terminates in step (2) or when time or space constraints are exhausted, in which case the move is made to the 1st level son with the best

"Best" is defined as either the maximum or minimum value depending on its level in the search tree. In step (5) the information obtained at the tip nodes is backed up into the search tree where it can be compared to other lines of play.

Most theoretical work dealing with the heurisitic search deals with the effect of restrictions on the heuristic and their resulting impact on the search. For example, if

guaranteed to find the minimal cost goal. By applying restrictions that allow for error in the heuristic we can determine theoretically how the heuristic search behaves under conditions of error (Harris, 1973; Harris, 1974). When dealing with chess we are forced to use an error prone heuristic or else no search would be necessary. Thus, it is critical that the search technique remain stable, even when misled by the heuristic. It is shown that the accuracy of the search degrades at worst only linearly with the error in the heuristic.

## The Search Heuristic

We begin our discussion of the problems of quiescence, sacrifices, and plan oriented play by introducing another forward estimating function called
argue later, the notion of a separate search heuristic, one that helps guide the search independent of the
tion in problem domains other than chess. We will argue the need to order the search on the basis of information other than that represented by
sions will now be determined by

For example, in tactical positions the estimator is of little use in searching for the best line of play since the notions typically measures, such as material advantage and pawn structure, become temporarily unimportant relative to an effective tactical threat. The
tries to forecast the tactical line and bias the search in this direction. Note that we are not saying that
the results of the tactics – if this could be done we would simply put the result directly in
oversensitive and signal when tactics *might* prevail. In chess cases we leave it to the search to investigate and verify the
tion. We hope to simply bias the search in the proper direction.

The need to separate
from our use of
be made on the board. In this way
the search, but does not directly affect move decisions, unless the
manifested by an improved
cessful search of the tactical line. Thus we associate moves with high
moves to make", and moves with high
being "good moves to investigate further".

## Quiescence Checking

In order to accurately predict tactical lines must be sensitive to much more than overt captures and checks, it must consider pins, forks, discovered attacks, back rank attacks, pawn promotion, and other phenonema likely to bring about an abrupt change in

piece advantage. In this sense the $\hat{d}$ measure is a quiescence check *par excellence.* When $\hat{d}$ is near zero then the $\hat{h}$ estimate can be considered accurate since there are no pending tactical threats. In this way the "horizon effect" (Berliner, 1973) can be avoided by stopping the search when all tip nodes of the search tree are quiet, in which case the search tree is said to have terminated.

Thus we are using the same estimator that finds tactical lines to define quiescence in a much more sophisticated manner than others (Greenblatt et al, 1967) have suggested. Typically only overt captures and checks are included in a quiescence check. $\hat{d}$ includes these even to the degree of signaling high values 1 ply after the check or capture to force continued evaluation of a forced line. But $\hat{d}$ also includes the tactical ploys that often precede sudden fluctuations in $\hat{h}$.

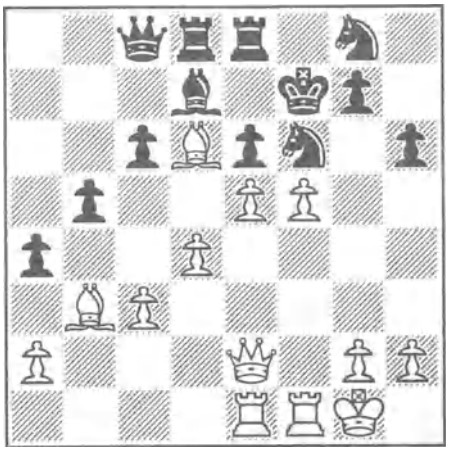
We must consider the problem that with this loose definition of quiescence, the tree may never terminate. In this regard we note that $\hat{d}$ features are not additive; that is, they do not cancel out. If both players have roughly equally promising threats, the $\hat{d}$ function must not sum to zero, as this board is anything but a quiet situation that can be accurately rated. By computing separate $\hat{d}$ values for each of the two players we can avoid this problem and can also define tree termination in a way that is more likely to occur. By separating each player's threats we can dynamically rate a board as stable when the son with the best $\hat{h}$ value has one sided threats that would only improve the rating. Using the same philosophy as alpha-beta cutoffs we can backup the quiet rating whenever one-sided attacks won't change the ordering even if they were to prove successful. In this way the root can eventually be rated as quiet even though many tip nodes of the search tree are not. This definition also makes sense from a search point of view since we would not wish to waste time searching a line of play whose outcome would not change the order of our ratings, while there remain nodes in the search tree whose ratings are unstable.

Quiescence checking in the heuristic search environment is on a scale relative to the entire search tree. The nodes with high $\hat{d}$ will be expanded first until they become quiet relative to other sections of the tree. In fact, one could view the search process as attempting to monotonically reduce the back-up $\hat{d}$ value of the root until it becomes close to zero; meaning that the dynamic rating of the root is accurate. If the value does not reach zero before it is time to move then the situation is too complex for accurate play, but we can at least make a reasonable move based on the information gained in reducing the $\hat{d}$ value of the root as much as possible.

In the alpha-beta minimax environmen the quiescence check must be made independent of other factors in the tree, making relative comparison impossible. The decision must be made at the point of maximum ply whether or not to continue the expansion. Since the algorithm is recursive there is no information about other areas in the tree. As such, the decision must be made on absolute grounds, requiring a conservative approach to make the algorithm terminate within the time constraint in all cases. Thus, most quiescence checks in the alpha-beta environment do not include all the factors necessary to determine whether a board is really quiet.

**Critical Sacrifices**

Another flaw of computer chess play is the inability to find or adequately defend from effective sacrifices. We will show how the search heuristic $\hat{d}$ can be used in solving this problem also. The following two boards are extreme examples of sacrificial play, but as such clearly demonstrate the



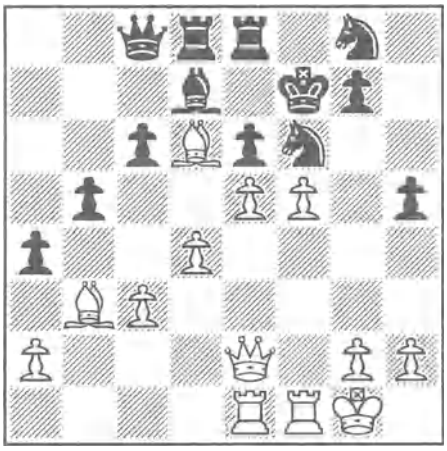
Board 1

problems computers face when required to search these lines of play. Board 1 is a famous 19-ply mating sequence described in Berliner (1974), that begins with a queen sacrifice. The line is 1 ♕e2-h5+ ♘f6×h5 2 f5×e6+ ♔f7-g6 3 ♗b3-c2+ ♔g6-g5 4 ♖f1-f5+ ♔g5-g6 5 ♖f5-f6+ ♔g6-g5 6 ♖f6-g6+ ♔g5-h4 7 ♖e1-e4+ ♘h5-f4 8 ♖e4×f4+ ♔h4-h5 9 g2-g3 any 10 ♖f4-h4 mate.

We are faced with the problem of finding such moves when they are successful, but also avoiding such moves when they are not. This latter concern is even more important since a program could play good chess without the ability to make 19-ply mates, but not if it continually enters such sacrifices when they are not successful.

The sacrificial move might be defined as a good move to investigate, but one to avoid making until its full effects are searched out. In the proposed scheme this would be a move with an incremental loss in $\hat{h}$ representing the sacrificed piece, and an artificially high $\hat{d}$ to temporarily overcome the loss in $\hat{h}$. This high $\hat{d}$ will bias the search to investigate the line in spite of the material loss.

Any search, minimax or heuristic, that uses a single value to compare boards and make moves, cannot separate the good from the bad sacrifices. The reasoning is that a 5-ply minimax search could not find the mate in a single search simply because of the depth of the sequence. But, it is entirely possible to *play* the mating sequence using a series of 5-ply searches. The problem arises in the first search, where we are forced to decide whether the queen sacrifice is successful or not, without being allowed to fully explore the sequence. If the heuristic rates the sacrifice as worthwhile then the move will be made without a thorough investigation. This of course, is very dangerous as Board 2 exemplifies. Board 2 is identical to Board 1 except that black's h-pawn is at h5 instead of h6. The queen sacrifice is not now successful because of this small change, since the king can eventually hide and avoid further checks. The line is: 1 ♕e2×h5+ ♘f6×h5 2 f5×e6+ ♔f7-g6 3 any ♔g6-h6 ending the checks. Surely it must be a sensitive heuristic to be *sure* of the outcome of a queen sacrifice in situations such as these. Since the stakes are so high, most programs will play it safe and will avoid the sacrifice.



Board 2

This is a result of the minimax forcing a premature decision without allowing a thorough investigation of the sequence.

The only way to avoid this two-sided dilemma is to separate the search ordering from the move ordering. Once we have allowed this distinction we could explore the queen sacrifice more deeply. If the sacrifice is successful this will be manifested by an improved $\hat{h}$ value. Until the $\hat{h}$ value of the sacrifice is rated as the best in the search tree we will not be tempted to make the move, but this does not restrain the search from investigating it further. Once again, the notion of a separate search heuristic, to guide the search to areas of the move tree that need further investigation before they can be accurately assessed, makes for a clean solution to this very difficult problem. For as long as move decisions and search criteria are based on a single value, sacrificial lines of play like those exemplified above cannot be accurately solved.

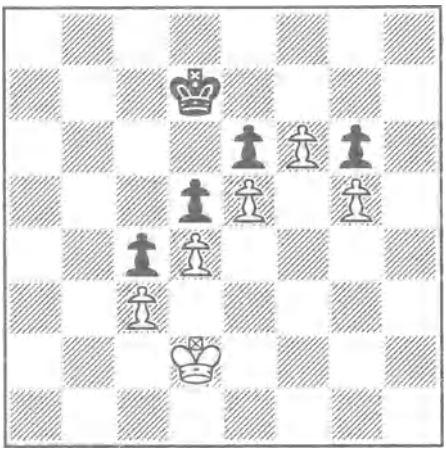
Clearly some tuning of the $\hat{d}$ heuristic is required to avoid investigating every seemingly interesting possible sacrifice. This is done by requiring substantial mobilization of attacking pieces combined with limited mobility of the attacked piece before the $\hat{d}$ rating will suggest the further consideration of a sacrifice. Both of these conditions exist in the above situation and would allow for the sacrificial line to be searched even beyond the 8th move which is not a check for White.

**Plan Orientated Play**

Computer chess programs are most often criticized because of their lack of using a plan to guide their play. The program may know to develop its pieces and castle early, thus playing the opening fairly well, only to flounder early in the midgame waiting for something to happen. Against human opponents the game changes rapidly enough so that the program must react to threats and can formulate short range attacks so the lack of a plan is not always apparent. When two computers play each other the probability of stagnation increases markedly.

Nowhere, however, is the lack of a plan more critical than in endgame play. In these cases a winning pawn advantage can be lost simply because a long range plan is needed to promote it. Let us consider some examples of such endgames and how the heuristic search environment provides a means of using long range goals. Our concern here is not the very difficult problem of suggesting what plans are worthy of consideration in a given situation, but the difficulties that occur when we try to accurately measure the effectiveness of a given plan.

A position suggested by Berliner (1973) is shown in Board 3. In this case the white king must simply flank the black pawn wall to support the advance of the the white pawn at f6. In this case the plan itself is quite clear, but the manner in which it must be carried out is hard for a computer to find. The problem appears difficult to a computer program because the white king must first move *away* from the desired position supporting the advance of the pawn at f6. We can expect that most chess heuristics will rate the moves ♔d2-e3 or f6-f7 initially better than the required ♔d2-c2. Given this error in the heuristic how would the minimax and heuristic searches react?



Board 3

Berliner gives a detailed scenario for the minimax. The move f6-f7 will soon lose the pawn so the rating drops accordingly. But the rating for the move ♔d2-e3 increases as the king progresses directly towards its goal. The rating for ♔d2-c2 will be relatively low since the distance from the king and its goal is widened. At the bottom of the 5-ply search tree the minimax will compare the relative closeness of the king to the passed pawn obtained by ♔d2-c2. The choice is quite clear and the minimax backs up a descendant of ♔d2-e3 not realizing the futility of the move.

How then can the heuristic search, operating with the same error prone heuristic, find the winning move? It should not be surprising that the heuristic search can solve similar puzzles such as the 9-puzzle (Nilsson, 1971). It is just for situations such as this in which the heuristic search has been most successful. The search scenario proceeds as follows.

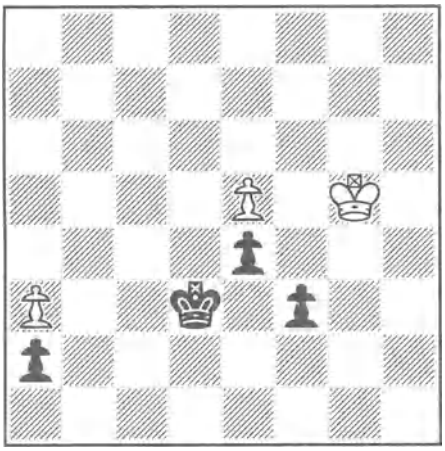
The moves f6-f7 and ♔d2-e3 will undoubtedly be expanded first because of their high ratings relative to ♔d2-c2. In fact the move ♔d2-c2 would remain unexpanded for quite some time until all other moves demonstrate their lack of merit. Soon f6-f7 will lose the pawn and be rated below ♔d2-c2. However, the descendants of ♔d2-e3 will at first improve as the king progresses. When the king reaches f4 the king will be forced to retreat. This retreat plus the build up of these lines will soon diminish their ratings until the point that ♔d2-c2 is expanded.

In the previous example there was one obvious plan and were it not for "getting over the hump" of the pawn wall, the solution would be easily found by any search technique. A similar problem arises when there are multiple plans that could apply in a situation and the best plan requires more moves to become effective.

Consider the situation in Board 4. Here, Black could formulate two plans to exploit his pawn advantage. He could aggressively

attack the weak white pawn at h6 by moving ♔e6-f6. Black could also try to create a passed pawn in two ways by moving either ♔e6-d6 or ♔e6-f5. The first two moves lead to a draw, but this is not evident for 6 ply. The 3rd plan wins, but is not clearly evident for about 15 ply. Thus a program that searches 5 ply on all 3 lines will be unable to distinguish the good plan and is likely to draw the position.

White



Black

Board 4

When searching with multiple plans in mind is not enough to rate by simply taking the best value obtained by any plan. This would tend to bias the search towards the plan that has the most immediate payoff. This, of course, can prove to be short sighted, often ignoring the better plan. For example, if we are required to rate Board 4 using only one value, then we are forced to determine statically which plan is best. When this determination is in error, as will often be the case, the search would be biased away from investigating the best plan. It is actually necessary to keep separate ratings of each plan and either perform the search sequentially for each plan or perform it in parallel using the multiple heuristic values. The former case would seem prohibitive with regard to time. The latter case would require backing up multiple information for each plan. This would severely hinder the ability to make alpha-beta cutoffs since the cutoff would have to occur for all plans. On the other hand, the heuristic search could simply continue the expansion of both plans taking into account the $\hat{g}$ factors so that small "humps" to be crossed, and a bias in $\hat{d}$ to investigate all plans to the depth necessary to accurately assess their potential.

Thus, as plan orientated play is introduced into computer chess, it is clear that part of its implementation must be in biasing the search deep enough to accurately measure the success of the plan. The search heuristic $\hat{d}$ is one means by which this can be carried out.

**Impact upon the General Heuristic Search**

We have seen the utility of the search heuristic $\hat{d}$ in chess. The use of this heuristic came about because the most effective order of expansion differed from the order determined by the $\hat{g}+\hat{h}$ measure. That is, some nodes were expanded, not because of high $\hat{g}+\hat{h}$ ratings, but because their $\hat{h}$ rating could not be accurately measured. In these cases the search itself must be used to help accurately rate the node.

Considerations such as this already appear in the literature of the heuristic search, although in a different terminology. In applying the heuristic search to solve the Travelling Salesman Problem, Pohl (1973) uses $\hat{h}$ with two factors that are really search heuristics. Both of these factors tend to more effectively order the sequence of expansions. Since they are based on search criteria (depth in the tree) and not based on the node itself, we could label them as part of the search heuristic $\hat{d}$.

This relabelling allows us to distinguish the heuristic factors relating to the node itself, $\hat{h}$, and those relating to the search process $\hat{d}$. Once this is done we can standardize the definition of $\hat{f}$ to be $\hat{f} = \hat{g} + \hat{h} + \hat{d}$ and recognize that this "biasing of $\hat{f}$" is an important aspect of the algorithm itself and will be necessary in many problem domains where high performance is required.

One spin-off of work on a high performance chess program has been this recognition of the need for a separate search heuristic. There are other insights to the general technique of heuristic searches that can be gained from high performance chess. We use the formalism of the heuristic search

to abstract the ideas of chess programmers and, if possible, apply them in other search domains.

An example of this is Berliner's "Refutation Description", in which he reduces the search space by passing information up the tree. It is applied in chess when a certain line of play thought to be good, ends in a catastrophe. At this point details of the catastrophe, such as the attacker, attack path, attackee, and flight path are passed back up the tree. From this point only moves that avoid the catastrophe are considered.

If we describe this technique in the terminology of the heuristic search, we can try to apply it to other problem domains. Whenever the heuristic rises suddenly (gets worse) after the expansion of a node, then we have detected some sort of catastrophe; since the path was considered to be the best we had, or we wouldn't have been expanding it. If we can now isolate the reason for this sudden rise in $\hat{h}$, this information could be used to reduce the search space.

Hopefully, continued work on high performance problem solving systems such as chess will continue to contribute searching techniques that can be applied effectively across problem domains.

# 4.4 A Theory of Evaluative Comments in Chess with a Note on Minimaxing

**Donald Michie**

First published in *The Computer Journal,* volume 24, number 3, 1981, pp. 278-286. Copyright Heyden & Son Ltd., 1981. Reprinted by permission of John Wiley & Sons Ltd.

Classical game theory partitions the set of legal chess positions into three evaluative categories: won, drawn and lost. Yet chess commentators employ a much larger repertoire of evaluative terms than this, distinguishing (for example) a 'drawn' from a 'balanced' position, a 'decisive' from a 'slight' advantage, an 'inaccuracy' from a 'mistake' and a 'mistake' from a 'blunder'. As an extension of the classical theory, a model of fallible play is developed. Using this, an additional quantity can in principle be associated with each position, so that we have not only its 'game-theoretic value' but also its 'expected utility'. A function of these two variables can be found which yields explications for many evaluative terms used by chess commentators. The same model can be used as the basis of computer play. It is shown to be easier to justify, and to adjust to realistic situations, than the minimax model on which state of the art chess programs are based.

## Introduction

The game tree of chess contains about $10^{46}$ positions (Good, 1968) a substantial proportion of which are terminal. The rules of the game assign a value to every terminal position, +1, 0 or −1 according to whether the position is won, drawn or lost for White. These values can be backed up the game tree using the minimax rule, so that in principle every position can be given a value, including the initial position. This last is known as 'the value of the game', and is widely conjectured to be 0 for chess. If this conjecture is correct, and if both sides play faultlessly, i.e. only execute value-preserving moves (it follows from the 'back-up' method of assigning values that there is at least one such move available from every non-terminal position), then the game must end in a draw. A fragment of a hypothetical game tree is depicted in Fig. 1. In Fig. 2 the method of attaching game-theoretic values to positions is illustrated.

An evaluation function could, in principle, map board positions into a larger set of values, making it possible to express a distinction between positions which are 'marginally' won and positions which are 'overwhelmingly' or 'obviously' won, or between drawn positions in which White, or Black, 'has the edge' and drawn positions which are 'equally balanced', and so forth. Two circumstances suggest that a useful purpose might be served by multi-valued functions.

(i) Chess Masters and commentators have developed a rich descriptive language for the expression of such distinctions.

(ii) Computer chess programs employ real-valued functions for evaluating terminal positions, not of the game tree which is too large, but of the lookahead tree. Values backed up from the lookahead horizon are used to select the next move. We lack a formal basis for assigning definite interpretation to such values.

There is thus a need for a stronger theory of position-evaluation. This paper discusses chess, but the treatment is general and covers all two-person zero-sum games of perfect information without chance moves.

## Requirements of a theory

A good theory should explicate a variety of commentators' concepts. Table 1 is a representative list. Where a conventional symbol is available it precedes the verbal comment.

## Main features of the theory

The game-theoretic model presupposes perfect play, whereas in the real-life game of chess (whether human or computer) both

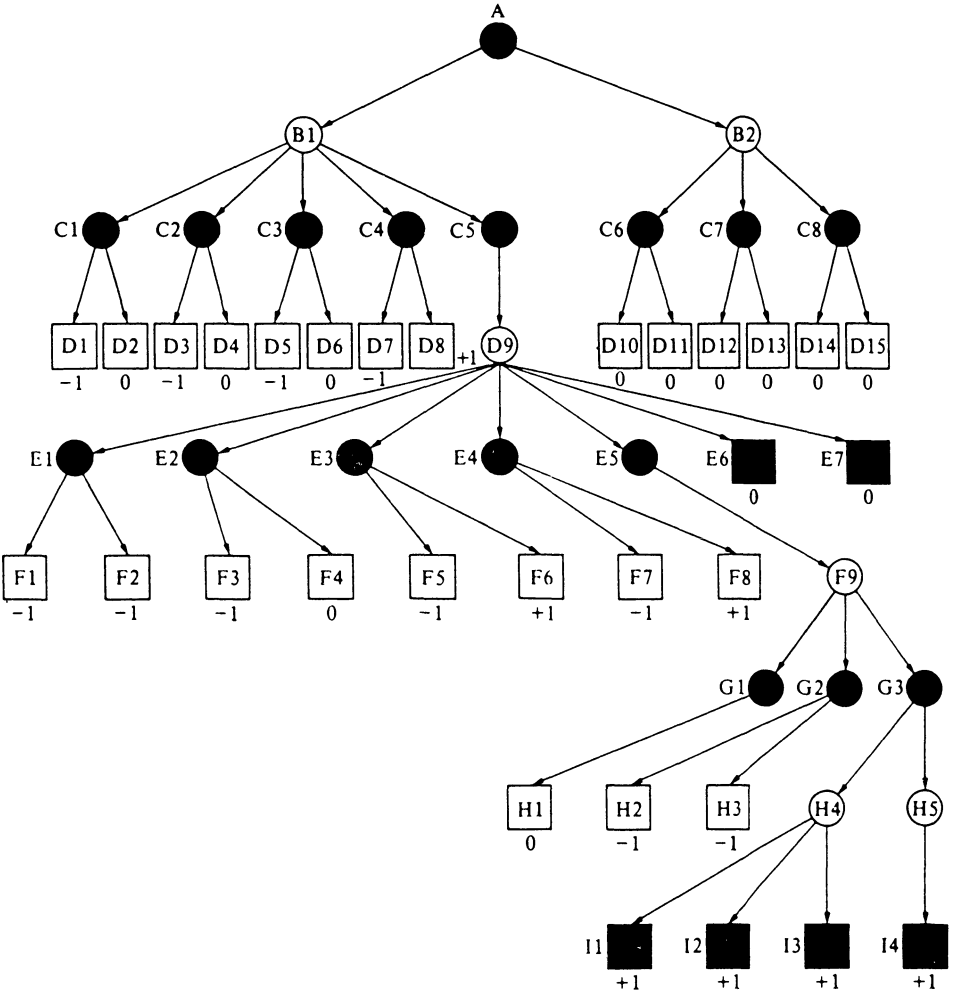



*Fig. 1 A game tree with its terminal nodes (shown as squares) labelled with outcome values from the set (+1, 0, −1). Shading of the remaining nodes (circles) indicates which player has the move.*

sides are susceptible to error. Our theory is based on this distinction, and presents the following main features:

(1) We follow Good (1968) and interpret the values of terminal positions as *utilities* as though the game were played for a unit stake. values for preterminal positions are then calculated as *expected utilities*. In order to avoid confusion we shall refer to these throughout as 'expected utilities' or 'scores', never as 'values', reserving the latter term for game-theoretic values.

(2) A model of imperfect but skilled play is developed. Chess skill appears in this model as an adjustable parameter running from O (random play) to ∞ (perfect play).

(3) In the new model the classical game-theoretic treatment appears as a special case.

## The Calculation of Expected Utilities

Consider a state $s_0$, from which transitions to successor states $s_1, s_2, s_3, \ldots s_n$ can occur with respective probabilities $p_1, p_2, p_3, \ldots p_n$. Let us suppose that these successor states have associated utilities $u_1, u_2, u_3, \ldots u_n$. Then the expected utility associated with $s_0$ is

$$\sum_{i=1}^{n} p_i u_i$$

It follows trivially that if we interpret as

utilities the values attached by the rules of chess to the terminal positions then the values assigned to the non-terminal positions by minimaxing can be interpreted as expected utilities. In this special case the *p*s associated with those arcs of the game tree which carry a change of game-theoretic value are all 0. Consequently, the evaluation of $\sum_{i=1}^{n} p_i u_i$ at each node reduces to obtaining the 'min' or the 'max' of the successor-values according to whether White or Black has the move. The above specification is ambiguous in the case when two or more of the moves applicable to a given board position are value-preserving. We can either select one of these at random and assign a probability of unity to it and zero probabilities to the rest, or we can divide the unit probability equally among them. In the case of error-free play, calculation of expected utilities according to either procedure leads to the same result. As the basis of a model of actual play we shall adopt the second alternative, which is illustrated in Fig. 2.

We now relax the game-theoretic condition that at each choice-point on the tree there is a probability of unity that a value-preserving move ('sound' or 'correct' move) is chosen, and we introduce the possibility of error. In constructing a model of error, we express the relative probabilities of making alternative moves from a given position as a monotonic increasing function (decreasing function for Black, since all utilities are expressed from White's standpoint) of the expected utilities of the corresponding successor positions. Thus the move leading to the highest expected utility will be chosen with highest probability (but not with probability 1 as in the game-theoretic error-free model), the move leading to the next highest expected utility with next highest probability and so on. We thus envisage an idealised player whose statistical behaviour reflects the rank-ordering of the expected utilities of chess positions. Using such a model it is again possible to label all the nodes of the tree, working upwards from the terminal nodes, but by a procedure which differs from the minimax method.

## The Notion of Discernability

In order to carry out some illustrative computations based on this idea, we now choose an actual monotonic function. No significance is claimed for the particular choice, since the points which we seek to establish are qualitative rather than quantitative. Certain ideas must, however, be reflected in any such function. A central one is that of *discernibility*. We conceive the player as standing upon a given node of the game-tree and looking towards its successors. These are labelled with their expected utilities, but the labels are not fully discernible to him. Discernibility is directly related to the strength of the player (the labels are fully discernible to an infinitely strong player) and inversely related to the number of moves separating the node from the end of the game: next-move mates and stalemates are fully discernible even to the beginner, but next-move expected utilities obtained by backing up are less so. Reflecting these considerations, we shall define the discernibility from a board state $s_0$ of the expected utility of a given successor state $s_j$ as:

$$d = (M + 1)^{[3(r_j+3)/(r_j+\epsilon)]} \quad (1)$$

where $M$ is the merit of the player in kilo-points of the US Chess Federation scale, so that $0 \leq M$, and $r_j$ is the number of moves that the value associated with $s_j$ has been backed up. The symbol $\epsilon$ denotes an arbitrarily small quantity introduced to avoid the expression becoming infinite for $r_j = 0$.

The expected utilities themselves are real numbers lying in the range from $-1$ through 0 to $+1$. They are interpreted as being in logarithmic measure, to base $d$. Using this base, we take the antilogarithms of the expected utilities associated with the $n$ successors of a given position as giving the *relative probabilities* with which a player of merit $M$ who has reached $s_0$ selects the corresponding moves. Thus, for the transition $s_0 \rightarrow s_j$,

$$p_j \propto d^{u_j} \quad (2)$$

Normalising these so as to obtain actual probabilities, $p_1, p_2, \ldots p_n$, the expected utility of a position is evaluated as $\sum_{i=1}^{n} p_i u_i$ where $u_i$ is the expected utility of the position generated by the $i$th member of the set of available moves. Starting at the terminal positions, this gives a method of assigning expected utilities to successively higher levels of the game tree until every position has been labelled.

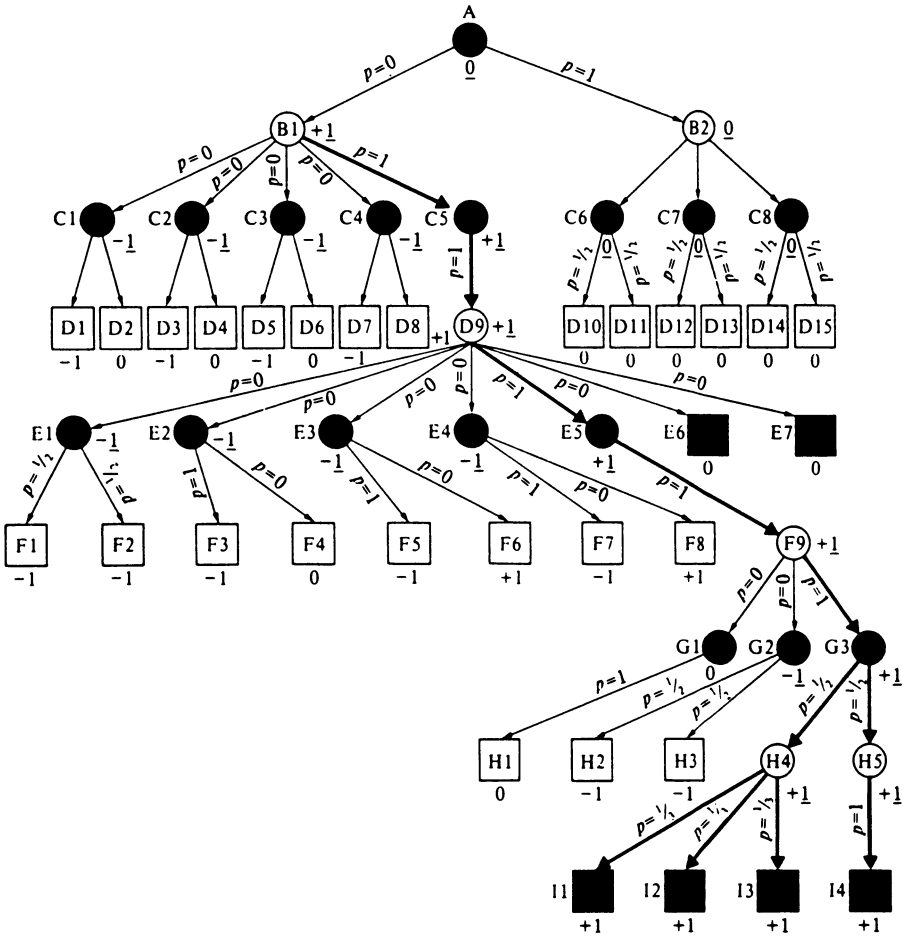



*Fig. 2 The game tree of Figure 1 with its non-terminal nodes labelled, (underlined values) by minimax back-up. White's best strategy from B1 is drawn with a heavy line. Arcs are marked with the conditional move-probabilities corresponding to perfect play: since the game-theoretic value of B1 is +1, Black chooses with probability 1 to move to B2.*

**Table 1 A representative list of commentators' comments**

| | | | |
|---|---|---|---|
| ( 1) | A dead draw (nothing that either player can do can avert a draw) | (12) | A mistake |
| ( 2) | A complicated position | (13) ?? | A blunder |
| ( 3) = | A balanced position | (14) | A strong move |
| ( 4) ± | White has a slight advantage | (15) !! | A very strong or brilliant move |
| ( 5) ± | White has a clear advantage | (16) !? | A brilliant but unsound move |
| ( 6) +− | White has a decisive advantage | (17) | Best move |
| ( 7) | A certain win for White | (18) (!) | Best move in difficult circumstances |
| ( 8) | A difficult position for White | | |
| ( 9) | A losing move | (19) | A safe move |
| (10) | An inaccurate move: White weakens his position | (20) | White should press home his advantage |
| (11) | White strengthens his position | (21) | Black should play for time |

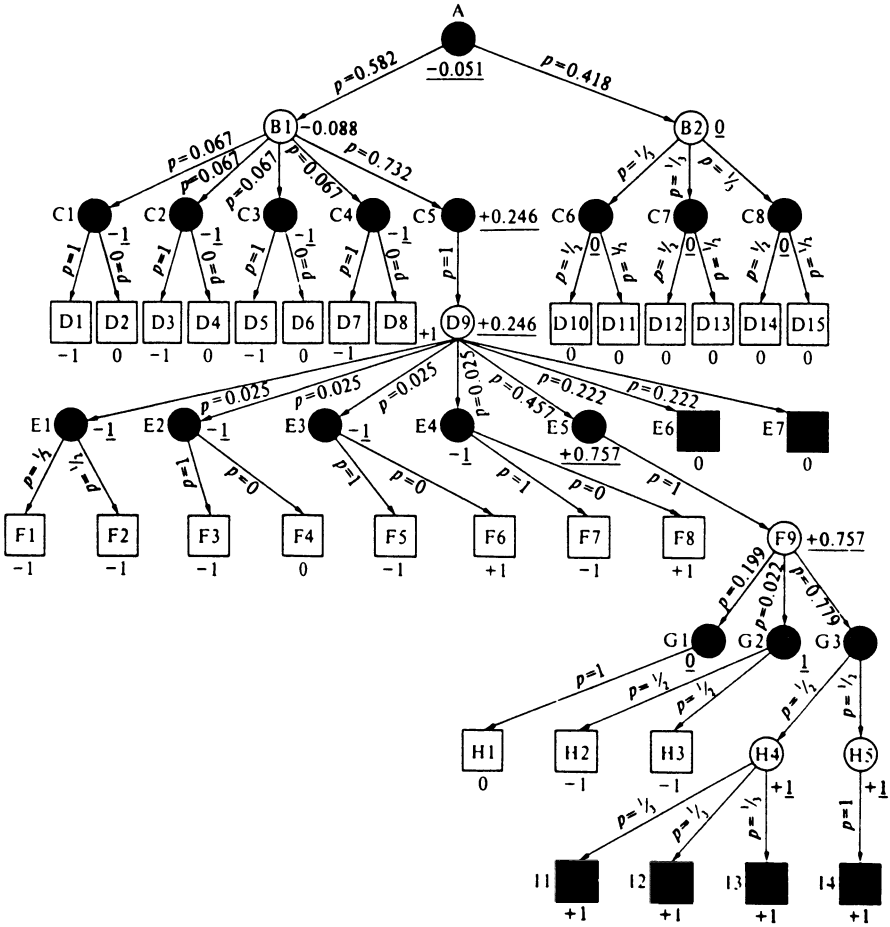



*Fig. 3 The game tree of Figure 1 and 2 labelled with expected utilities calculated from a model of fallible play. White has been credited with playing strength $M_W = 0{\cdot}2$ and Black has $M_B = 1{\cdot}4$. Conditional move-probabilities generated by this model are entered against the corresponding arcs and are used to 'back-up' expected utilities to successively higher levels. As before, backed up values are underlined.*

## A Sample Computation

Consider the terminal fragment of game-tree shown in Fig. 1. We shall illustrate step by step the calculation of expected utilities so as to label every node in the diagram. First we make assumptions for the playing strength $M_W$ and $M_B$ of White and Black respectively. If we are to extract examples of the broad range of evaluative concepts from so ultra-simplified a game tree we must set these strengths very low. Let us set $M_W = 0{\cdot}2$ and $M_B = 1{\cdot}4$: White is thus an abject beginner and Black a weak tournament player. In our model $M = 0$ implies random play. The notation $u(s)$ denotes the expected utility of position $s$.

H4: All successors have the same value, +1: $u(\mathrm{H4}) = +1$.

H5: There is only one successor, so the move-probability is unity: $u(\mathrm{H5}) = +1$.

G1: Unique successor: $u(\mathrm{G1}) = 0$.

G2: Equivalued successors: $u(\mathrm{G2}) = -1$.

G3: Equivalued successors: $u(\mathrm{G3}) = +1$.

F9: From proportionality (2) we have
**Move to** G1: $d^0 = 1 =$ relative probability.
**Move to** G2: $r = 1$, so, from Eqn (1), $d = 1{\cdot}2^{12} = 8{\cdot}915$.

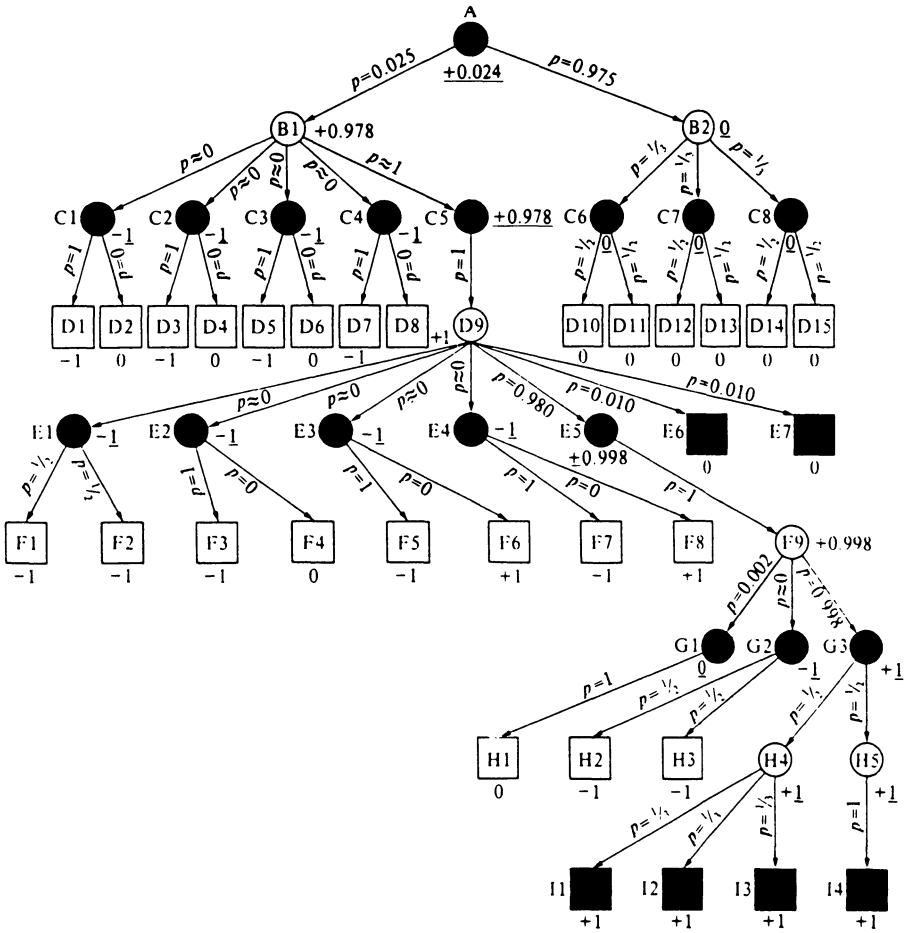



*Fig. 4 Expected utilities backed up the game tree using a different assumption about the strengths of the players, namely $M_W = M_B = 1 \cdot 4$; i.e. both players are of weak club standard. The expected utility associated with the root node now favours White, and the model of Black's play shows a 40:1 preference at this choice-point for the 'safe draw'.*

Relative probability = 1/8·915 = 0·1121.
**Move to** G3: $r = 2$, so $d = 1 \cdot 2^{7.5} = 3 \cdot 925$ = relative probability.
Normalised probabilities:
G1, 0·1985; G2, 0·0222; G3, 0·7792.
$u(F9) = (0 \cdot 1985 \times 0) + (0 \cdot 0222 \times -1) + (0 \cdot 7792 \times +1) = +0 \cdot 757$

E1: Equivalued successors. $u(E1) = -1$.

E2: $r = 0$. $u(E2) = -1$, and similarly for u(E3) and $u(E4)$.

E5: Unique successor. $u(E5) = 0 \cdot 757$.

D9: **Move to** E1: $r = 1$. $d = 1 \cdot 2^{12}$. Relative probability = 1/8·915 = 0·112 and similarly for moves to E2, E3 and E4.
**Move to** E6: Relative probability = 1, and similarly for move to E7.
**Move to** E5: $r = 4$, $d = 1 \cdot 2^{5.25} = 2 \cdot 604$. Relative probability = 2·0640
Normalised probabilities:
E1, 0·025; E2, 0·025; E3, 0·025; E4, 0·025; E5, 0·457; E6, 0·222; E7, 0·222 (total 1·001).
$u(D9) = (0 \cdot 457 \times 0 \cdot 757) - 0 \cdot 100 = 0 \cdot 246$.

C1: $r = 0$. $u(C1) = -1$, and similarly for $u(C2)$, $u(C3)$ and $u(C4)$.

C5: Unique successor. $u(c5) = 0 \cdot 246$.

C6: Equivalued successors. $u(C6) = 0$, and similarly for $u(C7)$ and $u(C8)$.

B1: **Move to** C1: $r = 1$. $d = 1 \cdot 21^{12}$. Relative

probability $= 1/8{\cdot}915 = 0{\cdot}112$ and similarly for moves to C2, C3 and C4.
**Move to** C5: $r = 6$. $d = 1{\cdot}2^{4.5} = 2{\cdot}272$.
Relative probability $= 1{\cdot}2240$.
Normalised probabilities:
C1, 0·06703; C2, 0·06703; C3, 0·06703; C4, 0·06703; C5, 0·73190 (total 1·00002).
C2u(B1) $= (0{\cdot}7319 \times 0{\cdot}246) - 0{\cdot}2681 = -0{\cdot}088$.

B2: Equivalued successors. $u(\mathrm{B2}) = 0$.

A: **Move to** B1: $r = 7$, $d = 2{\cdot}4^{4.286}$.
Relative probability $= 1{\cdot}391$.
**Move to** B2: Relative probability $= d^0 = 1$.
Normalised probabilities:
B1, 0·582; B2, 0·418
$u(\mathrm{A}) = (0{\cdot}582 \times -0{\cdot}088) + (0{\cdot}418 \times 0) = -0{\cdot}051$.

In Fig. 3 the tree is shown with expected utilities, calculated as above, attached to the nodes. The expected utility of the root node, A, turns out to be one twentieth of a unit in Black's favour – a 'slight plus' for Black. The analysis of Black's 'plus' is worth pursuing, for it illustrates certain fundamental concepts to which our theory is directed, in particular the idea that a *losing move* (in the game-theoretic sense of a transition for White to value −1 or for Black to value +1) can also be the 'best' move against a fallible opponent.

Note that Black can secure a certain draw by moving to B2. Note also that the move to B1 is a losing move in the game-theoretic sense, for White can then win by the sequence B1 → C5 → D9 → E5 → F9 → G3, as shown by the heavy line in Fig. 2. Yet the *expected utility* of the move, −0·088, is marginally better for Black than that of the 'correct' move (expected utility zero), and our model of Black, possessed of a weak tournament player's discernment, shows 58% preference for the move. The statistical advantage arises, as can be seen in inspecting the diagram, from the fact that play is switched into a subtree where the error-prone White has numerous opportunities for error presented to him. He has to find the needle of sound play in a haystack of hazards. In such a situation we sometimes say that Black sets 'traps' for his opponent. If the aesthetic features of the move to B1 appeal to the commentator, he may even use the annotation '!?', which we take to mean 'brilliant but unsound'. A sufficient increase in the strength of White could give cause to remove the '!' or even convert it into a second '?'. To illustrate this point we have recalculated the entire diagram after setting $M_W = M_B = 1{\cdot}4$, shown in Fig. 4. Here the move to B1 does not appear as 'best', nor even as a mistake, but as a blunder, and correspondingly our model of Black shows a preference of approximately 40:1 for B2.

Returning to the list of specimen evaluative comments in Table 1, we can now derive explications for them (Table 2). Wherever possible, an explication is expressed in terms of two functions of a board position, namely its game-theoretic value $v$ and its expected utility $u$. Where a move, rather than a position, is described, we use the notation $\Delta v$ and $\Delta u$ to denote the changes in the corresponding quantities affected by the move. We denote by $s_1$ the position from which the move is made and by $s_2$ the position which it generates. Some items of the original list have for completeness been differentiated into sub-concepts. Some of these would never appear in a chess book although under assumptions of very low playing strength they are generated by our model. Case 2 of (6) is an example of this: 'a decisive advantage' of this kind would characterise, for example, the initial position if Bobby Fischer gave Queen odds to a beginner.

We exhibit systematically in Table 3 various combinations of $u$ and $v$, entering in each case the evaluative comment which seems most appropriate.

## 'Tension'

The minimax value of $s$ can be regarded as in some sense summarising the values of the terminal nodes of the tree rooted in $s$. More obviously, the expected utility of $s$, which has the form of a weighted mean, constitutes a summary of a different kind of this same set of quantities. It seems natural to proceed to statistics of higher order, i.e. from representative values and means to variances. Might such second-moment statistics also possess recognisable meaning in terms of the chess commentator's vocabulary?

Good (1968) discusses a property of chess positions which he calls 'agitation'. He defines it by considering how sharply the estimated utility of a position is changed by investing a further unit of work in deepening the forward analysis. This quantity will necessarily be positively related to the

*Table 2 Explication of the evaluative comments of Table 1*

| | Comment | Explication |
|---|---|---|
| (1) | A dead draw | $v=0$ for all terminal descendants of $s$ |
| (2) | $s$ is complicated | The first few levels of the tree rooted in $s$ have branching ratios |
| (3) =, | $s$ is balanced | $v=0$ *and* $u \simeq 0$ |
| | *Case 1:* $s$ is lifeless | $\mathrm{var}(v_t) \simeq 0$ } see text |
| | *Case 2:* $s$ has high tension | $\mathrm{var}(v_t) \gg 0$ } see text |
| (4) ±, | White has a slight advantage | $v=0$ *and* $u>0$ |
| (5) ±, | White has a clear advantage (good winning chances) | $v=0$ *and* $u \gg 0$ |
| (6) +−, | White has a decisive advantage | $u \simeq +1$ |
| | *Case 1:* White has excellent winning chances | $v=0$ *and* $u \simeq +1$ |
| | *Case 2:* Although White's game is theoretically lost, he is almost bound to win | $v=-1$ *and* $u \simeq +1$ |
| | *Case 3:* An easy win for White | $v=+1$ *and* $u \simeq +1$ |
| (7) | A certain win for White | $v=+1$ *and* $u=+1$ |
| (8) | $s$ is difficult | $v \gg u$ |
| | *Case 1:* White needs accuracy to secure the draw | $v=0$ *and* $u \ll 0$ |
| | *Case 2:* White needs accuracy to secure the win | $v=+1$ *and* $0<u \ll 1$ |
| | *Case 3:* Although theoretically won, White's position is so difficult for him that he should offer a draw | $v=+1$ *and* $u<0$ |
| (9) | A losing move | $v(s_2)=-1$ *and* $v(s_1)>-1$ |
| (10) | An inaccuracy: White's move weakens his position | $\Delta v=0$ *and* $\Delta u<0$ |
| (11) | White's move strengthens his position | $\Delta v=0$ *and* $\Delta u>0$ |
| (12) ?, | A mistake | $\Delta v=-1$ *and* not $(\Delta u \ll 0)$ |
| (13) ??, | A blunder | $\Delta v<0$ *and* $\Delta u \ll 0$ |
| (14) !, | A strong move | $\Delta v=0$ *and* $\Delta u>0$ and s1 is difficult |
| (15) !!, | A very strong or brilliant move | $\Delta v=0$ *and* $\Delta u \gg 0$ |
| (16) !?, | A brilliant but unsound move | $\Delta v<0$ *and* $\Delta u \gg 0$ |
| (17) | Best move | $\Delta u$ is max |
| (18) (!), | Best move in difficult circumstances | $\Delta u$ is max *and* $s_1$ is difficult |
| (19) | A safe move | $\Delta v=0$ *and* $s_2$ is lifeless |
| (20) | 'White should press home his advantage.' The rationale for trying to shorten the game when ahead can be understood by noting in Fig. 3 how the advantage decays as we move backwards from the terminal positions. In Fig. 5 White, in moving from B1, has been given an additional option in the form of a move to C5·1, from which Black is forced to move directly to F9 (*S*-shaped arc in Fig. 5). Game theoretically the choice between moving to C5 and moving to C5·1 is equally balanced since they are both 'won' positions for White. But the expected utilities, +0·246 against +0·757, tell the true story, that is he incurs needless decay in a won position, especially if it is a complicated position (high branching ratio of immediately dependent tree), he multiplies his chances of error. Our model selects the move to C5·1 with 1·7 times the frequency of C5, with a corresponding increase of $u$(B1) (see Fig. 5). | |
| (21) | 'Black should play for time' is the complementary advice one should give to the *other* player in the foregoing situation. If our hypothetical node C5·1 had a second branch leading to D9 (shown as a broken line in Fig. 5), then Black should prefer it to F9. | |

*Table 3 Evaluative comments on positions (comments on moves are not shown here) corresponding to various combinations of expected utility, u, and game theoretic value, v.*

| | $v=-1$ | $v=0$ | $v=+1$ |
|---|---|---|---|
| **1.** $u=0$ | $s$ is virtually impossible (because of the unlikelihood that $u$ should be identically zero). | $s$ is a certain draw (dead draw). | $s$ is virtually impossible (because of the unlikelihood that $u$ should be identically zero). |
| **2.** $u=-1$ | $s$ is a certain win for Black. | $s$ is impossible. | $s$ is impossible. |
| **3.** $u=+1$ | $s$ is impossible. | $s$ is impossible | $s$ is a certain win for White. |
| **4.** $u \approx 0$ | White has excellent drawing chances. Black needs accuracy to ensure his win. | $s$ is a balanced position. | Black has excellent drawing chances. White needs accuracy to ensure his win. |
| **5.** $u \approx -1$ | An easy win for Black (decisive advantage). | Black has excellent winning chances. White needs accuracy to make sure of a draw. | White has a theoretical win but is almost bound to lose. |
| **6.** $u \approx +1$ | Black has a theoretical win but is almost bound to lose. | White has excellent winning chances. Black needs great accuracy to make sure of the draw. | An easy win for White (decisive advantage). |
| **7.** $-1$ h $u<0$ | Black has a mildly difficult win. | Black has a slight advantage. White needs care to make sure of the draw. | White needs extreme accuracy to make sure of of his win (a very difficult win for White). |
| **8.** $+1$ j $u>0$ | Black needs extreme accuracy to make sure of his win (a very difficult win for Black). | White has a slight advantage. Black needs care to make sure of the draw. | White has a mildly difficult win. |
| **9.** $-1<u$ h $0$ | Black has a clear advantage. | Black has good winning chances. White needs accuracy to make sure of the draw. | White has a theoretical win but is likely to lose. |
| **10.** $+1>u$ j $0$ | Black has a theoretical win but is likely to lose. | White has good winning chances. Black needs accuracy to make sure of the draw. | White has a clear advantage. |

variance of the distribution of $u$ values over the dependent sub-tree, and hence to the measure which we develop below for the 'tension' of a position. The former British Champion, Alexander, uses this term in an introductory chapter to *Fischer v. Spassky, Reykjavik 1972.* Alexander (1972) writes (see Fig. 6).

'Let me illustrate (a little crudely) this question of tension by comparing two openings:

A. (Giuoco Pianissimo) 1 e2-e4 e7-e5 2 ♘g1-f3 ♘b8-c6 3 ♗f1-c4 ♗f8-c5 4 d2-d3 d7-d6 5 ♘b1-c3 ♘g8-f6.

B. (Grünfeld Defence) See the Siegen game Spassky v. Fischer) 1 d2-d4 ♘g8-f6 2 c2-c4 g7-g6 3 ♘b1-c3 d7-d5 4 c4×d5 ♘f6×d5 5 e2-e4 ♘d5×c3 6 b2×c3 ♗f8-g7 7 ♗f1-c4 c7-c5.

The moves in example A are perfectly correct — but after five moves the game is as dead as mutton; it is too simple, too balanced, and is almost certain to lead to an early and dull draw. The moves in example B are objectively no better — but the position is full of tension; White has a powerful pawn centre but Black can exert pressure on it, and, if he survives the middle game, may stand better in the ending — the players are already committed to a difficult and complex struggle in which a draw is not very likely'.

A simple way of capturing the spirit of Alexander's definition within the framework

Giuoco Pianissimo

Grünfeld defence

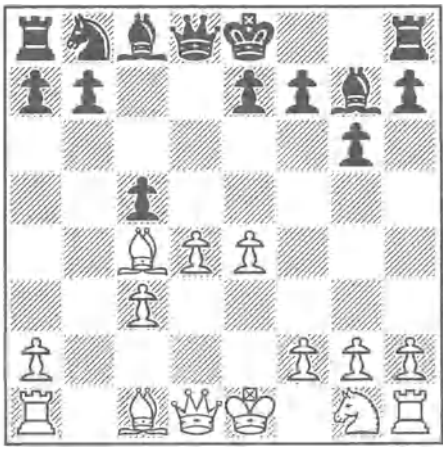


*Fig. 6 Positions of low and high 'tension' (from Alexander, 1972)*

of our theory is to use the *weighted mean square* of the terminal values of the tree rooted in $s$, i.e.

$$\mathrm{var}(v_t)=\sum_{t \in T} P_t V_t^2$$

where $T$ is the set of terminal positions and $p_t$ is the probability of arriving at the $t$th member of this set starting at $s$. A value of unity corresponds to maximal tension and a zero value to minimal tension (the latter can only be attained by a 'dead draw'). The tension of the root node of Fig. 3 is estimated by this method at 0·559. Referring to comment No. (3) above we assign this root node to Case 2 rather than to Case 1 of the category 'balanced'. Note that although 'tension' is calculated from game-theoretic values, $v_t$, use is made of the $u_t$s in the calculation of the probabilities, $p_t$, and hence the measure is affected by variations of the merit parameters $M_W$ and $M_B$. As soon as we postulate greater playing strength on the part of White some of the tension of the position is reduced. The tension of node A in Fig. 4 is only 0·024, reflecting the fact that the Black is almost certain to steer play into the 'dead draw' sub-tree.

Note that $\sum_{t \in T} P_t V_t^2$ is equal simply to the probability of a non-drawn outcome. But we have preferred to formulate the expression explicitly as a variance, since in realistic cases game-theoretic values are not likely to be available, or calculable in practice. The approximating formula $\sum_{t \in U} P_t y_t^2$ may then prove useful, where the $y_t$s have been assigned by some evaluation function (or by human intuition) to the members of $U$, the set of states on the lookahead horizon.

## Summary of Ideas So Far

We have extended the strict game-theoretic model of chess which assigns to board positions only three values: +1, 0 and −1. A good model should do justice to the profusion of chess commentators' evaluations. Specimen evaluative comments have been displayed as benchmarks against which to assess the extended theory. We have illustrated with worked examples a simple model based on the notions of utility and statistical expectation. Our model finds no particular difficulty in explicating the specimen evaluative comments. It also reduces to the game-theoretic model in the special case of error-free play.

## Application to Computer Chess

A worthwhile study would be to explore parts of a non-trivial sub-game of chess of which complete game-theoretic knowledge exists, as in ♔ + ♘ versus ♔ + ♖ (Bratko and Michie, 1980; Kopec and Niblett, 1980). The program's own comments on sample endgame play could be compared with the intuitions of experienced players.

A more satisfying use of the model would be for generating computer play. The

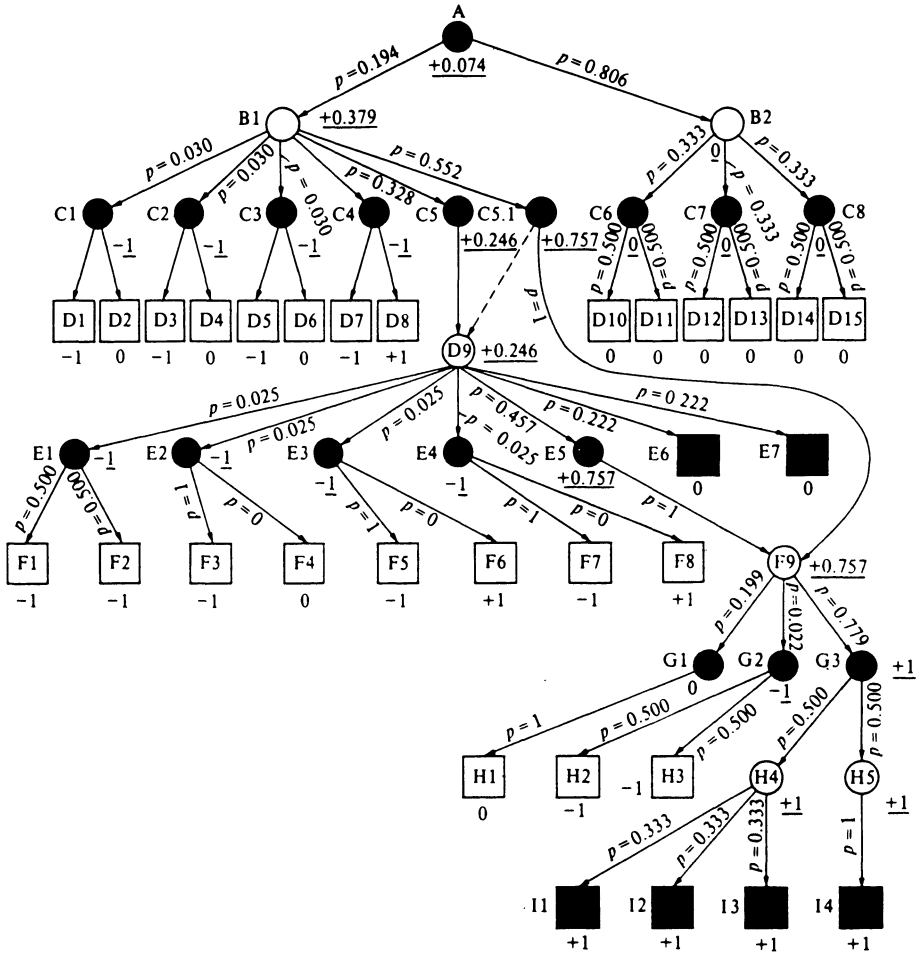



*Fig. 5 A modified version of Figure 3 in which a new node, C5-1, has been added leading to F9 (the broken line represents a hypothetical delaying move for Black, see text). Although without effect on the game-theoretic values of nodes lying above it in the tree, interpolation of this short-cut option tips the balance of expected utilities, so that at the root the move to B2 becomes 'best'.*

procedure exhibited earlier for calculating scores by backwards iteration from the terminal nodes of the game tree was derived from classical decision theory. State of the art tournament programs also use 'backed-up' scores and they base move-selection on them. But they follow the mini-max model. Might not such programs benefit from using expected utilities rather than minimax? After all, the near-universal adoption of the minimax rule in computer game-playing rests on no theoretical foundation.*

Minimaxing for purposes of computer play originated from the intuitions of Shannon (1950) and Turing (1953), neither of whom offered explicit justification. The rule's empirical record has done the rest.

When lookahead is conducted to the end of the game, the validity of minimaxing rests on its built-in guarantee against selecting a game-theoretically 'losing move'. The reader can remind himself of this by inspecting Fig. 2: the constant-value subtree rooted in a given node defines a value-preserving strategy for all play ensuing from that node, provided that we have some rule for tie-breaking among a node's equivalued successors. But Fig. 3 shows that against a *fallible* opponent, this concept of 'validity'

*Beal and Bratko (1983) have, however, recently established a sufficient condition.

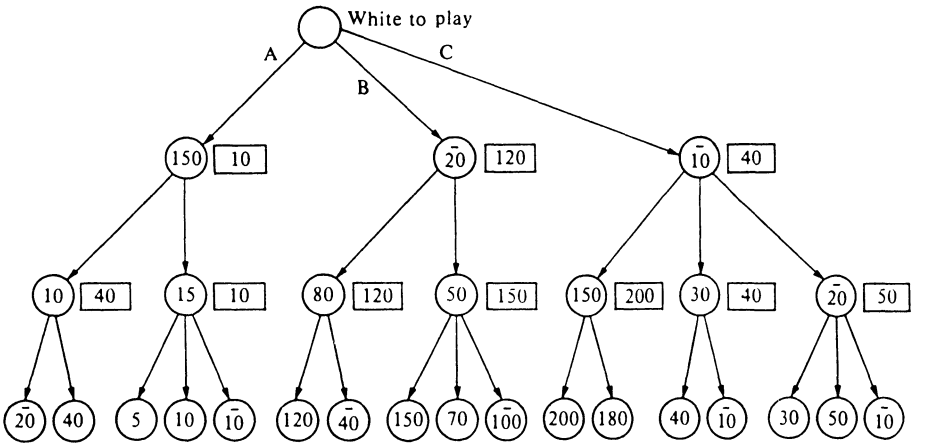



*Fig. 7 Terminal positions are shown as boxes in this lookahead tree, in which the nodes are marked with 'face scores' (bars over negative). Boxed figures are values backed up from the lookahead horizon. If move-selection were decided by face scores then move A would be chosen, but if backed-up scores then move B. What is the rationale for B?*

is harmful, for here a 'losing move' is Black's decision-theoretically best choice.

A further difficulty arises when computational resources do not permit complete lookahead. For this Shannon and Turing independently prescribed that the program should look ahead to some limited depth, and then assign to the terminal nodes of the lookahead tree estimates of their game-theoretic values supplied by an 'evaluation function' — typically a linear combination of terms corresponding to measurable features of the position (piece advantage, mobility, etc). These scores are then backed up by the minimax rule to the current position's immediate successors, in place of the desired but inaccessible game-theoretic values. The rule of play selects the successor with the most favourable backed-up score (move B in Fig. 7).

Except in the (unrealistic and uninteresting) case that the evaluation function approximates the game-theoretic value so closely that the decisions given by the rule are invariant with respect to depth of lookahead, this rule has lacked formal justification.* We are thus free to attribute its empirical success to the fact that it can be regarded as an approximation to a decision-theoretically correct rule of the kind developed earlier. Note that the larger are the values of $M_W$ and $M_B$, the closer is the approximation; in the limit the two models coincide.

The new model raises a point of particular relevance to the present situation in computer chess. Fast, partly parallel, special-purpose chess machines have recently been developed and interfaced to powerful computers (see for example Moussouris *et al.*, 1979). Chess programs of conventional type interfaced to such machines become capable of searching to an average depth in excess of 9-ply, almost twice that attained by chess masters (see de Groot, 1965; note that we are speaking of the average length of the longest branch of the lookahead tree). To give such a machine the best chances it should be endowed with a 'hunger for complexity'. The idea must be continually to drive for high tension positions avoiding simplifying exchanges where possible. In this way cognitive strain on the human player is intensified by the need for vigilance against tactical traps which may lie $\geq$ 9-ply deep. Such a policy calls for a model incorporating opponent fallibility.

*Beal and Bratko (1983) have, however, recently established a sufficient condition.

## Concluding Remarks

An objection to the theory here developed is that the opponent model is arbitrary. Two comments are in order.

(1) It is of no theoretical consequence what particular opponent model is used for illustration, provided only that it has the right overall properties. The reader is free to use the theory with any opponent model he pleases.

(2) No choice of opponent model is as arbitrary, or as inflexible, as minimax. Moreover, even on the basis of complete lookahead to the end of the game, minimax back-up does not yield the best strategy against a fallible opponent.

## Acknowledgements

Part of this work was done during two months study leave granted to me by the University of Edinburgh. An early version was communicated at the 1974 Summer Conference of the Society for Artificial Intelligence and the Simulation of Behaviour.

# 5. Analysis

There is some overlap between the content of this chapter and that of chapter 4. Here we start with an excerpt from Adriaan de Groot's famous book *Thought and Choice in Chess* (section 5.1). De Groot's work was based on the study of human chess masters' "protocols", in other words, the masters spoke their thoughts out loud while analyzing a chess position and De Groot was able to monitor their thought processes. Herbert Simon and William Chase have studied the perception of human chess players in a different way, and their work is described in section 5.2.

Mikhail Botvinnik was World Chess Champion from 1948-60 (with a brief gap) and from 1961-63. Since the 1960s he has been involved in the design of a chess program which is unique in the way that it attempts to imitate the thought processes of a human Grandmaster. His paper (section 5.3) gives an excellent insight into his ideas on chess programming, illustrated by an example of analysis by his program PIONEER. Botvinnik's program is able to find the brilliant, deep winning combination from a famous Botvinnik-Capablanca game, an idea which is inaccessible to today's "brute force" chess programs.

George Baylor and Herbert Simon describe (section 5.4) a program for finding mating combinations where the search is highly constrained by the fact that the defending player is always in check or under threat of mate next move. This paper may be considered alongside Larry Harris' work presented in the previous chapter – the Baylor and Simon paper is a specific example of the (later) heuristic search method described by Harris.

The chapter ends with an early paper devoted to the solution of chess problems of the "White to play and mate in two moves" variety. Prinz' work (section 5.5) is presented here purely for historic reasons – it is referred to by Turing and was one of the very first attempts to write a genuine computer program (as opposed to a hand simulation) for any aspect of the game.

# 5.1 The Sequence of Phases

**Adriaan de Groot**

Originally published as Part A of Chapter VII (The Organization of Thought Processes), of *Thought and Choice in Chess*, by A. D. de Groot, Mouton, The Hague, 1965, pp. 254-280. Reprinted by permission of Mouton Publishers.

**Section 50: The principle of interaction**

In Chapter IV we came to find the alternation of elaborative and integrative phases as the basic structure of the thought process. In the light of the organization and methodology of the thought process this means that *the subject (player) periodically returns to more general problems, especially to the main problem.* The structure of all thought processes in which a difficult problem must be dealt with is characterized by this alternation.

In Section 44 we saw that the alternation of phases corresponds to an *interaction between elaboration and total goal (problem) conception.* 'During an elaborative phase the problem in its present state of development determines the course of the thought process via the schematic anticipation contained in the goal awareness; conversely, in the transitional phases the results of the elaborations cause a more or less drastic transformation of the problem' (p.184).

In many respects the processes involved in this interaction have already been analyzed in the previous pages. The influence in one direction – the results of elaborations on the total goal conception – was discussed in Chapter VI, while in a sense Chapters IV-VI form one continuous plea for the importance of the influence in the other direction. In particular, there has been much emphasis on how the moments or components of the total problem conception, via the schematic anticipation, influence the direction and methodology of the ensuing elaborations. Type, core problem, estimated value, each plan, each goal-setting and problem formulation, each solution proposal that is more than a blind stab, each alternative grouping, each shade of favoritism, each hunch about the value of the position or of a variant, or of a possibility, each hunch about necessity or desirability of certain actions, etc. – all of these components of the total goal conception may, at any moment, exert a decisive influence on the course of the thought process. The total goal conception also contains the subject's appreciation of the board problems at a given moment (cf. Section 45) which in its turn is often of decisive importance for the ensuing thought operations inasmuch as operations-goals follow from board goals.

From the protocols we can easily infer the following relationship between components of the total goal conception – anticipations in particular – and ensuing thought activities:

1. Via the schematic anticipation(s) in which the components of the total goal conception become operative, the components determine[1] the general direction of the investigation, especially *the selection of moves and plans to be considered.*

2. The components of the total goal conception, in particular the subject's appreciation of the board goal and board problems, determine – often in considerable detail – the specific (board) means that are supposed to be adequate: considerable strategies and tactics. In terms of the investigation of possibilities, this means that the selective influence of the subject's board goal conception is not restricted to immediate solving propositions in the present position but extends

1 The question whether or not the immediately following operations are *completely* determined by the components of the total goal conception – as would correspond to Selz's conception of thought dynamics – will be deferred until Chapter IX.

to *the selection of considerable continuations in envisaged positions.*

3. At a certain point in the thought process the components of the total goal conception determine to a large extent *the manner in which the following investigations are to be set up:* whether the subject will explore rather neutrally, try to build up a preference, try to maintain a favorite (deeper investigation), or try to prove the superiority of a nearly chosen move, either by recapitulating its merits or by final strengthenings and/or rejections of alternatives (positive and negative proof).

4. Further, the components play an important role in the transitions to new variations, moves, or plans. In particular, the *decisions to reject or accept* (temporarily) certain solution proposals or possible continuations appear to be based on the relation between quantitative outcomes and quantitative anticipations. Both the mathematical sign and the magnitude of the difference between the subject's current expectancy and the quantitative results of his investigation of a proposed solution are of decisive importance for what will happen to the proposal.[2]

5. The same holds for *the detection as well as rectification of errors, weak continuations, and insufficient solution proposals.* Whenever the goodness or badness of a partial result seems 'improbable' it is distrusted: an improvement for the underdog is sought and most times found. It is clear that a result can only be judged 'improbable' if a pre-set quantitative anticipation exists — either the subject's general expectancy or a specific expectancy for a specific variation. The mathematical sign of the difference between anticipated and calculated result determines the direction in which an improvement is sought, while the magnitude of the difference is of importance for the subject's determination and/or thoroughness in his search for improvements. If a solution proposal (move) is found at fault in this way, an immediate re-investigation will follow — or finally a rejection (cf. point 4 above). Again, the exhaustiveness of the re-investigation is related to the magnitude of the difference.

In the foregoing chapters we have given so many instances of the influence of various anticipations on the ensuing elaborative operations that a general review of the individual components of the mechanism of this influence appears superfluous. In the rest of this section, therefore, the analysis of the interaction will be restricted to the quantitative moment, i.e., to the remarkable regulatory function that the estimated value, the maximal (and minimal) expectancy, and the more specific quantitative expectations appear to wield in chess thinking.

In the game of chess, in its present form at any rate, the quantitative argument is always the decisive force. Perhaps it would be nobler to defeat one's opponent in a risk ridden attack on the King rather than to win smoothly in an uninteresting endgame that follows an exchange of Queens, yet the modern master will elect to exchange Queens if this course leads to a clearer and greater *advantage.* The manner of playing, the character of the position, of the plan of action, and of the entire game are but of secondary importance compared to the advantage or disadvantage derived from the quantitative reckoning. One need only look at the way in which some insipid chess columnists and analysts annotate games: they talk only of advantages and disadvantages, errors, less strong moves, the best move, a somewhat better position, etc., without ever mentioning the qualitative elements. Although such analyses are certainly not the most fascinating, the mere fact that a chess game can be treated in such a fashion attests to the importance of the quantitative.

In the protocols, too, the quantitative predominates as, for example, in the formulation of results. Indeed, obtaining an 'advantage' or avoiding a 'disadvantage' is the only objective and overriding motive. Wilhelm Steinitz, who laid the foundation of the modern positional game, speaks of the 'accumulation of small advantages' as the aim in a positional game. (Steinitz 1889 and 1895). When the player supplants one qualitative goal (plan) by another, the character of the game changes drastically, but the more important quantitative relationships remain the same — if both sides

2 The importance, from the point of view of machine simulation of human thought, of the ever present *pre-set expectancy* — a major component of the subject's total goal conception — will also be discussed in Chapter IX.

play well. Especially during the transition from the middle game to the endgame – but by no means exclusively at this stage of a game – one frequently finds that 'advantages' of one type are traded in for advantages of another type.

From all this it should be easy to understand the important regulatory function fulfilled by the quantitative anticipations in the thought process. The following instances are taken from protocol ($M_3$; A) – see below.

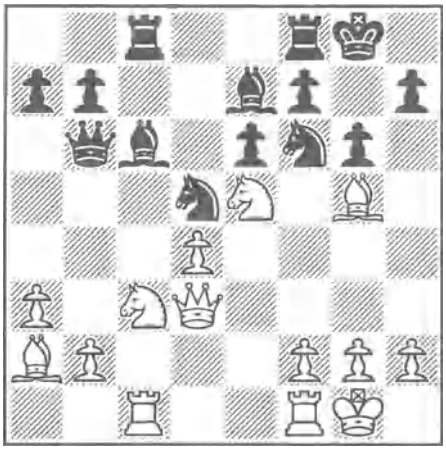
At the beginning of ($M_3$; A), after some ten seconds, the estimated value is formulated (line 1): 'White's position is superior in any case.' Expressed in a rating: estimated value = expectancy = 8. But the position is complicated and contains combinatory possibilities. For that reason there remains a wide margin of uncertainty; it could even be that the position is won. In other words: Subject $M_3$ can tentatively set his maximal expectancy at 9 or 10. Thus he directs his search towards a decision. This maximal expectancy is now the operative quantitative anticipation.

The quantitative goal determines the direction of the investigation (point 1, p. 255) as well as the considerable means (point 2). Only forceful moves offer a chance of success, i.e., of attaining 9 or 10. More important, however, is the fact that *the results of the variations are measured up to the maximal expectancy.* Every result is rejected that does not stand the (rigorous, self-imposed, anticipatory) test. The outcome of this (relative ) measurement is decisive for the 'success' or 'failure' of the attempted solution and hence for the next linking (whether it will be cumulative or subsidiary). The maximal expectancy thus regulates to a large extent the sequence of operations; it helps to regulate the entire methodology involved in rejecting and accepting solution proposals[3] (point 4).

3 This regulatory function is characteristic for actual games where the maximal expectancy varies – not so in puzzles, chess problems, or studies where it is fixed (e.g., White to play and win).

**(M3; position A) Subject: S. Landau**
**Experimenter: de Groot**
**Time = 15 minutes.**
**Test conducted December 8, 1939**

DE GROOT – SCHOLTENS, DUTCH CH. PRELIMS. 1936. Position A after 16 ... ♕d8-b6?



*Position A*

Let's have a look. White's position is superior in any case. To search for a combination in connection with f6 and d5.

To figure out 5

1 ♗a2×d5 with possibly ♘e5-d7 afterwards, in case Black takes back with Bishop. 1 ... ♗c6×d5, then later ♘e5-d7; so 1 ... e6×d5 is forced; but one does not get very far with that either. 10

Let's look at

1 ♘e5×c6. Also important. Calculate 2 ♘c3×d5 to see if that provides any direct advantage. 1 ♘e5×c6 ♖c8×c6 2 ♘c3×d5 ♘f6×d5 3 ♗a2×d5 ♖c6×c1 probably leads to nothing. 4 ♖f1×c1 ♗e7×g5 15 but wait a moment, also 4 ♗g5×c1 is possible. What then? Let's have another look:

1 ♘e5×c6 ♖c8×c6 2 ♘c3×d5 ♘f6×d5 20 3 ♗a2×d5 ♖c6×c1. Now something else: 4 ♗g5×e7 ♖c1×f1+ 5 ♕d3×f1 ♖f8-e8 6 ♗e7-c5 probably to the advantage of White. But it can be played otherwise: 5 ... e6×d5 6 ♗e7×f8 – haven't achieved 25 much then – leads indeed to nothing. Look for something else:

1 ♘e5×c6 ♖c8×c6 2 ♘c3×d5 ♘f6×d5. Now something else? No, doesn't yield much. 29

30 1 ♘c3×d5 ♘f6×d5; what then? 2 ♗g5×e7 ♘d5×e7 – nothing. There is no decisive combination. Then maybe an ordinary attacking move:

1. h2-h4 for instance; but what then
35 on 1 ... ♕b6×b2, is there any compensation for the pawn? Probably not.

1. ♗g5-h6 maybe. See if anything comes out of that. (Short pause)

40 Still I keep looking at taking on d5, if there isn't something in it. 1 ♘c3×d5 or

1 ♗a2×d5. If 1 ♗a2×d5 then 1 ... ♘f6×d5 is impossible. 1 ... ♗c6×d5 would be forced, therefore – but that has its drawbacks
45 too: loss of the exchange. Yes,

1 ♗a2×d5 is the move. With that White gets the advantage. 1 ... e6×d5 is forced, and then a favourable position is reached. Play 2 f2-f4 for instance. Maybe we can
50 get even more out of it.

1 ♗a2×d5 e6×d5; 2 ♘e5×c6 to win a pawn possibly? No, that doesn't work. Let's look at 2 ♕d3-f3. Then 2 ... ♕b6-d8 is forced. What then?

55 1 ♗a2×d5 e6×d5. A combination with 2 ♗g5×f6 ♗e7×f6 3 take on d5 and ♘e5-d7 – insufficient.

1 ♗a2×d5 followed by 2 f2-f4 is probably the best. Maybe a better con-
60 tinuation? Yet possible perhaps to get material advantage? Search for something.

Once again:

1 ♘c3×d5 ♘f6×d5 2 ♗g5×e7 ♘d5×e7.
65 Does that yield anything? No, better is

1 ♗a2×d5 e6×d5; and now? What further? In any case:

68 1 ♗a2×d5 is good.

This regulation becomes especially clear as the maximal expectancy fluctuates during the course of the problem development. ($M_3$; A) remains the case in point: Gradually negative deviations undermine the maximal expectancy (interaction) until finally, in line 31, the conclusion is 'There is no decisive combination.' The margin of uncertainty surrounding the estimated value is reduced by the fact that the maximal expectancy drops. As an immediate consequence, other moves now come into consideration: 'Then maybe an ordinary attacking move' (points 1 and 2). But these ordinary attacking moves (1 h2-h4 and 1 ♗g5-h6) do not satisfy the expectancy either, and $M_3$ says, 'Still I keep looking at taking on d5, if there isn't something in it.' But now he does so with somewhat tempered expectations. Curiously, the ensuing re-investigation of 1. ♗a2×d5 ends up with the same variant of which he said in line 9: 'but *one doesn't get very far* with that either.' Now he avers: 1 ♗a2×d5 *is the move.* With that White gets the advantage' (line 46). This example once again proves that formulations of results must be taken relatively.

*Are* there also 'absolute' results? Are there results which are just unfavorable and unpleasant in an absolute sense, which 'do not go'? Does it occur that results are unequivocally and absolutely rejected, or that they are absolutely favorable, advantageous, or pleasant so that they are unequivocally accepted? The absolute conception would correspond to a sort of 'trial and error' theory of thinking as proposed by, e.g., Thomson (1924). Under the relative conception what is decisive for the success of a thought operation is not the (absolute amount of) 'pleasure' created by the result but the *agreement* of the result with the (quantitative ) anticipation – in line with Selz's theory. For chess thinking we must concede that *both* cases occur. For clear-cut cases – the outright winning or losing results, such as checkmate, winning a piece, etc. – there is no need to evaluate in relation to a maximal expectancy. Since most quantitative results are not so extreme, however, results are generally 'measured' against quantitative expectations.

A psychologically interesting question is whether we must think of this process as one of 'comparing' the two values, expectancy and outcome. Such a description does not appear adequate though: it is too intellectualistic. We can rather say that the subject is analyzing variations with an anticipated *degree or level of satisfaction* in his mind. In fact, subjects often express themselves in terms of feelings, such as 'not (completely) *satisfactory*,' or that does not *appeal* to me either,' especially when the difference between expectancy and outcome is slight. The abundance of terms, terms like '(un)pleasant,' 'nice,' (un)satisfactory,' 'appealing,' etc., speak for an interpretation of the expectancy as a (temporary) level of *aspiration*, or in Lewin's (1926, transl. 1951) terminology: as a

quantitatively measured 'quasi-need' that must be fulfilled. Even when towards the final Phase of the thought process quantitative results are no longer measured against expectations but are actually compared to each other, the psychological process is not just a matter of intellectual comparison of quasinumerical values. It is rather 'satisfaction (or pleasure) values' that are compared.: 'I *like* that better,' This *appeals* more to me,' etc.

We return now to the regulatory function of the quantitative expectations. There are still a few complications to be mentioned.

First, as appears from points 4 and 5 above, negative deviations from the expectancy lead to *rejection* of a solving proposition only after several attempts at strengthening have been undertaken: minor corrections, variant reinforcements, re-investigations (cf. Section 51). Rejection is a last resort which is only called upon when the means of strengthening run dry. Exactly where the 'exhaustion point' will be reached depends – apart from the objective degree of exhaustion which could be defined by an objective, expert analysis of the solving proposition in question – on the phase of deepening of the thought process, i.e., on the state of development of the subject's total goal conception. During an early, still exploratory investigation the exhaustion point will be reached much sooner, *ceteris paribus*, than during deeper investigation near the end of the thought process (cf. Section 52).

Second, positive deviations from the expectancy hardly ever lead to immediate *acceptance* of a solving proposition or variant. Again, attempts at improving the opponent's counterplay (broadening), checking operations, re-investigations, and, particularly in the final Phase, a recapitulation generally precede the decision to accept (the validity of) the possibility. Moreover, before the decision is taken, the (maximal) expectancy goes up – possibly beyond the level that was attained: 'Maybe we can get even more out of it.' In any case, at the moment the move decision is made there is no longer any positive deviation but rather *agreement* between anticipated and estimated result. This is in line with Selz's conception: in its quantitative component the solution corresponds to the schematic anticipation in the mind of the subject.

Note again the regulatory function of the quantitative expectations in the process. One result of this regulation is the serious consequences that a wrong appraisal of a position may have. If the expectancy is too low, for example, the upshot may be an inadequate search for improvement possibilities, in this case in the subject's own play. As a result the stronger variants are not likely to be discovered. In practice this is a common source of weak play: the player overlooks a winning, favorable, or sufficient defensive move because he does not realize that the position is as good as it is.

Protocol ($M_3$; A) provides – as does practically every other protocol for that matter – a few examples of expectancy-induced checking investigations and attempts at strengthening. Thus the thorough investigation of 1 ♘e5×c6 (lines 20-26) renders a negative outcome, relative to the maximal expectancy ('haven't achieved much then; leads indeed to nothing') which first elicits an attempt at strengthening:

27 Look for something else: 1 ♘e5×c6 ♖c8×c6 2 ♘c3×d5 ♘f6×d5. Now something else? No, doesn't yield much.

It appears that now the 'exhaustion point' is reached for immediately afterwards another move has its turn: 1 ♘c3×d5, etc.

The fact that the result of the elaboration is at variance with the expectancy thus induces a search for improvement, the discrepancy being *a signal that something is probably amiss*. In the following fragment from another protocol this specific anticipation is explicitly expressed:

($M_4$; ... ♕e7):

75 Oh, no, the Queen takes back on d7 and the Rook on b5 hangs. *But that can probably be improved upon.* Yes, first 4 ... ♕g5

The great practical importance of this inducement to checking is indirectedly demonstrated by the observation that omissions and calculational errors tend to remain undiscovered especially in situations where a discrepancy is lacking! This occurs, for example, when in a calculation the subject is unlucky enough to choose a weak move for both sides. The end result

may still agree with the expectancy because the errors cancel each other out.

Striking examples of cancellation are found in protocols ($M_2$; C) and ($G_4$; A). Both subjects overlook a staightforward defense in an investigated continuation and thus get involved in the calculation of a number of lengthy variations that are completely irrelevant to an objective analysis.

The conclusion should *not* be, however, that these calculations are worthless for the thought process. They always retain some value as sample variations. They illustrate the possibilities of the position and can also have heuristic value. In ($G_4$; A) subject G4 via the calculations 1 ♖f1-e1 ♕b6×b2? 2 ♗a2×d5? (2 ♘e5-c4 wins a piece, even after 2 ... ♗c6-b5!) finally arrives at the correct solution 1 ♗a2×d5.

Overlooking such errors and omissions need not have an adverse influence on the choice of move; indeed this seldom happened in the experiments – at least not with the master subjects. The primary function of a sample calculation is often *to sharpen and test the current quantitative and qualitative anticipations.* But if the intuitive appraisal is reasonably accurate a priori (as it mostly is with master subjects) spurious verification cannot do much harm.

Apart from the expectancy, i.e., the overall appraisal of one's chances, other more specific anticipations, both quantitative and qualitative, can have a regulatory function.

For each it is possible to trace the mechanism of interaction. A few specimens must suffice, however.

The productive significance of the *anticipation of urgency* is quite clear. It is found in particular in the master protocols of position B. The subject would not work so hard on the King's side attack, plan A, without the 'impression that I must proceed actively' ($G_5$; $B_1$), line 62, or without the hunch: 'If I do nothing, I have the idea that I am gradually going to lose' ($M_2$; $B_2$), line 8. The same holds for ($M_5$; ♖d1): 'Something must happen immediately; if Black can exchange pieces, White won't win so easily' (line 85). Here as in ($M_2$; B) it is the unsatisfactory results of the search for calmer moves which intensify the feeling of having to hurry; another example of the converse influence in the interaction process; calculational results modifying the anticipations and therewith the subject's total problem conception.

The *anticipation of (objective) solvability* is another case in point. Like the anticipation of urgency it may have a productive influence on the energy expended in investigating in a certain direction; for example ($M_5$; ♗h7+): 'I've a distinct feeling that it's somehow or other over' (line 153).

An *intuitive faith in a certain possibility* may have the same effect ('intuitive' in the sense of: based on evidence that is experienced as insufficient). In addition, such anticipations – as for that matter any specific expectation – often have an important influence on the setup of the investigation (point 3); for example, the order in which the variants and moves are investigated. In trying out sample variations one first examines the *putative* main variation; conversely, in calculating the branches of the proof one begins 'with the *probably* bad countermoves – in order to be able to eliminate those' ($M_1$; A)*, line 14. Again in both cases it is obviously the hunches and expectations – thus anticipations – which help to regulate the course of things.

## Section 51: The hierarchy of subsidiary[4] methods

In the analysis of thought processes certain typical sequences appear to support the hypothesis of a constant *hierarchy* of subsidiarily linked methods (or 'program') in the subject's mind. Selz's investigations led him to the following general law (cf. Section 20): If a solution attempt fails, the same method is again applied but with new material; a new method is only applied when the available material is exhausted. When all available methods are exhaused a 'delay-until-later' follows – itself a method of thought (cf. Section 33). In terms of chess: If a calculation leads to unsatisfying results, other *variations* are tried out (attempts at strengthening) while the same plan and first move are retained; if the variations fail, other *first moves* are tried out; if these also fail the entire direction of the investigation is changed (another *plan*).

4 It must again be stressed that it is the etymological meaning of 'subsidiary' that is availed here, i.e., the notion is of a reserve supply, one free from an inferior or subordinate status or capacity. (From Latin *subsidium*, the troops stationed in reserve in the third line of battle.)

There was just such a development in ($M_3$; A); cf. Section 50.

However, except for the possibility of a provisional delay in the investigation of a certain plan or first move (non-immediate re-investigation, cf. Section 42; progressive deepening, cf. Section 52) the method delay-until-later, that is, delay the solution of the main problem, cannot be applied by the chess player since he has to decide on a move. We have seen for the case of ($M_3$; A) what happens instead: there is a change of direction again (another plan or move), but this change is coupled with *a drop in the maximal expectancy.* Such a drop occurs in many other protocols as well, particularly in the later stages of the thought process when all available plans fail to satisfy the quantitative expectations. Indeed, this is again related to the important distinction between an assigned problem in which a 'right' solution must be found and the choice-of-move-problem in chess in which *the player himself defines the requirements of a solution.* In the former, in which a quantitative goal is given, the subject *must* 'delay-until-later' when he cannot find a solution; in the latter, when the problem solver (chess player) is stuck, he has another recourse: simply lower the quantitative goal (expectations).

Aside from the repeated appearance of certain successions of operations, the existence of hierarchical subsidiary linkings may appear from the formulation of operations-goal statements in the protocols. In particular, the typical expression 'Let's *first* investigate (try out, look at, etc.) ...' implies – at least if a future *subsidiary* linking is actually meant – the existence of a priority that may result from a general, methodical hierarchy. In favorable tactical positions such as position A, for instance, the subjects generally investigate forceful moves first; they say 'Let's first have a look at what can be taken,' ($G_1$; A) or 'Now, let's look in some more detail at the possibilities for exchange' ($G_5$; A). The investigation of other types of moves: non-forceful, non-taking, non-exchanging moves only takes place if the former category becomes barren. The priority of forceful over calm moves is so general, indeed, that we have every reason to suppose that a fixed hierarchical order of subsidiary methods underlies the systems of methods (programs) of all subjects. Analogously, other types of subsidiary linkings in the protocols suggest the existence of hierarchical orderings that all experienced subjects have in common.

In the following *hierarchy of subsidiary methods* all presumably general 'priority rules' that could be derived from the analysis of the entire experimental material are summarized.

Assume that the subject has just calculated a certain variation of a move possibility that fits into a plan and that this variation yields a (quantitatively) unsatisfactory result. What happens now? Which types of operations can replace the one that failed? Which priority rules obtain?

1. The simplest and the first possibility which the subject may consider is the investigation of another own-variant of the same continuation; that is, an *attempt at strengthening.* This attempt is not (immediate) re-investigation but simply continued investigation.[5]

2. If such clear strengthening possibilities are absent, the subject will turn to the investigation of a different first move; if, in the previously generated *backlog*[6] *of moves* implementing the same plan, there is one move which was already subsidiarily linked (considered as a second or substitute) to the rejected one, this move has highest priority.

3. If there is no (longer a) backlog, then certain *qualitative results* of the prior investigation may have led to or now *lead to the discovery* of a different first move (or moves) within the same plan.[7]

4. If no more first moves present themselves, then the subject may consciously *search for another first move* in the transitional phase: 'Let us look around for

5 With regard to attempts at strengthening it is somewhat disputable whether the linking is really subsidiary. If there is any difference with *Selz's* (conditionally) composite solving methods of 'corrective completion,' where a *subordinate* method (correction) is supposed to be cumulatively linked to certain types of 'incomplete' – instead of unsatisfactory partial results, the difference is slight. The reader is referred to Section 20, p.72, and to Chapter IX, where this ambiguity along with others is more fully discussed.

6 Definition: A *backlog* is any sustaining reserve.

other possibilities' ($M_2$; A) line 23.

5. If the moves within the framework of the plan have been (temporarily) used up without having yielded satisfactory results, then there frequently follows a *re-investigation* of the move with the smallest negative deviation from the expectancy.

Here, too, we may speak of attempts at strengthening (as under 1) but now in the form of a re-investigation, either immediate or non-immediate.

6. If such attempts at strengthening are not possible or have already failed, then the subject will relinquish the plan and fall back on his *backlog of plans*. This transition may be attended by a drop in the quantitative expectations, or at least of the maximal exceptancy. From here on (7-10) a drop is likely to occur.

7. If there is no longer a backlog, then an alternative plan may be sought that is a *modification of the previous plan.*

8. If this does not work, the subject searches for basically *different ideas* (a new plan).

9. If no new conceptions of the solution emerge and if other plans have already been used up without having yielded satisfactory results, then, after a short problem analysis (transitional phase), the subject resorts to a *re-investigation of* the relatively best *plan.*

10. If such a re-investigation of one or more plans already took place, possibly via several phases of deepening, then a more penetrating *conflict analysis* certainly ensues in a lengthy transitional phase and bears the characteristics of a *crisis of expectations.* Finally, the subject either decides upon an even deeper and more systematic reinvestigation, or at least upon a recapitulation of the results obtained. On this basis the choice then has to be made.

This hierarchy is certainly not complete, but it does contain the most important types of subsidiary linking ranked according to priority of applicability.[8] Of course there is no protocol that contains all ten one after another; many steps are skipped in an actual sequence of operations. Indeed, the hierarchy is only meant to provide the framework from which appropriate subsidiary methods can be *successively* drawn. Exactly which methods and how many of them the subject will select depends on at least three factors: (1) the position itself (objective board problem); (2) the subject's individual preferences, techniques, insights, etc. (i.e., his personal program, cf. Section 59); and (3) how far the subject has progressed in the problem (the current content of his total goal conception).

There is one other restriction. The hierarchy is only valid in problem situations where the subject (player) is still *searching* for a solution, for a good move. If he is in the stage of striving for proof, then the hierarchy is of little avail. When the whole purpose of working out a move possibility is its elimination (negative part of the proof), then there is an entirely different relationship between success and failure, between cumulative and subsidiary linking. A bad result is a success since the elimination of a possibility has succeeded. Whereas in the Phase of searching and investigating the subject may resort to an attempt at strengthening (method 1) after a 'bad result,' such an own-branching in the Phase of striving for proof is now a cumulatively linked attempt to complete the negative part of the proof. Likewise, the investigation of a move from the backlog (2) or one just discovered (3) now corresponds to the broadening aimed at including several moves in the negative part of the

7 The order of priority of subsidiary methods 2 and 3 may be reversed – depending, among other things, on the state of development of the problem (exploratory versus investigatory and proof calculations) and on the type of position. In a still rather exploratory analysis of a tactically complex positon, for instance, the backlog, if any, cannot be expected to cover all relevant forceful moves, so the subject generates the new move(s) as a modification of the previously tried one(s). Method 2 may then be used at a later stage of the thought process (a later phase of progressive deepening).

8 In machine simulation terminology, the way in which the subject tackles the problem is very similar to the way in which a clever programmer might build a pop up (push down) list. The programmer could first put in the most drastic methods (thus, e.g., plan changes go to the bottom of the stack) and the last the least drastic ones (e.g., move alternatives near the top). The accessibility of the subsidiary methods would then more or less correspond to the subject's priority of applicability.

proof. Indeed the parallel might possibly be extended as far as 10. That is not necessary, however, since the only point is to show that as important as this hierarchy is, it can not in its present form be generalized to all phases of the thought process.

**Section 52: The method of progressive deepening and the scrapping of a possibility**

In Chapter IV, Section 32, p. 106, the concept of 'progressive deepening' was introduced. The term denotes a remarkable phenomenon peculiar to rather lengthy thought processes that are needed for solving difficult choice problems. The analysis of a certain idea (plan), move, or variant proceeds progressively in successive phases of (re-) investigation, either immediately or non-immediately. The investigation not only broadens itself progressively by growing new branches, countermoves, or considerable own-moves, but also literally deepens itself: the *same* variant is taken up anew and is calculated further than before. The term 'progressive deepening' is meant to include both aspects, which can only be artificially pried apart.

Before dwelling on this phenomenon we give a graphic description based on protocol ($M_2$; $B_1$) (cf. Section 30). Careful study will reveal five phases of deepening, namely: (1) lines 1-19 (first Phase); (2a) lines 20-35; (2b) lines 36-48; (3) lines 49-61; (4) lines 62-78 (striving for proof, final Phase). The diagrams show clearly how much and what is calculated in each Phase; Phase 2a and 2b have been taken together to collectively form the second phase, in order to increase the generality of the structuring.

The mature tree contains all of the moves mentioned in protocol ($M_2$; $B_1$). Each quadrant illustrates clearly what occurs during the Phase with which it is concerned, how the calculations successively broaden and deepen. For examples of 'deepening' in the literal sense 1 ... h7-h5 and 1 ... ♗f4-g5 are cursorily investigated in the second Phase and several moves deep in the third. The first move 1 ... ♖a8-b8 reappears in each of the four Phases (even twice, non-immediately connected, within the second Phase of exploration). The re-investigations of 1 ... ♖a8-b8 are all non-

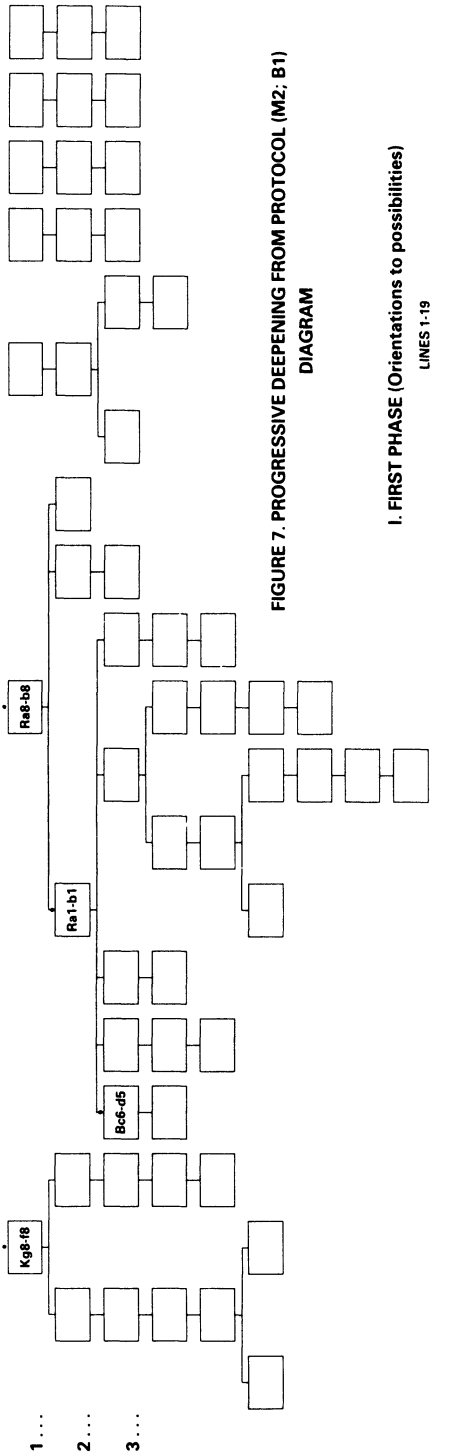
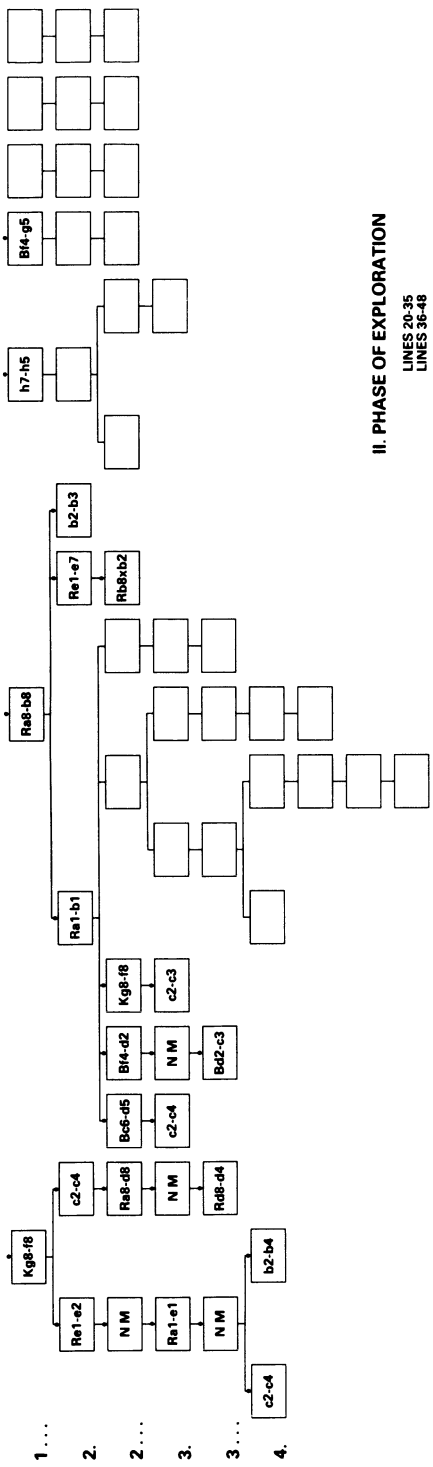


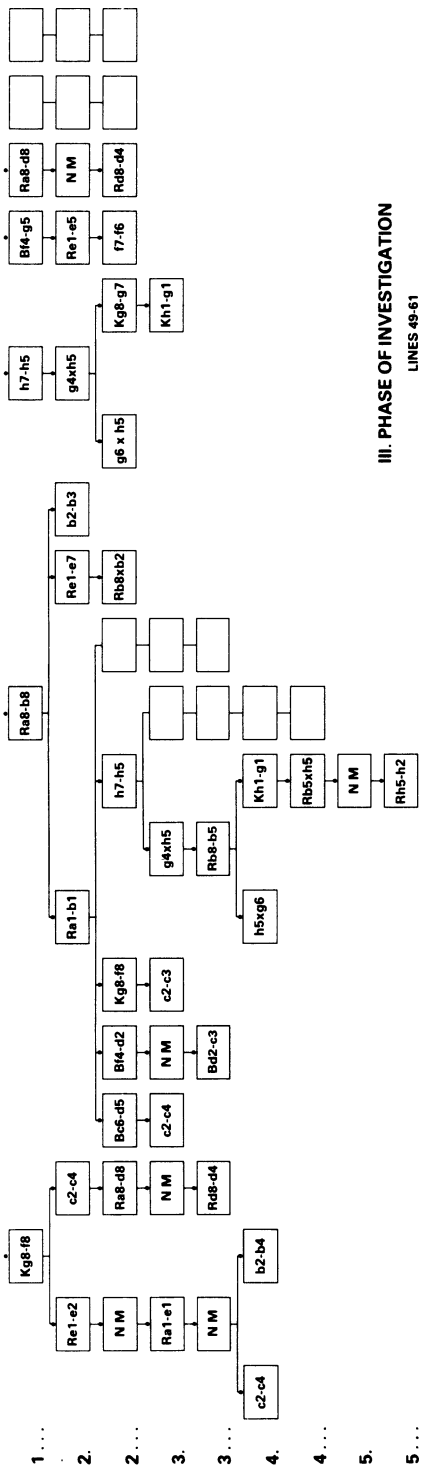

FIGURE 7. PROGRESSIVE DEEPENING FROM PROTOCOL (M2; B1)
DIAGRAM

I. FIRST PHASE (Orientations to possibilities)
LINES 1-19



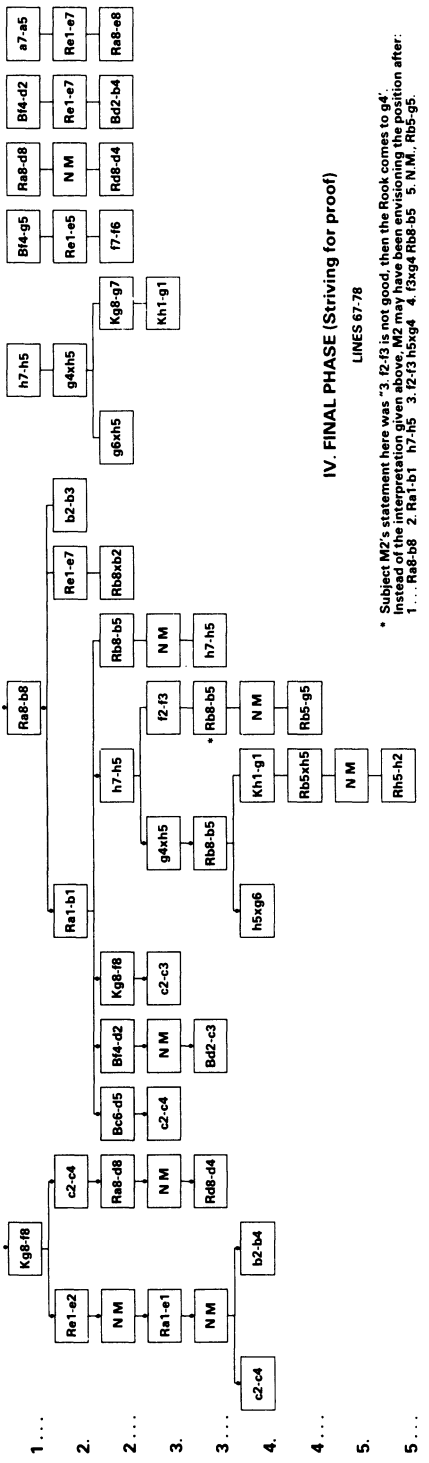

## II. PHASE OF EXPLORATION

LINES 20-35
LINES 36-48




## III. PHASE OF INVESTIGATION

LINES 49-61




**IV. FINAL PHASE (Striving for proof)**

LINES 67-78

* Subject M2's statement here was "3. f2-f3 is not good, then the Rook comes to g4". Instead of the interpretation given above, M2 may have been envisioning the position after: 1. . . . Ra8-b8 2. Ra1-b1 h7-h5 3. f2-f3 h5xg4 4. f3xg4 Rb8-b5 5. N.M., Rb5-g5.

immediate, except one, namely, the clear-cut transition to the final Phase (striving for a proof), lines 62-67, after a favorable result is obtained for the first time.

In principle, such Phase-trees may be constructed for every protocol (cf. Section 43), but only the lengthier thought processes show a marked structure of progressive deepening. Most of them can be divided into *four Phases of deepening* (that is, one more Phase than was distinguished in Chapter V in the discussion of the basic structure; the 'main part' is now broken down into two Phases). The four Phases are:

1. The *First Phase of Orientation,* especially orientation to possibilities. What we find here is largely 'looking at' the consequences of moves and general possibilities in a certain direction.

2. The *Phase of Exploration.* The subject tries out rather than 'investigates' possibilities for action. He calculates a few moves deep a few sample variations, or what he considers to be the main variation; if these are unsatisfactory he puts the move(s) in question temporarily aside.

3. The *Phase of Investigation.* There is a deeper, more serious search for possibilities, strengthenings, etc., that are quantitatively and qualitatively quite sharply defined. The investigation is more directed and much more exhaustive: more variants are calculated and they are calculated more deeply.

4. The *Final Phase of Proof.* The subject checks and recapitulates, he strives for proof; the obtained results are made into a subjectively convincing argument. A certain completeness is sought in the calculation of results, be it for the positive or negative part.

In accordance with Selz's conception of subsidiary methods (cf. Section 51) a subsidiarily linked new method is only applied when the available material (for the first method) is exhausted. The principle of progressive deepening now shows that such *'exhaustiveness' must be conceived of relatively.* Progressive deepening comes into being because in the first parts of the thought process the subject is content with orientation and exploration of possibilities, that is, with a provisional and incomplete investigation; thus he drops a possibility before the material is objectively exhausted.

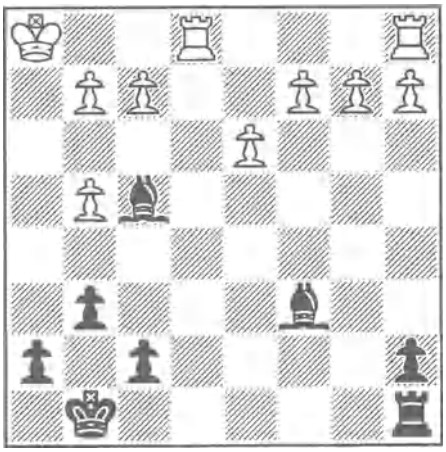
EXPLANATION OF SYMBOLS

[ ] Moves yet to be discovered.

[Z1] The move (of a variation) mentioned, whether or not for the first time, in the Phase under consideration.

[Z1] The move (of a variation) already mentioned in a previous Phase.

[NM] Assume that *some* move is made at this juncture; continue calculating the variation, however, without specifying *what* move. That is, make a 'no move' (N.M.).



*Position B* *Black to move*

Remarkably enough, a truly exhaustive coverage is rare even in the third and fourth Phases. The total argument as implied in a protocol as a whole, although 'subjectively convincing,' is usually incomplete and lacks objective proving power. Certain parts, not self-evident yet essential for an objectively correct solution, are missing. This is often a matter of *force majeure:* the possibilities simply cannot be completely worked out in one's head — and not even in an analysis in objectively unsolvable positions. In other cases it is clearly impractical to strive for exhaustiveness. When, for instance, a certain move is probably the best one and cannot do much harm in any case, then it is rather pointless to conduct too thorough an analysis to prove it. Even when verification is possible, precious time is rather saved for more difficult moments — a thought habit from practice which carries over to the experimental situation. Finally, unintentional incompleteness also figures in many protocols: overlooking essential possibilities in own- and counterplay.

Thus the measure of 'relative exhaustiveness' that determines a transition remains as crucial a question for the final choice of move as for the preceding transitions in the process of progressive deepening: What requirements must be fulfilled for the subject to consider the results of his investigation convincing enough to reject or accept a move? In particular, to accept *the* move and then to play it? On what factor does this decision depend if the criterion of exhaustion of the material is variable and relative and therefore insufficient as an explanation?

Let us confine ourselves to the quantitatively most important case: *the mechanics of provisional or definite rejection, the scrapping of a possibility.*

We have said that this always happens when an attempted solution 'fails' — but this expression cannot but mean that an 'exhaustive' investigation of the possibility in question yields only unsatisfactory results. How should this conception be specified? Which factors must in general be taken into consideration to account for the fact that at a certain moment the subject drops a possibility?

1. In general we may retain the rule that says that the subject will drop a possibility (move, plan, idea) when the results of the elaborations are unsatisfactory, that is, when they are *lower than the maximal expectancy.*

However, this rule does not always hold during the orientation to possibilities in the first Phase where a highly favorable (looking) move may be put temporarily into the backlog. Even in later Phases a favorable result may occasion a rise in the maximal expectancy under the device: 'If it's that easy, perhaps I can get even more out of it some other way.' The satisfactory possibility is then set aside (temporarily rejected) as a minimum solution which guarantees a minimal expectancy.

2. The more unsatisfactory the results of the investigation, i.e., *the higher the (negative) proving power,* the easier and more definitive the rejection will generally be. Whether or not unsatisfactory results have sufficient proving power depends on several factors:

a. The *completeness* of the elabor-

ations. One sample variation has little proving power while an exhaustive calculation of all relevant branches has much. We are concerned of course with the subjective proving power which depends on the degree of completeness the subject *thinks* he has reached.

b. The *differences* between the quantitative results of the variants and the maximal expectancy. The greater these negative deviations, the stronger, in general, the tendency to reject the possibility. A continuation that could entail a mating net or loss of a piece is dismissed sooner than one that might involve the loss of a slight positional advantage or an unimportant Pawn.

c. The *degree of uncertainty within the appraisal of the envisaged end positions* of the variants. The narrower the margin of uncertainty,[9] the sooner the subject arrives at a rejection, especially a definitive rejection. The degree of uncertainty is dependent not only on the subject's skill and self-criticism (a personal factor) but also on the character of the envisaged position. In general he strives for as definite results as possible by calculating the variants until a relative rest point is reached, that is, an envisaged end position in which the complications are over and the appraisal may be based on a reasonable degree of certainty, that is, until he reaches a *dead position.* He tries to avoid end positions where pieces are still *en prise,* where pieces must be recaptured, where the King is in check, or where serious threats are still present; in such cases he rather calculates a step further, even during an orientative investigation. He strives for an even 'deader' position. Of course, this is not always possible. All subjects investigating the position A variant 1 ♘c3×d5 ♘f6×d5 2 ♗a2×d5 ♗e7×g5; etc., feel compelled, at least in the provisional calculations, to stop in the middle. $G_2$ says in line 24: '... with many capturing possibilities, but on first sight not convincing.' If need be, experienced players are able to assess such positions, but the margin of uncertainty is of course wider.

d. Finally the proving power depends on *the reliability of the quantitative expectancy* against which the results are measured. If, for instance, the maximal expectancy is rather speculative and if the subject tries for an immediate win without really believing (anticipation!) that one exists, then a move that renders no more than a 'favorable position' is not rejected so quickly or so definitively. Had his expectations of gain been more certain, then he would be more prone to reject. Naturally, we have to do here with a subjective estimate (anticipation). This measure plays an important part in the (likewise subjective) proving power.

3. The presence of a negative deviation (1) and the actual degree of proving power of the results (2) do not yet fully explain the mechanism of rejection, since *how much* proving power is needed to satisfy the subject varies greatly from case to case. It appears to depend on:

a. *The skill of the subject* and *the energy he is willing to spend* on the choice of move.

b. *The character of the position.* The degree to which it is possible to attain a reasonable completeness of the analysis and/or a reasonable certainty of quantitative expectations, on the one hand, and of result appraisals, on the other, varies greatly. The subject judges (anticipates) this and adjusts his requirements correspondingly.

c. The specific function and place of the (preceding) *investigation* within the thought process. The subject's requirements for proving power (and, with that, depth and extensiveness) differ, depending on hs present purpose: orientation, exploration, investigation, or proof (see Figure 7). The stage of progressive deepening determines, as it were, the approximate amount of energy to be spent on a particular investigation. In the early Phases the subject's requirements for rejection are easily satisfied; he strives for a quickly attainable, tentative rounding off with a tentative decision; in the later Phases he strives for greater certainty and completeness, at least in the elaborations essential for the final argumentation. It is, in fact, the gradual increase in the subject's proving power requirements that brings about the

9 In programming terminology: *The reliability of the evaluation function* for the type of position evaluated is judged and taken into account by the human subject. In the terminology of this book this is another example of an *anticipation* determining the course of the process.

phenomenon of progressive deepening.

d. *Other special anticipations and components of the total goal conception* may dispose the subject to be satisfied with less proving power or, on the other side, to strive for greater certainty. A few examples:

– The interval of uncertainty around result appraisals may be asymmetric in that the gain, at best, is small while the loss – if something goes wrong – may be fatal. Such a situation, as judged (anticipated) by the subject, naturally expedites the rejection of the possibility concerned. The involved *risk* derogates from the attraction of the possibility even if the (incomplete) results attained by actual calculations are hardly negative.

– Since risky moves require keener and deeper calculations than safe moves, another element is present in such a rejection; namely, *the avoidance of voluminous calculations.* Indeed, this may be an autonomous motive. Continuations that can only be reasonably evaluated via elaborate calculation are deferred sooner. From ($M_1$;A)*, comments after the experiment:

20 (1 ♗a2×d5) ♗c6×d5 2 ♘e5-g4 would be very strong. White stands better; temporarily let it be.

Even though the envisaged position after 2 ♘e5-g4 is anything but a natural resting point (dead position) and even though the tentative result is only slightly negative anent the maximal expectancy, M1 puts ♗a2×d5 aside. He explains this himself in the following lines:

23 A stupid move perhaps?
1 ♗g5-h6; if that should prove even stronger, then I don't have to figure out all the variants of 1 ♗a2×d5

This remark clearly expresses a shirking away from elaborate calculations and further exposes the economy of thought inherent in the method of progressive deepening.

In a similar way it is possible to enumerate a number of factors pertinent to the transition to the move decision. Of prime importance is the subjective conviction, in this case the proving power of the total, usually dual, argumentation. The degree of proving power needed to satisfy the subject depends on about the same factors as pointed out in 3 above: the skill and the available energy of the subject (the latter often determined by the clock in an actual game), the character of the position, the size of the risks, etc. Here, too, the argumentation is almost always left incomplete; there remains a certain amount of uncertainty anent the estimated value and the quantitative results of the elaboration. This uncertainty is subjectively removed by what could be called an *intuitive and emotional completion of the argumentation.* As a last resort a 'feeling that ...,' an impression, a hunch, sometimes even an unadulterated emotional preference tips the balance in favor of some move.

The principle of progressive deepening and the many other factors on which transitions and decisions are dependent make the entire methodology of the thought process much more complicated than the simple schema *Denk*-psychological theory would have it be. The strict linkings: failure of a solution attempt → resort-to-the-subsidiary-method, turn out to be only definable with many qualifications and only conditionally applicable, in reality. The concept 'failure of a solution attempt,' which was already made relative by the introduction of a variable self-imposed maximal expectancy, now proves to be even less of a mainstay. All kinds of temporary 'failures' are possible, dependent on variable proving power requirements. These requirements, including the criterion of satisfactory completeness (exhaustiveness) of the analysis, in their turn depend on the stage of the thought process and on a number of the subject's specific judgments and anticipations. Finally, the objective 'incompleteness' of the final argument together with its 'intuitive completion' by the subject point to the important role played by nonobjectifiable and even non-rational factors that the rigid theory of *Denkpsychologie* does not take into account.

## Section 53: The elaborative phases and their sequence

If anywhere, it is within the elaborative phases where we can speak of sequences of automatically performed operations. The more microscopically we examine the phase structure, the more we come across automatic processes and sequences that must be governed by an extensive system of

fixed linkings. Calculating a variant and working out a plan are, in fact, operations which in any experienced chess player are completely controlled by routine. These operations, envisaging a position included, are executed so often in practice that the process runs almost automatically, be it faster and easier with one player than another. Each elaboration is, of course, controlled by its own specific goal-setting but consists in itself of a complicated, sometimes highly branched system of subproblems and of mutually subordinated and co-ordinated subphases – as we have seen in former sections.

We shall delve no further into the microdynamics of these elaborations for the simple reason that the material obtained by thinking aloud does not allow for a systematic analysis. A few special points will be taken up in Section 55 when methods are described.

The question we now pose concerns the sequence of elaborative phases. We just examined why a subject scraps possibility p; the question now is why does he turn to possibility q and not to r, for instance. *Which factors determine the order in which possibilities or variants are to be investigated?*

In answering this question we shall disregard the exact moment when the subsidiary linking of q to p comes into being in the mind of the subject. We did see in Section 48, however, that it can happen before, during, or after the investigation of p, in the last case in the transitional phase. Here we are only concerned with the relation between p and q – if there is one.

1. The first factor may be called *indirect determination:* the subject turns from possibility p to q because q is the only remaining possibility yet to be considered, either by itself or within the framework of a certain plan. When does it appear?

First, whenever one of two pre-stated alternative possibilities is (provisionally) rejected and the other is taken up. In other words: Whenever there is an already established binary choice between moves, plans, variations, strategies, attacks, defenses, etc. From the importance and high frequency of alternative groupings, the importance of indirect determination is clear. Second, a subproblem may fall in threes, or fours, or more. In such a case the last part, when taken up is indirectly determined. Indirect determination is actually the rule later on in the thought process when the subject's thinking gets organized and structured into circumscribed groupings.

2. The investigation of possibility q may follow p because q promises to be *relatively the best* on the basis of prior scanning (and temporary dismissal) of q and other moves aside from p. Earlier quantitative results and expectations (anticipations) now determine the order of investigation, the most promising possibility coming first. This factor is very important, though limited by the necessary condition of prior scanning.

3. A third factor is the *relation* that may exist *between p and q with respect to their board function.* Very often the subject tries q on the strength of q-anticipations resulting from certain qualitative experiences with p. Possibility q may, for example, incarnate a modification of the idea basic to p; q may be a move that prepares the way for p; q may result from a permutation in the order of moves, or be an accelerated execution or otherwise improved version of p, etc. Possibility q may also be the reverse of p; namely, when p is disqualified by the qualitative results of its calculation so that a search in a completely different direction is begun; thus q is investigated.

Whatever the specific connection between the ideas basic to p and q may be, it is the subject's insight into the chessical causes of the relative failure of p which leads to the new solving proposition q. When the strength of a certain defense by the opposition becomes obvious during the investigation of p, that continuation is tried (q) which makes such a defense impossible; if the enemy King can retreat to safety after a normal chase (p), then an attempt is made to cut him off at the pass (q); if the launching of a carefree counterattack (p) is slightly retarded, then defensive measures (q) may first be called for; etc. Many similar p-q relationships appear in the protocols. If this relationship is of a typical nature, we find ourselves in the next group.

4. The choice of q after p may result from a general *priority rule.* In fact, we find *fixed orders, natural hierarchies of methods* at all levels. The existence of general subsidiary linkings of this type can be inferred from typical sequences in the pro-

tocols and accompanying remarks. At the beginning of the investigation of move p the protocol text gives, for instance: 'First (investigate) direct moves' – by way of a principle or rule – while in fact q is second to p in 'directness.' Statements of this tenor are found in sundry protocols. In general, direct moves appear to have priority over calmer moves. Protocol ($M_4$; ... ♕e7) – presented in full on pp. 249-253 – provides a more differentiated hierarchy: subject $M_4$ looks 'at direct combinations first,' then for aggressive positional moves (1 ... ♘d6-b5), next towards its preparation (1 ... ♗g7-f8) and finally for a waiting move (1 ... ♕d7-h3). Only a few priority rules appear to be completely independent of types of position, however. Thus, in positions where there is 'really very little cooking' tactically ($M_2$; ♔g2), line 22, the priority of forceful moves over calmer ones is hardly applicable and may be replaced by another: for instance, moves according to some active plan first, then waiting moves.

In general, priority rules are sensible and understandable. Thus, in most situations the investigation of calmer moves which maintain the tensions and the *status quo* is not economical until it is known that there is no immediate advantage to be gained or threat to be parried. For the calculation of *sample variations* of a move, in the Phase of exploration, the priority rule 'first direct continuations' may take on the meaning that the first variant to be examined must be the one that best exposes the underlying board intention or idea. Suppose that the idea is through one or more sacrifices to ensnare the enemy King in a mating net, or to gain space and open lines, then the acceptance of the sacrifices has priority over their refusal. Another example, taken from protocol A calculations: The *point* (underlying idea) of 1 ♗a2×d5 (and not 1 ♘c3×d5) is that in reply to black's 1 ... ♘f6×d5, 2 ♘c3×d5 wins a tempo (by attacking the Queen) and thereby wins a piece. That is why the variant 1 ... ♘f6×d5 is primarily the 'main variation' and takes precedence in every protocol where the subject calculates the three branches ensuing from 1 ♗a2×d5.

A different priority principle obtains with *proof calculations*. Here it is a striving for liquidation of subproblems that often appears to determine the order of priority. The subject may, for instance, first examine the least promising moves or continuations in order to delimit what is worth considering and so be able to focus his attention and to economize in his analysis. Subject $M_1$'s classic remark will be remembered:

($M_1$; A)*:

14 Now begin with the probably bad countermoves – in order to be able to eliminate those.

5. Finally we must allow for chance, for the factor of *serendipity* – a factor that entered the theory of thinking with Selz.

'Bad luck,' of course, is not an acceptable excuse for 'not seeing' something that cost a player the game. He should have seen to it that he did see it. Conversely it is, generally speaking, not 'good luck' when a player detects a hidden winning possibility. Still, the element of chance may be present within the thought process and, indeed, very often is. A player can be blamed for not seeing a possibility, but he cannot be blamed for not seeing it immediately. He may not see a certain possibility until the very last check-over, while his hand hovers over the wrong piece. In other words: Serendipity certainly plays a role in determining the order in which the subject discovers and investigates certain possibilities.

The fact that in position A subject $G_4$ only 'finds' the move 1 ♗a2×d5 after six minutes of thinking and calculating doesn't say anything against him; nor is $G_5$ any less a grandmaster because he discovers by chance – while calculating – the advantages of 1 ♗a2×d5, that is, without prior insight. It is quite obvious that the point in time at which the idea dawns depends to some extent on chance. In both thought processes we are concerned with 'calculational serendiptiy,' that is, during routine calculations there may be a sudden, coincidentally evoked means abstraction with respect to the main goal. In ($G_4$; A) for example, the tactical possibility 1 ♗a2×d5 emerges in connection with a calculation of 1 ♖f1-e1 ♕b6×b2, 2 ♗a2×d5 etc.

In addition to calculational serendipity 'perceptual serendipity' may lead to coincidental or immediate means abstractions that influence the order of elaborative phases. While scanning the board during the innumerable pauses and transitional phases, a certain constellation, certain relationships between the pieces, certain

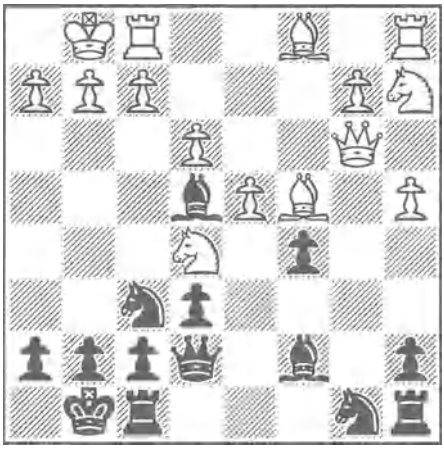
possibilities, 'may strike the subject's eye.' Thus one subject may see[10] move *a* first, while another subject's eye may fall primarily upon possibility *b* – resulting in a different order of solution proposals and elaborative phases.

Considering the attention that the more romantic conceptions of thinking and creativity have generally devoted to the phenomenon of sudden, apparently coincidental and unprepared findings, enlightenments, inspirations, brain waves, *Aha-Erlebisse,* etc., it is of some interest to evaluate their frequency and their importance in chess thinking generally. If such a find really is mainly coincidental, it can be expected to have an element of surprise for the subject that may be expressed in the protocol in the form of some exclamation like: 'Wait a minute ...,' or 'Aha,' or something similar. What does the analysis of protocols yield in this respect?

First, exclamations of the 'Aha'-type appear to be rare: in the entire material – including the short thought processes – we do not find even ten clear cases.

Second, among those found, a few represent the finding of a solution attribute rather than a new direct means for solving the main problem. When, for instance, it is stated in ($C_5$; A), line 118: 'Wait a minute, I can make him choose (by playing 1 ♘e5×c6) ...,' then no new move is discovered but solely a relationship: the necessity of the opponent's choosing between two possible lines of defense might be exploited in some way or other.

Third, notwithstanding the subject's surprise in the other cases, the find takes place in a phase of searching in a fairly well defined direction as seen from the text of the protocol. One example: the end of the first Phase of



*POSITION ... ♘b8-c6* *Black to move*

(M4; ♘b8-c6):

11 Let's see ... if there's something to be done. It would seem as though something must happen with those two Bishops.

15 I'm looking at something like 1 ... ♗c7×e5 and 2 ... ♘f6-g4. But that's not a very serious combination: he plays 3 f2-f3 on 2 ... ♘f6-g4. Aha, if I just play

20 1 ... ♘b8-c6 and 2 ... ♗c7×h2+ if he takes! 3 ... ♘f6-g4+; the King must go to g3 ....

After the failure of the 'not very serious' first solving proposition, the goal-setting 'Let's see ... if there's something to be done,' reclaims its rights; indeed, in the chance discovery something does 'happen with those two Bishops.' Thus we may certainly speak here of a search in a certain direction; a clear schematic anticipation precedes the discovery.

The same holds for most of the other cases.

We must conclude from this that the actual *brain wave* coming suddenly and surprisingly, apparently independent of a goal-directed search, plays but a minor role in chess thinking. In interpreting this finding we must, of course, keep in mind that a brain wave, an Aha-*Erlebnis,* is an experience: an insight accompanied by the *emotion* of surprise. Therefore an expert in any field is much less likely to be 'surprised' by his own findings than a beginner is. Also, the suprise criterion, although part of the definition, by no means guarantees the find to be of any value, nor does it seem to contribute very much conceptually to an explanation of the dynamics of productive

10 As always when we talk about 'seeing' in chess, it is the perception of the *relations* between the visible objects and not the perception of the objects (board and pieces) themselves.

thinking. These considerations are in accordance with general results of *Denk*-psychological investigations. The conspicuousness of quasi-unprepared, namely, 'surprising' findings, has led the more romantic schools – Gestalt psychology included – to overestimate the functional importance of such finds. Compare the reduced significance of the phenomenon of inspiration in musical creativity (Bahle 1936 and 1939).

On the other hand, the scarcity of brain wave-like experiences might to some extent be peculiar to chess thinking that is bound to actual game playing. In situations where a chess problem can be 'carried around' like creative problems for any length of time – e.g., an adjourned game position, some analytic position, or a problem of chess composition – actual brain waves might occur more often.

These last explications detract nothing, of course, from the importance that serendipity can have for the moment of 'finding' some solving means and, with that, for the sequence of investigative phases.

---

Because of the frequent references in this chapter to other parts of de Groot's book, I have left the original page references. Readers who wish to purchase the book, *Thought and Choice in Chess*, may like to know that there is a second edition in print (1978). The publisher, Mouton, has offices in The Hague, New York and Paris.

# 5.2 Skill in Chess

**Herbert Simon and William Chase**

First published in *American Scientist,* volume 61, July-August 1973, pp. 394-403.
Reprinted by permission of American Scientist.

As genetics needs its model organisms, its *Drosophila* and *Neurospora,* so psychology needs standard task environments around which knowledge and understanding can cumulate. Chess has proved to be an excellent model environment for this purpose. About a decade ago in the pages of this journal, one of us, with Allen Newell, described the progress that had been made up to that time in using information-processing models and the techniques of computer simulation to explain human problem-solving processes. (Simon et al, 1964). A part of our article was devoted to a theory of the processes that expert chess players use in discovering checkmating combinations (Simon et al, 1962), a theory that was subsequently developed further, embodied in a running computer program, MATER, and subjected to additional empirical testing. (Baylor et al, 1966).

The MATER theory is an application to the chess environment of a more general theory of problem solving that employs heuristic search as its core element (Newell et al, 1972). The MATER theory postulates that problem solving in the chess environment, as in other well-structured task environments, involves a highly selective heuristic search through a vast maze of possibilities. Normally, when a chess player is trying to select his next move, he is faced with an exponential explosion of alternatives. For example, suppose he considers only ten moves for the current position; each of these moves in turn breeds ten new moves, and so on. Searching to a depth of six plies (three moves by White and three by Black) will already have generated a search space with a million paths. Hence, if *every* legal move is considered (as would be the case in an exhaustive search), an enormous search space would be generated. Such a search is beyond the capacity of the human player, as well as present-day computers. Humans seldom search more than a hundred paths in choosing a move or finding a checkmate, and they seldom consider more than two or three possible moves per position.

The MATER theory postulates that humans don't consider moves at random. Rather, they use information from a position and apply some general rules (heuristics) to select a small subset of the legal moves for further consideration. For example, one powerful heuristic that MATER uses in finding checkmates is to examine first those moves that permit the opponent the fewest replies. A comparison of the MATER program with thinking-aloud protocols from human chess players confirms the importance of heuristic search as a basic underlying process.

While the MATER theory was successful in accounting for much of what was known about chess thinking in mating situations, some important empirical phenomena — some of them known when the theory was formulated, some of them discovered subsequently — eluded the theory's grasp. In this paper, after describing the phenomena, we should like to tell the story of a ten-year effort to account for the recalcitrant facts.

An important by-product of this effort has been to bring about a convergence of the theory of problem solving with theories that have been developed to explain quite different phenomena, which psychologists label "perception," "rote learning," and "memory". In the past, both theorizing and experimentation relating to these different kinds of tasks — problem solving, learning by rote, and remembering — have tended to go their separate ways. In the course of our story we will see how these theories come together to explain chess skill; we will see

the important constraint that a limited capacity short-term memory imposes on problem solving in chess and how this limit can be bypassed by specific perceptual knowledge acquired through long experience, stored in long-term memory, and accessed by perceptual discrimination processes.

## The phenomena

In Amsterdam, Adriaan de Groot (1965), who was the first psychologist to carry out extensive experiments on problem solving using chess as the task, also initially formulated his theory in terms of heuristic search. His subjects ranged from quite ordinary players to some of the strongest chess grandmasters in the world, including several former world champions. He was puzzled by one thing: none of the statistics he computed to characterize his subjects' search processes – number of moves examined, depth of search, speed of search – distinguished the grandmasters from the ordinary players. He could only separate them by the fact that the grandmasters usually chose the strongest move in the position, while ordinary players often chose weaker moves. Why were the grandmasters able to do this? Wherein lay their chess skill?

### *The perceptual basis of chess mastery.*

One clue to this riddle came when de Groot repeated and extended an experiment that had been performed earlier in the USSR (D Jakow et al, 1927).

He displayed a chess position to his subjects for a very brief period of time (2 to 10 seconds) and then asked them to reconstruct the position from memory. These positions were from actual master games, but games unknown to his subjects. The results were dramatic. Grandmasters and masters were able to reproduce, with almost perfect accuracy (about 93% correct), positions containing about 25 pieces. There was a quite sharp drop-off in performance somewhere near the boundary between players classified as masters, who did nearly as well as grandmasters, and players classified as experts, who did significantly worse (about 72%). Good amateurs (Class A players in the American rating scheme) could replace only about half the pieces in the same positions, and novice players (from our own experiments) could recall only aobut eight pieces (about 33%). There is a quite nice gradation on this perceptual task as a function of chess skill, and we have verified this in our own experiments (Chase et al, 1973a).

We went one step further: we took the same pieces that were used in the previous experiment, but now constructed random positions with them. Under the same conditions, all players, from master to novice, recalled only about three or four pieces on the average – performing significantly more poorly here than the novice did on the real positions. (The same result was obtained by W. Lemmens and R. W. Jongman in the Amsterdam laboratory, but their data have never been published, [Jongman, 1968]).

In sum, these experiments show that chess skill cannot be detected from the gross characteristics of the search processes of chess players but can be detected easily using a perceptual task with meaningful chess content. The experiment with random boards shows that the masters' superior performance in the meaningful task cannot be explained in terms of any general superiority in visual imagery. The perceptual skill is chess-specific. Moreover, a theory of problem solving in chess that does not include perceptual processes cannot be an adequate theory – cannot explain the superior ability of the strong player to choose the right moves.

### *Eye movements at the chess board.*

The second set of phenomena we must consider are also perceptual, but of a more recent discovery. Explanations in terms of heuristic search postulate that problem solving, and cognition generally, is a serial, one-thing-at-a-time process. (We are oversimplifying matters to make the issue clear, but the over-simplifications will suffice for the present.) Many psychologists have found this postulate implausible and have sought for evidence that the human organism engages in extensive parallel processing (Neisser, 1963). The intuitive feeling that much information can be "acquired at a glance" argues for a parallel processor. Of course, the correctness of the intuition

depends both on the amount of information that can actually be acquired and upon what is meant by a "glance." If a glance means a single eye fixation (lasting anywhere from a fifth of a second to a half-second or longer), then we know that there are high-speed serial processes (e.g. short-term memory search, visual scanning) that operate within this time range (Sternberg, 1969).

Thus, it is certainly interesting and relevant to find out how the human eye extracts information from a complex visual display like a chess position and to see whether this extraction process is compatible with the assumptions of the heuristic search theories.

A pair of Russian psychologists, Tichomirov and Poznyanskaya (1966), placed an expert before a chess position with instructions to find the best move, and they observed his eye movements during the first 5 seconds of the task. The eye movements were inconsistent with the hypothesis that the subject, during these 5 seconds, was searching through a tree of possible moves and their replies.

To describe further what Tichomirov and Poznyanskaya found, we must say a word about how the eye operates. The eye has a central region of high resolution, the fovea (about 1° in radius), surrounded by a periphery of decreasingly lower resolution. Most information about visual patterns is acquired while the fovea is fixated on them; and the eye moves abruptly, in so-called saccadic movements, from one point of fixation to the next. There are at most about four or five saccadic movements per second.

In Tichomirov and Poznvanskaya's record of the first 5 seconds of their subject's eye movements, there were about 20 fixations. Most of these centred on squares of the board occupied by pieces that any chess player would consider to be of importance to the position. There were few fixations at the edges or corners of the board or on empty squares. Moreover, a large number of the saccades moved from one piece to another, where the former piece stood in a "chess" relation – that is, an attack or defense relation – to the latter. For example, the eye would move frequently from a pawn to a Knight that attacked it, or to a Knight that defended it, or from a Queen to a pawn it attacked.

It is important to note that the saccadic movements were not random – therefore, that some information must have been acquired peripherally about the target square before the saccade began. From other evidence, we know that a strong chess player can *recognise* a piece within a radius of 5° to 7° from his point of fixation; for eye-movement studies show that he can frequently replace such a piece correctly on a board when he has had no closer point of fixation to it (Noordzij, 1967).

The Russian experiments are of interest for two reasons. First, while the saccadic eye movements themselves are serial, some parallel visual capacity appears to be operating, for, since the saccade is not random, information about the target square must be acquired peripherally. From what we know about search and scanning rates, it can be concluded that the processes of scanning the periphery for the next target square and preparing the next saccade must overlap in time with the processes of searching memory for the identity and function of a piece (or square) presently occupying the fovea. Visual scanning experiments show that an eye fixation does not allow enough time both to recognize a pattern in the fovea and to scan the visual periphery for a likely target for the next fixation unless the two processes overlap in time (Williams, 1966; Ellis et al, 1971).

Even more important, the Russian experiments confirm the existence of an initial "perceptual phase," earlier hypothesized by de Groot, during which the players first learn the structural patterns of the pieces before they begin to look for a good move in the "search phase" of the problem-solving process. The experiments of Tichomirov and Poznyanskaya have been repeated and confirmed both in Amsterdam and in our own laboratory. How shall we extend the heuristic search theory or problem solving to accommodate them?

## Explaining the eye movements

Among the ground rules that ought to be followed in building theories, one of the most important is the rule of parsimony. If, in order to explain each new phenomenon, we must invent a new mechanism, then we

have lost the game. Theories, gradually modified and improved over time, are convincing only if the range of phenomena they explain grows more rapidly than the set of mechanisms they postulate.

In the present instance, there are two ways in which we may seek to preserve parsimony as we extend the theory. First, we may examine our existing theory to see whether the mechanisms already incorporated in it might be adequate if they were reorganized. Second, if we need additional mechanisms to explain some of the phenomena, then, instead of inventing them ad hoc, we may draw upon mechanisms already postulated or known in other parts of psychology — mechanisms whose existence already has empirical support. We will explore both of these routes for improving the theory while preserving parsimony.

### *Perceptual processes in* MATER

Let us return to the MATER theory and see how much we must add to, or subtract from, it in order to account for the eye movement data. MATER, as noted earlier, is a program for discovering mating combinations by selective search. What is the basis for the selectivity? A fundamental idea imbedded in MATER is that forceful moves should be explored first, where a forceful move is one that accomplishes some significant chess function, like attacking or capturing a piece or restricting the movements of the opponent. Discovering the opportunities for forceful moves in any chess position involves perceiving the attack, defense, and threat relations that hold among pairs and clusters of pieces on the chessboard — it is basically a perceptual process.

Hence, if we examine MATER a level or two below the executive routine that organizes its search, we see that the program is composed chiefly of a collection of processes for noticing significant chess relations among pieces or squares. In the program as originally organized, these processes were enlisted in the service of the heuristic search for a mating combination. Are these noticing processes a sufficient base on which to build a theory of the eye movements?

### *The* PERCEIVER *program.*

It proved surprisingly easy to simulate the eye movements. It was not difficult to replace MATER's executive program with a new program that used the same perceptual processes to guide the scanning of the board, and when this was done, a good correspondence was found between the squares fixated during the first 20 saccades by the human player and the squares fixated by the program (Simon et al, 1969).

The program, dubbed PERCEIVER, operates in a very simple manner. With the simulated fovea fixated on a square of the board, information is acquired peripherally about pieces standing on nearby squares that attack or defend the fixated square, or that are attacked or defended by the piece on that square. Attention is then assumed to switch to one of these nearby squares, and, unless it immediately returns to the square already fixated, causes a saccadic movement to the new square. With the fovea fixated on the new square, the process simply repeats. A moment's reflection will convince the reader that a process having this structure will cause a biased random walk of the fixation point around the board, returning most frequently to those regions where relations among pieces are densest and spending little time on the edges of the board.

Figure 1 is one of the positions used by Tichomirov and Poznyanskaya in their eye-

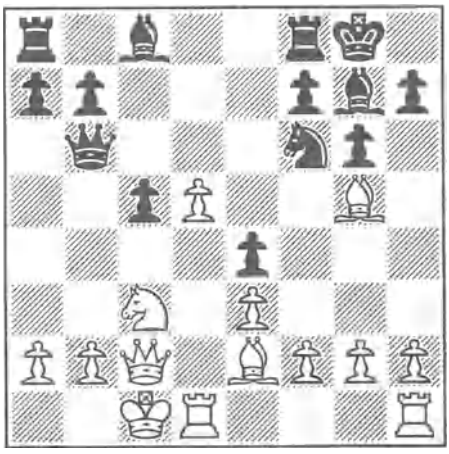


*Figure 1. In this middle game position, used by Tichomirov and Poznyanskaya in their eye-movement experiments, Black is to play.*

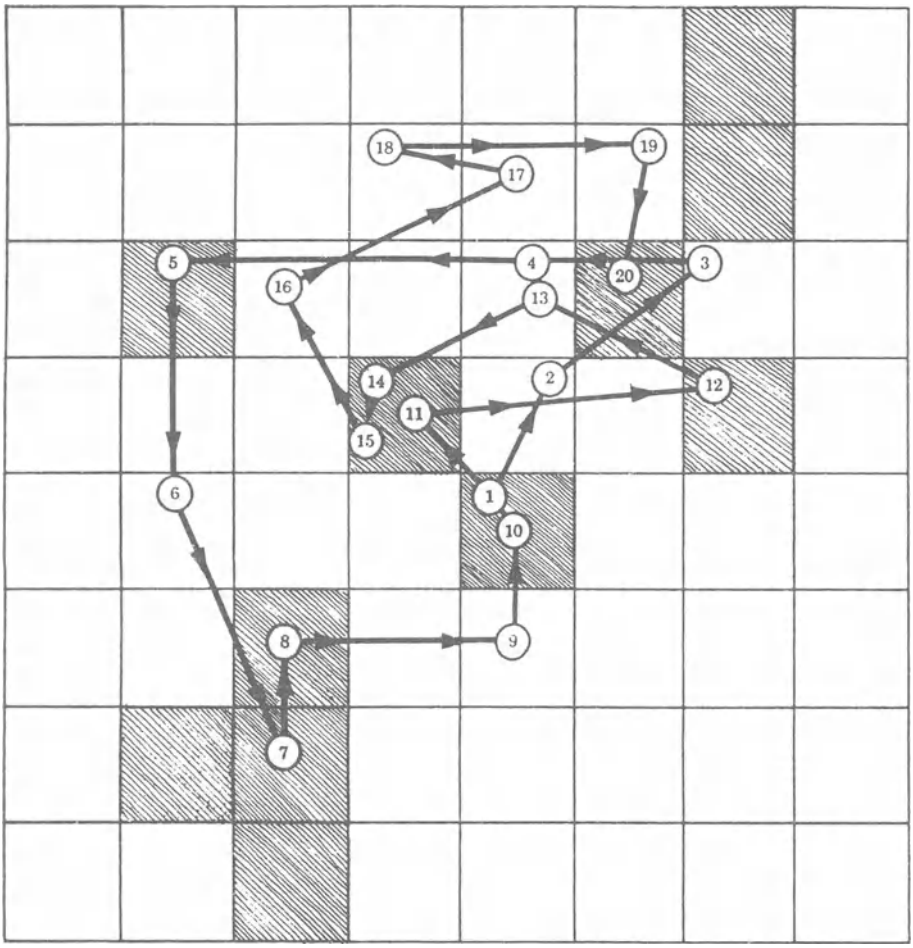



*Figure 2. Eye movements of an expert player are recorded for the first five seconds, by Tichomirov and Poznyanskaya. The ten squares occupied by the most active pieces (see Figure 1) are shaded.*

movement experiments; Figure 2 is a record of the first 20 fixations of their expert in this position; and Figure 3 shows the first 15 fixations produced by PERCEIVER in the same position. Of interest is the fact that the PERCEIVER simulation, by means of its simple mechanism of attending to attack and defense relations, shows the same preoccupation with the important pieces as does the human expert.

There are three points we need to make about this simulation. First, no new mechanisms were invoked; it was sufficient to reorganize the lower-level perceptual mechanisms of MATER. The difference between the behaviour of MATER and the behaviour of PERCEIVER lies largely in a difference in goal or motivation at different stages in the problem solving process. The empirical data from human subjects indicate that initially the player sets himself (not necessarily consciously or deliberately, but perhaps habitually) the task of acquiring information about the chess-significant relations on the board (PERCEIVER). Having acquired this information, he turns to generating moves and exploring their consequences (MATER). There would be no great difficulty in revising MATER to conform to this pattern — with the perceptual, information-gathering phase preceding the cognitive, heuristic search phase. As a matter of fact, one earlier computer chess program, written by Newell, Shaw, and Simon in 1958, had much of this flavor (Newell et al, 1958a),

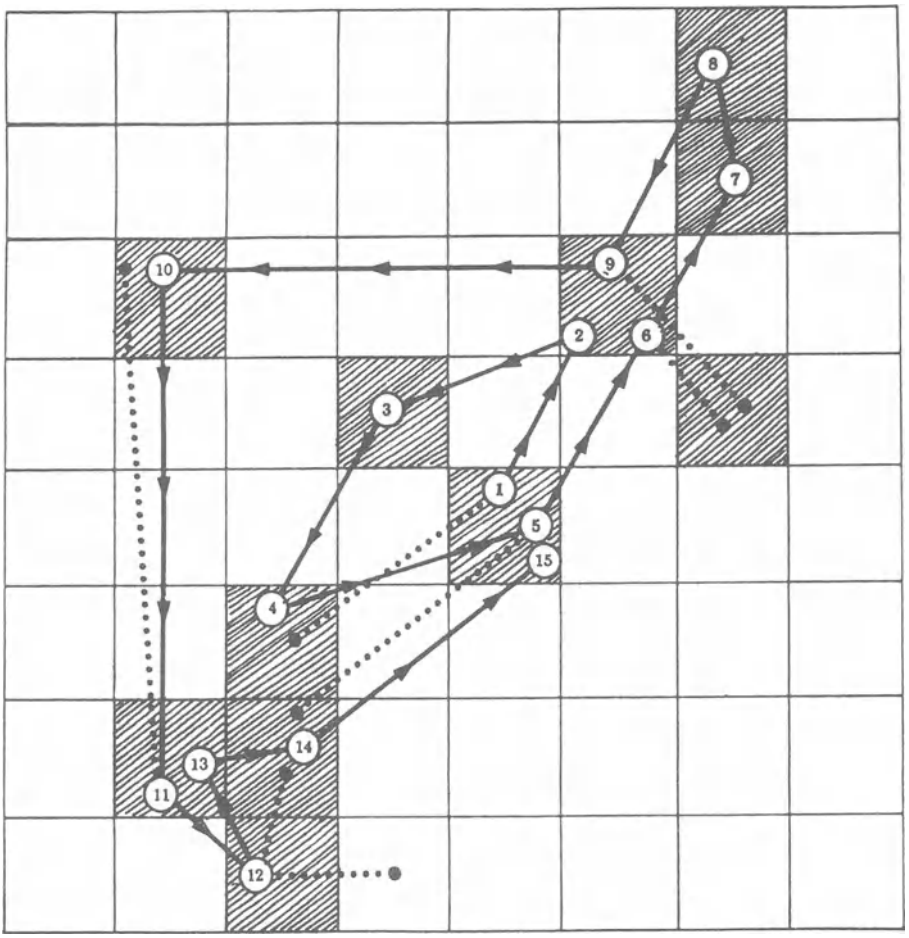



*Figure 3. The solid line represents eye movements and the broken lines represent relations noticed peripherally in this record of simulated eye movements during the period of initial orientation from the* PERCEIVER *program. The ten squares occupied by the most active pieces (see Figure 1) are shaded.*

and another such program is now being constructed by Berliner.

Second, there is nothing a priori parallel about PERCEIVER; the simple rules that drive the simulated eye around the chessboard are, in fact, serially organized, and it is a simple matter to simulate them in real time on standard computers. Even if realistic time parameters, estimated from human performance, were assigned to the various processes of PERCEIVER, it is still not clear that anything resembling a parallel process would be necessary. This problem is related to the third point.

Third, there is one level of perceptual processing that is finessed and one level that is entirely missing in PERCEIVER. The part that is finessed is the mechanism that recognizes the chess pieces in the first place. What is more important, while PERCEIVER notices attacks and defenses, it has no processes for organizing and remembering this information once it is attended to. But, as we shall see, the organizing process itself drives the eye movements. It is quite plausible that these missing processes operate partly in parallel with the scanning processes of PERCEIVER.

## The board reconstruction experiment

Nothing in the perceptual mechanisms we have described so far will allow us to account for the spectacular skill of chess

masters in reconstructing positions that they have seen for only a few seconds. Both MATER and PERCEIVER gloss over details of the process for recognizing a chess piece – noticing that it is a Bishop, say, rather than a pawn. Each piece is represented by a little bundle of features – its color, for example, and its type (King, Queen, etc.). The programs do not undertake to explain or simulate the feature extraction process, but simply assume that it is performed and that previous learning has stored in long-term memory the requisite information about the capabilities of the different kinds of pieces. More important, neither program contains any mechanisms for the recognition of meaningful, familiar patterns of pieces – neither program has a mechanism for the extensive storage in long-term memory of familiar patterns, nor indeed do they have a long-term memory of any complexity. But it is precisely this kind of pattern-recognition process that lies at the heart of the master's reconstructive ability.

*Elementary perceiver and memorizer.*

Still retaining our respect for parsimony, we note that there already exists in psychology an information processing theory to explain how feature-bundles can become familiarized, associated with other information in long-term memory, and used as components in larger organizations of structures. This theory, called EPAM (Elementary Perceiver and Memorizer), was initially developed by Feigenbaum to explain some of the principal empirical findings about the rote learning of nonsense syllables in the standard serial anticipation and paired-associate paradigms (Feigenbaum, 1961).

Among the striking phenomena that had been observed in rote learning are: (1) a characteristic shape of the serial position curve (in serial anticipation learning), (2) a three-to-one (approximately) time advantage in learning meaningful over meaningless and familiar over unfamiliar syllables, (3) certain characteristic differences in learning times between similar and dissimilar stimulus and response items, and (4) certain conditions that determine whether rote learning will have an incremental or an all-at-once appearance. EPAM has been successful in accounting for all of these phenomena (Simon et al, 1964).

The program of EPAM, and hence the theory it embodies, is quite simple. EPAM learns by growing a discrimination net – a tree-like structure whose nodes contain tests that may be applied to objects that have been described as bundles of perceptual features. When a familiar object is perceived, it is recognized by being sorted through the EPAM net. At the terminal branches of the EPAM net are stored partial "images" – also in the form of feature bundles – of the objects sorted to the respective terminals, together with other information about the objects.

The EPAM theory also plays an important role in explaining the eye movements. Recall that in the previous section, PERCEIVER was found inadequate because it contained no mechanism for recognizing pieces and patterns of pieces. A more complete theory of eye movements would require that PERCEIVER have access to EPAM.

The processes of EPAM influence the eye movements via the way the discrimination net is searched. Figure 4 illustrates a small section of the net with two terminal nodes. Observe that the nodes contain questions about the contents of specific squares; depending upon what is found at a square, a decision is made concerning which square to query next. In short, the EPAM net is organized as a set of instructions, albeit abstract, for scanning the board for familiar patterns. These instructions must then be interpreted by the perceptual system (PERCEIVER) in order to extract the information, and eye movements may well be necessary to execute the instructions. For small clusters of pieces, some of these successive recognition steps may be executed in a single foveal fixation, without saccadic movement. Thus, eye movements may be of two kinds: (1) initial familiarization, in which simple chess functions (attack, defense) are noticed, and (2) recognition, in which complex patterns are scanned.

This explanation of the eye movements gains additional support from the work of Noton and Stark, who developed independently a similar theory (Noton et al, 1971). They proposed that people's memory of a picture will determine how that picture is subsequently scanned for recognition, and

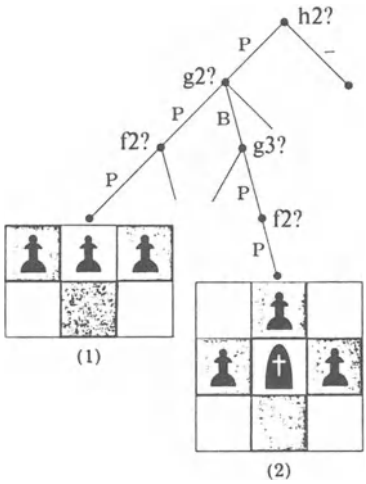



*Figure 4. A portion of the* EPAM *net for chess shows the terminal nodes for two patterns: (1) three pawns on second rank, and (2) fianchettoed Bishop. At each node is shown the test executed there. For h2, for example, read: "What piece stands on the h2 square?" The patterns at the terminal nodes are for illustrative purposes only: all the information needed to recognize the pattern is imbedded in the logic of the discrminiation net. The terminal node has the internal name of the node, an abstract symbolic reference (internal address) that can be stored in short-term memory as a single chunk.*

they presented evidence that, under the appropriate conditions, eye movements followed stereotypic "scanpaths" before the picture was recognized. EPAM makes this same strong assumption – that patterns are recognised by scanning the configuration for specific features in a particular order.

EPAM has a recursive structure. This means that any object, once familiarized and incorporated in the net, can itself serve as a perceptual feature of a more complex object. Thus, once the various types of chess pieces – Kings, pawns, Bishops – have become familiarized, these can become features of more complex configurations, say, a "fianchettoed castled Black King's position" (see Fig. 1 for this pattern in the upper-right part of the board). Once familiarized (and this particular pattern is known to every strong player), such a complex can, in turn, serve as a perceptual feature of a still more complex pattern – e.g. an entire chess position.

We have now illustrated the recursive structure of EPAM with a chess example, but the EPAM program was not constructed with this application in mind. In the context of rote verbal learning, the lowest level features in EPAM are the geometrical and topological properties of English letters. With familiarization, the EPAM net expands to encompass the letters themselves, which then can be used as components (test nodes) of nonsense syllables. Familiarization of the syllables, in turn, makes these available as components of syllable pairs or lists, and so on. Thus, EPAM postulates a single learning process, identical with what we have been calling familiarization, and a single kind of output of that process, a new unit or *chunk*.

The EPAM theory implies that the length of time required for a learning task will be proportional to the number of new chunks that have to be familiarized in order to perform the task. This implication also fits the empirical evidence very well, the basic learning time being about 5 seconds per chunk (Simon, 1969).

*Chunks and short-term memory.*

Finally, an additional mechanism, short-term memory, is needed in order to understand the reproduction experiment – a mechanism for holding all that information for the short period of time before it is recalled. George Miller, in order to account for the observed invariances in memory-span experiments, first postulated such a memory system with a constant capacity of about seven chunks (Miller, 1956). Miller showed that the well-known limit on the amount of information that can be held in short-term memory is not to be measured in bits, but in chunks – the capacity is about "seven, plus or minus two" familiar units of any kind. By acquiring new familiar units (e.g. octal digits) and learning to recode information in terms of those units (e.g. recoding from binary to octal), holding a constant number of *chunks* in short-term memory allows one to hold an increased number of *bits* (in the example, a gain of three to one). The chunk of EPAM theory has these same characteristics.

Since Miller's influential article was published, there has been a tremendous amount of research on short-term memory, and virtually every present-day theory

about cognitive processes incorporates such a memory system. Much research on thinking and problem solving has shown that, outside of strategies, the only other human characteristic that consistently limits performance in a wide variety of tasks is the small capacity of short-term memory. And without a short-term memory, EPAM theory by itself does not account for the verbal learning phenomena mentioned earlier. Short-term memory, then, is one of the basic cognitive capacities. For our purposes, we assume that what gets stored in short-term memory are the internal names of chunks (e.g. "fianchettoed castled Black King's position"), which serve as memory addresses or retrieval cues for information about the chunks in long-term memory.

Let us return now to the chessboard construction phenomena. From Miller's chunking hypothesis, EPAM theory, and the limited capacity of short-term memory, we would predict that a chessboard can be reconstructed from information held in short-term memory if, and only if, it can be encoded in not more than about seven familiar perceptual chunks. If a single piece on a particular square constitutes a chunk for a subject, then he should be able to recall only about seven pieces. If he can recall the positions of more than twenty pieces, then it must be that each chunk consists, on average, of a configuration of about three pieces.

We now have a proposed explanation for the remarkable ability of chess masters to reconstruct positions – an explanation that meets our requirements of parsimony. We have employed only mechanisms that are well rooted in other parts of psychological theory: (1) a limited-capacity short-term memory that can hold the names of only about seven chunks, (2) a vast repertoire of familiar patterns stored as chunks in long-term memory, and a recognition mechanism – the EPAM net – for getting at them, and (3) the related chunking process that builds these patterns and their retrieval mechanisms in the first place.

The next task is to find more direct ways to test the theory. Several routes are open: we can seek direct empirical evidence for the existence of these chunks and see if the memory span for chunks is of the order of seven; we can attempt to simulate the reproduction task using the mechanisms of the theory within a computer program; and we can calculate whether the hypothesis leads to reasonable estimates of the number of familiar chunks a chess master must have stored in long-term memory. We consider these in turn.

*Empirical identification of chunks.*

The logic we used in isolating the chunks was to see if, during the reconstruction of a position, chunk boundaries could be identified by long pauses. Time measurements have been used for identifying chunks in other experimental tasks. McLean and Gregg, for example, had subjects memorize permutations of the alphabet (McLean et al, 1967). They then timed the intervals (latencies) between successive letters in the subjects' recitals of the lists. They obtained convincing evidence that the permuted alphabet was stored in memory, not as a single uniform list, but as a hierarchy of segments; the individual letter segments most frequently were three or four letters in length. Within-chunk latencies were much shorter than between-chunk latencies.

Adapting this technique to our task, we videotaped subjects reconstructing chess positions and measured the latencies in placing successive pieces. In order to estimate what interval would correspond to a chunk boundary, we performed a second experiment, in which the subject also reconstructed a chess position but with the original position in view. The two boards were so placed that the subject had to turn his head to look from the one to the other. We found that, when the subject placed two or more pieces on the board without turning his head, each latency was almost always under 2 seconds. We assumed that, under these speeded conditions, subjects load a single chunk into short-term memory when they view the board and then look directly over and recall that chunk. (It would be inefficient, under these conditions, to store more than one chunk, because they would then have to store the chunk names – there isn't enough room in short-term memory to store the structural information comprising more than one chunk – and then at recall use each chunk name in succession to retrieve the chunk from long-term memory – a time-consum-

ing procedure.) We therefore assumed that, in the reconstruction task, a pause longer than 2 seconds indicated the retrieval of a chunk from long-term memory via the chunk name in short-term memory.

To check the plausibility of this 2 second criterion, we counted the number of chess relations that held between pairs of successively placed pieces. The relations counted were attacks, defenses, proximity, identity of type (e.g. both Rooks or pawns) and color. There was a strong negative correlation between numbers of relations and latency (see Fig. 5).

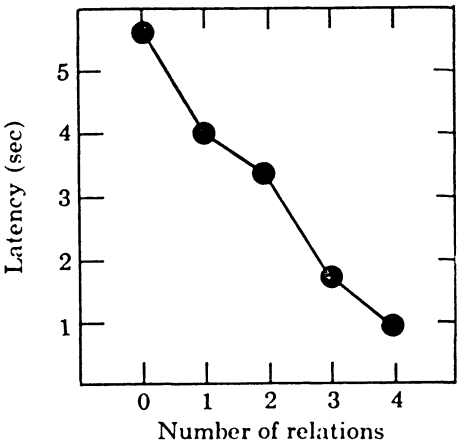



*Figure 5. Mean latencies between successively placed pieces in the reconstruction tasks are plotted as a function of the number of chess relations between the pieces.*

Next, we compared the pattern of frequencies of the between-chunk relations (greater than 2 seconds) with the pattern of the within-chunk relations (less than 2 seconds) and both of these with the pattern that would have been observed had the pieces been replaced in random order. We made this comparison for both forms of the reconstruction experiment — from memory and in sight of the board (see Table 1). For the two forms of the experiment, the within-chunk relational patterns were highly correlated (Pearson correlation coefficient of ·89), but these patterns were only slightly correlated with the corresponding between-chunk patterns (coefficients of ·12, ·18, ·10, and ·23) and not at all correlated with the random pattern (— ·04 and — ·03). On the other hand, the two between-chunk patterns were strongly correlated with each other (·91) and with the random pattern (·87 and ·81). Thus, there is strong evidence that the 2-second criterion in fact marks chunk boundaries.

*Table 1. Intercorrelation matrix for the Sight-of-Board Constructions (1 and 3), Memory Constructions (2 and 4), and Hypothetical Random Constructions (5).*

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1. Within chunk | | ·89 | ·12 | ·18 | —·04 |
| 2. Less than 2 sec | | | ·10 | ·23 | —·03 |
| 3. Between-chunk | | | | ·91 | ·81 |
| 4. Greater than 2 sec | | | | | ·87 |
| 5. Random | | | | | |

What was the nature of the chunks thus delineated? Most of them were local clusters of pieces in arrangements that recur with high frequency in actual chess positions. (The fianchettoed castled King's position mentioned earlier actually occurs in about ten percent of all recent games between grandmasters.) In the case of a subject who is a chess master, we were able to classify 75% of his chunks as highly stereotyped. Of the 77 chunks observed in his performance of the memory experiment, 47 were pawn chains, sometimes with a nearby supporting or blockading piece. Ten chunks were castled King's positions. Twenty-seven chunks were other clusters of pieces of the same color, and 19 of these were of common types: 9 consisted of pieces on their original squares in the back rank, and 9 of connected Rooks or connected Queen and Rook. These are configurations a chess master has seen thousands of times — as often as we have seen many of the familiar words in our reading vocabularies. There is as much reason to suppose in the one case as in the other that they are stored in his long-term memory and that he will usually recognize them when he sees them.

Thus far the empirical data support our theory, but we must mention one piece of evidence that is equivocal. If we accept the 2-second criterion for chunk boundaries, then we can measure directly the number of chunks our subjects are holding in short-term memory when they attempt to reconstruct the board. Our theory predicts that the number of chunks will be the same for strong and weak players, but that the average chunk size will vary by a factor of two or three with chess skill.

This prediction is not borne out fully.

When we compare, for example the data from the memory experiment for a chess master with the data for a Class A player, we find tht the master recalled about twice as many pieces as the Class A player, but the former's chunks averaged only about 50% larger than the latter's, while the average number of chunks he recalled also averaged about 50% more. The average sizes of the first chunks recalled by master and Class A player were 3·8 and 2·6, respectively; the average numbers of chunks per position were 7·7 and 5·7, respectively. Now the latter numbers are of the right order of magnitude – not far from the memory span of seven – but the difference between them is not predicted by the theory. At the moment, we have no good explanation for the discrepancy, but have simply placed it as an item high on the research agenda. Our hunch is that a less simplistic model of the structure of chunks and their interrelations, or of the organization of chunks in short-term memory, will be needed to attain a better second approximation.

*The MAPP simulation.*

A second approach to testing the theory of the chessboard reconstruction task was to build a computer program, MAPP, to simulate the observed phenomena (Simon et al, 1973). The general outlines of the program follow immediately from our description of the theory. The program contains a learning component to acquire and store in memory a large set of configurations of chess pieces and a performance component to carry out the board reconstruction task (Fig. 6).

Consider first the performance component. When a chess position is presented, the program must scan the board in some way in order to notice the pieces and their relation. The scanning program is a simplified version of PERCEIVER, hence can be viewed as a simulation of the eye movements and control of attention. When a piece is fixated (salient piece), an EPAM-like discrimination process seeks to recognize the cluster of pieces surrounding the fixated piece as a familiar chunk. If it is successful, the symbol designating this chunk is stored in short-term memory. This process is repeated at successive points of fixation until no more pieces become salient or short-term memory capacity is reached, whichever occurs first. Finally, in the reconstruction phase, the terminal information in the EPAM net is used to decode the symbols held in short-term memory into locational information for each of the pieces in a chunk and thus to reconstruct the position.

The learning component of MAPP is a simplified version of the portion of EPAM that grows or elaborates the discrimination net and stores information at its terminal nodes. The input to the program consists of many different configurations of pieces (of two to seven pieces each) that occur frequently as components of chess positions. If such a pattern has been familiarized previously, the program will simply recognize it; if it has not, it will discriminate it from patterns previously learned, will add tests to the EPAM net to implement the

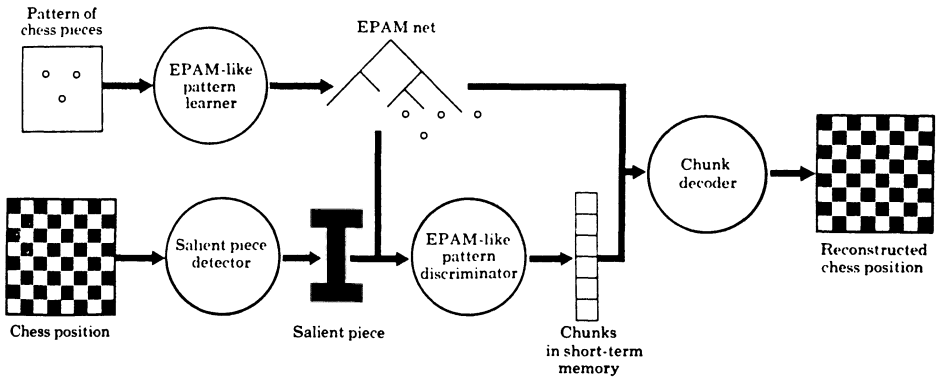



*Figure 6. A schematic representation of the principal components of MAPP shows the learning and performance processes used to reconstruct a chess position.*

discrimination, will create a new terminal node to designate the new pattern, and will store information about the pattern at that node.

Thus the MAPP program is a hybrid of a simplified PERCEIVER with a simplified EPAM; the finer details of those prior programs are not essential to demonstrating the phenomena. With a net of about 1,000 patterns, the performance of MAPP on the reconstruction task is about equal to that of a Class A player, twice as good as a beginner's but only half as good as a master's. In a typical set of positions, MAPP recalled 51% of the pieces placed correctly by the master, but only 30% of the pieces missed by the master, indicating that its chunks were not dissimilar from the master's. Finally, the within-chunk chess relations of pieces recalled successively by MAPP were highly similar to those of the human subjects, while the between-chunk relations were close to the random pattern.

*The chess master's vocabulary.*

We can extrapolate from the present performance of the MAPP program to estimate how large a vocabulary of chess patterns would have to be stored in the EPAM net to match the performance of the chess master. The distribution of different patterns by frequency is highly skewed, like the frequency distribution of words in natural language. Assuming that the patterns in the present MAPP net are those most frequently encountered in chess games, and assuming the same degree of skewness for chess patterns as for words, we can estimate that something of the order of 50,000 patterns would have to be stored to match the master's performance. Is this a plausible estimate from other viewpoints? We can check its plausibility in two ways.

First, there are no instant experts in chess – certainly no instant masters or grandmasters. There appears not to be on record any case (including Bobby Fischer) where a person has reached grandmaster level with less than about a decade's intense preoccupation with the game. We would estimate, very roughly, that a master has spent perhaps 10,000 to 50,000 hours staring at chess positions, and a Class A player 1,000 to 5,000 hours. For the master these times are comparable to the times that highly literate people have spent in reading by the time they reach adulthood. Such people have reading vocabularies of 50,000 words or more. If a chunk is a chunk is a chunk as to learning time ( as EPAM theory proposes), then we would expect the chess master to have a comparable chess vocabulary. Our estimate agrees well with that reached previously.

Finally, we may ask: given the variety of possible chess positions from well-played games, how big a vocabulary of patterns must we have so that each position could be represented by a distinct set of seven, or so, patterns? If $N$ is the number of possible positions, while $P$ is the number of patterns, then the requirement is $P^7 > N$. If $P = 50{,}000$, the $P^7$ is approximately $8 \times 10^{32}$. The latter number, in turn, is close to $6^{40}$. Now if we played chess games to a depth of 20 moves for each player and at each choice an average of 6 reasonable moves were available, approximately $6^{40}$ different games could be played. Since there are probably not, on the average, six reasonable moves at each choice point, 50,000 patterns should be more than enough to accommodate the positions that could be reached in such games. It should be emphasized that this estimate is very crude, since it does not take into account that some patterns are much more frequent than others. Nevertheless, it is reassuring that it gives results that are not inconsistent with those arrived at by other routes. Until we can get better data – possibly by expanding the EPAM net – it seems reasonable to assume that a chess master can recognize at least 50,000 different configurations of pieces on sight, and a grandmaster even more.

**Familiarity breeds competence**

If the MAPP theory provides an explanation – at least a first approximation – of the chess master's superior skill in quickly perceiving chess positions and then reconstructing them from memory, it leaves unexplained the link between this superiority and his chess playing prowess. How does the theory solve the riddle with which we began – that the statistics of the master's search appear indistinguishable from the statistics of the weaker player's search?

Two facts that have not been much

studied in the laboratory, but which are well known in chess circles, need to be mentioned. First, the master and grandmaster not only select good moves but they often – much oftener than weak players – notice these moves in the first few seconds after they look at a new position. Having noticed such a move, the master may continue to analyze the position for some minutes before he is satisfied that it is the best move – and sometimes his analysis will show that his first impulse was wrong. Nevertheless, his ability to notice moves "at a glance" is always astonishing to lesser players.

Second, although the average time per move in serious tournament chess is 3 to 4 minutes (which means that some moves are made rapidly, while others are brooded over for as much as half an hour), a master or grandmaster can beat players of inferior skill while taking only a few seconds per move and playing simultaneously against many players. His play in these games is not of the same quality as in his more deliberate tournament games, but it is strong enough to beat most experts and almost all players of lower class.

The most likely explanation of these facts is that the chess master is not only acquainted with tens of thousands of familiar patterns of pieces, but that with many of these patterns are associated plausible moves that take advantage of the features represented by the pattern (Chase et al, 1973).

Many of the basic heuristics that guide the search for good moves are based on the presence of a pattern on the board. For example, every chess player of even moderate skill is familiar with the advice: "If there's an open file, put a Rook on it." He knows that the advice is not meant quite literally, that what is really meant is "*consider* putting a Rook on it." The pattern of an open file will trigger the heuristic and initiate a move in the heuristic search. Some patterns (perhaps many hundreds) may actually be associated with an algorithmic solution – traps and combinations that lead to the guaranteed win of a piece, a checkmate, or whatnot – in which a series of moves may be played almost by rote.

Thus, we suggest that the key to understanding chess skill – and the solution to our riddle – lies in understanding these perceptual processes. The patterns that masters perceive will suggest good moves to them. The structure of the search process through possible moves will not be very different from that of weaker players; only the paths suggested by the patterns will be different.

Such a view of chess skill is quite amenable to theorizing in terms of production systems. By a *production* is meant a routine consisting of two parts: a *condition* part and an *action* part. The condition part tests the presence or absence of a specific (perceptual) feature (e.g. an open file); the action part, which is executed whenever the condition is satisfied (whenever the feature is recognized as being present), generates a chess move for consideration that is relevant to that specific feature (e.g. putting a Rook on the open file). A separate analysis routine can then carry out the tree search required for a final evaluation of proposed moves. The advantage of modeling human behaviour with production systems is that such systems are very simple and rulelike, avoiding many of the inflexibilities of algorithmic programming languages. They can mimic learning by simply adding new productions (Newell et al, 1972), and they have the perceptual flavor we need to simulate the pattern-recognition processes in chess.

While the evidence is not yet in, it becomes increasingly plausible that the cognitive processes underlying skilled chess performance have some such organization as this. Such a scheme would account for the association of chess-playing skill with the ability to recognize numerous perceptual patterns on the board.

There is another question which we haven't addressed directly, but whose answer is implicit in what we have been saying. The question is: how does one become a master in the first place? The answer is *practice* – thousands of hours of practice. This is implicit in the EPAM theory; what is needed is to build up in long-term memory a vast repertoire of patterns and associated plausible moves. Early in practice, these move sequences are arrived at by slow, conscious heuristic search – "If I take that piece, then he takes this piece ..." – but with practice, the initial condition is seen as a pattern, quickly and uncon-

sciously and the plausible move comes almost automatically. Such a learning process takes time – years – to build the thousands of familiar chunks needed for master level chess.

Clearly, practice also interacts with talent, and certain combinations of basic cognitive capacities may have special relevance for chess. But there is no evidence that masters demonstrate more than above-average competence on basic intellectual factors; their talents are chess-specific (although World Champion caliber grandmasters may possess truly exceptional talents along certain dimensions). The acquisition of chess skill depends, in large part, on building up recognition memory for many familiar chess patterns.

We now have an account of perceptual skills in chess that is consistent with theories drawn from other parts of psychology. There is no lack of tasks for continuing research, and the environment of chess continues to be one of the most fruitful for cognitive studies.

# 5.3 Decision Making and Computers

**Mikhail Botvinnik**

First published in *Advances in Computer Chess 3*, (M. R. B. Clarke, editor), Pergamon Press, Oxford, 1982, pp. 169-179. Reprinted by permission of Pergamon Press Ltd.

**Abstract**

The method of the chess program PIONEER is described, and use of the method in a wider class of decision making problems is discussed. Successful application to an industrial equipment repair problem is described.

## Introduction

Presumably the principal component of the concept of intelligence is decision making, but the decisions have to satisfy very stringent requirements: they must be intelligent. Decisions are not necessarily intelligent and one is entitled to regard the problem of intelligence as constituting only a part of the more general problem of decision making. Let us clarify this with an example.

Figure 1a is a schematic search-tree for an arbitrary chess position. If the maximum depth of variations is 3 ply the move that will be selected in these conditions will inevitably be so weak that the play cannot be regarded as intelligent, at least from the standpoint of a skilled chess player. If the depth of variation is increased to 15 ply the play may conceivably be considered intelligent; assume that line C marks the agreed lower bound on intelligence. With 30-ply variations play is very much above the lower bound of intelligence.

In Figure 1b we schematically plot the limiting depth of a decision as a function of available resources: curve A corresponds to non-optimal use of resources, and B to their optimal use. Two points are marked on the abscissa: point 1 corresponds to thelevel of resources at which the intelligence bound is reached optimally, and point 2 to the level of resources at which the same bound is reached by employing the resources non-optimally.

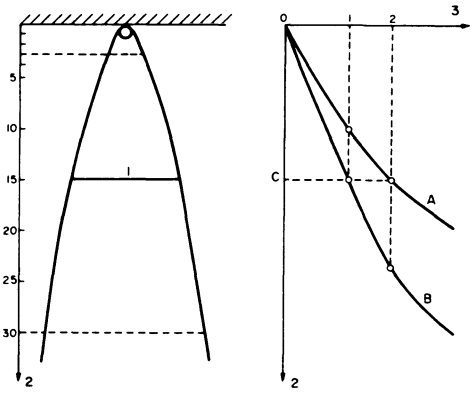



*Figure 1a (left). Schematic chess search tree. Figure 1b (right). Search depth as a function of resources: (A) non-optimal and (B) optimal. (1) Lower bound on intelligent solutions: (2) search depth, ply; (3) resources.*

The level of intelligence is thus reached in both cases. But the decision obtained by the non-optimal method *is not* intelligent because if the same resources were used optimally line C could not be considered as the intelligence level.

The level of intelligent decision is a function of the available resources with which it is made, the optimal use being mandatory. Given this approach, no computer chess program is intelligent if a non-optimal brute force search is employed.

## Enhancement of Resources by Specialisation

The resources of a computer or human nervous system can be augmented via specialisation, that is adaptation to a specific program of actions. The processor specially built by K. Thompson and J. Condon for the BELLE chess program is a unique achievement in this direction. A human being follows the same principle

when he adjusts (specialises) his nervous system to a predetermined sequence of actions, including chess playing. The result is a considerable gain in the speed at which the program functions; in other words the resources are augmented. The nervous system is apt to specialise more efficiently in a young person, so that achievements in creative activities are the greater the earlier one starts to master a specific field, for instance chess or piano playing. Likewise, a computer too must be specialised "early": preferably before the actual development.

Specialisation can be expected to yield best results only if the program makes optimal use of resources. One must start by elaborating the methodology and program and only after this begin designing a computer adjusted to the program utilising the optimised method. A chess player also refines an optimal way of employing his nervous system, adjusting (specialising) it and thus augmenting the resources available for solving a given problem.

**The Chess Master's Method: Level and Universality**

One of the profound ideas of Shannon (1950a), who with remarkable insight suggested examining the method of computerised decision making via chess simulation (at that time computers were still in their infancy), was that one possible approach is to use the method by which a chess master chooses his move (although Shannon mentioned certain qualifications). The point is that the method of search in reaching a decision must especially be highly sophisticated in two-sided games, otherwise the chances of winning become infinitesimal. In a one-sided game (for example, planning the economy) the method need not be so sophisticated. A chess master in action is a focus of the high quality both of the method employed and of specialisation of the decision-making system. Both these facets are tested and refined by a strong chess player while he works his way up the chess Olympus through the fire and brimstone of tournaments. Today the problem of whether the chess master's method is universal (for multidimensional search problems) still remains debatable. I side with Shannon in expecting that the universality will be proved by practice.

**The Chess Master's Method: Formalisation**

This is the aspect raising the greatest objections. A popular cliché states that reasoning in general, and the chess player's reasoning in particular, is a mystery beyond human mind. This opinion seems to stem from the fact that human thinking processes are not yet understood; it should be admitted though that this offers only a poor foundation for scepticism.

The work on comprehension, formalisation and programming of the method used by a chess master could only start with the assumption that final success is possible. The study is still far from completion but the results obtained so far enable us to be optimistic about the future.

Obviously, the main purpose of the work is not merely to have a strong chess computer but to arrive at a general technique of solving search problems. We are not too far from this goal and hope that once the master-computer based on the method of the master-player is realised this general method will also be formulated.

**The Chess Master's Method: The Principle Elements and their Generalisation**

The method is based on the idea of inexact goals. When a problem is being solved with a restricted depth of variation, that is with search tree cut off, the exact game goal (mate in chess) is useless as far as meaningful search is concerned. Furthermore, there exist problems which are inexact in principle, having no exact game goal, for example in planning. In the case of chess, an inexact goal is gain of material and in the planning of equipment repair schedules it is repair of a unit. Although the goals are simple the role they play is important.

Elementary goals (targets) may be numerous. In order to restrict their number, a chess player resorts to the method of "horizon": he takes notice of only those targets which can be hit within a time interval not exceeding the horizon.

At first glance a chess game appears to form a hierarchical structural entity. However, this approach will not lead to an economical solution. A chess master simulates the game as a three-level model, thereby facilitating his decision making. Level 1 is a chess piece attacking an opponent's piece along an attack trajectory;

level 2 is level 1 supplemented with pieces hindering the attack and pieces supporting it; level 3 (the highest) is an ensemble of second levels whose participation in the search can affect the choice of optimal variation. Level 3 is called the mathematical mapping of a position (MM).

In level 2, also called "zone of play", the opponent's pieces which could hinder the attack may be noticed too late when they cannot reach the battlefield, the square in question. The following procedure, typical for the method, is then used: the variation is traced back to the node from which the square is reachable for the piece. This unevaluated branch of search along which the procedure ascended is then erased from the computer memory; hence use of the term "pseudo-search" for this procedure.

The described algorithm processes both the variation break-off and the branch cut-off unconventionally. To the break-off associated with the limiting depth of variation it adds a meaningful (GIG-oriented) earlier break-off that is executed when expectation of regaining what was lost earlier has to be forsaken. Likewise, branches are cut off if there is no hope of improving the current optimal variation.

The leitmotiv of the algorithm is the trinity principle: hope, timeliness and maximum gain. Once we lose hope that a piece will arrive in time to take part in the struggle or that the gain may be increased, the piece is eliminated from the analysis.

Such a multilevel procedure enables one to achieve a significant reduction of the search because the program can decide about breaking off within a level, about eliminating levels, and about the sequence in which levels are included into the search.

This system of priorities, the one we call the *strong* priorities approach to including levels and pieces into the search, is the most important element of the algorithm. All chess programs operate with priorities of moves but the question is: what is the quality of assigning the priorities? The system of priorities in our PIONEER program seems to fit best that of the chess master.

Before discussing the priorities assigned in PIONEER we should consider the evaluation function, or rather that component of the function which calculates the *positional* value, which is closely linked with the priorities assigned to pieces entering the game. The positional value is of special importance in the terminal nodes of variations because it makes possible a prediction for the future for which no variations are yet available.

A considerable number of tentative "estimates" for a formalisation of the positional value proved ineffective; but we finally feel we are on the right track. This new apporach has not been tested yet but the positional component will hardly need serious modifications. Consider the scheme given in Figure 2. This represents a group of pieces and their trajectories. The diameter of a circle representing a piece is proportional to the material (estimated) value of the piece. Widths of lines tracing the trajectories are proportional to the passabilities of these trajectories; in Figure 2a the width is the same for all trajectories since their passability is not yet known. This passability depends in turn on the sign of the net result of possible exchanges on the squares along the trajectory. Comparing pieces to ladybugs, a piece participating in an exchange gets a "spot" from each exchange.

In Figure 2b we retain the material values of the pieces but change the widths of trajectories determined by the material passability of the trajectories.

Once the material passability of a trajectory is found we calculate its material priority $P_m = mW_m/t$, where $W_m$ is the material passability of the trajectory, $m$ is the value of the target, and $t$ is the time of advance along the trajectory. In the limit (directly before capture), $W_m = t = 1$ and $P_m = m$.

The numerical value $P_m$ of a trajectory is assigned to the spot on a piece associated with exchange on the given trajectory. By adding up all spots on a piece ($\sum P_m$) we obtain a current situational piece value corresponding to the real strength of the piece at a given moment of struggle on the chessboard.

This reassessment is reflected in Figure 2c. The width of trajectories is so far the same as in Figure 2b, but the diameters of piece circles are altered: now they correspond not to the material but to the current values of the pieces.

With the current values calculated we can find possible exchanges of these values on the squares of the trajectories and arrive

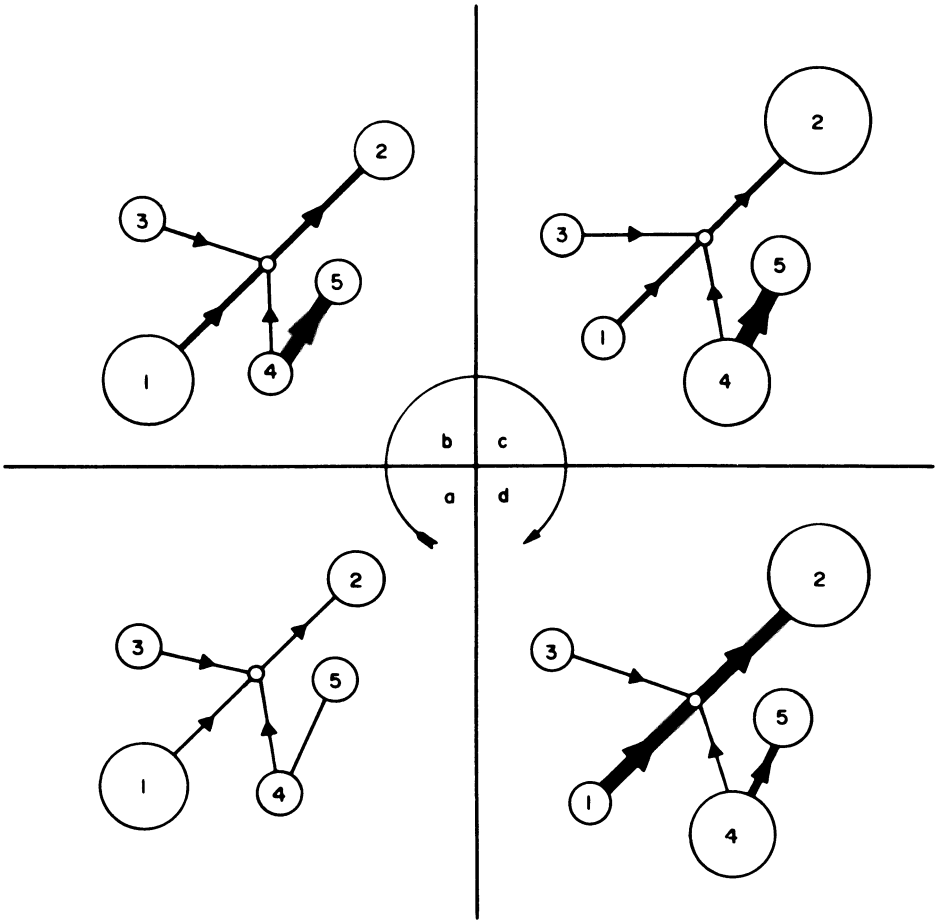



*Figure 2. Diagram of calculation of priorities of moves for the search (after B. Stilman).*

at the current passabilities of the trajectories (see Fig. 2d). The next step is to determine the current priority of each trajectory, $P_c = cW_c/t$, where $W_c$ is the current passability and $c$ is the current value of the target.

Finally the positional component of the evaluation function is obtained by adding up the values of $P_c$. Two types are possible.

First consider a chain of trajectories in which the first link is an attack trajectory with a single impassable square, the second link is a trajectory connected to this square also with a single impassable square, and so on, the last link being a *passable* trajectory connected to the impassable square of the last-but-one link. Let, for example, the number of links equal three. If the advance of a piece along the third trajectory makes the second one passable, and the advance along the second one makes the first one passable, the chain is considered passable. These passable chains of trajectories are taken into account at the final node of a variation when considering a positional sacrifice, a sacrifice of material for positional gains. Secondly, if there are no passable chains, the positional advantage is not compared to the material and the program struggles to transform impassable chains into passable and vice versa.

The evaluation function thus comprises (a) a material component and a positional component compared to the material, and (b) the positional component which is important in the case of material equilibrium.

Let us return to the assumed system of priorities. This general current priority

enables the program to select the first variation, one that is *quite close* to optimum. This aspect constitutes the strength of the chess masters's method: a multidimensional search problem degenerates to an almost zero search problem. Actually, calculation of the first priority variation must be followed by a test: could the value be augmented, does the variation differ from the optimal one? This test is done by ascending the variation. The branches of the subtree are analysed below each node of the tree. PIONEER moves along these branches as along rails (the corresponding sub-routine was called "follow the rails") and thus accumulates the information which enables it later to single out meaningless moves, those to be cut off at the nodes, and to assign priorities to the remaining meaningful moves. "Meaning" here is associated with the expectation of a variation with higher value.

The branching factor obtained thereby is nearly unity, in contrast to brute force search where this factor is always much larger than unity. These methods are thus completely different.

However, the calculated priority is not a rigid command. If it were a chess master would be guided only by this, and be sure of success. But in some chess positions the general priority evaluation gives a misleading recommendation resulting in an increased branching factor and excessive spreading of the search tree. In such cases a chess master resorts either to his own or to other masters' accumulated experience. These libraries of accumulated knowledge file the positions, in full or as excerpts, in which the general priority assignment proved ineffective. In choosing a move for the search the master, in addition to calculating the general priority, looks up the position book and checks whether a similar situation has been encountered before. If it has the master makes his final decision concerning the move in question by comparing the move corresponding to maximum priority with that actually made in the book of positions. Obviously, this approach is ineffective in keeping the branching factor close to unity if the general priority recommendation is erroneous and no past experience is available. The past experience book must therefore be as extensive as possible and be used both by master and computer. It is proper to add that middle and endgame books are no use in the brute force approach since addressing them means a vast expenditure of resources because of the tremendously large search trees. The master (and PIONEER) operates with only several tens of nodes in the search tree and thus is able to resort to accumulated knowledge in order to correct the general priorities.

The algorithm of the chess-playing PIONEER is easily generalisable. The concepts of the goal of inexact game, meaningful cut-off of variations and branches, trinity principle, multilevel approach, pseudosearch, positional assessment, general priority and book access check on optimality of a variation may be transferred to application-oriented multidimensional search problems with a view of transforming them to nearly zero search.

We believe that the chess method is sufficiently universal. I have mentioned that Shannon offered this opinion as early as 1950, pointing out in his paper on chess programming eight promising fields for the application of the chess method.

## Testing the Method

PIONEER was tested on a sufficiently complicated position (from the standpoint of a master) from a Botvinnik-Capablanca game, Rotterdam 1938 (Figure 3).

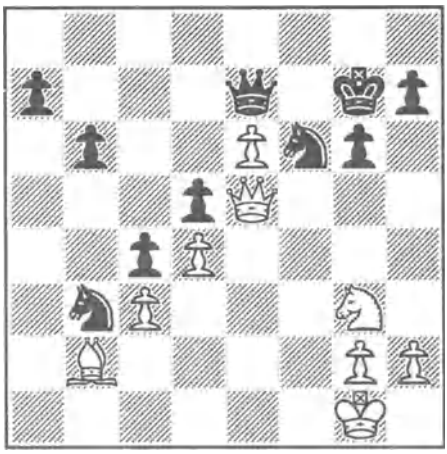


*Figure 3. Botvinnik-Capablanca, Rotterdam 1938.*

Figures 4, 5 and 6 show three search

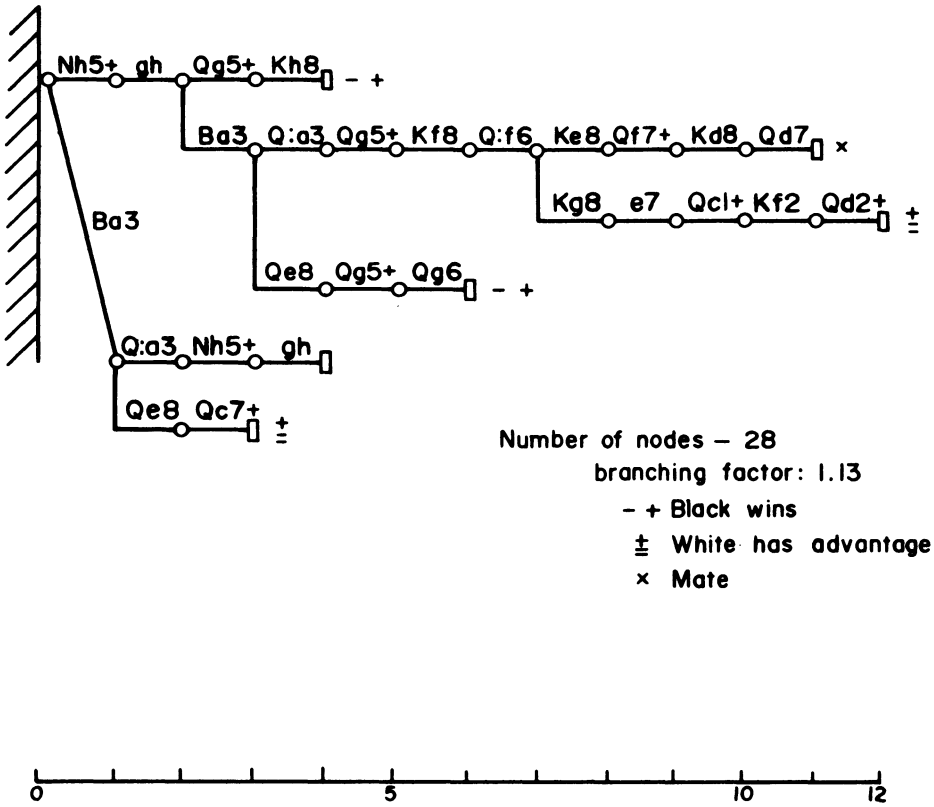



*Figure 4. Search tree of the player.*

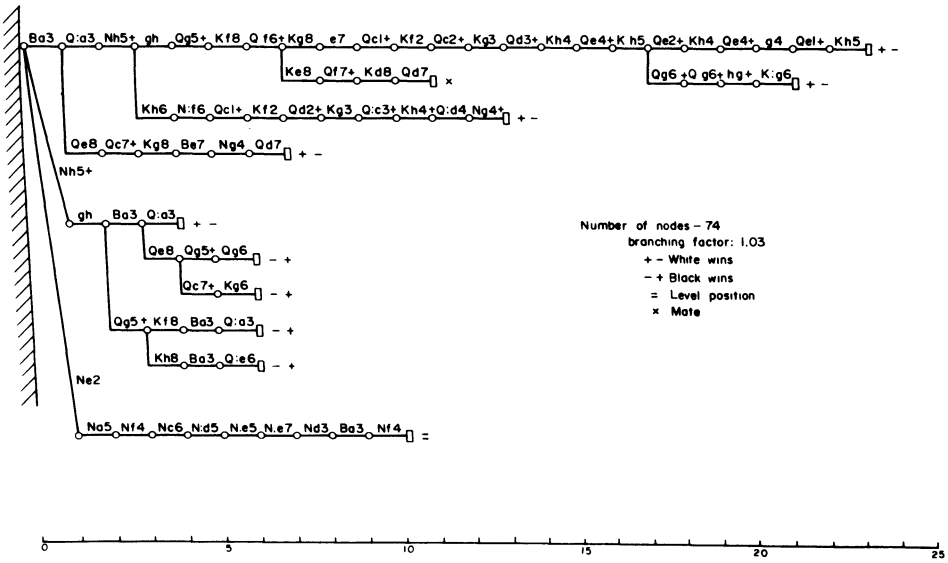



*Figure 5. Search tree of chess annotator.*

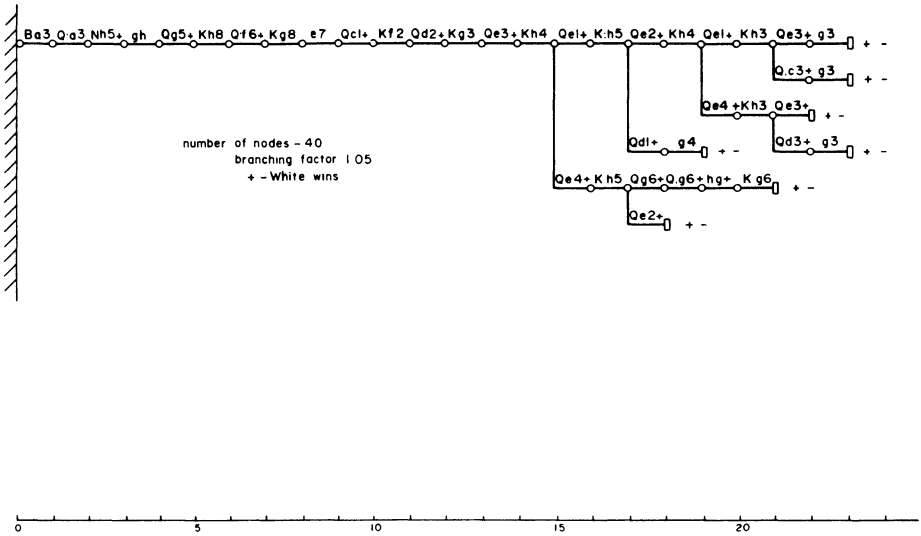



*Figure 6. PIONEER's search tree (under restriction on search time).*

trees: Figure 4 composed during the game and reconstructed from memory, Figure 5 based on an analysis I published after the game, and Figure 6 traces by PIONEER under time restriction conditions. The three trees are similar: note that the master's depth of variation is almost half that of the computer, with a correspondingly smaller number of nodes. The depth of variations is identical in the computed and in the annotated search trees but the annotator included more nodes since he was not working under time restrictions. The branching factor is close to unity in all three search trees.

The positional value was not tested; at the time is was not completed and, moreover, the play in the test example is forced so that this assessment was unnecessary.

We also decided to test the method on an applied search problem, namely the monthly planning of industrial equipment repairs. In terms of dimension this problem is at least commensurate with chess. Note that it is a distinctly one-sided game. However, the problem as it was formulated proved much simpler than chess. "Pieces" (units) were of one type, differing in power and time assigned for repair. Repair work had reliable resources; trajectories (repairs) are "passable", and the only restriction was on the power of the simultaneously repaired units, with a "rank" being assigned to each unit reflecting the priority with which it was to be included into the repair schedule. The main simplification was a fixed depth of planning (depth of variation) from the first to the last day of the month. The program did not analyse the situation for the following month and so the positional assessment was unnecessary.

The monthly planning proved to be quite successful, once the general method was formalised for the problem, and the algorithm and the program were compiled. The priority variation consumed about one per cent of the time necessary for chess problems. The branching factor was also nearly unity; the problem became search-free.

## Prospects of the Method

Since the universality of the "chess" approach to search problems has been tentatively confirmed, we expect PIONEER to be efficient in other applied fields, and first of all in economics.

One can obviously object that the method has not been fully formalised: indeed a computer cannot yet determine the set of concepts required by the method for a specific problem. This is true, but it is hardly expedient to put off work on this problem until the necessary experience has been accumulated.

We are thus hopeful that application of PIONEER to economics will be fruitful; and is economics not mankind's first problem?

# 5.4 A Chess Mating Combinations Program

**George Baylor and Herbert Simon**

First published in *Proceedings of the Spring Joint Computer Conference, 1966,* pp. 431-447. Reprinted by permission of the American Federation of Information Processing Societies Inc.

## 1. Introduction

The program reported here is not a complete chess player; it does not play games. Rather, it is a chess analyst limited to searching for checkmating combinations in positions containing tactical possibilities. A combination in chess is a series of forcing moves with sacrifice that ends with an objective advantage for the active side (Botvinnik, 1960). A checkmating combination, then, is a combination in which that objective advantage is checkmate.† Thus the program described here – dubbed MATER – given a position, proceeds by generating that class of forcing moves that put the enemy King in check or threaten mate in one move, and then by analyzing first those moves that appear most promising.

The organization of this paper centers around MATER's ability to analyze chess positions. After a brief look at the program's history in Section 2, the overall organization of the program is presented in Section 3, an organization which is designed to allow flexible movements in an analysis tree of possibilities. Then in Section 4 the "top level" comes under consideration; MATER's heuristics of analysis – the search rules and priorities, the search evaluators – that enable it to find a mate in the maze of possibilities. In Section 5 data on the performance of the program is presented. Finally, the programming language representation of the chessboard and chess pieces and the basic chess capabilities this affords are reported in the Appendix (they are reported in yet finer detail in Baylor, 1965).

## 2. History of the Program

MATER has led a checkered life. The original mating combinations program, as conceived by Simon and Simon (1962) was a hand simulation setting forth a strategy of search. According to hand simulations the program discovered mating combinations in 52 of the 129 positions collected in Fine's (1952) chapter on the mating attack. But a hand simulation is not a rigorous model and, as such, is itself sometimes prone to the imprecisions and ambiguities of many verbal theories. Indeed, Myron and May (1963) pointed out two such ambiguities in the specifications laid down by Simon and Simon (1962).

Newell and Prasad (1963) coded an IPL-V program (1963) which set up a chessboard, recorded positions, made moves, tested their legality, and performed a few other functions (see Appendix). This they overlaid with the beginnings of a mating program.

The first version of a mating program described here – MATER I – is the work of G. Baylor (1965). H. Simon and P. Simon (1962) using the Newell-Prasad (1963) chessboard. The program was revised during 1964 by Baylor into a second version, MATER II.

## 3. Organization of the Analysis Tree

*The Problem*

As stated in Section 1 the mating program analyzes chess positions. An analysis of a position – as the term is used here* –

†Sometimes the defender is able to avert the checkmate by incurring a heavy loss in material (pieces and/or Pawns). If the attacker's gain in material is indeed an "objective advantage" – the defender being left with no compensatory attacking chances – then such combinations would generally be called mating combinations, even though not ending in mate. The current version of the program confines itself to mating combinations in the narrow sense – those from which there is no escape. Inclusion of the broader class is an obvious extension.

*Chess players would probably prefer to define "analysis" as the finished product rather than the process of search, laying stress on the "right" moves and continuations rather than emphasizing how these were arrived at.

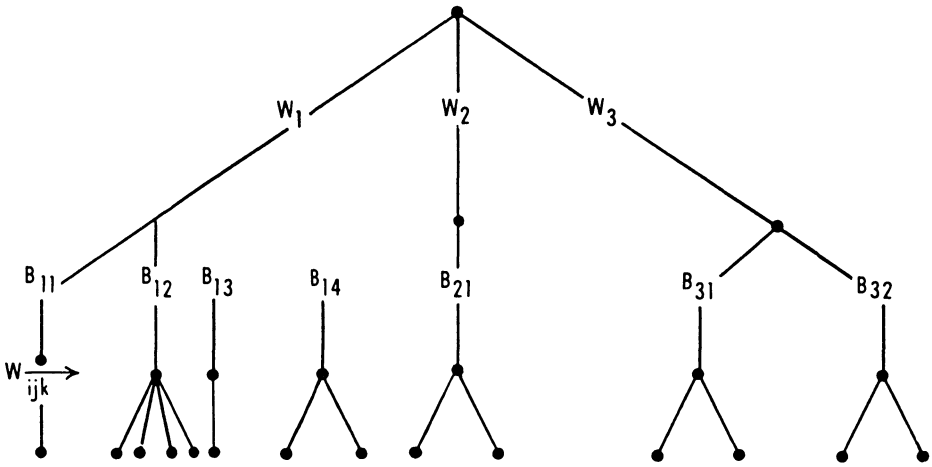



*Figure 1. The Analysis Tree (The W's are White moves, the B's replies.)*

consists of the set of moves and evaluations made in the course of resolving the choice-of-move problem. Taken together, moves and positions make up a tree of possibilities in which moves operate on positions to produce new positions (see Fig. 1) and on which evaluations of positions and of moves in achieving desired positions can be hung as desired.

The dots or nodes in Fig.1 denote positions (static states) and the lines between dots denote moves (operators) that transform one position into another.†

How should the analysis of a position be conducted? Figure 2 presents one simple scheme. Would this scheme be workable if it were made operational? For example, let:

1. "X" be defined as checkmate and the program be given the capability of generating moves in its service;
2. the criteria for deciding among moves be specified by certain rules of selection:
3. the program be given the capability of making moves and updating board positions:
4. a test be provided so that the program "knew" if it had achieved "X": and

5-8. the corresponding provisions be made for "Y", defined as escaping check, and for choosing among replies.

The answer is no, not quite. The scheme lacks a means for recovering from false starts, for retracing its steps when it runs into blind alleys. Indeed, what is lacking can be seen by considering the difference between actually playing a game of chess and analyzing a chess position: the course of analysis is fickle and reversible, whereas in an actual game a move one made cannot be unmade. In other words, the scheme outlined above needs provisions for unmaking moves and for abandoning seemingly unpromising positions as much as it needs the capability of making moves and pursuing promising positions. Ideally, one should like to be able to enter and reenter the move tree at any node (position) at any time and from there to proceed down any branch, old or new. Providing such capabilities for reinstating the right position at the right time is probably the central problem of organization at this level of the program, while making operational and making sense out of steps 1-8 above is probably the central problem at the next higher conceptual level. This section reports on the former problem: implementation of a flexible move tree. Section 4 is devoted to the latter: defining the problem and the heuristics of search.

†Simon and Newell (1956) have often drawn this difference equation analogy to the problem-solving process: given an initial state description and a desired state, the problem is to find a process description that operates on the initial state to produce the desired one. In discussion of the Logic Theorist (Newell et. al., 1962) for example, the logic expressions correspond to the static states, and the rules of inference, to the operators.

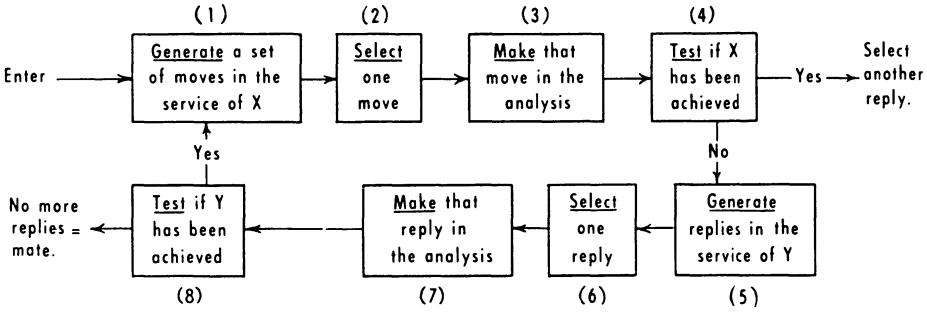



*Figure 2. A simple recursive mating scheme.*

**Building the Tree**

The notion of analysis as a tree search is misleading to the extent that it implies that each step consists solely of selecting a move from the many available alternatives. Actually, the process is more one of generating moves as one goes along, of building one's own tree. This is the very distinction Maier (1960, p.218) has drawn between decision making under conditions of uncertainty* and problem solving: "Decision making implies a given number of alternatives, whereas in problem solving the alternatives must be created. Thus, problem solving involves both choice behavior and the finding or creating of alternatives."

In every chess position, of course, the rules of the game place an upper limit on the number of possibilities that can be created; de Groot (1965) found that, averaged across the course of a game, the mean number of legal move possibilities lies somewhere between 30 and 35. This is a full-grown tree; the one the searcher actually builds is much smaller: on the average, four or five branches at the top node and smaller thereafter.

The first question addressed in this section is the technical one: How does one build a tree? How are limbs and nodes – moves and positions, respectively – structured into a tree of possibilities? (See the Appendix for the details of construction of moves and positions.) Second, and pursuing the metaphor, think of the chess player climbing the tree as he builds it. Crawling along a branch in one direction corresponds to making a move in the current position (node) while traversing it in the opposite direction corresponds to unmaking a move and restoring the previous position (node). This ability to back up the tree is what enables a player to abandon unpromising lines of investigation and start afresh. In starting afresh, moreover, the player may either reinvestigate either branches he has previously built or build new ones.

Third, as the player builds and climbs he also accrues and retains information. The information garnered en route and the use to which it is put are in large part what *Denk* psychologists have called the development of the problem. That is, the searcher's conception of the problem at any one time consists of the information he has about the problem, how he has evaluated this information, and even how it has shaped his definition of what the problem is (cf Duncker, 1945; and de Groot, 1965). Provisions for gathering information are considered in both this section and the next; the use to which information is put and the matter of problem development are more properly treated in the subsequent sections.

Most of the organizational problems are solved via the description lists of moves. For convenience of reference the entire list of possible attributes a move can take on is set forth in Table 1.

With respect to the first question – provisions for holding the tree together – a signal cell and attributes A46 and A47 to do the job. The signal cell contains the name of the most recent move made on the

*According to Luce and Raiffa (1957) decision making under uncertainty is the condition in which the outcomes of the various known alternatives are unknown.

*Table 1. Move Attributes A40 – From square*
A41 – To square
A42 – Special move (yes or no)
A43 – Man removed from castle list
A44 – Man captured
A45 – Square of captured *en passant* Pawn
A46 – Ancestor
A47 – List of descendants
A48 – Irreversibility of move (reversible; or irreversible)
A49 – Value of move (mate, no mate, no value)
A50 – Number of descendants
A51 – Man moved
A52 – Double check (yes or no)
A53 – Discovered check (yes or no)
A54 – Checking move (yes or no)
A55 – Descendant's list of mate threats
A56 – Threatened mating square
A58 – Mating piece on V(A56) square
A63 – Reply NOMV (no move)
A65 – Number of checking moves generated to date
A66 – Number of replies generated to date
A70 – List of King replies
A71 – List of capturing replies
A72 – List of interposing replies

board – the contemporary – while attributes A46 and A47 are its ancestor and its list of descendants, respectively. The log of the analysis is preserved by defining a dummy move, L31, which has on its description list the list of descendants attribute A47. Thus the course of analysis is linked together as a chain of moves with contemporaries linked by ancestor and descendant relations, as in the example of an analysis tree in Fig.3.

Second, because of the strong family ties just described, one can eventually crawl one's way down any branch of any node and then back up. That is to say, one can make or unmake any move. Two routines. E65 and E66, make and unmake moves, respectively. The procedure for unmaking a move is exactly the reverse. With the help of routine E11 the position list is restored, that is, the description lists of the pieces and squares affected are restored and the signal cells reset.

At any node a new limb may also be constructed (by routines E51 and E52; see Appendix) simply by specifying the "from square" and the "to square" (and special move status, if any) whereupon the move is added to the list of descendants, V(A47), and assigned an ancestor, V(A46).

Third, information gathered in the search for mate is stored on the description list of the move that gathers it. (See Table 1.) When a move is constructed, its ancestor is always assigned as a value of A46 and the man moved is always assigned as a value of A51. Conditionally, a move is assigned a value of A43 if a Rook or King is removed from the castle list, a value of A44 if a man is captured, a value of A45 if a Pawn is captured *en passant*, a value of A52 if it is a double check, a value of A53 if it is a discovered check, and a value of A54 if it is a checking move at all.

Evaluative information is also gathered: attribute A49 records the win-loss value of a move: mate, no mate, or no value. If a checking move has no descendants, that move mates; consequently, attributes A70, A71, and A72 record the kinds of replies to check. Attribute A47 lists the descendants in toto and the value of A50 is a count of them.

The point here is to illustrate how information is hung on the move tree as it is gathered. How the information is retrieved and utilized is a topic for the next section.

## 4. The Executive and Heuristics of Search

### *Introduction*

In a given position, what moves should be considered and in what order? Human chess players are known to be highly selective in the moves they look at, a selectivity based on their heuristics of search. Computers must also incorporate such selectivity. What follows then is a discussion of the search heuristics incorporated into the early version of the mating program, some measures on its search behaviour, a brief description of the routines that effect the move generation, and, finally, later developments added to a second version of the program, MATER II.

### *MATER I*

Restricting the mobility of the opponent's pieces is a recognized principle of chess strategy. It is particularly important in checkmating combinations since checkmate is defined as an unopposed attack on an enemy King whose mobility has been reduced to zero. Strategically this means the attacker strives to gain control over 1) the square enemy King occupies, as well as 2) all the squares contiguous to it that do not contain an enemy piece. If just

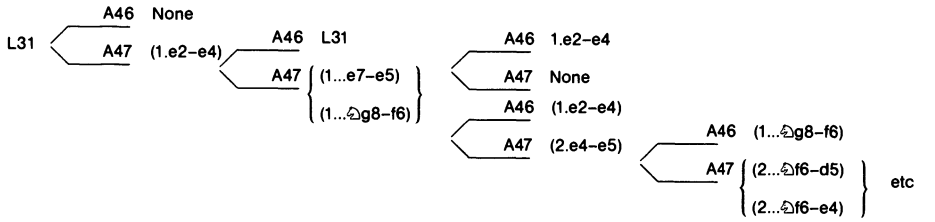



*Figure 3. Course of analysis linked as contemporaries, ancestors, and descendants.*

condition (1) obtains, the enemy King is simply in check: if just condition (2) holds, the enemy King is stalemated; while if both (1) and (2) hold, he is checkmated. Viewed in this light, checkmate is a process of acquiring controls or more and more restricting the enemy King's mobility. This principle is the cornerstone of the mating program.

The restriction of mobility principle applies to the generation and selection of moves as well as to decisions about when to abandon search in certain directions. Thus in the mating program: Only checking moves (in MATER I) and moves that threaten mate in one move (in MATER II) are generated for the attacker: the move selected for investigation is the one that most restricts the opponent's mobility; and search is continued down a chosen path only so long as the opponent's mobility is on the decline. This – the rate of growth of the search tree – is an alternative formulation to an evaluation function for terminating search in a particular direction.

Before illustrating the flow of control and the program's executive structure, it is necessary to introduce the notion of a try-list, a notion similar to the "pool of subgoals" in the Logic Theory Machine (Newell, Shaw and Simon, 1963). Since only one move can be tried out at a time in any particular position, other "eligible" checking moves must wait their turn on a list, L35 – the try-list. This list has two noteworthy properties: first of all, it is an ordered list, and second it is independent of a move's level.* Such independence proves powerful in directing search.

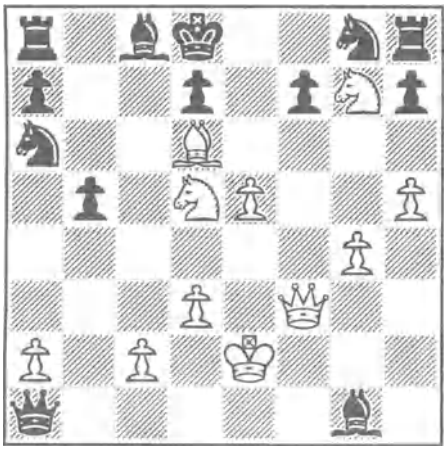
The list is ordered by a fewest-replies heuristic: highest priority goes to moves with the fewest number of legal replies, while checking moves with more than four legal replies are discarded entirely. Ties are broken by giving priority to double checks, then to checks that have no capturing responses, then to the order in which the checks were generated. The second property – that checks from all levels are mixed – effects the evaluative principle that search is continued down a particular path only so long as the opponent's mobility is on the decline; when the number of replies at some node in the current line of investigation is equal to or greater than the number of replies at some prior node, the current line is abandoned, the prior node restored, and the alternative that had once been passed over is tried. This nips in the bud unpromising proliferations in the move tree.

In addition to the notions just described – rate of growth serving to terminate search in a particular direction, the set of considerable moves serving to restrict the set of applicable operators in the given position, the try-list ordered by the fewest-replies heuristic serving to stipulate the application of offensive operators – some heuristics of chess strategy serving to stipulate the order of application of defensive operators can also be seen more clearly from the following illustrative position taken from Fine (1952) and MATER I's performance on it.†

The following layout is adapted to the simple recursive scheme of Fig.2.

*The level of a move refers to its depth in the move tree, i.e., how many moves out it is from the starting positon.

†They can also be seen more clearly within the picture of heuristic search in general in Newell and Ernst (1965).



(1) Generate checking moves: 1 ♘g7-e6+; ♗d6-e7+; ♗d6-c7+; ♕f3-f6+
(2) Select one move for further analysis:

(a)
1 ♘g7-e6+
1 ... ♔d8-e8
1 ... d7×e6
1 ... f7×e6
3

1 ♗d6-e7+
1 ... ♘g8×e7
1

1 ♗d6-c7+
1 ... ♘a6×c7
1

1 ♕f3-f6+
1 ... ♘g8×f6
1 ... ♘g8-e7
2

(b) Transfer checks to try-list; order them by their "u" values (number of replies)

| L35 | u |
|---|---|
| 1 ♗d6-e7+ | 1 |
| 1 ♗d6-c7+ | 1 |
| 1 ♕f3-f6+ | 2 |
| 1 ♘g7-e6+ | 3 |

(c) Select and delete top move from try-list L35: 1 ♗d6-e7+

(3) Make that move (1 ♗d6-e7+) in the analysis.
(4) Test if checkmate has been achieved: No.
(5) Generate replies to relieve check 1 ... ♘g8×e7
(6) Select "best" reply; 1 ... ♘g8×e7
(7) Make that reply (1 ... ♘g8×e7) in the analysis.
(8) Test if check has been relieved: Yes.

(1) Generate checking moves: 2 ♘g7-e6+
(2) Select one move for further analysis.

(a) For each check generate and count replies:

2 ♘g7-e6+
2 ... ♔d8-e8
2 ... d7×e6
2 ... f7×e6
3

(b) Transfer check to try-list: order them by their "u" values:

| L35 | u |
|---|---|
| 1 ♗d6-c7+ | 1 |
| 1 ♕f3-f6+ | 2 |
| 1 ♘g7-e6+ | 3 |
| 2 ♘g7-e6+ | 3 |

(c) Select and delete top move from try-list L35 1 ♗d6-c7+

(3) Make that move (1. ♗d6-c7+) in the analysis, restoring the board to the initial position.
(4) Test if checkmate has been achieved: No.
(5) Generate replies that relieve check: 1 ... ♘a6×c7
(6) Select best reply: 1 ... ♘a6×c7
(7) Make that reply (1 ... ♘a6×c7) in the analysis
(8) Test if check has been relieved: Yes.

(1) Generate checking moves: 2 ♘g7-e6+; ♕f3-f6+
(2) Select one move for further analysis.

(a) For each check generate and count replies:

2 ♘g7-e6+
2 ... ♔d8-e8
2 ... d7×e6
2 ... f7×e6
2 ... ♘c7×e6
4

2 ♕f3-f6+
2 ... ♘g8×f6
2 ... ♘g8-e7
2

(b) Transfer checks to try-list; order them by their "u" values:

| L35 | u |
|---|---|
| 1 ♕f3-f6+ | 2 |
| 2 ♕f3-f6+ | 2 |
| 1 ♘g7-e6+ | 3 |
| 2 ♘g7-e6+ | 3 |
| 2 ♘g7-e6+′ | 4 |

(c) Select and delete top move from try-list: 1 ♕f3-f6+

(3) Make that move (1 ♕f3-f6+) in the analysis, restoring the board to the initial position.
(4) Test if checkmate has been achieved: No.
(5) Generate replies to relieve check: 1 ... ♘g8×f6; ♘g8-e7
(6) Select best reply: 1 ... ♘g8×f6

(7) Make that reply (1 ... ♘g8×f6) in the analysis.
(8) Test if check has been relieved: Yes.

(1) Generate checking moves: 2 ♘g7-e6+; ♗d6-e7+; ♗d6-c7+
(2) Select one move for further analysis.

(a) For each check generate and count replies:

```
2 ♘g7-e6+      2 ♗d6-e7+
2 ... ♔e8-d8
2 ... d7×e6          0
2 ... f7×e6
    3

2 ♗d6-c7+
2 ... ♘a6×c7
    1
```

(b) Transfer checks to try-list: order them by their "u" values:

| L35 | u |
|---|---|
| 2 ♗d6-e7+ | 0 |
| 2 ♗d6-c7+ | 1 |
| 2 ♕f3-f6+ | 2 |
| 2 ♘g7-e6+ | 3 |
| 1 ♘g7-e6+ | 3 |
| 2 ♘g7-e6+ | 3 |
| 2 ♘g7-e6+′ | 4 |

(c) Select and delete top move from try-list L35: 2 ♗d6-e7+.
(3) Make that move (2 ♗d6-e7+) in the analysis.
(4) Test if checkmate has been achieved: Yes.
(6′) Select next best reply 1 ... ♘g8-e7
(7′) Make that reply (1 ... ♘g8-e7) in the analysis, restoring the board to the appropriate position.
(8′) Test if check has been relieved: Yes.

(1) Generate checking moves 2 ♘g7-e6+; ♗d6×e7+; ♗d6-c7+; ♕f6×e7+.
(2) Select one move for further analysis.

(a) For each check generate and count replies:

```
2 ♘g7-e6+      2 ♗d6×e7+
2 ... ♔d8-e8         0
2 ... d7×e6
2 ... f7×e6
    3

2 ♗d6-c7+      2 ♕f6×e7+
2 ... ♘a6×c7         0
    1
```

(b) Transfer checks to try-list: order them by their "u" values:

| L35 | u |
|---|---|
| 2 ♗d6×e7+ | 0 |
| 2 ♕f6×e7+ | 0 |
| 2 ♗d6-c7+′ | 1 |
| 2 ♗d6-c7+ | 1 |
| 2 ♕f3-f6+ | 2 |
| 2 ♘g7-e6+″ | 3 |
| 2 ♘g7-e6+‴ | 3 |
| 1 ♘g7-e6+ | 3 |
| 2 ♘g7-e6+ | 3 |
| 2 ♘g7-e6+′ | 4 |

(c) Select and delete move from try-list L35 2 ♗d6×e7+
(3) Make that move (2 ♗d6×e7+) in the analysis.
(4) Test if checkmate has been achieved: Yes.
(6′) Select next best reply: None.
(7′) Make that reply (None) in the analysis.
(8′) Test if check has been relieved: No
Mark MATE and print move tree (Fig. 4.)

This example also illustrates the criteria by which the order of application of defensive moves is accomplished: by "best" reply is meant that reply that seems most likely to give the attacker trouble. Thus the priority of defensive moves Black tries is, first, the capture of the most valuable White pieces by the least valuable Black pieces followed by King moves, interpositions, then order of generation. Again this is an attempt to clip unnecessary proliferations in the move tree: if there is a "killing" reply to a checking move, further analysis of that checking move would seem futile.*

*Measures of Search Behavior.*

Many measures of search behavior can be picked off an analysis tree like MATER I's of Diagram 36 (Fig. 4.) For example, the tree can be characterized by counting the number of positions or the number of moves. Simon and Simon (1962) call the total number of positions examined the "exploration tree"; in Diagram 36, above, the position

*This is the minimax assumption: namely that the opponent will make his strongest reply at every opportunity. McCarthy's *killer heuristic* (see Kotok, 1962) assumes that a killing reply to one checking move may be a killing reply to other checking moves and thus should be looked at first.

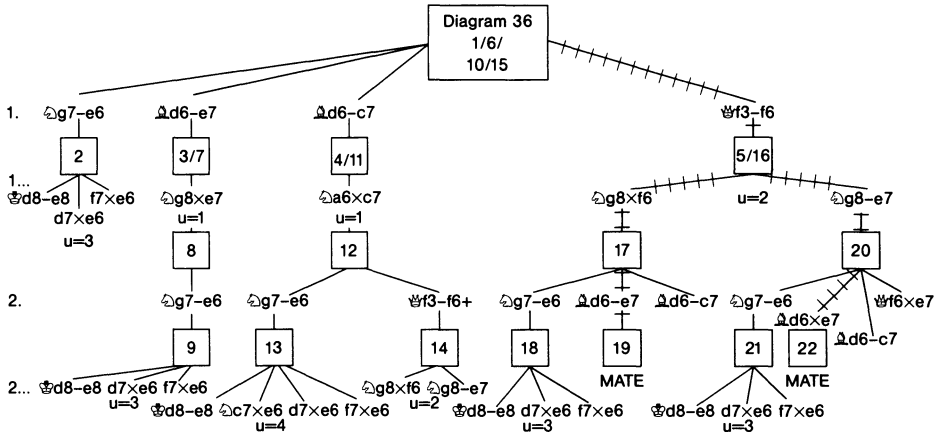



*Figure 4. MATER I's analysis tree of Diagram 36, Fine (1952).*
*(The u's are a tally of the number of replies by which the priority of moves on the try-list is established. The square boxes represent positions and the numbers in them trace the course of the investigation — the order in which the positions were taken up. The crosshatched branches trace the mating path.)*

count yields an exploration tree of size 16. In general, however, more moves are seen than positions are investigated, which is to say that some remain unexplored, such as the replies to 1 ♘g7-e6+ in the example above. The count of moves seen — the uninvestigated as well as the investigated ones — will be called the discovery tree; in Diagram 36, above, this move tally yields a discovery tree of size 36 (14 checks and 22 responses). One further refinement can be carved out of the exploration and discovery trees; namely, the "verification tree," which Simon and Simon (1962) define as the total number of positions required to prove the combination — the positions resulting from the single best move at each node for the attacker and from every legal move at each node for the defender; respectively, the positive and negative parts of the proof schema (de Groot, 1965). The verification tree "is precisely analogous to the correct path in a maze. It is a tree instead of a single path because all alternatives allowed to the defender must be tested" (Simon and Simon 1962, p.427). In Diagram 36, above, the branches of the verification tree are crosshatched, yielding a position count of size 6, or alternatively, a move count of size 5.

These measures do not reveal the time order in which the tree was generated. Human chess players are fickle tree climbers, "progressive deepeners," to use de Groot's (1965) term for the phenonenon: "The investigation not only broadens itself progressively by growing new branches, countermoves, or considerable own-moves, but also literally deepens itself: the *same* variant is taken up anew and is calculated further than before" (de Groot 1965 p.266). In other words, the search strategy is an important structural characteristic of the thought process. In Fig. 4 the order in which positions are taken up is captured by numbering the nodes (positions) in the analysis. These measures will be used for comparative purposes in Section 5.

*Routines.*

How are checking moves and replies actually generated in any given position? There would seem to be two tacks, corresponding to a one-many approach. In trying to find all the checks in a given position, for example, one could either radiate out from the enemy King and from each square, search for a piece that can get there and check (the one-many approach), or converge from the squares along the move directions of each attacking piece onto the enemy King's square (the many-one approach). If there are many pieces on the board, the former is the more efficient; if few, the latter.

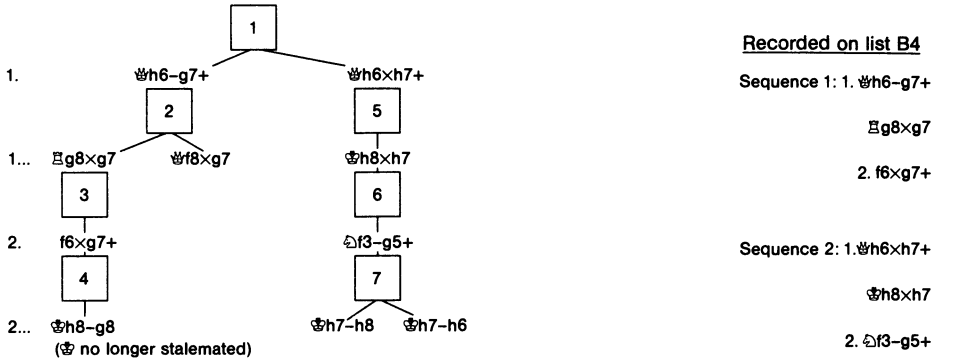



*Fig. 5A. MATER II's preliminary search in position 97.*

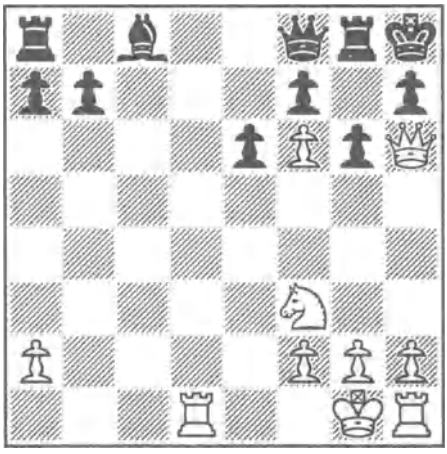
G1 is a master routine that procures all checks in a given position. It employs the many-one approach, calling subroutine G11 for Queen, Bishop, and Rook checking moves: G12 for Knight and regular Pawn checks: and G13 for double Pawn moves that administer check. Similarly, R21 procures all replies to a given checking move: R11 generates all the King moves that get out of check, R12, all the captures of the checking piece, and R13, all the interpositions. In this way the mating program is able to enumerate all checks and all replies in a particular position.

## *MATER II*

In designing search programs it is useful to distinguish the strategy of search from the information that is gathered during the search. The search strategy tells where to go next, and what information must be kept so that the search can be carried out. It does not tell what other information to obtain while at the various positions, nor what to do with the information after it is obtained. There may be strong interaction between the search itself and the information found, as in the decision to stop searching, but we can often find this as occurring within the confines of a fixed search strategy.
(Newell and Simon, 1965, pp.24-25)

MATER II adds a modification to MATER I's search strategy by bypassing the fixedness in the order of application of operators inherent in the trylist. The new search rule states: in the given position pursue immediately and in depth all checking moves that keep the enemy King stalemated (or nearly so), i.e., moves that can only be answered by captures and/or interpositions or, in the absence of both, by one and only one King move (the "nearly so" condition). In addition to altering the program's search strategy by telling it "where to go next," this procedure also gathers information about the position. In this respect it resembles what de Groot (1965) has called a "sample variation," a kind of trial balloon sent up for the express purpose of gathering information to direct subsequent investigation; in this sense it is orientative. Before turning to what information is gathered and how it is used, it should be mentioned that sometimes a sample variation pays off directly — the "sample moves" may be a path to a quick mate.

Specifically, a routine G10 conducts the preliminary search and, if no "easy mate" is found, records the sequence of moves investigated on a list B4. A routine G17 makes use of this information later in drawing up a plan of attack. Just how these routines operate can best be seen by



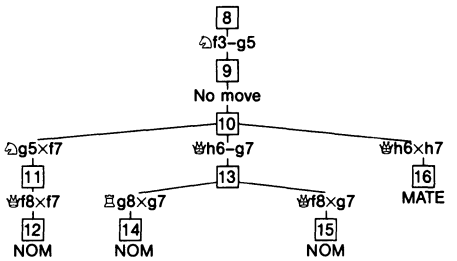



*Fig. 5B.* MATER II's *plan formation in position 97.*

considering in three parts MATER II's analysis for a particular position, Diagram 97 from Fine (1952).

The first part has to do with the preliminary search; the second with the use to which the recorded information is put in drawing up a plan: and the third with the exploration and verification of that plan.

The first 10 moves of the discovery tree are those in Fig. 5a. (Note that both the 1 ♕h6-g7+ and 1 ♕h6×h7+ sample variations are recorded on list B4).

Clearly, the "wishful thinking" goal of the sample variations went unfulfilled; the preliminary excursions did not yield mate. They do yield two sequences of forcing moves, however, that may be useful in constructing a plan of attack. Indeed, the routine G17 searches list B4 for the first move candidates and finds, in this example. 1 f×g7 and 1 ♘f3-g5. The former is rejected as illegal in the current position while the latter is deemed considerable. Note that the set of operators that may be applied in the initial position is expanded in MATER II to include moves other than just checking moves, yet the means by which these are generated continues to ensure a high degree of selectivity.

Routine G17 asks if a proposed move, in this case 1 ♘f3-g5, threatens mate in one move. It determines the answer by assuming that black does nothing on his turn, that is, by playing a "No Move" and then seeing if White can enforce an immediate checkmate. And, indeed, 2 ♕h6×h7 is mate. In other words, White leaves the actual problem space to seek a mate in a simplified planning space (see Newell, Shaw and Simon, 1962) and, in fact, the second part of the move tree is given over to solving the problem in the planning space (Fig.5B).*

Finally, the third stage is devoted to testing the soundness of the plan: that is, suppose Black tries to avert 1 ♘f3-g5 and 2 ♕h6×h7 mate. Can he? It happens that in this position he cannot so that the exploration tree and the verification tree are identical in this stage of the analysis. (The rather lengthy third stage is omitted here.)

MATER II contains one other highly selective mechanism for finding moves that threaten mate in one. Controls exerted over the enemy King's square and the squares in this immediate vicinity are built into a list structure called the King's Sector. For a given square five kinds of control have been defined.

1. No control – the enemy King can move to the given square.
2. Attacking control – the attacker can move to or capture on the given square.
3. Occupation control – one of the attacker's pieces occupies the given square.
4. Block control – one of the defender's pieces occupies the given square.
5. X-ray control – the attacker can unmask an attacker control by removing one of his own pieces (corresponding to a "discovery" in chess jargon) or he could unmask an attacking control but for an enemy interposer (corresponding to a "pin").

The King's sector, L40, is constructed by the four routines E91-E94. The complete structure of L40 and the information contained therein can best be seen in Fig. 6, the King's Sector for Diagram 70 from Fine (1952). Attribute Y3 has data term XO as its value, a tally of the number of uncontrolled squares in the Sector (see Fig. 6).

How is all this information retrieved and used by the program? First, a routine G14 tries to generate mate-threatening moves by converting an X-ray control into a second attacking control. For example, in Diagram 70, routine G14, seizing on the White Bishop's X-ray control of g8 proposes 1 ♗c4×f7 but then rejects the move because 2 ♗f7-g8 does not administer

*A hybrid version of MATER I and II would first have re-investigated 1 ♕h6-g7+ and 1 ♕h6×h7+, invoking the fewest-replies heuristic, transferring these two checks to the try-list, and elaborating them, before even considering moves which threaten mate in one. Unfortunately the statistics gathered on various versions of the program are too incomplete to say which search strategy is superior across positions, if in fact a correct strategy can be determined independent of position.

```
L40 9-0
9-0 0
    S64 (h8)
    9-1
    S63 (g8)
    9-2
    S55 (g7)
    9-3
    S56 (h7)
    9-4
    Y3 (No control over
        how many
        squares?)
    XO = 1
---------------
Knight  { S54 (f7)
directions  9-5
not used in  S47 (g6)
tallying XO  9-6 0

(h8) 9-1 0
         M0 (Man on h8?)
         M29 (Black King) 0
(g8) 9-2 0
         Y7 (X-ray control)
         9-10                    9-10 0
         Y6 (Attacking control)          M6 (White Bishop X-rays g8)
         9-11 0                          M22 (Black Pawn is X-rayed) 0
(g7) 9-3 0                       9-11 0
         Y6 (Attacking control)          M2 (White Knight attacks g8) 0
         9-12                    9-12 0
         Y4 (Block control)              M7 (White Knight attacks g7) 0
         M23 (Black Pawn blocks g7) 0
(h7) 9-4 0
         Y4 (Block control)
         M24 (Black Pawn blocks h7) 0
(f7) 9-5 0
         Y6 (Attacking control)
         9-13                    9-13 0
         Y4 (Block control)              M6 (White Bishop attacks f7) 0
         M22 (Black Pawn blocks f7) 0
(g6) 9-6 0
         Y6 (Attacking control)
         9-14 0                  9-14 0
                                         M4 (White Queen and White
                                         M2 Knight attack g6). 0
```

*Figure 6. IPL-V List Structure of King's Sector Controls (arranged in attribute-value pairs) of R. Fine's (1952) Diagram 70, from a game Alekhine-Supico, 1942.*

check, let alone mate. Second, a routine G19, given one attacking control, tries to add a second. Routine G19, seeing an attacking control over g7 in position 70, proposes to add another with the moves 1 ♕d6-f6 1 ♕d6-g6 and 1 ♕d6-h6. Since all three produce mate if Black does nothing, all three are accepted as considerable moves in the plan.

In summary, MATER II contains several mechanisms for generating a selective set of considerable moves. Incorporating MATER I's ability to generate all checking moves and all replies in a given position, MATER II goes on to generate mate-threatening moves based either on their earlier appearance in forced sequences of checking moves or on the function they serve in controlling key squares around the enemy King. Moreover, MATER II has a set of routines, R15, R16, R22, for generating defensive replies to a threatened mate.

MATER II also contains three principal mechanisms in its search strategy specifying the order in which moves are to be considered. Defensively, in reply to checking moves, captures are preferred to King moves, which, in turn, are preferred to interpositions; while in reply to one-move mate threats, captures are preferred to moves that defend the mating square as well as to interpositions and King runs. Offensively, search is directed by pursuing particular moves in depth so long as the enemy King remains very highly constricted, and then later by pursuing the move that leaves the opponent with the fewest replies. Each of these search evaluators rests on a single criterion: sometimes a line of search is determined because the defender is left with King moves in reply (nodes 4 and 7 in the move tree of position 97); sometimes a move is rejected because it does not produce immediate mate (nodes 11 and 12 in position 97): sometimes a move just never gets off the waiting list (node 2 in position 36): and checking moves with more than four legal replies are always rejected out of hand. Indeed, it is the thesis that these kinds of criteria, criteria based on features of the

task area, are what regulates chess players' choice-of-move decisions and form a good alternative representation to complicated weighting functions of the sort employed by Samuel (1963) in checkers and Bernstein and Roberts (1958) in chess. Even though mating combinations are the only facet of the game in which the final evaluators, MATE and NOMATE, are well defined* – a degree of certainty attained nowhere else in the game – the search directing decisions intermediate to the final choice of move must all be made on far less than certain criteria, just like the rest of the game.

Except for the sample variations recorded on the list B4, the information-gathering mechanisms rely on the description list of the move that gathered it, including the final evaluation, MATE or NOMATE, which are propagated back up the tree by the minimax inference procedure in an attempt to demonstrate the proof of combination.

## Interpretation and Results

### *Introduction*

With respect to the verification of simulation models in general, and problem-solving models in particular, two criteria for assessment seem to have emerged clearly: an achievement criterion and a process criterion. That is, can the model solve the class of problems it was designed to handle, and are its mechanisms for doing so equivalent to, or even comparable to, a human problem solver's? The answer to the first question is relatively straightforward; not so the second, however, since the requirements for equivalence or comparability of process are themselves open to question. In the present report, we shall not consider the questions of human simulation but will confine ourselves to a discussion of the achievement of the programs.

MATER I solves combinations which consist of uninterrupted series of checking moves, given that the defender at no node in the verification tree has more than four legal replies; MATER II solves combinations that begin either with checks or with one-move mate threats and checking moves thereafter. This limitation on the class of moves the program can see restricts severely the class of combinations on which the model can be tested. Nevertheless, the program has been tested on material taken from Fine's (1953) chapter on the mating attack. Solutions to one class of positions in the chapter call for an uninterrupted series of checking moves ending in mate (51/129 positions). Another class of positions is solved with one-move mate threats and checking moves thereafter (5/129 positions). In the residual class, mate can either be averted through a sacrifice of material or the mate is not "forced," as the term was defined in Section 1 (73/129 positions).

### *MATER I's Achievement*

MATER I found solutions to 43 of the 51 mating positions. The machine missed one combination entirely by failing to move a Pawn that gave a discovered check and exhausted available space before finding the other seven.† Table 2 breaks these 43 positions down according to certain structural measures of search behavior: the depth of search to mate (D), the mean size of the verification tree necessary to prove the combination ($\overline{VT}$ measured in moves), and the mean size of the discovery tree generated in searching for mate ($\overline{DT}$ measured in moves). These latter two measures were defined in Section 4.

*Table 2. MATER I's Performance on 43 Positions from Fine (1952).*

| N(positions) | D | $\overline{VT}$ | $\overline{DT}$ | $\overline{VT}/D$ | $\overline{DT}/D$ | $\overline{VT}/\overline{DT}$ |
|---|---|---|---|---|---|---|
| 15 | 2 | 3.5 | 15.5 | 1.8 | 7.8 | 4.4 |
| 11 | 3 | 5.4 | 24.6 | 1.8 | 8.2 | 4.6 |
| 14 | 4 | 9.9 | 61.5 | 2.5 | 15.4 | 6.2 |
| 2 | 5 | 11.0 | 56.0 | 2.2 | 11.2 | 5.1 |
| 1 | 8 | 17.0 | 108.0 | 2.1 | 13.5 | 6.4 |

Simon and Simon (1962, p.428) suggest depth, number of positions in the exploration tree, and number of positions in the verification tree as measures of the difficulty of combinations. They remark that

*"Well defined" is used in the sense that there exists a satisfactory test that enables the player to recognize the solution to his problem (McCarthy, 1956).

†In particular, MATER I failed to see 2 f5-f6+ in position 148. The seven positions that exhausted available space did so because the fewest replies heuristic failed to discriminate among alternative checking moves: among six alternative discovered checks in positions 41 and 100: among a large number of initial checks available in positions 109 and 140: and among a large number of checks in depth involved in a King hunt on the open board in positions 111, 130 and 157.

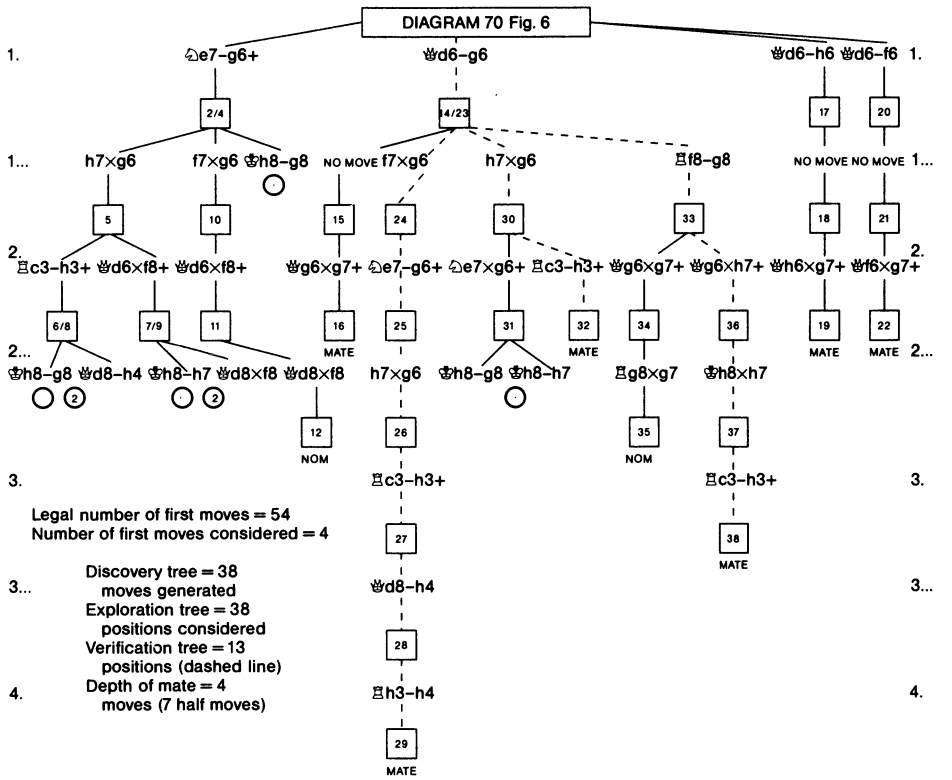



*Figure 7. MATER II's Analysis Tree and Order of Search on Diagram 70, from Fine (1952).*

the four positions in their sample "are not ordered in the same way with respect to the different measures of difficulty." Using depth, number of moves in the discovery tree ($\overline{DT}$), and number of moves in the verification tree ($\overline{VT}$) as equivalent measures, the data of Table 2 do show, with but one exception, the same ordinal relationship across measures, at least when averaged over the 43 positions.

Between depth and the size of the verification tree ($\overline{VT}/D$) there is a fairly close correlation – around two moves in the verification tree per move in depth. This confirms the Simon and Simon (1962) observation: the tree varies linearly, not exponentially, with depth, and it probably is this property that makes deep analysis possible in combinations. Neither $\overline{DT}/D$ nor $\overline{VT}/\overline{DT}$ shows any consistent relationship.

The only roughly constant ratio, $\overline{VT}/D$, has more to do with the characteristics of mating positions than with characteristics of the mating program. The only measure on the program's search behavior is $\overline{DT}$ and there seems to be no consistent relationship between it and the two measures on the combinations ($\overline{DT}/D$ and $\overline{VT}/\overline{DT}$).

### *MATER II's Achievement*

MATER II has been tested on all five positions from Fine (1952) that necessitated an initial threat of mate in one and checks thereafter. It solved three directly, the other two, because of a change in computer facilities, by hand simulation. The search tree for position 97 has already been described. In position 107 the initial Queen sacrifice as well as an unexpected Bishop sacrifice were easily spotted in the service of mate. The analysis tree of position 70 is given in Fig. 7. Note that the correct move and theme in position 70 derive from the celebrated

game of Marshall's for which spectators showered the chessboard with gold coins!* Of the two hand simulated runs position 95 required but the simple addition of a second control on the g7 square via 1 ♕h6, while position 113 required the move 1 ♖h5, which had been discovered in one of the exploratory sample variants.

*Conclusion*

In conclusion MATER's power stems from its ability to generate a small selective set of moves that merit investigation. Since most of the earlier chess programs (see the review in Newell, Shaw and Simon, 1963 and Kotok 1962) spent their analysis time processing the wrong moves, it would seem that MATER II's two major mechanisms for generating relevant moves – its reliance on the sample variations and on the control of key squares – warrant further research. MATER II's major weakness, on the other hand, lies in its poorly organized search strategy for using its selectivity at all points in the analysis process.

On the horizon, proposals have been made for strengthening the program's perceptual capabilities as well as altering its search strategy.

## Appendix

### The Basic Representation

*Statement of the Problem*

In this Appendix we describe the basic representation implemented by Newell and Prasad (1963). Two interrelated questions guided the choice of representation:

1. What are the necessary components of a chess representation?
2. How should this information be organized?

In response to (1): The program should be able to "see" the same things a human sees when he looks at a chessboard. Thus the program requires an internal representation of the squares and pieces on a chessboard and the relations among them, and a set of processes that can pick off and make use of these relations as needed. The former requirement is called "setting up the chessboard", the latter "move-making and board processing capabilities."

In response to (2): The game of chess provides an inhomogeneous collection of information out of which moves must be forged. Thus there must be enough variety in the representation to discriminate all the different kinds of moves: given that, the information should be stored in such a way that little space is allotted to moves that seldom occur (such as Pawn promotions, castling, etc.), and the dependence and division of information between routines and data should remain flexible and open to change and never solidify into a resistant collection of conventions (Newell, 1962). List processing languages are specifically designed to cope with such problems.

A chessboard is made up of squares, which lodge pieces, which make moves from one square to another. Objects in chess, like the 64 squares and 32 men, can be represented as symbols on lists, and moves can be represented as names of description lists with certain prescribed associations (such as the square *from* which a piece comes, the square *to* which it moves, and the kind of move in question). A chess position, moreover, can be fully described as a list of pieces and squares and a chess game as a list of moves that originate from a standardized initial position and terminate in a well- defined checkmate position.

*Setting up the Chessboard*

A chessboard is made up of eight ranks and eight files which rule off 64 squares. The sequence of symbols S1 ... S64 is used to denote these 64 squares.

In the data section of the program there are a list of ranks, L1, containing members R1 through R8, and a list of files, L2, containing members F1 through F8. Each rank is itself the name of a list containing eight member squares: e.g., R1 contains S1, S2 ... S8.

In the routines section of the program a super routine, E1, sets up a chessboard; it calls nine routines E2-E7, E9, E10, and E12, which do the work. Routine E2 builds the eight file lists, F1 through F8, out of the rank lists, R1 through R8. Then routine E12

*The program also finds the correct sequences of moves from the immortal Lewitzky-Marshall game, played in Breslau in 1912 (Diagram 69 in Fine, 1952): it is excluded from the count here since Lewitzky, had he not chosen to resign, could have averted mate at the cost of a piece.

takes each of the 64 squares and assigns it rank and file (x,y) coordinates, which are later used to compute another set of relations among squares.

*Squares.* For each square on a chessboard it is essential to know: 1) the name of its occupant, if there is one, and 2) the name of all its neighboring squares in the chess-legal directions. The first desideratum is effected by defining an attribute MO. "Man on Square?" on the description list of every square and assigning as its value the name of the piece occupying it — if there is one.

The extensive network of relations among squares, constituting all legal move directions in chess, is captured by defining 16 directions on the chessboard, beginning with D1 for the forward direction and continuing clockwise to D16 for forward and left (e.g., the Knight's move g1 to f3). Thus the even numbers (D2, D4, ..., D16) define the eight possible Knight move directions; half the odd numbers (D1, D5, D9, D13) define rank (horizontal) and file (vertical) directions: the other half (D3, D7, D11, D15) diagonal directions.

Routines E3, E4, E5, and E8 use lists L1, L2, L5, and L6 to build the network of relations among squares by assigning to each of the 64 squares all surrounding squares as values of each of the 16 directions that obtain. In the initial position, for example, the list structure of the square S8 — h1— would look like this:

| *Name of list* | *Attribute* | *Value* |
|---|---|---|
| Square h1 (S8) | Man on Square (M0)? | White King Rook M8 |
| | Square to the North (D1)? | Square h2 (S16) |
| | Square to the West (D13)? | Square g1 (S7) |
| | Square to the WNW (D14)? | Square f2 (S14) |
| | Square to the NW (D15)? | Square g2 (S15) |
| | Square to the NNW (D16)? | Square g3 (S23) |

Note that in such a list structure representation only those five of the 16 legal directions that are needed are defined; space in memory is not consumed by providing the "information" that the other 11 directions have *no* values, as it would seem to be in a matrix representation of this kind of data. (Cf the argument in Newell, 1962 pp.411-412, for a fuller statement of this view.)

*Men.* The 32 men take and retain their designations from their placement in the initial position: they are denoted by the sequence of symbols M1 ... M32.
For each piece, it is essential to know:
1. the square he occupies (attribute SO),
2. his type (attribute A1),
3. his color or side (attribute A2),
4. his legally permissible move directions (attribute A20), and,
5. his legally permissible capture directions (attribute A21).

The first of these is effected by defining an attribute SO, "Square on?" on the description list of every piece and assigning as its value the name of the square the piece currently occupies. The other four attributes assume the complete range of values in accord with the rules of chess.

In the data structure there are 11 lists that group the chess men by types or otherwise useful categories: White Pawns; Black Pawns; Bishops; Rooks; Knights; Queens; Kings; White Rooks, Bishops and Queen; Black Rooks, Bishops, and Queen: White Knights; and Black Knights.

For each side, moreover, there is a list giving the type of each man on that side, and another list giving the move directions of each type of man.

Routines E6 and E7 assign to each man his type, color, move directions, and capture directions. In the initial position, for instance, the list structure of M8 would look like this:

| *Name of list* | *Attribute* | *Value* |
|---|---|---|
| White King Rook (M8) | Man on what square (S0)? | Square h1 (S8) |
| | Type of man (A1)? | a Rook |
| | Color of man (A2)? | White (K10) |
| | Move directions (A20)? | a list of directions D1, D5, D9, D13. |
| | Capture directions (A21)? | a list of directions D1, D5, D9, D13. |

*Positions.* A chess position can be described fully in terms of squares and pieces. Since MATER is supposed to find a checkmate in any given position, obviously some representation is necessary for encoding a

particular position. This representation is a describable list called the position list, L10. Its main list consists simply of the name of each man present on the board and the name of the square each man occupies, arranged in attribute-value pairs. Its description list contains a set of special attributes pertinent to the characterization of the position: in particular, S65, the "Whose move is it?" attribute that flip-flops between K10 (White on move) and K11 (Black on move); S66, the name of the castle list that contains the Kings and Rooks still "eligible" for castling; S67, the signal cell that gets set when an *en passant* capture is in the offing; and S69, the name of the most recent move made on the board.

Routine E10 takes as input any position list – either the initial position or the mating position, which is read in from cards by routine E90 – and converts it into a set of associations between pieces and squares such that every piece has an attribute SO ("Square on?") with the square that piece occupies as value and every square has an attribute MO ("Man on?") with the name of the chess piece – if there is one – occupying that square as its value.

This ends what might be called the "static" perceptual relations on the chess-board. What follows is a bundle of basic routines that attempts to provide "dynamic" perceptual relations to the program.

### *Move-Making Capabilities*

Moves are the operators that transform one chess position into another. What are the common properties of chess moves? Each involves a piece, or sometimes two, going from one square to another. If the "to square" is already occupied, the move is called a capture. If the "to square" is on the eighth rank and the piece a Pawn, it is called a promotion. But in all cases the common "from-to" property holds.

This permits a move to be represented as the name of a description list containing a "from square" (as the value of an attribute A40) and a "to square" (as the value of an attribute A41). This information is sufficient to specify most moves. There is a special class of moves, however, which, while adhering to this "from-to" pattern, introduce some idiosyncratic properties of pieces. Each of the following five members in this class is assigned a different value to the special move attribute A42: King's side castling, Queen's side castling, a double Pawn move, an *en passant* response thereto, and a Pawn promotion.

Five steps are required to make a move: first, the move must be constructed; second, it must be tested for legality; third, for repetition of position; fourth, it must actually be made on the board; and fifth, it should be printed.

Routines E51 and E52 construct regular moves and special moves, respectively. Both create a new cell or symbol, which becomes the name of the move. Both take as input the square from which the piece is to move and the square to which it is to move and assign these as values of attributes A40 and A41, respectively. For the special move routine, E52, the type of move must also be specified as input; it is assigned as the value of attribute A42. The name of the man moved is also received as the value of another attribute A51, for reasons that will appear under step 3 below. The move 1 e2-e4 for example, would be represented as follows (where $a_1$ and $a_2$ are internal cell names):

| *Name of list* | *Attribute* | *Value* |
|---|---|---|
| $a_1$ | From Square (A40)? | Square e3 (S13) |
| | To Square (A41)? | Square e4 (S29) |
| | Man moved (A51)? | White King Pawn (M13). |

Similarly, the special move e7-e8 ♕ would be represented in the following format:

| *Name of list* | *Attribute* | *Value* |
|---|---|---|
| $a_2$ | From Square (A40)? | Square e7 (S53) |
| | To Square (A41)? | Square e8 (S61) |
| | Special Move (A42)? | Promotion to Queen |
| | Man moved (A51)? | White King Pawn (M13). |

Second, a routine E18 checks to insure that a newly constructed move is legal. The routine test, for example, whether a Bishop is moving through a Pawn, whether a Rook is making a Bishop move, whether a player is castling through check, and the like. The output is a simple "+" ("yes, the move is legal") or "–" ("no, the move is illegal").

Third, according to the laws of chess a threefold repetition of position constitutes a draw, and, according to the laws of computers, a loop. Before a move is executed, therefore, a routine E55 tests if the move

under consideration has been played before in the same position. The position could not have occurred before if the move is irreversible, that is, if once the move is made on the board no subsequent set of legal moves can ever regain the exact same position. Captures, Pawn moves, and castling are all irreversible. Thus when a capturing move is constructed (step 1), it is given an attribute A44 with the man captured as its value. When a castling move is constructed, its status as a special move is recorded as the value of attribute A42. And for a Pawn move, a record is kept via the man moved attribute, A51. Routine E56, called by E55, tests for any of these three conditions to declare a move irreversible. If none of them obtains, E55 must take some further comparisons between the A40 and A41 values of the proposed move and earlier moves.

Fourth, a routine E65 makes a regular move on the board; routines E71-E75 and E81-E85 execute the special moves. A move is made by updating the position list, which is done in two steps: first, with the assistance of routine E11, the description lists of the pieces and squares affected by the move are updated. Second, the signal cells – S65, S66, S67, and S69 – affected by the move are reset.

Since different routines are needed to make each of the five special moves, these routines are simply associated with their respective special move values in the data section of the program.

Fifth, there is a print routine, E16, which prints out the name of the move, the "from square," the "to square," and the man captured, if any.

*Additional Board Processing Capabilities*

In addition to the move-making capabilities just described, there is a second group of routines intended to provide the machine with some more of the perceptual capabilities a human possesses; these are the board processing routines which provide answers to questions asked of the board. Routine E13 finds the direction, if one exists, between two given squares. Routine E14 tests to see if there is a piece between two squares in a given direction, and E24, if there is one and only one piece between two squares in a given direction. E15 tests if a piece is under attack, and routine E26 asks specifically if that piece is the enemy King. Routine E33 tests whether a given square is under attack, while E34 builds up the list of men of a particular color attacking a given square. Routine E36 tests if a given square is defended. These are some of the more important "building block" routines used in constructing the move tree.

*Processing Speed*

It might be supposed that since the program was written in an interpretive language, IPL-V, without any attempt to provide a special machine-language representation for primitive board manipulations, that it would be a very slow player. This has not proved to be the case – a tribute to the advantages of selectivity over machine brute force. The most difficult mates, requiring the examination of about 100 positons, were achieved in about 10 minutes on a CDC G-20 – which would be equivalent to about three minutes for the IPL-V system on the IBM 7090. An excellent human player might be expected to take ten minutes or more to discover and verify the mate in a position of this difficulty.

**Ackowledgments**

We would like to acknowledge the assistance of G. A. Forehand, B. F. Green, A. Newell, M. R. Quillian, and R. F. Simmons.

---

This investigation was supported in part by Research Grant MD-07722-01 from the National Institutes of Health, and by the System Development Corporation while the author was a summer associate there.

# 5.5 Robot Chess

**D. Prinz**

First published in *Research,* 1952, pp.261-266. Reprinted by permission of Butterworth and Co. (Publishers) Ltd.

*In recent times digital computers have been increasingly used as aids in the study of scientific and economic problems. Dr. Prinz describes how such a machine can be 'programmed' to solve simple chess problems successfully, though it does so more slowly than a human chess player.*

The age old dream of machines playing games of strategy, like draughts or chess – and playing them well enough to beat human opponents – has recently been revived for two reasons. One is the realization that problems of economics can be analyzed by means of logical structures similar to those implied in games of strategy. J. VON NEUMANN and O. MORGENSTERN have recently discussed this approach in great detail (1947). This seems to lift the interest in game-playing machines out of the realm of mere entertainment into the class of respectable scientific research; if machines can be made to play games, they should be able to solve economic problems of similar structure.

The second reason is the advent of the 'electronic brains' or, to give them their proper name, universal high speed electronic digital computers. The emphasis is on the 'universal'. The same machine can be made to solve a problem in aircraft design in the morning, compute optical lens systems in the afternoon and work out PAYE wage lists in the evening. You can make it do all this without any alterations in its design, without plugging in new connections, simply by providing it with suitable information, usually contained on a punched tele-printer tape or similar device. The problem is no longer 'making a machine to play chess' but rather 'making a machine play chess'. You do not have to be an inventor or engineer; all you need is paper, pencil and patience. You also need access to an existing electronic computer, including the apparatus for converting your notes into a punched paper tape, and a knowledge of the 'code' used by the machine *i.e.* the system for writing down information in such a form that the machine may understand and carry out your instructions.

The complete list of instructions (or 'orders' or 'commands') required for any particular problem is called a 'programme', and the technique of designing programmes, or of 'teaching' the machine what to do, is known as 'programming'. Subsequences of instructions in a programme are often called 'routines'.

The public interest in robot chess is reflected in the recent literature: I mention only two papers which have appeared in the *Science News* series and which deal with the theoretical aspects of the problem (Davies, 1950) and describe special machines built in the past (Byard): they may be consulted for further references. In addition, the daily press has taken notice. In a report to the *Manchester Guardian* (13th November, 1951) ALISTAIR COOKE has mentioned an American firm which was said to have challenged any human chess player to a match against its electronic machine. However, inquiries in the United States have so far failed to confirm this report and it seems that the first concrete approach to the problem is the work carried out on the machine* installed at the University of Manchester. This work will be briefly described here.

## The problem

At the outset it must be made clear that the aim of this experiment was rather modest. No attempt was made to let the machine play a successful game of chess. What has been done is a demonstration of the ability of the machine to solve simple chess problems.

*Built by Messrs Ferranti Ltd, Manchester.

A chess problem, as every player knows, is the problem of finding a move, the 'key move', which will mate the opponent, irrespective of his replies, in a given number of further moves. Only the simplest case, the 'mate in two', has been treated, and even so, some of the moves permitted by the rules of chess (double pawn moves, castling and the exchange of a pawn for a piece on reaching the last row) have been excluded to make the programme as simple as possible. Finally, the possibility of a stale-mate was admitted as a solution.

To anticipate the result of the experiment, it has turned out that the machine, when provided with the programme to be described, can make chess moves and find the solution of chess problems within the limits stated above; but it does so much more slowly than a human chess player.

One reason is that the machine is built to carry out arithmetic operations, and the moves on the chess board have to be broken down into a series of such operations before the machine can deal with them. If you are told that the white King is in one corner of the chess board, and that the black King is in another corner, the remainder of the board being empty, you know at once that neither King is in check. But the machine is in the position of a man who has to carry out an arithmetic operation to find which square to examine next and, having arrived there, finds instead of a chessman a piece of paper telling him which arithmetic operations this chessman is allowed to perform in order to reach other squares. Altogether, several hundred operations may be needed to find out whether or not the King is in check, and even at electronic speeds the whole process may take an appreciable fraction of a second.

Another reason is that the present programme does not eliminate unlikely moves: the machine is forced to investigate every possible move, until the solution is found. Even for a two-mover, the total number may run into several thousand on the average, and will be below a thousand only for particularly simple positions. Since every move has to be followed by a test for legality, it may consume something like 1 or 2 seconds, and the total time required for the solution may be several hours. The extremely simple problem illustrated in *Figure 1* took about 15 minutes to solve: most chess players could find the solution in less time than this.

Since the rules of chess usually impose a time limit, the machine would stand no chance of winning a game against a human player; but it could (with minor modifications to the programme) be made to play a game against another machine, or against itself. The possibility must, of course, not be excluded that by refined programming, by the use of faster machine, or both, a superior game played by a universal computer may finally be realized, not to mention the possibilities of special machines built for the purpose of playing chess only.

## The Machine

Before going into a detailed description of the programme, it will be necessary to state briefly the essential features of the electronic computer.

The nucleus of the machine is a store in which patterns, consisting of two different units (like the dots and dashes of the Morse code), can be kept. These patterns can be interpreted as numbers, letters or other pieces of information. The exact correspondence does not matter here. The non-mathematical reader may think of the Morse code the teleprinter code, or the Braille script. For the mathematical reader it may be sufficient to hint at the binary number system, a form of digital representation in which the only digits used are 0 and 1. Neither is the actual form of storage important at this point. Suffice it to mention that the Manchester machine employs two forms of storage, one using magnetized elements on the surface of a rotating drum (similar in principle to magnetic tape or wire recording), the other using electrical charges on the screens of cathode ray tubes, similar to those contained in television receivers. The punched tape containing the original information supplied to the machine may be regarded as an additional, external store.

In the cathode ray or 'electronic' store the information is arranged in successive lines, each line containing either a number or an instruction. The lines themselves are numbered consecutively and the number of a line is often called its 'address'. This address or number of a line must be distinguished from the number contained in the line.

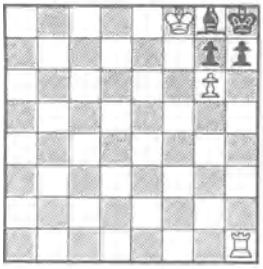


*Figure 1*

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
| 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 |
| 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
| 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 |
| 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

*Figure 2*

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 |
| 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 |
| 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 |
| 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 |
| 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 |
| 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 |
| 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 |
| 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
| 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

*Figure 3*

An instruction consists of two parts, the address and the function part. When the machine 'reads' an instruction, it rearranges its internal connections just as an automatic telephone exchange connects you to another subscriber when you supply information by dialling a telephone number. More specifically the machine will be 'connected' to the line the address of which forms the address part of the instruction, and it will then either carry out an operation on the number contained in this line, or send the result of a previous operation to this line. Thus, an instruction of the form '53,A' may be interpreted to mean 'add the number standing in line No. 53 into the addition register', and the instruction '54,B' may mean 'send the contents of the addition register to line No. 54'. The time required to read and carry out an instruction is different for different computers: for the Manchester machine it is about one thousandth of a second.

Under the influence of a 'control unit' the machine goes normally through the instructions in their natural sequence and obeys them one by one. However, amongst these instructions there are some which in turn react on the control unit: their effect is to interrupt the normal sequence, causing the machine to 'jump' to some other instruction in the store. In particular, it is possible to go back to a previous instruction so that the same cycle of instructions is repeated over and over again. By making these 'control transfer instructons' dependent on certain arithmetic conditions it is possible to leave this 'loop' of repeated cycles after a specified number of times, or when a specified result has been obtained.

Since instructions are kept in the store together with numbers, they can themselves become the object of arithmetic operations. In particular, it is possible to modify their address part. Thus, by adding 1 to the address part, the instruction '53,A' can be altered to '54,A'. This facility is quite fundamental for the applicability of machines of this type. Suppose we wanted to add together the numbers in lines 100, 101, 102 ... up to, say, 199. If we had to write down all the instructions 100, A, 101, A ... 199, A explicitly and punch them out, there would not be much point in having a machine obeying these instructions at a rate of a thousand per second. Instead, we only write '100,A', followed by instructions adding 1 into the address part, and a control transfer instruction leading back to the beginning and so conditioned that the machine stops (or goes to some other sequence) when '199,A' is reached.

Now consider another case. Suppose we have a table of numbers stored in, say, lines No. 200, 201, ... 249. Assume that the machine has computed a number $x$ within this range and put it into line 50. We wish to carry out an operation (symbolized, say, by the letter C) on the number standing in line No. $x$ of our table. All we need is an instruction '0,C' combined with an addition of the contents of line No. 50 to the address part of this instruction. The resulting instruction '$x$,C' will do what we want.

What this means is that the machine can 'look up' a table or directory just as we can find the logarithm of a given number from a mathematical table, or the telephone number of a given person in the directory. Expressed mathematically, the machine can find, and operate on, the values of a function $f(x)$ of an integral variable $x$ provided the function f is tabulated in the store in such a way that line No. $x$ contains the

function value f($x$).

It should be mentioned here that the Manchester machine has a special facility for curtailing this process of modifying instructions. By making use of this facility, or by using conditional control transfer instructions, it is possible to make the machine go to different subsequences according to the number contained in a specified storage line. Thus, by allocating different numbers to the six chessmen (*e.g.* King = 2, Knight = 3), the machine can be made to enter a sequence dealing with King's moves when a certain storage line contains 2, and to enter a sequence dealing with Knight's moves when this line contains 3.

## The Programme

It will now be possible to describe the essential features of the programme by means of lists or tables, consisting of pieces of information contained in successive store lines. The machine goes through these lists under the control of 'counters', certain numbers (kept in other storage lines) which are increased by 1 or 2 at appropriate stages in the programme.

To describe the first of these lists, imagine all the thirty two pieces* numbered consecutively, *e.g.* from 0 to 7 for the eight white Pawns: 8, 9 for the white Knights; 10, 11 for the white Bishops; 12, 13 for the white Rooks; 14, 15 for the white King and Queen; the black pieces are similarly numbered from 16 to 31. The number thus allocated to each piece is called the 'piece counter' and symbolized by $j_p$. It must be distinguished from the 'piece number' $p$ which characterizes the six different pieces irrespective of colour, for instance, as follows:

| Pawn | King | Knight | Queen | Bishop | Rook |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |

We can now describe any particular position by means of a list consisting of 32 consecutive lines which we will number $a$, $a$+1, $a$+2, ... $a$+31, and which are so allocated to the 32 pieces, that line $a+j_p$ corresponds to the piece with the piece counter $j_p$. Each of these lines contains the number $s$ of the square on which the piece is to be found: if the piece is not on the board at all the line will contain some special symbol, *e.g.* a negative number, indicating this fact. This list will be referred to as the 'a-table'.

An additional list, the 'b-table', describes a position in a different way. It allocates successive lines, say, $b$, $b$+1, $b$+2 ... to the squares on the board so that line $b+s$ coresponds to square No. $s$ and contains information regarding the piece occupying this square, such as its piece counter $j_p$, its piece number $p$ and a special symbol indicating its colour. A further symbol is used if the square in question is empty.

Moves from one square $s$ to another square $t$ are made by adding certain numbers to the original square number $s$ to obtain $t$. For instance, we may allocate numbers from 0 to 63 to the 64 squares as indicated in *Figure 2*; then a move one step to the right will be effected by the addition of 1, one step upwards by the addition of 8 *etc.*

However, this scheme involves a particular difficulty when we come to the edge of the board. When the machine has, say, reached square 7 and tries a step to the right, it will compute the new square as 8 which is obviously wrong. We must, therefore, devise a scheme which will tell the machine when the edge of the board is reached. The solution is indicated in *Figure 3* which shows a 10×10 board in which the squares are numbered from 0 to 99. Of these 100 squares, only the central 64 constitute the proper or active board while the remaining 36 form a border. If $s$ is the number of a border square then line $b+s$ will contain a symbol informing the machine that the limits of the active board have been exceeded.

A single-step move to the right is still carried out by the addition of 1, but a single step upwards now means addition of 10. To move diagonally upwards and to the right, we have to add 11. In general, there is a direction number $d$ associated with each of the possible 8 directions. Counting these directions by a direction counter $j_d$, we can put successive numbers of $d$ into successive lines $c+j_d$ of a list starting with line $c$. Beginning with moves to the right and proceeding in an anti-clockwise sense, we thus get the following 'c-table' or 'direction table':

*In correct chess parlance Pawns are not included among the 'pieces'. This distinction will not be made here.

| $j_d$ | *Line No.* | $d$ |
|---|---|---|
| 0 | $c$ | + 1 |
| 1 | $c+1$ | +11 |
| 2 | $c+2$ | +10 |
| 3 | $c+3$ | + 9 |
| 4 | $c+4$ | − 1 |
| 5 | $c+5$ | −11 |
| 6 | $c+6$ | −10 |
| 7 | $c+7$ | − 9 |

The 'step counter' $j_s$ counts the number of steps of a move in a given direction. A move of $j_s$ steps in the direction $d$ is produced by the addition of the product $d \times j_s$ to the original square number $s$. For the King and, according to the restrictions imposed on the programme, for Pawns as well, $j_s$ can only have the value 1.

Knight's moves are composed of horizontal or vertical single-step moves, followed by a diagonal single-step move. In principle, the same direction table can be used, but in practice it has been found more convenient to have a seperate list.

In addition to $j_p$, $j_d$ *and* $j_s$, there is a colour counter $j_c$ which decides whether the machine is making a white or black move. The values of $j_c$ have been chosen as 0 for white, 16 for black. This choice is related to the fact that corresponding white and black pieces are 16 lines apart in the a-table. Instructions modified by $j_c$ will select a piece of the correct colour from this list.

In what follows it will be convenient to talk about a first and second, white and black turn, during each of which a number of moves are tried out. With this nomenclature, White tries out all its first-turn moves until a solution is found. Black answers each of these moves with his first-turn moves until one is found for which no white second-turn move leads to a mate, and so on. A more detailed analysis of the structure characterizing the mate-in-two problem is given at the end of this article.

Let us now consider the case that White has made a number of first-turn moves without finding a solution. How does the machine construct the next first-turn move? White's last first-turn move was characterized by certain values of $s$, $p$, $j_p$, $j_d$ and $j_s$ ($j_c$ being zero) and the machine has stored these numbers, together with the a-table and the b-table describing the original position, in the magnetic store in order to free the electronic store for the information needed in the moves tried out during the other turns. At the beginning of the new moves, all these data are transferred back into the electronic store.

The machine then starts by looking up the line containing the piece number $p$. Under the control of this number, it will then enter a sequence related to the piece having this number. If, for instance, $p=5$, it will enter the sequence for Bishops moves: they are characterized by odd direction counters. Next, the step counter $j_s$ is increased by unity (except when $p$ indicates a Pawn, Knight and King). Then the machine finds the number $s$ of the square from which the last move had been made and the previous direction counter $j_d$. By means of these numbers, the machine forms the sum $t=s+d \times j_s$; this is the number of the square reached by the new move (which goes one step further than the previous one in the same direction).

Having found $t$, the machine looks up the b-table to examine the contents of this square. It may turn out that this square is one of the border squares, or that it is occupied by another white piece, in which case the move has been found to be impossible. We shall deal with this case later and assume for the moment that the new square is either empty or occupied by a black piece *i.e.* that the machine has found a possible move*. Under the control of the symbols for 'black' or 'empty' which the machine has found in line $b+t$, it now enters a routine to test whether the move is legal *i.e.* to ensure that the move has not put the white King in check. This is a complicated process the description of which must also be deferred for the moment. If the move turns out to be illegal the machine returns to the process of counting up and trying out the next move. If it is legal then the new position (possibly also modified by the taking of the black piece on square $t$) is recorded by suitable entries in the a-table and the b-table, and both these tables, together with the new values of the counters, are transferred to the magnetic store, Later, the new position will be used as a starting point for Black's first-turn moves, while the new set of counters will indicate the next move to be made when it

*In cases of Pawn moves the criteria for possibility are suitably modified to conform with the rules of the game.

is White's first turn.

If desired, the machine can be made to print every white first-turn move on its output type-writer. This is useful to ensure, at regular intervals, that the machine is still in working order.

Let us now return to the case where a tentative move made by the machine is found to be impossible, as indicated by the symbols ('border square' or 'white') found in line $b+t$. The machine will then increase the direction counter by a number depending on the piece *e.g.* 1 for King and Queen, 2 for Bishops and Rooks. Again, there are two possibilities. If the new direction counter does not exceed its maximum value 7, the machine will try a single-step move in the new direction and then proceed as before. If $j_d$ does not exceed this number, indicating that all possible directions for this piece have been tried out, the machine is made to enter another sequence which increases the piece counter $j_p$ by 1. Once more, we have two different cases. If the new $j_p$ exceeds 15, *i.e.* if all the 16 pieces have been dealt with, then the position is exhausted; for the white first-turn moves, this means that the problem has no solution at all and the machine will be made to stop or to print out a remark to this effect. If, on the other hand, the new $j_p$ does not exceed 15 then the machine examines line $a+j_p$ in the a-table. Here the process divides again. If the contents of this line indicate that the piece with the counter $j_p$ is not on the board at all, the machine will increase $j_p$ again and go on as before. If the piece is on the board, then line $a+j_p$ will contain the number $s$ of the square containing this piece. The machine will then record $j_p$ and $s$ in their proper storage lines, and will look up line $b+s$ in the b-table where it finds the piece number $p$. This is also recorded and the machine goes back to the process described initially, *i.e.* to a sequence constructing a new move appropriate to the piece $p$, but in this case a single-step move in the first of the possible directions.

The same processes occur in the other three turns of the game, and it is only necessary to add that the first move in any turn is preceded by resetting the counters to their initial values.

The routine for testing legality uses techniques similar to that for finding the next possible move. Assuming we are in a white turn, the machine starts by examining the contents of line $a+15$ where it finds the square on which the white King stands. From this square as a starting point, the machine then explores the board by a series of imaginary moves, leading to various squares $t$. By looking up line $b+t$ in the b-table, the machine then finds out whether $t$ is outside the active board; if not, whether it is empty; and if not, whether the piece standing on it is white or black. Except for the last case, the machine will then immediately construct the next imaginary move. In case of a black piece, the machine examines whether the rules governing this piece are in such a relation to the move just made that the King is under attack. Thus, a black Rook can endanger the white King only if the imaginary move just made had an even $j_d$: for the black King, $j_s$ must have been 1, and similarly for the other pieces. (Knights are treated in a slightly different way; the machine makes imaginary Knight's moves and tests whether or not the value of $p$, found in line $b+t$, equals 3, the piece number allocated to Knights).

The move is illegal if a piece attacking the King can be found; it is legal if the process exhausts the position without finding such a piece. As mentioned before, the need for scanning the greater part of the board whenever a move is made makes this process one of the most time-consuming operations in the programme.

We have seen that the programme for the solution of mate-in-two chess problems must contain a routine for the construction of the next possible move, a routine to check this move for legality, and various sequences for recording the moves made and the positions obtained. All these pieces must now be linked together by a master routine reflecting the structure of the problem as a whole, and ensuring that the various turns are entered in the correct sequence.

This structure is illustrated by the diagram in *Figure 4.* Each of the 4 blocks marked W1, B1, W2, B2 represents one of the turns. Each block is provided with two 'entries' 1, 2 and two 'exits' 3, 4. Entry 1 corresponds to the case of the first move in a turn, with all the counters set to their initial values. Entry 2 is the general case of a move following a previous move of the same turn. Exit 3 indicates that a legal move has been found; exit 4, that the position supplied to the turn has been

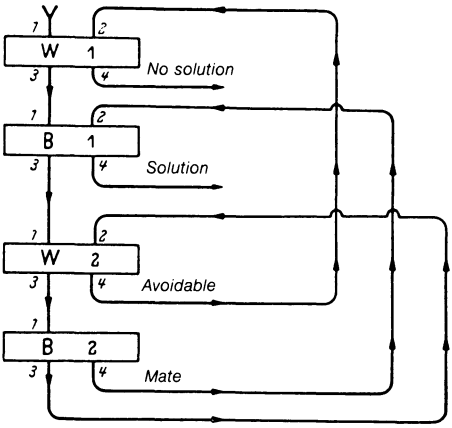



*Figure 4*

exhausted before such a move has been found.

Let us look at the diagram in detail. Right at the beginning, turn W1 is entered at 1. Normally, it will find a legal move, and its exit 3 will lead to entry 1 of B1, the first move in Black's first turn. If B1 finds a legal move, its exit 3 will lead to entry 1 of W2, and similarly, a legal move of W2 will lead from its exit 3 to entry 1 of B2.

If now B2 finds a legal move (exit 3) then White must try another second-turn move, *i.e.* exit 3 of B2 must go to entry 2 of W2, but if B2 finds no legal move, Black is mate (or stale-mate) and must try another first-turn move. This means a connection from exit 4 of B2 to entry 2 of B1.

If, after a first-turn move by Black, White tries out all its second-turn moves without achieving mate, then White must try another first-turn move: this gives the connection from W2, exit 4, to W1, entry 2. On the other hand, if after a W1 move Black exhausts the position without finding a move avoiding mate, then this move by White was the key move; for this reason exit 4 of B1 is marked 'solution'. No solution exists if W1 tries all legal moves without finding one leading to an unavoidable mate; this is exit 4 of W1.

Once the structure of the problem has been translated into numerical form, it is not difficult to arrange a scheme causing the machine to produce a set of suitable numbers at an exit from a turn, and then to interpret these numbers to find the correct entry turn for its next operation. This scheme is carried out by the master routine. The by now familiar technique of looking up and modifying tables is again employed, and it will hardly be necessary to go into further details. Moreover, I trust that this brief outline will have been sufficient to give the reader an indication of the methods that might be used to treat structural or logical problems occurring in other fields by means of electronic computers.

# 6 Writing a Chess Program

A whole book could be devoted to this one subject. In fact there are already at least two such books in print (Spracklen et al. 1978, and Levy 1983). In the present chapter there is a need for selectivity, and so I have restricted myself to three papers.

Norman Whaland's tutorial (section 6.1) provides a useful background for anyone wishing to write their own program. In particular the flow chart of the alpha-beta algorithm is extremely useful.

David Wilkins describes how chess knowledge can be used in the search process (section 6.2). His program PARADISE uses chess knowledge to discover plans during the analysis process and to guide the search in order to confirm that a particular plan is best. Computer chess enthusiasts may well have read complaints from strong human chess players that programs lack ability to plan, and it is well known in chess that "even a bad plan is better than having no plan at all!" I would strongly recommend that anyone wishing to write his own program should make a special study of this paper, in conjunction with Harris' work on heuristic search. There is a wealth of good ideas to be explored, more interesting and possibly more rewarding than those employed in writing a "brute force" program.

The paper by John Birmingham and Peter Kent (section 6.3) describes how a program can recognise a mating situation before the mate has occurred. This can be of great use in the tactical search, since a mate threat can be examined further to determine whether or not it is part of a successful variation. I would argue in favour of the inclusion of such a routine in any chess program – after all, the object of the game is to force mate.

# 6.1 A Computer Chess Tutorial

**Norman Whaland**

First published in BYTE , October 1978, pp. 168-181. Copyright © 1978 McGraw-Hill, Inc., New York 10020. Used with permission.

On February 20 1977, the Minnesota Open chess tournament was won by a computer program, Northwestern University's CHESS 4.5. This was a far better result than any program had previously achieved, considering that all the other entrants in the tournament were human beings. An improved version, CHESS 4.6, went on to wrest the world computer chess championship from the Soviet program KAISSA (Jennings, 1978). Professional chess players are beginning to worry about the competition from machines. They would seem to have little to fear at the moment, however. The consensus is that CHESS 4.5's tactical skill is impressive but its strategy is weak.

Against such competition, what can a personal computer experimenter expect to accomplish? Perhaps a great deal. There have been few new ideas in computer chess since Claude Shannon (1950a) outlined the basic principles in a paper published in 1950. (The superiority of CHESS 4.6 is due primarily to faster hardware.) Experimenters can participate in the search for the conceptual breakthroughs that will be needed before computer programs can be a match for the best human players. With that thought in mind, this article deals with the questions: What is a good structure for a chess program? What are the major functions that it must perform? In what directions can we seek innovations?

### The Game Tree

To get a notion of what a chess program must do, let's look at a position from an actual game (see figure 1).

First we must grasp the important features of the position. White has an extra pawn, a passed pawn far from Black's King. Black's mobility is very limited: neither the Knight nor the Rook can move. Black's Bishop is attacking White's Rook and, indirectly, the Bishop behind it. Of less importance, because of Black's lack of mobility, is the fact that two of White's pawns are unguarded. White's task is to save his Rook and to profit from Black's lack of mobility. White should win if he can find satisfactory solutions to these problems.

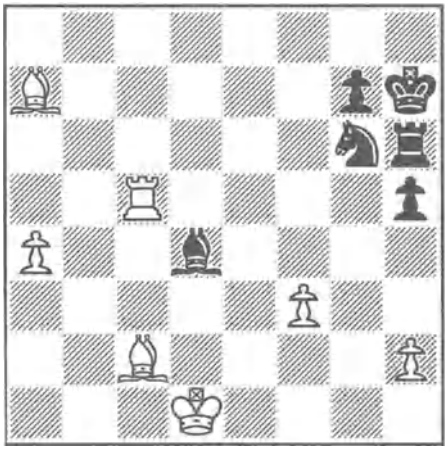


*Figure 1: White to play. This example illustrates some of the basic problems of strategy and tactics that must be evaluated by any chess playing computer in a typical position. The computer (White) must evaluate a variety of possibilities: two good first moves for White include 1 ♖c5-c7 and 1 ♗c2×g6+. 1 ♖c5-c7 threatens ♗a7×d4. Therefore Black must either exchange bishops or gain time by the counterattack 1 ... ♗d4-e5. If 1 ... ♗d1×a7, White must complete the exchange by playing 2 ♖c7×a7 or 2 ♗c2×g6+, and* so on. The position is analysed in detail in the game tree shown in figure 2.

Next we calculate variations — sequences of moves that we would visualize in an actual game before deciding on a move to play. We will follow a systematic procedure that will serve as a first approximation to a computer program. We construct a tree

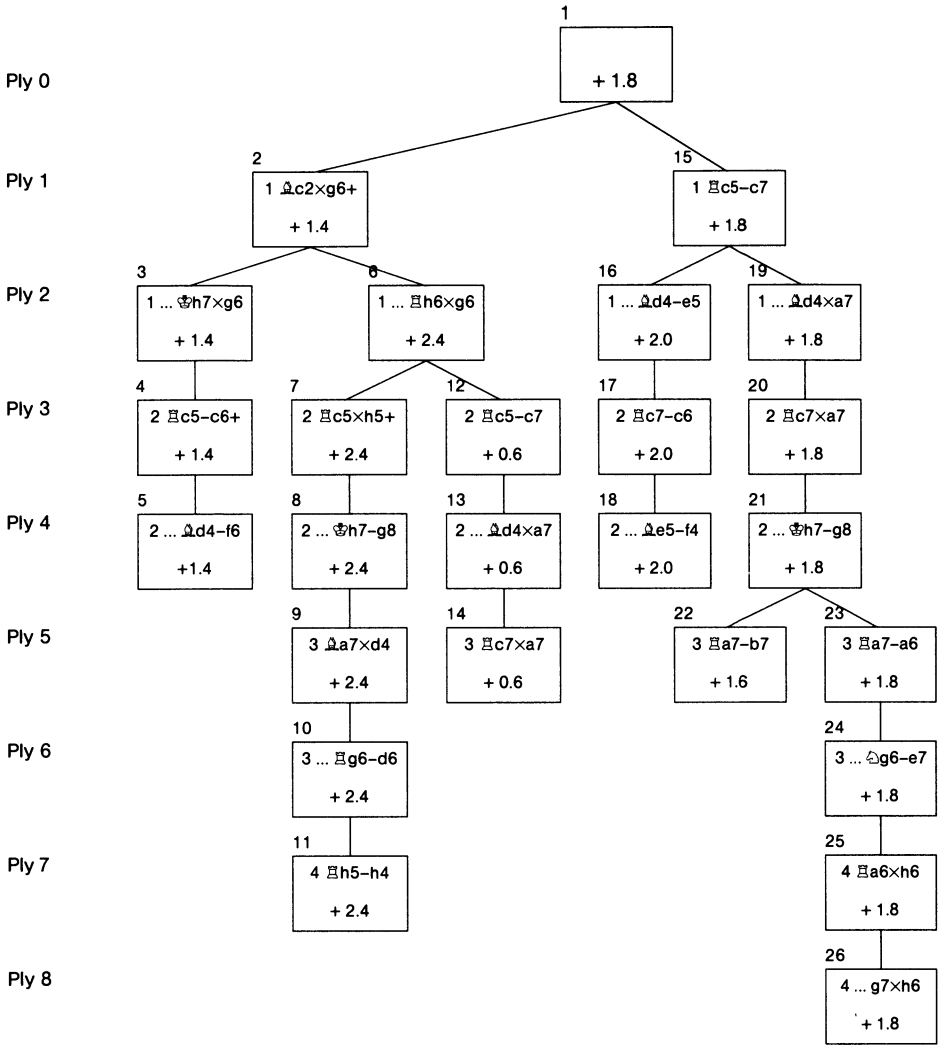



*Figure 2: A game tree developed from the position in figure 1. Each node represents a position; the root, the initial position. The move leading to the position is written in the top of the box, the evaluation of the position in the bottom. The number above the box identifies the node. A node's ply number is its distance from the root.*

whose nodes represent positions and whose edges represent moves. The variations are the paths from the root to the leaves. Initially, the tree will consist of one node representing the given position. We expand the tree as follows:

Expansion – Choose a leaf that has not been marked as final. (If one cannot be found, the expansion phase is ended). Either mark is final or select a set of legal moves in the position represented by the node. To the leaf attach *sons* representing the positions reached by the moves. Repeat from the beginning.

This procedure might yield the tree shown in figure 2. The size of the tree has been limited somewhat for illustrative purposes. Some of the variations I considered and rejected are not included. Most programs generate much larger trees since it is hard to build into a program the chess knowledge needed for rigorous selection of

moves. The length of paths in the tree is expressed in *plies (half moves)*. A move consists of a play by one player and a response by the other; a ply is a move by one player alone. Because the term *move* can be confusing (the chess literature speaks of three *moves* ahead for example, but are two or three moves by the opponent meant to be included?), in discussions of chess programming one speaks more precisely of a 5 or 6 ply look ahead.

In the expansion procedure, no rule was given for deciding whether to expand a node or for selecting the moves. To gain insight into the way human players make these choices, let us consider the variation that runs down the right hand side of the tree. In the initial position, Black threatens ... ♗d4×c5. White can either make a counterthreat or move his Rook to guard the Bishop. Thus the possible moves include 1 ♗c2×g6+, 1 ♖c5-c7 and 1 ♖c5-a5. I rejected the last alternative because the Rook would have less mobility on a5 and it seemed unimportant to keep it on the fifth rank. 1 ♖c5-c7 threatens ♗a7×d4 and moving the Bishop to another diagonal allows ♗a7-e3, attacking Black's Rook. Therefore, Black must either exchange bishops or gain time by the counterattack 1 ... ♗d4-e5. If 1 ... ♗d4×a7, White must complete the exchange by playing 2 ♖c7×a7 or 2 ♗c2×g6+. The latter move was omitted because the reply 2 ... ♖h6×g6 leads to the position at node 13 (see figure 2), already seen to be unsatisfactory for White. After 2 ♖c7×a7 White threatens ♖a7-a6 followed by the exchange of all the pieces and the triumphant advance of the Queen's Rook pawn (a-pawn). Black must play 2 ... ♔h7-g8 or 2 ... ♔h7-h8. The square closer to the center was chosen on general principles.

Free of threats at last, White can move in pursuit of his goals. White could play 3 ♖a7-b7 preparing to position his Rook behind the passed pawn; 3 ♔d1-d2 bringing his king toward the centre; 3 a4-a5 advancing the passed pawn; or 3 ♖a7-a6 to force the exchange of rooks. Somewhat arbitrarily, I included in the tree only the two moves that seemed best – 3 ♖a7-a6 forced the knight to move. Obviously it should approach the passed pawn, but it is not immediately clear which square is best. After 3 ... ♘g6-e7 4 ♖a6×h6 g7×h6 the assessment is clear; having two passed pawns in a minor-piece ending, White should win easily. There is no need to consider other knight moves, because the effect on the evaluation of the position would be too small to affect White's choice of move in the initial position.

From this brief discussion we can see some of the factors that determine the selection of moves. When there is a definite threat, it is necessary either to answer the threat directly or to make a counter-threat. Otherwise you must decide which goals are most important and choose the moves that best advance these goals. When two moves have similar effects, not much is learned by including both in the tree, particularly at a deep level.

We can also see some of the reasons for terminating a node (that is, choosing not to expand it). In this example, a node is terminated when the position can be evaluated sufficiently well or when the previous move was not forcing and the side to move has no forcing move that accomplishes anything. At node 14, for example, it is already clear that White doesn't have a won position, and it follows that one of his moves must have been a mistake. Thus we can evaluate the position sufficiently well (but not accurately: further analysis would lower the estimated evaluation given in the figure).

At node 5 White has the forcing move 3 ♗a7-e3, but after 3 ... ♖h6-h8 his position hasn't improved.

We consider these moves but don't add them to the tree, because the resulting position is merely compared with the position at node 5, not evaluated.

Once the tree is complete, the next step is to evaluate the terminal positions:

> Evaluation – Label each leaf with the value of the position from the point of view of the player whose turn it is to move in the *initial* position. Positive values mean the player has the advantage; negative values mean the player's opponent does. A value of ± 1 means an advantage barely enough to win; a value of ± 2 means an easy win (see figure 2).

In the present example, material, mobility and pawn structure were the most important factors in making the evaluation. In a middle game position, king safety would also be taken into consideration.

The final step is a completely mechanical procedure called the *minimax* algorithm, which is guaranteed to choose the best move provided the evaluations are accurate and that the best move at each node is included in the tree.

> Backup – Select an unevaluated node, all of whose sons have been evaluated. If the node is at even ply, label it with the maximum of the sons' values; at odd ply, choose the minimum. Repeat from the beginning until all of the nodes have been evaluated. Then choose the move leading to the ply-1 node with the greatest value.

This method of assigning evaluations to non-terminal nodes is based on the assumption that each player always makes the best move. The minimax algorithm will *not* always choose the move that affords the best winning chances against a weak opponent.

Our 3 part procedure for generating a game tree is somewhat unnatural. For one thing, a person analyzing a position would return to the expansion phase if the moves originally selected didn't work out as well as expected. Also, the evaluation phase reflects the human assessment process poorly. No provision is made for recording degree of confidence in the evaluation. Human players make relatively coarse absolute evaluations: they judge which of two similar positions is better, but do not attempt to assign slightly different values to them.

In chess programs, expansion, evaluation and backup are carried out simultaneously. One reason is that time can be saved by using backed-up values to demonstrate that some nodes need not be expanded at all. For example, the variation 1 ♗c2×g6+ ♖h6×g6 2 ♖c5×h5+ gives White a great advantage. We say that 2 ♖c5×h5+ refutes 1 ... ♖h6×g6. Once one refutation is found, it is pointless to look for another. 2 ♖c5-c7 need not be considered if not considered already.

What does this mean in terms of the minimum algorithm? Once node 3 has been assigned the value +1.4, we know that the value of the minimizing node 2 will not be any greater. Similarly, once node 7 has the value +2.4, we known that the value of the maximizing node 6 will not be any less. Therefore the minimax algorithm will not choose the value of node 6, and it can be eliminated from the tree without expanding node 12. Although in this example only one branch can be eliminated in this way, it is an important method for limiting the size of the "bushier" trees generated by chess programs.

We have seen that there are three major aspects of chess reasoning that need to be analyzed to create a chess program: selection, termination and evaluation. The handling of these functions by existing programs is only a crude approximation to the human reasoning process. It has proven particularly difficult to limit the number of moves considered at each node without inadvertently eliminating the best move. Consequently, CHESS 4.6 uses no selection or termination at a depth of less than six plies, and generates trees with hundreds of thousands of nodes. Even those programs that exercise some selection generate, in most cases, trees too large to store in programmable memory. Fortunately our procedure can be reformulated so as to require only a small part of the tree to be retained in memory at any time.

### The Depth-First Minimax Procedure

A tree can be traversed systematically by the following procedure:

> Start at the root – At each step, move to the leftmost unmarked son and mark it. If there is no unmarked son, move to the father. If there is no father, stop. (The terms *son, father* and *brother* are analogous to those in a family tree.)

The depth-first minimax procedure traverses the game tree in this way, simultaneously doing the expansion, evalation and back-up. Storage is required only for one path from root to leaf and for the brothers of the nodes on the path.

Figure 3 shows one way to organize the procedure. The processing is centered on the *ply table,* so named because it is indexed by the ply number i. The entries in the ply table correspond to nodes in the game tree. Each entry contains three fields; L(i), a pointer to the list of moves selected at the node; M(i), the move currently being processed; and E(i), the evaluation. The data area P contains the board position As the tree is traversed, P is modified to

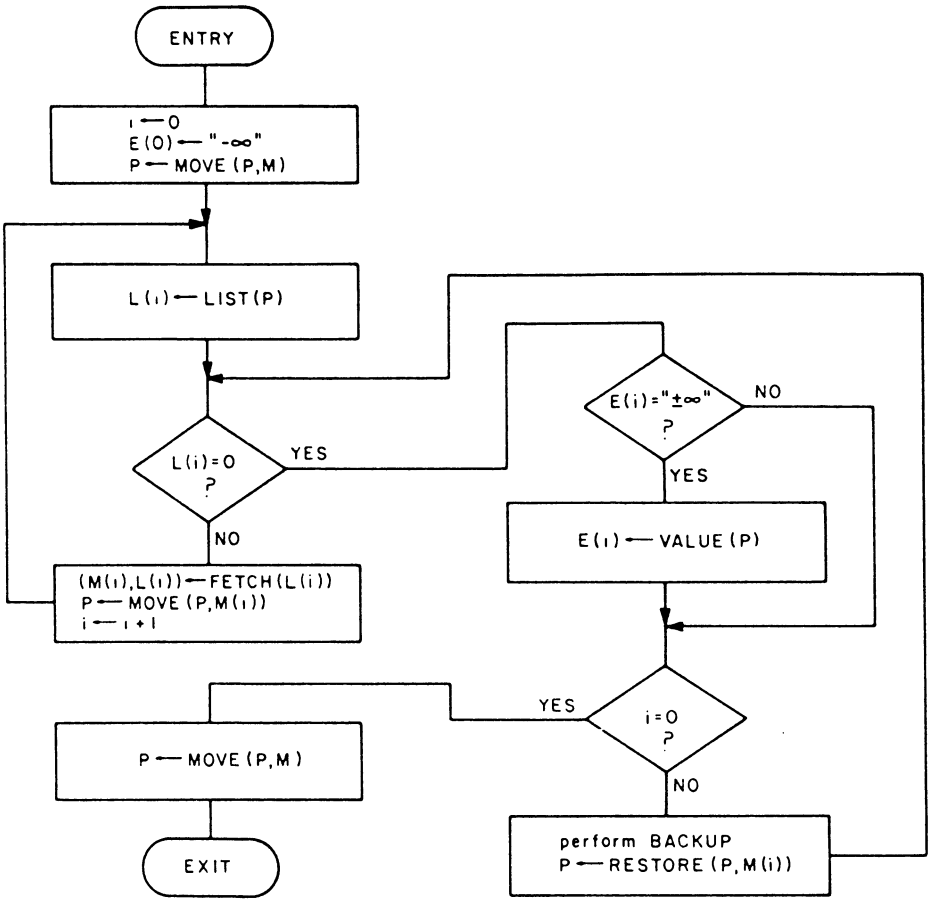



*Figure 3: A routine to choose a move. A ply table (so called because it is indexed by the ply number, i) is used to choose moves. (A ply is a move on the part of the player, two plies equal one chess move.) The entries in the ply table correspond to nodes in the game tree (see figure 2). Each entry contains the three fields: L(i), a pointer to the list of moves selected at each node, M(i), the move currently being processed; and E(i), the evaluation. Most of the subroutines are written as functions in order to show which data areas they use and affect. Only those data areas that play a central role are indicated. ∞ refers to a number which is larger than any returned by subroutine* VALUE. *Its additive inverse − ∞, is used as the initial value of E(0).*

show the position at the current node. At the start of the routine, the position is as it was presented to the opponent. The routine applies the move in location M to the position, chooses its move, stores it in M, and applies it to the position.

The subroutines named in figure 3 are discussed briefly here and in greater detail in the following sections. MOVE applies a move to the board representation P. It may also up-date auxiliary information describing the position. RESTORE simply reverses the changes made by MOVE. LIST generates the list of selected moves and places a pointer to the list in L(i). If the list is empty, L(i) is set to equal zero. FETCH moves the first move on the list to M(i) and advances the pointer L(i) to the next move. VALUE places the evaluation of a terminal position in E(i). BACKUP moves the evaluations E(i) up the table in accordance with the minimax rules.

Programs that generate a large tree generally use a depth-first search and have an overall structure similar to that shown in figure 3. The inflexibility of the depth-first search is a significant disadvantage, though.

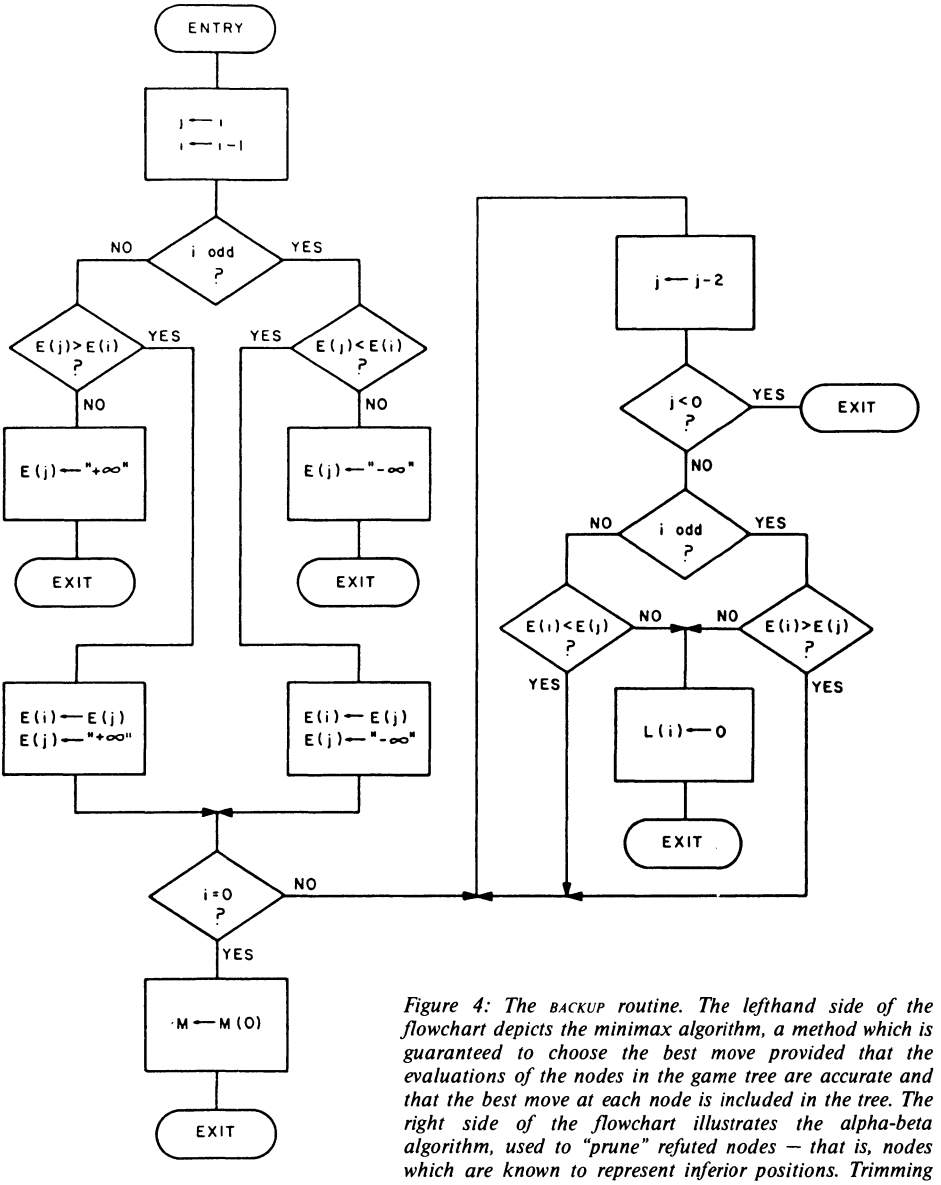



*Figure 4: The* BACKUP *routine. The lefthand side of the flowchart depicts the minimax algorithm, a method which is guaranteed to choose the best move provided that the evaluations of the nodes in the game tree are accurate and that the best move at each node is included in the tree. The right side of the flowchart illustrates the alpha-beta algorithm, used to "prune" refuted nodes — that is, nodes which are known to represent inferior positions. Trimming the tree in this way reduces the amount of information that must be stored in memory and speeds up the evaluation process (see text).*

For example, suppose that shallow analysis of the first ply-1 move casts doubt on its value. Time might be saved by proceeding at once to the other moves and returning to the first move only if they seem no better. But in a depth-first search, the decision to terminate a variation cannot be changed on the basis of later information. Consequently, programs that generate small trees usually maintain the entire tree in programmable memory. Then it is possible to skip around in the tree, expanding those nodes that look most promising. Although such programs aren't structured like depth-first programs, they perform many of the same functions, and so the following discussion

of the subroutines partially applies to them.

### The BACKUP Routine

The movement of values up the tree is controlled by BACKUP, which also prunes refuted nodes. The procedure is shown in detail in figure 4.

The minimax part of the procedure manipulates the fields E(i), which can contain initial values, provisional values, and final values. The initial values are $-\infty$ for even ply and $+\infty$ for odd ply, where $\infty$ is a number larger than any returned by VALUE. E(i) is always set to the initial value when the table entry is not being used. The values produced by VALUE are final values. Whenever a final value E(i) appears in the ply table, BACKUP compares it with the value E(i−1). E(i) replaces E(i−1) if i is even and E(i−1) is greater than E(i) or if i and E(i−1) is less than E(i). E(i−1) then contains a provisional value. A provisional value becomes final when the move list at its ply becomes empty. Whenever E(1) replaces E(0), M(0) is saved in M. As a result, M ultimately contains the first move in the list L(0) that produces a maximum final value in E(1).

### The Alpha-Beta Algorithm

The elimination of refuted moves from the tree is accomplished by a procedure called the *alpha-beta* algorithm. (See, for example, Slagle and Dixon, 1969). To get a clear understanding of the alpha-beta algorithm, let's view the minimax procedure as a contest to determine which leaf's evaluations will reach the root. The provisional evaluations in the ply table are obstacles to the process of an evaluation up the tree. Maximizing nodes stop evaluations that are too small; minimizing nodes stop those that are too large. Suppose that i is odd and that a new evaluation has just been backed up into E(i). Because E(i) is now smaller, it may not be able to get past the provisional values in maximizing nodes higher in the tree. To find out, the alpha-beta algorithm searches at even ply for an evaluation greater than or equal to E(i). If one is found, the level-i node is terminated by setting L(i) to zero. The procedure is similar when i is even. The termination of a node by the alpha-beta algorithm is called an *alpha-beta cutoff* or just a *cutoff*.

### The LIST Routine

The decisions that determine the size and shape of the game tree are made by LIST. It has three main functions: termination of nodes, selection of moves and the sorting of the list of moves. If the program is to play under a time limit LIST must also monitor the elapsed time and modify its decisions accordingly. Existing programs handle these functions in widely differing ways. Their selection and termination procedures range from trivial to complex. It's discouraging that the trivial methods have so far yielded the best results, for surely a sophisticated LIST routine will be needed for first-rate chess.

Most chess programs condition termination primarily on depth and the availability of certain types of forcing moves. The simplest method would be to terminate always at some fixed depth. Then VALUE would have to give special handling to positions with an exchange in progress, lest material be reckoned incorrectly. Consequently, many programs use two depth limits. Beyond the first limit are selected only certain forcing moves, typically checks and captures. Termination occurs, of course, when there are none. At the second depth limit termination always occurs.

Other criteria for termination have been tried. The Ostrich program (developed on a Data General Supernova minicomputer at McGill University in Montreal, Canada) terminates variations in which material is sacrificed and not recovered within three plies (Newborn, 1975). Several people have suggested that termination should occur only if the position can be accurately evaluated. The Newell-Shaw-Simon program (Newell et al. 1958) used this philosophy. When the entire tree is maintained in programmable memory, termination decisions as such need not be made at all. For example, the program COKO expands those nodes that promise the greatest yield of information, no leaf being permanently excluded from consideration.

For selection and sorting, LIST might assign to each legal move a plausibility rating designed to indicate the possibility that the move will prove best. Many programs don't explicitly assign a rating;

nevertheless, it is convenient to imagine that their decisions are based on an implicit rating. Selection and sorting can then be done as follows: select all moves with ratings greater than some threshold. If too few moves are selected, add highest-rated moves to make up the minimum number. (The threshold and number of moves might depend on depth). Sort the selected moves by rating.

For sorting, the requirements on the rating procedure are stringent. It suffices that moves good enough to cause cutoffs often appear early in the list. Occasional inaccurate ratings will merely increase the processing time, not cause a blunder. The number of cutoffs can be markedly increased by simply assigning high ratings to a few easily defined categories of moves: captures, checks, moves by attacked pieces, etc. Another simple rating method is to assign a high rating to moves that have proven to be good in other parts of the tree. For example, the "killer" heuristic assigns to a refutation found at one node a high rating at its brother nodes. This heuristic works well in positions containing threats, because all moves that ignore the threat can be refuted by the same reply.

For selection, the plausibility rating must be more accurate. A best move markedly better than the second best move must only rarely receive a rating low enough to cause its rejection. Simple criteria that are adequate for sorting are bound to fail. The rating must be based on all the move's important effects, which in turn can be determined only by elaborately tracing the relationship of the pieces. For this reason, programs that use selection generally maintain a tactical description of the position. In the program we are considering, it is the responsibility of the MOVE routine to keep such information current.

### The VALUE Routine

The evaluation is usually computed as a sum of numerical scores, each representing one aspect of the position. Chess programmers tend to include only those aspects that are easiest to program. Unfortunately they are not always the most important ones. The traditional chess literature is more explicit about evaluation than about selection and termination. The books in the bibliography are particularly helpful.

The evaluation of the position depends mainly on material mobility and vulnerability. The calculation of material is straightforward, although experts disagree about the exact values of the men. CHESS 4.6 uses the values: P=100, N=325, B=350, R=500 and Q=900. Like most programs, it adjusts the material score to encourage the exchange of pieces when ahead in material. The values of the pieces vary with the strategic character of the position: Rooks are better when the opponent has weak pawns, Knights are better in blocked positions, and so on. Such considerations are important, but I know of no program that takes them into account.

The assessment of mobility is more difficult. Counting the legal moves of each man is easy but inadequate. It is necessary also to take into account the exclusion of men from squares controlled by the opponent and the immobilization of men by defensive functions, such as the shielding or guarding of a man or important squares. Detecting these factors is complicated and may involve tracing the relationships between several men.

Under vulnerability we have to consider unguarded pieces, the safety of the King, weaknesses in the pawn structure and pieces exposed to attack by less valuable men. Pawn weaknesses are easy to detect, and most programs take them into account. Measuring danger to the King is more complicated, but it is easy to detect some of the relevant features, such as disturbances of the King's pawn cover or the absence of friendly minor pieces nearby. Detecting unguarded and exposed pieces seems to be relatively simple, but oddly it is often neglected.

### The MOVE Routine

Because of the rapid expansion of the game tree with depth, most of the processing time is spent in selecting and evaluating the terminal positions. It is therefore desirable for MOVE to maintain, along with the current position, information helpful to the LIST and VALUE routines. For example, it is more efficient for MOVE to keep track of changes in the material score than for VALUE to scan the board to do the same thing. Also, some programs maintain lists of the locations of each side's men to facilitate the generation of moves.

We have seen that sophisticated LIST and VALUE routines would have to detect relationships between the men. Since each move changes only some of the relationships, it is more efficient to compute them in MOVE than to compute them all from scratch in LIST and VALUE. In general, the features needed for selection are the same as those needed for evaluation. For example, a backward pawn affects the evaluation and also suggests moves for both sides. The possessor of the pawn will try to advance it or protect it, while his opponent will try to prevent its advance and win. Likewise any advantage suggests moves to maintain and exploit it; any disadvantage, moves to eliminate or mitigate it.

## Levels of Skill

The United States Chess Federation rates its members at eight levels of skill based on performance in tournaments. In descending order they are Senior Master, Master, Expert and Classes A through E. From time to time computer programs have played in rated tournaments. Until recently their performance has been in the Class C or Class D range. Against this background the strong showing of CHESS 4.5 startled everyone. At the conclusion of the Minnesota Open its rating had risen to Expert. It is still too early to assess its true strength, however. Although it is strong tactically, its grasp of strategy is well below the Expert level. The weak showing of Class A players against CHESS 4.5 was caused largely by their unfortunate tendency to get into tactically complex positions, thereby playing into the computer's strength. The program may not be so successful once people learn how to play against it.

The sudden improvement in CHESS 4.5 coincided with the transfer to a faster machine, enabling it to search two plies deeper in most positions. This supports the belief that chess skill depends mainly on the number of moves one can see ahead. It's difficult to give a precise equivalence between depth of search and level of skill, though. The following rule of thumb is, I think, close enough to the truth to give some idea of the design requirements for strong programs. Let a search depth of four plies correspond to Class C, and assume that each additional two plies yields an increase of one level of skill. Thus, play at the Expert level would require a 10 ply search.

## The Exponential Explosion

The depth of search is limited by the increase in the size of the game tree with depth. Suppose that B moves are selected at each node. This number is called the *branching factor* or *fanout*. If D represents the depth of search, the tree has $B^D$ leaves; the tree grows exponentially with depth. A typical position might have 30 legal moves, and if no selection is exercised, the tree will have 27,000 leaves at a depth of three plies. This is probably already too large a tree to examine with a microcomputer in a reasonable time. We have not, however, yet taken alpha-beta pruning into consideration.

| Search Depth | Number of Terminal Nodes |
|---|---|
| 2 | 59 |
| 3 | 929 |
| 4 | 1,800 |
| 5 | 27,900 |
| 6 | 54,000 |
| 7 | 837,000 |

*Table 1: Tree size as a function of search depth (D), assuming exhaustive search and the maximum possible number of alpha-beta cutoffs. The branching factor (B) is assumed to be 30. The number of terminal nodes is $2B^{D/2}-1$ when D is even and $B^{(D+1)/2}+B^{(D-1)/2}-1$ when D is odd.*

The effectiveness of the alpha-beta algorithm depends on how well the move list is sorted. The greatest possible reduction in tree size is achieved when the best move is always first on the list. Table 1 shows the tree size under this condition, assuming a branching factor of 30. Clearly, exhaustive search beyond six plies is impossible for a small computer. To play strong chess a microcomputer will have to use selection. The question is: how much?

To derive a relationship between the branching factor and the depth of search, we have to make some assumptions. Let us assume that we must limit the size of the tree to 10,000 leaves, and that the alpha-beta algorithm reduces the effective branching factor from B to $B^{2/3}$. Then table 2 gives the desired relationship. Although

much guesswork went into this table, it seems safe to conclude that an Expert-level program must be very fast or very selective.

| Branching Factor | Depth of Search | Tactical Skill |
|---|---|---|
| 31.6 | 4 | Class C |
| 15.8 | 5 | |
| 10.0 | 6 | Class B |
| 7.2 | 7 | |
| 5.6 | 8 | Class A |
| 4.6 | 9 | |
| 4.0 | 10 | Expert |
| 3.5 | 11 | |
| 3.2 | 12 | Master |
| 2.9 | 13 | |
| 2.7 | 14 | |
| 2.5 | 15 | |

*Table 2: Depth of search (D) and tactical skill as a function of the branching factor (B). It is assumed that the alpha-beta algorithm reduces the effective branching factor to $B^{2/3}$ and that 10,000 terminal nodes can be processed. These assumptions yield the formula $b=10^{6/D}$.*

## The TECH Program

How simple can a program be and still play reasonable chess? The TECH program (Gillogly, 1972) was developed in order to answer that question. It would be a good model to follow if you want to have a running program in the shortest possible time. Despite its simplicity, or perhaps because of it, TECH placed higher in computer chess tournaments than some of the more complicated programs. It is good enough to defeat only inexperienced human players, but that is true of most programs. For the newcomer to chess programming the design of a TECH type program would be a good way to gain experience.

TECH considers all moves to a fixed depth, beyond which it considers only captures. The evaluation of terminal positions is based only on material. Hence there is no need for a VALUE routine; the evaluation is computed on the run whenever captures occur. When the program has an advantage of two pawns or more, it reduces the value of its own pieces slightly so that exchanges are favoured. TECH sorts moves for two purposes; to increase the frequency of alpha-beta cut-offs, and to bring factors other than material to bear on the choice of move. At ply 2 and lower, captures are considered first and the killer heuristic is used. The positions at ply 1 are assigned a rating that includes such factors as the number of legal moves, the advancement of the center pawns, and the proximity of the pieces to the center, to the enemy king, and to passed pawns. The program expands the ply-1 nodes in descending order of the rating, which thus breaks ties in the backed-up evaluation.

Because TECH does very little processing at each node, it is able to generate a relatively large tree. Cutoffs are frequent; basing the evaluation only on material ensures that the alpha-beta comparison will often give an equal result. The ply 1 rating procedure could be made more elaborate without slowing down the program noticeably. It would be interesting to see how much the program's play could be improved in this way.

## New Directions

Chess programming is still a young field. There are many ideas that have never been tried or never been developed sufficiently to determine their value. Experimentation by computer enthusiasts could play a major role in developing the innovations that will be needed for a Master-level chess program. Some of the less successful chess programs use ideas worth further consideration. Papers describing some of these programs are listed in the bibliography. Additional ideas can be found by comparing the behaviour of programs and human players.

## Some Ideas for the Future

Chess games between computers are often dull because the programs don't follow any plan. They pursue general goals such as development and control of the center, but don't formulate goals specifically appropriate for the position at hand. Goals are represented in the evaluation and rating procedures. Setting a specific goal is accomplished by making changes in the procedures. For example, the general goal of

center control might be implemented in part by a term in the evaluation polynominal for the number of pieces bearing on the center. A routine for setting specific goals might add a term for the number of pieces bearing on a center square that the routine had determined to be particularly important.

Here are some of the types of specific goals that occur frequently:

- Get control of a key square
- Attack an area of the board where the opponent is weak.
- Free an immobile piece.
- Save an attacked man.
- Maneuver a particular piece to a square where it will have a strong influence.

It should be fairly easy to determine how to modify the evaluation and rating procedures in such a way as to set these goals. However, it might be difficult to devise a procedure for choosing the specific goals.

Most chess programs spend almost all of their time considering silly moves. There are two main types of silly moves: moves irrelevant to the important goals of the position, and sacrifices that gain nothing that could be worth the cost. To safely reject irrelevant moves, the program must avoid overlooking important goals, lest it reject a vital move and blunder. Because of the difficulty of writing a comprehensive goal setting routine, it is not surprising that highly selective programs haven't performed well. Nevertheless, it is only a matter of time until enough chess knowledge is formalized to permit accurate selection. In the meantime, it might be possible to devise an algorithm that would reliably identify at least some of the silly moves.

Chess programs usually compare moves or positions by assigning numbers to them and then comparing the numbers. This method precludes certain possibilities of reliably rejecting moves. Suppose we have an algorithm that, given two similar positions, lists all of the important differences between them, together with limits on the effect each difference could have on the evaluation. It is then sometimes possible to determine which position is better, even though it might not be possible to evaluate either position reliably. We saw an example of this in the analysis of figure 1, where 2 ... ♔h7-g8 appeared to be clearly better than 2 ... ♔h7-h8. The position-comparing algorithm could be used for selection and for a variant of a alpha-beta pruning. We meet with a familiar difficulty, however: the algorithm would have to incorporate comprehensive knowledge in order to avoid overlooking important differences.

To summarize, a program to play Master-level chess might contain algorithms to:

- Find the important features of the position.
- Determine the relevant goals and rate their importance.
- Compare two similar positions to determine whether one is clearly better than the other.
- Select a list of reasonable moves in a given position.

Each algorithm would use the results of the previous ones in the list. The program would contain much chess knowledge which would best be represented in a form both compact and easily alterable.

**Prerequisites**

How good a chess player do you have to be to tackle some of these problems? Most people need only a basic understanding of chess strategy and the ability to find simple combinations. Far more important than chess knowledge is the ability to teach what you know to a very dull, nonhuman pupil. You will have to be able to state explicitly the reasons for the choices you make while analyzing a chess position. It's not as easy as it sounds. Above all, it's important to keep in mind that writing a chess program is a big project. A methodical approach, using structured programming and careful documentation, is absolutely essential.

**Concluding Remarks**

In this article I have tried to cover the basic ideas of chess programming and indicate some new directions for experimentation. I hope that many of you will be stimulated to get involved in this growing field of research.

## Glossary

*Analysis:* the calculation of variations in order to assess a position or find the best move.

*Backward pawn:* a pawn that lags behind the pawns on the adjoining files. When the opponent has no pawn on the file, a backward pawn is usually a serious weakness.

*Development:* the process of initially moving the pieces from their original squares.

*Diagonal:* a diagonal row of squares on the chessboard.

*File:* a vertical row of squares on the chessboard.

*Material:* the chess pieces considered as assets. A pawn is traditionally considered to have a material value of one unit. Programs often use smaller units to avoid using fractions for positional advantages.

*Minor piece:* a Knight or a Bishop.

*Passed pawn:* a pawn not hindered by enemy pawns on its file or on adjoining files.

*Piece:* a chess piece other than a pawn.

*Positional advantage:* any advantage other than an advantage in material.

*Rank:* a horizontal row of squares.

*Strategy:* that aspect of chess concerned with long-range planning.

*Tactics:* that aspect of chess concerned with move-by-move changes in the position. Tactics include the methods for winning material and advancing strategic plans.

*Variation:* a sequence of moves considered as one of several from a given starting position.

# 6.2 Using Patterns and Plans in Chess

**David Wilkins**

First published in *Artificial Intelligence,* volume 14, 1980, pp. 165-203.
Reprinted by permission of North-Holland Publishing Company, and of the author.

**Abstract**

The purpose of this research is to investigate the extent to which knowledge can replace and support search in selecting a chess move and to delineate the issues involved. This has been carried out by constructing a program, PARADISE (PAttern Recognition Applied to DIrecting SEarch), which finds the best move in tactically sharp middle game positions from the games of chess masters. It encodes a large body of knowledge in the form of production rules. The actions of the rules post concepts in the data base while the conditions match patterns in the chess position and data base. The program uses the knowledge base to discover plans during static analysis and to guide a small tree search which confirms that a particular plan is best. The search is "small" in the sense that the size of the search tree is of the same order of magnitude as a human master's search tree (tens and hundreds of nodes, not thousands to hundreds of thousands as in many computer chess programs).

Once a plan is formulated, it guides the tree search for several ply and expensive static analyses (needed to analyize a new position) are done infrequently. PARADISE avoids placing a depth limit on the search (or any other artificial effort limit). By using a global view of the search tree, information gathered during the search and the analysis provided by the knowledge base, the program produces enough terminations to force convergence of the search. PARADISE has found combinations as deep as 19 ply.

## 1. Introduction

One of the central concerns of artificial intelligence is expressing knowledge and reasoning with it. Chess has been one of the most popular domains for AI research, yet brute force searching programs with little chess knowledge play better chess than programs with more chess knowledge. There is still much to be learned about expressing and using the kind of knowledge involved in playing chess. CHESS 4.7 is probably the best chess program, and its authors make the following comments in Slate et al (1977):

> But why is the program so devoid of chess knowledge? It is not because we are deliberately trading it for speed of execution.... It is not quite true to say that we don't know how to feed additional knowledge to the program.... Our problem is that the programming tools we are presently using are not adequate to the task. Too much work is needed to encode the sizable body of knowledge needed to significantly improve the quality of play.

Human masters, whose play is still much better than the best programs, appear to use a knowledge intensive approach to chess (see Charness, 1977). They seem to have a huge number of stored "patterns", and analyzing a position involves matching these patterns to suggest plans for attack or defense. This analysis is verified and possibly corrected by a small search of the game tree. Unlike the searches conducted by most programs, the human master's search usually has specific goals to accomplish and returns useful information which affects the analysis of other lines. The purpose of this research is to investigate the issues involved in expressing and using pattern-oriented knowledge to analyze a position, to provide direction for the search, and to communicate useful results from the search.

To reduce the amount of knowledge that must be encapsulated, the domain of the program has been limited to tactically sharp middle game positions. The phrase "tactically sharp" is meant to imply that success can be judged by the gain of a material rather than a positional advantage. The complexity of the middle game requires extensive search, providing a good testing ground for attempts to replace search with knowledge and to use knowledge to guide the search and communicate discoveries from one part of the tree to another.

Since in our chosen domain, the attacking side can generally win material with

correct play, the program uses different models for the offensive and defensive players during its search of the game tree. The offensive player uses a large amount of knowledge, suggests many esoteric attacks, and invests much effort in making sure a plan is likely to work before suggesting it. The defensive player uses much less knowledge and effort, tries any defense that might possibly work, and tries only obvious counter attacks. Trying every reasonable defense enables the offense to satisfy itself that a line really does win. The size of the tree is kept small because the offensive player finds the best move immediately in most cases.

For a knowledge based program to achieve master level performance, it seems necessary that the knowledge base be amenable to modifications and additions. PARADISE provides for easy modification of and addition to its knowledge base (without adversely affecting program performance), thus allowing the knowledge to be built up incrementally and tuned to eliminate errors. This is accomplished by having the knowledge base composed of largely independent production rules which are written in a straightforward Production-Language (Wilkins 1979).

The goal is to build an expert knowledge base and to reason with it to discover plans and verify them with a small tree search. As long as this goal is reached, the amount of execution time used is not important. It is a design decision to sacrifice efficient execution in order to use knowledge whenever reasonable. When a desired level of performance is reached, the program could be speeded up considerably by switching to a more efficient representation at the cost of transparency and ease of modification.

Because it grows small trees, PARADISE can find deeper combinations than most chess programs. The position in Fig. 0 is an example of such a combination from an actual master game. White can mate by sacrificing his queen and playing nine additional moves (against black's line of longest resistance), the next to last move not being a check or capture. Thus the search tree must go at least 19 ply deep to find this combination. This is considerably deeper than any other current chess program could probe in a reasonable amount of time, but PARADISE solves this problem after generating a search tree of 109 nodes, in 20 minutes of CPU time on a DEC KL-10. (PARADISE is written in MacLisp.) The fact that PARADISE can accurately analyze all relevant lines in only 109 nodes shows that the program makes good use of its knowledge.

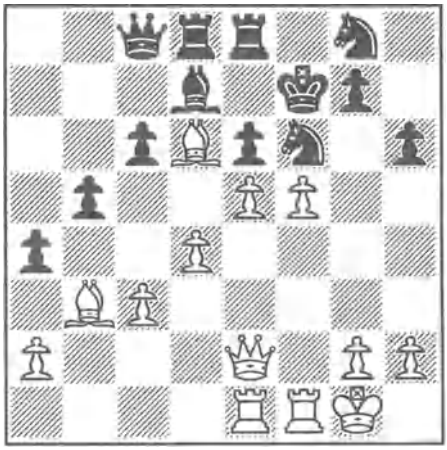


*Fig. 0. White to move*

| | | |
|---|---|---|
| 1 | ♕e2-h5+ | ♘f6×h5 |
| 2 | f5×e6+ | ♔f7-g6 |
| 3 | ♗b3-c2+ | ♔g6-g5 |
| 4 | ♖f1-f5+ | ♔g5-g6 |
| 5 | ♖f5-f6+ | ♔g6-g5 |
| 6 | ♖f6-g6+ | ♔g5-h4 |
| 7 | ♖e1-e4+ | ♘h5-f4 |
| 8 | ♖e4×f4+ | ♔h4-h5 |
| 9 | g2-g3 | any |
| 10 | ♖f4-h4 mate | |

## 2. Overview of PARADISE

PARADISE uses a knowledge base consisting of about 200 production rules to find the best move in chess positions. Every production has a pattern (i.e., a complex, interrelated set of features) as its condition. Each production can be viewed as searching for all instances of its condition pattern in the position. For each instance found, the production may, as its action, post zero or more concepts in the data base for use by the system in its reasoning processes. A concept consists of a concept name, instantiations for a number of variables, and a list of reasons why the concept has been posted. The data base can be considered as a global blackboard where productions write information. The productions are built up by levels, with more primitive patterns (e.g., a

```
((DMP1)
  (NEVER (EXISTS (SQ) (PATTERN MOBIL DMP1 SQ)))
  (NEVER (EXISTS (P1) (PATTERN ENPRIS P1 DMP1)))
  (ACTION ATTACK (OTHER-COLOUR DMP1) (LOCATION DMP1)) (THREAT (WIN DMP1)) (LIKELY 0)))
```

*Fig. 1. Production rule in* PARADISE *which recognises trapped pieces.*

pattern which matches legal moves) being used as building blocks for more complex patterns. A search tree (the major component of most chess programs) is used to show that one move suggested by the pattern-based analysis is in fact the best.

Fig. 1 shows one of the simplest productions in PARADISE. It attempts to find an offensive plan which attacks a trapped defensive piece. The first three lines in Fig. 1 constitute the pattern or condition of the production rule while the fourth line constitutes the action. The first line specifies one variable that must be instantiated. Because of its name, the variable must be instantiated to a defensive piece that is not a pawn. The next two lines in the production access two more primitive patterns that have already been matched, MOBIL and ENPRIS. The MOBIL pattern matches each legal move that can be made without losing the moving piece. If there are no squares which match MOBIL when DMP1 is the first argument, then DMP1 is trapped. Only in this case will there be any possible instantiations for DMP1 after the second line in Fig. 1 is matched. The third line prevents the pattern from matching if the trapped piece is already en prise. The idea is that in this case DMP1 should be captured outright rather than attacked. Thus this production matches only trapped pieces which are not en prise.

(In Fig. 1 the action of the production rule posts an ATTACK concept. This concept gives instantiations for two variables and tells the system that the opposite color of DMP1 would do well to attack the square which is DMP1's location. In addition one reason is given for suggesting this concept. This reason consists of attributes describing the threat of the concept and how likely it is to succeed. Concepts are discussed in more detail later in this paper. The more primitive patterns (such as MOBIL) and the language in which productions are written are described in detail in Wilkins (1979).

To offset the expense of searching for patterns in the position, problems must be solved with small search trees. In fact, PARADISE can be viewed as shifting some of its search from the usual chess game tree to the problem of matching patterns. PARADISE shows that its pattern-oriented knowledge base can be used in a variety of ways to significantly reduce the part of the game tree which needs to be searched. This reduction is large enough that a high level of expertise can be obtained using a large but still reasonable amount of resources. This reduction is also large enough that PARADISE avoids placing a depth limit (or any other artificial effort limit) on its search. The various ways in which the knowledge base is used are briefly mentioned below.

*Calculating primitives.* PARADISE's knowledge base contains about twenty patterns (productions without actions) which are primitives used to describe a chess position. These patterns are matched once for each position. They range from matching legal moves to matching patterns like MOBIL and ENPRIS. The primitives are relatively complex and calculating them is expensive.

*Static analysis* Given a new position, PARADISE does an expensive in-depth static analysis which uses, on the average, 12 seconds of cpu time on a DEC KL-10. This analysis uses a large number of productions in the knowledge base to look for threats which may win material. It produces plans which should guide the search for several ply and should give information which can be used to stop the search at reasonable points.

*Producing plans* The knowledge base is used to produce plans during static analysis and at other times. Through plans, PARADISE uses its knowledge to understand a new position created during the search on the basis of positions previously analyzed along that line of play, thus avoiding a static analysis. Plans guide the search and a static analysis is only occasionally necessary in positions created during the search. Producing a plan is not as simple as suggesting a move, it involves generating a large amount of useful information to help avoid static analyses along many different lines and to help detect success or failure of the plan as

it is executed. (Examples are given later in this paper.)

*Executing plans.* The execution of a plan during the tree search requires using the knowledge base to solve goals specified in the plan. This use of the knowledge base could be viewed as a static analysis where the system's attention is focused on a particular goal or aspect of the position. Such a focused analysis can be done at a fraction of the cost of a full analysis which examines all aspects of the position.

*Generating defensive moves.* PARADISE does static analyses and executes plans only for the offense (the side for which it is trying to win material). To prove that an offensive plan succeeds, the search must show a winning line for all reasonable (i.e., any move which might be effective) defensive moves. Productions in the knowledge base are used to generate all the reasonable defenses.

*Quiescence searches.* When a node is a candidate for termination of the search, PARADISE uses a quiescence search to determine its value. This quiescence search may return information which will cause the termination decision to be revoked. Productions in the knowledge base provide much of the knowledge used in PARADISE's quiescence search. Quiescence searches in PARADISE investigate not only captures but forks, pins, multi-move mating sequences and other threats. Only one move is investigated at each node except when a defensive move fails.

*Analysis of problem upon failure.* When either the defense or offense has tried a move that failed, the search analyzes the information backed up with the refutation in an attempt to solve the problem. The knowledge base is used in this analysis to suggest plans which thwart the opponent's refutation. In this way PARADISE constructs new plans using the information gained from searching an unsuccessful plan.

*Answering questions.* The search often requests certain kinds of information about the current position. The knowledge base is used to provide such information. The most frequent requests ask if the position is quiescent and if there are any obvious winning moves in the position.

The above list describes eight general functions the knowledge base performs for PARADISE. This paper explains how plans are produced and executed. The productions, the static analysis, the tree search, and mechanisms for communicating information from one part of the tree to another are discussed in Wilkins (1979).

## 3. The Need for Concepts

To understand a chess position, a successful line of play must be found. Human masters understand concepts which cover many ply in order to find the best move. A knowledge based program must also have the ability to reason with concepts that are higher level than the legal moves in a chess position. The following position (Berliner, 1974) illustrates this.

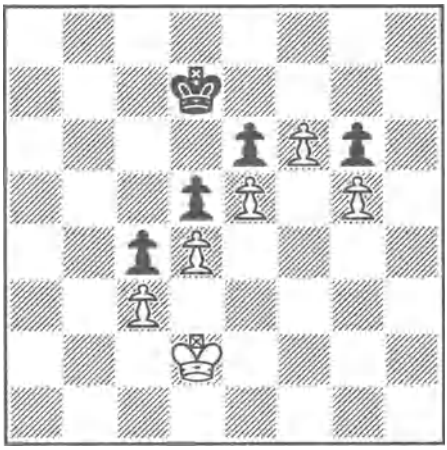


*Fig.2. White to move.*

In the above position, most computer chess programs would search the game tree as deep as their effort limit allowed and decide to play ♔d2-e3 which centralizes the king. They cannot search deep enough to discover that white can win since white's advantage would take more than 20 ply to show up in most evaluation functions used by computer programs. The point is that the solver of the above problem uses conceptual reasoning to solve it, and the concepts must be higher level than the legal moves in a chess position. A human master would recognize the following features (among others) in the above position; the black king must stay near f8 to prevent white's f-pawn from queening, the pawns form a blockade which the kings can't penetrate, and the kings can bypass this

blockade only on the a-file.

For a knowledge based program to solve this problem, it must be able to express concepts at the level of conceptualization used above. The system must combine and reason with these concepts in unforeseen ways. In the above example, the blockading concept would be used to develop the plan of moving the white king to the a-file in order to bypass the blockade but if black and white had pawns on b5 and b4 respectively then the blockading concept would be used to decide that neither king can invade the other's territory. Expressing and using concepts is a major problem for a knowledge-based chess program.

To clarify the problems involved in used concepts to reason about chess, a comparison to a well-known 'expert' system such as MYCIN is helpful. The following phrase is the action part of a typical MYCIN production rule (from Davis 1976):

*then there is suggestive evidence (0.7) that the identity of the organism is bacteroides*

Such an action effectively adds to the plausibility score for some possible answer. This presupposes that any one writer of all the rules knows every possible solution since such rules will never create a diagnosis that has not been mentioned in the action part of some rule.

Using actions similar to those used in MYCIN rules would be similar to a chess system where the action part of each rule was to add to the plausibility score of either
(1) one or more of the legal moves in the position or
(2) some prespecified object which could lead (through other rules) to recommending a legal move.

Such an approach is not satisfactory for the type of chess reasoning needed in the above problems. Concepts at a higher level then legal moves must be used in this reasoning, and objects cannot be prespecified (e.g. the extent of blockade must be created dynamically). Thus, PARADISE must reason by linking comcepts to create plans instead of filling in prespecified slots.

In most current expert production systems (such as MYCIN), the action parts of the rules are in some sense part of the solution or a pointer to something which can produce a solution. In chess the system must discover lines of play that were not implicit in the knowledge base. Thus the word 'create' is used. MYCIN has a small solution set (120 organisms) and can mention each solution in its rules, while a chess program must discover an enormous number of lines using a set of productions which is much smaller in size (by an order of magnitude or more) than the number of plans it must recognize. This can be seen by looking at the actual production rules: MYCIN's actions mention the names of objects while PARADISE's actions involve many variables whose instantiation is not determined until execution of the action.

## 4. Knowledge Sources and Concepts

When a production in PARADISE matches, it may have an action part which posts various concepts in the data base. The program is able to express and use concepts by dividing its knowledge base into various *Knowledge Sources.* Each Knowledge Source (KS) provides the knowledge necessary to understand and reason about a certain abstract concept. In its simplest form, a KS is a group of productions which knows about some abstract concept, and a list of variables such that an instantiation of the variables represents a specific concept which corresponds to this abstract concept. For example, the production in Fig.1 posts an ATTACK concept which corresponds to the ATTACK KS. The ATTACK KS in PARADISE has two variables, COL and SQ, and contains productions which know how to attack a particular square SQ for the side COL. (The ATTACK KS has more structure than this, but it is irrelevant to the present discussion.) With this KS, the system can use the abstract concept of attacking a square in its reasoning, in the expression of plans, and in the communication of discoveries from the tree search.

If one production knows more than one abstract concept, it may be in more than one KS. When a concept in the data base (described below) has a corresponding KS (i.e., they have the same name), PARADISE will eventually treat that concept as a subgoal and use the corresponding KS to solve the subgoal in an attempt to produce a plan to realize that concept. (This may be done by creating other concepts.) There may also be concepts that have no corresponding KS and are not treated as subgoals. Such concepts are inspected by patterns attempting

to match and by searching routines. When a concept corresponds to a KS, it will sometimes be referred to as a goal or subgoal.

Concepts are posted in the data base by the actions in production rules. KSes then use the information in these concepts as subgoals to eventually produce a plan of action. When a new KS is executed, the patterns which matched in previous KSes can no longer be accessed Thus concepts in the data base must contain all the information that will be needed to execute other KSes and to guide the search. Each KS expects a certain amount of information in a concept which it executes. (A concept is 'executed' when it is used as a goal by its corresponding KS.). At the very least, a KS expects an instantiation of its arguments (such as COL and SQ in the ATTACK KS). The ATTACK KS also expects some information on what the intentions of COL are once SQ is attacked. This information is needed so that the reason for wanting to attack SQ will not be destroyed or negated by an attempt to attack. Among other things KSes usually expect information on how likely a goal is to succeed and on how threatening a goal is (this is useful in deciding if a sacrifice is warranted).

PARADISE stores information about concepts on a list of attribute-value pairs (or property list). A particular concept is expressed by giving its name, and instantiation of its arguments, and a list of attribute-value pairs, called an *attribute list.* Frequently, instances of a concept with identical argument instantiations but different attribute lists are posted for different reasons by different productions. Since many productions do not care about the values in the attribute lists, it is imperative to treat all concepts with the same name and instantiation (though different attribute lists) as just one concept. Otherwise, many productions would needlessly be executed repeatedly on different versions of the same goal. Some productions do access the information in the attribute lists, and it is necessary to keep each attribute list distinct for these productions. For these reasons, all concepts in PARADISE with the same instantiation are combined into one concept (which is executed as a goal only once), but the attribute list of this one concept is replaced by a list of attribute lists, one for each reason the concept was suggested.

The productions can access this structure as one concept but can still access the necessary information about each attribute list. When a production accesses the values of attributes, it may not care which list the values come from, or it may require that the values for each attribute come from the same list, or it may want the 'best' (in some sense) value for each attribute whether the different values come from the same list or not, or it may want values for one attribute to come from lists which have some particular value for another attribute. This accessing of the information in a concept is so complex that it can be looked on as a pattern matching in itself. Thus the production in a KS can be looked upon as matching patterns in the given chess position and matching patterns in different concepts.

Concepts and their accessing are quite complex. Intuitively, the productions in PARADISE are not making deductions, per se, but coming up with ideas that may or may not be right. Thus there are no 'facts' that have certain deduced values that can be reasoned with. The complexity of later analysis requires that a production essentially put down 'why' it thought of an idea. In this way, other productions can decide if the idea still looks right in light of what is known at that time. Most production systems avoid this complexity because the firing of a production is a deduction which adds a new piece of knowledge to the system. This piece of knowledge is assumed right and the system is not prepared to later decide that it wasn't right to make that decision.

To summarize, KSes provide abstract concepts that are at a higher level than the productions themselves, and form the language in which PARADISE thinks. Concepts and KSes are used in the static analysis reasoning process, to formulate plans for communicating knowledge down the tree, to quickly access relevant productions during execution of a plan, and to communicate discoveries made during the search.

## 5. Plans

The goal of the static analysis process in PARADISE is to produce plans. Plans seem to play an important role in the problem solving process of move selection for good human chess players. Alexander Kotov in his book *Think Like a Grandmaster* (1971, p.148) writes:

... it is better to follow out a plan consistently even if it isn't the best one than to play without a plan at all. The worst thing is to wander about aimlessly.

Most computer chess programs certainly do not follow Kotov's advice. In a knowledge based program, the cost of processing the knowledge should be offset by a significant smaller branching factor in the tree search. Plans help reduce the cost of processing knowledge by immediately focusing the program's attention on the critical part of a new position created during the search, thus avoiding a complete re-analysis of this new position. Having plans also reduces the branching factor in the search by giving direction to the search. Consider, for example, that most programs may try the same poor move at every alternative depth in the tree, always re-discovering the same refutation. PARADISE can detect when a plan has been tried earlier along a line of play and avoid searching it again if nothing has changed to make it more attractive. Most programs also suffer because they intersperse moves which are irrelevant to each other and work towards different goals. PARADISE avoids this by following a single idea (its plan) down the tree.

Plans in PARADISE can be considered as special instances of concepts since each plan has a list of attribute lists. Instead of a list of instantiations for a set of variables (as in a concept), a plan has an expression in the *Plan-Language.* The Plan-Language expresses plans of action for the side about to move, called the offense. In general, the offense wants to have a reply ready for every defensive alternative. A plan cannot therefore be a linear sequence of goals or moves but must contain conditional branches depending on the opponent's reply. When the offense is on move, a specific move or goal is provided by the plan. When the defense is on move, a list of alternative sub-plans for the offense may be given. Each alternative begins with a template for matching the move made by the defense. Only alternatives whose template matches the move just made by the defense are tried in the search. PARADISE has the following templates which adequately describe defensive moves in terms of their effects on the purpose of the plan being executed (P and SQ are variables which will be instantiated in an actual plan to particular pieces and squares, respectively):

(P SQ) matches when the defense has moved the piece P to the square SQ.
(NIL SQ) matches when the defense has moved any piece to SQ.
(P NIL) matches when the defense has moved P (to any square).
(ANYBUT P) matches when the defense has moved some piece other than P.
nil matches any defensive move.

The plan for an offensive move can be one of two things: a particular move or a goal. A particular move is simply the name of a piece and a square. Such a plan causes PARADISE to immediately make that move with no analysis of the position other than checking that the move is legal. A goal is simply the name of a KS followed by an instantiation for each argument of the KS. When executing a goal, PARADISE will post a concept corresponding to the KS in the data base by using the given instantiations and the attribute lists of this plan as a whole. The KS will then be executed for this concept. Any plan produced by this execution will replace the goal in the original plan and this modified plan will be used. This process expands and elaborates plans. If executing the KS does not produce a plan then the original plan has failed.

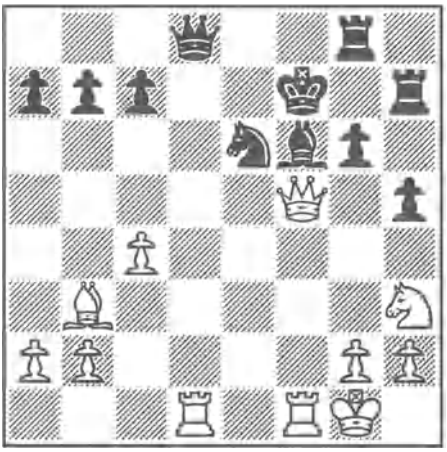


*Fig. 3. White to move*
*A plan produced by PARADISE:*

```
(((WN g5)
 (((BNg5) (SAFEMOVE WR d7)
  (((BK NIL) (SAFECAPTURE WR BR))
   ((ANYBUT BK) (SAFECAPTURE WR BK))))
    ((BNg5) (CHECKMOVE WR d7) (BK NIL)
```

(SAFECAPTURE WR BQ) )))
((THREAT (PLUS (EXCHVAL WN g5) (FORK WR BK BR))) (LIKELY O))
((THREAT (PLUS (EXCHVAL WN g5) (EXCH WR BQ))) (LIKELY O)))

---

Fig. 3 shows a problem and one of the plans PARADISE's static analysis produces for it. ('WN' means white knight, 'g5' means the square g5, etc.) The internal representation of PARADISE has not been printed unambiguously since the term 'WR' does not specify which white rook. However, it should be obvious to which piece the names refer. The last two lines of the plan are attribute lists. The first five lines are the Plan-Language expression for the plan which can be read as follows:

*Play ♘h3-g5. If black captures the knight with his knight, attempt to safely move the rook on d1 to d7. Then, if black moves his king anywhere try to safely capture the black rook on h7, and if black moves any piece other than his king, try to safely capture the king with the rook. A second alternative after black captures the knight is to attempt to safely move the rook on d1 to d7 with check. Then, if black moves his king anywhere try to safely capture the black queen with the rook.*

SAFEMOVE, SAFECAPTURE, and CHECKMOVE are all KSes in PARADISE. After playing ♘h3-g5 and ♘e6×g5 for black in the tree search, PARADISE will execute either the CHECKMOVE goal or the SAFEMOVE goal. Executing the SAFEMOVE goal executes the SAFEMOVE KS which knows about safely moving a piece to a square. This KS will see that ♖d1-d7 is safe and produce this as the continuation of the original plan, causing the system to play d1-d7 with no further analysis. If for some reason d1-d7 was not safe then the SAFEMOVE KS will try to make d7 safe for the rook (by posting a SAFE concept). If some plan is found, the original plan will be 'expanded' by replacing the SAFEMOVE goal with the newly found plan. In general, the newly found plan will contain (SAFEMOVE WR d7) though it may come after a sacrificial decoy or some other tactic. If posting the SAFE concept does not produce a plan, then the SAFEMOVE goal has failed, and the alternative CHECKMOVE goal will be tried. If it also fails, a complete analysis of the position must be done. If black had answered ♘h3-g5 with a king move, it would not match the template (BN g5) and a complete analysis of the position would be undertaken without attempting to make d7 safe for the rook.

By use of conditionals, a plan can handle a number of possible situations which may arise. The most important feature of the Plan-Language is that offensive plans are expressed in terms of the KSes. This has many advantages. Relevant productions are immediately and directly accessed when a new position is reached. The system has one set of abstract concepts it understands (the KSes) and does not need to use a different language for plans and static analysis. The system has been designed to make the writing of productions and thus the forming of KSes reasonable to do. Thus the range of plans that can be expressed is not fixed by the Plan-Language. By forming new KSes, the range of expressible plans may be increased without any new coding (in the search routines or anywhere else) to understand the new plans. This is important since the usefulness of the knowledge base is limited by its ability to communicate what it knows (e.g., through plans).

Given a number of plans to execute, the tree search must make decisions about which plan to search first, when to forsake one plan and try another, when to be satisfied with the results of the search, and other such things. To make such decisions, it must have information about what a plan expects to gain and why. Such information was available to the productions which matched to produce the plan and must be communicated in the attribute lists of the plan so as to provide many different types of access to this knowledge. Unlike most search-based chess programs, PARADISE must use its information about a plan at other nodes in the tree since it executes a plan without analyzing the newly created positions.

To use the information about plans in an effective manner, PARADISE divides a plan's effects into four different categories which are kept separately (i.e., their values are not combined in any way). These four categories are:

THREAT which describes what a plan threatens to actively win,

SAVE which describes counterthreats of the opponent a plan actively prevents,

LOSS which describes counterthreats of the opponent not defended against and

functions the first piece to move in a plan will give up by abandoning its current location (thus providing new threats for the opponent), and

LIKELY which describes the likelihood that a plan will succeed.

These four categories have emerged during the development of PARADISE. Experience seems to indicate the system must have at least these four dimensions along which to evaluate plans in order to adequately guide the tree search.

Since the values for these categories must evaluate correctly at different nodes in the tree, they cannot be integers. Instead they (except for LIKELY) are expressions in the Threat-Language (Wilkins, 1979) which can describe threats in a sophisticated manner. These expressions are evaluated in the context of the current position and may return different values for different positions. The value of LIKELY is an integer. This integer represents the number of unforced moves the defense has before the offense can accomplish its plan. The most likely plans to succeed have a LIKELY of zero (every move is forcing). Both attribute lists in Fig. 3 have a likelihood of zero.

## 6. Creating Plans

A detailed description of the static analysis process in PARADISE is given in (Wilkins, 1979). The concepts and KSes used to produce the plan in Fig. 3 are briefly described here. The system begins by posting a THREAT concept, the THREAT KS having productions which look for threats. Two different productions in THREAT post MOVE concepts for moving the white rook to d7. (One recognizes the skewer of the black king to the rook, and the other recognizes the simultaneous attack on the king and queen where any king move leaves the queen open to capture with check). After executing the THREAT and ATTACK KSes, the system executes the MOVE KS on the 50 MOVE concepts which have been posted by productions. In addition to d7, there are MOVE goals for moving the rook to c7, e7, f8, d6, c6, b6, and a6 as well as many goals for moving other white pieces. A production in the MOVE KS, that recognizes that the proposed move is unsafe, matches for the white rook to d7 move goal. This production posts a SAFE concept for making d7 safe for the white rook. After the MOVE KS is finished, 7 SAFE goals have accumulated. While executing the SAFE KS for the white rook to d7 goal, one production notices that the black knight blocks the white queen's protection of d7. This production posts a DECOY goal in order to decoy the black knight so that the queen can support the white rook on d7. The DECOY KS is executed on the 7 DECOY goals which have been produced, and one production posts a FORCE goal suggesting ♘h3-g5+ as a move to decoy black's knight. It is up to the FORCE KS to decide if ♘h3-g5+ is forcing enough to provoke a move by the black knight. There are 9 FORCE goals in all and the production which finds ♘h3-g5+ forcing posts an INITIAL-PLAN concept which is exactly like the plan shown in Fig. 3.

After executing the CAPTURE KS, the INITIAL-PLAN KS is executed on the 5 INITIAL-PLAN goals which have been produced. (Most of the MOVE goals failed without leading to the posting of a plan). This KS looks for functions given up by a plan or threats not defended against by a plan. Since the plan of Fig. 3 checks the opposing king, a production posts a FINALPLAN concept for this plan. While executing a different goal, the INITIAL-PLAN KS notices that the white queen is under attack and is not defended by the plan in question. This causes execution of the DEFENDOFFENSE KS which produces 6 DEFENDTHREAT goals to save the white queen. The DEFENDTHREAT KS in turn posts 6 INITIAL-PLAN concepts, and executing the INITIAL-PLAN KS a second time produces a total 10 FINALPLAN oncepts for the position in Fig. 3. The attribute lists of these plans rate only two of them as being likely to win material. These two are the one in Fig. 3, and a plan starting with ♖d1-d7 immediately. The plan in Fig.3 is rated highest and is tried first. This static analysis took more than 24 seconds of cpu time on a DEC KL-10. The static analysis process is very expensive, and PARADISE uses its plan to avoid doing such analysis.

Each static analysis is done entirely by the production rules which can easily be modified to increase or make more precise the knowledge available to the system. Each concept (and plan) produced has a REASON attribute listing each production that matched in the process of suggesting the concept. (REASONS have been omitted in Fig. 3.) This provides a good explanation of exactly why PARADISE believes something

and is helpful for quick debugging and easy modification. The attribute lists of each concept contain enough information so that the system essentially knows 'why' it is doing something. Thus the wrong ideas can be discarded readily and combinations of ideas that are antithetical to each other can be avoided. Few ridiculous plans are suggested.

## 7. How Detailed Should Plans be?

The system must quickly recognize when a plan is not working and abandon it before effort is wasted investigating poor lines. On the other hand, it is expensive to completely analyze a position, so the system wants to use its plan as long as possible. To achieve a balance, plans must adequately express the purpose they intend to achieve. Productions must carefully formulate their Plan-Language expressions at the right level of detail using appropriate goals and descriptions of defensive moves. When executing a plan, the system will retrieve a goal or move from the plan to apply to a new position. Care must be taken to ensure that this move or goal is reasonable in the new position. Plans may become unreasonable when they are too general or too specific. In either case, they have failed to express their purpose.

The example of Fig. 3 illustrates the issues involved. Consider the first three ply of the plan as ((WN g5) (BN g5) (SAFEMOVE WR d7)). Suppose this had been expressed as ((WN g5) NIL (WR d7)). Playing ♖d1-d7+ after ♘e6×g5 is reasonable but if black moves his king instead, playing ♖d1-d7+ will lead the search down a blind alley. Thus, (WR d7) is too specific and does not express white's purpose. This problem can be cured by a more specific description of black's reply (e.g., (BN g5) instead of NIL) since the template for black's move will not match if black moves his king. In actual fact, PARADISE does not know for sure that ♖d1-d7+ will be safe after ♘h3-g5+ and ♘e6×g5 although in this particular position it is). To avoid mistakenly playing ♖d1-d7 when it is not safe, PARADISE uses a SAFEMOVE goal to more accurately express the purpose of the plan. For example, the plan could be ((WN g5) NIL (SAFEMOVE WR d7)). If black answers ♘h3-g5+ with ♘e6×g5 then the SAFEMOVE goal is reasonable and quickly produces ♖d1-d7+ as the move to play after verifying its safety. If black answers with a king move then the SAFEMOVE goal is not what white wants to do, but little is lost since ♖d1-d7+ is not safe and the line will be rejected without searching. However, if likely plans had existed for making d7 safe, the search may still have been led astray. For this reason, PARADISE uses (BN g5) as a template in this plan. The purpose of ♘h3-g5+ is to decoy the black knight to g5 and this template most accurately expresses this purpose.

Now let us consider the following more general plan: ((WN g5) NIL (SAFELY-ATTACK BK)). If black answers with ♘e6×g5 then the SAFELY-ATTACK KS should generate ♖d1-d7+ as a safe attack on the black king. If, instead, black moves his king then this KS should generate a check by the white queen or knight which would also be reasonable. Thus this plan produces reasonable moves for every black reply without ever causing a re-analysis of the new position. This would be a good plan if PARADISE knew (from its patterns) that it could get the black king in trouble after any reply. However, it only knows of the skewer of the black king to the rook, the threat of capturing the queen with check, and the fact that the white knight threatens the black king, rook, and knight from g5. It is accidental that the SAFELY-ATTACK goal works after black retreats his king. In the general case, such a goal would produce many worthless checks that would mislead the search. Thus this plan is too general to describe white's actual purpose.

It is very important to get the correct level of detail in a plan. The plan should handle as many replies as possible without causing a re-analysis, but it should avoid suggesting poor moves. The templates for describing defensive moves in the Plan-Language and the various KSes have been developed to allow PARADISE's plans to accurately express their purpose. The results have been quite satisfying: the productions in PARADISE now create plans which rarely suggest poor moves but which can still be used for as many ply as a human might use his original idea.

## 8. Using Plans to Guide the Search

PARADISE's tree searching algorithm is described in detail in (Wilkins, 1979). Only a brief overview is given here in order to

provide background for an example problem solution by the program. PARADISE's tree searching algorithm uses different strategies at the top level to show that one move is best. Once a strategy is selected, a best-first search is done. The value of each move is a range within which PARADISE thinks the 'true' value of the move lies. The program narrows these ranges by doing best-first searches until it can show that one move is best. By using the knowledge base to control the search and by using information discovered during previous searches PARADISE produces enough cutoffs to force convergance of its search without a depth limit or other artificial effort limit.

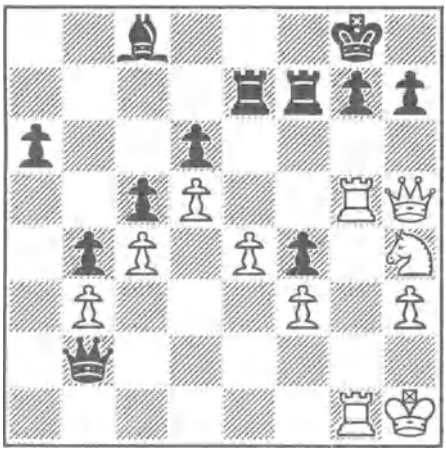


*Fig. 4. White to move.*
*A plan produced by* PARADISE:
(PLY 1)
BESTPLAN:
(((WQ h7) (BK h7) (CHECKMOVE WR h5) (BK NIL) (ATTACKP BK))
((HINT 1) (THREAT (PLUS (WIN BK) (EXCHVAL WQ h7))) (LIKELY 1)))

Fig.4 shows position 49 from (Reinfield, 1958). PARADISE does a static analysis on this position which uses 18 seconds of cpu time and suggests two plans. Fig.4 shows the plan suggested as best. To show how such plans are used to guide the search, PARADISE's search for this position is sketched below.

PARADISE begins by doing a best-first search of the above plan. Execution of this plan commences by playing ♕h5×h7+ and producing a new board position. The defense suggests both legal moves, ♔g8×h7 and ♔g8-f8, but tries ♔g8×h7 first. Since this move matches the (BK h7) template in the plan, PARADISE has ((CHECKMOVE WR h5) (BK NIL) (ATTACK BK)) as its best plan at ply 3 of the search. Since it is in the middle of executing a plan, PARADISE does not look for better alternatives at this point. The best-first search only looks for better alternatives when a new plan is being selected.

Execution of the CHECKMOVE KS produces ((WR h5) (BK NIL) (ATTACK BK)) as a plan which causes ♖g5-h5+ to be played. Black plays his only legal-move at ply 4, ♔h7-g8 which matches the (BK NIL) template in the plan, and leaves (ATTACK BK) as the current plan. Thus the original plan is still guiding the search at ply 5. Executing the ATTACK KS posts many concepts, including MOVE and SAFE concepts, and ((WN g6) ((NIL (SAFEMOVE WR h8)) (NIL (SAFEMOVE WN e7))) is produced as the best plan in this position. This replaces (ATTACKP BK) in the current plan, thus elaborating it. After playing ♘h4-g6, the position in Fig.5 is reached.

*Fig. 5*
(PLY 6)
INITIAL DEFENSIVE MOVES:
((BA a3) (BB d7) ((BB b7) (BR f5) (BR f6) (BR f8))

Productions which know about defending against ♖h5-h8 suggest the 6 black moves in Fig. 5. Black plays ♗c8×h3 first in an effort to save the bishop from the ensuing skewer. At this point the plan branches. Since both branches begin with a null

template, they both match any black move at ply 6. Thus PARADISE has two plans at ply 7: (SAFEMOVE WR h8) and (SAFEMOVE WN e7). Before executing a plan for the offense, PARADISE executes the QUIESCENCE KS which looks for obviously winning moves. (This was done at ply 3 and 5 also, but did not suggest any moves.) Here ♖h5-h8 is suggested by the QUIESCENCE KS which causes the (SAFEMOVE WR H8) plan to be executed immediately. PARADISE plays ♖h5-h8, finds that black is mated, and returns to ply 6 knowing that black's ♗c8×h3 leads to mate. All this has been accomplished without a static analysis; the original plan has guided the search.

At ply 6, black uses KSes to refute the mating line. This involves analysis of the information produced by the previous search. No new moves are found since all reasonable defenses have already been suggested. PARADISE has a causality facility (see Wilkins, 1980) which determines the possible effects a move might have on a line of play. Using information generated during the search of the mating line, the causality facility looks for effects a proposed move might have (such as blocking a square which a sliding piece moved over, vacating an escape square for the king, protecting a piece which was attacked, etc). The causality facility recognizes that neither ♗c8-d7 nor ♗c8-b7 can affect the mating line found for ♗c8×h3 so they are rejected without searching.

Black plays ♖f7-f5 next, the causality facility having recognized that this move opens a flight square for the black king. Again PARADISE has both (SAFEMOVE WR h8) and (SAFEMOVE WN e7) as plans at ply 7. Both SAFEMOVE goals would succeed, but ♖h5-h8 has a higher recommendation and is played first. Black plays his legal move at ply 8, ♔g8-f7.

The original plan no longer provides suggestions at ply 9. PARADISE must now look for better alternatives or do an expensive static analysis to suggest a new plan. Before trying either of these, the program executes the QUIESCENCE KS is an attempt to find an obviously winning move. ♖h8-f8 is suggested and PARADISE immediately plays this move without doing a static analysis. This is mate so PARADISE returns to ply 6 to look for other defenses.

Both ♖f7-f6 and ♖f7-f8 are tried and both are quickly refuted by playing ♖h5-h8+ from the SAFEMOVE goal of the original plan, ♔g8-f7, and ♖h8-f8 from the QUIESCENCE KS. The search then returns to ply 2 and tries ♔g8-f8 in answer to ♕h5×h7+. The template in the original plan does not match ♔g8-f8 so there is no plan at ply 3. However, the QUIESCENCE KS quickly suggests ♕h7-h8 and PARADISE returns from the search convinced that ♕h5×h7+ will mate. This result shows that ♕h5×h7+ is best, so no other best-first searches are initiated.

PARADISE's plans are not always accurate but space prevents presentation of a longer search. The above analysis uses about 130 seconds of cpu time on a DEC KL-10. It goes to a depth of 9 ply while creating only 21 nodes in the tree. Because of the guidance provided by the original plan, no static analysis was performed except on the original position. By comparison, the TECH2 program (an improved version of TECH: Gillogly, 1972) at a depth setting of 6 discovers that it can win material with ♕h5×h7+ although the horizon effect hides the mate from it. For this analysis, TECH2 uses 210 seconds of cpu time on KL-10 and creates 439,459 nodes by making legal moves (as well as making 544,768 illegal moves which are retracted).

## 9. A Typical Medium-Sized Search

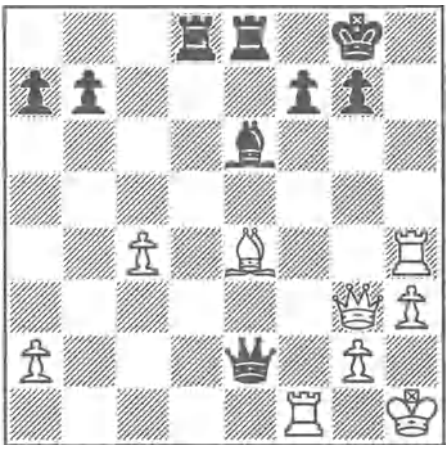


*Fig. 5.1. White to move*

This section presents the actual protocol produced by PARADISE while solving problem 82 in (Reinfeld, 1958). (The program prints squares in algebraic notation to avoid confusion.) This is a typical protocol

produced by PARADISE while solving a problem in which the program finds the best line immediately. Problem 82 in (Reinfeld, 1958) was chosen so that comparisons can be made with the search tree produced by the CAPS program which is given in (Berliner, 1974). Such a comparison is interesting since CAPS has more knowledge about tactics than any other program that plays a full game of chess.

PARADISE's initial static analysis on the above position uses about 20 seconds of cpu time, and produces 7 plans. The one beginning with ((WB h7) has a LIKELY of 0 and is searched first. The other six plans have a LIKELY of 1 and begin with the following moves: (WQ a3) (WR h7) (WR h8) (WQ f2) (WQ f3) (WQ e1). The protocol which follows was printed by PARADISE as it analyzed this problem. Explanatory comments have been added in a smaller font.

In the search, the program refuses to search plans that are not likely to succeed when it has already achieved a significant success. This is done because it is not reasonable to invest large amounts of effort trying ideas which don't seem likely to succeed when a winning line has already been found. The validity of this cutoff rests on PARADISE's accurate calculation of LIKELY values (see Wilkins, 1979). This could produce errors, though not serious ones since the errors would be the selection of a winning move which was not the 'best' winning move. No such errors have been made in any position PARADISE has analyzed.

(TOPLEVEL (VALUE.0))
PLY 1 (EXPECT.320)
BESTPLAN: (WB h7) (BK f8) (WQ a3)
NEWBOARD: (e4 WB h7)

The value of the initial position is 0. (White is trying to achieve a positive score.) The best plan specifies moving the WB to h7 and if black replies by moving his king to f8, the WQ-a3 is played. The reasoning behind the suggestion of this plan is fairly specific. PARADISE knows that after ♗e4-h7+black must move his king either to h8 where white has a discovered check, or to f8 where white can deliver a second check to which black's king cannot reply. The expectation of this plan (after 'EXPECT'), which is calculated from the attribute lists of the plan, is 320 (90 is the value of a queen, so 320 threatens mate). Wherever the word 'NEWBOARD' occurs, PARADISE constructs a new board position by playing a legal move.

(PLY: 2) DEFMOVES: ((BK f8) (BK h8))
NEWBOARD: (g8 BK f8)

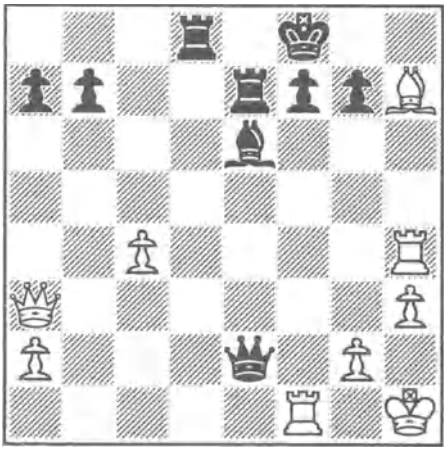
The word 'DEFMOVES' labels the list of moves under consideration by the defensive search. (Each move is the first part of a plan under consideration.)

TRY CURRENT PLAN
PLY 3 (VALUE.0) BESTPLAN: ((WQ a3))
NEWBOARD: (g3 WQ a3)

The phrase 'TRY CURRENT PLAN' means that the program is obtaining the next move by executing a plan that was inherited from an earlier analysis. In this example the current plan specifies an actual move rather than a goal, so the move is made immediately without calculating the primitives in this position. Applying the evaluation function to this position yields 0 (given after the word 'VALUE').

(PLY: 4) DEFMOVES: ((BR e7))
NEWBOARD: (e8 BR e7)

PARADISE knows (BR d6) will not help; CAPS searches it after refuting (BR e7).



NULL PLAN (VALUE . 0) (STATIC-ANALYSIS 20.4 SECONDS) (4 PLANS)
PLY 5 BESTPLAN: (((WB d3) ((NIL (CHECKMOVE WR h8)) (NIL (SAFECAPTURE WB BQ))
(NIL (SAFEMOVE WR h8) (((BK NIL) (SAFECAPTURE WR BR)) ((ANYBUT BK)(SAFECAPTURE WR BK)))))))
(NEWEXPECT . 90)
NEWBOARD: (h7 WB d3)

The current plan is finished so the system does a static analysis which takes 20.4 seconds of cpu time and produces 4 plans. The best one begins with (WB d3) and continues with either a rook check on h8 threatening mate, the capture of the black queen by the bishop, or the rook move to h8 followed by a skewer

of the black king to the black rook. Once again the static analysis accurately recommends the winning plan. This is an improvement over the performance of CAPS which finally suggested B-d3 as a defensive move since it protected the white pawn and white rook which are both en prise. The expectation is now 90, having been recalculated from the new current plan which only expects to win the black queen.

(PLY: 6) DEFMOVES: ((BR d3)(BR d3)
(BQ d3)(BQ f1)(BP g5)(BQ d3)(BK g8))
NEWBOARD: (d8 BR d3)
TRY OBVIOUSLY WINNING MOVE
PLY 7 (VALUE. -33) BESTPLAN: ((WR h8))
(NEWEXPECT . 607)
NEWBOARD: (h4 WR h8)
(EXIT OFFENSE (VALUE 1300.1300))

The system finds ♖h4-h8 as an obviously winning move. It knows the move will mate (although it plays it to make sure), so it doesn't check the current plan. If the system hadn't been sure of the mate, it would have executed the current plan after noticing that ♖h4-h8 duplicates the (CHECKMOVE WR h8) goal in the current plan. A range is returned as the value, but both the top and the bottom of the range are 1300 (the value for mate) since the value has been exactly determined.

(PLY 6) REFUTE: ((WR h8))
(PLY 6) DEFMOVES: ((BQ d3)(BQ f1)(BP g5)(BQ d3)(BK g8))
CAUSALTY: (BQ d3) LINE: (d8 BR d3) NO
NEWBOARD: (e2 BQ f1)

PARADISE backs up to ply 6. A counter-causal analysis tries to refute the move ♖h4-h8 by white but no new moves are suggested. The causality facility compares the proposed ♕e2-d3 move to the tree produced for the ♖d8-d3 move and determines that ♕e2-d3 cannot help. ♕e2-d3 is therefore rejected without searching and ♕e2×f1+ is played (the causality facility approves it). The causality facility makes use of considerable information returned by the searching process. CAPS, which also has a causality facility, searches both ♖d8-d3 and ♕e2-d3 in this position. In fact, more than half the moves rejected by causality in the remainder of this protocol are searched by CAPS.

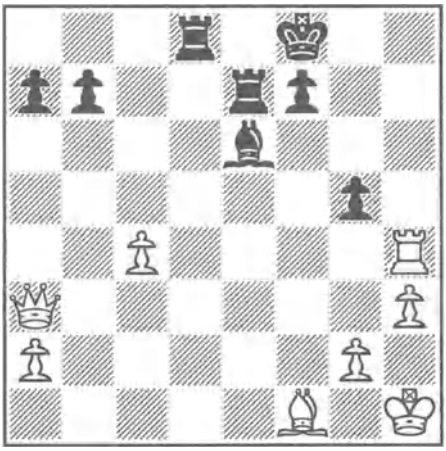
OBVIOUSLY WINNING MOVE DUPLICATED
TRY CURRENT PLAN
PLY 7 (VALUE. -50) BESTPLAN: ((WB f1))
NEWBOARD: (d3 WB f1)

♗d3-f1 is suggested as a winning move, but it duplicates the SAFECAPTURE goal in the current plan (and is not a sure mate) so the current plan is tried.

(PLY: 8) DEFMOVES:((BP d1))
NEWBOARD: (d8 BR d1))
(VALUE . 49) TRY QUIESCENCE SEARCH
(PLY:9) NEWBOARD: (h4 WR h8)
(QUIESCENCE VALUE (1300 . 1300))

The current plan is finished so the offense tries a quiescence to see if the value holds up. A value of 1300 is returned so the search backs up with success for white.

(PLY 8) REFUTE: ((WR h8))
(PLY:8 DEFMOVES : ((BP g5)(BR d2)(BR d7)
(BP g6)(BP f5)(BP f6)(BK e8)(BK g8))
NEWBOARD: (g7 BP g5)



(VALUE. 49) TRY QUIESCENCE SEARCH
(PLY: 9) NEWBOARD: (h4 WR h8)
DEFENDING MOVE SELECTED
(PLY: 10) NEWBOARD: (f8 BK g7)
(PLY: 11) NEWBOARD: (h8 WR d8)
DEFENDING MOVE SELECTED
(PLY: 12) NEWBOARD: (e7 BR d7)
(PLY: 13) NEWBOARD: (d8 WR d7)
(PLY: 14) NEWBOARD: (e6 BB d7)
(PLY: 15) NEWBOARD: (a3 WQ a7)
DEFENDING MOVE SELECTED
(PLY: 16) NEWBOARD: (d7 BB c6)
(QUIESCENCE VALUE (109 . 109))
(EXIT OFFENSE (VALUE 109 . 109))

A quiescence search to a depth of 16 shows that g7-g5 fails for black. The quiescence search knows that white is ahead and that the white rook on h4 is not in danger, yet it plays aggressive moves (at plys 9 through 15) instead of being satisfied. This seems to generate nodes unnecessarily. PARADISE does this because it is a cheap way to get better results. The quiescence search is very inexpensive compared to the regular search. The system thinks it has enough information, yet it may be necessary to search this line again if the

'true' value is not found. Since PARADISE sees an inexpensive way (quiescence searching) to improve the result, it risks generating a few unnecessary (though inexpensive) nodes in order to avoid a possible re-search later.

(PLY 8) REFUTE: ((WR h8))
(PLY: 8)DEFMOVES: ((BR d2)(BR d7)(BP g6)(BP f5)(BP f6)(BK e8)(BK g8))
CAUSALITY: (BR d2) LINE: (d8 BR d1) NO
CAUSALITY: (BR d7) LINE: (d8 BR d1) NO
CAUSALITY: (BP g6) LINE: (g7 BP g5) NO
NEWBOARD: (f7 BP f5)

Black tries other moves at ply 8. Causality rejects ♖d8-d2 and ♖d8-d7 on the basis of the tree generated for ♖d8-d1, and it rejects g7-g6 using the g7-g5 tree.

(VALUE . 49) TRY QUIESCENCE SEARCH
(PLY: 9) NEWBOARD: (h4 WR h8)
DEFENDING MOVE SELECTED
(PLY: 10) NEWBOARD: (e6 BB g8)

(PLY: 11) NEWBOARD: (a3 WQ a7)
(QUIESCENCE VALUE (59 . 59))
(EXIT OFFENSE (VALUE 59 . 90))

Again the quiescence search confirms the offensive success. This time the value of 59 does not meet the expectation so the offensive search returns a range of 59 to 90 since the expectation of 90 may have been acheived if a static analysis had been done. PARADISE notes in the tree that searching here again may improve the value from 59 to 90.

(PLY 8) REFUTE: ((WR h8))
(PLY: 8) DEFMOVES: ((BP f6)(BK e8)(BK g8))
CAUSALITY: (BP f6) LINE: (f7 BP f5) NO
NEWBOARD: (f8 BK e8)

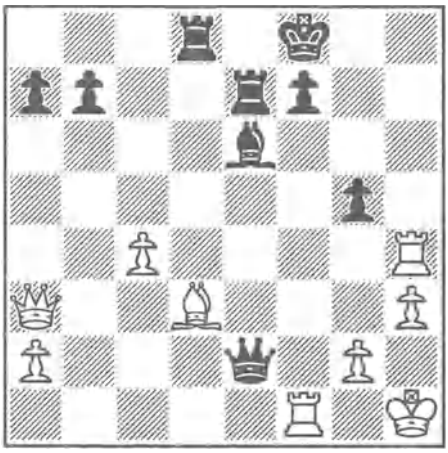
(VALUE . 49) TRY QUIESCENCE SEARCH
(PLY: 9) NEWBOARD: (a3 WQ a7)
(QUIESCENCE VALUE (59 . 59))
(EXIT OFFENSE (VALUE 59 . 90))

(PLY 8) REFUTE: ((WQ a7))
(PLY: 8) DEFMOVES: ((BK g8)(BR a8)(BP a6)(BP a5)(BP b6))
CAUSALITY: (BK g8) LINE: (f8 BK e8) NO
CAUSALITY: (BR a8) LINE: (d8 BR d1) NO
CAUSALITY: (BP a6) LINE: (d8 BR d1) NO
CAUSALITY: (BP a5) LINE: (d8 BR d1) NO
CAUSALITY: (BP b6) LINE: (f8 BK e8) NO
DEFMOVES: ()

(PLY 7) BESTPLAN: ((CHECKMOVE WR h8))
TERMINATE: PLAN SUCCEEDED
(EXIT OFFENSE (VALUE 59 . 90))

The offensive search terminates since a significant gain has been achieved. PARADISE again notes in the tree that searching other plans here may improve the value. The range of 59-90 is returned.

(PLY 6) REFUTE: ((WR h8)(WR h8)(WR h8)(WB f1))
(PLY: 6) DEFMOVES: ((BP g5)(BK g8))
NEWBOARD: (g7 BP g5)



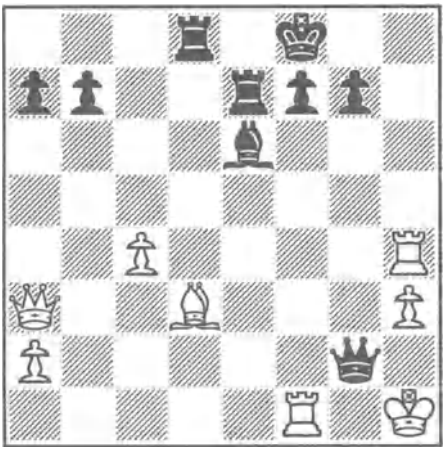
OBVIOUSLY WINNING MOVE DUPLICATED
TRY CURRENT PLAN
PLY 7 (VALUE . 0) BESTPLAN: ((WB e2))
NEWBOARD: (d3 WB e2)

(PLY: 8) DEFMOVES: (BP h4))
NEWBOARD: (g5 BP h4)

(VALUE . 49) TRY QUIESCENCE SEARCH
(PLY: 9) NEWBOARD: (a3 WQ a7)
(QUIESCENCE VALUE (59 . 59))
(EXIT OFFENSE (VALUE 59 . 90))

(PLY: 8) DEFMOVES: ()
PLY 7 BESTPLAN: ((CHECKMOVE WR h8))
TERMINATE: ALPHA BETA

(PLY 6) REFUTE: ((WB e2))
(PLY: 6) DEFMOVES: ((BQ e3)(BQ d2)(BQ e5)(BQ g2)(BQ a2)(BK g8))
CAUSALITY: (BQ e3) LINE: (d8 BR d3) NO
CAUSALITY: (BQ d2) LINE: (d8 BR d3) NO
CAUSALITY: (BQ e5) LINE: (d8 BR d3) NO
NEWBOARD: (e2 BQ g2)



TRY OBVIOUSLY WINNING MOVE
PLY 7 (VALUE .-10) BESTPLAN: ((WK g2))
NEWBOARD: (h1 WK g2)
(PLY: 8) DEFMOVES: ()

The defensive search now does a null move analysis (see Wilkins, 1979). It has no idea what to do so it lets the offense make two moves in a row. However, the offense calls the quiescence search which knows it is the defense's move. It decides the defense can escape from the threat of ♖h4-h8 so it calls the position quiescent.

(VALUE .89) TRY QUIESCENCE SEARCH
assumed escape from (WR h8)
(QUIESCENCE VALUE (89 . 89))
(EXIT OFFENSE (VALUE 89 . 89))

(PLY: 8) DEFMOVES: ()
(PLY: 6) REFUTE: ((WK g2))
(PLY: 6) DEFMOVES: ((BQ a2)(BK g8))
CAUSALITY: (BQ a2) LINE: (d8 BR d3) NO
CAUSALITY: (BQ g8) LINE: (g7 BP g5) NO
(PLY: 6) DEFMOVES: ()

PLY 5 BESTPLAN: ((WQ a5) NIL (CHECKMOVE WQ d8))
no unlikelys after success: quit

(EXIT OFFENSE (VALUE 59 . 89))

The search backs up to ply 5 and tries the next plan suggested by the static analysis. This plan has a LIKELY of 1 and the result already achieved is so successful that PARADISE rejects the unlikely plan without searching.

(PLY 4) REFUTE: ((WR h8)(WB d3))
(PLY: 4) DEFMOVES: ()
PLY 3 LAST PLAN
(EXIT OFFENSE (VALUE 59 . 320))

The search backs up to ply 3. There are no more plans to execute here, but a static analysis has not yet been done so the program makes a note of this. The top of the value range is changed to 320 to reflect the possibility of achieving the original expectation by doing a static analysis to find plans.

(PLY 2) REFUTE: ((WQ a3))
(PLY: 2) DEFMOVES: ((BK h8))
NEWBOARD: (g8 BK h8)

NULL PLAN (VALUE . 0) (STATIC-ANALYSIS 14.2 SECONDS) (2 PLANS)
PLY 3 BESTPLAN: ((WB d3))
(NEWEXPECT .320)
NEWBOARD: (h7 WB d3)

The defense tries ♔g7-h8 at ply 2 and the original plan does not match this move so a static analysis is undertaken. Again PARADISE finds the right idea immediately. To compare, CAPS did not have the necessary knowledge and tried 3 other moves first, generating 80 nodes to disprove them, before stumbling onto ♗h7-d3+

(PLY: 4) DEFMOVES: ((BK g8))
NEWBOARD: (h8 BK g8)

(VALUE . 0) TRY OBVIOUSLY WINNING MOVE

PLY 5 BESTPLAN: ((WB e2))
(NEWEXPECT . 90)
NEWBOARD: (d3 WB e2)

(PLY: 6) DEFMOVES: ()
(VALUE .99) TRY QUIESCENCE SEARCH
(QUIESCENCE VALUE (99 . 99))
(EXIT OFFENSE (VALUE 99 . 99))

(PLY:6) DEFMOVES: () PLY 5 LAST PLAN
(EXIT OFFENSE (VALUE 99 . 320))
(PLY 4) REFUTE: ((WB e2)) (PLY: 4)
DEFMOVES: ()

PLY 3 BESTPLAN: ((WB e4))
(NEWEXPECT . 10)
TERMINATE: FORWARD PRUNE
(EXIT OFFENSE (VALUE 99 . 320))

The search backs up to ply 3 and the offense tries ♗h7-e4+ which has been suggested by the static analysis. Its expectation is 10 and ♗h7-d3+ has already achieved 99 so a forward prune occurs.

(PLY 2) REFUTE: ((WB d3))
(PLY: 2) DEFMOVES: ()

PLY 1 BESTPLAN: ((WQ a3) NIL (CHECKMOVE WB h7))
no unlikelys after success: quit

The search returns to the top level to try the next plan but it is terminated because it has a LIKELY of 1 and the search is over. CAPS invests much effort at this point searching other offensive moves.

BEST MOVE: (e4 WB h7) VALUE: (59 . 320)
PRINCIPAL VARIATION:
1 (e4 WB h7)(g8 BK f8)
2 (g3 WQ a3)(e8 BR e7)
3 (h7 WB d3)(e2 BQ f1)
4 (d3 WB f1)(f7 BP f5)
5 (h4 WR h8)(e6 BB g8)
6 (a3 WQ a7)

TOTAL TIME: 297 SECONDS
NODES CREATED: 36 (22 REGULAR, 14 QUIESCENCE)
STATIC ANALYSES: 2

WHERE TIME WAS SPENT:
49% calculating primitives
33% quiescence searching (overlaps primitive calculations for 14 nodes)
11% static analysis
9% defense determining initial move
3% disk IO
0% causality facility, evaluation function, creating board positions

Many things should be noticed in this example. The plans do an excellent job of guiding the search: only two static analyses are done in the entire search. The plans are so accurately specified by the analysis that they never once lead the search off the correct line. White's play in the search is error-free. The causality facility makes good use of information returned from the search. Many black moves which would otherwise have been searched are eliminated in this manner. The range values accurately express the potential of each node so that the system does not have to waste effort determining the 'true' values. The best first search strategy plays a minor role in this example.

On this problem, CAPS generated a tree of 489 nodes in 115 seconds to obtain a principal variation of ♗d3-h7+, ♔g8-f8, ♕g3-a3+, ♖e8-e7, ♗h7-d3, ♕e2×f1+, ♗d3×f1, ♖d8-d1. This is slightly inaccurate since there is a mate in one for white at the end of this variation which CAPS's quiescence analysis did not recognize, but it is clear that CAPS understands the problem and its solution. PARADISE generates only 36 nodes to CAPS's 489 for the following three reasons (primarily):

— PARADISE has more knowledge available during static analysis and can accurately analyze a position and produce good plans. CAPS generates many nodes by not playing the correct offensive move first on some occasions.

— PARADISE returns more useful information from its search and can therefore use its causality facility to eliminate moves that CAPS searches.

— PARADISE has a best-first strategy while CAPS is committed to finding the 'true' value (within alpha-beta) of each node it searches. This enables PARADISE to terminate as soon as some information is discovered without having to look for better alternatives.

This comparison shows the advances PARADISE has made in the use of knowledge. It is not meant to belittle CAPS, which pioneered some of the techniques basic to this approach. In fact, CAPS is the only program with which a comparison is appropriate. For example, on this problem CHESS 4.4 (running on a CDC-6400) produces a tree with 30,246 nodes in 95 seconds of cpu time without finding the solution (which is too deep for it). Looking at the details of such a tree would not be helpful.

## 10. Measuring PARADISE's performance

To aid in evaluating performance, PARADISE

was tested on positions in the book *Win At Chess (Reinfeld, 1958)*. This book contains tactically sharp positions from master games which are representative of tactical problems of reasonable difficulty. PARADISE's knowledge base was developed by writing productions which would enable the program to solve, in a reasonable manner, fifteen chosen positions from among the first 100 in *Win At Chess*. The 85 positions not chosen were not considered during this development. This development process produced one version of the program, called PARADISE-0. Six positions were then picked at random from the remaining 85 and accurate records were kept on the work involved in getting the program to solve these reasonably. The version of the program which solves all 21 positions is called PARADISE.

PARADISE's performance on a problem is classified into one of three categories:
(1) problem solved as is (possibly with a minor bug fix),
(2) problem not solvable without a change to the program or a major change to the knowledge base, or
(3) problem solvable with a small addition to the knowledge base.

Category 3 helps measure the modifiability of the knowledge base. It is meant to include solutions which require no changes whatsoever to the program and only a small addition to the knowledge base. Small means that it takes less than 20 minutes total of human time to identify the problem and write or modify one production which will enable PARADISE, with no other changes, to solve the problem in a reasonable way (i.e., no ad hoc solutions).

Of the six positions chosen at random, two were in category 1, three were in category 3, and one was in category 2. The latter one inspired the only program changes between PARADISE-0 and PARADISE. These results speak well for the modifiability of the knowledge base, but shed little light on the generality of the program. To better test generality, PARADISE has been tested on the first 100 positions. These positions are divided into 5 groups of 20 and Reinfeld claims an increase in difficulty with increase in group number. (Eight end-game positions were eliminated, so the groups actually have 18, 19, 18, 20, and 17 positions.) PARADISE is considered to solve only the problems in category 1, while PARADISE-2 solves both category 1 and 3 problems. PARADISE would become PARADISE-2 simply by leaving in the productions written to solve problems in category 3.

Since the 21 developmental problems had not been solved by one version of the program, these were tried first. Adding new knowledge during program development would (hopefully) not adversely affect performance on earlier problems, but this had to be confirmed. For example, productions added to the knowledge base to solve the last 5 developmental problems might produce so many suggestions in the first 5 problems that the search would become untractable in those problems. The testing proved that the new knowledge had no adverse effects. All 21 problems fell into category 1: thus the same version of the program solves them all. In every case, the analysis is either the same as or sharper than that produced by developmental versions of the program. This result provides strong evidence that the knowledge base is easily modifiable. If productions are written carefully and intelligently (a skill the author developed while writing productions for the developmental set), they do not appear to adversely affect the system's performance on positions unrelated to the new productions. This is an essential quality for a modifiable knowledge base.

Table 1 shows what percentage of these problems can be solved by PARADISE, PARADISE-2, CAPS (Berliner, 1974), TECH

*Table 1. Percentage of problems solved by various chess players*

| | PARADISE | PARADISE-2 | CAPS | TECH | CHESS 4.4 | CLASS A HUMAN |
|---|---|---|---|---|---|---|
| Group | I 78% | 100% | 67% | 78% | 94% | 89% |
| Group | II 68% | 95% | 74% | 84% | 95% | 95% |
| Group | III 94% | 100% | 61% | 61% | 78% | 94% |
| Group | IV 70% | 95% | 50% | 40% | 70% | 80% |
| Group | V 65% | 94% | 41% | 47% | 76% | 53% |
| All | 92 75% | 97% | 59% | 61% | 83% | 83% |

(Gillogly, 1972), CHESS 4.4 (Slate et al., 1977) running on a CDC 6400, and a human player rated as class A (Berliner, 1974). PARADISE was limited to forty-five minutes of cpu time per problem while the other programs were limited to five minutes.

PARADISE already exhibits more generality in this domain than programs like TECH and CAPS. PARADISE-2 outperforms all the programs and the human. This shows that these problems do not push the limits of the expressibility of the production language nor the ability of the program to control the tree search. PARADISE does well on Group III because seven of those twenty problems are in the twenty-one problems on which PARADISE was developed.

There are twenty problems solved by PARADISE-2 but not by PARADISE, but only thirteen productions were written to solve them. In two instances, an already existing production was modified. In five instances the same production solved two different category three problems. This is a strong indication that the productions being written are fairly general and not tailored to the specific problem. These results indicate that the generality of PARADISE is reasonable and that the modifiability of the knowledge base is excellent. There are three problems not solved by PARADISE-2 which are discussed in the next section.

PARADISE uses more cpu time and produces trees with considerably fewer nodes than other programs. The example search in Section 8 is a problem from Group III, and tree sizes of 21 nodes for PARADISE and 439, 459 for TECH-2 are fairly typical. Table 2 compares the average tree size (in number of nodes) for PARADISE-2, CAPS and CHESS 4.4 on the problems they solved in these 100.

*Table 2. Comparison of average tree size on first 100 problems (for solved problems)*

| | PARADISE 2 | CAPS | CHESS 4.4 |
|---|---|---|---|
| Group I | 19.8 | 167.8 | 24,907.3 |
| Group II | 28.7 | 226.4 | 34,726.0 |
| Group III | 35.0 | 206.1 | 24,200.1 |
| Group IV | 58.4 | 453.6 | 31,538.1 |
| Group V | 48.4 | 285.3 | 25,917.5 |
| All 100 | 38.1 | 260.6 | 28,496.8 |

PARADISE attempts to use knowledge whenever possible in order to produce trees of the same order of magnitude as those produced by human masters. Table 2 shows that this has been accomplished for the most part. CAPS uses a lot of knowledge but invests only about one-fifth of a second per node calculating. CAPS relies on the search to correct mistakes made by inadequate knowledge and this results in trees one order of magnitude larger than those produced by PARADISE. CHESS 4.4 has little chess knowledge and relies almost entirely on search to discriminate between moves. It generates trees that are three orders of magnitude larger than those generated by PARADISE. The two knowledge-orientated programs (CAPS and PARADISE) grow larger trees for the deeper combinations (Groups IV and V), just as most humans would. CHESS 4.4's tree size seems unrelated to the depth of the combination. CHESS 4.4 is of course the best chess player of these three programs.

Table 3 shows where PARADISE invests its resources. The statistics are compiled over the 89 problems which PARADISE-2 solved. They give some idea of the size of tree PARADISE grows, the amount of time it spends, and where this time is spent. (CPU time is in seconds on a KL-10 processor.)

Table 3 shows that calculating primitives

*Table 3. Resource investment in PARADISE (89 solved problems)*

| | Mean | Highest | Lowest | Standard Deviation |
|---|---|---|---|---|
| CPU time for whole program | 332.9 | 1958 | 19 | 396.6 |
| Nodes created during search | 38.06 | 215 | 3 | 41.9 |
| % CPU time calculating primitives: | 52.85 | 74.3 | 34.1 | 8.47 |
| CPU time per static analysis: | 12.22 | 26.5 | 2.2 | 5.14 |
| Number of static analyses per problem: | 3.73 | 35 | 0 | 5.96 |
| % CPU time doing static analysis: | 11.61 | 33.7 | 0.0 | 10.04 |
| % nodes requiring static analysis: | 8.95 | 33.3 | 0.0 | 8.07 |
| % CPU time executing knowledge base: | 91.6 | 97.7 | 81.5 | 2.92 |

is PARADISE's most computationally significant activity. The fact that less than 9% of the nodes generated had static analyses done on them shows that the plans do a good job of guiding the search. Because of this, static analyses are not a major overall expense even though an average one takes more than twelve seconds of cpu time. The different cpu time limit given to PARADISE in Table 1 is due to the availability of statistics, but is not as unfair as it seems. As Table 3 shows, PARADISE-2 uses an average of only 5.5 minutes of cpu time on the 89 problems it solves. Also, PARADISE could probably be speeded up by a factor of 2 or more with more efficient production matching. As Table 3 shows, the program spends more than 91% of its time in the inefficient production interpreter.

## 11. How PARADISE Goes Wrong

PARADISE-2 was not able to solve 3 of the 92 problems after small additions to the knowledge base. A failure by PARADISE can be classified in one of the three following categories:

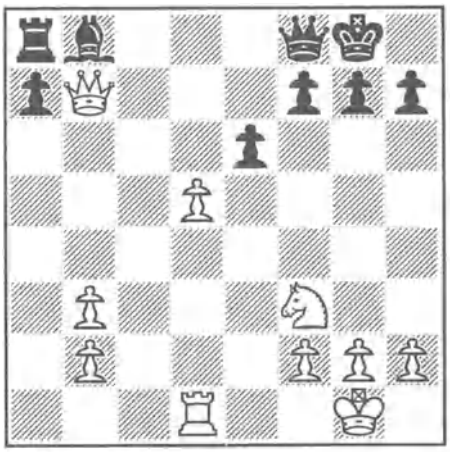
(1) The best plan is never suggested. PARADISE tries every plan that has been suggested and then quits without having achieved an acceptable result since it has run out of ideas. This can be cured by filling a hole in the program's knowledge so that the best plan will be suggested.

(2) The search becomes unbounded. If a position is rich in possible attacks and defenses, with most combinations running many ply before a quiescent position is reached, then PARADISE may use an unreasonable amount of time searching. During testing, PARADISE was limited to 45 minutes of cpu time per problem.

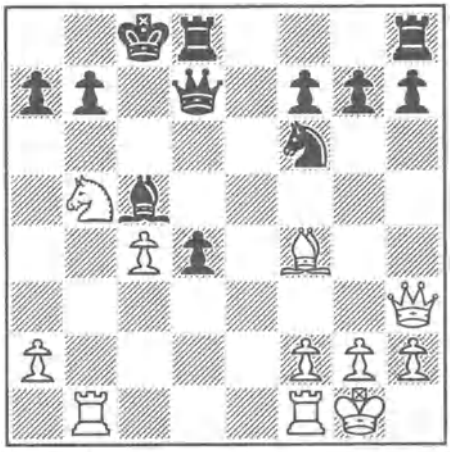
(3) A mistake is made in the analysis. For example, PARADISE may not solve a problem (even though recommending the best move) because the causality facility has eliminated a good defensive move which should have been searched, or because the quiescence analysis has made a large error (small errors must be tolerated).

Of the twenty problems solved by PARADISE-2 but not by PARADISE, 19 fell into category 1 initially. One fell into category 3 because of a mistaken quiescence analysis, but an added production fixed the problem. There were no failures caused by the search becoming unbounded. The 3 problems

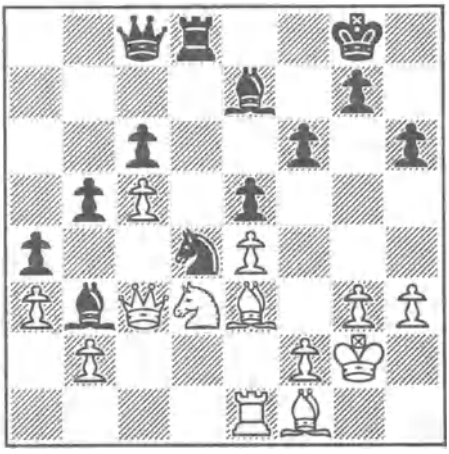
*Figure 6. Problems not solved by PARADISE-2*



*problem 31 white moves*



*problem 71 white moves*



*problem 91 black moves*

which PARADISE-2 could not solve are shown in Fig. 6, and are examined in detail below.

In problem 31, white plays d5-d6 and wins the black bishop which must move to save the black rook. If ♕b7×a8 is played immediately, black replies ♗b8×h2+ and wins white's queen. (White still has a won position after this, but it is an involved pawn ending which goes well beyond the depth to which any current program can reasonably search.) PARADISE-2 could easily solve this problem by including a production which tries to block a checking move by a piece blocking a support of an en prise piece. Such a production could be expressed in only a few lines in the production-language. However, this is unacceptable because it is an ad hoc solution. The 'correct' way to find d5-d6 is by analyzing the refutation of ♕b7×a8. This move is suggested by PARADISE's counter-causal analysis but is not searched because offensive counter-causal moves are tried only if they are captures (see Wilkins, 1979). Changing this restriction would involve a program change, so PARADISE-2 cannot solve this problem. This problem would be classified as a category 3 failure.

Problem 71 is a category 1 failure. The initial static analysis does not suggest ♘b5×a7+ which wins for white by enabling the white queen to move to a3 and then up the a-file. The static analysis has the goal of moving the white queen to a3 and then up the queen rook file, so a fairly simple production in the MOVE KS suggests ♘b5×a7+ at the top level. With this new production PARADISE-2 at first got the correct answer, but a bug in one of the productions in the DEFENDMOVE KS prevents some good defensive moves from being tried at ply 4. When this defensive production is fixed, the search becomes unbounded (i.e., uses more than 45 minutes of cpu time) because of the many defensive and offensive possibilities. This is the only example of a category 2 failure during testing of the program.

Position 91 is also a category 1 failure. Black can win a pawn and a much superior position by playing ♗b3-e6. Black threatens mate with his queen and knight and white must play ♗e3×d4 to avoid it, allowing e5×d4 2 ♕-any ♗e6×h3+ winning a pawn. Whenever PARADISE suggests mating sequences, it first expends a large amount of effort doing a sensitive analysis of the upcoming possibilities. Problem 91 must be understood on a similar level of specificity, since PARADISE uses this specific analysis to control the search. However, such a specific production would require more than thirty minutes of human effort to design and implement so PARADISE-2 was not given credit for this problem.

## 12. Comparison to Plans in Robot Problem Solving

Plans are used in many domains of AI research, especially robot problem solving. ABSTRIPS (Sacerdoti, 1974), NOAH (Sacerdoti, 1975), and BUILD (Fahlman, 1974) are examples of planning systems in robot problem solving environments. Plans in these systems are very different from those in chess so a general description of these differences is more appropriate than a detailed comparison. Robot planning is frequently done in abstracted spaces which have been defined. These abstracted spaces omit details such as whether a door is open or not. There are a small number of such details, and the problem solver has some specific knowledge for each such detail which it can apply when required. For example, the problem solver has a procedure for opening a door should the door be closed when the robot wants to go through it. Such an approach is much more difficult in chess, at least with mankind's current understanding of the game. It is not clear how to define abstracted spaces. Small details are very important in chess and cannot be readily ignored, even for determining the first step of a plan. For example, a chess planner should not ignore the fact that a piece is pinned on the assumption that it can be unpinned when the plan calls for it to move. Thus plans in PARADISE are all at the same, very detailed level.

In the robot planning system, the effects of an action are well defined. The program can easily and quickly determine the exact state of the world after it has moved a blue block. In chess, moves (actions) may subtly affect everything on the board. For example, a piece that was safe may no longer be (even though the move made has no direct effect on the piece), and the system may need to make an expensive calculation to determine this. Chess plans cannot make many assumptions about the state of the world in the future, but must describe the

expected features of new states in the plan itself and then test for these features while executing the plan. Consider the monkey and bananas problem. If the experimenter is sinister, he could connect the bananas to the box by pulleys and rope, invisible to the monkey, so that when the box is pulled under the bananas, the bananas are pulled up so they cannot be reached from atop the box. In a sense, the robot planners assume that no such crazy side effects will happen, while a chess plan must prepare for such things. This manifests itself in PARADISE in at least two ways. First, plans are not composed of simple actions but are goals which may require the knowledge in many productions to be interpreted. This postponement of evaluation permits checking of many complex features in the current world state. Second, PARADISE has been designed to make it easy to produce new concepts (by writing productions and forming KSs) for expressing plans whenever they are needed to handle new 'crazy side effects'.

In most robot planning systems, tests for having achieved the goal or the preconditions of an action are trivial and give well defined answers. In chess, there may be only very subtle differences between a position where a particular action is good and positions where the same action is wrong. It is hard to know when a goal has been acheived, since there is always a chance of obtaining a larger advantage (except when the opponent has been check-mated). Thus in chess it is necessary for the plan to provide a considerable amount of information to help in the making of these decisions. This is done in PARADISE through the attribute lists in plans.

The number of things PARADISE can consider doing at any point in a plan is about an order of magnitude larger than the number of things most robot problem solvers usually contend with. There are an average of about 38 legal moves in a chess position (see De Groot, 1965). A robot usually has a much smaller number of possible actions which cannot easily be eliminated (for example, going to one of a few rooms or picking up one of a few objects). When planning farther in the future than just the first action, the chess planner has many more choices than the 38 legal moves. The inability to make assumptions about future states often prevents mentioning actual moves in a plan. Instead a description of the intent is needed, and the number of such descriptions is much larger than the number of legal moves that might be made. A robot usually has a much smaller number of possible actions, so the two types of plans explode at very different rates.

To summarize, the robot planners have the flavor of establishing a sequence from a small number of well-understood operations until the correct order is found, while chess planners have more a flavor of 'creating' the correct plan from the many possibilities. For this reason, a system which produces good chess plans needs a large amount of knowledge and non-trivial reasoning processes to produce plans.

Probably the most important idea PARADISE has for the robot problem solvers is that a plan may be viewed as a way to control what parts of a large knowledge base will be used to analyze each situation during plan execution. Plans in PARADISE can be viewed as telling the system what knowledge to use in analysis, thus defining a perspective for the system to use in new situations. Some amount of analysis from this new perspective is almost always done before the plan can continue executing. In most robot planning systems plans are more accurately viewed as telling the system what action to take. PARADISE tries to delay execution in order to bring more knowledge to bear.

## 13. Comparison to Plans in Chess

Pitrat's program (Pitrat, 1977), which solves chess combinations, is the most important example of the use of plans in chess. The language Pitrat uses for expressing plans has four statements with the move statement and modification statement being the most important. The move statement specifies that the piece on one particular square should move to another particular square.It may also be specified that the move must be a capture. The modification statement describes a modification to be made to a particular square. This can be one of four things: removal of a friend, removal of an enemy, moving a friend to the square, or getting an enemy to move to the square.

Plans in PARADISE provide much more flexibility in expression than Pitrat's move statement. In PARADISE, types of moves

other than captures can be specified (e.g. safe moves and safe capture moves). Also, particular squares do not need to be named in PARADISE since it can use a variety of goals to express the plan. The important square to move to may change depending on the opponent's move so it is not always possible to specify such a square in advance. For example, if white makes a move which traps and attacks black's queen then (after some desperado move by black) white would like to capture black's queen wherever it may be. PARADISE can express the plan of capturing the queen anywhere, while Pitrat's move statement cannot since the planner does not know which particular square the queen will be on.

Pitrat's modification statement is a goal which the system tries to accomplish. These modifications are too simple to express the purposes behind their suggestion. For example, the system may want to decoy the black queen to make a square that it protects safe for white. Pitrat would express this goal as removing an enemy from the black queen's location which does not express the purpose of the plan. It will work well for the winning combination but will also allow many wrong moves since the queen may be decoyed to a square from which she can still protect the square in question. PARADISE can avoid this since it has the ability to express its purposes. It can specify that the black queen must be decoyed to make the particular square safe, and only decoys which remove the black queen's protection will be considered. PARADISE would also specify that the move after the decoy should be safe so it will not be tried unless the preparations have accomplished their purpose. PARADISE attempts to express the purpose of each plan while Pitrat's plans express side effects that will happen if the purpose is accomplished (e.g., a square becoming vacant). Unfortunately, the same side effects may also happen when the purpose is not accomplished, causing the system to waste effort searching poor lines.

Plans in PARADISE have two more major advantages over the plans in Pitrat's system. First, PARADISE has conditionals which allow specification of different plans for different replies by the opponent. The advantages of this are obvious: the system can immediately try the correct plan instead of searching an inappropriate plan and backtracking to correct itself. Second, PARADISE's plan language is modifiable. Simply by writing new productions, new goals and concepts can be created for expressing plans and the system will automatically understand these new plans. This property is necessary for any system that wishes to extend its domain or incrementally increase its expertise. Significant additions to Pitrat's plan language would seem to require a major programming effort.

Despite the shortcomings of the plans in Pitrat's program, they are much more sophisticated than any plans previously used in computer chess programs. Pitrat's program performs well in its domain. It processes nodes much faster than PARADISE and therefore can handle larger trees. The plans do an adequate job given these contraints. It may be the case that programs will obtain better performance using Pitrat's approach of larger trees and less sophisticated plans, but the use of knowledge is only starting to be investigated by programs such as PARADISE and it is too early to draw conclusions.

## 14. Summary

PARADISE exhibits expert performance on positions it has the knowledge to understand, showing that a knowledge-based approach can solve some problems that the best search based programs cannot solve because the solutions are too deep. The knowledge base is organized into KSes which provide concepts for PARADISE to use in its reasoning processes. These concepts are higher level than the ideas produced by simply matching patterns. A single pattern cannot recognize a whole plan of action in a complex chess position; both pattern-level ideas and these higher concepts are necessary in the reasoning process.

The concepts provided by the KSes are used to create plans during static analysis. The concepts communicate enough information that they can be rejected after being posted. This keeps antithetical ideas from being combined to produce ridiculous plans. The same concepts are used to express plans for guiding the search, and the KSes execute the plans during the search. By communicating plans down the tree, PARADISE can understand new positions on the basis of its analysis of previous positions. By having particular goals in

mind when looking at a new position, the system quickly focuses on the relevant part of the knowledge base. Plans in PARADISE express their purpose fairly well, without being too general or too specific. They contain enough additional information to enable the system to decide when a line has succeeded or failed. Because of the successful communication of knowledge through plans, PARADISE is able to solve a number of chess combinations with small trees and without a depth limit on its search.

PARADISE's knowledge base is amenable to modification. Production rules (and therefore KSes) can be quickly written and inserted in the knowledge base to improve system performance without adverse affects. Since KSes are used in both planning and static analysis, modifications can improve both without program changes. The range of expressible plans can easily be increased.

## 15. Issues

One of the major issues in creating concepts is their generality. How 'high-level' should a concept be? If the concepts are too general then too many details are lost, and the system does not have the necessary facts to do certain reasoning. If not general enough, the system leans towards the extreme of needing a production for every possible chess position. The concepts in PARADISE have been constructed to be as general as possible without sacrificing expert-level performance to the loss of detail. The factor limiting generality is the ability to communicate details in the attribute lists of concepts. The generality in PARADISE has made concepts in the data base so complex that production must essentially match patterns in these concepts.

The tradeoff of generality and specificity in concepts is closely related to the tradeoff of search and knowledge. With much specific knowledge the use of knowledge becomes expensive, but fewer mistakes need to be corrected in the search. Such a knowledge-based approach can solve deep problems (e.g., PARADISE has no depth limit), but may not be able to solve many problems, even easy ones, because of holes in its knowledge. With more general knowledge, analysis is cheaper but more time must be spent correcting mistakes in the search. Such a search-based approach often leads to a program which does a full-width search. Such programs almost always solve problems within their depth limit, but cannot solve deep problems and suffer from problems like the horizon effect (see Berliner, 1974).

PARADISE uses a mix of search and knowledge which involves more knowledge and less search than previous programs which include chess middle games in their domain. The strength of the program depends on the completeness of the knowledge base. Programs like CAPS and Pitrat's do not search all legal moves and may also have holes in their knowledge. One might expect PARADISE to miss combinations more often than these two programs since its more specific knowledge suggests fewer moves, but this has not been the case in the comparison of CAPS and PARADISE on the test positions. PARADISE recognizes mating attacks that both CAPS and Pitrat's program miss. PARADISE's knowledge base appears complete enough to outperform CAPS on the first 100 positions in Reinfield (1958), even on positions within CAPS' depth limit. PARADISE-2's knowledge base is complete enough to outperform CHESS 4.4 on these problems.

The primary justification for the cpu time used by PARADISE and the biggest advantage of the approach used in PARADISE is the extendability of the knowledge base. This means the effect of holes in the knowledge can easily be reduced in PARADISE. The play of most programs cannot be noticeably improved with easily made program or knowledge modifications. Improved play would involve programming a more sophisticated analysis, or controlling the search better so that it could search deeper. (Play can, of course, be improved by switching to a significantly faster computer in most cases.) It would be hard to incrementally improve performance of a program by increasing its depth limit since each increment of one ply multiples the effort required by the branching factor (usually about 5 of 6 for programs with full width searches that have a reasonable move ordering for alpha-beta). The possibility of incremental improvement through improved analysis is discussed below.

Programming a more sophisticated static analysis in any of the programs mentioned

in this paper (except PARADISE) would probably not be easy for the programmer. Even if it were, these programs are committed (by the tree) to limit the amount of processing per node. CAPS spends more time per node than the others, but still processes five nodes per second of cpu time. CHESS 4.7 processes around 3600 nodes per second on a Cyber 176. If CHESS 4.7 spent even an additional ten milliseconds per node in analysis, the size of the tree it could search in a given amount of time would be greatly reduced. Easy incremental improvement of CHESS 4.7 (and similar search-based programs) by making analysis progressivly more sophisticated does not seem possible. This means the theoretical limitations imposed by the depth limit cannot easily be overcome.

The time constraints on CAPS and Pitrat's program would permit slight increases in the time spent on analysis. Such slight increases cannot involve very specific knowledge, so any knowledge which recognizes a new tactic would suggest more moves to be searched in many positions. This increases the size of the search tree exponentially. These programs are already running efficiently and would either have to develop new search control mechanisms, or use significantly more cpu time to cope with this increase in branching factor.

For PARADISE's knowledge base to achieve the completeness needed to rival the best search-based programs (in middle tactics), many productions may yet have to be added. There is no evidence that suggests large numbers of productions will harm PARADISE's performance (although problems may arise). New productions do not seem to significantly increase the branching factor when the added productions are at the right level of specificity (see Wilkins, 1979).

New productions do not significantly increase the analysis effort largely because plans guide PARADISE's search so well that a static analysis is done at less than nine percent of the nodes generated. Thus most new productions will rarely be executed. Even when executed, productions usually do not match and the system can often determine this with little effort. Even when matched, well-written productions take only a few milliseconds to a few hundred milliseconds to match. Compared to the 12 seconds of cpu time PARADISE spends on a typical analysis, this is not significant. The limiting factor in PARADISE's incremental improvement appears to be the ability to recognize more complex tactics at the correct level of specificity.

Weighing the above tradeoffs in order to judge which approach is best depends on the current technology. With developments such as faster machines, parrallelism, or better hashing techniques, programs which rely on search can search deeper and perform better. The development of better pattern recognizers might increase the power of PARADISE's production language allowing PARADISE to perform better. There is little doubt that with current machines and our current understanding of how to use knowledge, good search-based programs solve a larger class of problems with less expense than do the best knowledge-based programs. However, techniques for using knowledge are only beginning to be developed and understood. This research shows ways to effectively use a large knowledge base which can be easily extended. As progress is made in representing and using knowledge, a knowledge-based program may eventually be able to approach (and even surpass) the performance of human chess masters.

**Acknowledgements**

The author is indebted to Hans Berliner for his continuing assistance and counsel during this research, and to the Stanford Artificial Intelligence Laboratory which provided the necessary environment and tools for doing this research.

# 6.3 Mate at a Glance

**John Birmingham and Peter Kent**

First published in *Advances in Computer Chess 2* (M. R. B. Clarke, editor), Edinburgh University Press, Edinburgh, 1980, pp. 122-130. Reprinted by permission of Edinburgh University Press, and of the authors and AERE Harwell, Oxfordshire, where their work was carried out.

**Abstract**

Current tournament level chess programs use several different systems to detect checkmate within the game tree.

This paper discusses two popular systems and then describes a routine incorporated into the MASTER program (Birmingham et al., 1980) which detects mate and certain forced mate situations ('pseudo' mate) at the static level.

A set of rules is given, by which mate or 'pseudo' mate positions may be recognised given a small amount of knowledge.

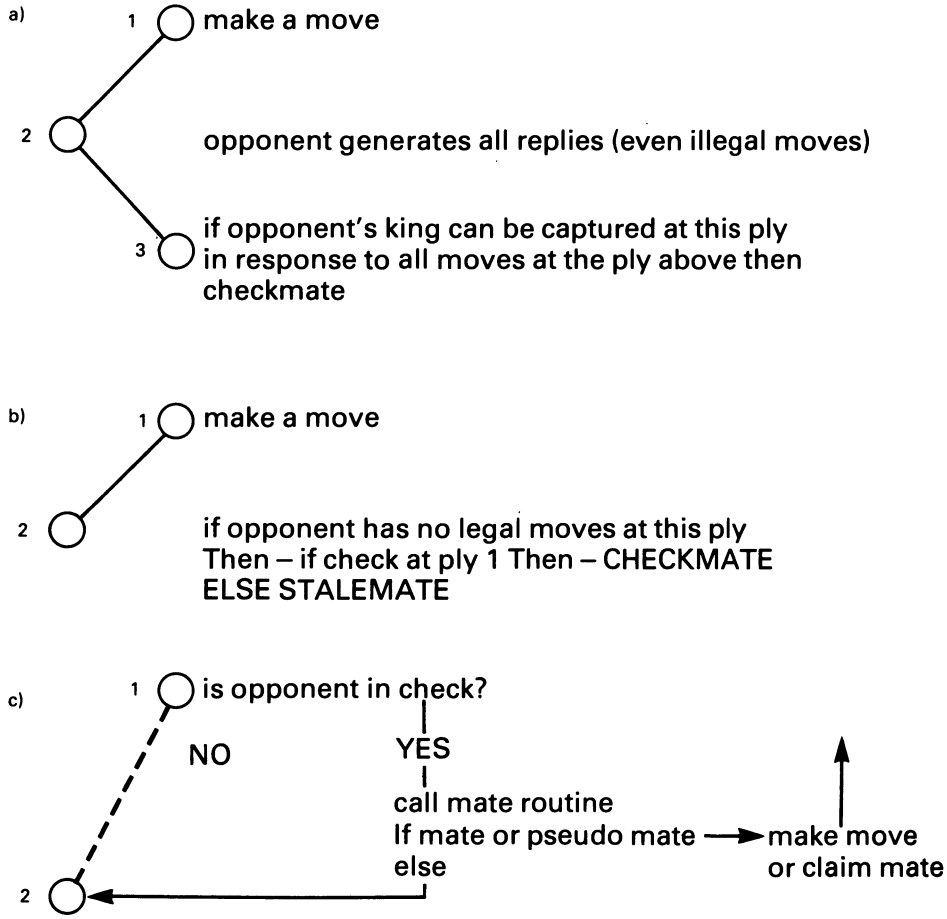



*Fig. 1*

## Introduction

A chess player's 'one shot' look at the board is, in fact, rarely as simple an exercise as the recognition of a pattern. For many mate positions, the player has already played a sequence of moves (tree searched) which have forced the winning move. However, when suddenly confronted by a position in which there is a one-move mate, or a forced mate in *n*, even the expert player can miss the obvious and/or spend a considerable time 'tree searching' the board. If this were not true, the chess problemist would have a lean time! There are many (famous) examples in the chess literature of missed mates of this sort.

The modern chess program is rarely now diverted by such distractions as a queen *en prise* when there is a checkmate in the offing – provided, always, that the checkmate is: (a) within its lookahead; and (b) if within its lookahead, the program is capable of recognising the fact. There are several methods used to detect checkmate in game-trees, which may be summarised as shown in figure 1.

Both methods 1a and 1b need at least one extra ply from a mating move to verify that the position is in fact mate, and in game situations where time is short and a limited depth of search has to be employed (5 ply). Programs that use this method often cannot recognise mate in 2. As will be shown, method 1c can recognise a mate move at ply 1, a mate in 5 from ply 1, and has found mate in 9 using a 7-ply tree (18-ply mate!). The current MASTER program (level 4.6) uses a combination of methods 1b and 1c. Method 1b is used where method 1c would either prove too expensive or would involve third and fourth order effects.

## Knowledge Requirements for Static Mate/Pseudo Mate Detection

To implement 'mate at a glance' routines in a chess program, certain elements of information are essential, and some assumptions must be made with respect to the attacking/defensive powers of certain pieces, some obvious and some more subtle. They may be summarised as follows: (a) an explicit knowledge of the control which each piece has over any given square of the board; (b) a knowledge, for any given square, of the attacking piece, and its colour; and (c) 'X-ray' attacks, 'absolute' pins and implied pins.

In the MASTER program, a set of arrays is maintained by the Evaluation routine(s), which store the following information:

MC The number of controls which 'my' pieces have on any given Board square.

YC The number of controls which 'your' pieces have on any given board square.

MP The home squares of each of my pieces which control any given square.

YP The home squares of each of your pieces which control any given square.

MX Blocked vector information for my ♗,♖,♕ (X-rays)

YX Blocked vector information for your ♗,♖,♕ (X-rays)

MBIAS my King controls a list of squares adjacent to the Kings

YBIAS your King controls a list of squares adjacent to the Kings

From the above, the program can – by simple table lookup – detect:

a) which pieces attack/defend any given square

b) which pieces are pinned by what and to what

c) the degree of control, for either side, of squares around the Kings

d) which pieces prevent my or your pieces from attacking or defending significant squares of the board. E.g. if 'I' move 'your' piece, I could mate in a given square.

In order to make the above tables more useful, certain deductions have to be made about the attacking powers of certain pieces in special configurations.

(a) *Pawns*: can only attack/defend diagonally but block in a forward direction.

(b) *Bishops and Queens*: control extra squares when looking through pawns of their own colour, in the correct direction of travel.

From figure 2 it can be seen that the pawn on c3 puts 1 control on each of b4 and d4. The bishop on a1 controls squares b2, c3 and d4. Because in a swap-off sequence on d4, both pawn and bishop can take part; both b4 and d4 are doubly controlled by white, although this is not immediately obvious.

(c) *Vector Pieces* (♖,♗,♕): When doubled (or tripled) in ♖♖, ♖♕, ♗♕ combinations add extra controls to 'X-rayed' squares.

From figure 3, we can see that the queen on a1 adds controls on the 'a' file to the

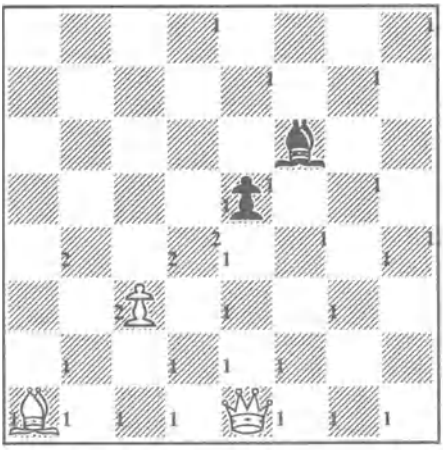


*Figure 2*

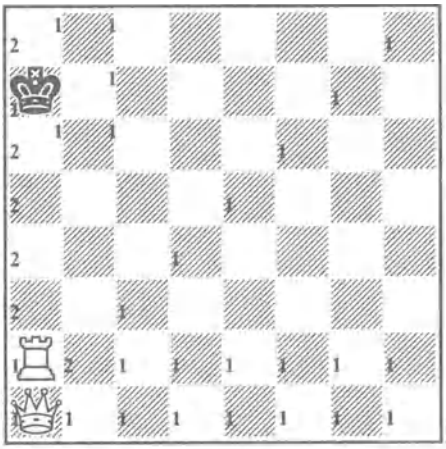


*Figure 4*

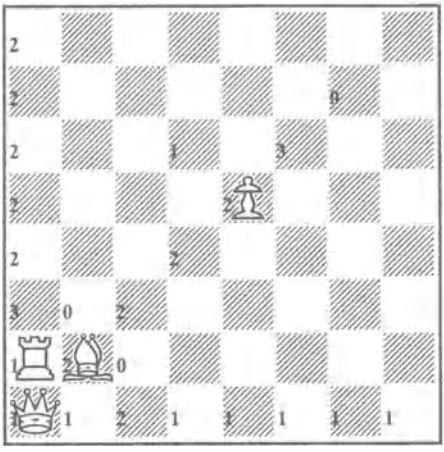


*Figure 3*

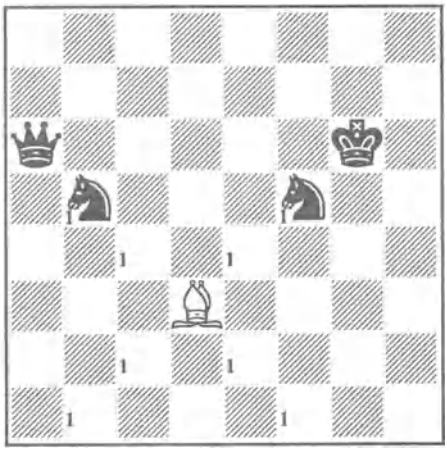


*Figure 5*

squares controlled by the rook, and to the a1-h8 diagonal squares controlled by the bishop. N.B. that the square f6 is triply controlled by White due to the pawn case.

The *important exception* to the above cases is that the square containing the opponent's king *cannot* be controlled in this way. In figure 4 the square of the black king (a7) has only one control on it (that of the rook). If the extra control of the queen were considered effective then the position would be flagged as a 'double check', which is obviously incorrect. Note however, the square behind the king (a8) is controlled by both rook and queen.

(d) *Implied controls (X-rays) by Vector Pieces (Pins)*: In figure 5, White's bishop on d3 has an 'X-ray' attack on the black queen on a6 and on the black king on g6. Thus both of Black's knights are pinned.

For 'mate at a glance' purposes, the two types of pin must be treated differently, since the pin on the knight on b5 is not absolute in that it can legally move; but the pin on the knight on f5 is absolute, since to move it would place the king in check.

(e) An often overlooked point: pieces do not control their home squares.

## Detection of Mates and 'Pseudo' Mates

From the above information, the following set of rules can be derived.

(It should be noted at this point we will give only the rules that have been implemented in MASTER, but will show the

extensions of these rules that will be implemented in the future, and the reason for their exclusion at this stage.) The rules may be used to detect: (a) checkmate, and (b) 'pseudo' mate, a position in which the opponent can only postpone mate by a series of useless blocks.

*Given* the enemy king is in check (one of *my* pieces has a control on the square of the enemy king)

*then* if all the squares adjacent to the king are blocked by his pieces

*or* controlled by my pieces

*or* blocked by both my and his pieces and my pieces are defended (controlled by others of my pieces)

*then* mate can be detected for the following cases:

1. For a checking piece, adjacent to the king, attacked only by the king (figure 6).
2. For a double check (figure 7).
3. For a knight which is not attacked (figure 8).

[*Note.* The logical extension of this case, e.g. a knight which is attacked by one of your pieces but which is defended by my other knight, has not been implemented, because the following complications prove too expensive at present to compute, in that they require a high degree of recursion. E.g. the piece which attacks the first knight may, on capturing, reveal a secondary check on my king, or pin on my second knight against my king, etc.]

4. For pieces adjacent to the king which are attacked only by pinned pieces – and the king (figure 9).
5. For vector pieces (rook, queen, bishop) on the rank, file or diagonal, subject to the following tests:

5a. Pieces which threaten to block on the square adjacent to the king must be pinned to the king (figure 10). The bishop on g7, if not pinned would successfully block on f8.

5b. Pieces which threaten to block do not create an escape square which 'I' do not control (figure 11). This is an example of a 'pseudo' mate. The block by the knight is useless since it is on a square which is not controlled by one of its own pieces and consequently can be taken.

5c. That you cannot block with a pawn on a square which is controlled by another of your pieces (figure 12: see also 5d). The pawn on d3 can successfully block the check since the square d2 is controlled by the queen. Note: blocks by the ♕ on d2

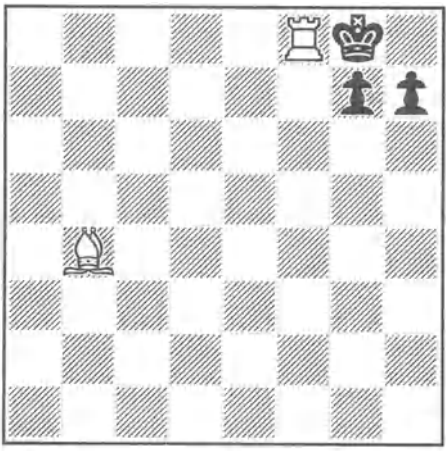


*Figure 6*

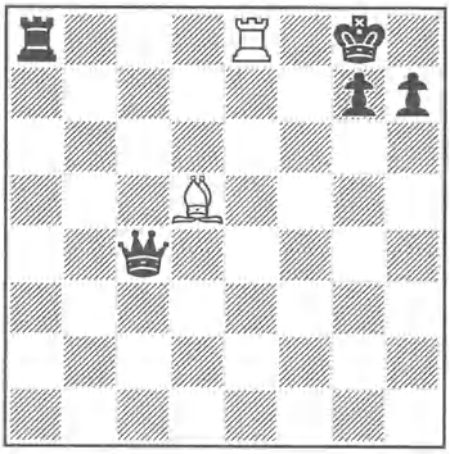


*Figure 7*

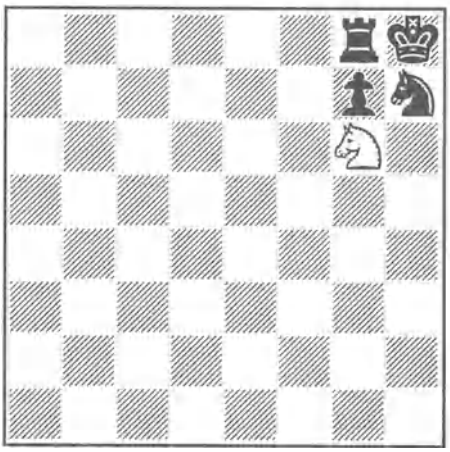


*Figure 8*

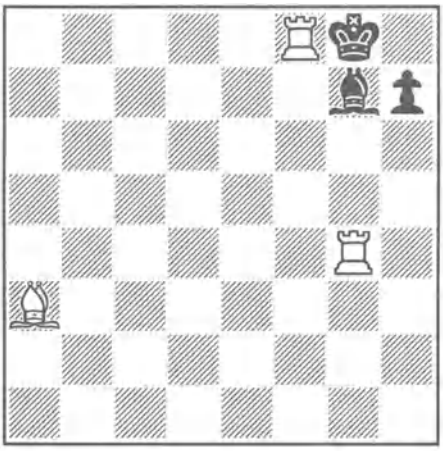


*Figure 9*

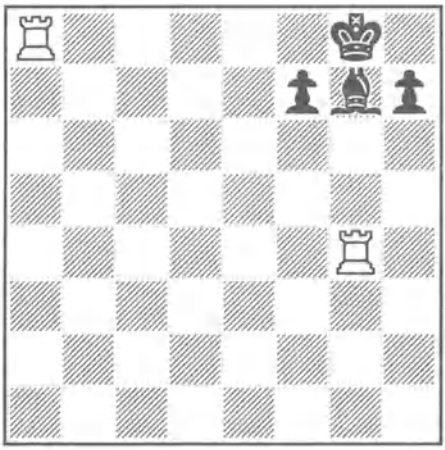


*Figure 10*

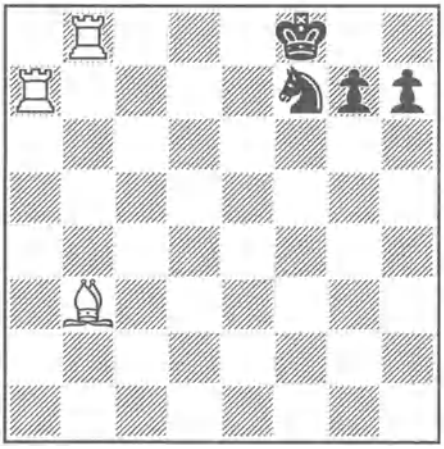


*Figure 11*

*Figure 12*

or the bishop on f2 would both be unsuccessful, e.g. pseudo mates would be claimed.

5d. That your blocking piece does not discover check on my king (figure 13). Although moving to an unprotected square, the pawn reveals a check on my king. The bishop block on f2 gives 'pseudo' mate.

5e. That blocking pieces do not reveal secondary blocks (figure 14).

The knight block on b8 reveals a secondary block by the rook. (♘d7-f8 or ♕h4-d8 would both be pseudo mate.)

[Note: This is the current implementation. The logical extensions of this rule involve recursive computing which is not economical.]

e.g. 5.e.1. That blocking pieces do not reveal secondary blocks either adjacent to the king, or

5.e.2. on squares which are controlled by other pieces (figure 14).

5.e.3. That blocking pieces do not reveal secondary blocks that in turn discover check on my king.

5.e.4. etc., etc.

5f. If you have *more than one control on a blocking square* and either piece is a knight or rook (figure 15; see also 5g).

5g. If you have more than one control on a blocking square, do bishop or queen control the square through a pawn? (figure 16). The position as shown is a 'pseudo' mate position. The pawn block is useless. However the square c1 is of interest since it

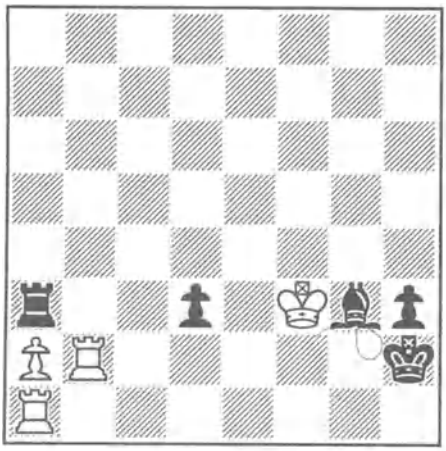


*Figure 13*

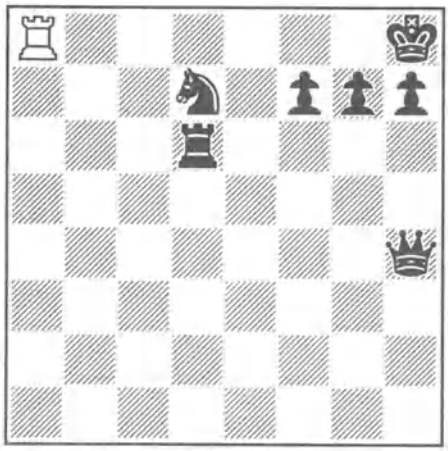


*Figure 14*

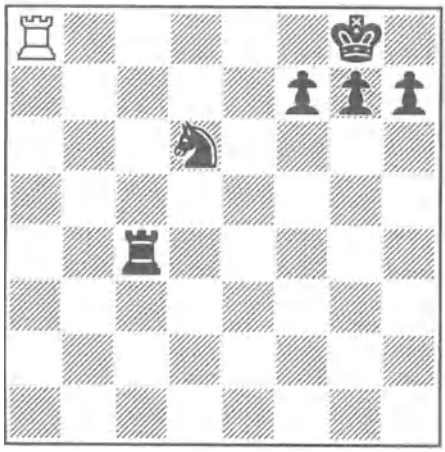


*Figure 15*

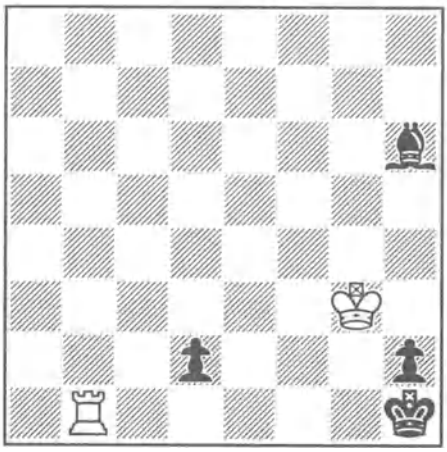


*Figure 16*

is doubly controlled, by pawn and bishop. This case must be detected as an exception to the general rule that a doubly controlled square on the vector implies rule 5f.

5h. If you can block with a pawn, will the blocking square then be controlled by rook or queen? (figure 17). This is an exception to the rule that no squares on the vector are controlled by the opponent in that the apparently useless block by the pawn creates a control (by the rook).

*then* if you have no block moves on the checking vector *then checkmate else* 'pseudo' mate. (Useless blocks defer mate.)

*Note.* For the current implementation of the routine, as soon as second, third, fourth, etc, order effects, other than those given are detected, then the program returns a flag instructing the tree searcher to follow the sequence. This technique is used to save time in what would otherwise be a very expensive routine.

## Some Examples

1. The position in figure 18, when tested, was claimed at ply 1 by the program after it had evaluated only two nodes (it looked at ♘f6×d7 first), and .001 seconds of elapsed time. Prior to the inclusion of the routine, the program needed 2 ply (3 ply by the figure 1a example) to find the mate.

2. The position in figure 19 was equivalent to a mate in 11 ply for MASTER (mate in

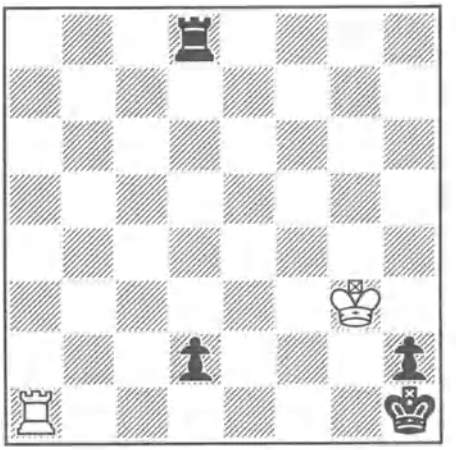


*Figure 17*

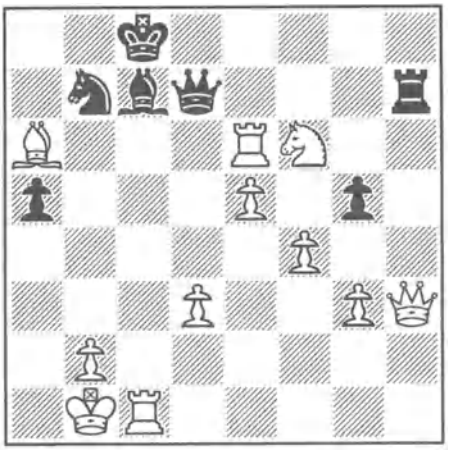


*Figure 18*

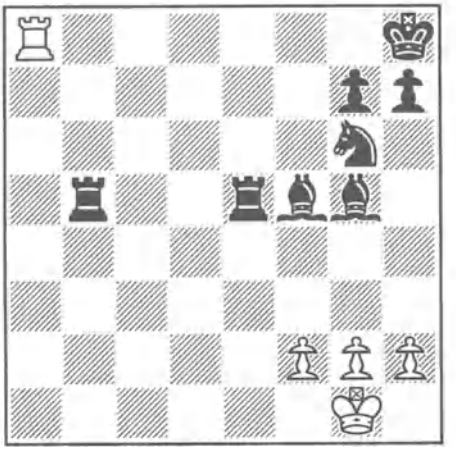


*Figure 19*

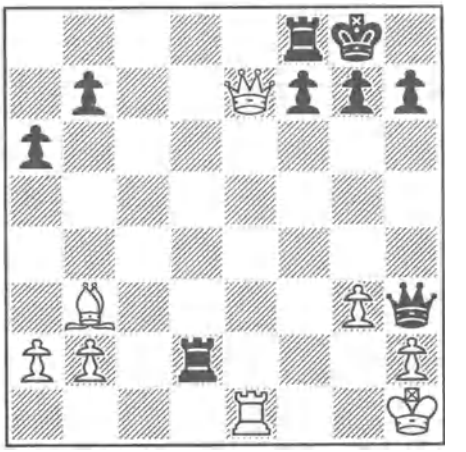


*Figure 20*

12 or 13 ply for conventional methods). It was detected at ply 1 and was the first node examined. If detected at ply 7 of the tournament program, it would enable the program to claim 'mate in 17 ply', e.g. mate in 9, which is outside the range of most conventional programs (if not all).

## Spin Off

### *1. Application to Tree Pruning*

The main routine, which is now a permanent feature of the MASTER program, is called at all plies of the tree when a check is detected. Apart from its immediate value in detecting certain classes of mate and 'forced mates', it has proved surprisingly effective in pruning the game tree. Branches can be cut ruthlessly when mate position is detected, generally 1-2 plies earlier than with conventional methods.

When coupled with the feedover technique, which ensures that profitable lines/moves are examined early in the tree, it can considerably reduce the number of nodes examined at each ply. Consider the position shown in figure 20. The following table shows the nodes/ply from a 5-ply run in which the position ocurred. Figures are given for the program with (a) no mate routine; (b) mate routine giving scores but not cutting the tree; and (c) mate routine used to cut the tree.

| Ply | (a) | (b) | (c) |
|---|---|---|---|
| 1 | 42 | 42 | 42 |
| 2 | 1809 | 1303 | 530 |
| 3 | 525 | 231 | 184 |
| 4 | 2861 | 1761 | 793 |
| 5 | 1982 | 262 | 262 |
| | 7219 | 3599 | 1811 |

A gain of approximately 74% was made in the tree at a cost of approximately 2% in the evaluation routine.

*2. Detection of 'Interesting' Squares*

For example, in figure 21 the rook could deliver mate on square a8, if the bishop could be captured or moved.

The routine has been modified to 'flag' any move which attacks pieces which control a square in which mate would otherwise be possible. In the example, moves ♘c4-a5, ♘c4-d6, (♖a1-a7, ♖a1-a8) would be flagged, and the tree search would then be instructed to examine in depth the effects of the swap-offs.

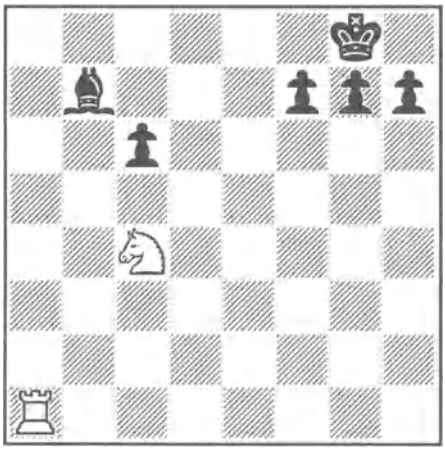


*Figure 21*

The overall effect of the modification (which is currently being researched) is that the program now shows a degree of planning in its play, which has had a marked effect in some aspects of its game.

# 7 Special Purpose Software and Hardware

Chess masters, to a certain extent, talk in their own language. This language has a vocabulary rich in content, which if understood by a computer could make the task of programming chess somewhat easier. Mike Clarke has suggested that a special problem orientated language be designed to facilitate chess programming, and so far as I know his work is unique in the field. His paper (section 7.1), ought to inspire further work on the subject, and the design of such a language would be an excellent topic for a Ph.D. thesis.

Most of the research that has investigated special purpose chess systems has dwelt on hardware. The earliest chess machine, apart from the various "automata" that were operated by humans hidden inside them, was an electromechanical device designed by a Spanish engineer named Torres y Quevedo. The machine plays the ending of king and rook against king, winning in less than 50 moves against any defence. Torres' method represents, to my mind, the very first working "Expert System". A series of rules determine what should be done at each stage, depending on certain conditions, in much the same way as today's rule-based expert systems.

Two rather more modern approaches to special purpose chess hardware are also discussed in this chapter. The CHEss Oriented Processing System (CHEOPS) was designed and programmed at M.I.T. by a team that included Richard Greenblatt, whose work a decade earlier is described in section 2.4. CHEOPS was certainly an improvement on the Greenblatt program of 1966-7, but unfortunately it has never competed in computer chess tournaments so we are unable to compare its strength with the various generations of the Northwestern program and with its other rivals.

The work of Ken Thompson and Joe Condon at Bell Telephone Laboratories (section 7.4) is, in contrast, well tested. BELLE took over the mantle from the Northwestern program by winning the 1980 World Championship in Linz, Austria, and until 1983 its phenomenal speed (more than 200,000 position evaluations per second!!) gave it sufficient power to dominate the computer chess world. In October 1983, during the next World Championship competition, the program's authors were presented with a $5,000 prize from the Fredkin Foundation, for producing the first chess program to gain the official title of National Master. BELLE had become a U.S. Master by achieving a rating of 2200 or more (2203 to be precise), but ironically lost its World Championship title to CRAY BLITZ during that very same 1983 tournament.

# 7.1 Some Ideas for a Chess Compiler

**Mike Clarke**

First presented at the NATO symposium on "Human Thinking – Computer Techniques for its Evaluation" at St. Maximin, 1971. Reprinted by permission of the author.

**Abstract**

A brief survey of chess programming is given, and some points of difficulty identified. It is suggested that chess programming requires a problem oriented language, and an illustration is given of such a language based on the syntax of Algol.

**Introduction**

Programs to play chess have been written and discussed almost as long as computers have been in existence, and a critical summary of what has been achieved in these twenty years has been given recently by Levy (1971). The two strongest programs, those of Greenblatt et al. (1967) and Slate and Atkin (1969) play at about the level of the 500th best player in Britain. They are both tours de force of machine code programming designed specifically to play chess, and work on the by now familiar principles initiated by Shannon (1950). In the position to be analysed a few moves, usually between four and seven, are selected on the basis of some rather ad hoc coding of a few chess heuristics. The tree of possible variations is grown, pruned at a fixed depth or when the position becomes quiet in some sense, and a numerical evaluation function computed for the terminal positions, usually as a weighted sum of more chess heuristics. These values are then backed up the tree by some efficient minimaxing procedure, such as the alpha-beta algorithm, to select the variation for which the highest possible terminal value can be forced. This procedure reached its ultimate development with Samuel's (1967) draughts (checkers) program which was also a learning program in the sense that the program itself rather than the writer of the program varied the weights in the evaluation function in the light of its experience.

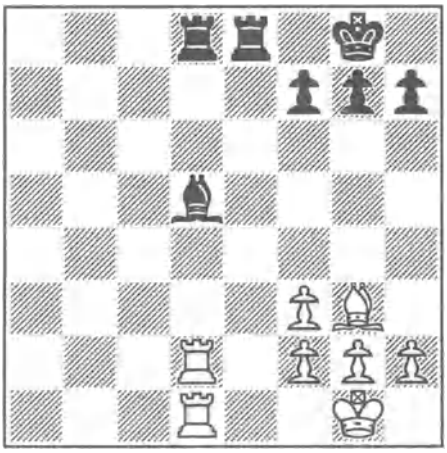
Samuel's program plays draughts at nearly world championship level. Similar programs for simpler games, such as kalah, (Slagle and Dixon, 1969), are probably unbeatable by human players. One of the difficulties of using simple games that are not often played by people is that it is sometimes hard to see exactly what has been achieved. If chess was hardly ever played the two programs mentioned above would probably be considered unbeatable. As it is, there is an extensive chess culture and their performance can be seen to fall short of good human play in many respects.

**Comparison of Machine and Human Play**

In an interesting series of experiments De Groot (1965) measured the ability of both experts and non-experts to reproduce chess positions after being shown them for only a few seconds. He found that success in doing this was highly correlated with chess ability, indicating that the experts had developed a particularly efficient mental representation of the position. Unfortunately his results in that paper were purely quantitative and I have been unable to trace a subsequent paper. It would have been interesting to know what kind of relationships between the pieces were reproduced correctly by the experts in order to give some clue as to what representation of a chess position they used. De Groot also found that good players think that they consider on average less than two moves as at all playable in analysing a position during play. In other words there are on average less than two branches from each node in the tree of variations to be analysed, compared with between four and seven for

the chess programs. This, of course, enables the good human player to look at least twice as far ahead for the same amount of work.

A second, related, reason for the comparative weakness of chess programs is their lack of ability to make use of and collate the results of "local" analyses in order to deduce new moves worth analysing. Chess programs usually start with a wide selection and reject by brute force; people seem to start with a small selection of moves and add to it on the basis of what they find. As an illustration consider the following simple chess position.



The human player as White might begin by thinking 1. ♖d2×d5 won a piece. On analysis he would see that 1. ♖d2×d5 ♖d8×d5 2. ♖d1×d5 ♖e8-e1 is mate. This would prompt him to try 1. h2-h3 to give the king some room, a move that without some analysis appears motiveless as ♖e8-e1 is not an immediate threat. Apparently Black can now save his piece, but further analysis shows that the bishop is pinned and cannot be protected. The machine using a numerical evaluation function would also try 1. ♖d2×d5 and find 2. ... ♖e8-e1 mate, but it would then reject this line completely and try the apparent second best at the top of the tree, perhaps some irrelevant "threat" such as 1. ♘g3-h4. If 1. h2-h3 was tried at all, it would be as a result of some general "king-safety" heuristic rather than by reasoning from a specific line of play. To simulate reasoning of this kind, a qualitative description of the position must be maintained throughout analysis in terms similar to those of the human player. This description must be backed up the tree as the result of the analysis and used to suggest further lines of play. It is interesting to note that the example given above would present even harder problems to the machine if Black was to play, because he would have to realise that 2. h2-h3 is a real threat on White's part, forcing him to play 1. ... ♖d8-d7 2. h2-h3 ♖e8-d8 to save the bishop.

One recent approach to this problem of specifying a good description of a position is that of Botvinnik (1971) who constructs what he calls the mathematical map of a position based on the squares that the pieces can attack in one or more moves. The rest of this paper is concerned with a more conventional approach, that of suggesting that chess programs should be written in a problem oriented language which, because the syntax is based on Algol, will be called Algol 64.

## Description and Examples of the Language

A program written in Algol 64 is an algorithm for finding a move in a given chess position. Hopefully the design of the language is such that a good player will be able to write a better algorithm to find a better move than a bad player. The basic idea is that just as in Algol 60 the user can give names to variables of different types such as integer, real, or boolean, and manipulate them in expressions, so in Algol 64 he can give names like KINGSQUARE or CAPTURE to variables of type square and move. A list of types and their range of values is given below. The algebraic chess notation is used with abbreviations for the pieces, W♘ for white knight and so on.

| Type | Range of Values |
|---|---|
| rank | 1 - 8 |
| file | a - h |
| square | a1 ... h8 |
| move | all combinations of 2 squares |
| position | all board configurations |
| piece | W♗, W♘, B♔ etc. or free (unoccupied) |
| colour | White, Black |
| integer | 0, ±1, ±2 ... |
| predicate | true, false |

In addition to these, lists of variables of the same type can be formed, referred to as squarelist, movelist and so on, and there is one other type, vector, a list of mixed types for describing positions. Syntax is Algol-like for procedure declarations, blocks, conditional expressions and transfers of control, but to improve legibility assignment is denoted by becomes, equality by is and iteration through the elements of a list by

for < variable name > in < list name > do.

There are no arithmetic operators; where necessary arithmetic is done by procedure call. In the examples, user defined names will be in upper-case, constants in lower-case, and reserved words in underlined lower-case.

It will be clear that not all move constants represent legal moves in a given position. The basic system procedure for testing this is predicateprocedure POSSIBLE (POSITION, MOVE) and the corresponding movelistprocedure MOVES (POSITION). Other system functions are required for dissecting moves and finding pieces moved as in the table below.

To enable these to be implemented efficiently there is an internal representation of a position as a set of linked lists – for each square a list of all moves to it, from it and through it, for each move a list of all squares that it moves through. In this context all pieces are considered to move to the edge of the board regardless of pieces in the way. This implies that the legality of moves is dealt with by explicit programming in the source language as are anomalous moves such as the double pawn move, en passant capture, castling and promotion. The presence of rank, file and square constants makes these complications relatively easy to incorporate. Other important consequences of using this wide class of moves are that pins and discovered attacks can be readily detected by dissecting move paths and that the side effects of making a move are well defined below source level, a fact which is important when we come to consider the question of efficiency. Some examples are now given to illustrate how a legal move generator could be coded. A move will be called feasible if there are no pieces in its path.

```
predicateprocedure FEASIBLE(MOVE,POSITION);
move MOVE ; position POSITION ; begin
if IN(MOVE,MOVES(POSITION)) is false then
FEASIBLE becomes false ;
else begin square ASQUARE ;
  FEASIBLE becomes true
  for ASQUARE in PATH(MOVE,POSITION) do
  if PIECE(ASQUARE,POSITION) isnot free
    then FEASIBLE becomes false ;
  if CPT(MOVE,POSITION) is
    CPF(MOVE,POSITION) then FEASIBLE
    becomes false ;
end else ; end feasible ;
```

In a similar way, by examining the paths of moves through FROM(MOVE) a predicate-procedure PINNED(MOVE,POSITION) could be defined.

A list of legal moves for one side could then be set up by

```
movelistprocedure LEGALS(POSITION,COLOUR)
position POSITION ; colour COLOUR ; begin
  move AMOVE ;
LEGALS becomes null comment sets up the
  null list ;
for AMOVE in MOVES(POSITION) do if
  PINNED(AMOVE,POSITION) is false and
  FEASIBLE(AMOVE,POSITION) and
  CPF(AMOVE,POSITION) is COLOUR then
  LEGALS becomes
  ADDTOLIST(AMOVE,LEGALS) ; end legals ;
```

| | | |
|---|---|---|
| squareprocedure | FROM(MOVE) | the square from which a piece moves |
| movelistprocedure | MOVESTO(SQUARE,POSITION) | the moves to SQUARE in POSITION |
| squarelistprocedure | PATH(MOVE,POSITION) | the squares over which MOVE goes |
| colourprocedure | COLOUR(PIECE) | self explanatory |
| colourprocedure | OTHER(COLOUR) | the other colour |
| pieceprocedure | PIECE(SQUARE,POSITION) | the piece on SQUARE |
| colourprocedure | CPT(MOVE,POSITION) | the colour of the piece on the square MOVED to |

The four procedures that follow show how a minimal chess program could be written, where the moves analysed at each level are checks and captures or, if in check, moves out of check, and the evaluation of a position is solely in terms of mate and material. INFORMATION is a procedure that for a given position and colour computes the required predicates and movelists. DESCRIPTION combines these into a move generating procedure, MOVESWORTHANALYSING, and an evaluation procedure, VALUE. BETTER is a predicate for comparing two evaluations. EVALUATE is a minimaxing routine that calls itself recursively – a more sophisticated version would employ the alpha-beta algorithm. Apart from the position and colour of the side to move the arguments of this procedure are as follows. VARIATION is a movelist leading to the terminal position that EVALUATE finds is the best that can be forced, and EVALUATION is the result of VALUE being applied to this position. DESCRIPTION and BETTER are the procedure names mentioned above, so that the rules for move generation and comparison of positions can be made arguments of EVALUATE. It can now be seen why type vector is necessary – if it did not exist then EVALUATE would have varying numbers of arguments as the number of elements in the evaluation was changed.

```
procedure INFORMATION(POSITION,COLOUR,
   CHECK,MATE,MATERIAL,INCHECKFROM,
   CHECKS,CAPTURES,ANTICHECKS) ;
position POSITION ; colour COLOUR ;
   predicateCHECK,MATE;integerMATERIAL;
squarelist INCHECKFROM ; movelist
   CHECKS,CAPTURES,ANTICHECKS ;
begin move AMOVE ; square ASQUARE ;
   CHECKSbecomesCAPTURESbecomesnull;
   for AMOVE in LEGALS(POSITION,COLOUR)
      do begin if CPT(AMOVE,POSITION) is
      OTHER(COLOUR) then CAPTURES
      becomes ADDTOLIST(AMOVE,
      CAPTURES) ; if IN(TO(MOVE),
      CHECKSQUARES(POSITION,OTHER
      (COLOUR),PIECE(AMOVE,POSITION)))
      then CHECKS becomes
      ADDTOLIST(AMOVE,CHECKS) ;
   end AMOVE loop ;
   comment IN is the list membership
   predicate,CHECKSQUARES is a procedure
   giving the list of squares from which a
   king of given colour can be checked with
   a given piece ;
INCHECKFROM becomes null ;
MATERIAL becomes 0 ;
for ASQUARE in SQUARES(POSITION) do begin
piece PIECEONIT ; PIECEONIT
   becomes PIECE(ASQUARE,POSITION) ;
MATERIAL becomes if COLOUR(PIECEONIT) is
   COLOUR then PLUS(MATERIAL,VALUE
   (PIECEONIT)) else
MINUS(MATERIAL,VALUE(PIECEONIT)) ;
comment PLUS and MINUS are the
   arithmetic operators, VALUE is a lookup
   for the value of a piece, queen equals
   nine pawns, etc. which can be set by the
   programmer ;
if IN(ASQUARE,CHECKSQUARES(POSITION,
   COLOUR,PIECEONIT)) then
   INCHECKFROM becomes
   ADDTOLIST(ASQUARE,INCHECKFROM) ;
end ASQUARE loop ;
CHECK becomes if INCHECKFROM isnot null ;
if CHECK then comment here would come a
   block to generate ANTICHECKS,moves to
   escape check, using the information in
   INCHECKFROM. There are three
   possibilities, capture the checking piece,
   interpose or move the king. If
   INCHECKFROM contains more than one
   square it is double check and only king
   moves are legal ;
MATE becomes if CHECK and ANTICHECKS is
   null ;
end information ;

procedure DESCRIPTION(POSITION,COLOUR,
   MOVESWORTHANALYSING,VALUE) ;
value POSITION,COLOUR ; position POSITION;
colour COLOUR ; vector VALUE ; movelist
MOVESWORTHANALYSING ;
begin predicate CHECK,MATE ;integer
   MATERIAL ; squarelist INCHECKFROM ;
   movelist CHECKS,CAPTURES,ANTICHECKS;
   INFORMATION(POSITION,COLOUR,CHECK,
      MATE,MATERIAL,INCHECKFROM,
      CHECKS,CAPTURES,ANTICHECKS);
   MOVESWORTHANALYSING becomes if
   CHECK then ANTICHECKS
   else UNION(CHECKS,CAPTURES) ;
   VALUE becomes MATE,MATERIAL ;
   comment this is a new kind of
   assignment for making up lists from
   elements ;
end description;

predicateprocedureBETTER(AVALUE,BVALUE);
vector AVALUE,BVALUE
```

```
comment HEAD,TAIL are the usual list
  dissectors ;
begin if HEAD(AVALUE) then begin BETTER
      becomes false; goto OUT ; end ;
  if HEAD(BVALUE) then begin BETTER
      becomes true ; goto OUT ; end ;
  BETTER becomes if
      HEAD(TAIL(AVALUE)) ge
      HEAD(TAIL(BVALUE)) ;
OUT :
end better ;

procedure EVALUATE(POSITION,COLOUR,
  VARIATION,EVALUATION,DESCRIPTION,
    BETTER) ;
value POSITION,COLOUR ; position POSITION ;
  colour COLOUR ;vector EVALUATION ;
  movelist VARIATION ; procedure
  DESCRIPTION ; predicate procedure
  BETTER ;
begin vector VALUE ; movelist
  MOVESWORTHANALYSING
  DESCRIPTION(POSITION,COLOUR,
    MOVESWORTHANALYSING,VALUE) ;
  if MOVESWORTHANALYSING is null then
    begin EVALUATION becomes VALUE ;
    VARIATION becomes null ; end ;
  else begin move AMOVE ; vector VALUE ;
    movelist PATH ;
  EVALUATION becomes WORSTPOSSIBLE ;
  comment this is an initialising vector
    than which all others are better ;
  for AMOVE in MOVESWORTHANALYSING do
    begin
  EVALUATE(MAKEMOVE(POSITION,AMOVE),
    OTHER(COLOUR),PATH,VALUE,
    DESCRIPTION,BETTER) ;
if BETTER(VALUE,EVALUATION) then begin
    EVALUATION becomes VALUE ;
    VARIATION becomes
      ADDTOLIST(AMOVE,PATH) ; end ;
    end AMOVE loop;
  end else block ;
end evaluate ;
```

It will now be clear that an Algol-like language such as this provides the facilities needed to tackle the example given above. For suppose that if the MATE predicate is found to be true in procedure INFORMATION a movelist ANTIMATES is generated, moves that would make MATE false in the final position. These can then be passed back up the tree in EVALUATE, as EVALUATION is, and any that are legal in the current position, POSITION, given as arguments to EVALUATE to perform a further analysis. Black now has another move and can try to generate moves that avoid loss on d5, of which there are none, the bishop being pinned. It is perhaps worth noting here that moves to prevent an opponent's move, MOVE, can be generated quite easily. They are either captures on FROM(MOVE), pins of FROM(MOVE), interpositions on one of the squares in PATH(MOVE), or moves that protect TO(MOVE). As a threat is defined by a list of moves, VARIATION, representing the result of a local analysis, this provides a general method of dealing with threats. Whether it works at all well in practice is something that can only be settled by running the program.

## Implementation

As the title implies, no implementation exists yet. The plan is to write a preprocessor which converts a program in Algol 64 into Algol 60. Names of variables are retained, but substitutions are made for type declarations, assignment, conditional and for statements. Type predicate becomes boolean, types move and position integer arrays, and the rest integers. In the case of lists the integer is a pointer to a reserved area of store. List assignment statements are substituted by calls to list processing procedures, and for < variable > in < list > do by statements of the form for next in list while next not null do. The Algol 60 compiler deals with the labels, procedure calls, block structure, recursion and so on.

## Summary

An Algol-like language specifically designed for chess programming has been dscribed which retains the elements of Algol associated with procedures, block structure, recursion and transfer of control, but substitutes variable types and operations related to chess. The object of this is to provide a framework for simulating the way in which people think about chess positions, particularly with regard to the deductions they can make from the results of several local analyses. The question of learning or concept formation has not been considered. It seems premature to do so until one knows

what one is trying to learn.

One further point in favour of a specialised chess programming language is that it could be used for communicating chess algorithms, thus saving much of the duplication of effort that is apparent at present.

**Acknowledgement**

I would like to acknowledge the help of David Till and Derek Brough in many helpful discussions.

# 7.2 Robots

**Henri Vigneron**

First published (in French) in *La Nature*, 1914, pp. 56-61. English Translation first published in *Chess and Computers*, D. N. L. Levy, Batsford, London, 1986, pp. 14-23.

Mr. Torrès y Quevedo, the Spanish engineer, has been invited by the Franco-Spanish Research Centre to show his work in Paris. For this purpose he has brought with him some of the apparatus and machines which he has constructed and which are on display in the new laboratory of Physical and Experimental Mechanics at the Sorbonne, boulevard Raspail, whose director, Mr. Konigs, gave a warm welcome to the Spanish scientist and his machines.

Mr. Torrès runs the Laboratorio de Automatica in Madrid, set up in 1907 by the Spanish government. This allows him to carry on his research into calculating machines whilst at the same time constructing machines for teaching and for the scientific research of various laboratories which depend on the State. In this way Mr. Torrès is freed from financial problems and can show total impartiality in his dealings with the scientists who turn to him for help.

Mr. Torrès, who has created among other things a most ingenious model of an airship, has been kind enough to provide us with highly interesting comments about himself, his work and his machines. The latter can be divided into two groups: robots and algebraic machines. The term 'robot' is often applied to a machine which imitated the appearance and movements of a man or animal. So we are usually dealing with a machine containing its own source of energy which drives it (a spring, for example), and capable of performing certain actions, always the same, without any external influence. The most famous of these robots are the work of Vaucanson, such as the fluteplayer which he described in a paper in 1728. In 1741 he exhibited a duck which could perform all the animal's functions, including feeding and digestion. Unfortunately his collection has not come down to us in its entirety, being scattered in many German museums. He had donated it to Queen Marie Antoinette for the Academie des Sciences, but as the king was dabbling in such matters against her wishes, she placed little value on Vaucanson's collection which was thus dispersed before reaching its destination.

There is another kind of robot which is much more interesting. Rather than imitate man's gestures, it imitates his actions and can occasionally replace him. The self-guiding torpedo, the scales which can weigh and differentiate coins automatically, and thousands of other well-known mechanisms represent examples of this type of robot. Many other examples which are much more interesting can be found in factories.

The major part of industrial progress is brought about by producing machines which can tackle work hitherto done by man. Gradually we are managing to automate most operations which were originally carried out by workmen, and we talk about complete automation when production can be achieved by the sole use of machines. Mr Torrès divides this latter type of robot into two groups according to whether the circumstances which regulate their action are of a continuous or an intermittent nature.

Let us consider the self-guiding torpedo as an example of the first group. The horizontal rudder mechanism, whose task is to maintain the torpedo at a more or less fixed depth, is controlled by a chamber of compressed air which reacts to the water-pressure, and by a pendulum. Variations in depth result in the movement of a metal strip which separates the chamber of compressed air from the surrounding water; variations in tilt result in movement, relative to the torpedo, of the pendulum

which remains vertical. The horizontal rudder is linked to the pendulum and the metal strip in such a way that each of these new movements compel it to return the torpedo to the required depth. We thus see that the problem here is to establish fixed mechanical linkages between three variables: pendulum, metal strip, rudder. This is the same type of problem as all those which are studied in the theory of kinematics as applied to the construction of machines, and is of no special interest to us here.

In the robots of the second group fixed linkages play no part. On the contrary, the aim is to change these links suddenly when circumstances demand. The principle is to stop or start a pulley, open or close a valve etc., usually with a very rapid movement. In other words the automation is effected by a sudden intervention at a given moment, thus controlling each different action of the machine.

In descriptions of machines we can find countless examples of these sudden interventions, but it is clear that this kind of automation does not form part of the theory of kinematics and has never been systematically studied. Mr Torrès proposes to devote to it a special chapter of the theory of machines entitled 'automation' in which he would consider methods of constructing robots with much more complex programming (command) systems.

These robots will have SENSES such as thermometers, gyroscopes, dynamometers, pressure guages etc. Data received by the latter will be transmitted in the form of a movement: for example, a needle traversing a graduated scale.

These robots will have LIMBS i.e. mechanisms capable of carrying out given "orders" to perform certain operations. Such "orders" can be given by very simple means, even if complex operations are involved. This can be seen in the case of certain famous clocks like the ones at Rouen, Basle and Strasbourg, where a mechamism similar to the one used in an alarm-clock triggers off the movement of mechanical dolls which perform various actions.

Finally, these robots will have the necessary POWER in the form of batteries, water, compressed air etc., in order to keep the machines running and enable them to execute the essential operations.

Furthermore, and for Mr Torrès this is the main problem of automation, the robots must be able to CHOOSE intelligently i.e. carry out the required operation after taking into account the data they are receiving from their "senses" or even the data already acquired. In other words, they should be able to imitate human beings by adapting their behaviour to the existing conditions.

In theory there is no difficulty in constructing apparatus for supplying the sense data, or in providing machines which will perform the operations designated by the robot. However, when it comes to constructing a robot which will determine its actions according to its ASSESSMENT of relevant data, the general opinion is that this can be done only in a few very simple cases. For example, it is thought that automation is possible for certain purely mechanical manual operations of a workman, whereas it will never be possible for those operations which demand the intervention of the mental faculties.

Mr Torrès disagrees; for him it is self-evident that we can construct a robot all of whose actions depend on certain circumstances, numerous or otherwise, so long as we have a body of "rules" which we can impose arbitrarily at the design stage. This "program" will clearly have to be efficient enough to determine in all circumstances, without any doubt, the actions of the robot.

Not only does Mr Torrès consider that the problem is not insoluble, but on the contrary he has provided us with a very elegant solution in the shape of a chess-playing machine. We shall describe this marvel of ingenuity later, once we have explained the guiding principles behind Mr Torrès method of constructing robots.

These principles basically depend on the use of an extremely simple electro-mechanical method. We said earlier that as a general rule any variation in the circumstances influencing the action of the robot will be indicated by a movement. Let us assume that it is a switch which is moved; instead of a needle moving along a graduated scale we will have an arm sweeping through a series of points, contacting each in turn.

If there are n switches and if we call the number of points linked with each of them

$P_1$, $P_2$, $P_3$ ... $P_n$, then the total number of possible positions to be considered will be the product $P_1 \times P_2 \times P_3 \ldots \times P_n$. To each of these positions a certain operation will be linked and triggered off by a very simple mechanism such as the attraction of the armature of an electro-magnet. Thus, each position will have its own electro-magnet and in order to carry out a specific operation the electrical connections will have to be arranged in such a way that each electro-magnet will be activated when the corresponding switch is in line. In the simplest case, when only one element is involved, the solution is the one presented in figure 1. The variations of this single element are reflected in the movements of switch M which contacts in turn each of the points A, B, C, D. In the diagram the connection is being made with electro-magnet E, so the operation linked with this will be carried out as soon as the electric circuit is completed at K.

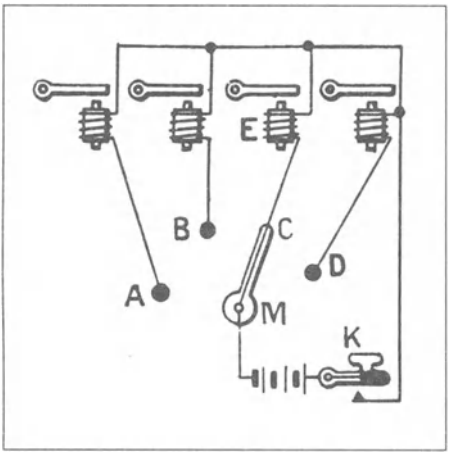



*Figure 1. Diagram showing the connections which allow four different operations to be carried out at choice.*

In figure 2, there are three main switches M, N and P. The second involves in its movement another switch $N^{I}$, and the third brings in the five switches $P^{I}$, $P^{II}$, $P^{III}$, $P^{IV}$, $P^{V}$.

M can take up positions A, B. N can take up positions E, F, G. P can take up positions R, S, T, U.

This system, then, allows 24 operations which can be carried out as soon as the corresponding electro-magnet is activated by completion of the circuit.

Of course, we can increase at will the number of switches and the number of points connected with each of them. In other words, we can increase indefinitely the number of particular cases that the robot will have to 'consider' when controlling its operations. There is no theoretical problem here, as there is no essential difference between the simplest machine and the most complex robot. Both can be reduced to a material system dependent upon the physical rules applied in their construction. The sole difference is that when these rules are complex, involving a certain amount of reasoning to deduce the corresponding manœuvres, the machine which carries out the manœuvres appears to possess in itself this ability to choose intelligently.

Indeed, this is the impression created by Mr Torrès' chess-playing machine. The object is to mate with rook and king against king, with a human chess player conducting the defence. As we have already stated, certain rules have to be established (programming) which the machine must always follow and which determine its response to any defence adopted by its opponent who has the black pieces.

Here are the rules applied by Mr Torrès in constructing the robot:

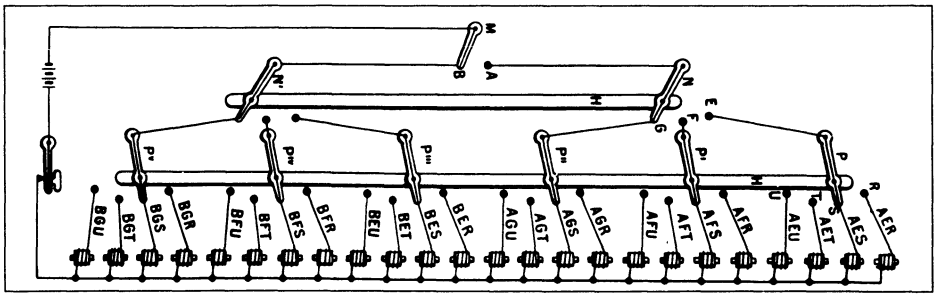



*Figure 2. Diagram showing how 24 different operations can be "selected".*

| The black King | | | | | |
|---|---|---|---|---|---|
| is in the same zone as the rook | is not in the same zone as the rook and the vertical distance between the black king and the rook is | | | | |
| | more than a square | one square, with the vertical distance between the two kings being | | | |
| | | more than two squares | two squares, with the number of square representing their horizontal distance apart being | | |
| | | | odd | even | zero |
| The rook moves away horizontally | The rook moves down one square | The king moves down one square | The rook moves one square horizontally | The white king moves one square towards the black king | The rook moves down one square |
| 1 | 2 | 3 | 4 | 5 | 6 |

*Figure 3. The six possible operations of Torrès' machine*

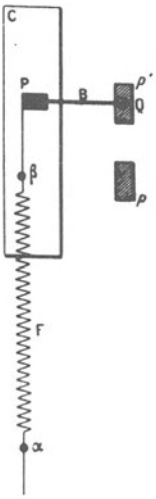



*Figure 4.*

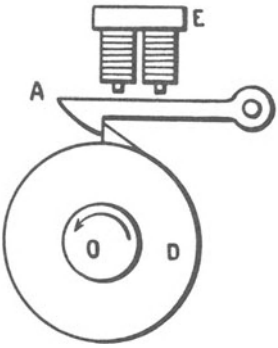



*Figure 5.*

If the opponent plays an illegal move, a light comes on and the robot refuses to make a move. Once three such illegal moves have been made, the robot ceases to play altogether.

If, on the contrary, the defence plays correctly, the robot will carry out one of six operations, depending upon the position of the black king. In order to achieve this, Mr Torrès uses two zones on the chessboard: the one on the left consisting of the a-/b-/c- files, and the corresponding one on the right consisting of the h-/g- and f- files. We then have six operations as shown in figure 3.

How are these operations carried out? Before we turn to the full picture given in figure 6, let us consider figures 4 and 5 which use the same graphical notation.

Rectangle C in figure 4 represents a slide which can move vertically. It has an arm B fixed to it at P and whose end Q can contact points $P$ or $P^1$ thus connecting either circuit. The wavy line F represents an extensible electrical conductor (e.g. spring) which connects moving point $\beta$ to fixed point $\alpha$ without hindering the movement of the slide.

In figure 5, the disc D is dragged round by friction from the central moving spindle

O unless prevented by the catch A. Each time that the electro-magnet E attracts its armature A, then disc D will make one complete rotation compelling the robot to carry out a specific manœuvre.

Now let us examine the brain of the robot:

Corridor R indicates the horizontal position of the white king.

Corridor R′ indicates the same for the black king.

Corridor T indicates the horizontal position of the white rook.

Corridor $R_1$ indicates the vertical position of the white king.

Corridor $R'_1$ indicates the same for the black king.

Corridor $T_1$ indicates the vertical position of the white rook.

There are eight discs in the diagram, 1, 1′, 2, 3, 4, 4′, 5, 5′ which execute the following specific manœuvres:

1 The rook moves to the a-file.
1′ The rook moves to the h -file.
2 The rook moves one square down.
3 The king moves one square down.
4 The king moves one square to the right.
4′ The king moves one square to the left.
5 The rook moves one square to the right.
5′ The rook moves one square to the left.

At the same time that the robot moves the piece, it also moves the corresponding slide to indicate the new position.

When the defender moves, the robot

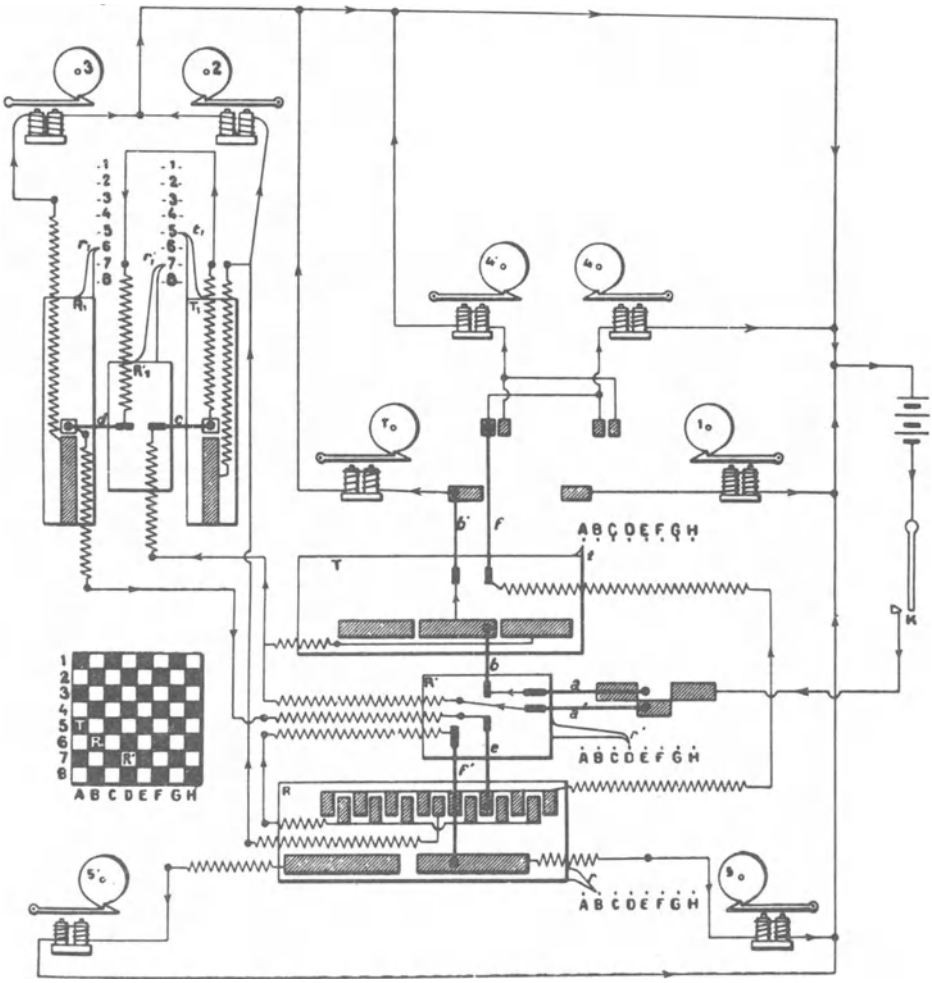



*Figure 6. Diagram showing how the chess player performs the various operations according to the positions of the pieces.*

begins by comparing the new position of the black king with the one it occupied previously. If the move is illegal, a light comes on. Otherwise the circuit is completed at K and the robot carries out one of the six operations according to its programmed rules (see figure 3):

*No. 1* Let us suppose that the black king is in the same zone as the white rook, for example on c2. The current passes along arms a and b then on to b′ connecting with the electro-magnet 1′ in the present case since the rook is in the left-hand zone. If the rook were in the right-hand zone, electro-magnet 1 would be the one activated.

*No. 2* The black king is not in the same zone as the rook and the vertical distance between them is more than one square. The current passes either through *a′* (the black king is in neither zone) or through *a* and *b* (the black king is not in the same zone as the rook). From there it goes to *c* and on to electro-magnet 2.

*No. 3* In the third hypothesis, the vertical distance between the black king and the rook in different zones is one square, but the vertical distance between the two kings is more than two squares. The current passes from *c* to *d*, then to electro-magnet 3.

*No. 4* If, in the preceding conditions, the vertical distance between the kings is two squares and the horizontal distance between them is an odd number of squares, the current passes from *d* through *e* and *f*. It then goes to electro-magnet 4 if the rook is on the a- or g- file and to electro-magnet 4′ if the rook is on the b- or h-file.

*No. 5* With the same conditions as the preceding case but an even number of squares being the horizontal distance between the kings, the current passes from *d* through *e* and *f′*. It then goes to electro-magnet 5 if the black king is to the right of the white king, or to electro-magnet 5′ if the opposite is the case.

*No. 6* Finally, if both kings are on the same file (i.e. the horizontal distance is zero), the current activates electro-magnet 2 via *d* and *e*.

Such are the principles behind Mr Torrès' chess-playing machine. However, what we have been unable to describe is the ingenuity required to create such a robot. It is only when one sees it at work, checking the opponent's moves then carrying out the correct manœuvre in reply that you can understand the truth of Mr Torrès' words: 'It is not difficult to conceive the theoretical possibility of a robot determining its action at a given moment by weighing up all the circumstances it must take into consideration to carry out its assigned task. Equally one can visualize a robot carrying out a series of actions in order to achieve a certain "result".'

In Mr Konigs' laboratory Mr Torrès also displayed other machines which were just as original. The telekine, a machine which carries out orders sent by wireless, interpreting them correctly whilst taking into account various external factors. Algebraic machines representing continuous functions by means of movements which are continuous too. A machine for solving algebraic equations, and other machines, all of which are clear evidence of the knowledge and ingenuity of this eminent engineer.

Torrès' chess machine is still in good working order and can be seen in the museum at the Polytechnic in Madrid — DNLL.

# 7.3 CHEOPS: A Chess-oriented Processing System

**J. Moussouris, J. Holloway and R. Greenblatt**

First published in *Machine Intelligence 9* (J. E. Hayes, D. Michie and L. I. Mikulich, editors), Ellis Horwood, Chichester, 1979, pp. 351-360. Reprinted by permission of Ellis Horwood Ltd.

**Abstract**

A chess-oriented processing system was built, capable of extremely rapid search in the game-tree. This report describes the special hardware designed for the system. There is also a brief summary of some software developed to make this facility easier to use, and to integrate it into existing AI-oriented programs.

**Introduction**

Computer chess was one of the first areas of research into machine intelligence. Some of the founding fathers of computer science lavished their attention on the problem (Turing, 1950; Shannon, 1950a). But after a few decades of development, the best chess programs are still no match for the best human players. In this paper we describe a chess-oriented processing system (CHEOPS) which we hope will boost the rate of progress in this area.

Ever since Claude Shannon wrote his classic paper on computer chess in 1950, debate has raged on the potential strength of his methods. By the early sixties, his fundamental ideas of minimax search and the later addition of alpha-beta cutoff were fairly well understood. These ideas have formed the basis of most chess programs, and virtually all strong ones, to the present day.

As computer chess developed during the 1960s, there became apparent a dichotomy in approaches to the search algorithm. Brute-force programs played surprisingly well, using algorithms which incorporated relatively little chess-specific knowledge, but concentrated on attaining the highest possible searching speed. On the other hand, much effort was expended on programs which greatly reduce the size of the search tree by using heuristic methods, which mimic some of the complexities of human techniques of play.

The strength of brute-force programs derives both from speed and from simplicity. The search strategy used is very straightforward; for example:

> Examine all moves for a certain number of plies (called the basic depth),and all capture chains to indefinite depth beyond that. Search by a simple recursive depth-first enumeration. After the basic depth, update the current best-so-far (mimimax) scores alpha or beta when necessary, using the material balance as a static evaluation function (sometimes simple positional terms are added in as well). Pop from a given node when there are no more successors to that position, or when one of the successors has already been shown to be an adequate rejoinder to the previous move (alpha-beta cutoff).

In the course of this search, the effectiveness of cutoffs is improved by generating moves at each node in a specific order: for example, captures of high-valued pieces by pawns first, then captures by pieces, then non-capturing moves by pawns, etc. Only modest amounts of memory are required to hold state information during the search. With sufficient static ordering (as in the example given), the algorithm requires only a push-down list containing the ancestors of the current node together with the corresponding alpha-beta values found so far at each level. (A more complex ordering might require the push-down list to hold the set of legal moves possible at ancestors of the current node, still a modest amount of memory).

Brute-force programs are usually rather small and simple. Furthermore, they are quite easy to debug. On the other hand, a moderately strong human chess player can learn to predict what variations the program will and will not see with given settings of the parameters (modulo 'oversights' on his part, of course).

The CHEOPS system is capable of executing brute-force chess programs at speeds considerably faster than any of the general purpose machines commonly available for chess research – for example, more than a hundred times faster than a Digital Equipment Corporation PDP10-KA10. Such a speed increase corresponds to an exhaustive search 6 to 8 plies ahead in about two minutes, as compared to 4 or 5 plies for current programs. One of the aims of the CHEOPS project is to determine just how quickly the standard of play improves with increased depth of lookahead in brute-force programs, as measured by ratings in play with human competition.

As an alternative to the brute-force approach, a variety of heuristic or knowledge-based programs have been developed (Levy, 1976). These also incorporate the basic search described by Shannon, but the trees searched are very much smaller: perhaps 100 to 1000 nodes are examined in the course of making a move instead of several hundreds of thousands of nodes. Two basic heuristic methods are used to achieve this reduction in tree size:

i) Instead of examining all moves from a given position, the program attempts to identify the interesting ones and examines those only.

ii) Instead of having a relatively fixed basic depth, the program attempts to adjust the search depth to the types of positions encountered, proceeding deeper only if necessary to clarify the situation.

Although these methods are clearly right in that they correspond closely to what human masters do, in practice certain difficulties have been encountered in attempting to apply these ideas. Consider, for example heuristic (i), which reduces to writing a program which tests a move for being interesting. How many different criteria are there for determining whether a given move could possibly be interesting? Let us assume a moderate number, for example 50. Supppose at a given time we have procedures for checking for 49 of these criteria coded and debugged, but have not yet programmed the fiftieth one. We find that our forty-nine fiftieths completed program plays very weak chess! Every time a move good for the fiftieth reason comes up, the program overlooks it, and hence may produce arbitrarily gross blunders, perhaps permitting checkmate in a few moves. More commonly the loss occasioned will be smaller but in tournament chess even a single blunder which loses a piece without compensation usually results in eventual loss of the game.

In chess, only within a context supplied by very nearly perfect tactics can anything called strategy or positional play exist. To arrive at this state of affairs the program must be nearly finished: in its earlier stages of development, a supposedly sophisticated program will be unable to beat even rank beginners. One can imagine that this is very discouraging for the developers of the program.

One of the aims of the CHEOPS system is to develop ways to combine brute-force lookahead with the knowledge-based approach, so that they support each other in their complementary characteristics. In the simplest case, a brute-force program running in CHEOPS can serve as a backstop to preserve the morale of the developers by assuring a respectable level of play. It also gives them an opportunity to obtain badly needed feedback before their program is completed, and it is too late to introduce major changes.

Sophisticated chess programs tend to be very large and complicated, and difficult to debug adequately. This in itself can largely explain their poor performance in computer chess championships. If a sophisticated program is designed with the knowledge that CHEOPS will be available, the size and complexity of the program can be much reduced. By supplying tactical support to the heuristic methods mentioned above, CHEOPS frees the high-level program to deal with strategic issues of play.

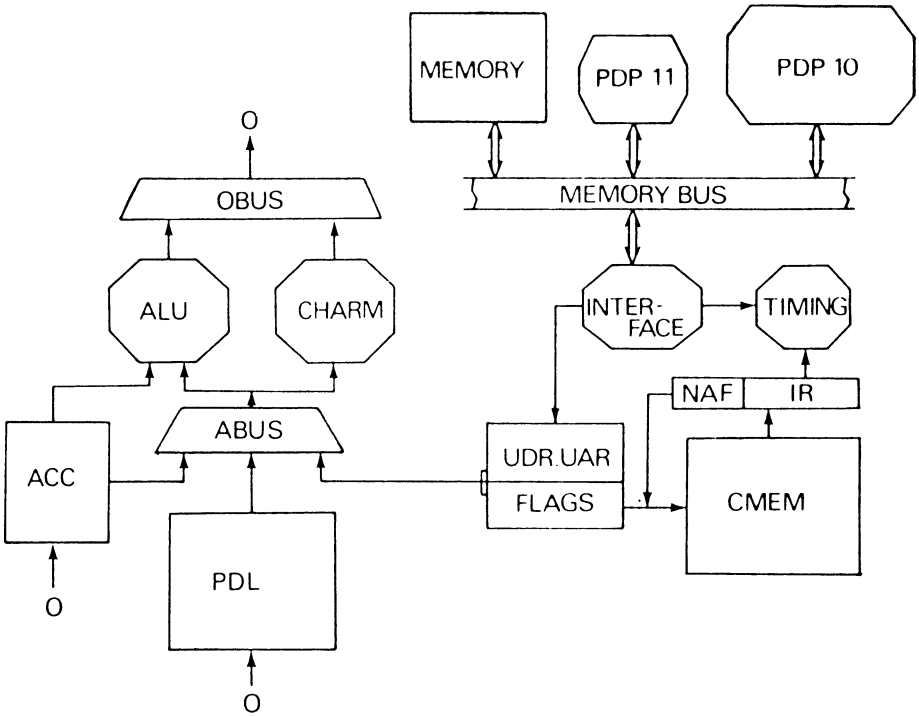



*Figure 1.* CHEOPS *Block Diagram*

## 1. Hardware

### *1.1 The anatomy of the* CHEOPS *processor*

The core of the CHEOPS system is a high-speed microprogrammable processor which is specially designed to execute chess-oriented algorithms. This processor incorporates all the major data paths of general purpose machines (see left side of Fig.1). In particular, it has a 16-bit arithmetic logic unit (ALU) which accepts operands selected from an ABUS and from a set of 16 accumulators (ACC), and produces a result on the output bus (OBUS). The ABUS selects data from a variety of sources – including accumulators, a 1024-word push-down list (PDL), and various flags and interfacing registers.

In addition to the ALU, however, the CHEOPS processor possesses a powerful chess array module (CHARM), which contains most of the chess-specific logic of the system. Like the ALU, this module takes its inputs from the ABUS, and outputs results into the OBUS, whence they may be written into any of the internal memories of the machine. The role of CHARM, however, is to perform the basic non-numeric operations which are required by all chess-playing algorithms. For example, CHARM can accept a record from the PDL of the last move tried from a given position, and generate the next possible move in a specified order. It can also detect conditions such as king-in-check and raise the appropriate flag, all in a single cycle. Thus CHARM acts as a powerful "combinatorial engine" that reduces most fundamental chess operations and decisions to the execution of a single CHEOPS micro-instruction.

### *1.2 The chess arrays*

The chess-specific power of CHARM derives from an 8 × 8 hardware chess array, whose design was originally suggested by Prof. Edward Fredkin. The purpose of this array is to find a specified subset of legal board moves. Each of the 64 squares retains information about the colour and type of piece (if any) occupying it. The basic operation of the board array is as follows: first a signal indicating side-to-move is applied; then an

individual square is designated as destination-square. The output from the array consists of 64 lines which specify which of the 64 squares contain pieces of the moving colour which can capture or move to this destination. Alternatively, a square can be designated as the originating-area, in which case the 64 lines assert possible destinations for moves from this square. Special control bits can cause the array function to be more specific: for example, it can discriminate captures from non-capturing moves, or pawn from noble captures. Also, one control bit causes the entire board to be designated as originating or destination area, so that such general questions as "any captures possible?" can be answered in a single operation.

The logic which drives the 64 square lines is a purely combinational 8 × 8 array, containing about a dozen TTL DIPS in each cell. For example, an OR-gate (actually, a NAND-gate in inverted logic) senses whether any of the 8 squares immediately adjacent to a given square has been designated; and if the given square contains a King as well, it asserts "King origin here" signal. Similarly, other OR-gates detect the presence of possible destinations for pawn and knight moves from the given square. Sliding pieces are handled differently: for example, a signal asserting "destination to Northeast" trickles through squares in each NE-SW diagonal until it encounters an occupied cell, which then asserts "Queen origin" or "Bishop origin" if the corresponding piece exists. (Since the selectors through which these sliding piece signals trickle form the longest signal path in the arrays, they are implemented in Schottky TTL). All the various kinds of origin signals are gathered together and selectively enabled by control function bits (that is, depending on whether we are currently generating pawn or noble moves) to drive the 64 corresponding square lines which form the final output of the array.

Design and construction of the chess array was greatly facilitated by use of the Stanford Drawing System. A "template" drawing was prepared for the logic required in an individual square of the array, with the signal names indexed by rank and file. Then macros were written which repeated this circuit 64 times, replacing the indices with the successive square locations. Other macros took care of edge connections and actual location of the DIPS on Augat boards. Finally, the program generated a wire list deck which was used to drive automatic wire-wrapping machines. This system not only obviated manual construction of the machine, but thanks to its error messages and file-sharing capabilities, greatly economized on the design process itself, and automatically produced a standardized set of documentation. With the help of this automated design aid, the entire design and construction of the CHEOPS hardware was completed with about one and a half man-years of effort.

### *1.3 The array module*

The inner structure of the array module itself (see Fig. 2) is dictated by the need to drive the array at a rate approaching design capacity. Most of this structure amounts to a hardwired implementation of the inner DO loops which scan pieces and squares for the next move or capture. In particular, a square scanner (S-SCAN) is provided which looks at the 64 square lines, and given a last square number (LS) generates the number of the next active square line (in raster order). Similarly, a piece scanner (P-SCAN) takes as input the piece existence register (PXR). The PXR contains a bit for each piece, which is set for each piece that currently exists on the board. Given a last piece number and the PXR, P-SCAN produces the next one, in order of the value of the pieces.

CHARM also contains memories which keep track of the board image in convenient forms. The square list records the piece on each square; and the piece list, the location of each piece. Both of these memories, with their corresponding scanners, are driven by data paths organized into two busses: the piece bus (PBUS) and the square bus (SBUS). (Note that in Fig. 2, input enters the tops of memory boxes, addresses enter the sides, and data is output from the bottom).

These 8-bit busses are entirely internal to CHARM. When the 16-bit input to CHARM arrives from the ABUS, it is divided into a left byte and a right byte, which enter the PBUS and SBUS respectively. In this piece-square format, a chess move can be coded within the machine as two words: the first describing the piece moved and origin square, and the second, the piece captured

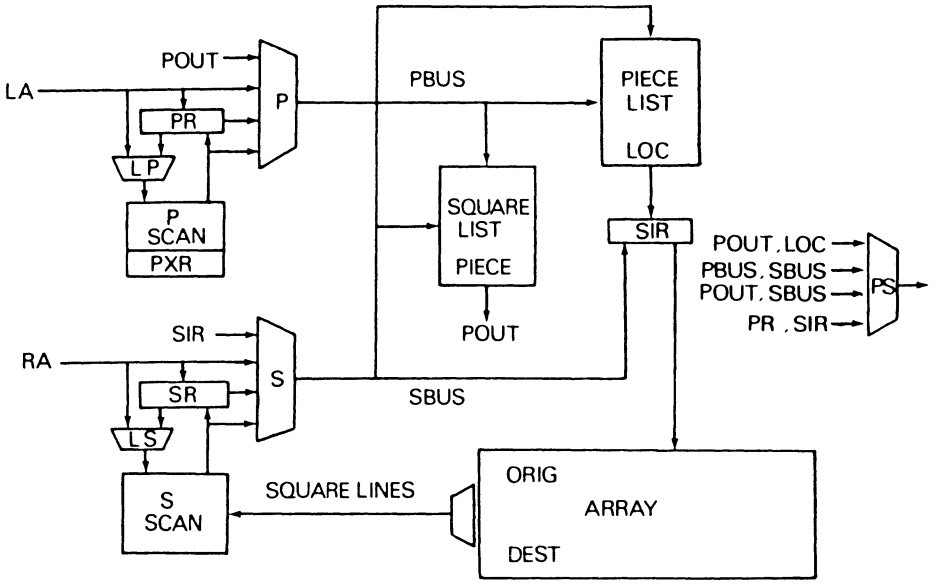



*Figure 2. Chess Array Module*

(if any) and destination square. Selectors within CHARM allow such data to be routed very flexibly. It can either be fed directly into the scanners, or be buffered in the piece and square registers (PR, SR), or loaded into the board memories. The output of CHARM (which goes into the OBUS – see Fig. 1) also maintains the piece-square format. These two halves of the output word are selected by the piece-square bus (PSBUS) from a variety of alternative sources within the module – for example PBUS,SBUS or POUT,LOC or PR,SIR (see Fig 2, right side).

Some basic operations of the chess array module are as follows. A move is *made* (rather than generated) in two CHARM cycles: first the origin square is cleared, then the moving piece is deposited on the destination square. A capture takes three cycles, an additional one to clear out the captured piece. (Although the captured piece would get properly overwritten on the board, the extra cycle is required to update the piece existence register (PXR)). Each of these cycles involves routing the desired square to the SBUS and the desired new contents of that square to the PBUS. Once this is done, activating a single control line will cause the piece-list, square-list, and piece-existence-register to get updated simultaneously. Unmaking a move is similarly accomplished by restoring the piece moved to the origin and then the piece captured to the destination on the board.

During this unmaking process, the PR can be loaded with the piece moved, the SIR (designated square input to the arrays) with the origin, and the SR with the destination. Then CHARM is immediately ready to generate a next move by scanning the array lines to find the next square to which the same origin piece can move. If there are no more moves by this piece, the next existing piece is produced by P-SCAN, its location is loaded into the SIR and S-SCAN outputs the first possible destination for this piece. There is no need to enumerate all the legal moves on the PDL. Instead, only the most recently searched move (the one in fact played in the current variation) is retained there. Later, when that move is reverted, CHARM will concurrently be generating the next move to look at, should it be decided to do so.

CHARM and the ALU work in close co-ordination during the search process. In a typical case, the blocks that are pushed and popped on the PDL for each node searched consist of four words. The first two describe the corresponding move to CHARM, the third is an alpha or beta value which the ALU checks for minimaxing and alpha-beta

cutoffs, and the fourth records such special conditions as castling rights, ghosts for *en passant* captures, promotion, etc. Just as the ALU makes available Overflow and Carry-out bits to the branching control logic of the processor, so does CHARM generate a variety of flags which influence the flow of control in the machine – for example, ANYMORE-PIECES (from P-SCAN), ANYMORE-SQUARES (from S-SCAN), KING-IN-CHECK and CASTLING-POSSIBLE (from special logic on the array). The co-ordination of CHARM and the ALU is thus facilitated by their parallel roles not only in the data paths but also in the CHEOPS processor.

### *1.4 Microcontrol*

The sequence of operations in CHEOPS is determined by a large (1K by 64 bit) RAM control memory (CMEM on right of Fig. 1). Each micro-instruction consists of four 16-bit words. At the beginning of a machine-cycle, all four of these words are loaded from the current microaddress into the instruction register (IR), from which the bits are routed to the various function control, selector, and write enable bits in the machine. The first word of each instruction includes a next address field (NAF), and a conditions field, which specify a set of four flag bits to be IOR-ed with the four lower-order bits in the NAF to produce the next microaddress. Hence every micro-instruction is potentially a 16-way branch. There is also an OP-code bit which determines whether the rest of the instruction should be decoded as an ALU or array operation. In the ALU case, the remaining words specify ALU function, A source, accumulator source, OBUS destination, etc. In the array operation case, they specify array function, bus select and write enable for memories internal to CHARM, and the A source and O destination for communication between CHARM and the rest of the processor.

The first word of an array instruction (specifying array function and internal bus select) is latched in a separate array instruction register (IRA) during ALU operations, so that arithmetic can be done during array operations that take longer than one cycle to settle. Thanks to this technique, the cycle time of the machine is about 180 nanoseconds. In other respects, the timing is completely straightforward. The IR is latched at the beginning of each cycle, setting up addresses for the sources and all the data path selectors, and specifying the function control bits for the ALU and CHARM. During the middle of the cycle, the data trickles through CHARM and the ALU and settles on the OBUS. At the end of the cycle, the destination is clocked and a new IR is strobed in. Slightly higher speed might have been achieved by attempting to overlap the RAM writes, but the resulting complexity would have made it more difficult for the user to microprogram the special-purpose machine.

The remainder of the CHEOPS system consists of a PDP-11 and a time-shared PDP-10 which communicate with the special processor through a special bus interface (see upper right part of Fig. 1). In the crudest mode of interaction, the general purpose machines can stop CHEOPS, read and write directly into memories and microstore, and restart at an arbitrary address. Without stopping the processor, the machines can effect CHEOPS through flag bits, and can read the contents of a status register. Finally, CHEOPS itself can initiate data transfers or interrupts on the UNIBUS. These various facilities provide a basis for efficient communication between algorithms running on the CHEOPS processor and high-level programs running on the other computers.

## 2. Software

### *2.1 Utility programs*

A series of utility programs were written to make CHEOPS accessible to the user from video terminals on the time-shared PDP-10 system. A chess micro-assembler (CHASM) accepts mnemonic specifications of field for CHEOPS micronode. It also resolves address labels and efficiently allocates blocks for branch destinations. A CONSOLE program provides all of the usual facilities of a console to the special processor – for example, start, stop, and single-step, examine and deposit registers, load microcode, etc. In addition it automatically translates data into alphanumeric or other implicit data types, recalls the CHASM labels corresponding to micro-addresses, displays square lines and board image, examines the stack of moves in the current game, and executes rapid store and restore of the state

of the entire machine from disc file. Finally, it implements a variety of debugging features that facilitate the finding of speed-dependent errors, as well as other less subtle problems with CHASM code.

### *2.2 Baisley's TECH II program*

Alan Baisley of the MIT Artificial Intelligence Laboratory has developed a number of brute-force oriented programs, the simplest of them resembling Gillogly's TECH Program in many respects (Gillogly, 1972). The moves from the top level position are ordered according to plausibility by a program on the PDP-10, using a variety of measures, such as pawn structure, control of centre, and king safety. The resulting positions are shipped over to the CHEOPS processor, where an alpha-beta minimax search with material balance as an evaluator is performed, returning a principal variation. This search contains a few refinements, such as a weighted depth parameter, which is incremented by 1 for normal moves and by 0 for replies to check. Out of the positions which receive the highest evaluation from CHEOPS, the first (hence presumably most plausible) is actually played. Special high-level routines deal with endgame situations.

### *2.3 The Greenblatt program*

The Greenblatt MACHACK Program (Greenblatt et. al., 1967) is also being integrated into the CHEOPS system. The CHEOPS processor plays two kinds of roles in this context. First, as a background task, it does a Baisley-type brute-force search from the top position, the result of which is used for comparison with the final choice made by the heuristic program. Second, as a foreground task, it does short searches (that is, 3 plies exhaustive, plus capture chains) from nodes within the game-tree searched by MACHACK, to provide a tactical backstop for the two kinds of heuristics mentioned in the introduction.

In particular, CHEOPS performs these mini-searches to rectify the action of the plausible move generator, alerting the high-level program to possible material gains for either side that might otherwise be missed. Finally, a mini-search is also initiated during static evaluation of a terminal node. If the result does not agree with the heuristic evaluation, or if CHEOPS indicates future material exchange in an apparently stable position, MACHACK will proceed deeper to clarify the situation.

**Acknowledgements**

This research was carried on jointly at the Laboratory for Computer Science and at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Professor Edward Fredkin was instrumental in the original conception and design of CHEOPS. The special processor was developed in parallel with Thomas Knight's CONS special processor for the LISP machine. P. Jarvis and R. Lebel helped in the use of the Stanford Drawing System; W. Freeman designed the interface to the PDP-10 and PDP-11; and C. Waldrop aided in the main-frame construction. A. Baisley provided his expertise in brute-force oriented chess algorithms. We would also like to thank J. Cocke for his encouragement and interest in this work.

# 7.4 Belle Chess Hardware

**Joe Condon and Ken Thompson**

First published in *Advances in Computer Chess 3* (M. R. B. Clarke, editor), Pergamon Press, Oxford, 1983, pp. 45-54. Reprinted by permission of Pergamon Press Ltd.

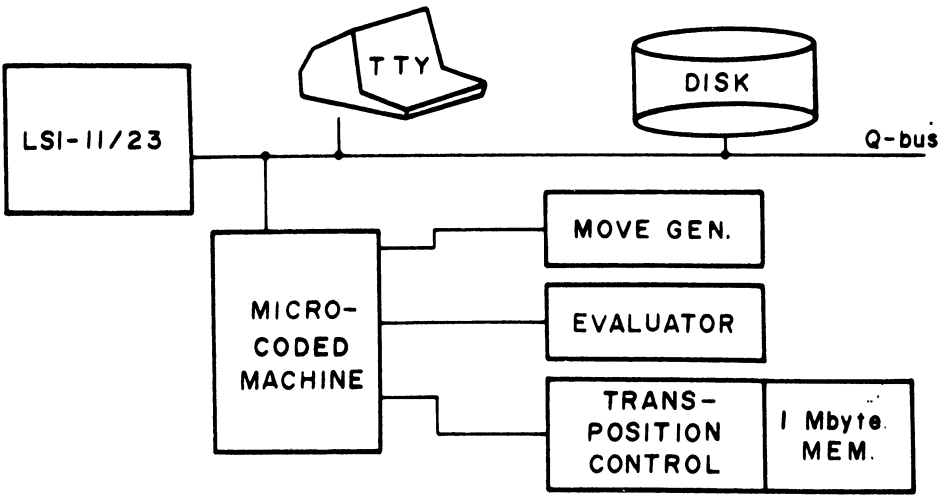
**Abstract**

The computer chess program BELLE is currently the World Computer Chess Champion and the North American Computer Chess Champion. In human play, BELLE has consistently obtained master performance ratings. This paper describes the special-purpose hardware that gives BELLE its advantage: speed.

**Introduction**

The first version of BELLE was written in 1973 and first competed in the fourth ACM Championships in Atlanta. A slightly revised version competed the next year in San Diego. This experience made it painfully clear that computer chess belonged to the fastest computer. During this time, others were working on chess hardware. Moussouris, Holloway and Greenblatt (1979) constructed a highly publicized machine but failed to include an evaluation function. Despain designed a chess machine, described by Babaoglu (1977), but never obtained funding for full construction. The authors built a small (25 chip) machine and entered it in the 1977 World Computer Chess Championships in Toronto. This first machine at best broke even on speed, but it whetted our appetites and gave us valuable experience. The next machine was quite a bit larger (325 chips). It had three separate sections - a move generator, a position evaluator and a transposition table. The alpha-beta search (Kunth and Moore, 1975) was still carried out by a minicomputer. This hardware/software combination proved to be comparable to the largest mainframes. It entered the 1978 (Washington DC) and 1979 (Detroit) ACM Championships, placing first and second respectively.

The latest machine was designed and constructed over a period of six months and contains approximately 1700 chips. The overall structure is the same as the previous machine, but the alpha-beta search has been embedded into hardware. The machine is controlled by an LSI-11/23



running a general purpose time-sharing system.

The LSI-11 contains the chess programs, opening book, and operating system software. It interfaces to a micro-coded alpha-beta machine that in turn controls a move generator, an evaluator and a transposition table memory.

The software version of BELLE examined chess positions at about 200 positions per second. The first chess machine did not change the speed much. The second machine increased the speed to about 5000 positions per second. The current machine runs at about 160,000 positions per second. These numbers are hard to compare, but the chess-specific computing power has increased about 800 times while the general purpose computing has decreased by a factor of 5.

## Move Generator

The move generator is arranged in an 8 × 8 array of similar (except for edge effects) combinatorial circuits. The move generator performs two transformations. The transform to be performed is selected by a single wire *op-code*. The op-codes are named FIND-VICTIM and FIND-AGGRESSOR.

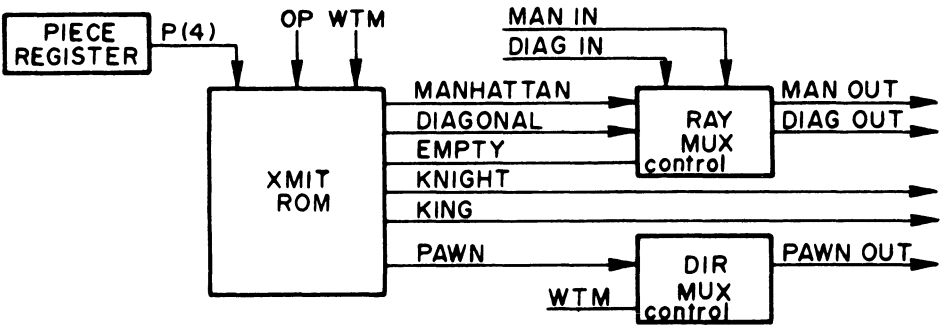
The FIND-VICTIM operation locates the highest valued piece under attack. The FIND-AGGRESSOR op-code finds the lowest valued piece attacking a given square. The given square is, of course, the previously found VICTIM square. The AGGRESSOR and the VICTIM squares are then the *from* and *to* squares for a chess move.

### *Transmitter*

Each of the 64 circuits on the move generator consists of a transmitter, a receiver, a four-bit register holding the piece and a one-bit disable memory. Each transmitter sends signals to its (chess) neighbours. Likewise the receiver accepts signals from its neighbours.

The transmitter generates attacks radiating from a square. The main component of the transmitter is a read-only memory (ROM). The ROM generates a set of wires active if the square contains the corresponding piece. The global signal WHITE-TO-MOVE is true if White is on the move.

The rays (manhattan and diagonal) are propagated or generated at a square by a multiplexor selected with the square being empty or not. There are eight ray multiplexors on each square; one for each ray direction. The pawn attack outputs are multiplexed north or south selected by WHITE-TO-MOVE.

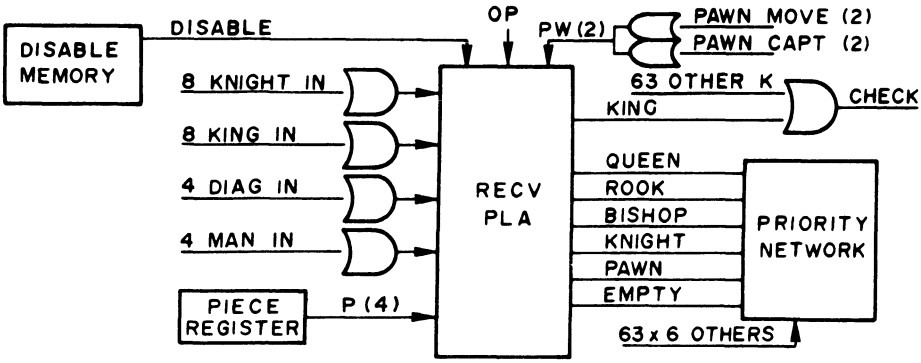



### *Receiver*

The receiver section analyzes the attacks generated by the transmitters. The main component of the receiver is a programmed logic array (PLA).

The eight knight attacks are *OR*ed together. Similarly the eight king attacks, the four diagonal (bishop) ray attacks, the four manhattan (rook) ray attacks, and the two pawn attacks are *OR*ed together. The PLA accepts the concentrated attack signals, the resident piece, WHITE-TO-MOVE, the disable bit and the op-code. It generates seven priority signals corresponding to the value of the attacked piece.

The king attacked is *OR*ed over all 64 squares and is treated specially as a CHECK condition code for the micro-coded pro-




cessor. The job remaining is to select the highest priority of the remaining 64 × 6 signals and to discover where this signal comes from. This is done in a two level priority encoding tree. The first level is performed on a local group of 4 × 4 squares. The second level priority selection determines which prioity level is highest and which 4 × 4 section generated this level. This information is combined with the location on the 4 × 4 section to complete the task.

Move generation is performed as follows. The micro-code asserts FIND-VICTIM. This causes each transmitter to activate signals corresponding to the piece on each square. The receiver sections then activate priority leads for all enabled attacks. The priority tree finds the highest enabled attacked piece. This takes place in ripple delay time through all the circuits involved (roughly 250ns).

The micro-code latches the address of the attacked piece and then asserts FIND-AGGRESSOR. This causes only the addressed victim transmitter to radiate as the union of all pieces (the super-piece). The same circuitry as FIND-VICTIM is then used to find the attacker of this victim. The PLA will match an attack by the super-piece on a corresponding piece (for example, a bishop attacked by the super-piece, but only down a diagonal). The priorities are inverted by the PLA so that the lowest valued aggressor on the selected victim is found. This operation takes another 250ns.

The micro-code latches the aggressor location and then makes the indicated move. (The aggressor is moved to the victim and the old aggressor is cleared.) After the processing of this move, the original victim and aggressor are restored. The aggressor is disabled and the next lowest priority aggressor is found until there are no more aggressors on this victim. The victim is disabled and all potential aggressors (friendly pieces) are re-enabled. Then the next victim is found and then all aggressors on this victim. This continues until all victims are exhausted.

This process is recursive. The 64 disable bits represent the state of the move generation. This state has to be saved and all disable bits must be cleared as a new move generation is started. The 64 bits are really a 64 word stack of 64 bits each. The state is saved by incrementing the address on the disable stack. The state is restored by decrementing the stack address.

The move generator contains some random logic to detect the special chess moves: castling, pawn promotion and en passant. Pawn promotion could have been included in the normal move generation, but the priority would have been roughly that of a pawn move rather than that of a queen capture. Instead, a hardware flag, PRO-ONLY, causes the affected PLAS to generate only promotions. When the flag is off, no promotions are generated. Thus a separate move generation loop exists in the micro-code to generate the promotions first. Since promotions are fairly rare, the micro-code usually only executes a single FIND-VICTIM + PRO-ONLY to determine that there are no promotions.

### Incremental Evaluation Function

The evaluation function is broken into two evaluation functions, the incremental evaluation function and the slow evalu-

ation function. As the name implies, the incremental function consists of a series of accumulators that are incrementally updated as the chess board is altered. A two-phase machine tracks all write operations to the chess board. As a square is written, the incremental evaluator reads the old contents, issues a SUBTRACT cycle, then allows the new contents to be updated and issues an ADD cycle.

*The Material Register*

The total board material is maintained in a pair of registers – one for White, one for Black. Two bits in each register are allocated to count each of the queens, rooks, bishops and knights. Four bits count the number of pawns. Several material evaluations are accomplished by ROMS addressed by the material registers. The total material is calculated as queen = 900, rook = 500, as given in beginning texts. Certain cooperating pieces are given bonuses such as a bishop pair and a queen plus knight. Minimum mating material is evaluated by discounting a bishop or knight with no supporting pawns. A material balance is evaluated to encourage trading pieces, but not pawns, when ahead. Another ROM attached to the material registers calculates the stage of the game for each side. This is done by estimating the amount of enemy material on the board.

*King Position*

The location of each king is maintained incrementally. There are also four 9-bit registers indicating existence of friendly pawns on each wing. Four ROMS evaluate each of the possible pawn formations assuming the king is behind the pawns and it is a middle game. A fifth ROM examines the actual king location and selects the pawn structure corresponding to the king's location. This ROM also evaluates the king's centralization assuming it is an end game. The actual evaluation is a weighted sum of the king safety and king centralization selected by the stage of the game.

*Piece Position*

Every piece has a value for occupying a square. This value is maintained incrementally. Pawns are more valuable for being centralised and for being advanced. Knight mobility is statically evaluated based on the squares that they occupy. Knights and bishops are penalized for being in their original square.

*Hash Code*

The incremental evaluator maintains a 48-bit hash code of the current position. This is used by the transposition table described below. For every piece on every square (16 × 64 = 1024) a ROM generates a different constant. This constant is XORed into the incremental hash accumulator on every ADD or SUBTRACT.

All of the above incremental evaluations are summed and the total is made available to the micro-code as a "fast evaluation". The fast value is available to the micro-code in less than an instruction time. Thus the micro-code can alter the board in one instruction and use the fast evaluation in the next. A conservative fast evaluation can quite often be used to cause a beta cutoff without the need for a full evaluation.

**Slow Evaluation**

The slow evaluation consists of small positional considerations added to the fast evaluation to obtain the full position evaluation. The slow evaluation is asynchronously initiated on every write to the chess board. It is an eight cycle sequence taking about 2μs. If another occurs before the sequence is finished, the operation is started again. Like the move generator, the slow evaluator consists of 64 similar circuits. Two separate evaluations are accomplished over the eight cycles.

*Pawn Evaluation*

Each cycle examines a file for pawn formations. The friendly and enemy pawns are examined on the file in question and on the file to each side. Separate evaluations are obtained for passed pawns, isolated pawns, backward pawns and doubled pawns. These values are summed and accumulated over the eight files.

*Ray Evaluation*

On every square, there is a finite state machine. Its input is a ray state, the piece occupying the square, a bit from the pawn evaluator indicating that this square is behind enemy lines, a bit from the pawn evaluator indicating that this square is a piece outpost, and a bit from the king location ROM that this square is near the enemy king. The output of the finite state machine is an output ray state and a 4-bit evaluation. All 4-bit evaluations are summed in an adding tree to yield a 10-bit ray evaluation to be accumulated over the eight slow evaluator cycles. The input state to each finite state machine is selected by an 8-to-1 multiplexor connected to the output states of the square's nearest neighbours. The multiplexor is selected by the phase of the eight evaluator cycles. Thus on the first phase, all of the finite state machines are connected north to south – evaluating rook and queen rays in that direction. The next cycle connects the finite state machines north-east to south-west. After the eight cycles, every ray on the chess board has been examined and accumulated.

The accumulated slow evalaution is added to the fast evaluation and made available to the micro-code. A condition code indicating that the slow evaluator is busy is available to the micro-code. Thus the micro-code can wait for the full static evaluation (fast plus slow) to become stable.

**Transposition Table**

The transposition table is an associative memory addressed by current board position. It is used as a cache of recently evaluated positions. It is also used to detect repeated positions. For every position remembered, two 16-bit data words are available to the micro-code. The use of these words will be described later.

The transposition table is implemented by a small fast (90ns cycle) micro-processor. The micro-processor manipulates the control leads of a standard commercial bus containing a megabyte of memory. The memory is viewed as 128K 8-byte positions. The positions are addressed by 16 of the 48 incremental hash code bits and WHITE-TO-MOVE. Thus the memory is divided into positions with white to move and positions with black to move. Four of the eight content bytes are used to hold the two 16-bit data words used by the micro-code.The remaining four content bytes hold the remaining 32 hash-code bits. The extra hash code is used to resolve collisions in the original hash addressing. There will be a false match approximately every $2^{32}$ probes. (Note that $2^{32}$ times the probe interval is about 30 hours.)

The micro-processor accepts and executes four instructions from the main micro-code. *Clear* writes zero in all of the memory. *Read* will compare the 32-bit hash code at the current hash address to the incremental hash code. If it matches, the two 16-bit data words are read and made available to the micro-code. *Write* will overwrite the current hash address position with 32-bits of incremental hash code and two 16-bit words made available from the micro-code. *Test* is used for diagnostics and not during normal operation.

The memory speed is approximately a microsecond per 16-bit word. Thus a *read* operation takes 1 μs for a typical miss and 4 μs for a hit. A *write* operation takes 4 μs. The *clear* operation takes a half-second.

**The Micro-Code**

The micro-code is implemented on a 1K × 64-bit horizontal micro-processor. Its main function is to perform an alpha-beta search using the move generator, evaluation functions and transposition table. The clock for the micro-code runs in two phases. The first phase is of constant duration and selects the timing of the second phase. Thus each micro-instruction selects its own execution time. The shortest instruction (a simple jump on a condition) takes 100ns. The longest instruction (maximum delay through the move generator) takes 375ns.

The micro-code simultaneously controls the gating on four separate data buses – the 16-bit board address bus, the 5-bit piece type bus and the 10-bit micro-instruction counter bus. Each of the buses has a 256 word read-write stack memory. All of the stacks are addressed by a 6-bit increment/decrement counter that points at the base of a four word stackframe. This stack pointer also addresses the stack of disable

bits in the move generator. A stack offset of −8 to +7 comes from each micro-instruction. Thus a micro-instruction can access any word in the last two stack frames, the current stack frame or the next stack frame. (This is just convention since there is nothing magical about the current frame.)

*The Value Bus*

The value bus is used to perform the minimal calculations necessary to implement the alpha-beta sarch needed in chess. The value bus gates 16-bit values to and from the stack, transposition table, evaluators and the LSI-11. The only operations available on these are add and compare. For diagnostic purposes, there are also connections on the value bus to and from the other buses.

Eight of the value bus lines can be sensed as condition codes in the micro-code. Commands from the LSI-11 are decoded by a binary search on these value bus lines. Flags stored in the transposition table are also tested with these condition codes.

*The Board Address and Piece Type Buses*

The board address and piece type buses are used to read and write the board memories in the move generator, incremental evaluator and slow evaluator. The stacks on these buses hold the altered contents of the chess board after a move has been made so that the state of the chess board can be restored.

*The Micro-instruction Counter Bus*

The micro-instruction bus is used to gate micro-instruction jump addresses. The stack is used to hold recursive return addresses.

**The Algorithm**

The micro-code accepts and executes nearly fifty comands from the LSI-11. Most are diagnostic in nature. Only one command is important from a chess point of view – execute a full width alpha-beta search to a given depth. The approximate algorithm used is given in an Algol-like language below.

```
search(α, β, n)
{ if (no_moves())
  { if (in_check())
      return (MATE);
    else
      return (DRAW);
  }
  if (!in_check())
    n = n−1;
  while (next_move())
  { make move();
    if (n > 0)
      v = −search(−β, −α, n);
    else
      v = −quies(−β, −α);
    restore_move();
    if (v > α)
    { if (v ≧ β)
        return (β);
      α = v;
    }
  }
  return (α);
}

quies (α, β)
{ if (in_check())
    return (search(α, β, 0);
  if (fast_eval()−MARGIN ≧ β)
    return (β);
  v = slow_eval();
  if (v > α)
  { if (v ≧ β)
      return (β);
    α = v;
  }
  while (next_capture())
  { make_move();
    v = −quies(−β, −α);
    restore_move();
    if (v > α)
    { if (v ≧ β)
        return (β);
      α = v;
    }
  }
  return (α);
}
```

*Search* will perform an *n*-ply full width alpha-beta search. Getting out of check does not count as a ply. At each leaf of the full width search, a quiescence search is performed. The quiescence search examines captures to an arbitrary depth and also all moves are examined to get out of check.

Not shown in the C code is the transposition table. On every full width node, but not at quiescence nodes, the transposition table is consulted. If it matches the current position, some processing is performed as described below. If the processing does not cause the node to be exited, then the current position is written in the transposition table entry along with an ACTIVE flag. This is a warning to descendant nodes that a subsequent match should be treated as a repeat of the position (DRAW). When processing of this node is complete, one of three results is stored in the transposition table entry: (1) a beta cutoff and the beta value; (2) an alpha cutoff and the alpha value; (3) a true value and that value. Along with the type of table entry and value is stored the depth to which this entry is valid.

Now it is possible to describe the processing done when entering a node after a match in the transposition table. First if the ACTIVE flag is set the node is immediately exited with a value of DRAW. If the depth in the node is not as great as the desired depth of search, then normal processing continues. If this node was an alpha cutoff, that means the value of the position is less than or equal to the stored alpha value. This is implemented by setting the current beta parameter to the stored alpha value plus one. This is done only if it improves (lowers) the current beta. This reduction of beta can cause an alpha cutoff on the current node. This is checked with a possible early exit from the search. Next, if the stored value is a beta cutoff, then the stored beta value (minus one) can replace the current alpha value. Likewise, this is only done if it improves (increases) alpha and when done it can cause a beta cutoff. Lastly, if the stored table entry is an exact value, that is returned.

A quiescent node that has a fast evaluation beta cutoff takes about 3 $\mu$s. The average execution time for a node is between 5 $\mu$s and 10 $\mu$s depending on position.

## Construction

The chess hardware is built on ten large wire-wrap boards. The move generator is made out of four identical boards. Each move generator board implements a 4 × 4 quarter of the chess board. Each part is constructed as if if were the south-west quarter of the chess board. Addresses to the other boards are altered to acheive the desired rotations and reflections. The colours of the pieces along with WHITE-TO-MOVE are inverted to the north-east and north-west boards so that pawns will move in the right directions. The evaluator is also made from four identical boards each implementing a 4 × 4 section. The microcode, LSI-11 interface, and the second level of the move generator priority tree all fit on a board. The incremental evaluator, slow evaluation controller, slow evaluation accumulator and transposition table microprocessor fit on the last board.

Most of the components are low power Schottky (LS) logic. This logic family is slightly slower than Schottky (S) logic, but draws much less power, thus reducing power supplies and cooling. The complete machine (LSI-11, disk, transposition memory, chess machine and power supplies) fits in a box 46cm × 48cm by 71cm tall. It weighs about 60kg. It is portable, but one has to be dedicated to take it anywhere.

# 8 The Endgame

It is well known that computer programs play far less strongly in the endgame than they do during the earlier phases. This is because the endgame requires a much deeper *understanding* and far more *long range planning*, and these are aspects of chess which have not yet been converted into computer programs.

Some of the earliest work on chess endgames was the Ph.D. thesis written by Barbara Huberman at Stanford University. She did not discuss the endgame in general, but concentrated her efforts on the problem of extracting information from a chess book in such a way that it could be programmed. In this sense, Huberman was one of the early "knowledge engineers" in the field of Expert Systems. Her work was to program the endings: king and rook v king; king and two bishops v king; and king, bishop and knight v king. She did this using the concept of "better than", so that her program could make simple comparisons between the merits of various positions. The earliest work on the endgame which is included in this compendium is the B.Sc. thesis written by Kenneth Church at M.I.T. (section 8.1). Church discusses a particular class of pawn endgames, called co-ordinate squares problems, and generalises from blocked pawn formations to less specific king and pawn endings. Church's work has planning implicitly built-in, and in my opinion it is the most significant advance to have been made in endgame programming.

Planned search in the endgame is also the subject of the paper by Jaap van den Herik (section 8.2). It is another example of "intelligent" search for the best move, and as such is a pointer for future workers in the field of endgame programming.

Certain standard endgames have been programmed in such a way as to allow perfect or near-perfect play by computers. This work started in the Soviet Union, with a routine to play the endgame of king, queen and g-pawn (or b-pawn) against king and queen. International Grandmaster David Bronstein, who was once a challenger for the World Championship, actually reached this endgame in a tournament in the Soviet Union in 1975. During the adjournment he telephoned the programmers who looked up their database and told Bronstein how to play. When the game was resumed, Bronstein followed the program's recommendation and eventually won (his opponent deviated from the expected line of play). After the game it was discovered that the program actually had an error in the key variation, overlooking a stalemate possibility, but I understand that this mistake was immediately corrected.

That particular ending has long been considered by masters and grandmasters to be one of the most difficult to play well. The computer of the 1980s plays such endings by creating a complete database of all feasible positions, and by backtracking from positions which are definitely won or definitely drawn the program is able to decide exactly what move should be made from every possible position, what the result should be, and in how many moves the game should end. This technique has now been used by Ken Thompson's BELLE in a number of fundamental endgames, with remarkably interesting results:

*(a) king and queen v king and rook*

This is a win for the queen, but not an easy one. International Grandmaster Walter Browne, who has won the U.S. Championship on many occasions, did not believe that he could fail to win this ending against BELLE within the prescribed limit of 50 moves. He tried and drew! BELLE always puts up the best defence, deferring the inevitable defeat until the last possible moment.

*(b) king and rook v king and knight*

In general this is a draw, but there are many more losing positions (and ways to lose) than chess experts had suspected. Although the ending should normally be drawn, it is by no means as easy for the defending player as one might believe after

consulting an endgame book.

*(c) king and two bishops v king and knight*

It was thought that this configuration is sometimes a win and sometimes a draw. BELLE has shown that a win can always be forced, though from some positions it requires as many as 67 moves to win the knight. Because of Thompson's work it will be necessary for F.I.D.E. to modify the Laws of Chess, by adding this configuration to the class of positions that cause the 50-move rule to be extended to 100 moves.

It is interesting that for this particular ending the winning process is so obscure, and lacking in chess logic, that a human endgame expert – Grandmaster Pal Benko – could not believe the program's winning method. In chess it is normally possible to say that a move wins because it achieves or helps to achieve some sub-goal on the winning path. The win with the two bishops against the knight often requires moves whose only explanation is that they win!

*(d) king, rook and bishop v king and rook*

Again this is normally a draw, although certain traps need to be avoided. The Swedish International Grandmaster Ulf Andersson has won this ending at least twice, against strong opponents, so the drawing technique cannot be said to be easy. BELLE has in fact shown that there exist some positions from which it is possible to force a win but not in less than 50 moves. So here is another configuration which required modification to the Laws of Chess, and in fact the Laws were amended as from January 1st 1985 to include this possibility.

The ending of king, rook and pawn v king and rook, first attracted the attention of chess programmers in 1974. I was talking to the programmers of CHAOS, during the A.C.M. tournament in San Diego, and they expressed doubt at my statement that within a year they would not be able to program a computer to play this configuration correctly for both sides, winning with the extra pawn whenever a win was possible, drawing with a pawn less whenever the game really should end in a draw. The discussion closed with a $100 bet, and at the following year's tournament they paid up, admitting tha the task was so difficult that they had not even been able to start on it.

Being rather greedy, I made a similar bet with Dr. Arlazarov of KAISSA fame during the 1977 World Computer Championship in Toronto. This time the bet was a case of Scotch (if I lost) against a case of Vodka. We agreed that Yuri Averbakh, President of the U.S.S.R. Chess Federation and a renowned endgame expert, would act as arbiter. Just about one year later I heard that I owed Arlazarov a case of Scotch, and section 8.3 of this compendium describes how he and his colleagues collected the wager.

# 8.1 Co-ordinate squares: A Solution to many Chess Pawn Endgames

**Kenneth Church**

First published as a B.Sc. thesis at the Massachusetts Institute of Technology, 1978. Reprinted by permission of Massachusetts Institute of Technology, and of the author.

**Abstract:**

The solution to a number of AI problems depends upon knowledge representation. The theory of *Co-ordinate Squares* is a framework for representing the relevant ideas in a large class of king and pawn endings. Some endgame problems which require twenty to thirty plies of analysis using a more traditional representation, a branching tree of moves, can be efficiently solved using this theory.

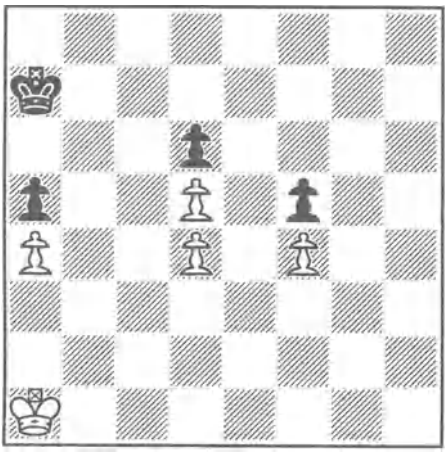


*Figure 1 white to move wins; black to move draws [Fine (1941), problem #70] [Averbakh (1974), problem # 671] [Lasker 1901]. (The author of* PEASANT[1] *estimates that this problem would require 25,000 hours of cpu time. [Frey (1977a), page 129])*

[1] PEASANT, the only chess program designed to play only king and pawn endgames, uses a depth first alpha-beta branching tree strategy. The static evaluator is:

$10 * \mathrm{MAT} + 5 * \mathrm{PP} - \mathrm{PRO} + K_1 + K_2 + R$

MAT ≡ the difference between the number of white pieces and the number of black pieces

PP ≡ the difference between the number of white passed pawns and black passed pawns

PRO ≡ the number of moves the most advanced white pawn must take before promotion minus the number of moves for the most advanced black pawn

$K_1$ ≡ factor measuring king distance from the pawns: five points deducted for every space that separates the king from the "center of gravity" of the pawns

$K_2$ ≡ three points if the king has opposition

R ≡ ten points times the rank of each pawn that is passed and cannot be stopped by the defending king.

The forward pruning heuristic eliminates king moves which take the king more than two columns or three rows from the nearest piece and king moves to the edge of the board if there would otherwise be eight legal king moves. The *killer heuristic* (Frey, 1977a chapter 3) is used to improve the effectiveness of alpha-beta. Also, moves are sorted so that captures and promotions are first. A position is defined to be a terminal node if one of the following conditions holds:

1. The maximum preset depth is met.
2. One side has one or two pawns and the other has none. (There is a special static evaluator for these case.)
3. There is a queen on the board and the last move was not a promotion.
4. There is a passed pawn which cannot be caught by the enemy king and can outrace all enemy pawns with a move to spare.
5. The depth is equal to that of a node where a win can be guaranteed. (This appears to be a special case of alpha-beta.)
6. The same position has occured previously at the same depth in the tree.
7. Stalemate
8. The position is equivalent to a parent position which occurs four plies higher in the tree.
9. The winning side allows draw by repetition.

The most serious design error is that rule six is too weak. A better condition is to terminate if the position has been reached in the tree search at any depth. Especially in these endgames, this is a very serious error.

**What are Co-ordinate Squares?**

A good representation of a chess endgame should be based upon as large an invariant as possible. That is, the concept of a move is too elementary and too transient to show what is going on. A tree of moves is consequently too large because the branches don't contain enough information. If the representation models something which doesn't change during the game or changes only very occasionally, then the structure will be much more manageable.

In this work, the representation is a map of *co-ordinate squares* [Averbakh (1974), chapter 7]. This map shows us how one side, the defender, is co-ordinating with the other, the attacker. Intuitively, the defender is co-ordinating with the attacker if, for any move the attacker makes, the defender has a defending reply. Since we have limited ourselves to the restricted domain of king and pawn endgames, we can use domain specific constraints to determine whether the defender is co-ordinating with the attacker.

The most important observation is that there are two distinct classes of moves in king and pawns endgames, king and pawn moves. If we can de-couple these two classes, then there would be a relatively small manageable number of distinct positions that could result after an arbitrary series of moves. It is relatively easy to understand what will happen after an arbitrary series of reasonable pawn moves. Pawns can rarely queen without king assistance. Usually the pawns have to be passed or half-passed to be a threat. One method for locating the threatening pawns will be outlined in detail later in this paper. King moves are much more difficult since kings have a much larger branching factor than pawns. However, since there can be no more than $64^2$ ways to place two kings on the board, the number of positions resulting after an arbitrary series of king moves is sufficiently constrained for our purposes. The final step, deciding how the king and pawn moves are coupled, is the most heuristic, requiring considerable domain specific knowledge. We will first study king moves, then pawn moves, and finally the coupling effect between the two classes of moves.

**King Co-ordination**

Let one king be the attacking king and the other be the defender. Can the defending king defend all the attacking king's threats? If we can answer this question for all ways of placing the two kings, then we have the map of co-ordinate squares. Since the attacking king is free to move wherever he chooses whereas the defending king is obligated to move to a defense of the attacking king, the map of co-ordinate squares is organized as sets of defenses for each *attacking square*, that is, for each placement of the attacking king.

What is a defense? A defense must satisfy two conditions: first it must be a possible defense according to a static evaluator. Secondly, it must be connected to other defenses so that there is a defensive reply to any single move by the attacker. The second condition is sufficiently strong to assure that the defending king can still defend after an arbitrary series of king moves by the attacker.

The fact that defenses are invariant is not completely obvious. One might imagine that, even though the defending king is holding against all threats, the attacker could still win by moving away and then returning at a more advantageous time. However, the connectedness condition is designed so that defenses will be invariant. Intuitively, if the attacker could out maneuver the defender, then there must be a position where the attacker can make some move for which the defender has no reply even though the defender was holding before the attacker moved. Since this contradicts the connectedness criterion, defenses must be invariant.

Given an attacking square, it is possible to determine the *defense set*, the set of all defenses to a given placement of the attacking king. We say that the defending king co-ordinates with the attacking king if, after the defender's move, the defending king occupies a square in the defense set of the attacking king. Since defenses are invariant, we can determine if co-ordination can ever be achieved by checking the defense set of the attacking king. If the location of the defending king is not the defense set, then we know the defender is lost because he cannot achieve co-ordination. More generally, we can determine the result of any chess endgame (win, loss, or draw) if we know whether or not co-ordination can be achieved. There are only three possibilities: both sides are co-ordinating with each

other, white is co-ordinating with black but not vice versa, or only black is co-ordinating with white. These positions are drawn, won for white, and won for black respectively.

Only slightly more effort is necessary to find the best move. The king trying to win must maintain his winning position and avoid an infinite repetition. In terms of co-ordinate squares, maintaining a winning position simply means that the winning king must co-ordinate with the other king and prevent the other king from co-ordinating with him. The author has spent very little effort developing an attractive algorithm for avoiding repetition because co-ordinate squares so constrains the possibilities that almost anything will do. To draw, it is only necessary to avoid a lost position, a position where it is impossible to co-ordinate with the other king. Unfortunately, this theory doesn't help a lost position. One strategy might be that the losing king should play the move which gives the attacking king the greatest opportunity to make a mistake.

The difficulty with co-ordinate squares is that it is necessary to determine the defense sets in order to know whether or not co-ordination can be achieved. Unfortunately, this task can be extremely difficult and is most likely impossible in the general case. Much of this paper will concern itself with heuristics for generating these sets in specific cases.

The simplest case (such as figure 1 shown above) is where neither side has any pawn mobility and the defending king must defend all the opponent's threats (he cannot generate his own more immediate threats called *counter attacks*). Although this example is the first and most elementary to be considered, the reader is challenged to solve it. Even experts, USCF rating of 2000 to 2200, have been stumped. At any rate, for the purposes of explanation, let us consider black to be the defender. We are thus exploring the question of whether black is co-ordinating with white.

## Finding Targets

First of all, we must determine the targets, which squares the attacking (white) king ultimately wants to acheive. Let us define a target for a particular side *s* to be a square that side *s* would like to occupy with his king. Since there are times when both sides can achieve targets, we must quantify the value of obtaining a target. We will assume that if side *s* can occupy the target square, side *s* will queen a pawn in *n* moves, no matter where the defending king is. The value of a target is defined to be the *n* moves before promotion. In general, it is very difficult to locate targets and assign values to them. For a first approximation we will assume that the attacker wins as soon as he captures a pawn. In figure 1, the targets are the black pawns on squares a5[2], d6, and f5.

There are two possible errors due to this simple-minded scheme for selecting targets which will be considered later. First, targets need not contain a pawn. Consider a passed pawn on the seventh rank. If the attacker can defend the pawn and the queening square without blocking the pawn, then the attacker will queen in one move. Certainly, there are less extreme examples. A second possibility is that the defender may still be able to defend even though the attacker has obtained a target. The most common case is mounting a counter-attack, a more immediate threat than the attacker's. For now, we can ignore these problems since they don't greatly influence our example (figure 1).

## Finding possible defenses

The first criterion, analogous to a static evaluator, finds all possible defenses, considering only races for targets. That is, for now we will assume that the defender will lose unless his king is as close to every target as the attacking king. We need a few definitions before we can precisely determine the defense sets (dsets.)

## Legality

A position is legal if the attacking king and the defending king, on a and d, respectively, are not adjacent and the attacking king is not attacked by a defender's pawn and the defending king is not attacked by an attacker's pawn. (The argument attacker is a binary variable, whose value may be either WHITE or BLACK.)

---

Church's footnote [2] explains algebraic notation. This has already been done on page vii – DNLL.

legal (attacker, a, d) ≡
  ~ adjacent (a, d) &
  ~ in-check (attacker, d) &
  ~ in-check (other-side (attacker), a)

**Distance**

We have three kinds of distances, white's distance, black's distance, and the distance on an open board. White's distance between two squares, a and b, is the number of white king moves it would take to move from square a to square b considering pawns as the only obstacles. That is, the white king can move to any square which does not contain a white pawn and is not attacked by a black pawn. Distance on an open board is the number of king moves assuming no obstacles.

**Closeness**

The attacking king on square a is closer to the target square t than the defending king d if the attacker's distance from a to t is less than the defender's distance from d to t.

closer (attacker, a, d, t) ≡
  distance (attacker, a, t) ≤
  distance (other-side (attacker), d, t)

**Defense Set**

A square d is in the defense set (dset) of an attacking side attacker (either WHITE or BLACK) and a square a iff d is legal for the defender and d is as *close to* every target as a.

dset (attacker, a) ≡
  {d | legal (attacker, a, d) &
  $\forall_{t \in \text{targets}}$ ~ closer (attacker, a, d, t)}

Getting back to figure 1, let us find the defense set of c4 where white is the attacker. c4 is two moves from target a5, three squares from d6 (c5 is illegal for white), and seven from f6. According to this criterion, black can defend c4 from a7, b7, c7, a6, b6, or a3. (See figure 2 shown below.) In other words, if white's king is on c4, then black's pawns are safe only if the black king is on one of the above defenses a7, b7,c7, a6, b6, or a3. We will see later that most of these defenses are inadequate. Also, we will generalize the meaning of defense to mean that black does not lose, not just that he holds all his pawns.

**The Triangulation Criterion**

The second criterion accounts for arbitrarily complex king maneuvers some of which chess players call triangulation, opposition, distant opposition, co-ordination, tempo-ing, or zugzwang [3] by assuring the connectedness condition mentioned above. (We use the subscripts *a* and *d* to denote attacking and defending squares, respectively.) The observation is: if defense sets A and B defend square $a_a$ and $b_a$ respectively, and if the attacking king can move from $a_a$ to $b_a$ in one move, then the defending king must be able to move from any square $a_d$ in A to some square $b_d$ in B in exactly one move. Squares not meeting this constraint, hereafter called the triangulation criterion, are removed from the defense sets.

---

[3] These terms are difficult to define either because they are advanced chess concepts that the author doesn't fully understand or because they have no precise definition.

*triangulation* — Taking a longer than necessary route toward a target so that the position will recur with other side on the move.

*opposition* — The state where the two kings are on the same rank or file and there is exactly one square in between.

*diagonal opposition* — The state where the two kings are on the same diagonal and there is exactly one square in between.

*distant opposition* — The state where the difference in ranks of the two kings and the difference in files are both even.

*co-ordination* — (relatively new term) The state where there exists a defensive reply to wherever the attacking king moves.

*tempo-ing* — A catch-all term to cover arbitrarily complex attempts to repeat the position with the other side on the move. Although triangulation is the most common method, there are other possibilities such as pawn maneuvering.

*zugzwang* — A state where the side to move has only bad moves but would be better if the other side had the move.

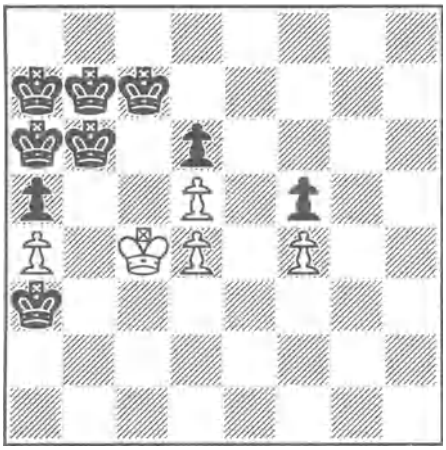


*Figure 2 The black kings mark the elements of the defense set (before triangulation)*

A side can move from square *a* to square *b* if the two squares are adjacent and the destination square *b* is legal for the side *attacker* assuming that the defending king is on square *dking*.

can-move? (attacker, a, b, dking) ≡
adjacent (a, b) & legal (attacker, b, dking)

Given two attacking squares, $a_a$ and $b_a$, and their respective defense sets before triangulation, A and B, we want to find the new defense sets after triangulation. We need two functions to manipulate defense sets, FETCH-DSET and REMOVE. The function FETCH-DSET retrieves the defense set of a given attacking square from the map of co-ordinate squares. The REMOVE function removes a given defense from a given dset with side effects.

triangulate: proc (attacker, $a_a$, $b_a$)
A := fetch-dset (attacker, $a_a$)
B := fetch-dset (attacker, $b_a$)
defender := other-side (attacker)
for all $a_d$ in A do
if $\sim\exists_{b_d \in B}$ [can-move? (attacker, $a_a$,$b_a$,$a_d$) → can-move? (defender, $a_d$, $b_d$, $b_a$)]
then remove ($a_d$, A)

Returning to our example (figure 1), let $a_a$ be c4 and $b_a$ be b5. Then A= {a7, b7, c7, a6, b6, a3} (as previously calculated) and B={} according to the first criterion. Does a7 still defend c4 after triangulation? Since white is threatening to move from c4 to b5, all defenses to c4 must either prevent white from moving to b5 or must be adjacent to some defense of b5. Since there are no defenses to b5, all defenses of c4 must prevent white from moving to b5. In particular, a7, failing to keep white out of b5, is an inadequate defense of c4. Similarly, b7, c7, and a3 are removed from A. At this point A={a6, b6}.

It should be apparent that defenses depend upon neighbouring defenses. After having heavily reduced the defenses to c4, we should suspect that the reduction might propagate. For example, a8 defends against b3 before triangulation. Since white can move from b3 to c4, black must be able to move from a8 to some defense of c4. Before we eliminated a7 from the set of c4, black could move from a8 to a7. We can see that reducing the defense set of c4 causes the defense set of b3 to shrink.

This propagation effect explains why a6 is eventually removed from the defense set of c4. Notice that there is no defense to h5. Consequently, the only defense to h4 is g6. Defenses to g3 must be on the f file. Continuing this reasoning, the algorithm discovers the constraint given in Fine, the black king cannot allow the white king to be two files closer to the king side. If black attempted to defend c4 from a6, he would lose his pawn on f5.

**Selected results**

| white attacking square | black defense set |
|---|---|
| b5 | { } |
| c4 | { b6 } |
| d3 | { c7 } |
| c3 | { b7 } |
| b3 | { a7,c7 } |
| d2 | { c8 } |
| c2 | { b8 } |
| b2 | { a8, c8 } |
| d1 | { c7 } |
| c1 | { b7 } |
| b1 | { a7, c7 } |
| a3 | { b8, b7, d7, d8 } |
| a2 | { b8, b7, d7, d8 } |
| a1 | { b8, b7, d7, d8 } |
| ... | |

The extremely small size of the defense sets illustrates just how constrained black really is. A similar calculation with white as the defender yields much larger sets, as would be expected.

With white to move, the above map tells us that black cannot co-ordinate with white

because it is after black's move and the black king on a7 does not occupy a square in the defense set {b8, b7, d7, d8} of the white king on a1. If black were on b8, b7, d7, or d8 the game would be drawn. However, since this is not the case, black is lost. White can demonstrate the win only with ♔a1-b1. Anything else would be a grave mistake, allowing black to co-ordinate. Let us consider the other possibilities, ♔a1-a2 and ♔a1-b2. ♔a1-a2 could be answered with either ♔a7-b7 or ♔a7-b8. After the other possibility, ♔a1-b1, black must play somewhat sharper, and only ♔b7-a7 or ♔b8-a7 will save the game.

On the other hand, with black to move, black can co-ordinate, drawing the game, with either ♔a7-b7 or ♔a7-b8. Neither Fine nor Averbakh suggest ♔a7-b8 because that defense is somewhat "anti-theoretical" meaning that it gives away the twisted distant opposition.[4]

**The Triangulation Criterion Viewed as a Waltz Filter.**

Waltz [Winston (1975), chapter 1] developed a filtering process to search a line drawing scene. The problem is to find all possible ways to label the lines (– for concave, + for convex, and either ← or → for a boundary) in a scene so that lines meet in physically realizable vertices. Waltz's algorithm first assigns each vertex a set of possible labels, i.e. physically realizable. The next and final step is the filtering process which assures that connected vertices are labeled consistently, that two vertices connected by a line are labeled the same way at both ends. The procedure is to iterate through the vertices checking each possible label for consistency with each connected neighbor. If any possible label is found to be inconsistent with a neighbor, that label is removed from the possible labels. It is possible that the effects of removing a label might propagate to other vertices. After every removal, the possible propagations are immediately considered.

The triangulation procedure is remarkably analogous to Waltz's filter. The vertices correspond to the possible locations of the attacking king. A defense set can be thought of as a set of physically realizable ways to label a vertex. Two attacking squares are connected if the attacker can move from one attacking square to the other. Just as two vertices connected by a line must be labeled consistently, the defense sets of two attacking squares must be consistent. That is, the defender must be able to move from any square in one defense set to a square in the other defense set.

This analogy shows us that we can apply the triangulation constraint in time proportional to the number of vertices (~64) times the number of possible labels (~64), which is pretty good. Actually we can probably do even better.

How many comparison operations, in this case applications of the triangulation criterion, must we perform? They must be performed until the graph, i.e. map of co-ordinate squares, has "settled", in that future comparison operations will not shrink a defense set. At this point the graph is stable. It is easy to show that the procedure will terminate. A comparison operation can either leave the defense sets alone or shrink them. Since the sets are of finite size, only finitely many removal operations can be performed. If all sets are emptied, no further removals will shrink them, and the graph is therefore stable, so the algorithm halts. Hopefully, though, the map will settle before every element is removed.

The time to triangulate is related to the number of comparison operations required. Waltz has given us the upper bound discussed above. However, in general, it is possible to improve upon this upper bound by comparing the sets in the optimal order. The most constrained sets should be compared with their neighbors first because they are most likely to shrink their neighbors. We can assure that the most constrained sets will be considered first if we keep a sorted queue of sets to be considered.

How do we decide if one set is more constrained than another? The best answer is to employ some domain restricted knowledge. For example, we might use a heuristic which suggests that attacking squares closer to targets tend to be more constrained. However, in this work, we

[4] Opposition can be twisted by certain pawn formations. Since the opposition is usually an asset, it is understandable why both Fine and Averbakh might overlook the fact that the opposition is not necessary for black to draw.

have chosen a much simpler and domain independent ordering heuristic. Given two vertices, *a* and *b*, associated with possible labels sets *A* and *B*, we define *a* to be more constrained than *b* if set *A* is smaller than set *B*.

The queue inprovement leads to some other optimizations. In Waltz's filter, the propagation is checked immediately after a set is reduced. An improvement is to re-queue a vertex after the associated possible labels set has been reduced. This assures that the vertices will be considered in the proper order. By checking for propagation immediately, this order cannot be guaranteed.

Another advantage of the queue is that it can avoid considering some vertices multiple times. Imagine that a comparison operation shrinks a defense set which is already on the queue. Instead of queuing the vertex twice, the vertex is simply re-ordered on the queue. If we didn't have the queue to remember which vertices must be looked at, then we would have to compare the vertex against its neighbors twice. It is very difficult, though, to see exactly how good the queue is.

**Bottleneck Defenses**

An important observation is that the triangulation heuristic correctly weeds out most mistaken defenses generated by the first criterion as long as they are sufficiently far from targets. Erroneous defenses close to targets might survive triangulation since there is a greater chance that the erroneous defenses will be next to a correct defense. At any rate, it is easy to find examples where triangulation corrected an error in the first criterion. One such example is that b7 doesn't even defend c4 in a simple race and yet b7 is included in the defense set of c4 until the triangulation heuristic removes it. This insight leads us to the conclusion that only squares near targets must be calculated carefully. Including too many squares sufficiently far from targets in defense sets will be corrected by the triangulation.

In particular, the reason that b7 is included in the defense set of c4 is that the route from b7 to a5 passes through a bottleneck (b6, a6) which white can seal off with ♔b5. Fortunately, in this case, the bottleneck caused too many squares to be included in a defense set. However, this need not always be the case. For example, what is white's defense set when black's king is on h5? (When we are looking for white's defense set, black is considered the attacker.)

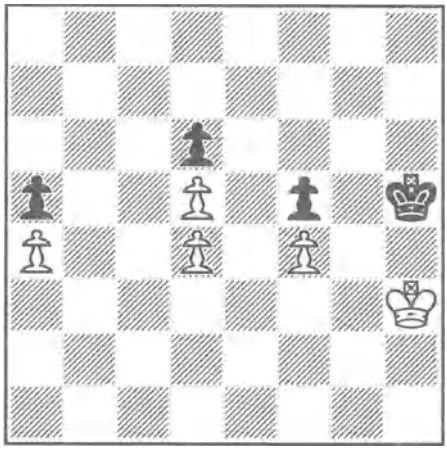


*Figure 3 Why isn't black's king closer to d5?*

For the white king to get to d5, white has to move all the way around his d4 pawn (h3-g3-f3-e3-d3-c4-d5), a total of six moves. Since there are no pawn obstacles hindering the black king, black can reach d5 in only four moves (h5-g4-f4-e4-d5), long before white can defend it. The algorithm deduces that black is closer to d5 than white.

The reason black isn't closer to d5 is that black cannot break through the bottleneck on g4 and h4, black's only entrance to white's territory and white's pawns. How do human chess players almost immediately "see" that the bottleneck holds? There are two components to a bottleneck defense. First, we must suspect that a bottleneck defense might work and secondly, we must check that it is connected to other defenses. The later connectedness condition is our familiar triangulation criterion. Suspecting a bottleneck defense is somewhat more difficult. After considerable experimentation, we have arrived at the following quite satisfactory heuristic.

A possible bottleneck defense (before triangulation) exists when the defending king d is in a position to block a bottleneck square on the path from the attacking king a to a target t. To do this, the defending king must occupy a square which is, from the attacker's point of view, no farther from the target than the attacker's king.

Secondly, the defending king must be between the attacking king and the target. In other words, using what mathematicians call the triangle inequality, the distance between the kings must be strictly less than the distance from the attacking king to the target.

bottleneck (attacker,a,d) ≡
distance (attacker, d, t) ≤ distance (attacker, a, t) & distance-on-open-board (d, a) < distance (attacker, a, t)

A bottleneck defense is an alternative defense to the "closer" defense previously discussed.

dset (attacker, a) ≡
{d | legal (attacker, a, d) &
$\forall_{t \in \text{targets}}$[~closer (attacker, a, d, t) ∨
bottleneck (attacker, a, d, t)]}

In figure 3, h3 is a bottleneck defense because the white king on h3 is holding a square which is just as many black moves from d5 as the black king on h5. Also, white's king is between the black king and the target on d5 since the distance between the two kings is only two whereas the black king is four moves from d5.

This definition of bottleneck defenses is somewhat more heuristic than many others in this paper. What sorts of errors might result and how might they be corrected? When the attacking king is very close to the target it is impossible for the defender to be in between the attacking king and the target. At large distances, though, our bottleneck definition can suspect some inadequate defenses. For example, let white be the attacker. Even though c8 does not defend a8, the heuristic given above labels c8 as a possible bottleneck defense to a8. The triangulation criterion will weed c8 out because black has no defense to white's straight forward threat a8-a7-a6-a5.

## Counter-Attack Defenses

A much more serious restriction with the above algorithm is that is does not yet consider counter attack defenses.

Consider the problem below with the white king on f5 and the black one on f7. If white races from f5 after the black pawn on a5 he will win black's pawn.

Consequently, f7 is not in the defense set

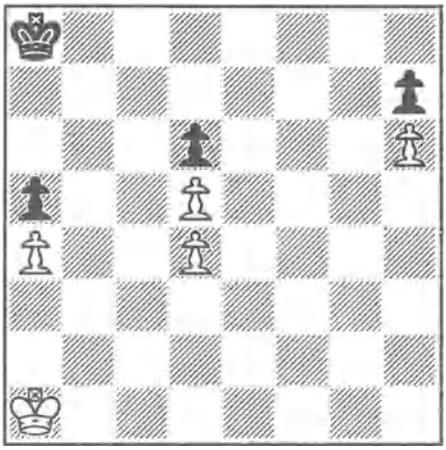


*Figure 4 drawn (either side to move) [Averbakh (1974), problem #673]*

of f5. The problem is that if white should attempt to capture the a5 pawn, black could launch a counter attack on white's h6 pawn, winning the game.

These counter attacks are yet another alternative to the passive defenses already considered when creating defense sets. A square is a counter attack defense if capturing one of the attacker's targets allows the defender to queen a pawn in at most one move after the attacker will queen one of his. In other words, this covers the case where the kings race off to different sides of the board. (It does, however, avoid coplications of evaluating the queen endgames that might result.)

dset (attacker, a) ≡
{d | legal (attacker, a, d) &
$\forall_{t \in \text{targets}}$[~closer (attacker,a, d, t) ∨
bottleneck (attacker, a, d, t) ∨
counter-attack (attacker, a, d, t)]}

It is rather difficult to give a precise definition of a counter attack defense because it is implemented with a rather large number of heuristics. Basically, there is a counting procedure which computes the number of moves to capture a pawn target which is blocking a friendly pawn, move out of the way of the friendly pawn, and queen that pawn.

In the figure 3 with the white king on f5 and black's on f7, white can capture black's a5 pawn in 5 moves and queen 5 moves later. Black's defense is that he can capture

white's h6 pawn in2 and queen in 6 more. With the counter attack defenses, not only does f7 defend f5 but so do a4 and a host of other squares well inside white territory. Although the defense sets are no longer so attractively small, the core of usually obtainable squares is only slightly larger than before. The fact is that many squares well inside of white's territory are perfectly good defenses. We will discuss methods to eliminate unreachable defenses in the section called Forward Pruning.

An additional problem with counter attack defenses is that triangulation, as previously defined, tends to remove the counter attack defenses. Consider the above problem with the white king on c4 and black king on h6. Having captured the pawn on h6, black has obtained his target position. To win the game, though, black will have to play some pawn moves to queen his h7 pawn. The problem is that triangulation assumes that all good positions are connected with only king moves. We can see that although h6 defends c4, there is a square adjacent to c4, namely b5, which cannot be defended by any square adjacent to h6. This shows that the counter attack defenses, unlike passive defenses, are discontinuous. Once h6 has been captured, the black king cannot make any more progress toward a white target. This problem has been solved in the current implementation by preventing triangulation from removing a counter attack defense from a defense set if the defending square is closer to the attack target than the attacking king. Not all counter attack defenses are protected from triangulation, only those where the defending king is guaranteed to reach the counter attack target no matter what the attacker does. In our example (figure 4), f7 could be removed from the defense set of f5 (both f7 and f5 are 2 from h6) but it isn't because g6 cannot be removed from the defense set of e4 (g6 to h6 is 1 but e4 to h6 is 3.) This is the first case where king moves and pawn moves are coupled.

b5 {h6}
f5 {e7, f7, h6, a4, b4, a3, b3}
g5 {e7, f7, h6, a4, b4, b3, a3}
c4 {b6, g6, h6, a4}
e4 {d8, e8, f8, g8, d7, e7, f7, f6, g6, h6, g5, h5, a5, b4, c4, g4, a3, b3. a2, b2, c2}
b3 {a7, c7, f7, f6, h6, d5, f5, h5, g5, d4, e4, f4, g4, h4, d3, e3}
c3 {b7, f7, f6, g6, h6, d5, f5, g5, h5, a4, e4, f4, g4, h4, a3, e3}
d3 {c7, f7, f6, g6, h6, d5, f5, g5, h5, a4, b4, f4, g4, h4, a3, b3, a2, b2}
b2 {a8, b8, c8, e8, f8, g8, e7, f7, f6, g6, h6, d5, f5, g5, h5, a4, b4, c4, d4, e4, f5, g4, h4, d3, e3, f3, g3, h3, d2, e2, f2, d1}
c2 {b8, c8, e8, f8, g8, e7, f7, f6, g6, h6, f5, g5, h5, a4, b4, c4, d4, e4, f4, g4, h4, a3, e3, f3, g3, h3, a2}
...

Apparently the defense sets are not nearly as constrained as in problem Fine (1941) # 70 because black can defend the king side entrance from e7 instead of g6. Consequently there isn't the long propagating constraint from the king side. Notice that the defense set of b2 and that of c2 overlap on two adjacent squares, b8 and c8, a strong indication of a drawn position. To draw, all black has to do is get to either b8 or c8 when white is on either b2 or c2. Once he has reached one of these squares, he can move between for ever. It is the attacker's responsibility to break an infinite repetition. It is easy to verify that the position is indeed drawn by completing the rest of this map.

**Stalemate defenses**

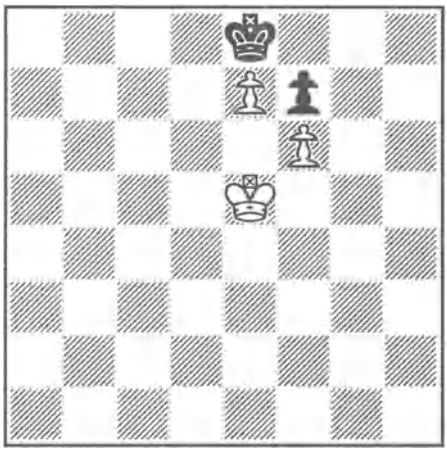


*Figure 5 White cannot make progress by moving toward the queenside. (Fine, 1941, problem 29)*

If white moves to d6, c7 or c8 black will be stalemated, a rather sufficient defense.

Obviously, the first criterion must know what stalemate is and consider stalemate as yet another alternative to the closeness, bottleneck, and counter attack defenses. It is worth noting that these stalemate defenses can have more global effects because of the triangulation criterion.

### Pawn moves

Now that we have discussed king co-ordination, it is time to consider how pawn moves might effect the kings. Stalemate depends upon a complete lack of pawn moves. Also, the reason king moves are discontinuous in counter attacks is that after the defending king has captured the target, the defender must finish the counter attack with pawn moves. So, in fact, we have discussed some pawn moves. In this section, though, we would like to explicitly mention some other important themes centred about pawns.

### Pawn Moves to Gain Tempo[5]

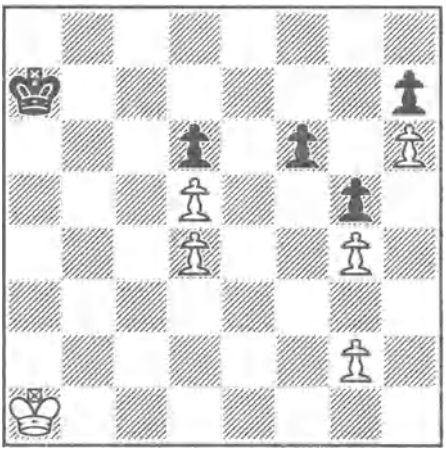


*Figure 6 white to move wins; black to move draws [Averbakh (1974), problem #676]*

When the attacker has an extra tempo, he may consume the tempo at an optimal time, forcing the defender to play. Intuitively, the tempo is an option to pass. The solution is to build one map of co-ordinate squares assuming there is no tempo. Construct another map using the constraint that for any square $a_1$ in a defense set A, there must be an $a_2$ in A such that $a_1$ is adjacent to $a_2$. After applying triangulation to the new map of co-ordinate squares, we have the map which accounts for the tempo. To illustrate that this approach does in fact work, some selected results are shown below.

f5 { e7 }
a4 { b6, d5, c4, d4, g4, g2 }
b4 { a6, d5, d4, g4, g2 }
c4 { b6,g4,g2 }
e4 { d8, d7, g4, g2 }
a3 { b7, d5, c4, d4, e4, f4, g4, h4, c3, d3, e3, g3, f2, g2, f1, h1 }
b3 { a7, d4, f4, g4, h4, d3, f2, h2, f1, g1, h1 }
c3 { b7, f4, g4, h4, c3, g3, f2, g2, h2, f1, g1, h1 }
d3 { c7, f4, h4, g3, f2, g2, h2, f1, g1, h1 }
a2 { b8, d5, a4, b4, c4, d4, e4, f4, g4, h4, c3, d3, e3, g3, c2, d2, e2, f2, g2, h2, e1, f1, g1, h1 }
b2 { a8, b4, c4, d4, e4, f4, g4, h4, d3, ... }
c2 { b8, b4 ... }
d2 { c8, d4, e4, ... }
a1 { b7, a5, b5, d5, b4, c4, d4, ... }
b1 { a7, b5, d5, b4, c4, d4, ... }
c1 { b7, d5, c4, ... }
d1 { c7, d5, ... }
...

*We can see that white can win by ♔a1-b1 and black can draw by ♔a7-b7.*

### Passed Pawns

A pawn p is passed if the opponent does not have a pawn on a rank, between p and the queening square on the same file as p or on an adjacent file. Both a7 and e5 are passed in Figure 7. Clearly, these pawns must affect the co-ordinate squares since, if one king should leave the *square of a passed pawn*[6], the region where the king can catch the pawn running toward the last

[5] Tempo is yet another difficult term to define. In this case, white has one more pawn move than black so he may in effect, pass once instead of moving his king.

[6] The square of a passed pawn is the region where the defending king can catch the pawn before it promotes. It is called a square because, unless there are external obstacles, the region tends to be square. A very good approximation is that the defending king must be closer to either the queening square or the pawn than the attacking king and the defending king must be as close to the queening square as the pawn. The actual implementation considers a few other details such as the fact that pawns can move two squares from the second rank.

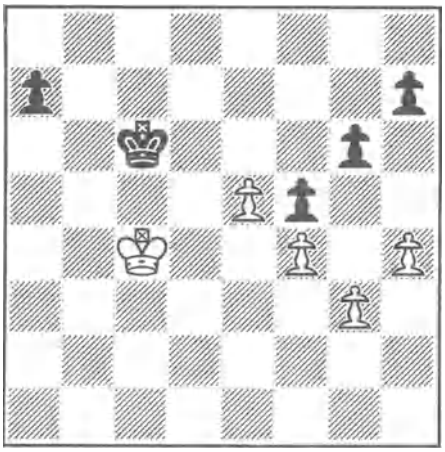


*Figure 7 White to move and win [Fine (1941), problem #88]*

rank, then the pawn will promote into a queen. Also, these pawns affect the targets. Suppose the white king is on f7. There is no place the defending king can be and prevent white from queening in only three pawn moves.

First, let's consider the targets. If a pawn is passed, then the squares on adjacent files and two ranks ahead of the passed pawn are targets. If the passed pawn is on the seventh, then the targets are only one rank ahead of the pawn. In our example (figure 7), we add f7 and d7 to the list of white's targets and b5 to black's targets.

The decision to make b5 a target has some problems. If black can occupy b5, then he is guaranteed of making progress with his a7 pawn. However, he is not guaranteed of queening the a7 pawn in a fixed time period. Similarly, a7 is not as valuable a white target as the others since reaching it does not obviously guarantee a pawn promotion. The notion of target needs some work.

Neverthless, while we are on the subject of targets, it is clear that some squares occupied by a pawn are not targets. In particular, if the pawn is protected by another pawn, then there is no way to win the target. It is a waste of effort to include such protected pawns in the target list. Similarly, if a pawn is protected by a sufficient threat, such as a passed pawn threatening to queen extremely quickly, then the pawn is also not a target. For example, g3 is not a black target since white would queen his pawn long before black could do anything from g3. In this problem, black has no targets and the white targets are a7, d7, and f7.

To consider the effect of the square of passed pawns we have to be more precise about what a target means. Once a target has been captured, the attacker will queen a pawn in a certain number of additional moves. For example, white will queen three moves after he reaches the target on f7. Sometimes capturing a target does not clearly lead to the queening of a pawn. In this case, the target is associated with a large time. The target a7 is a good example of a target which does not explicitly block a friendly pawn. However, since capturing it does free something and does eventually lead to a pawn promotion, we consider the time before such a promotion to be large but not infinite.

Computing the number of moves before queening is also part of the counter attack calculation. As in the counter attack calculation, we have to compute the time for the attacker to queen a pawn by force and the time for the defender's threat. If a king is outside the square of a passed pawn, then the time to queen an attacking pawn has an upper bound of the time to queen the passed pawn. It turns out that this is sufficient knowledge to understand problem 88.

## Deflection Sacrifices

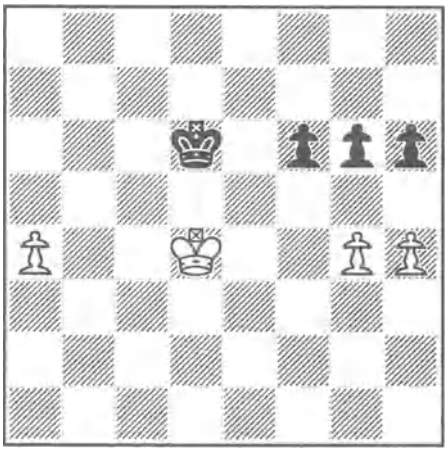


*Figure 8 White to move and win [Fine (1941), problem #62]*

White's winning idea is to push his out-

side passed a4 pawn forcing black's king over to the queen side. With black's king out of play, white can easily pick up black's king side pawns.

How might this idea fit into our way of thinking? Whenever the attacker has a passed pawn, the defending king can be forced over to catch it. We can calculate exactly which square the defending king will be forced to. The pawn, before being captured, will advance as many moves as there are files between the pawn and the defending king. If the defending king cannot defend the position from where he captures the passed pawn, then the pawn sacrifice works. In other words, the defending king, after capturing the sacrificed pawn, must have at least a plausible defense (first criterion) in order to defend against the deflection sacrifice.

**Potentially Passed Pawns**

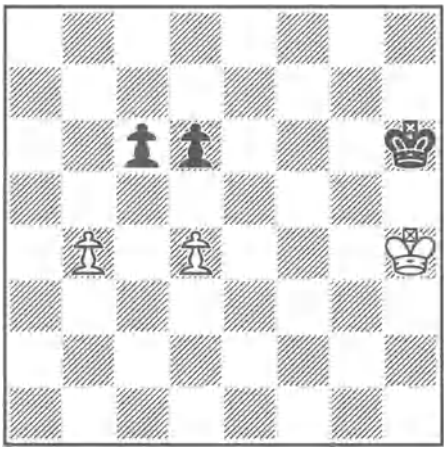


*Figure 9 White to move and win [Fine (1941), problem #51]*

White's winning idea is that his b4 pawn is potentially passed and so far outside that black's king cannot stop it. The difficulty of this problem is realizing that the pawn on b4 is potentially passed, that there is a series of moves that makes b4 passed by force.

One solution is a branching tree of pawn moves. Since there are rarely many legal pawn moves, the tree is sufficiently bounded. Unfortunately, although the tree correctly finds the potentially passed pawns, it does not tell us anything about possible king intervention. For example, if the white king were on h8, then the pawn sacrifice leading to the potential passed b4 pawn would not work. The white king must be in the square of d5 to win this problem.

An important observation is that there are not very many different ways to create a passed pawn. The solution taken here is to model the different situations with a set of patterns. For each pattern, there are associated experts which can compute the effects of different king positions. Should a pattern lead to an unstoppable pawn, the expert will return the number of pawn moves before the pawn queens. Unless the defender can generate a counter attack before the pawn queens, he will lose.

What should these patterns look like? We would like to exploit as many symmetries as possible in order to limit the number of patterns. The left-right symmetry can be easily exploited. Much more importantly, it is possible to split the pawns into *pawn clusters*, independent groups of pawns. The reasoning behind pawn clusters is that pawn moves are extremely local; pawn moves in one cluster do not affect other clusters.

**Pawn Cluster**

A pawn cluster consists of two sets, one of attacking pawns and one of defending pawns. The attacking pawns are separated by no more than one file. If there is a file without an attacking pawn, then that file must contain a defending pawn. Otherwise there would be two pawn clusters. The defending pawns are all pawns on the same file or a file adjacent to an attacking pawn.

The most important key on a pawn is the number of attacking pawns and the number of defending pawns in a cluster. Other features which key certain experts are the ranks of the pawns, backward pawns, doubled pawns, pawns split by a single file, locked pawns, protected pawns, and half passed.

The above problem, Fine #51, contains a single pawn cluster. There is an expert which classifies the cluster into the two on two split attacking pawn category. An expert in this category knows about the "split pawn sac" which forces a pawn promotion in four moves. Two more difficult examples, Fine #61 and #82, are shown

below in the evaluation section.

Most clusters of three or fewer attacking pawns have been worked out. More work is necessary to consider the larger clusters.

**Co-ordination Compared with a Branching Tree.**

In this section we wish to compare the co-ordinate squares procedure as discussed above with the more standard depth first branching tree procedure exemplified by such machines as PEASANT. This section is not intended to determine which approach is the best. We merely plan to explore their relative merits and trade-offs. It is our belief that co-ordinate squares, as a generalization of Waltz's filter, is important to Artificial Intelligence, even if there is a better approach to solving the same problems.

It will be assumed that the branching tree strategy employs a hash table to remember all the nodes previously visited. (In fact, part of the problem with PEASANT is that it, for some inexplainable reason, removes many entries from the hash table.) In many ways, the hash table is analogous to the co-ordinate squares map. Both data structures list every position considered associated with a value for the position. In a hash table representation the value is whatever was passed back up the tree whereas the value in the co-ordinate squares representation is whether the position is an adequate defense. Just as we have considered the co-ordinate squares map to be the understanding of a chess position, the hash table can be considered the results of a branching tree search. It is possible to determine easily the best move using either data structure. Furthermore, there is an argument that both searches require equal time because the limiting factor in all of these endgames is the small number of distinct positions resulting after an arbitrary series of moves. If both searches will eventually find all of these distinct positions then it really doesn't matter in which order they are searched.

Although it is very difficult to refute the above argument, the work here is based upon the intuitive belief that in a practical situation a branching tree will explode given half a chance. The co-ordinate squares approach restricts the number of positions to be considered before starting whereas a branching tree discovers that the number of positions is limited only after the hash table contains all of these positions. Restricting the number before the search has the advantage of preventing any chance of an explosive search. However, it may suffer by failing to consider something important.

Perhaps the greatest advantage with the co-ordinate squares approach is that the order of evaluating each node has not yet been specified. In a branching tree algorithm, the order must be depth first in order to employ alpha-beta pruning. In the co-ordinate squares implementation, triangulation considers the smallest defense set first. In the first example, Fine #70, the defense sets of b5, g5, d3 and c4 would be considered long before the defense sets of a2, b1, and b2.

This, though, leads to a serious drawback, co-ordinate squares cannot use a number of pruning techniques used in branching trees such as alpha-beta. Since the number of positions is sufficiently limited, failing to prune a number of these positions means that the process will take longer, not that it will explode.

A much more minor difference is that the map of co-ordinate squares needs less space than a hash table used in a branching tree strategy because the hash table must contain, in addition to the information in the co-ordinate squares map, a token indicating whether or not a particular node is in the hash table. This token is implicitly contained within the co-ordinate squares map since the map is complete. In addition, not having to check the token represents quite a saving if one considers the number of hash matches, different series of moves resulting in the same position, there are in these problems.

Since the co-ordinate squares map is complete, we don't have to recalculate the entire map after each move is played. No matter what move is played in the game, the new position must have been considered in the co-ordinate squares map. The machine can find the co-ordinate moves almost immediately from the old co-ordinate squares map. In a tournament situation, this machine would take a long time as soon as the king and pawn endgame was entered and every move afterward would be played extremely quickly. Most good human chess players spend a relatively long time planning and then play

a series of moves very quickly.

Saving a co-ordinate squares map is analogous to saving a hash table, which is rarely done in branching tree strategies because the number of nodes, in general, will be too large. The number of nodes is sufficiently small in this work partly because of the nature of king and pawn endgames and partly because the algorithm described here went to great lengths to limit the number of nodes to the different ways to place the kings.

Another advantage of co-ordinate squares is that the triangulation calculation can be optimized to run much faster than the analogous branching operation. For example, the defense sets can be represented as a 64 long bit string. The inner loop of the triangulation calculation on two sets A and B could be implemented by generating the set of squares adjacent to the elements of set A and intersecting the result with set B. Using this representation, intersection requires one AND operation. Similarly, the generation of the adjacent squares can be implemented with a fair amount of parallelism[7]. On the other hand, it is considerably more difficult to implement parallelism in a forward reasoning strategy, using traditional computer hardware.

The most serious disadvantage with co-ordinate squares is that the number of terminal nodes must be extremely limited and that each one of these must be superficially considered by the first criterion. A branching tree can avoid considering unobtainable defenses with pruning techniques such as alpha-beta. To restrict the number of nodes to the different ways of placing the kings (~ 4000) so that co-ordinate squares would be practical, the solutions to the above problems employed a number of domain specific tricks, such as tempo and counter attacks. A branching tree is more general and may even be faster if most nodes are unobtainable. Clearly there are times when one of the approaches, co-ordinate squares or branching tree, will be more appropriate. The perfect chess machine may have to understand both and understand the trade-offs.

## Forward Pruning

One complaint with co-ordinate squares is that quite a number of unreachable positions are considered. In this section, two heuristics will be discussed to reduce the number of positions that we must consider. Experimentally, these heuristics represent a fifty percent time saving.

The first heuristic is based upon the observation that the defending king must be able to reach a defense before the attacking king can reach the attacking square. Although this idea seems perfectly obvious, some care must be taken. First of all, this heuristic changes the meaning of a defense. Before, a defense set contained all defenses. Now the defense sets contain only those defenses which the defender can reach. It is inportant to consider this subtle point when examining these new defense sets.

A possible source of error may arise if this heuristic removes too many defenses. Even though the defender may not be able to reach some defenses if the attacker moves directly toward an attacking square, the defender may need one of these defenses if the attacker should triangulate, take a longer path so the defender will be on the move. It is very difficult to generate an example because cases are so rare and are usually quite complicated. However, it is necessary to compute just outside the area the defender can actually reach.

A second observation is that the attacker should not consider moving to every attacking square. It is a waste of time for the attacking king to run away from all the targets. We will only consider attacking squares which make progress. That is, if square a is farther from the initial attacker's king position than square b, then a must be as close to some target as b in order to make progress. Again, experimentally it was discovered that this should be relaxed for squares adjacent to the initial placement of the attacking king since the initial position may already be so close to some target that the only way to make progress is to back up temporarily.

Certainly both of these ideas are highly

---

[7] One way to generate the squares adjacent to a defense set A would be to OR the squares adjacent to each row $r_j$ of A. The squares adjacent to a row could be computed by table lookup. Since there are only $2^8$ different ways to label a row of a defense set, the table size would be 256. In this way, we compute a whole row in parallel. It goes without saying, that we could generate the squares adjacent to a set completely in parallel using a special purpose processor.

heuristic; they neither catch all unreachable defenses nor are they completely correct. However, for our purposes, the time savings outweigh the possible errors that may result.

### Cost Analysis

The space requirements of the co-ordinate squares map is rather minimal. We can represent each defense set as a 64 bit string, one bit for each square. If the i-th bit is set then square i is in the defense set. With this notation, the entire co-ordinate squares map requires only $64^2$ bits.

The time costs are somewhat more difficult to analyze because the author does not fully understand the effects of several time saving heuristics such as the queue modification to Waltz's algorithm or the pruning conditions. At any rate, the time cost is mainly due to two factors, the time to apply the first criterion to each position and the time to apply the triangulation criterion.

The first has an upper limit which is linear with the number of positions being considered using the straight forward approach, literally applying a static evaluation to each position. It is possible to do even better if the static evaluator is organized in a more sensible way, keyed on themes instead of positions. For example, we know that that defending king must be in the square of an attacker's passed pawn. (We won't consider counter attacks at this point.) One way to implement this constraint is for the static evaluator to try all placements of both kings, testing to see that the defending king is in the square of the attacker's pawn. A more intelligent approach is to design a procedure which generates those squares within the square of the passed pawn. Only these squares would be considered as possible defenses. This second approach is probably faster than linear with the number of positions being considered.

The time to triangulate also has an upper limit which is constant with the number of positions being considered. The argument, as outlined in the section relating the triangulation criterion to Waltz's algorithm, is that the time is linear with the number of vertices (attacking positions) times the number of labels (defending positions).

In practice, the co-ordinate squares program consumes about 110K words on a PDP-10. Problems require one to ten minutes of cpu time, most typically about five. When the defending king is lost in the initial position according to the criterion, the program halts without considering triangulation. These problems take only one minute, the time to initialize the data structures, find targets, and measure distances from the targets to every square. The most expensive problems have a number of potentially passed pawns. The current implementation checks all the possible sacrifices to create a passed pawn for each position. A major improvement would be for the experts which know about pawn clusters to return the constraints on the two kings. Currently, they have to be called for each different placement of the kings.

At any rate, this performance demonstrates that co-ordination squares is sufficiently efficient for tournament play. Once the co-ordinate squares map has been found, every move for the rest of the game can be placed almost instantly. A ten minute time investment is quite acceptable. Also, the co-ordinate squares map can be drawn during the opponent's move. Of course, since order of magnitude improvements in computer performance can be expected, this ten minute cost could easily be reduced to one in the not too distant future. There are existing machines which are almost an order of magnitude faster than the KA PDP-10 currently being used. (After this document had been completed, we converted the code to run on the new LISP MACHINE currently being developed in the AI lab at MIT. This version solves most problems in less than half minute of real time.)

## Future Work

There is still a considerable amount of work to be done. In this section, we will list a few common themes either completely missing from the existing co-ordinate squares program or desperately in need of refinement. Most of these involve pawn moves since the coupling between king moves and pawn moves is not well understood.

### Shifting Square

Every passed pawn has a square, a region where the defending king could still catch

the pawn before it queens. If the attacker has two passed pawns, though, the two squares are coupled in a very strange way. As the defending king attacks the closer passed pawn, the other one advances r-ducing its own square. If the defending king should continue after the first passed pawn, he may end up outside the other one's pawn square. If the defending king should attack the second passed pawn, then the first can advance thus protecting the first. Some simple cases of this have been implemented.

### Pawn Moves which Change the Critical Squares

*Figure 10 White to move and win [Fine (1941), problem # 58]*

White can win by pushing his d4 pawn. If black trades pawns, it is easy to show (with techniques developed in this paper) that black is lost. Should black retreat his king to d7, white can trade pawns himself. Using zugzwang, white can occupy d5. Once on d5, white can advance the c4 pawns to c5. After black trades d6 for c5, white has a won king and single pawn ending.

The techniques developed in this paper fail to understand a number of the ideas involved in the above solution. First of all, when a pawn moves it attacks different squares. Consequently, some squares the kings used to be allowed on are now off limits and some squares that had been off are now legal. Also, moving the pawns can move some targets. For example, after 1 d4-d5+ c6×d5+ the target on c6 is gone.

What is missing is an understanding of how pawn moves can change targets and co-ordinate squares. Since there are rarely more than a few pawn moves and even fewer are sensible and fewer yet can change critical squares, it should be possible to identify these rare but important pawn moves. However, these pawn moves are the most difficult area for future research.

### Single Pawn Endings

Another reason that this work does not solve the above problem is that the algorithm does not know which trivial single pawn endings are won and which are drawn. There is no machinery to make the final deduction, that after black trades d6 for c5, the single pawn ending is won for white. Since these endings are not difficult and have been completely understood by PEASANT [Frey (1977a), chapter 5] and others (Tan, 1972; Perdue et al., 1977), it should not be too difficult to add the lacking knowledge.

### Undermining Pawn Chains

One solution to figure 11 (below), credited to Grigoriev, is to bring the black king back to b7 and advance the a7 pawn, forcing a pawn trade. After the pawn trade, white's d5 pawn, no longer protected by a pawn, can be captured. Again, there is nothing particularly difficult about this theme. Nevertheless, it has not been implemented.

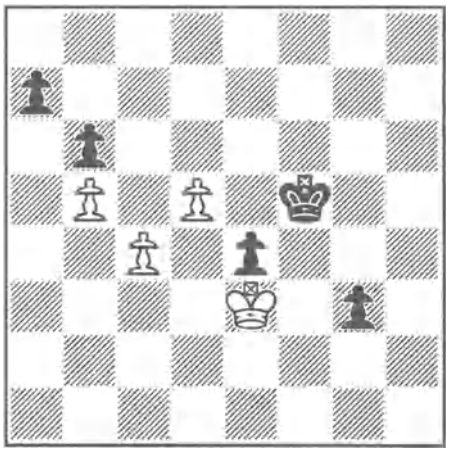


*Figure 11 Black to move and win [Averbakh (1974), problem #710]*

To conclude this section we must feel that there is a considerable amount of work to be done before a machine will play endgames perfectly or even consistently better than good humans. However, the fact that we can even list the most important work to be accomplished is very encouraging. Winston often talks about the "small size of infinity", observing that relatively few themes can accomplish a complicated task which, before serious study, appeared to involve almost an infinite number of themes.

**Evaluation**

The author of PEASANT, Monroe Newborn, chose to evaluate his program with sixteen problems selected from Fine. Although PEASANT finds the correct move in eleven of them, it appears to see the win in only six. A number of the correct moves are forced since all other moves lead to immediate disasters. There is a significant difference, though, between avoiding short range disasters and finding a long range winning theme.

Four of these problems have already been discussed, problems 29, 51, 58, and 70. The first two are well understood by both programs, neither understands problem 58, and problem 70 requires co-ordinate squares. PEASANT is much faster on problem 29 because it is relatively shallow, 11 plies. Even more importantly, since black can occupy only two squares, d7 and e8, alpha-beta cutoffs are much more useful than usual. Co-ordinate squares, as currently implemented, considers quite a number of ridiculous black defenses. On the other hand, PEASANT couldn't solve problem 70 because it requires about 30 plies.

Both figure 12 and figure 13 involve a potential passed pawn queening before the defending black king can catch it. PEASANT, lacking the useful knowledge regarding potential passed pawns, has to carry out the general tree search. It can't find the solution to problem 82 in eight plies since black has some delaying moves on the king side. Our program solves both of these in just about one minute of cpu time.

The solution is found by a pattern match on the queenside pawn cluster in both figure 12 and figure 13. The expert associated with each of these two patterns knows a number of stereo-typical sacrifices to consider. The winning sacrifice in figure

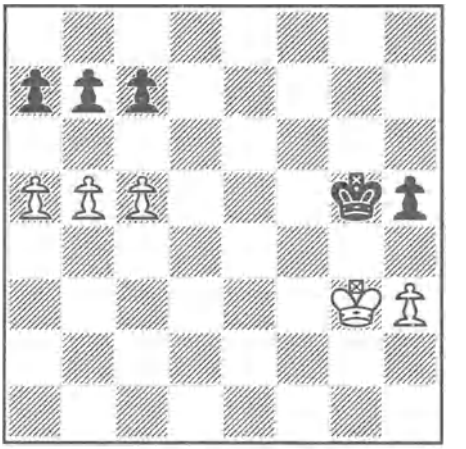


*Figure 12 problem 61; white to move and win*

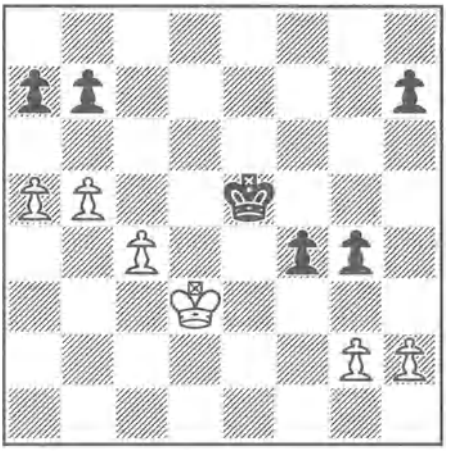


*Figure 13 problem 82; white to move and win.*

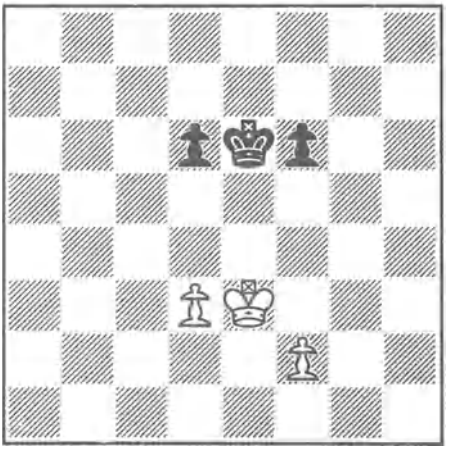


*Figure 14 problem 53; white to move and win.*

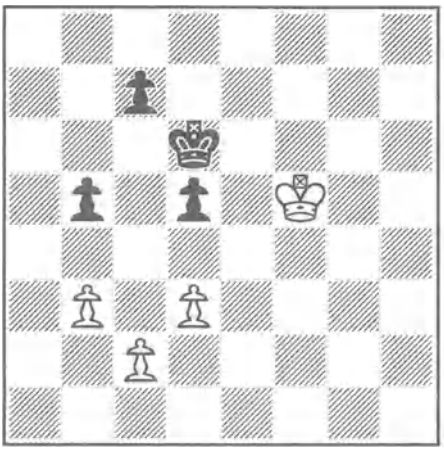


*Figure 15 problem 67; white to move and win.*

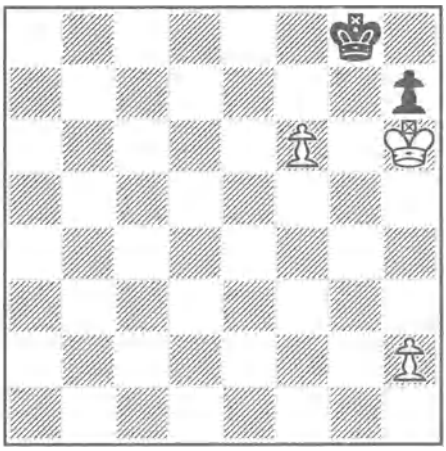


*Figure 16 problem 25; white to move and win*

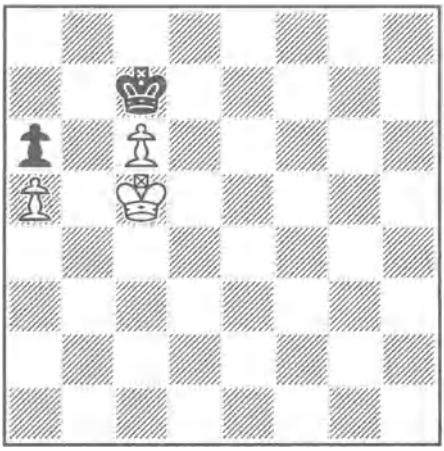


*Figure 17 problem 26; white to move and win*

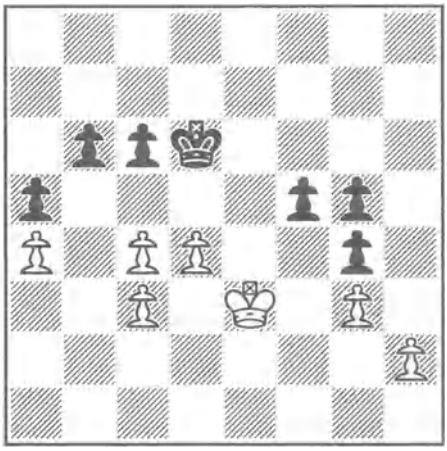


*Figure 18 problem 80; black to move and win*

12 is 1 b5-b6, c7×b6; 2 a5-a6 and white has a winning passed pawn. Similarly, the solution to figure 13 is: 1 c4-c5, Ke5-d5; 2 c5-c6, b7×c6; 3 b5-b6 and white queens.

Neither program can solve problems 53 and 67. PEASANT cannot carry out the 20+ ply search whereas the co-ordinate squares program fails to account for the changing critical squares. Also, the notion that white can decide how many tempos to consume with a pawn on the second rank is not considered by the existing co-ordinate squares program.

Although these problems (25 and 26) look very similar, problem 25 is much harder. Neither program can solve 25; both can find the best move in problem 26. There is some question whether PEASANT can find the win in problem 26, though. Since, in problem 26, all moves immediately lose the c6 pawn except ♔c5-d5, PEASANT does find the winning move. To see that black must lose his pawn involves a better understanding.

The difficulty with problem 25 is that white has to lock the h-pawns at a favourable time. Wherever the h-pawns are locked, the critical squares will be significantly changed. Also, white must realize that his pawn on h2 can advance either one or two, so white can choose how many tempos he wants.

Neither program can find the win in

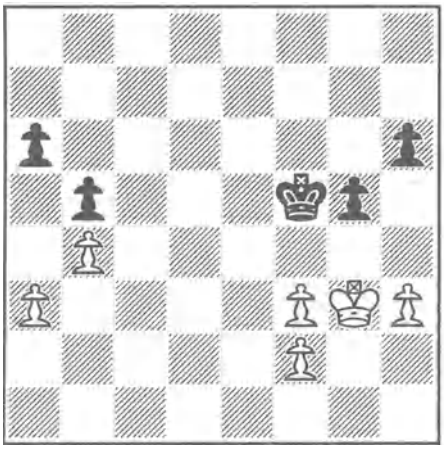


*Figure 19 problem 76; white to move and win*

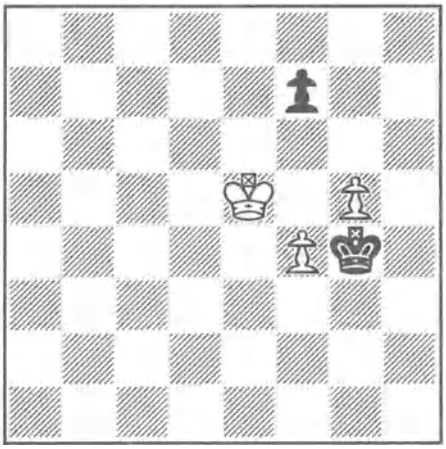


*Figure 20 problem 42; white to move and win*

*Figure 21 problem 66; white to move and win*

*Figure 22 problem 90; white to move and win*

either problem 80 or problem 76. Problem 80 involves a shifting square of two potentially passed pawns, b6 and f5. Unfortunately, since the pawn cluster experts do not know about double pawns, the co-ordinate squares program cannot identify the queenside potentially passed pawn.

The winning idea in problem 76 is that white can sac the f3 pawn, changing the co-ordinate squares so that white eventually wins a black pawn and the game. Again, this is beyond both programs.

The correct move, ♔e5-e4, is forced in problem 42 because everything else loses a pawn immediately. However seeing the win is somewhat more complicated but not too complicated for the co-ordinate squares program.

The win in problem 66 is extremely straight forward, white races after the black pawns. PEASANT, though, does have trouble because there are quite a number of useless pawn moves.

Problem 90 is a hopeless mess; Fine's solution is a mere 37 plies. A number of necessary concepts are missing from the co-ordinate squares program. Most notably, shifting square. There is too much going on in this problem to discuss how a program might solve it.

It is easy to see how white can win the e4 pawn in problem 100A. Unfortunately,

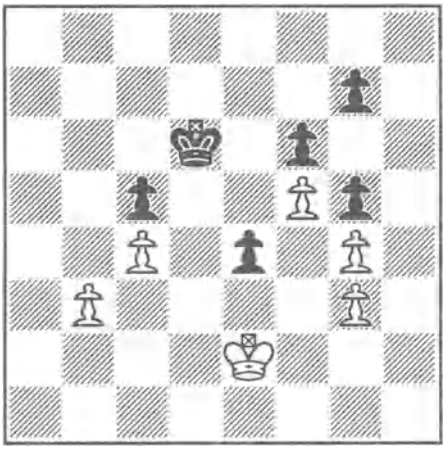


*Figure 23 problem 100A; white to move and win*

co-ordinate squares believes that the game is over once that target is obtained. It happens that it can also see the win after winning the pawn. It really should check that this is indeed the case, though. PEASANT isn't capable of seeing the win after winning the e4 pawn.

## Conclusion

In conclusion, although this work has shown that co-ordinate squares can solve several endgames otherwise requiring 20-30+ plies of analysis, the question remains, what are the limits to the approach? At first, the algorithm appears to be exponential with the number of potential pawn moves. That is, one way to solve problems with pawn moves is to map the co-ordinate squares for any pawn structure that could result. However, if we think about what the pawn moves mean, then we can do considerably better. For example, a tempo, having one more pawn move than the opponent, is an option to pass. We have shown how to incrementally modify the co-ordinate squares map to account for the tempo. Similarly, the effects of a passed pawn can be considered without building separate squares maps for each square the passed pawn could occupy before it queens. It should be possible to state exactly how pawn moves and king moves are coupled.

At first it was believed that the strategy would break down when more pieces are introduced. Even that is not so clear; pieces, especially slow ones such as knights, in many cases must avoid zugzwang by co-ordinating with each other. In fact, Averbakh has written another book about knights (Averbakh, 1977).

The key observation that the strategy depends upon is that all these positions have a very limited number of terminal nodes, distinct positions that could result after an arbitrary number of moves. Since there are so few terminal nodes, it is possible to apply a static evaluation function to each one (the first criterion) and then observe how the nodes are connected (triangulation).

Using the queue improvement to Waltz's algorithm, co-ordinate squares starts with the most constrained nodes and works backwards toward the least constrained. It happens that the constraints tend to be centered near the targets, and that the initial placements of the kings tend to be relatively far from the targets, in the more difficult endgame problems such as Fine #70 (figure 1). Working backward from the target positions to the initial positions, co-ordinate squares is somewhat similar to backward reasoning, just as a depth first tree search from the initial position toward the target positions is forward reasoning. The usefulness of this analogy is in the observation that it is often possible to halve the exponential growth of a search by building backwards from the terminal nodes and forward from the initial position simultaneously. If this could be applied to co-ordinate squares as described above, then the defense sets would not contain a number of unobtainable defenses. For example, in Fine problem #70 (figure 1), a4 is a defense to c4 for black even though there is no way for black to ever reach a4. The section on forward pruning discussed existing heuristics to weed out many unobtainable defenses.

We have designed a system that thinks about the chess domain more as a graph of possible positions than as a tree. Consequently, when the number of positions is small, as it tends to be in king and pawn endgames, the procedure works surprisingly well. In general, as the number of interesting positions grows, the space tends to look more and more like a tree. That is, it becomes less likely that there will be two sensible lines leading to the same position. In a general chess position,

co-ordinate squares is not the optimal representation. Within our restricted endgame domain, which has long been a weak point of chess machines, the co-ordinate squares procedure shows great promise. As a generalization of Waltz's filter, this work may have some implications in other domains besides the chess endgames.

**Acknowledgements**
I wish to thank Kurt VanLehn for an extremely profitable conversation which led to the queue improvement to Waltz's algorithm. Tim Anderson, Kurt VanLehn, and especially my father, Russell Church, were among the large number of very supportive readers. And, of course, I musn't overlook Richard Greenblatt, my adviser.

# 8.2 Goal-Directed Search in Chess End Games

**Jaap van den Herick**

First published as *Delft Progress Report 5*, Department of Mathematics, Delft University of Technology, 1980, pp. 253-279. Reprinted by permission of Delft University of Technology, and the author.

The game of chess has been in use as a testing ground for artificial intelligence research for over thirty years. Recent research into computer chess has focused on endgames. Programming the end game has proved to be surprisingly difficult; even programming correct play in the most elementary end games has turned out to be recalcitrant, if 'correct' is taken to mean: winning, within the rules of the game, whenever possible.

This paper first discusses chess thinking and pattern-based reresentation of knowledge and subsequently examines a model in which pattern-knowledge is combined with conventional depth-first searching. The aim of the model is to investigate whether the use of pattern-knowledge is as important for human chess skill as it is for the chess skill of the machine. To equal the strength of a chess grandmaster with an ELO rating of 2650, a chess program should at least commit no mistake in the elementary end games.

The balance between implemented knowledge and possible searching is determined by the degree in which the playing strategy is exhibited in textbooks, subject to the requirement that the balance struck should be clear to human players. The model is instantiated by an algorithm for the end game King, Knight and Pawn vs. King (KNPK). Examples are given and a comparison between machine play and the thinking of some chess grandmasters is made. Finally, an extension of the model to end games more complex than the one mentioned is made; this additional complexity is felt by human players as well.

## 1. Introduction

Initially, the aim of research in the field of computer chess as a testing ground for artificial intelligence research was to create a chess program playing a game indiscernable from that of a human being. Taking chess as a micro-world, this means that the aim was to construct a chess program that would pass the Turing test (Turing 1950). The idea behind it was that one might thus gain insights into human thinking.

However, chess programmers have reached the Turing milestone *without* simulating human chess thinking. They have developed techniques which enable the program to play an acceptable game, judging the latter by the moves produced by the program.

One of the best indications of having passed the Turing test has been the participation of computer programs as opponents in simultaneous exhibitions of grandmasters. For example, international grandmaster Pfleger did not identify a chess-playing program as one of his opponents during a simultaneous performance (Hamburg, Aug. 1980); after the exhibition he could hardly believe this to have been the case.

In the current state of chess-playing programs two important questions emerge. Firstly, on the practical side: why do programs not play well in the end game? Secondly, on the theoretical side: what has been learnt about human chess thinking from the chess program?; or, complementarily: can we describe the essence of human chess thinking and its relation to chess programs better than before?

## 2. The End Game

The game of chess can be divided into three parts: the opening, the middle game and the end game. Chess programs will (most of the time) survive opening problems by referring to a standard library containing the usual book variations. In the

middle game, too, they are playing quite well by now. But unless they have reached an important material advantage before the end game, accidents may mar their end play. Of course, if a computer program plays against another computer program, the other one is subject to the same accidents, hence it is not predictable which program will win.

In end games, the issue is not computing as such, but handling and trying to reach favourable positions. The highest goal in chess is to mate; however, in a normal ending, before reaching that goal, many intermediate goals (subgoals) have to be attained. It is hard to make these subgoals conceivable to a program if the latter is based on tree searching and uses a uniform evaluation function (as already pointed out by Shannon in the section "Another Type of Strategy" of the first well-known article about computer chess Shannon (1950) and more recently by Berliner (1974) and again by Bratko, Kopec and Michie (1978).

In most endings there are many exceptions, which enormously complicate the strategy the program should follow. Hence, chess programmers have divided the end game into various types of endings according to the ideas of end-game publicists such as Averbach (1979), Chéron (1970-73), Euwe (1940-41), Fine (1941) and others. Programmers have tried to find special strategies for certain configurations.

The end games can be denoted by canonical names, introduced by Michie (Berliner, 1978). In this system the abbreviations of the chess pieces are King (K), Queen (Q), Rook (R), Bishop (B), Knight (N) and Pawn (P). The King and Rook *vs* King ending is then denoted by KRK. More generally, the white pieces on the board are noted, starting with the King and then the black ones, also starting with the King.

Many studies of (elementary) end games have been reported from 1968 onwards. At first, they were mostly based on tree searching: Huberman (1968: KRK, KBBK, KBNK), Tan (1972: KPK; 1974: KBKP; 1977: PAWN-ONLY POSITIONS).

Afterwards, there was a tendency to base them on a representation of knowledge without searching: Bramer (1975: KRK; 1977: KPK), Beal (1977: KPK), Beal et al. (1980: KPK).

Another approach has emerged over the last few years, viz. the construction of data bases: Clarke (1977: KPK, KRK), Thompson (1977/79: KQKR and several other four-piece endings), Michie (1977), Arlazarov and Futer (1977/79: KRPKR, KQP(g7)KQ).

Recent research has concentrated on the design of a computer language AL1 (Advice Language 1) to simplify existing methods for handling patterns and pattern-orientated strategies (Bratko, Kopec and Michie, 1978). By using advice tables, these authors have constructed programs for the KRK and the KRKN endings (Bratko et al., 1980; Kopec et al., 1980).

More complete overviews of studies of several (elementary) endings can be found in other publications (Clarke, 1980; Frey, 1977; Van den Herik, 1980a).

In fact, no chess grandmaster has any problem at all with elementary end games such as KRK, KBBK AND KBNK. He learned how to handle these endings in his youth from a chess book or by seeing them played. However, to the beginner the KBNK ending is hard to acquire, and most general chess programs do not now succeed in mating from an arbitrary position (except if they have a special routine to do so (Kent et al., 1980).

Most of the chess grandmasters are in the same position as the beginner when they have to play the KQP(g7)KQ ending or the KQKR ending (Fenner, 1979; Michie, 1977), though, mostly, they intuite whether the position is a winning one (the pattern is recognized as such), but the (actual) way to win is not known and still has to be discovered (searching according to patterns known).

## 3. The thinking Grandmaster vs. the 'thinking' machine

After reaching the milestone of programming a computer to play a game of chess indiscernable from that played by a human being, chess programmers raised their sights. The new aim was to bring their programs up to the level of grandmaster or even to that of the world champion.

The problems arising out of this new aim are: firstly, although the output of a chess machine looks like the output of a chess player, what do we know about their respective thinking processes? This is important because we want to explain, if possible, the differences in playing strength between an average chess player and a grandmaster in

terms of differences in concepts in which the former is lacking; such concepts might be embodied in *ad hoc* procedures.

Secondly, if chess programs have reached the plying strength of an average chess player by a completely different approach to the game of chess, is it possible to pursue this path or should one look for another route? And if looking for another route is to be recommended, should one start from scratch by again simulating human chess thinking or should one start from the point reached now, adding goals and plans as presumably stored in the brains of a grandmaster?

Considering this problem, let us notice that computers have possibilities human beings do not have – the reverse is, although true, irrelevant to our discussion –, so we must use these possibilities as best we can. This means that searching will remain an important feature in chess programs. Besides, we have to provide additional knowledge to the program, i.e. rules and data, e.g. in the form of sets of rules and/or data bases.

If we define *understanding* as real knowledge of the relation between cause and effect, in which 'cause' is built up from rules and data, then *understanding a chess position* implies the ability to predict the consequences of taking some decision.

As a clue to the difference in playing strength between a grandmaster and an average player, the research of A.D. de Groot (1946, 1975) and R.W. Jongman (1968) is important. Unfortunately, their findings are not directly transferable to programs; their work does not furnish us with goals to be reached by the machine. The main point of the research of De Groot is that the grandmaster "knows" how to handle a certain position and that is why he immediately "sees" the "right" move.

General chess programs do not know anything about special positions, they are just searching for a move. They do not make any effort towards understanding the position. As in every day life, so with chess positions: in order to understand the board of configuration the program needs knowledge. According to the domain chosen, one may add knowledge to the program. This seems very easy, because there are many chess books containing techniques, end-game tricks and examples, especially exceptions to the general rules. The problem is that it turns out to be hard to implement this knowledge in a direct or obvious way.

This paper describes a model for a subdomain of the end game. The model is based on pattern recognition (the knowledge of a grandmaster) and on goal-directed search (the way of thinking of a grandmaster). Before describing the model in detail we shall first pay some attention to the representation of knowledge, especially to that of pattern-knowledge, as suggested by Bramer (1975, 1977).

## 4. Representation of Knowledge

Chess programs with conventional techniques, such as fast tree searching, and using uniform heuristics have proved not to be successful in producing the right move in the end game, as already mentioned in section 2. In 1975, M.A. Bramer produced a new approach to the programming of the KRK end game and in 1977 he applied this approach to the KRK as well as to the KPK ending. Bramer introduced the concept of equivalence classes and did not admit any search at all. A summary of his ideas follows (Bramer, 1975; Bramer, 1977; Bramer et al., 1979; Bramer, 1980).

The problem is to construct an algorithm to find a move for a chosen side (e.g. White) in any position p, for a given end game. The basic-move finding algorithm is then as follows:

( i) generate the set Q of all immediate successors (Black to move) of p;
( ii) find the highest ranked element of Q, say q;
(iii) play the move corresponding to q.

The difficulty is, of course, the ranking, i.e. the ordering of the elements of Q. To achieve step (ii), an implicit ranking is defined on the set Q* of all legal BTM (Black to move) positions for the end game in question. The idea of Bramer is to assign each position belonging to Q to exactly one of a number of disjoint and exhaustive classes, which partition the set Q*. It is intended that each subset of the partition, termed a *class*, should correspond to some significant static feature of the end game as perceived by chessplayers, e.g. 'Black is in check'.

We shall apply a fundamental theorem of equivalence relations: "Let F be an equivalence relation in Q*. Then the quotient set

Q*/F is a partition of Q*." According to this theorem the programmer may list features $F_1$, $F_2$, $F_3$, ... etc. to induce a partition in the above sense. Hence, the positions belonging to one subset are equivalent in the sense that they have one definite feature in common, e.g. 'Black is in check'.

If a position q satisfies the feature $F_i$ we shall define

for position q $F_i \equiv$ true

In general, the feature is a complete predicate function. Because it is essential that a position should belong to one and only one class, it may seem that the predicate functions will be very complicated (e.g. Black is in check *and* black is not checkmated *and* White's Rook can*not* be taken). This means for a position q:

$\forall_j \neq i : F_i \supset \neg F_j$,

to be read as "for all j unequal to i the truth of $F_i$ implies the non-truth of $F_j$."

To avoid this problem, Bramer has given a special role to the order of execution of the tests. For a position q this can be expressed by

$\forall_j < i : F_i \supset \neg F_j$.

Furthermore, looking for a match of a position with a pattern is stopped as soon as a match is found. The advantage of this sequential ordering is illustrated by the following example (adopted from Bramer, 1975):

We have the classes:

1. Black is checkmated;
2. Black is stalemated;
3. Black can immediately capture White's Rook;
4. The Kings are in vertical opposition, with Black in check along the rank.

The position: White: ♔e5, ♖d7; Black: ♔e7 belongs to class 3, *not* to class 4, although superficially one would say that condition 4 holds. Yet it does not: the predicate function or rule of class 4 is to be interrupted as "the Kings are in vertical opposition, with Black in check along the rank *and* Black is *not* checkmated *and* Black is *not* stalemated *and* Black can*not* imediately capture White's Rook."

To effect the closure of the equivalence relation in Q* it is necessary to introduce a residual class with the empty feature $F_n$ such that

$F_n \equiv \forall_j < n : \neg F_j$,

to be read as "the truth of the last feature $F_n$ holds if and only if all the other features $F_j$ are false."

It is clear that several different partitions of Q* are possible, depending on the choice of the features in a given end game. But even grandmasters will sometimes have different or differently ordered lists of features in mind. Often it is impossible to decide which is the better one.

Some criteria for the quality of a program might be:

(a) the program does not succeed in all positions, but in say 95% of all positions (e.g. due to cycling);

(b) the program is win-preserving, but may exceed the fifty-move rule;

(c) the program plays correctly, but may need more moves than necessary;

(d) the program plays optimally, i.e. the program always takes the shortest way to a win, if any.

Other criteria may be:

(e) the maximal acceptable size and/or complexity of the functions;

(f) the minimal acceptable speed of the program.

The highest ranked element of Q, mentioned in step (ii) of Bramer's algorithm is the position with the highest class value. In order to break ties Bramer introduces a number of associated functions, the number of which may vary among classes. It is useful to remark that for positions in the same class the evaluation *functions* are always the same although the values assumed may vary among positions.

Bramer needs 11 equivalence classes in his program for the KRK end game. Later on, it turned out that his program was suboptimal with this number of equivalence classes, because positions were shown to exist which took 17 moves to mate instead of 16 (the theoretical maximum).

Bramer used his approach for solving the KPK ending too. Here the number of classes mounted to 38 (Bramer, 1980).

In actual practice such a method cannot be used for the KNPK ending (cf. Van den Herik, 1980a). It is widely conjectured (Bratko et al., 1978; Bramer, 1980) that this method is bound to fail in all four-piece endings, because the number of equivalence classes would increase beyond reason.

This implies that in end games it is more appropriate to use search as an aid to the implementation of knowledge. The amount of knowledge which has to be implemented will enormously decrease by allowing only

a little searching. As Bramer states (1980) the reverse is true for conventional chess-playing programs which start by searching and use knowledge in order to reduce the amount of search required.

## 5. A model for Representing Pattern-knowledge Combined with Searching

The model to be described below is based on a combination of implementing chess knowledge, represented in the form of equivalence classes, and depth-first tree searching. It represents a step in the direction we have suggested in section 3. More attention will be devoted to 'knowledge' rather than to computing *per se*, because knowledge is the determining factor in the play of a grandmaster.

In an elementary end game, a chess grandmaster knows after a first glance at the board what is involved; he is then able to take the right decision. He considers only a small number of all the possible moves (sometimes only one). So the problem may well be formulated as 'how to construct a program that simulates the chess grandmaster in this respect'. At the moment this is still impossible, because we know too little about human chess thinking, and it is even considered doubtful whether it will ever be possible. A program which has the same chess move output as the grandmaster and which operates with plans, goals and directed search as the grandmaster does would be a major stride forwards.

The combination of pattern-knowledge and search involves points to be discussed:

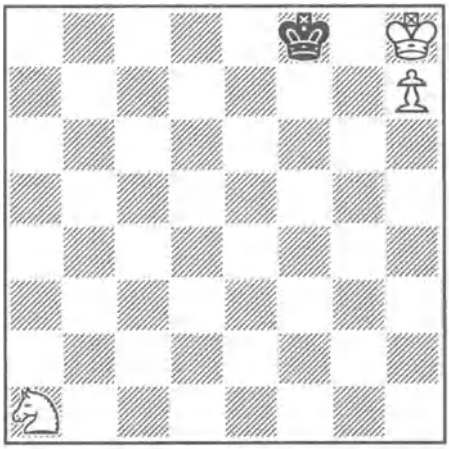
5.1. what triggers the major decisions;
5.2. the depth of the tree;
5.3. the width of the tree;
5.4. the form of the evaluation functions;
5.5. the evaluation of the terminal positions in the tree;
5.6. the method of backing up.

We shall take the KNPK end game as the subdomain of our model, especially with a rook Pawn, because in this four-piece end game the greatest number of exceptions occur and these are hardest for a program to handle correctly. As is usual, we take White to be the strongest side and assume that White is first to move; otherwise we shall explicitly state so.

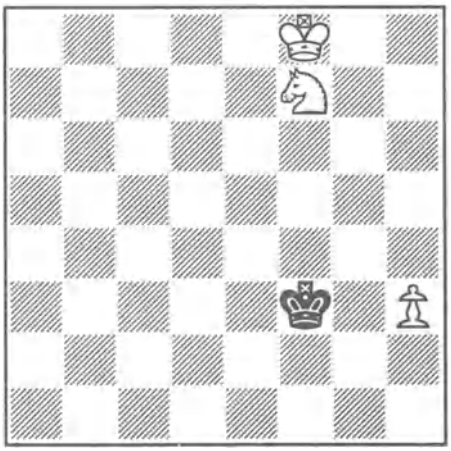
### *5.1. What Triggers the Major Decisions*

As pointed out in section 4, it is impossible to give a complete set of general rules for the successors of a given position applicable to finding immediately the best successor. We may attempt to apply *some* well-defined set of rules. If, in a succeeding position, no rule of this set is applicable, we shall generate all the black moves from this position. We then arrive once more at positions with White to move. For these positions we shall try the applicability of another, but analogous set of rules.

It is preferable not just to evaluate the position with Black to move (as Bramer did), but also after each black move. This implies two distinct sets of evaluation functions. The advantage of this approach is that further searches are inhibited, regardless of their depth. Let us exhibit three examples.



*Ex. 1 White to move (The preceding moves might have been ♖f3×f8† ♔e8×f8.)*



*Ex. 2 A less exceptional position. White to move.*

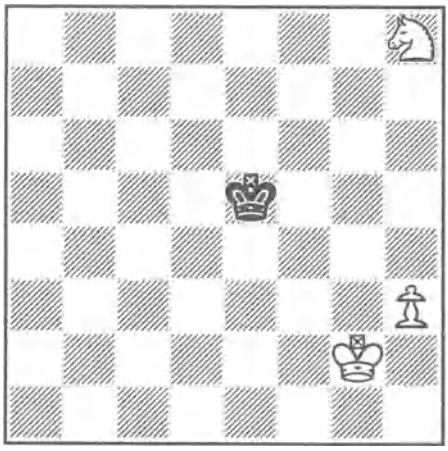
(Ex. 1) Searching should be stopped as soon as possible, because the position is a draw. Searching on would be of no use; so for the position with the black King on f7, it would be a winning position for White.

(Ex.2) After 1. h2-h4? or 1. ♘d7-g5†, the black moves 1. ... ♔f3-f2 or 1 ... ♔f3-g2 etc, are obvious blind alleys. The evaluation function after Black's move should recognise this and prune the tree now, not waiting for the evaluation after White's next move to, be performed.

(Ex.3) After 1. ♔g2-g3 ♔e5-f6 2. ♔g3-g4 ♔f6-g7, the knight is to be lost on the next move.

So White must be alert not to allow his Knight to be imprisoned in the corner, where it is bound to be taken. Searching has to be stopped as soon as it is clear that the Knight is in peril of being imprisoned by the black king (i.e. after 1. ... ♔e5-f6).

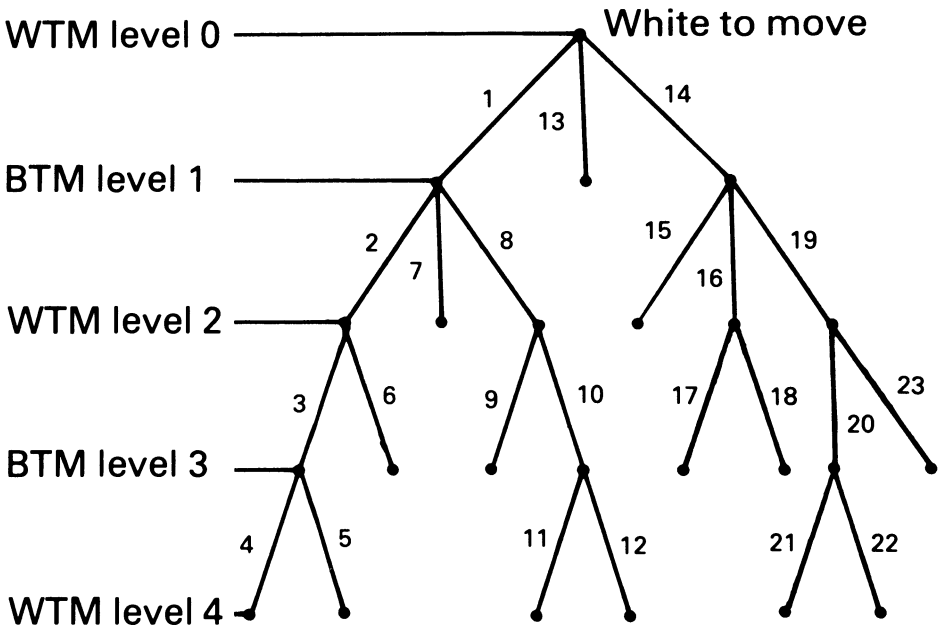
An important remark concerning Ex. 2 is that the black moves have to be pruned while evaluating the move, not while generating it. The idea of generating *only plausible* black moves can be a source of mistakes, because all possible black moves have to be considered in order to ensure a correct white continuation.



*Ex. 3 A more complex example of stopping the search. White to move.*

From the foregoing considerations, supported by these examples, it will be understood that we decided to look for a possible stop in the tree at each level. The aim of our construction will be correct play, in the sense of (c) of section 4.

Our decision leads naturally to a tree structure of the following general shape:




*[The numbering of the branches corresponds to the sequence of traversal of this tree].*

After each white move in this tree, i.e. at the odd levels, the position (node) is evaluated by *stop*. The function *stop* yields a predicate value; it can be seen as the element controlling in a node whether searching should go on (*stop* ≡ false) or not (*stop* ≡ true). An analogous process will be pursued at the even levels, i.e. after a move of Black. As already pointed out, the analogon to *stop*, being distinct from *stop*, will be distinguished from the latter by bearing a different name, say *halt*. In 5.4 we shall discuss the form of these evaluation functions in more detail.

Having reached the depth set as a maximum, (see the final paragraph of subsection 5.2), values have to be assigned to each position so reached (regardless of the truth or otherwise of *stop* or *halt*). On these terminal nodes, the functions *whiteval* and *blackeval* are defined. The function *whiteval* is used in positions where white is to move; the converse obtains for *blackeval*. For details see 5.4.

In summary, our model constructs a tree with a variable but limited depth and a fanout which will vary with the level (ply). Both depend on the given position.

### *5.2. The Depth of the Tree.*

The decision on the depth of searching desired should depend on the limitations imposed. Of course, time is always a limitation; so is the size of the storage available. However, some characteristics *can* be manipulated. As a case in point, our construction involves a tree in which branches of limited depth are encouraged; in 5.3 the widths of (sub) trees will be similarly subject to limitation if possible.

One ideal possibly to be aimed at is: when the program is permitted an n-ply search, it would refrain from searching further whenever a search through fewer plies had yielded sufficient information for a move to be decided upon. Subsection 5.4 will show that this ideal can be approached by proper matching of *stop* and *halt* with the generation of plausible moves for White only.

As against this, it can easily be seen that it is next to impossible to reach this ideal for the simple reason that in the limit this construction would not be a decision made after n−2 or n−1 plies' searching, but after one ply only. This means, in general, the limit of goal-directed search is direct pattern recognition. This would bring us back to our point of theoretical departure. Nor is this as surprising as it would seem, because chess is, in terms of game theory, a finite game with complete information (Von Neumann et al., 1944); the trouble is that to all practical intents and purposes it has to be treated as an infinite game.

In order to handle this pseudo-infinite nature we have to determine the depth of search appropriate to playing the KNPK ending well. In practice, it has turned out that with good evaluation functions a search of three plies is sufficient to make a program play a correct KNPK end game, where 'correct' is win-preserving, avoiding repetitions in a winning position and observing the fifty-move rule (cf. (c) of section 4).

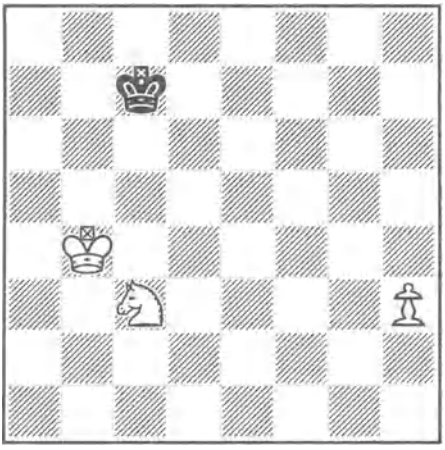
In the present implementation of the program the depth of search is variable and may be set by the user. As will be seen later, the moves found by a three-ply and a six-ply search are not always the same, even though both are winning. In terms of human players, one may say: a six-ply search allows the computer to "understand" more of the position because it looks more deeply, while at three plies the computer "knows" already, after a cursory analysis, what it has to play, i.e., it immediately "sees" the right move.

### *5.3. The Width of the Tree*

Not only a chessplayer, but even a programmer ignorant of chess will consider the searching inept when a computer program follows a path such as 1. ♔b2-a1 2. ♔a1-b1 3. ♔b1-b2, when the King has to go to the King-side of the board. In the case of the KNPK end game it will be clear to the chess player that in the position shown overleaf, for instance, searching along some paths should be eliminated from the outset.

It is superfluous to consider moves like 1. ♔b4-a5, 1. ♔b4-a4 or 1. ♔b4-a3; some expertise allows us to skip 1. ♔b4-b5 and 1. ♔b4-b3 too. Similarly, it is a waste of time to pursue the consequences of 1. ♘c3-a4, 1. ♘c3-b5+, 1. ♘c3-a2 or 1. ♘c3-b1.

In the example given, the possibility to prune these moves is almost self evident. In order to reduce the width of the tree we feel they *should* be pruned. Under what



*White to move*

circumstances to do so constitutes the heart of our problem. As a solution we introduce a heuristic function, called *reject*, which transforms our generator of all legal moves into a generator of all plausible moves for this end game. We believe that a plausible move generator must always be introduced into any elementary end game. Of course, we have to be cautious when introducing such a generator: not every situation is as simple as our paradigm. Especially when the Kings are in opposition, a move which gives some benefit at some succeeding stage should be produced. It would be regrettable if the program would ultimately exclude such a move because it had been rejected as implausible at some earlier stage: missing the winning continuation once is worse than occasionally computing an additional move and its branches.

We shall give part of the function *reject* for the plausible knight moves, in order to give an impression of the plausible move generator. In the interests of clarity we introduce some notation and some conventions. Let us assume that the chess squares are numbered from 11 to 88 (mailbox notation [Turing (1950); Frey (1977a); v.d. Herik (1980); Swets et. al. (1978)] and that the generation of the moves is +8, +19, +21, +12, −8, −19, −21, −12 in that order. The identifier 'wn' contains the position of the white Knight in the mailbox notation, 'bk' the position of the black king. The function 'mod' determines the possible file-position(s) of a piece, 'dis' computes the distance between squares on the board. The counting of the generated moves is done by 'try'.

Then the relevant part of the function can be represented as below, to be read as: "*reject* will have the effect of setting *reject* true if the position of the white Knight modulo 10 is less than 6 *and* the distance between the black King and the white Knight is greater than 1 *and* the current move examined is number 1 (i.e. +8) *or* the current move is number 8 (i.e. −12); else if the position of the white Knight ... " etc..

The part of the function *reject* for the white King is even more complicated due to having to count with the opposition of the black King. For Black one might also be tempted to prune the fanout in the nodes in which Black is to move. But as already mentioned it is better not to skip *any* black move, since the white player has to reckon with all possible counterplays. If Black's move is suicidal (for example moving his King out of the quadrate), then this should be apparent from the evaluation function.

### *5.4. The Form of the Evaluation Functions*

In our program we need four evaluation functions. As stated in 5.1, the function *halt* is similar to *stop*; they are the controlling elements for the searching procedure in the internal nodes. They work at alternate plies. The design of these functions is therefore similar. The function *blackeval* is similarly related to *whiteval*. The latter two, it will be recalled, are the evaluation functions in the terminal nodes.

In section 4, we have already given a description of a partition of positions into equivalence classes. We shall apply this technique to all four of the functions. [We note in passing that *stop* ≡ true also triggers the evaluation of the current position by causing a value to be assigned to *valuepos*,

```
reject = mod (wn,10) < 6 && dis(bk,wn) > 1 && (try == 1 || try == 8)
  || mod (wn,10) <5 && dis(bk,wn) > 2 && (try == 2 || try == 7)
  || wn ÷ 10 < wp ÷ 10 && (try == 6 || try == 7)
  || wn ÷ 10 < wp ÷ 10 && mod(wn,10) < 7 &&
  (try == 5 || try == 6 || try == 7 || try == 8),
```

for which see below.] The difference between *stop* and *whiteval* is that *whiteval always* has to assign a value to a position, whereas *stop* may conclude: since the current position does not match any pattern described in the equivalence classes given, it must belong to the residual class. Belonging to the residual class implies *stop* ≡ false: the search must go on, i.e. the tree must be traversed onward.

The set of equivalence classes of *stop* can be divided into three subsets. The first subset is that set of classes which leads to a win, irrespective of Black's play. The second subset is the set of classes which implies that at least some move(s) of Black will result in a draw. The third subset is the residual class where it is uncertain whether the position is winning or drawn.

When a pattern (i.e. a class feature) matches the current board position, a value has to be assigned to that position. Each has a value of its own (its class value), but there is not sufficient, because the value of that particular position is required. The position value is a class value plus or minus values depending on observed characteristics, e.g. on the distance from the Pawn to the promotion square, the distances WK-WP, WN-WP, BK-WP and so on. Any distance is expressed as the fewest number of moves for a King to go from one square to the other. In the value given by the evaluation function the current level of search also enters. The values assigned to each class have been chosen on heuristic grounds, as has the ordering of the equivalence classes.

The preceding exposition gives rise to the following remarks:

(i) The scoring function, i.e. the actual evaluation of a position, is many-to-one, because there are so many terms for the distance among the pieces that several positions may yield the same value. Moreover, the range of values would otherwise have to be at least as large as the amount of possible positions in the KNPK ending.

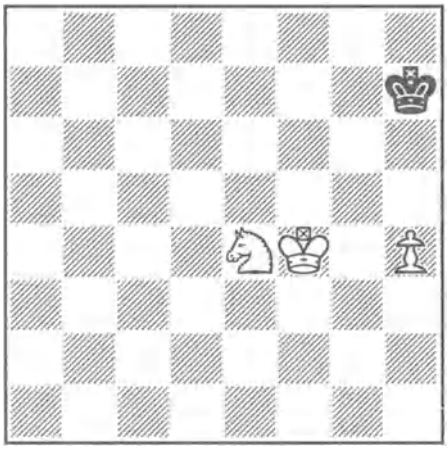
(ii) The values of the actual positions belonging to different classes may change the order of the values of the moves expected when only looking at the *class* values. This is a necessary condition, because otherwise

(a) the Pawn would not move ahead;

(b) the number of equivalence classes would be approximately five times larger due to all the various cases with the Pawn on h2 to h7 having to be distinguished.

We shall illustrate this by the example below:

A favourable pattern to reach is [♔g(i), h(i−1)], where i ∈ {3, 4, 5, 6, 7, 8} (e.g. ♔g5, h4). We temporarily disregard special positions of the Knight (e.g. imprisoned in the corner) and consider:



*White to move.*

The program would play 1 ♔f4-g5, because from the equivalence class [♔g(i), h(i−1)] it obtains a higher value than the position after 1 ♔f4-g4 gets from the equivalence class [♔g(i), h(i)]. But after Black's move, say 1 ... ♔h7-g7, the program has to play the move 2 h4-h5 as an orderly continuation. The latter position belongs to the class [♔g(i), h(i)] and is, only regarding class values, *less favourable than* [♔g(i), h(i−1)]. Of course, one must not hold the position ♔g(i), h(i−1) and invariably start playing the Knight. In our case the program would play 2 h4-h5, because it gets credit (or fewer penalty points) for approaching the back rank. We show this by exhibiting the relevant part of *stop*:

```
if (wk == wp + 9 && wnsafe)
   valuepos=745-depth+2*dis(bk,wp)
            -dis(wn,wk-2)-20*dis(h8,wp)
            -abs(mod(wn, 10)-5)
else ('comment' next equivalence class)
if (wk == wp - 1 && wnsafe)
   valuepos=733-depth+2*dis(bk,wp)
            -dis(wn,wk-2)-20*dis(h8,wp)
            -abs (mod(wn, 10)-5)
```

Evaluations

| | |
|---|---|
| ♔g5 ♘e4 h4 ♔h7 : 669<br>♔g4 ♘e4 h4 ♔h7 : 658 | leading to a preference for ♔f4-g5 |
| ♔g5 ♘e4 h5 ♔g7 : 675<br>♔g5 ♘d6 h4 ♔g7 : 668 | leading to a preference for h4-h5 |

The aim of the program is to bring the Pawn to the back rank. Mostly this would imply queening the Pawn, but sometimes this is not possible because of the ensuing stalemate. Then if necessary we adopt the next lower reachable goal, i.e. to promote the Pawn to a Rook. (For example, in the position White: ♔f5, ♘d5, h7; Black: ♔f7; the promotion 1 h7-h8(♖) or at least 1 h7-h8(♗) is forced, otherwise Black is able to draw.)

Sometimes, Black's response may promote us to a mating capability. While that is not our main goal in the KNPK ending, we shall exploit this whenever it occurs. So with White: ♔g6, ♘e5, h7; Black: ♔h8; the program will, of course, play 1 ♘e5-f7 mate, but with White: ♔g6, ♘f7, h7; Black: ♔f8, the program will play 1 h7-h8 (♕)+ and not 1 ♔g6-f6. Of course, we could incorporate this into the evaluation function *stop*, but we consider this refinement beyond the scope of our goal in the KNPK endgame (i.e. patterns would have to be incorporated which would lead to the series of moves: 1 ♔g6-f6 ♔f8-e8 2 ♔f6-e6 ♔e8-f8 3 h7-h8 = ♕ mate.

In 5.1, it has been pointed out how the search proceeds when *stop* or *halt* yield false for a current position. During this search, it is possible to reach a terminal node, which means we have reached the maximal depth of search allowed. In this case we have to evaluate an actual position by *whiteval* and *blackeval*. These functions have been constructed analogously to *stop* and *halt*, but their classes depend on less detailed conditions. If a position does not belong to one of these few classes, then it clearly belongs to a residual class which will assign a value to the position. An important feature in the function *blackeval* is opposition in all three possible ways.

The number of equivalence classes within the four functions is: 22 for *stop*, 12 for *halt*, 4 for *whiteval* and 7 for *blackeval*. The number of classes in the function *stop can* be lowered preserving correct play, but then searches will be extended, contrarily to the stated objective of our model, viz. to add as much knowledge as may be fit in order to achieve a maximal curtailment of search effort.

### *5.5. The Evaluation of the Terminal Positions in the Tree*

The value assigned to a position does not allow us to derive the position itself and not even always the position's class. The most we can say is that one of the three following statements holds:

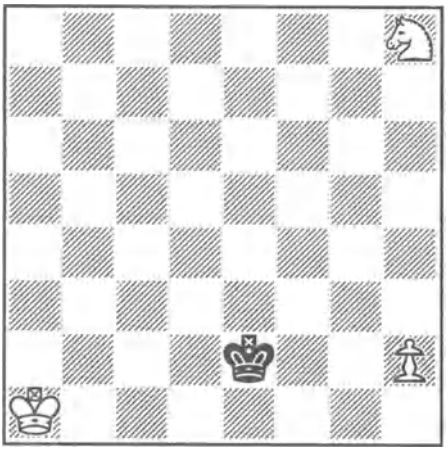
(i) values in the range 501 to 1000 imply a win;

(ii) values in the range 101 to 500 imply the impossibility to decide between a win and a draw;

(iii) values in the range 1 to 100 imply a draw if Black counters properly.

On this scale of values, 1000 is the unique *mated* value.

It may seem rather strange that, in an elementary end game like KNPK, there are positions where the program cannot decide whether it is winning or not. Yet the same may happen to a human player. As an example, we offer the following, to be discussed in more detail in section 7:



*White to move.*

As a rule the human chess player will not proceed by computing very many variants, let alone the full range of variation. In the position above he will reason: whether it is a win or not, pushing the Pawn will make it a draw, because then the black King may successfully go in pursuit of the Pawn; the

white King is too far from his Pawn to protect it, therefore let us try to manoeuvre the Knight closer to the Pawn to provide protection.

This reasoning leads to moves such as 1 ♘h8-g6 or 1 ♘h8-f7, without the player being sure that it is a first move towards victory.

### *5.6. The Method of backing up*

After having made the terminal evaluation of a certain position by means of the appropriate evaluation functions, we have to back up the value in the tree, i.e. to transfer the value from a terminal node to some node nearer the root. Many techniques exist for tree traversal; we have chosen the $\alpha$-$\beta$ minimax technique with some of the refinements taken from (Knuth et al., 1975; Maurer, 1979). When playing the black moves, the chopper technique (Birmingham et al., 1977) has been introduced as a further refinement. With this technique tree searching is inhibited when there is only one possible move. This refinement is not applicable to White because in the KNPK endings White always has a choice of moves.

The order of search for the white moves is: pawn move(s) first, followed by king moves with knight moves last. The reason for this order is that the aim of the program is to bring the Pawn to the back rank. Whether the Pawn can move ahead can most often be read off from the equivalence class. Investigating the king moves after the pawn move(s) is also natural, because in most cases the King has to cooperate with the Pawn in order to let the Pawn move forward.

A possible refinement would have been subsequently to order the black moves by the killer heuristic. For the moment we consider this tree technique beyond our scope. In our case, generating the black countermoves in a special order does not save much time, because a three-ply search will always be sufficient (subsection 5.2).

## 6. Implementation

The program is written for a PDP 11 system and runs on the model 70 of the Department of Computer Science in the Delft University of Technology. It is written in the Language C, programmed in the structured program way, built modularly and uses C functions (Kernighan et al., 1978).

In its present form the program, always playing White, starts with an interactive phase. Here the program requests the starting position, which may be chosen from a library by typing in its address, or by typing in a new position in algebraic notation. The program next asks whether it should stop after each white move, allowing a human to input the moves for Black; if not, the program will play the black moves too. Thereafter answers must be supplied to further questions such as

– Who is to move first?
– What depth of search is desired?
– Whether details of the procedure are to be output?

This interactive form is used for tests and demonstrations. When the program is to be ultimately incorporated as a module in a complete chess-playing program, this interactive phase will be replaced by an initialisation phase where answers to the same question will have to be supplied. Structure diagrams and listings are available on request.

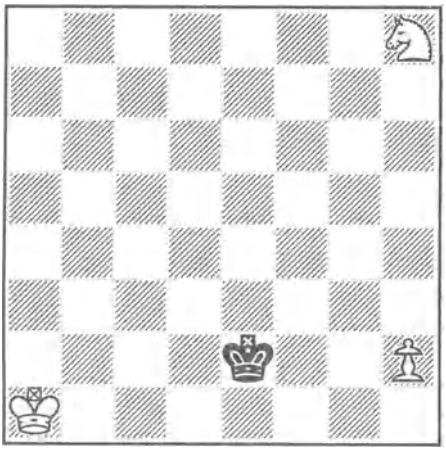
## 7. Grandmaster Performance and Machine Performance

We have used over 300 initial KNPK configurations for testing the program. Thus, the program had to determine its moves in over 2000 different positions. The configuations were taken from actual chess games and from chess books, while some were constructed by chess-playing users. Of course, this does not amount to a *proof* that the program will deal correctly with all positions in the KNPK end game.

We have not yet discussed optimal play for two reasons. Firstly, in our goal-attaining approach we did not focus on optimal play, but rather on correct play, as defined in subsection 5.1. Secondly, there are positions in which it is far from clear which move leads to the optimal path to win. The only feasible test for optimality would be to compare the program's play with a data base to be created for the KNPK end game and which, by its construction, would be guarateed to contain none but optimal plays. We have not done so. The creation of such a data base has been found to be full of pitfalls (Michie, 1977; Bratko et al., 1980). Moreover, testing the program against even a fully proven data

base constitutes a selective, not an exhaustive test and so brings us no nearer to a proof.

Let us go back to the example mentioned in 5.5:



*White to move.*

It is evident that straightforwardly advancing the Pawn 1. h2-h4 will lead to a draw (1. h2-h4 ♔e2-f3 2. h4-h5 ♔f3-g4 3. h5-h6 ♔g4-g5 4. h6-h7 ♔g5-h6 or 1. h2-h4 ♔e2-f3 2. ♘h8-g6 ♔f3-g4 3. ♔a1-b2 ♔g4-h5 and so on).

To most chess players the solution of this problem will not be obvious, yet many will consider 1. ♘h8-g6 or ♘h8-f7 a good move if they are in doubt whether it will win.

Three grandmasters, Euwe, Larsen and Timman were confronted with this position. They voiced their opinions about it. All three suggested 1. ♘h8-g6. One grandmaster was certain that it would be the first move to victory: in a flash of perception he recognized the pattern (subgoal) he wanted to reach. Another grandmaster was also certain that this was the best move, but he could not immediately decide whether it was a winning or a drawn position. The third grandmaster too felt certain about the move, but he first thought the position was drawn. After a few seconds, he recognized the special, stable pattern, belonging to the class [♘f3, h2], and changed his statement to "Maybe it is a winning position." In this context, 'stable' refers to the Knight protecting the Pawn in such a fashion that Black's King cannot capture the Knight either without leaving the quadrate.

Let us now consider the moves produced by our program. It plays 1. ♘h8-g6 at three plies and 1. ♘h8-f7 at six plies followed by very similar continuations. At three plies, with Black's moves as played by the program, the following sequence is produced (numbers in parenthesis are the values yielded by the evaluation functions).

| | | | | |
|---|---|---|---|---|
| 1. | ♘h8-g6 | (264) | ♔e2-f3 | (266) |
| 2. | ♔a1-b2 | (269) | ♔f3-g4 | (266) |
| 3. | ♘g6-e5† | (268) | ♔g4-f4 | (264) |
| 4. | ♘e5-d3† | (280) | ♔f4-f3 | (266) |
| 5. | h2-h3 | (508) | ♔f3-e4 | (510) |
| 6. | ♘d3-f2† | (512) | ♔e4-f5 | (511) |
| 7. | ♔b2-c3 | (513) | ♔f5-e6 | (514) |
| 8. | ♔c3-d4 | (516) | ♔e6-f5 | (515) |
| 9. | ♔d4-e3 | (517) | ♔f5-e6 | (518) |
| 10. | ♔e3-f4 | (520) | ♔e6-f6 | (647) |
| 11. | h3-h4 | (652) | ♔f6-g6 | (653) |
| 12. | ♔f4-g4 | (654) | ♔g6-h6 | (659) |
| 13. | h4-h5 | (672) | ..... | ..... |
| ... | ..... | ..... | ..... | ..... |

At six plies, the following is seen:

| | | | | |
|---|---|---|---|---|
| 1. | ♘h8-f7 | (261) | ♔e2-f3 | (264) |
| 2. | ♘f7-e5† | (262) | ♔f3-f4 | (265) |
| 3. | ♘e5-d3† | (266) | ♔f4-f5 | (491) |
| 4. | ♘d3-f2 | (505) | | |

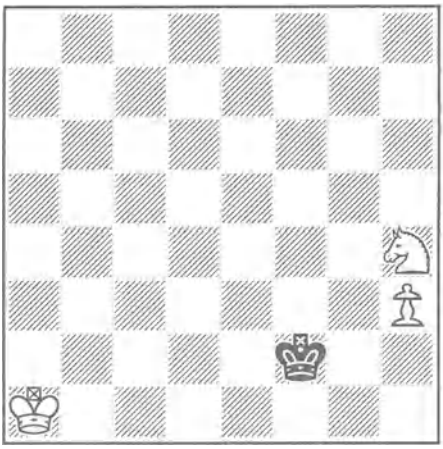
with a similar continuation as before.

It is of some interest that, at eight plies, the program "sees" that it can manoeuvre the King to the Pawn.

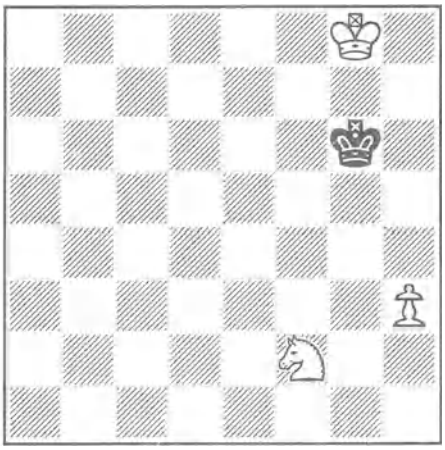
It starts with 1. ♔a1-b2 which is in fact also a good start for a win. The continuation was: 1. ♔a1-b2! ♔e2-f3 2. ♘h8-g6 ♔f3-g4 3. ♘g6-e5+ ♔g4-f4 4. ♘e5-d3+ ♔f4-f3 5. ♘d3-f2 ♔f3-f4 6. h2-h3 ♔f4-g3 7. ♔b2-c3 ♔g3-f4 8.♔c3-d3 ♔f4-f3 9. h3-h4! ♔f3-f4 10. ♔d3-e2 ♔f4-f5 11. ♔e2-f3 ♔f5-g6 12. ♔f3-g4 etc...

Another advantage of the method described in section 5 becomes evident in a simple position, in which the program, by using the equivalence classes, immediately "understands" how to react, without deep computation.

The position below has been submitted to the following programs: the European champion, Master (Birmingham and Kent), the co-champion of the Netherlands, IGM (Van Diepen) and Chess program (Comeau). Recently (Nov. 1980), it has also been submitted to the then current versions of eight different micro-computer chess programs,



*White to move.*



*White to move*

such as Chess Challenger, Sargon and SS III. None of the eleven programs found the way to win. Even at maximal depth of search, all programs produced 1. ♘h4-f5? or 1. ♘h4-g6? or even worse.

The only correct move here is 1. ♘h4-g2. Once found, it is quite obvious that this move is a solution of the position problem; also, it is easy to implement. The point is to understand *why* it is superior. This is because it creates a stable pattern (i.e. [♘g(i), h(i+2)] and allows the King to approach the Pawn. The KNPK computer program plays 1. ♘h4-g2; a three-ply search nicely shows up the benefit of the combination of searching and pattern-recognition: the path 1. ♘h4-g2 ♔f2-g3 2. h2-h4 leads to the stable pattern ♘g2, h4.

The remark that a stable pattern is good seems self-evident, nevertheless it should not be implied that it always leads to *optimal* play. Nor need it, because our aim is *correct* play. A case in point is the configuration diagrammed below.

This configuration shows a stable pattern and therefore belongs to the class [♘f(i), h(i+1)], but White is confronted with Black's King being in vertical opposition. The program's play depends on the depth allowed.

Assume a three-ply search and a good human opponent, the continuation might be

1. ♔g8-f8 ♔g6-f6 2. ♔f8-e8 ♔f6-e6
3. ♔e8-d8 ♔e6-d6 4. ♔d8-c8 ♔d6-c6
5. ♔c8-b8 ♔c6-d6 6. ♔b8-b7 ♔d6-d7
7. ♔b7-b6 ♔d7-d6 8. ♔b6-b5 ♔d6-d5
9. ♔b5-b4 ♔d5-d4. 10. ♔b4-b3 ♔d4-e3
11. ♔b3-c3 and so on.

Observing White's sequence of moves, little doubt remains that the program "knows" how to win. Yet the path to a win might be shortened by temporarily surrendering the stable pattern e.g. 1. ♘f2-g4 ♔g6-g5 2. ♔g8-g7 ♔g5-h4 3. ♘g4-f2. In order to incorporate this improvement into the program, one would have to distinguish yet further subgoals within the main aim, leading to additional equivalence classes. Whether one is tempted to perfect the program by this improvement or not, when all is said and done the fact remains that chess is not a game of moves, but a game of plans and concepts, especially in the end game.

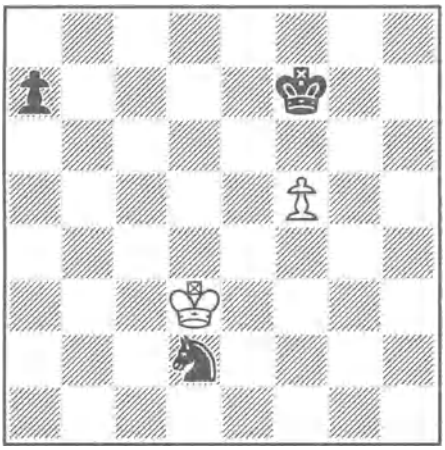
## 8. Generalizations

Looking back at the model of section 5, we recall that only five functions were used (*stop, halt, whiteval, blackeval* and *reject*) to describe the subdomain KNPK ending with a rook Pawn. Compared to the size of a general chess-playing program, the incorporation of these five functions is only a minor extension, which would, however, produce a considerable improvement in quality. A data base of perfect reliability might play optimally, but would have to hold over one million configurations. For a general chess-playing program the other routines, such as output routines, move generators and so on are already available. To attain the level of a master, the program must play end games such as the KNPK faultlessly.

How relevant is the KNPK ending to practice? Human players have learned to

play the endings from text books. In these text books the main strategic lines and the exceptions to these lines can be found. Yet, while playing, one does not always have all textbook knowledge at one's fingertips; equally, it may happen that no specific information has been recorded at all.

Let us give an actual example: in the game Salov-Majorov (Soviet Union, 1979) the following configuration appeared (Orbaan, 1980):



*Black to move*

Black's actual play was 1. ... ♘d2-b3? and after 2. ♔d3-c4 ♘b3-a5+ 3. ♔c4-b5 ♘a5-b7 4. ♔b5-a6, followed by 5. ♔a6×a7, the game finished in a draw. In fact, Black might have won from the start had he recognised the special stable pattern [♘c6, a7]. He then should have played: 1. ... ♘d2-f3 2. ♔d3-c4 ♘f3-e5+ 3. ♔c4-b5 ♔f7-f6 4. ♔b5-a6 ♘e5-c6 and 5. ... a7-a5.

This ending can be regarded as a generalization of the KNPK ending. The themes correspond, except that Black first has to ascertain that the presence of the white Pawn on f5 is irrelevant to the character of the ending. As a program rule, this irrelevance might be expressed thus: create a stable pattern, capture the Pawn and continue by the normal KNPK program. This is one possible generalization of KNPK-like endings.

Yet another generalization of our model is first to subdivide the more difficult endings into phases (i.e. trying to reach specified subgoals in a stated order). Examples of such difficult endings are KNNKP or KRPKBP (e.g. Timman-Velimirović, Rio de Janeiro, 1979). In spite of a great deal of research and experience being extant for the KNNKP ending, no *proven* definitive results seem to be available (Euwe et al., 1977; Bijl, 1980). The Bridier tables, referred to by Chéron (1970-73) and Bijl (1980), provide useful assistance for the construction of the equivalence classes. Employing this aid preserves the essence of our model, viz. the goal-directed search. This is yet another paraphrase of our firm belief that only the *combination* of pattern-knowledge *and* searching will enhance the capabilities of chess programs in end games. *In fine*, two remarks seem in order as a consequence of the efforts described above.

– It will take a vast amount of time to implement, say, Chéron's four volumes on the end game (1970-73). But it will certainly take less time than it took to arrive at the vast amount of chess knowledge now codified in book form, such as his.

– End game books contain erroneous evaluations by experts and grandmasters. Indeed, some computer experiments have already revealed errors and omissions in the published theory of the end game (Michie, 1977; Strohlein, 1970). However, this imperfect knowledge has not prevented grandmasters from achieving ELO ratings of 2650. The conclusion seems inescapable: it does not take invariably perfect play to be a grandmaster!

**Acknowledgements**

The author wishes to thank H.J.M. Lombaers and S.J. Doorman for many valuable suggestions about the substance of this paper. For their careful scrutiny of the English text he is indebted to I.S. Herschberg and R.M. Cooke.

The competent technical assistance of B. Broere and of Ms J.W. Pesch, who did all the typing, is also gladly acknowledged as having been essential to the production of this publication.

# 8.3 Computer Analysis of a Rook Endgame

**V. Arlazarov and A. Futer**

First published in *Machine Intelligence 9* (J. E. Hayes, D. Michie and L. I. Mikulich, editors), Ellis Horwood, Chichester, 1979, pp. 361-371. Reprinted by permission of Ellis Horwood Ltd.

**Abstract**

This paper describes an algorithm for dividing the set of positions in a single-pawn Rook end-game into winning ones for White and drawing. Data arrangement and processing methods are described whereby within reasonable computer time, large (up to about 1500-milion bits) data arrays can be studied; a number of procedures are put forward, such as search for problem symmetry, levels of data arrangement, data restructuring, batch processing of data, and others.

## 1. Introduction

In programming a computer for chess, the end-game is of special significance. The reason is that there are approximately as many alternative moves in an end-game position as in the middle game, namely several dozen. For playing, however, strategies many moves deeper are needed, and so the methods of tree search to a limited depth (Arlazarov et al., 1970; Donskoy, 1974) do not work very well in end-games.

On the other hand, full analysis of certain end-games with a few pieces seems a promising idea. This implies that for each position the result of the game and strategies of the players are determined. A substantial step forward in this direction was the analysis of a single-Pawn Queen end-game (the King, the Queen, and a Pawn against the King and the Queen) with a White Pawn on g7 (Komissarchik et al., 1974).

This paper will discuss a single-Pawn Rook end-game with the Pawn in an arbitrary position.

There, a set of positions is taken up where White has the King, a Pawn, and a Rook against Black's King and Rook (KPR-KR). For each position (of four pieces, a Pawn, and the move) it is required to determine whether it is a win for White, and indicate the best possible move.

Note that the size of such a set of positions is about 1500-million ($64^4 \times 48 \times 2$). If each position takes one bit, then practically all the resources of one of today's large computers will be used up. And each computer operation for one position takes up to one hour (with the computation rate of 500 thousand operations per second). Therefore if the previous algorithm (Komissarchik et al., 1974) were simply transferred into this position it would require about one thousand hours of computer time.

This paper will propose a general procedure of end-game study. Then an algorithm is described whereby the set of positions in a KPR-KR end-game is divided into winning and drawing ones. For each winning position we determined the number of moves to win the position, so that the best move is indicated for each position. The data structuring and processing methods and procedures, which may be of special interest, are described.

## 2. A Study Procedure

Any arbitrary end-game is studied in two stages. First positions are defined which incorporate successful moves leading to another end-game. Then positions are written in sequence that are reducible to the desired one in one move, two moves, etc.

Let us give a more detailed explanation of each stage. White has, for instance, the following moves to other end-games. The pawn can become a Queen or a Rook, or the

Black Rook can be captured, possibly together with promotion. If any of these moves is successful, or leads to a winning position, the original position is assumed won for White in zero steps and is included into the class $W_0$ (zero rank). If there are no such moves in a position or if none leads to a victory, the position is ranked as *NW*, non-ranked White.

In a similar way, Black can take the Pawn or Rook and secure a draw (or a victory, which will not be discussed hereafter). Thus a set $B_0$ is obtained. Positions without such a move are the *NB*, non-ranked Black, class. Now let us take up a set of positions from *NB* which has no moves in *NW*. These are positions all moves from which lead to the complement of *NW*, or they are Black's games lost in one move. These positions will be referred to as the set of the first Black rank, $RB_1$.

Remove $RB_1$ from *NB* and take up $RW_1$ (the first White rank), or those positions of *NW* from which there is a move into $RB_1$. These are White's wins in one move. Remove them from *NC*, etc. Fig. 1 illustrates this ranking procedure. (Here $A^{-1}$ denotes the class of positions which may have at least one move in to the set $A$ and the symbol $\setminus$ denotes set-subtraction).

The procedure ends when the set of positions of the next rank, $RB_i$ or $RW_i$, is empty. The sets *NB* and *NW* at that time are draws with moves of Black or White respectively. This ranking is consistent with Zermelo's approach (1913) if the end positions are those of the sets $W_o$ and $B_o$.

Let us apply this procedure to all the end-games obtained from KPR-KR (referred to as *junior*). The end-games will be taken successively upward, starting with the simplest ones. In this way the positions inferior to the current ones will have been studied. Fig. 2 shows the graph of end-

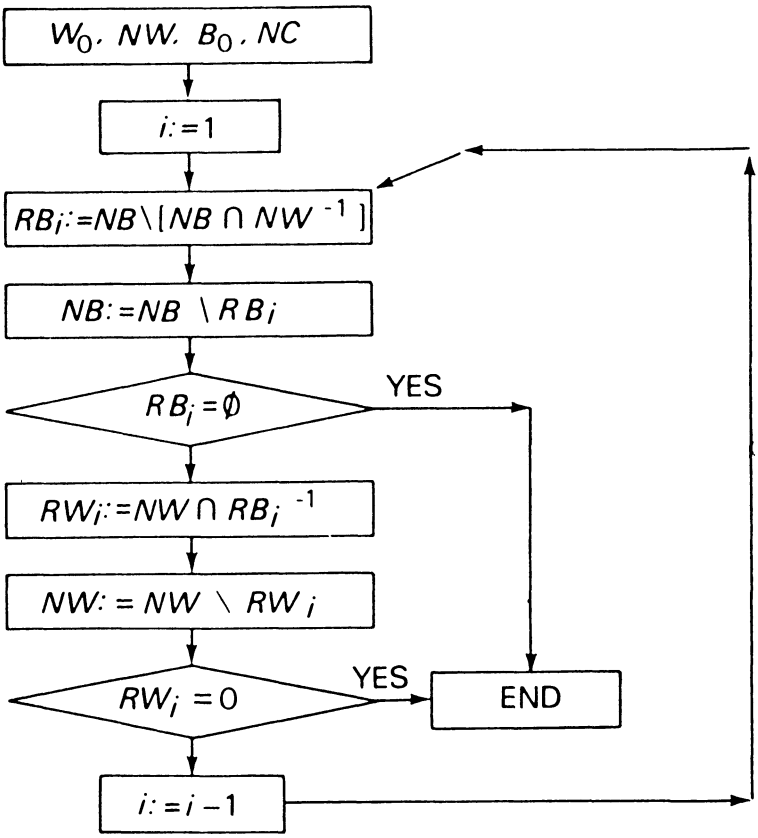



*Figure 1*

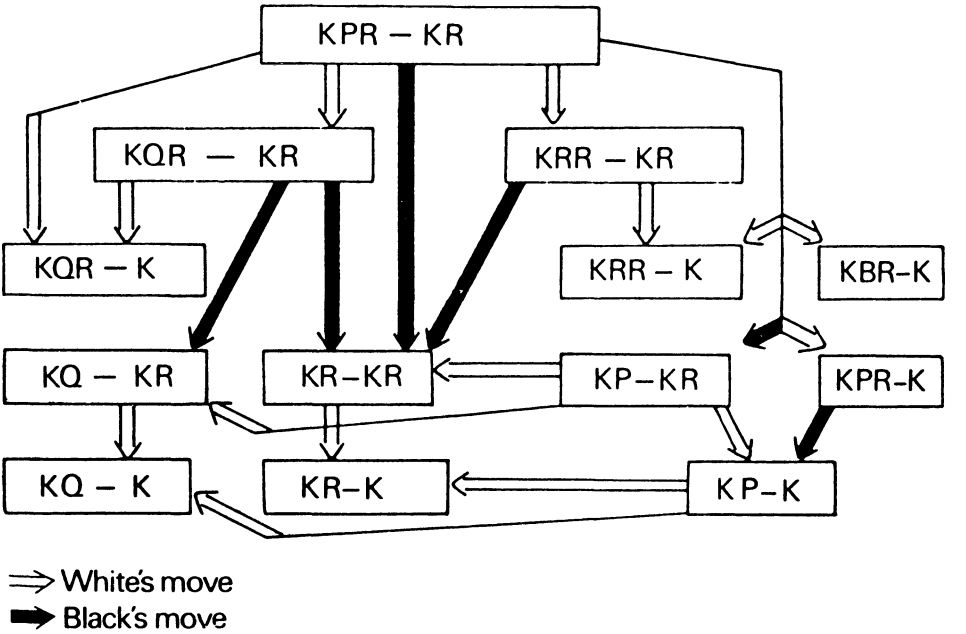



*Figure 2 K = King, P = Pawn, R = Rook, Q = Queen, B = Bishop.*

games.

Each of the junior end-games was studied separately and in different ways. None of them is of interest to chess players. For our problem, however, we needed complete sets of winning positions or drawing positions or both for each of them.

### 3. Reducing a KPR-KR End-Game to a Four-Piece One

Because of the vertical symmetry of the chessboard the KPR-KR ending can be studied only for one half of possible Pawn positions. Besides this, the Pawn can move to another column only by capturing, which would result in another end-game. For these reasons the whole study is four independent analyses of the a, b, c, and d column.

Now let us show that one column can be studied in steps the size of a square, in other words, the problem is reducible to study of a four-piece end-game with the Pawn fixed.

This reduction can be performed by a method analogous to the junior end-game procedure with a move by the Pawn regarded as obtaining a new end-game. In other words, in studying positions with the Pawn at the $j$th row an array can be obtained of $P_j$ losing positions with Black's move. Then, in studying a square of $(j\text{-}1)^{th}$ row, White's positions having a Pawn move into that array are included in the zero rank set.

In this approach, however, the position ranking becomes artificial because the position rank denotes the minimal number of moves until the Pawn is advanced rather than the minimal number of moves to a victory. This kind of ranking is an incentive for White to advance the Pawn even at the risk of heading into a difficult, albeit winning, position. In real games the advance of the Pawn should often be stalled; the larger pieces should secure good positions and then win the game quickly.

To obtain a natural ranking in studying a square of the $j$th row we store the array $P_j$ as a merger of sets of successive Black ranks $P_j = \bigcup_i P_j^i$ where $P_j = RB_i$ are sets of positions with a Black's move lost in $i$ moves (with the Pawn at the $j^{th}$ row). Now in processing a square of the $(j-1)^{th}$ row, positions with White's move of the Pawn in $P_j^i$ are assigned to $RW_i$, the $i$th White rank; Black's losses are entered in the array $P_{j-1} = \bigcup_i P_{i-1}^i$ etc.

This procedure significantly reduces the data storage for one problem to $64^4 \times 2, 32$

million positions, which with one bit allocated per position means about four million bytes.

## 4. Methods of Data Processing and Structuring

Data arrays of the size of about a million bytes can be stored only in magnetic media. As follows from section 2 these arrays need to be repeatedly processed to go through the ranking procedure.

This is why economical access to media is an urgent requirement. We will enumerate the basic methods and procedures whereby this, essentially an exchange problem, can be solved at a practically possible processor rate. A reasonable time of program operation was needed, a mere 60 computer hours.

### *4.1 The data structure*

This requires three levels: distribution of data components into blocks (a block being an exchange unit), the location of units in the array, and allocation of arrays among shelf media. Thanks to the existence and interaction of the first and second levels, the number of accesses to the magnetic media is not high. The third level requires, in particular, the use of different kinds of media, sequential and random access.

The number of the latter is usually limited in each computer configuration, we needed them for performing the data *re-ordering* algorithms. Thus in the program for creating a set of positions of a subsequent rank, transposition algorithms for 4096 × 4096 square matrices were needed. (When a Black rank is obtained, Black chessmen retreat and White chessmen are fixed, and the reverse is true). To reduce the output arrays of the program to a format convenient for computer playing of a Rook end-game, a transposition algorithm was developed for a rectangular matrix of 60 × 16 million. Thanks to transposition, each unit of the arrays was processed the first time it was accessed.

### *4.2 Formats of data arrays*

As is often the case, in our problem two conflicting requirements had to be met, reduction of the amount of the data stored and ease of its use. To resolve this conflict, two chief formats were used, unpacked and packed. In the former the address of a four-piece end-game position in the data array is defined as a linear function of the position of these pieces. In the latter the positions are stored as a list.

### *4.3 Buffering*

In all parts of the program the processor and the exchange channels operated in parallel. When a new rank was constructed, three buffer spaces were used. At each instant a block of the array, for example *NW*, was read into one of them, another was accessed by the processor, and in the third the renewed state of the array was written into the medium. Once an operating cycle was completed, the regions changed their function by circulation. The program itself needed appropriate restructuring because all its commands were adjusted for using the appropriate part of the store.

### *4.4 Developments of improved exchange software*

Convenient macro-operations were developed for array opening and closure. Also, for compatibility with the multi-buffer configuration and for making the exchange periods approach the potential of the media, the macro-operations of exchange with the tape and the disc had to be improved.

## 5. Program Implementation of the Algorithm

### *5.1 Batch processing of positions*

In analysis of end-games the positions were handled individually in almost no part of the program. The argument and the operation result are position sets: a *document* of 4096 bits (positions of two chessmen change), a *board* of 64 bits (the position of one chessman is variable) or still smaller sets of bits.

The method proved especially effective in transposition algorithms for unpacked arrays (Sec. 5.2) and back-up (Sec. 5.3).

### *5.2 Transposition*

A square matrix of unpacked format is

transposed in two stages. First each row of the original matrix is divided into equal parts. Parts of different rows belonging to the same groups of columns are assembled and written into the medium. This is performed repeatedly (twice in our case) until the set of columns of the original matrix can be stored in the core memory in its entirety. Then in the memory the transposition is completed by the method put forward by M. M. Bongard, involving permutations and conjunctions of sets of bits.

The algorithm takes somewhat less than a minute for a 4096 × 4096 matrix.

A square matrix in packed format is transposed in a similar way. The matrix is divided several times. Each of its squares is covered by some part denoted by the ordinal number of the column. The transposition time for a packed array is proportional to its size and is the same as the unpacked transposition time for about 500,000 nonempty elements.

The output of the ranking program was a totality of arrays $P_j^i$ (see Sec. 3). To facilitate computer playing of Rook end-games, this totality should be recorded. For each $j$th position of the Pawn a rectangular $a \times 2^{24}$ matrix should be transposed where the number of ranks $a$ varies for different $j$ from 20 to 60. Originally, this matrix was represented as rows, or as successive subarrays, each represented in a packed format. In transposition each block of these subarrays was transferred to a disc, a random access medium. Concurrently, an inquiry file $\tilde{S}$ is made in main memory, where for each block $B$ of the disc the disc address of that block and the number of the column of the matrix for the first element in $B$ are entered.

Then the inquiry file $S$ is ordered in terms of the numbers of the columns; an $\tilde{S}$ file is obtained. Now the processor memory successively calls block of the disc in compliance with the movement in $\tilde{S}$. Elements of similar columns are grouped and entered into the medium.

This routine takes 20-30 minutes for each $j$.

### *5.3 Backing moves*

The argument and the result of the operation *MRK*, that is, backward moves by the Rook and the King, is a document of unpacked format, $64^2 = 4096$ bits.

Let the White pawn be in square $p$; the document $D$ assumes positions $m$ and $n$ of the Rook and the King of one colour and inside the values $k$ and $l$ of squares of the same pieces of the other colour. Let us refer to pieces taking the squares $p$, $m$ and $n$ as external (for the document $D$) and those taking $k$ and $l$ as internal. Also, let us define the position of the Rook in terms of the coordinates $m$ and $k$ and that of the king, $n$ and $l$; in other words, the Rook is assumed senior to the King.

A document is required, $D = MRK(D)$ whose unit bits will be associated with positions with at least one move by internal pieces into some position recorded in $D$.

The operation *MRK* is represented as a merger of two operations, *MR*, moves of the Rook, and *MK*, moves of the King: $MRK = MR \cup MK$; in other words, we will have two intermediate documents, $D_R = MR(D)$ and $D_K = MK(D)$ which will be merged later: $MRK(D) = D_R \cup D_K$.

The document $D_R = MR(D)$ is obtained as a totality of boards $D_R(q)$, $0 \leq q \leq 63$ where $D_R(q)$ denotes a portion of the document $D_R$, namely 64 successive bits associated with the position of the Rook at the square $q$.

Let the internal Rook be at the square $q_c$. Then there are fourteen squares, $q_1, q_2 \ldots q_2$, within its range in an empty board. Remove from this set those squares that are shielded by external chessmen. The remaining squares make a set $Q = [q_i]$.

Let us take up a totality of positions recorded in the board $D(q_i)$ of the original document $D$, $q_i \in Q$. From this totality remove positions associated with the positions of the internal King between the squares $q_i$ and $q_c$. Then for each of the remaining positions there is a move of the internal Rook from the square $q_i$ to the field $q_c$. Therefore to implement this move, first, the board $D(q_i)$ should be moved to $D_R(q_c)$ and, second, to make the bits associated with the fields between $q_i$ and $q_c$ zero. Perform this operation for all $q_i$ from $K$ and unite the results in $D_R(q_c)$.

To save computer time, for each square of the chessboard a separate program without cycles was written which implements the moves of the Rook to that field (with the chessboard empty). The

program was not written manually; instead, a sequence of its commands (approximately 64 × 50 long) was developed by another program. The resulting back-up program was corrected for each set of positions of external chessmen which shielded certain squares from the internal Rook.

The operations of *MK* (moves of the king) over the document *D* are performed separately for each board $D(q)$, $0 \leq q \leq 63$. The board $D(Q)$ leads to the board $MK(D(q))$ where one's are in the bits of those squares from which there is at least one move of the King to the unity bit of the board $D(q)$. The *MK* operation for one board takes several logical commands of the computer.

The entire *MRK* operation over a document of $64^2$ bits took five to six thousand computer commands, or about a minute of computer time for each rank.

## 6. Computer Results

In program operation, obtaining first ranks of each square took a few minutes and for subsequent ranks, 30 to 45 seconds. Each column took about 15 computer hours.

Computations revealed that in the seventh row the number of ranks varied from 20 to 23 (for different columns). This number increased with decreasing ordinal number of the row and amounted to 50 by 60 by the second row. The number of positions won by White with the move of White varied from square to square within 1.5 to 4.5 million and with Black's move from 0.5 to 5.5 million.

Only these positions, or losses of Black, were retained in the program output, which required, depending on the position of the Pawn, from two to eight million bytes. The overall size of this set of positions for all the squares was about 60 million and their storage required about 120 million bytes.

After classification of all the positions into winning for White and drawing, the same procedure was used to classify the drawing positions into enforced and positional draws. Enforced draws are those achieved by capture of the White Rook or Pawn in several moves in positions of the KPR-KR type. For human players the outcome in such positions is a foregone conclusion; the fraction of such positions, especially for the last rows, is tremendous. Thus, for the square d7 there are about 2 million draws, achieved with a move by Black, and among those only 35,000 are positional.

In positions which take White the longest to win (60 moves) the white Pawn is on the field b2. These positions are shown in Figs. 3-6 (Black's move):

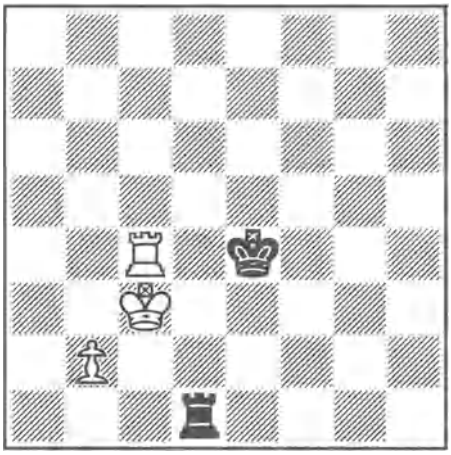


*Figure 3*

What follows is the optimal way to victory from the position of Figure 3 (1):

1 ... ♔e5 2 ♖c5+ ♔d6 3 ♔b4 ♖b1 4 ♖c2 ♖f1 5 ♔b5 ♖f5+ 6 ♔b6 ♖f8 7 ♖d2+ ♔e5 8 ♔c7 ♖f4 9 ♔c6 ♖c4+ 10 ♔b5 ♖c8 11 ♖h2 ♖b8+ 12 ♔c6 ♔e6 13 ♖h6+ ♔f7 14 ♖h7+ ♔e6 15 ♖h2 ♔e7 16 ♔c7 ♖b3 17 ♖e2+ ♔f7 18 ♔c6 ♖b8 19 ♔c5 ♖c8+ 20 ♔d5 ♖d8+ 21 ♔c4 ♖c8+ 22 ♔d3 ♖d8+ 23 ♔c2 ♖c8+ 24 ♔b1 ♖b8 25 ♖e3 ♖d8 26 ♔c2 ♖c8+ 27 ♔d3 ♖b8 28 ♔c3 ♖c8+ 29 ♔d4 ♖d8+ 30 ♔c5 ♖c8+ 31 ♔d6 ♖b8 32 b3 ♖b5 33 ♔c6 ♖b8 34 ♖d3 ♔f8 35 ♔c5 ♔e7 36 b4 ♖c8+ 37 ♔b5 ♖b8+ 38 ♔a4 ♖a8+ 39 ♔b3 ♖b8 40 ♖d4 ♔e6 41 ♔c4 ♔e5 42 ♖d5+ ♔e6 43 b5 ♖c8+ 44 ♖c5 ♖b8 45 ♔b4 ♔e7 46 ♔a5 ♔d6 47 ♖g5 ♖c8 48 ♔b6 ♖d8 49 ♔b7 ♖d7+ 50 ♔a6 ♖e7 51 b6 ♖e3 52 ♔a7 ♔c6 53 ♖g6+ ♔b5 54 ♖d6 ♖f3 55 b7 ♖a3+ 56 ♔b8 ♖c3 57 ♖d2 ♔c6 58 ♖a2 ♖b3 59 ♔c8 ♖e3 60 ♖c2+ ♔d6 61 b8=♕+. White easily wins in the resultant position.

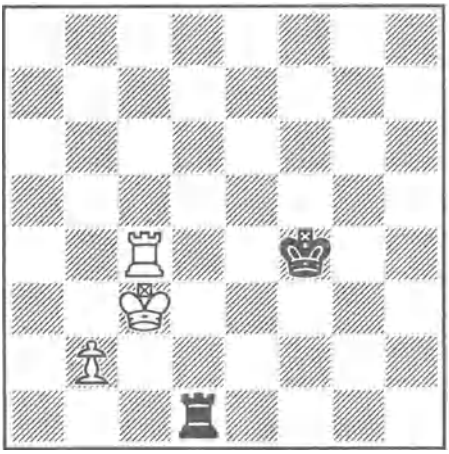


*Figure 4*

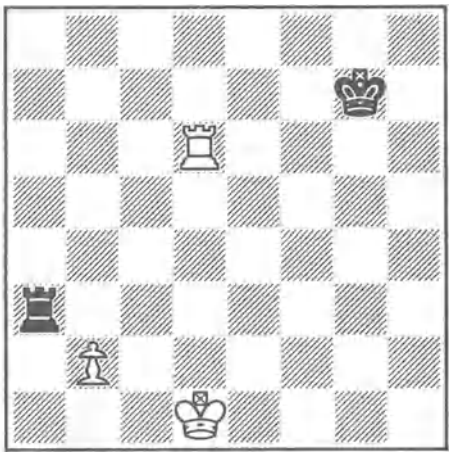


*Figure 6*

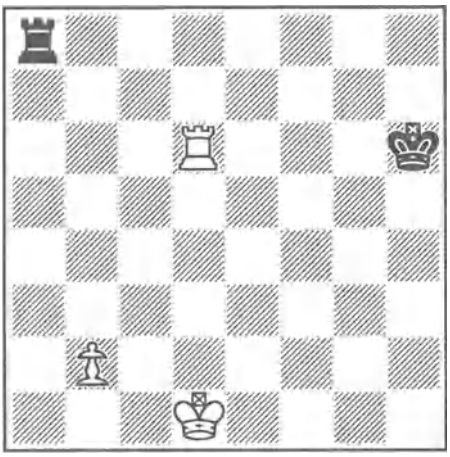


*Figure 5*

**Acknowledgements** The authors are indebted to A. R. Bitman, Master of Sports in Chess, for valuable consultations. They would also like to thank the following colleagues in the Institute who helped in programming the algorithms: A. G. Aleksandrov, A. M. Baraev, V. N. Deza, I. A. Faradzhev, Ya. Yu. Gol'fand, T. P. Il'ina, E. A. Komissarchik, and A. V. Uskov.

**Editors' note**

The reader should be aware that not all possible derived sub-games are covered in the authors' Fig. 2. Some of these (eg. KN-KR) would require further computations for sub-division according to the "won-drawn-lost" criterion.

# 9 Games Played by Chess Programs

For the sake of historical completeness, this chapter includes every game from every "important" computer chess tournament up to the end of 1987. I have also included a selection of games against human players, indicating, where appropriate, the Elo rating or title of the human opponent.

Many of the games given here are of no intrinsic value, other than being of historic importance. I have included them because I felt that they deserved to be preserved for posterity, so that it will be posssible to compare the style and strength of chess programs at any time in the future with those of the best programs of bygone years.

In addition to including all the games of the most important chess computer chess tournaments, I have given the full results of these events.

# 9.1 Games from the ACM Tournaments

Every year since 1970, the Association for Computing Machinery (ACM) has held a computer chess tournament as part of its annual conference. The 1970 event was organised by Monroe Newborn and Kenneth King of Columbia University. It was the first time that a chess tournament had taken place in which all the contestants were computer programs. The event was such a success that it immediately became a regular fixture at the ACM annual conference, and the following year I was invited to be tournament director and commentator at the tournament – a role I held for 10 successive years.

Although the event is known as the North American Computer Championship, it has allowed the participation of suitable strong European programs. For this reason, one can argue that the ACM tournaments are virtually World Championship events, although only one of them (1983) was actually the official World Computer Championship.

**ACM 1970 – New York**

| | rate | perf | 1 | 2 | 3 | total |
|---|---|---|---|---|---|---|
| 1 Chess 3.0 | 1450 | 1632 | 3+ | 4+ | 5+ | 3 |
| 2 Daly CP | 0 | 1182 | 5+ | 3– | 6+ | 2 |
| 3 Coko III | 1300 | 1287 | 1– | 2+ | 4= | 1½ |
| 4 J. Biit | 0 | 1197 | 6+ | 1– | 3= | 1½ |
| 5 Schach | 1025 | 937 | 2– | 6+ | 1– | 1 |
| 6 Marsland CP | 0 | 511 | 4– | 5– | 2– | 0 |

*Round 1, August 31, 1970*

**Marsland CP – J. Biit** 1 c4 ♘f6 2 d4 e6 3 ♕d3 ♘c6 4 ♘f3 d5 5 ♘e5 d×c4 6 ♕×c4 ♗b4+ 7 ♗d2 ♗×d2+ 8 ♔×d2 ♘×e5 9 ♕c5 ♘e4+ 0-1

**Chess 3.0 – Coko III** 1 c4 e5 2 ♘c3 ♘c6 3 ♘f3 ♗c5 4 e3 d6 5 d4 ♗b4 6 ♗d2 ♘f6 7 ♘d5 ♗×d2+ 8 ♕×d2 0-0 9 ♘×f6+ ♕×f6 10 d×e5 d×e5 11 ♗d3 ♗g4 12 ♗e4 ♖ad8 13 ♕c2 h5 14 0-0 ♗×f3 15 ♗×f3 h4 16 ♖ad1 ♘b4 17 ♕e4 ♘×a2 18 ♕×b7 ♖×d1 19 ♖×d1 ♕b6 20 ♖a1 ♕×b7 21 ♗×b7 ♖b8 22 ♖×a2 ♖×b7 23 f3 ♖b4 24 c5 c6 25 g3 h×g3 26 h×g3 ♖c4 27 ♖×a7 ♖c1+ 28 ♔f2 ♖c2+ 29 ♔e1 ♖×b2 30 ♖c7 ♖g2 31 g4 e4 32 f×e4 ♖×g4 33 ♖×c6 ♖×e4 34 ♔f2 ♔h8 35 ♖c8+ ♔h7 36 c6 g5 37 ♔f3 f5 38 ♔f2 f4 39 c7 f×e3+ 40 ♔e2 ♖e7 41 ♖h8+ ♔×h8 42 c8(♕)+ ♖e8 43 ♕×e8+ ♔g7 44 ♕e6 1-0

**Daly CP – Schach** 1 e3 d5 2 ♕h5 ♘c6 3 ♗b5 g6 4 ♕e5 f6 5 ♕d4 e5 6 ♗×c6+ ♗d7 7 ♗×d7+ ♔e7 8 ♕×d5 ♕×d7 9 ♕×b7 ♕d8 10 ♕c6 ♗h6 11 d3 ♖b8 12 ♕c5+ ♔e6 13 ♕×a7 ♖b6 14 ♕a4 c6 15 ♕c4+ ♔e7 16 ♘c3 ♔e8 17 ♘a4 ♖b8 18 ♕×c6+ ♕d7 19 ♕×d7+ ♔×d7 20 ♘b6+ ♔c6 21 ♘a4 1-0

*Round 2, September 1, 1970*

**J. Biit – Chess 3.0** 1 d4 ♘f6 2 c4 e6 3 ♘c3 ♗b4 4 e3 ♗×c3+ 5 b×c3 ♘c6 6 d5 ♘e7 7 d×e6 f×e6 8 ♖b1 ♘c6 9 ♗d3 ♕e7 10 ♘f3 e5 11 ♗f5 e4 12 ♘d4 ♕c5 13 ♖b5 ♕×c4 14 ♕b3 ♕×b3 15 a×b3 ♘×d4 16 e×d4 0-0 17 0-0 a6 18 ♖c5 d6 19 ♗×c8 d×c5 20 ♗e6+ ♔h8 21 d×c5 ♖ae8 22 ♗c4 ♘g4 23 ♗e2 ♘e5 24 ♗e3 g6 25 ♖d1 ♖f7 26 ♖d4 ♘c6 27 ♖d2 ♔g7 28 ♗c4 ♖ff8 29 ♖d7+ ♖e7 30 ♖×e7+ ♘×e7 31 ♗d4+ ♔h6 32 ♗e5 ♖c8 33 h4 c6 34 ♗e6 ♖e8 35 ♗f7 ♖d8 36 ♗c4 ♖d1+ 37 ♔h2 ♘d5 38 g4 g5 39 h×g5+ ♔×g5 40 ♔h3 ♘f4+ 41 ♗×f4+ ♔×f4 42 ♗e2 ♖d2 43 ♗f1 ♖×f2 44 ♗c4 ♖f3+ 45 ♔h4 ♖×c3 46 ♗g8 e3 47 ♗c4 ♖×c4 48 b×c4 e2 49 g5 e1(♕)+ 50 ♔h5 ♕h1# 0-1

**Coko III – Daly CP** 1 e4 e6 2 ♘c3 ♕g5 3 d4 ♕h4 4 ♘f3 ♕g4 5 h3 ♕h5 6 ♘b5 ♗b4+ 7 c3 ♗a5 8 ♗f4 a6 9 ♘×c7+ ♗×c7 10 ♗×c7 ♘c6 11 d5 e×d5 12 e×d5 ♘ce7 13 ♕d4 ♘×d5 14 g4 ♕g6 15 ♕×d5 ♕c2 16 ♗c4 ♘h6 17 ♕e5+ ♔f8 18 ♗d6+ 1-0

**Schach – Marsland CP** 1 d4 e5 2 d×e5 1-0

*Round 3, September 2, 1970*

**Schach – Chess 3.0** 1 d4 d5 2 c4 d×c4 3 ♘f3 ♘f6 4 e3 e6 5 ♗×c4 c5 6 0-0 a6 7 ♕e2 ♘c6 8 ♘c3 c×d4 9 e×d4 ♘×d4 10 ♘×d4 ♕×d4 11 ♖d1 ♕h4 12 g3 ♕g4 13 f3 ♗c5+ 14 ♔h1 ♕g6 15 ♗e3 ♗×e3 16 ♕×e3 0-0 17 ♕c5 ♗d7 18 a3 ♖fc8 19 ♕b4 ♗c6 20 ♖d3 b5 21 ♘×b5 a×b5 22 ♗×b5 ♗×b5 23 ♕×b5 ♖cb8 24 ♕c4 ♖×b2 25 ♖b3 ♖d2 26 ♕c6 ♖e8 27 ♖e1 ♖ed8 28 ♖b7 ♕d3 29 a4 ♖c2 30 ♕b6 ♕×f3+ 31 ♔g1 ♕g2# 0-1

* Many of the tournament tables that appear in this chapter include Elo ratings. The "rate" column represents a pre-tournament rating based on human tournaments in which the program participated. The "perf" column represents the program's performance in the tournament. The ratings have been computed by Ken Thompson, at the Bell Telephone Labs, Murray Hill, New Jersey, and were first published by the I.C.C.A.

**Coko III – J. Biit** 1 d4 d5 2 ♘c3 ♘c6 3 ♗f4 ♗f5 4 ♘b5 ♖c8 5 a4 e6 6 ♘f3 ♘f6 7 e3 ♘e4 8 ♗d3 ♗b4+9 c3 ♗d6 10 ♗xe4 ♗xe4 11 ♗xd6 cxd6 12 0-0 a6 13 ♘a3 0-0 14 b4 ♕f6 15 a5 ♘b8 16 ♖c1 ♘c6 17 ♕e2 ♕g6 18 ♕d1 ♗d3 19 ♖e1 ♖fd8 20 ♕d2 ♗e4 21 ♘h4 ♕f6 22 f3 ♕xh4 23 g3 ♕g5 24 dxe4 dxe4 25 ♖f1 f6 26 ♖f4 d5 27 ♖h4 ♖c7 28 ♖d1 ♖e7 29 ♖e1 e5 30 ♕d1 exd4 31 exd4 g6 32 ♘c2 e3 33 ♕f3 ♖e4 34 ♖xe4 dxe4 35 ♕xe4 f5 36 ♕e6+ ♔g7 37 ♕xe3 ♕g4 38 ♕f4 ♕xf4 39 gxf4 h5 40 h4 ♔f6 41 ♘e3 ♖h8 42 d5 ♘a7 43 ♘c4 ♘b5 44 ♖e3 ♖c8 45 ♘b6 ♖xc3 46 ♖e6+ ♔f7 47 ♘d7 ♘c7 48 ♘e5+ ♔f8 49 ♖f6+ ♔g7 50 ♖xg6+ ♔h7 51 ♖g5 ♘xd5 52 ♖xh5+ ♔g7 53 ♖xf5 ♖c1+ 54 ♔h2 ♖c2+ 55 ♔g1 ♖c1+ 56 ♔f2 ♖c2+ 57 ♔e1 ♖h2 58 ♖g5+ ♔h7 59 ♖h5+ ♔g8 60 ♖g5+ ♔h7 61 ♘d3 ♘f6 62 ♖f5 ♔g6 63 ♖e5 ♖xh4 64 ♖c5 ♖h1+ 65 ♔e2 ♖h2+ 66 ♔e3 ♖a2 67 ♔f3 ♖a3 68 ♔e3 ♘g4+ 69 ♔e4 ♘f6+ 70 ♔e3 ♘g4+ 71 ♔e2 ♔f6 72 ♖g5 ♖a2+ 73 ♔f3 ♘h2+ 74 ♔e4 ♖e2+ 75 ♔d5 ♖d2 76 ♔e4 ♖e2+ 77 ♔d5 ♖d2 78 ♔d4 ♘f3+ 79 ♔e3 ♖xd3+ 80 ♔xd3 ♘xg5 81 fxg5+ ♔xg5 82 ♔e3 ♔f5 83 ♔f3 ♔e5 84 ♔e3 ♔d5 85 ♔d3 ♔e5 86 ♔e3 ♔d5 87 ♔d3 ♔e5 ½-½

**Marsland CP – Daly CP** 1 e3 e5 2 d4 ♕g5 3 h4 ♕f6 4 ♕f3 ♗b4+ 5 ♘c3 ♕xf3 6 ♘xf3 ♗xc3+ 7 ♗d2 ♗xb2 8 ♖b1 e4 9 ♘e5 ♗a3 10 ♗b4 ♗xb4+ 11 c3 ♗xc3+ 12 ♔d1 d6 13 ♗b5+ ♔e7 14 ♔c2 dxe5 15 ♔xc3 ♗e6 16 d5 ♗xd5 17 ♗c4 ♗xc4 18 ♔xc4 b6 19 ♖bd1 ♘d7 20 f3 exf3 21 ♖d2 ♔f6 0-1

**ACM 1971 – Chicago**

| | rate | perf | 1 | 2 | 3 | total |
|---|---|---|---|---|---|---|
| 1 Chess 3.5 | 1452 | 1669 | 8+ | 2+ | 3+ | 3 |
| 2 Tech | 1247 | 1386 | 4+ | 1– | 6+ | 2 |
| 3 Genie | 0 | 1411 | 7+ | 6+ | 1– | 2 |
| 4 David | 0 | 1139 | 2– | 7+ | 5= | 1½ |
| 5 CCCP | 0 | 1081 | 6– | 8+ | 4= | 1½ |
| 6 Coko III | 1300 | 1173 | 5+ | 3– | 2– | 1 |
| 7 Schach | 1025 | 924 | 3– | 4– | 8+ | 1 |
| 8 Mr. Turk | 0 | 512 | 1– | 5– | 7– | 0 |

*Round 1, August 2, 1971*

**Chess 3.5 – Mr. Turk** 1 ♘f3 d5 2 ♘c3 ♘f6 3 d4 e6 4 ♗g5 ♘c6 5 ♗xf6 gxf6 6 e3 a5 7 ♗b5 f5 8 ♗xc6+ bxc6 9 ♘e5 c5 10 ♕h5 1-0

**CCCP – Coko III** 1 e4 e5 2 ♘f3 ♘f6 3 ♘xe5 ♕e7 4 d4 d6 5 ♗b5+ c6 6 ♗xc6+ ♘xc6 7 ♘xc6 ♕xe4+ 8 ♗e3 ♕xg2 9 ♖f1 ♗h3 10 ♘d2 bxc6 11 ♕f3 ♕xf3 12 ♘xf3 ♗g2 13 ♘d2 ♗xf1 14 ♔xf1 d5 15 c4 dxc4 16 ♘xc4 ♘g4 17 h4 ♘xe3+ 18 fxe3 f5 19 a4 ♗e7 20 h5 g6 21 h6 ♗g5 22 ♘d6+ ♔e7 23 ♘c4 ♗xh6 24 b4 f4 25 exf4 ♗xf4 26 ♖e1+ ♔f6 27 b5 ♖ad8 28 bxc6 ♖xd4 29 ♘b2 ♖c8 30 a5 ♖xc6 31 ♔g1 ♗h2+ 32 ♔xh2 ♖c2+ 33 ♔h3 ♖c3+ 34 ♔h2 ♖d2+ 35 ♔g1 ♖g3+ 36 ♔f1 ♖f3+ 37 ♔g1 ♖g3+ 38 ♔f1 ♖f3+ 39 ♔g1 ♖xb2 40 a6 ♖a3 41 ♖f1+ ♔g5 42 ♖f2 ♖xf2 43 ♔xf2 h6 44 ♔g2 ♔f4 45 ♔h1 ♔g3 46 ♔g1 ♖a1# 0-1

**Genie – Schach** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗a4 ♘f6 5 0-0 b5 6 ♗b3 ♘xe4 7 ♘a3 ♗xa3 8 bxa3 d5 9 ♖e1 ♗g4 10 ♖e3 ♗xf3 11 ♖xf3 0-0 12 ♖f5 g6 13 ♖f3 ♕d6 14 d3 ♘f6 15 ♗g5 ♘g4 16 ♖g3 ♘xf2 17 ♔xf2 ♕xa3 18 ♗c1 ♕c5+ 19 ♔f3 e4+ 20 dxe4 dxe4+ 21 ♔e2 ♘d4+ 22 ♔f1 ♔h8 23 ♖g5 ♘f5 24 ♕d7 ♕d6 25 ♕xf5 ♕d1+ 26 ♔f2 e3+ 27 ♗xe3 ♕xa1 28 ♕e5+ ♕xe5 29 ♖xe5 ♖ad8 30 ♗c5 ♖g8 31 ♖d5 ♖ge8 32 ♖xd8 ♖xd8 33 ♗xf7 ♖d2+ 34 ♔e3 ♖d7 35 ♗d4+ ♖xd4 36 ♔xd4 c6 37 g4 g5 38 ♔c5 h6 39 ♔xc6 ♔h7 40 c4 1-0

**Tech – David** 1 e4 e6 2 d4 ♕h4 3 ♘c3 ♘c6 4 ♘f3 ♕h5 5 ♗d3 ♕g4 6 0-0 f6 7 ♗e3 a6 8 ♕e2 g5 9 d5 exd5 10 exd5 ♘d8 11 ♗f4+ ♘e7 12 ♗xc7 f5 13 d6 ♘e6 14 dxe7 ♔xe7 15 ♘d5+ ♔e8 16 ♘f6+ ♔f7 17 ♘xg4 fxg4 18 ♕e5 gxf3 19 ♕xh8 ♘xc7 20 ♕xh7+ ♗g7 21 ♕f5+ ♗f6 22 gxf3 ♘e6 23 ♗c4 d5 24 ♗xd5 ♗d7 25 ♗xe6+ ♗xe6 26 ♕e4 ♖b8 27 f4 gxf4 28 ♕xf4 ♖g8+ 29 ♔h1 ♗d5+ 30 f3 b5 31 ♖fd1 ♖d8 32 ♕f5 ♗e6 33 ♕xf6+ ♔xf6 34 ♖xd8 ♔e7 35 ♖e1 ♔xd8 36 ♖xe6 a5 37 ♖e5 a4 38 ♖xb5 ♔c7 39 ♔g2 ♔c6 40 c4 ♔c7 41 f4 ♔c6 42 ♔f3 ♔c7 43 ♔e4 ♔c6 44 f5 ♔c7 45 ♔d5 ♔c8 46 f6 ♔d7 47 ♖b8 a3 48 f7 axb2 49 f8(♕) b1(♕) 50 ♕g7# 1-0

*Round 2, August 3, 1971*

**Tech – Chess 3.5** 1 e4 c5 2 ♘f3 ♘c6 3 d4 cxd4 4 ♘xd4 ♘f6 5 ♘c3 d6 6 ♗c4 e6 7 0-0 a6 8 ♗e3 ♘e5 9 ♕e2 ♕c7 10 ♗d3 ♘eg4 11 f4 e5 12 ♘f5 ♗e6 13 ♘d5 ♘xd5 14 exd5 ♗xf5 15 ♗xf5 ♘h6 16 ♕d3 g6 17 ♗e4 f5 18 ♗f3 e4 19 ♕d4 ♗g7 20 ♕b4 exf3 21 ♖xf3 ♕xc2 22 ♕xb7 0-0 23 ♖f2 ♕d3 24 ♖e1 ♘g4 25 ♖d2 ♕e4 26 ♖de2 ♖fb8 27 ♕d7 ♕xd5 28 b3 ♘xe3 29 ♖xe3 ♗d4 30 ♕c7 ♗xe3+ 31 ♖xe3 ♖c8 32 ♕e7 ♕d1+ 33 ♔f2 ♖c2+ 34 ♖e2 ♕d4+ 35 ♔f3 ♖c3+ 36 ♖e3 ♕d1+ 37 ♔f2 ♕d2+ 38 ♖e2 ♕xf4+ 39 ♔g1 ♖c1+ 40 ♖e1 ♖xe1+ 41 ♕xe1 ♕e4 42 ♕xe4 fxe4 43 ♔f2 ♔f7 44 ♔e3 d5 45 ♔f4 ♔e6 46 b4 ♖c8 47 h3 ♖c2 48 a4 ♖xg2 49 b5 axb5 50 axb5 g5+ 51 ♔e3 ♔e5 52 b6 d4# 0-1

**Coko III – Genie** 1 d4 d5 2 ♘f3 ♘f6 3 ♗g5 ♗g4 4 ♘c3 ♘e4 5 ♘e5 ♗e6 6 ♘xe4 dxe4 7 c4 ♘d7 8 ♘xd7 ♗xd7 9 e3 f6 10 ♗f4 ♗e6 11 ♕h5+ ♔d7 12 d5 ♗g8 13 ♕f5+ e6 14 dxe6+ ♗xe6 15 ♕xe4 c6 16 ♖d1+ ♔e8 17 ♖xd8+ ♔xd8 18 ♕xe6 ♗b4+ 19 ♔e2 ♖e8 20 ♕g4 g6 21 ♕h4 g5 22 ♕xh7 ♗e7 23 ♕d3+ ♔c8 24 ♗d6 ♔d7 25 ♗xe7+ ♔xe7 26 ♕h7+ ♔e6 27 ♕e4+ ♔d6 28 c5+ ♔xc5 29 ♕d4+ ♔b5 30 ♔d1+ ♔a5 31 b4+ ♔a4 32 ♕c3 ♖ad8+ 33 ♔c2 ♖d2+ 34 ♔xd2 ♖d8+ 35 ♔c2 ♖d2+ 36 ♕xd2 ♔a3 37 ♕c3+ ♔xa2 38 ♔c1 f5 39 ♔c2 f4 40 ♔c1 g4 41 ♔c2 f3 42 ♔c1 fxg2 43 ♔c2 gxh1(♕) 44 ♔c1 ♕xf1+ 45 ♔d2 ♕xf2+ 46 ♔c1 ♕g1+ 47 ♔c2 ♕xh2+ 48 ♔c1 ♕h1+ 49 ♔c2 ♕b1+50 ♔d2 g3 51 ♕c4+ ♕b3 52 ♕xb3+ ♔xb3 53 e4 ♔xb4 54 e5 g2 0-1

**David – Schach** 1 e4 e5 2 ♘c3 ♘f6 3 d4 d6 4 ♘f3 ♘g4 5 h3 ♘h6 6 dxe5 dxe5 7 ♕xd8+ ♔xd8 8 ♗xh6 gxh6 9 ♖d1+ ♗d6 10 ♘xe5 ♘d7 11

♘×f7+ ♔e7 12 ♘×h8 ♗e5 13 ♘d5+ ♔f8 14 f4 ♗×h8 15 ♘×c7 ♗×b2 16 ♘×a8 ♗c3+ 17 ♔f2 1-0

**Mr Turk — CCCP** 1 d4 d5 2 ♘f3 ♗g4 3 ♘a3 ♗×f3 4 e×f3 ♘c6 5 c3 ♕d6 6 ♘b5 ♕d8 7 a4 a6 8 c4 a×b5 9 c×d5 ♘a5 10 b4 ♘c4 11 h3 ♕d7 12 ♗d2 ♘b2 13 ♕e2 ♘×a4 14 ♖a2 ♕e6 15 ♕e3 ♕d7 16 ♗e2 f5 17 ♕f4 g5 18 ♕×g5 ♘f6 19 ♗d3 ♖g8 20 ♕×f5 ♕×f5 21 g4 ♕×d3 22 ♔d1 ♘c3+ 0-1

*Round 3, August 4, 1971*

**Genie — Chess 3.5** 1 e4 c5 2 ♘f3 ♘c6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 d6 6 ♘×c6 b×c6 7 ♗f4 e5 8 ♗e3 ♗e6 9 ♗g5 ♖b8 10 b3 ♗e7 11 ♗c4 ♗×c4 12 b×c4 ♕d7 13 ♖b1 ♖×b1 14 ♘×b1 ♘×e4 15 ♗c1 0-0 16 ♕h5 ♖b8 17 ♘d2 ♘×d2 18 ♗×d2 ♖b1+ 19 ♕d1 ♖×d1+ 20 ♔×d1 ♕g4+ 21 f3 ♕×g2 22 ♖e1 ♕×f3+ 23 ♔c1 f5 24 ♖e3 ♕f1+ 25 ♗e1 ♗g5 26 c5 ♗×e3+ 27 ♔b2 ♕×e1 28 c×d6 ♕b4+ 29 ♔a1 ♗d4+ 30 c3 ♗×c3# 0-1

**Coko III — Tech** 1 e4 e5 2 ♘f3 ♘c6 3 ♗c4 ♘f6 4 d3 d5 5 ♗×d5 ♘×d5 6 e×d5 ♕×d5 7 ♘c3 ♗b4 8 0-0 ♗×c3 9 b×c3 0-0 10 ♘g5 ♗f5 11 ♖b1 f6 12 c4 ♕c5 13 ♘h3 ♗×h3 14 ♗e3 ♘d4 15 g×h3 ♕c6 16 c3 ♘f3+ 17 ♔h1 ♘d2+ 18 f3 ♘×f1 19 ♕×f1 f5 20 ♖b5 f4 21 ♖c5 ♕e6 22 ♗c1 c6 23 d4 ♖ae8 24 ♖×e5 ♕g6 25 ♖×e8 ♕×e8 26 ♕f2 ♕e6 27 ♕f1 ♖f5 28 h4 c5 29 d5 ♕d6 30 ♕h3 ♕e5 31 ♕f1 ♕×c3 32 d6 ♕d4 33 ♕e2 ♕×d6 34 ♕e8+ ♖f8 35 ♕a4 ♖f5 36 ♕e8+ ♖f8 37 ♕a4 ♕e6 38 ♕b3 ♕e2 39 h3 ♖d8 40 ♗×f4 ♖d1+ 0-1

**CCCP — David** 1 e4 e6 2 d4 ♘c6 3 ♘c3 g5 4 ♗b5 a6 5 ♗×c6 d×c6 6 ♗e3 b5 7 ♕h5 b4 8 ♗×g5 ♘f6 9 ♗×f6 ♕×f6 10 e5 ♕g7 11 ♘e4 ♕×g2 12 ♕f3 ♕×f3 13 ♘×f3 ♗h6 14 ♘f6+ ♔e7 15 a3 b×a3 16 ♖×a3 ♖b8 17 0-0 ♖×b2 18 c4 a5 19 ♖×a5 ♖b3 20 ♔g2 ♖×f3 21 ♔×f3 ♗d2 22 ♖c5 ♗c3 23 ♖×c6 ♗b7 24 ♘d5+ e×d5 25 ♖×c7+ ♔e6 26 ♖×b7 d×c4 27 ♖b6+ ♔f5 28 ♖f6+ ♔g5 29 ♖g1+ ♔h5 30 ♖f5+ ♔h6 31 ♖f6+ ♔h5 32 ♖f5+ ♔h6 33 ♖f6+ ♔h5 34 ♖f5+ ½-½

**Schach — Mr. Turk** 1 d4 d5 2 c4 d×c4 3 ♘f3 ♘h6 4 ♕a4+ ♘c6 5 e4 1-0

*2nd Place Playoff, September 1971*

**Genie — Tech** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 ♘f6 4 0-0 ♗c5 5 ♘c3 d6 6 d4 e×d4 7 ♘×d4 ♗d7 8 ♘f5 0-0 9 ♗g5 ♘e5 10 ♘a4 ♗×b5 11 ♘×c5 ♗×f1 12 ♘×b7 ♕b8 13 ♘e7+ ♔h8 14 ♘×d6 c×d6 15 ♗×f6 g×f6 16 ♕×f1 ♕b4 17 ♘d5 ♕×e4 18 ♘×f6 ♕×c2 19 ♕c1 ♕×c1+ 20 ♖×c1 ♔g7 21 ♘h5+ ♔g6 22 ♘f4+ ♔f5 23 ♘d5 ♖ad8 24 ♖e1 ♖b8 25 b4 ♘d3 26 ♖e2 ♘×b4 27 ♘c3 ♖fe8 28 a3 ♖×e2 29 ♘×e2 ♘d5 30 h4 ♔e4 31 h5 ♖d8 32 h6 ♖c8 33 g4 ♔f3 34 ♘g3 ♔×g4 35 a4 ♔f3 36 a5 ♖g8 37 a6 ♖×g3+ 38 f×g3 ♔×g3 39 ♔f1 ♘b4 40 ♔e2 ♘×a6 41 ♔e3 ♘c5 42 ♔d4 ♔f4 43 ♔d5 ♘e4 44 ♔c6 ♔e5 45 ♔b7 a5 46 ♔c6 a4 47 ♔d7 a3 48 ♔e7 a2 49 ♔×f7 a1(♕) 50 ♔e7 ♕c1 51 ♔d7 ♕×h6 52 ♔c6 ♕e3 53 ♔c7 d5 54 ♔c6 d4 55 ♔d7 d3 56 ♔c6 d2 57 ♔d7 d1(♕)+ 58 ♔c6 ♕a7 59 ♔b5 ♕da4# 0-1

**ACM 1972 — Boston**

| | rate | perf | 1 | 2 | 3 | total |
|---|---|---|---|---|---|---|
| 1 Chess 3.6 | 1452 | 1693 | 3+ | 4+ | 2+ | 3 |
| 2 Tech | 1252 | 1301 | 5+ | 6+ | 1— | 2 |
| 3 Ostrich | 1300 | 1288 | 5+ | 6+ | 1— | 2 |
| 4 Coko III | 0 | 1319 | 1— | 7+ | 5+ | 2 |
| 5 Schach | 1025 | 982 | 2— | 8+ | 3— | 1 |
| 6 USC CP | 0 | 1003 | 8+ | 2— | 4— | 1 |
| 7 MSU CP | 0 | 803 | 4— | 3— | 8= | ½ |
| 8 Leverett CP | 0 | 730 | 6— | 5— | 7= | ½ |

*Round 1, August 13 1972*

**Chess 3.6 — Ostrich** 1 e4 c5 2 ♘f3 ♘c6 3 d4 c×d4 4 ♘×d4 e5 5 ♘f3 ♕b6 6 ♘c3 ♗c5 7 ♕d2 d6 8 ♘d5 ♕d8 9 b4 ♗d4 10 c3 ♗b6 11 ♘×b6 a×b6 12 ♗c4 ♗e6 13 ♗×e6 f×e6 14 ♘g5 ♕d7 15 0-0 h6 16 ♘f3 0-0-0 17 ♖d1 ♘f6 18 ♕e2 ♕c7 19 ♗b2 ♘e7 20 ♘d2 d5 21 c4 d4 22 ♕d3 ♘g6 23 g3 ♕e7 24 ♗a3 ♔c7 25 ♖ab1 ♖h7 26 ♖e1 ♖d7 27 ♔g2 ♕d8 28 h3 ♘g8 29 ♖b3 ♕g5 30 ♔h2 ♖f7 31 ♕e2 ♘f6 32 ♖f3 h5 33 h4 ♘g4+ 34 ♔h1 ♕e7 35 ♔g2 ♖×f3 36 ♘×f3 ♕f6 37 ♗c1 ♔d6 38 ♗g5 ♕f7 39 ♗d8 ♖h8 40 ♗×b6 ♔e7 41 ♘g5 ♕e8 42 b5 ♕c8 43 ♖b1 ♖f8 44 ♖f1 ♖e8 45 ♕f3 ♖f8 46 ♕a3+ ♔d7 47 ♗c5 ♖h8 48 ♘f7 ♖h7 49 ♖c1 ♘h6 50 ♘d6 ♕c7 51 b6 ♕c6 52 ♖e1 ♖h8 53 ♔g1 ♘g4 54 ♖f1 ♖h6 55 ♘f7 ♕×e4 56 ♘×h6 ♘×h6 57 f3 ♕c6 58 ♔f2 ♔d8 59 ♖b1 ♕d7 60 ♔g2 ♘f5 61 ♔f2 ♕c8 62 ♖b5 ♔d7 63 ♖b2 ♔c6 64 ♕b4 ♔d7 65 ♕b5+ ♕c6 66 ♖d2 ♕×b5 67 c×b5 ♔d8 68 ♖d3 ♘ge7 69 a4 g6 70 a5 ♔d7 71 a6 b×a6 72 b×a6 ♔c8 73 a7 ♔b7 74 ♖a3 ♔a8 75 b7+ ♔×b7 76 a8(♕)+ ♔c7 77 ♖a7# 1-0

**Schach — Tech** 1 d4 d5 2 c4 d×c4 3 ♘f3 ♘f6 4 e4 b5 5 d5 ♘g4 6 ♗g5 ♘f6 7 ♗×f6 e×f6 8 ♗e2 ♗c5 9 ♘c3 ♗d7 10 0-0 0-0 11 ♘h4 ♕e7 12 a3 ♘a6 13 ♘f5 ♕e5 14 ♘e3 c6 15 d×c6 ♗×c6 16 ♘×c4 b×c4 17 ♗×c4 ♘c7 18 ♕g4 ♗d4 19 ♖ac1 ♗×c3 20 ♖×c3 ♗×e4 21 ♖e1 ♗f3 22 ♖×e5 ♗×g4 23 ♖e7 ♘e6 24 ♗×e6 ♗×e6 25 g3 ♖fe8 26 ♖×e8+ ♖×e8 27 ♖c7 ♗d5 28 f4 a6 29 ♖d7 ♗e4 30 ♔f2 ♔f8 31 b4 ♖e6 32 ♖d8+ ♔e7 33 ♖d1 ♖d6 34 ♖e1 ♖d2+ 35 ♔g1 f5 36 h4 ♖g2+ 37 ♔f1 ♖×g3 38 a4 ♖b3 39 h5 ♖×b4 40 a5 ♖b1 41 ♔f2 ♖×e1 42 ♔×e1 ♗f3 43 h6 g×h6 44 ♔f2 ♗d5 45 ♔e3 ♔d6 46 ♔d4 ♗e6 47 ♔d3 ♗a2 48 ♔d4 ♗d5 49 ♔e3 ♔c6 50 ♔d4 h5 51 ♔e5 ♗e4 52 ♔d4 h4 53 ♔e5 h3 54 ♔d4 ♔b5 55 ♔e5 ♔×a5 56 ♔d4 h2 57 ♔e5 h1(♕) 58 ♔d4 h6 59 ♔e5 ♕h4 60 ♔d6 ♕×f4+ 61 ♔d7 ♔b6 62 ♔d8 h5 0-1

**USC CP — Leverett CP** 1 e4 ♘f6 2 e5 ♘e4 3 d3 ♘c5 4 ♘c3 ♘c6 5 ♘f3 d5 6 ♗e3 d4 7 ♘e4 ♘×e5 8 ♘×e5 ♘a4 9 ♘g5 ♘×b2 10 ♕f3 ♘×d3+ 11 ♗×d3 ♗e6 12 ♘e×f7 ♕d5 13 ♘×h8 ♕×f3 14 g×f3 ♗×a2 15 ♗×d4 0-0-0 16 ♖×a2 ♖×d4 17 ♘e6 ♖d6 18 ♘×f8 ♖f6 19 ♘×h7 ♖×f3 20 ♖×a7 c5 21 ♘g6 ♖f7 22 ♗e4 e6 23 ♘g5 ♖f6 24 ♖×b7 ♖×g6 25 ♗×g6 ♔×b7 26 ♘×e6 c4 27 ♘×g7 ♔c6 28 ♗f7 ♔b5 29 ♖g1 ♔b4 30 ♖g4 ♔a3 31 ♗×c4 ♔b2 32 ♔d2 ♔a3 33 ♔c3 ♔a4 34 h3 ♔a5 35 ♖g5+ ♔a4 36 ♗b3+ ♔a3 37 ♖a5# 1-0

**MSU CP – Coko III** 1 e4 e5 2 ♗b5 c6 3 ♗c4 d5 4 e×d5 c×d5 5 ♗b5+ ♗d7 6 a4 ♗c5 7 ♘f3 ♗×b5 8 a×b5 e4 9 d4 e×f3 10 0-0 f×g2 11 ♔×g2 ♗d6 12 ♕g4 ♘d7 13 ♕×g7 ♕f6 14 ♖e1+ ♘e7 15 ♗h6 ♕g6+ 16 ♕×g6 h×g6 17 ♗g7 ♖×h2+ 18 ♔g1 ♖h7 19 ♗f8 ♗h2+ 20 ♔g2 ♘×f8 21 b6 a6 22 ♖a4 ♘e6 23 ♖×e6 f×e6 24 c4 d×c4 25 ♖×c4 ♘d5 26 ♘a3 ♘×b6 27 ♖b4 ♘d5 28 ♖c4 ♖h4 29 b4 b5 30 ♖c6 ♔d7 31 ♖c5 ♖g4+ 32 ♔h3 ♖g5 33 ♖×d5+ e×d5 34 ♔×h2 ♖h8# 0-1

*Round 2, August 14, 1972*

**Coko III – Chess 3.6** 1 d4 d5 2 ♘c3 ♘f6 3 ♘f3 ♗g4 4 ♕d3 ♘c6 5 ♘e5 ♗d7 6 ♗g5 ♘b4 7 ♕d1 ♗f5 8 ♘d3 h6 9 ♗×f6 e×f6 10 a3 ♘×d3+ 11 c×d3 ♗e6 12 ♕b3 b6 13 e4 c6 14 ♕a4 ♕d7 15 e×d5 ♗×d5 16 ♘×d5 ♕×d5 17 ♖c1 ♔d7 18 ♕c4 ♗d6 19 ♕×d5 c×d5 20 ♗e2 ♖ae8 21 a4 f5 22 b3 ♗b4+ 23 ♔d1 ♖c8 24 ♖×c8 ♖×c8 25 ♗f3 ♔d6 26 ♔e2 ♖e8+ 27 ♔d1 ♗c3 28 g4 f×g4 29 ♗×g4 g6 30 f4 ♗×d4 31 b4 h5 32 ♗e2 ♖c8 33 ♖f1 ♗e3 34 ♔e1 ♖c2 35 f5 ♖a2 36 f×g6 f×g6 37 ♖f6+ ♔e5 38 ♖×g6 ♔d4 39 ♔d1 ♖×a4 40 b5 ♖a1+ 41 ♔c2 ♖e1 42 ♗d1 ♖f1 43 h4 ♖f2+ 44 ♔b3 ♖f4 45 ♗×h5 ♖×h4 46 ♗f3 ♔c5 47 ♖c6+ ♔×b5 48 ♗×d5 ♖d4 49 ♗f3 ♖×d3+ 50 ♖c3 ♖×c3+ 51 ♔×c3 a5 52 ♗d5 a4 53 ♔d3 ♔c5 54 ♗e6 ♗d4 55 ♔c2 b5 56 ♔d2 ♔d6 57 ♗f7 ♔e5 58 ♔e1 ♔f6 59 ♗d5 b4 60 ♔e2 b3 61 ♔f3 b2 62 ♗e4 a3 63 ♗b1 ♔e5 64 ♗a2 ♗c3 65 ♗b1 ♗d2 66 ♗a2 ♗g5 67 ♗b1 ♔d5 68 ♗a2+ ♔d4 69 ♗b1 ♔c3 70 ♔e2 ♔b3 71 ♔d1 a2 0-1

**Tech – USC CP** 1 e4 c5 2 ♘f3 ♘c6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 d6 6 ♗c4 e5 7 ♘f5 ♗e6 8 ♕d3 ♘b4 9 ♗b5+ ♗d7 10 ♗×d7+ ♕×d7 11 ♕e2 g6 12 ♗g5 ♘h5 13 a3 f6 14 a×b4 f×g5 15 ♘d5 ♕c6 16 0-0 ♘f4 17 ♘×f4 g×f4 18 b5 ♕b6 19 ♘h4 ♗e7 20 ♘f3 a6 21 ♕c4 g5 22 ♖fd1 g4 23 ♘d2 h5 24 ♖a4 ♗g5 25 ♕e6+ ♔d8 26 ♘c4 ♕×b5 27 ♕×d6+ ♔c8 28 ♘b6+ ♕×b6 29 ♕×b6 ♖h6 30 ♕c5+ ♖c6 31 ♕f8+ ♔c7 32 ♕×a8 ♖×c2 33 ♕g8 ♗h4 34 ♖c4+ ♖×c4 35 ♕×c4+ ♔b8 36 ♖d7 a5 37 ♕c7+ ♔a7 38 ♕×b7# 1-0

**Ostrich – MSU CP** 1 c4 ♘c6 2 d4 ♘f6 3 d5 ♘b8 4 ♘f3 ♘a6 5 ♗g5 ♘e4 6 ♘bd2 ♘×g5 7 ♘×g5 f6 8 ♘ge4 f5 9 ♘g3 f4 10 ♘h5 ♘c5 11 b4 e5 12 b×c5 g6 13 e3 f×e3 14 f×e3 ♗×c5 15 ♕f3 ♕h4+ 16 ♘g3 ♖f8 17 ♕e2 ♗b4 18 ♕d3 g5 19 0-0-0 ♖f2 20 ♘f3 ♕g4 21 ♘×e5 ♕×d1+ 22 ♔×d1 d6 23 ♘e4 ♖×f1+ 24 ♖×f1 d×e5 25 ♘f6+ ♔e7 26 ♕×h7+ ♔d8 27 ♕h8+ ♔e7 28 ♕e8+ ♔d6 29 ♘e4# 1-0

**Leverett CP – Schach** 1 ♘c3 d5 2 e4 d×e4 3 ♘×e4 e5 4 ♕h5 ♘c6 5 ♘f3 ♘f6 6 ♘×f6+ g×f6 7 ♗c4 ♕d7 8 ♕×f7+ ♕×f7 9 ♗×f7+ ♔×f7 10 0-0 ♖g8 11 d3 ♗c5 12 ♗e3 ♘d4 13 ♘×d4 e×d4 14 ♗f4 ♗d6 15 ♗g3 ♗e6 16 a4 ♖g5 17 h4 ♖g4 18 b4 ♗×g3 19 f×g3 ♖×g3 20 ♖f4 ♖d8 0-1

*Round 3, August 15, 1972*

**Chess 3.6 – Tech** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 ♘f6 4 0-0 ♗c5 5 ♘c3 d6 6 ♗×c6+ b×c6 7 d4 e×d4 8 ♘×d4 0-0 9 ♗g5 ♗g4 10 ♕d3 ♗×d4 11 ♕×d4 ♖b8 12 ♗×f6 ♕×f6 13 ♕×f6 g×f6 14 b3 ♖b4 15 h3 ♗e6 16 g4 ♖d4 17 ♖ad1 ♖×d1 18 ♘×d1 ♔g7 19 ♘e3 ♔g6 20 f4 ♔g7 21 ♔g2 ♖b8 22 ♔f3 ♖b5 23 c4 ♖a5 24 f5 ♗d7 25 ♖f2 ♖e5 26 ♖d2 a6 27 h4 c5 28 ♘d5 ♗c6 29 ♘×c7 ♗×e4+ 30 ♔f4 h5 31 g×h5 a5 32 ♖×d6 ♗×f5 33 h6+ ♔g6 34 h5+ ♔×h5 35 ♖×f6 ♖e2 36 ♔×f5 ♖f2+ 37 ♔e5 ♖h2 38 ♘d5 ♔g5 39 ♘c3 ♖h4 40 ♖×f7 ♔×h6 41 ♘e4 ♖h5+ 42 ♔d6 ♔g6 43 ♖a7 a4 44 ♖×a4 ♔f7 45 ♖a7+ ♔g6 46 a4 ♖f5 47 a5 ♖f3 48 ♖b7 ♔f5 49 ♘×c5 ♖c3 50 a6 ♖h3 51 a7 1-0

**USC CP – Coko III** 1 e4 e5 2 ♘f3 d5 3 e×d5 ♕×d5 4 ♘c3 ♕a5 5 ♗b5+ ♗d7 6 a4 ♗×b5 7 ♘×b5 ♘d7 8 c3 ♘gf6 9 b4 ♕b6 10 d4 c6 11 d×e5 c×b5 12 ♕d3 ♘g4 13 ♗e3 ♘×e3 14 f×e3 b×a4 15 ♖×a4 ♕e6 16 ♕b5 ♕b3 17 ♖f1 ♕×c3+ 18 ♔e2 ♕c6 19 ♕a5 ♕c4+ 20 ♔f2 b5 21 ♖aa1 ♗×b4 22 ♕a6 ♗c3 23 ♖fc1 0-0 24 ♖a3 ♘c5 25 ♖c2 ♘×a6 26 ♖c×c3 ♕e6 27 ♘d4 ♕×e5 28 ♖×a6 ♖fd8 29 g3 b4 30 ♖c4 ♖d6 31 ♖ac6 a5 32 ♖c8+ ♖d8 33 ♖8c5 ♖d5 0-1

**Schach – Ostrich** 1 d4 ♘f6 2 c4 e6 3 ♘c3 ♗b4 4 ♕c2 0-0 5 e3 ♘c6 6 ♗d2 a5 7 d5 e×d5 8 c×d5 ♗×c3 9 ♗×c3 ♘×d5 10 0-0-0 ♘cb4 11 ♗×b4 ♘×b4 12 ♕c4 d5 13 ♕b3 ♗f5 14 a3 ♗c2 15 ♕c3 ♘a2+ 16 ♔×c2 ♘×c3 17 b×c3 ♕g5 18 f4 ♕g6+ 19 ♔b2 ♕b6+ 20 ♔a2 a4 21 ♖d3 ♕b3+ 22 ♔a1 ♕×a3+ 23 ♔b1 ♖a5 24 ♘h3 ♖b5+ 25 ♔c2 ♕a2+ 0-1

**MSU CP – Leverett CP** 1 d4 ♘f6 2 ♗g5 ♘e4 3 ♘f3 ♘c6 4 ♕d3 ♘×g5 5 ♘×g5 e6 6 ♘e4 ♗b4+ 7 c3 f5 8 ♘c5 ♗×c5 9 d×c5 ♕e7 10 ♕b5 a6 11 ♕c4 0-0 12 e3 g5 13 ♗e2 f4 14 ♗h5 h6 15 e4 a5 16 ♔d2 ♔g7 17 a4 ♔h8 18 h3 ♖d8 19 ♖h2 ♖a7 20 f3 ♕f6 21 ♘a3 ♖a8 22 ♘b5 ♕e5 23 ♗g6 ♔g7 24 ♗h5 ♖h8 25 ♔d3 ♖f8 26 b3 ♘b4+ 27 ♔d2 ♘c6 28 ♖a2 ♘e7 29 ♕d4 ♕×d4+ 30 ♘×d4 e5 31 ♘b5 ♖a6 32 ♘×c7 ♖c6 33 ♘e8+ ♖×e8 34 ♗×c8 ♖×c5 35 ♔d3 d5 36 ♖d2 ♗e6 37 ♖b2 b6 38 ♗h5 ♗d7 39 ♖f2 ♗c6 40 ♖d2 ♔h8 41 ♗f7 d×e4+ 42 f×e4 b5 43 b4 a×b4 44 c×b4 ♖c1 45 ♗d5 b×a4 46 ♗×c6 ♘×c6 47 ♖b2 ♖g1 48 b5 ♖d1+ 49 ♔e2 ♖d6 50 b×c6 h5 51 ♖b8+ ♔h7 52 ♖e8 ♖×c6 53 ♖×e5 h4 54 ♔d3 a3 55 ♖e7+ ♔g6 56 ♖a7 ♖d6+ 57 ♔c2 ♖c6+ 58 ♔d3 ♖d6+ 59 ♔c4 ♖c6+ 60 ♔d4 ♖d6+ 61 ♔c3 ♖c6+ 62 ♔d3 ♖d6+ ½-½

*2nd Place Playoff, August 1972*

**Ostrich – Tech** 1 c4 e5 2 ♘c3 ♘f6 3 e4 ♘c6 4 d3 ♗c5 5 ♗g5 0-0 6 ♘f3 d6 7 ♗e2 ♗e6 8 0-0 ♘d4 9 ♘d5 c6 10 ♗×f6 ♘×f3+ 11 ♗×f3 g×f6 12 ♘c3 ♗d4 13 ♕b3 ♕a5 14 ♘e2 ♖ab8 15 ♖ae1 b5 16 ♘×d4 e×d4 17 ♕a3 ♕×a3 18 b×a3 b×c4 19 ♖c1 ♖b5 20 d×c4 ♖c5 21 ♖fd1 ♗×c4 22 ♖×d4 ♗e6 23 ♖cd1 ♔g7 24 h4 ♖c8 25 ♖×d6 ♗×a2 26 h5 ♗e6 27 ♖1d3 ♖b8 28 ♗d1 ♖c8 29 f4 ♖e8 30 ♖g3+ ♔f8 31 h6 ♔e7 32 ♖d2 ♖c8 33 f5 ♗c4 34 ♖g7 ♖e5 35 ♖×h7 ♖×e4 36 ♗c2 ♖h4 37 g3 ♖g8 38 ♖g2 ♖d4 39 ♔h2 1-0

**Tech – Coko III** 1 e4 e5 2 ♘f3 ♘f6 3 d4 ♗b4+ 4 ♗d2 ♗×d2+ 5 ♘b×d2 e×d4 6 ♘×d4 0-0 7 ♗c4

♘×e4 8 ♘×e4 d5 9 ♗×d5 ♕×d5 10 f3 ♘c6 11 c3 f5 12 ♘×c6 ♕×c6 13 ♘g5 ♖e8+ 14 ♔f2 h6 15 ♕b3+ ♗e6 16 ♘×e6 ♖×e6 17 ♖ad1 ♕b6+ 18 ♖d4 c5 19 ♖d5 c4+ 20 ♕×b6 a×b6 21 a4 ♖×a4 22 ♖×f5 g6 23 ♖d5 ♖a2 24 ♖b1 ♔f7 25 f4 ♖e4 26 ♔f3 ♖e7 27 f5 g×f5 28 ♖×f5+ ♔e6 29 ♔e4 ♖a5 30 g4 h5 31 ♖×h5 ♖×h5 32 g×h5 ♔d6+ 33 ♔d4 ♖h7 34 ♖d1 ♖×h5 35 ♔×c4+ ♔c6 36 ♖d2 b5+ 37 ♔d4 ♖d5+ 38 ♔e3 ♖e5+ 39 ♔d4 ♖d5+ 40 ♔e3 ♖e5+ 41 ♔f4 ♖h5 42 ♔e4 ♖h4+ 43 ♔e5 ♖h5+ 44 ♔e4 ♖h4+ 45 ♔e5 ♖h5+ 46 ♔e6 ♖h6+ 47 ♔e7 ♖h7+ 48 ♔e6 ♖h6+ 49 ♔e7 ♖h7+ 50 ♔f6 ♖h6+ 51 ♔f5 ♔c5 52 ♔e5 ♖h5+ 53 ♔e4 ♖h4+ 54 ♔e5 ♖h5+ 55 ♔e4 ♖h4+ 56 ♔e3 b4 57 c×b4+ ♖×b4 58 ♖c2+ ♔d5 59 ♖d2+ ♔e5 60 ♖c2 ♖b3+ 61 ♔e2 ♖h3 62 ♔e1 ♔e4 63 ♖c4+ ♔d5 64 ♖c7 b5 65 ♖d7+ ♔e4 66 ♖e7+ ♔d4 67 ♖d7+ ♔e4 68 ♖e7+ ♔d4 69 ♖e2 b4 70 ♔d2 ♖d3+ 71 ♔c2 b3+ 72 ♔c1 ♖h3 73 ♔d2 ♖d3+ 74 ♔e1 ♖h3 75 ♔d2 ♖d3+ 76 ♔e1 ♖h3 77 ♔d1 ♔d3 78 ♖f2 ♔d4 79 ♔d2 ♖d3+ 80 ♔e2 ♖e3+ 81 ♔d2 ♖d3+ 82 ♔e2 ♖e3+ 83 ♔d1 ♖h3 84 ♖f4+ ♔e5 85 ♖b4 ♖d3+ 86 ♔e2 ♖h3 87 ♔d2 ♖×h2+ 88 ♔c3 ♖c2+ 89 ♔×b3 ♖c6 90 ♖g4 ♖b6+ 91 ♔c3 ♔f5 92 ♖d4 ♔e5 93 b4 ♖c6+ 94 ♔d3 ♖b6 95 ♖e4+ ♔d5 96 ♖f4 ♔e5 97 ♖e4+ ♔d5 98 ♖f4 ♔e5 99 ♔e3 ♖b5 100 ♖c4 ♔e6 101 ♔e4 ♖e5+ 102 ♔d4 ♖d5+ 103 ♔e4 ♖e5+ 104 ♔d4 ♖d5+ 105 ♔e3 ♔d6 106 ♔e4 ♖e5+ 107 ♔d4 ♖d5+ 108 ♔e4 ♖e5+ 109 ♔d4 ♖d5+ 110 ♔e3 ♔e6 111 ♔f4 ♖f5+ 112 ♔e3 ♖d5 113 ♔f4 ♖f5+ 114 ♔g4 ♖d5 115 ♖c6+ ♔e5 116 ♔f3 ♖d3+ 117 ♔e2 ♖b3 118 ♖c4 ♖b2+ 119 ♔e3 ♖b3+ 120 ♔d2 ♔d5 121 ♖f4 ♔e5 122 ♖c4 ♔d5 123 ♖f4 ♔e5 124 ♖g4 ♖b2+ 125 ♔e3 ♖b3+ 126 ♔e2 ♖b2+ 127 ♔d3 ♔f5 128 ♖e4 ♖b3+ 129 ♔c2 ♔×e4 130 ♔×b3 ♔d4 131 b5 ♔c5 132 ♔a4 ♔b6 133 ♔b4 ♔c7 134 ♔c5 ♔c8 135 ♔c6 ♔d8 136 ♔b7 ♔d7 137 b6 ♔e7 138 ♔c7 ♔e8 139 b7 ♔f7 140 b8(♕) ♔f6 141 ♔d6 ♔g6 142 ♔e6 ♔h6 143 ♔f6 ♔h7 144 ♕e8 ♔h6 145 ♕g6# 1-0

**Coko III – Ostrich** 1 d4 ♘f6 2 ♘f3 d5 3 ♘c3 ♘c6 4 ♘e5 e6 5 ♘×c6 b×c6 6 ♗g5 ♗b4 7 ♕d2 0-0 8 ♗×f6 g×f6 9 a3 ♗×c3 10 ♕×c3 ♕d6 11 ♕f3 e5 12 d×e5 f×e5 13 e4 f5 14 e×f5 ♗×f5 15 ♕g3+ ♕g6 16 c4 ♖ab8 17 ♕×e5 ♖fe8 18 ♕×e8+ ♕×e8+ 19 ♗e2 ♖×b2 20 0-0 ♕×e2 21 c×d5 c×d5 22 h4 c5 23 ♖ae1 ♕h5 24 ♖e3 ♕×h4 25 ♖e8+ ♔f7 26 g3 ♕d4 27 ♖e3 c4 28 ♖f3 ♔g6 29 ♖c1 ♕d2 30 ♖a1 c3 31 ♖f4 c2 32 ♔g2 ♖b1 33 ♖a2 c1(♕) 34 ♖h4 ♕g1+ 35 ♔f3 ♖b3# 0-1

**ACM 1973 – Atlanta**

| | rate | perf | 1 | 2 | 3 | 4 | tot |
|---|---|---|---|---|---|---|---|
| 1 Chess 4.0 | 1579 | 1664 | 7+ | 5= | 8+ | 2+ | 3½ |
| 2 Chaos | 1477 | 1482 | 12+ | 10+ | 6+ | 1– | 3 |
| 3 Ostrich | 1300 | 1362 | 8– | 11+ | 10+ | 6+ | 3 |
| 4 Tech II | 0 | 1417 | 10– | 12+ | 5+ | 9+ | 3 |
| 5 Dartmouth CP | 0 | 1388 | 9+ | 1= | 4– | 8= | 2 |
| 6 Tech | 1243 | 1267 | 11+ | 8+ | 2– | 3– | 2 |
| 7 Belle | 1200 | 1200 | 1– | 9= | 12= | 10+ | 2 |
| 8 Coko IV | 1300 | 1285 | 3+ | 6– | 1– | 5= | 1½ |
| 9 Ga Tech CP | 0 | 1167 | 5– | 7= | 11+ | 4– | 1½ |
| 10 The Fox | 0 | 1150 | 4+ | 2– | 3– | 7– | 1½ |
| 11 USC CP | 0 | 959 | 6– | 3– | 9– | 12+ | 1 |
| 12 Ches | 0 | 882 | 2– | 4– | 7= | 11– | ½ |

*Round 1, August 26, 1973*

**Chess 4.0 – Belle** 1 e4 e5 2 ♘f3 ♘c6 3 d4 e×d4 4 c3 d×c3 5 ♗c4 c×b2 6 ♗×b2 ♗b4+ 7 ♘c3 d6 8 ♕b3 ♗×c3+ 9 ♗×c3 ♕e7 10 ♗×g7 ♕×e4+ 11 ♗e2 ♘d4 12 ♗×d4 f6 13 ♕b2 ♕f5 14 0-0 h5 15 ♘h4 ♕g5 16 g3 ♖h7 17 f4 ♕d5 18 ♗f3 ♕c4 19 ♗×f6 ♘×f6 20 ♕×f6 ♕c5+ 21 ♔h1 ♕c2 22 ♖fe1+ ♗e6 23 ♕×e6+ ♔f8 24 ♗×b7 d5 25 ♕f6+ ♔g8 26 ♗×d5+ ♖f7 27 ♕×f7+ ♔h8 28 ♕f6+ 1-0

**Coko IV – Ostrich** 1 d4 ♘f6 2 c4 e6 3 ♘d2 ♗e7 4 e4 0-0 5 e5 ♘e8 6 ♘gf3 d6 7 ♗e2 d×e5 8 d×e5 ♘c6 9 0-0 ♗c5 10 ♘b3 ♕×d1 11 ♖×d1 ♗b4 12 ♗e3 ♗e7 13 ♘c5 b6 14 ♘d7 ♗×d7 15 ♖×d7 ♖d8 16 ♖ad1 a5 17 a4 h5 18 ♘e1 ♖×d7 19 ♖×d7 h4 20 ♗f3 ♘×e5 21 ♖×e7 ♘×c4 22 ♗c6 ♘×e3 23 f×e3 ♘d6 24 ♖×c7 ♘c4 25 ♘f3 ♘×b2 26 ♘×h4 g5 27 ♘f3 f6 28 ♖b7 ♖c8 29 ♗d7 ♖c1+ 30 ♔f2 ♘d3+ 31 ♔g3 ♘c5 32 ♖b8+ ♔f7 33 ♗e8+ ♔e7 34 h4 ♘e4+ 35 ♔h3 ♖h1+ 36 ♔g4 f5+ 37 ♔h5 g4 38 ♗c6 g×f3 39 g×f3 ♘f6+ 40 ♔g5 ♖g1+ 41 ♔h6 ♘g8+ 42 ♔h7 ♘f6+ 43 ♔h6 ♖b1 44 ♖b7+ ♔d6 45 ♗b5 ♔c5 46 ♖c7+ ♔b4 47 ♖c6 ♖e1 48 ♖×e6 ♘d5 49 e4 f×e4 50 ♖×e4+ ♖×e4 51 f×e4 ♘c3 52 e5 ♘×b5 53 a×b5 ♔×b5 54 e6 ♔c6 55 ♔g6 ♔d6 56 ♔f7 a4 57 e7 ♔c5 58 ♔e6 ♔b4 59 e8(♕) b5 60 ♕f8+ ♔b3 61 h5 a3 62 ♕f3+ ♔c4 63 ♕×a3 b4 64 ♕×b4+ ♔×b4 65 h6 ♔c5 66 h7 ♔c4 67 h8(♕) ♔d3 68 ♕e5 ♔c2 69 ♕e3 ♔b2 70 ♕d3 ♔a2 71 ♕c3 ♔b1 72 ♔d5 ♔a2 73 ♔c4 ♔b1 74 ♕d2 ♔a1 75 ♔b3 ♔b1 76 ♕b2# 1-0

**Tech – USC CP** 1 e4 e5 2 ♘f3 ♘f6 3 d4 ♘×e4 4 ♗d3 d5 5 ♘×e5 f6 6 ♕h5+ ♔e7 7 ♘g6+ h×g6 8 ♕×h8 ♔f7 9 0-0 c5 10 ♗×e4 d×e4 11 d×c5 ♘a6 12 ♗e3 b6 13 ♘c3 b×c5 14 ♘×e4 ♕b6 15 b3 ♗f5 16 f3 ♘b4 17 ♖f2 ♖e8 18 ♕h4 g5 19 ♕h5+ ♗g6 20 ♕g4 ♔g8 21 ♕d7 a6 22 c4 ♖d8 23 ♕h3 ♘d3 24 ♖e2 ♖e8 25 ♕d7 ♘b4 26 ♖d1 ♘c6 27 ♖d5 ♘b8 28 ♕a4 ♖c8 29 ♔h1 ♗f7 30 ♖f5 ♗e6 31 ♗×c5 ♕c7 32 ♘×f6+ g×f6 33 ♖×e6 ♗×c5 34 ♖f×f6 ♗e7 35 ♖f5 ♖d8 36 ♖d5 ♖f8 37 c5 ♔f7 38 ♕e4 ♖c8 39 ♖×e7+ ♕×e7 40 ♖f5+ ♔e8 41 ♖e5 ♔d8 42 ♖×e7 ♖×c5 43 ♕d4+ ♔×e7 44 ♕×c5+ ♔f6 45 ♕d6+ ♔f7 46 ♕×b8 ♔e6 47 ♕b6+ ♔d5 48 ♕×a6 ♔e5 49 h3 ♔f4 50 ♕f6+ ♔g3 51 ♕×g5+ ♔f2 52 b4 ♔f1 53 b5 ♔f2 54 b6 ♔f1 55 b7 ♔f2 56 b8(♕) ♔e1 57 ♕e3+

♔f1 58 ♕b5# 1-0

**The Fox – Tech II** 1 e4 e5 2 f4 e×f4 3 ♗c4 ♘f6 4 ♘c3 ♘c6 5 ♘f3 ♘×e4 6 0-0 ♗c5+ 7 d4 ♘×c3 8 b×c3 ♗d6 9 ♕e2+ ♕e7 10 ♗d2 0-0 11 ♕×e7 ♘×e7 12 ♖fe1 ♘c6 13 ♗d3 f6 14 ♖ab1 ♖d8 15 ♖b5 g5 16 ♖f5 ♗e7 17 ♖b5 d5 18 ♖bb1 ♗d6 19 h3 ♔g7 20 ♖e2 ♖b8 21 ♔f2 g4 22 ♘g1 ♗d7 23 h×g4 ♗×g4 24 ♘f3 ♖d7 25 ♖h1 f5 26 ♖e6 1-0

**Chaos – Ches** 1 d4 d5 2 c4 e6 3 ♘c3 d×c4 4 e4 ♘c6 5 ♘f3 ♘a5 6 ♗e3 b6 7 ♘e5 f6 8 ♕h5+ ♔e7 9 ♕f7+ 1-0

**Ga Tech – Dartmouth** 1 e4 c5 2 ♘f3 d6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 a6 6 ♗c4 e6 7 0-0 b5 8 ♗d3 b4 9 ♘a4 ♕a5 10 c4 ♗d7 11 b3 ♗×a4 12 b×a4 ♘bd7 13 ♖e1 ♕e5 14 ♗e3 h5 15 ♕c2 ♘g4 16 f4 ♕c5 17 ♘×e6 ♘×e3 18 ♘×c5 ♘×c2 19 ♗×c2 d×c5 20 ♔f2 f6 21 e5 f×e5 22 f×e5 0-0-0 23 e6 ♘f6 24 e7 ♖d2+ 25 ♖e2 ♖×e2+ 26 ♔×e2 ♗×e7 27 a3 ♔c7 28 a×b4 c×b4 29 ♖b1 ♔c6 30 g3 ♔c5 31 ♗b3 ♖d8 32 ♖d1 ♖×d1 33 ♔×d1 ♗d6 34 ♔d2 ♘g4 35 h3 ♘f6 36 ♔e3 ♗×g3 37 ♔f3 ♗e5 38 a5 ♗c7 39 ♗a2 ♔d4 40 ♔e2 ♔c3 41 c5 ♘d7 42 c6 ♘b8 43 ♗d5 h4 44 ♔e3 ♗×a5 45 ♗e4 b3 46 ♗d5 ♔c2 47 ♗f3 ♗b6+ 48 ♔f4 b2 49 ♗e4+ ♔c1 50 ♔f3 ♘×c6 51 ♔e2 ♘e5 52 ♔e1 a5 53 ♔e2 a4 54 ♔e1 a3 55 ♔e2 a2 56 ♔e1 a1(♕) 57 ♔e2 ♕a6+ 58 ♔e1 ♕a5+ 59 ♔f1 ♕b5+ 60 ♔g2 ♕e2+ 61 ♔h1 ♕×e4+ 62 ♔h2 ♕e2+ 63 ♔h1 ♕f1+ 64 ♔h2 ♘f3# 0-1

*Round 2, August 26, 1973*

**Dartmouth – Chess 4.0** 1 e4 c5 2 ♘f3 ♘c6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 d6 6 ♗e3 ♘×d4 7 ♗×d4 e5 8 ♗e3 ♗e7 9 ♗b5+ ♗d7 10 ♕d3 0-0 11 0-0-0 ♘g4 12 ♗×d7 ♕×d7 13 ♘d5 ♗h4 14 f3 ♘f2 15 ♗×f2 ♗×f2 16 ♖d2 ♗c5 17 b4 ♗d4 18 c3 ♗b6 19 ♘×b6 a×b6 20 b5 ♔h8 21 ♕d5 ♖fc8 22 ♔b2 ♖c5 23 ♕×d6 ♖×b5+ 24 ♔a1 ♕c8 25 ♕d3 ♖ba5 26 ♖b1 ♕c7 27 ♖b4 ♖a3 28 ♖c4 ♕e7 29 ♖b4 ♕f6 30 ♖b5 ♖3a4 31 ♕d5 ♕h4 32 ♖×b6 ♖4a5 33 ♕d6 ♕e1+ 34 ♖b1 ♕h4 35 ♕d7 b5 36 ♕×f7 ♕×h2 37 ♕d7 ♕f4 38 ♕d8+ ♕f8 39 ♕×f8+ ♖×f8 40 ♔b2 ♖fa8 41 ♖a1 ♖5a6 42 ♖d5 ♖g6 43 g4 ♖f6 44 ♖×b5 ♖×f3 45 ♖×e5 ♖b8+ 46 ♔c2 ♖c8 47 ♔b3 ♖f×c3+ 48 ♔b4 ♖c2 49 ♖e7 ♖b2+ 50 ♔a3 ♖g2 51 e5 ♖×g4 52 e6 ♖g2 53 ♔a4 ♖b2 54 ♖a7 h5 55 a3 ♖c4+ 56 ♔a5 ♖c5+ 57 ♔a4 g5 58 e7 ♖c4+ 59 ♔a5 ♖b8 60 ♖e1 ♖c5+ 61 ♔a4 ♖cc8 62 ♖e5 ♖a8 63 ♖×a8 ♖×a8+ 64 ♔b5 ♖e8 65 ♖×g5 ♖×e7 66 ♖×h5+ ♔g7 67 ♔b6 ♖e6+ 68 ♔b5 ♖e7 69 ♔b6 ♖e6+ 70 ♔b5 ♖e7 71 ♔b6 ½-½

**Coko IV – Tech** 1 d4 d5 2 ♘d2 ♘f6 3 c4 ♘c6 4 ♘gf3 ♗f5 5 c×d5 ♕×d5 6 e3 e5 7 ♘×e5 ♘×e5 8 d×e5 ♕×e5 9 ♕b3 ♗e4 10 ♗b5+ c6 11 ♗c4 0-0-0 12 ♘×e4 ♕×e4 13 0-0 ♘d5 14 f3 ♕e5 15 f4 ♕e4 16 ♗d3 ♕e6 17 f5 ♕e5 18 ♖f3 ♗c5 19 ♗d2 ♗×e3+ 20 ♗×e3 ♘×e3 21 ♖e1 ♖×d3 22 ♕×d3 ♕×b2 23 ♖e2 ♕c1+ 24 ♔f2 ♘g4+ 25 ♔g3 ♖d8 26 ♕c2 ♕g5 27 h4 ♕h5 28 ♖e7 ♖d5 29 ♖f4 ♘h6 30 f6 g×f6 31 ♔h3 ♘f5 32 ♖e8+ ♔c7 33 a4 h6 34 g3 c5 35 ♕c4 ♕d1 36 ♖f2 h5 37 ♕f4+ ♔c6 38 ♕e4 ♔d7 39 ♖×f5 ♖×f5 40 ♖e7+ ♔d6 41 ♖d7+ ♔×d7 42 ♕×f5+ ♔e7 43 ♕×c5+ ♔e6 44 ♕e7+ ♔×e7 0-1

**The Fox – Chaos** 1 e4 c5 2 ♘f3 ♘c6 3 ♗b5 ♘f6 4 0-0 ♘×e4 5 ♘c3 ♘d6 6 d3 ♘×b5 7 ♘×b5 d6 8 ♕e2 ♗g4 9 ♗f4 e5 10 ♗g5 f6 11 ♗h4 ♗e7 12 ♘c3 0-0 13 ♘d5 ♘d4 14 ♕d1 ♘×f3+ 15 g×f3 ♗e6 16 ♘×e7+ ♕×e7 17 ♕e2 ♕f7 18 ♗g3 b6 19 ♕e3 ♗×a2 20 b3 ♗×b3 21 c×b3 ♕×b3 22 ♖a6 ♕b5 23 ♖a2 a5 24 ♕e4 ♕b3 25 ♖d2 ♖ae8 26 ♖c1 d5 27 ♕e3 a4 28 f4 e×f4 29 ♕×f4 d4 30 ♕d6 ♖e6 31 ♕d7 a3 32 ♕b7 ♖fe8 33 ♕d7 ♖e1+ 34 ♖×e1 ♖×e1+ 35 ♔g2 a2 36 ♗b8 a1(♕) 37 ♗d6 ♕d5+ 38 ♔h3 ♕e6+ 39 ♕×e6+ ♖×e6 40 ♗c7 ♕e1 41 ♖a2 ♕f1+ 42 ♔g4 ♕×d3 43 ♖a8+ ♔f7 44 ♖d8 f5+ 45 ♔h4 ♕f3 46 ♖d7+ ♔g6 0-1

**USC CP – Ostrich** 1 e4 e5 2 d4 e×d4 3 ♕×d4 ♘c6 4 ♕d1 ♘f6 5 ♗g5 ♗e7 6 ♗×f6 ♗×f6 7 c3 0-0 8 ♗e2 d6 9 ♔f1 ♗e6 10 h3 d5 11 f3 d×e4 12 ♕×d8 ♖f×d8 13 f×e4 ♘e5 14 ♔f2 ♗h4+ 15 g3 ♗e7 16 ♘f3 ♗c5+ 17 ♔g2 ♘d3 18 b4 ♘f2 19 b×c5 ♘×h1 20 ♔×h1 ♗×h3 21 ♔h2 ♗g4 22 ♘d4 ♖×d4 23 ♗×g4 ♖c4 24 c6 ♖×c6 25 ♗f5 ♖c4 26 a3 g6 27 ♗h3 ♖×e4 28 ♖a2 ♖ae8 29 ♗d7 ♖d8 30 ♗h3 c6 31 ♖b2 b5 32 ♗g2 ♖e6 33 g4 f6 34 ♗f3 ♖d3 35 ♗h1 a5 36 ♗g2 ♖de3 37 ♗h1 ♔g7 38 ♔g2 ♖e1 39 ♔h2 ♖6e2+ 40 ♖×e2 ♖×e2+ 41 ♔g3 ♖e1 42 ♗×c6 ♖×b1 43 ♔f4 ♖b3 44 a4 b×a4 45 c4 a3 46 ♗e8 a2 47 ♗a4 ♖c3 48 ♗b5 a1(♕) 49 ♗a6 ♕f1+ 50 ♔e4 ♕d3+ 51 ♔f4 ♕f3# 0-1

**Ches – Tech II** 1 e4 e5 2 ♘f3 ♘c6 3 ♗c4 ♘f6 4 d3 d5 5 e×d5 ♘×d5 6 ♕e2 ♗g4 7 ♗×d5 ♕×d5 8 c4 ♗b4+ 9 ♔f1 ♕d6 10 ♘c3 0-0 11 ♕e4 f5 12 ♕e3 ♘d4 13 ♖b1 ♘×f3 14 g×f3 ♗h3+ 15 ♔e2 ♗g2 16 ♖g1 ♕g6 17 ♘b5 f4 18 ♕e4 ♗×f3+ 19 ♔×f3 ♕×g1 20 ♕d5+ ♖f7 21 ♘×c7 ♕d1+ 22 ♔g2 f3+ 23 ♔g3 ♕g1+ 24 ♔h4 ♗e7+ 25 ♗g5 ♕×g5+ 0-1

**Belle – Ga Tech** 1 e4 e5 2 ♘c3 ♘f6 3 ♗c4 ♘c6 4 f4 ♘×e4 5 ♘f3 ♘×c3 6 d×c3 d6 7 0-0 ♗e6 8 ♗×e6 f×e6 9 f×e5 ♘×e5 10 ♘×e5 d×e5 11 ♕h5+ ♔d7 12 ♖d1+ ♗d6 13 ♕×e5 ♕f8 14 ♗g5 h6 15 ♗h4 a5 16 ♖d3 a4 17 ♖f1 c5 18 ♖×f8 ♖h×f8 19 ♖×d6+ ♔c8 20 ♕×c5+ ♔b8 21 ♖×e6 ♖a6 22 ♗g3+ ♔a8 23 ♕×f8+ ♔a7 24 ♗b8+ ♔a8 25 ♗c7+ ♔a7 26 ♗b8+ ½-½

*Round 3, August 27, 1973*

**Tech – Chaos** 1 e4 c5 2 ♘f3 ♘c6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 e6 6 ♗c4 ♗b4 7 ♗g5 ♗×c3+ 8 b×c3 0-0 9 0-0 ♘e5 10 ♗×f6 ♕×f6 11 ♕e2 ♕f4 12 f3 ♘×c4 13 ♕×c4 ♕e3+ 14 ♖f2 b6 15 ♖d1 a5 16 ♕b5 ♕×c3 17 ♕×b6 ♕c4 18 a3 ♗a6 19 ♕×a5 d5 20 ♕b4 d×e4 21 f×e4 e5 22 ♘f5 ♖fb8 23 ♘e7+ ♔h8 24 ♕×c4 ♗×c4 25 ♘d5 f6 26 a4 ♖×a4 27 c3 ♖a2 28 ♖c1 ♖bb2 29 ♖×b2 ♖×b2 30 h3 ♗d3 31 c4 ♗×e4 32 g4 ♖g2+ 33 ♔f1 ♖h2 34 ♖c3 ♖h1+ 35 ♔f2 ♖h2+ 36 ♔g3 ♖g2+ 37 ♔h4 g5+ 38 ♔h5 ♔g7 39 ♘e7 ♗g6+ 40 ♘×g6 h×g6# 0-1

**Chess 4.0 – Coko IV** 1 e4 e5 2 ♘f3 ♘f6 3 d4 e×d4 4 e5 ♕e7 5 ♕×d4 ♘c6 6 ♕c3 ♘e4 7 ♕c4 ♘f6 8 ♕e2 ♘d5 9 ♗g5 ♕e6 10 c4 ♘b6 11 a4 ♗b4+ 12 ♘bd2

0-0 13 g3 f6 14 e×f6 g×f6 15 ♗h6 ♖e8 16 ♕×e6+ ♖×e6+ 17 ♗e3 d5 18 c×d5 ♘×d5 19 ♗c4 ♖d6 20 0-0 ♗e6 21 ♗h6 f5 22 ♖ae1 ♗f7 23 ♘g5 ♗h5 24 ♘e6 ♖c8 25 ♗b3 ♗f7 26 ♘g5 ♗×d2 27 ♘×f7 ♔×f7 28 ♗×d2 ♘d4 29 ♗d1 c5 30 ♗h5+ ♔f8 31 ♗g5 h6 32 ♗h4 ♔g7 33 ♖e5 a6 34 ♖fe1 b5 35 a5 f4 36 ♗e7 ♘×e7 37 ♖×e7+ ♔h8 38 ♖e8+ ♖×e8 39 ♖×e8+ ♔g7 40 g×f4 ♘c6 41 ♖c8 ♔f6 42 ♗f3 ♘×a5 43 ♖×c5 ♖d3 44 ♗g4 ♘b7 45 ♖c6+ ♖d6 46 ♖c7 ♘a5 47 ♔g2 ♘c4 48 b4 1-0

**Tech II – Dartmouth** 1 e4 c5 2 ♘f3 d6 3 ♘c3 e6 4 d4 ♗d7 5 d×c5 d×c5 6 ♘e5 ♕h4 7 ♗e3 ♘f6 8 ♘b5 ♘a6 9 ♘×d7 ♘×d7 10 ♕f3 ♘e5 11 ♕f4 ♕×f4 12 ♗×f4 ♘g4 13 ♘d6+ ♗×d6 14 ♗×d6 ♘b4 15 ♗b5+ ♔d8 16 0-0 ♘×c2 17 ♖ad1 ♘d4 18 ♗×c5 e5 19 ♖d3 a6 20 ♗c4 ♖c8 21 ♗×d4 ♖×c4 22 ♗×e5+ ♔c8 23 ♗×g7 ♖g8 24 ♗d4 ♖a4 25 ♖c1+ ♔b8 26 h3 ♖d8 27 ♗e5+ ♘×e5 28 ♖×d8+ ♔a7 29 ♖c3 ♖×a2 30 f4 ♘c6 31 ♖d7 ♔b6 32 ♖b3+ ♔c5 33 ♖b×b7 ♖a1+ 34 ♔f2 f6 35 ♖d5+ ♔c4 36 b3+ ♔c3 37 ♖c5+ ♔d3 38 ♖×c6 ♖a5 39 ♖c4 ♖a1 40 ♖d7# 1-0

**Ostrich – The Fox** 1 ♘f3 ♘f6 2 ♘c3 d5 3 d4 ♘c6 4 ♗f4 ♗f5 5 ♘b5 ♖c8 6 ♘×c7+ ♖×c7 7 ♗×c7 ♕×c7 8 e3 ♕d6 9 ♗d3 ♕b4+ 10 c3 ♕×b2 11 ♗×f5 ♕×c3+ 12 ♘d2 e6 13 ♗c2 ♗a3 14 0-0 0-0 15 ♖b1 ♗b2 16 a4 ♕a3 17 ♗b3 ♗c3 18 ♕c2 ♗a5 19 ♖a1 ♕d6 20 f3 ♗b4 21 ♖a2 ♘a5 22 ♖f2 ♖e8 23 ♖b2 ♘×b3 24 ♕×b3 ♗×d2 25 ♖f×d2 ♕c7 26 ♖dc2 ♕d7 27 ♕×b7 ♕×b7 28 ♖×b7 ♖a8 29 ♖×a7 ♖×a7 30 ♖c8+ ♘e8 31 ♖×e8# 1-0

**Belle – Ches** 1 e4 e5 2 ♘c3 ♕h4 3 ♘f3 ♕h5 4 g4 ♕×g4 5 ♘×e5 ♕×d1+ 6 ♘×d1 ♘f6 7 ♗c4 d5 8 ♗×d5 ♘×d5 9 e×d5 ♗c5 10 0-0 ♗h3 11 ♖e1 ♘a6 12 ♘g6+ ♔d7 13 ♘×h8 ♘b4 14 ♘×f7 ♘×c2 15 b4 ♗×b4 16 ♖e3 ♘×e3 17 ♘×e3 ♖f8 18 ♘g5 ♗f5 19 a3 ♗d6 20 ♗b2 h6 21 ♗×g7 ♖e8 22 ♘×f5 ♖e1+ 23 ♖×e1 h×g5 24 ♘×d6 ♔×d6 25 ♖e5 g4 26 d4 c6 27 ♗f8+ ♔d7 28 ♖e7+ ♔d6 29 ♖e5+ ♔d7 30 ♖e7+ ♔d6 31 ♖e5+ ½-½

**Ga Tech – USC CP** 1 e4 e5 2 ♘f3 ♘f6 3 d4 ♘×e4 4 d×e5 ♗c5 5 ♗e3 ♗×e3 6 f×e3 ♘c6 7 ♘bd2 ♘×d2 8 ♕×d2 ♘×e5 9 ♘×e5 d6 10 ♘f3 0-0 11 0-0-0 ♖e8 12 ♗c4 ♗g4 13 h3 ♗d7 14 ♕d5 ♖b8 15 ♕×f7+ ♔h8 16 ♖he1 a6 17 e4 ♖f8 18 ♕h5 b5 19 ♗d5 ♕e7 20 e5 ♖f5 21 ♕g4 d×e5 22 ♕e4 ♕d6 23 ♗b3 ♕h6+ 24 ♔b1 ♗c6 25 ♕×f5 ♗×f3 26 ♕×f3 ♕g5 27 ♕c6 a5 28 ♖d7 b4 29 ♕d5 ♖e8 30 ♗c4 a4 31 ♕b5 c6 32 ♕×c6 a3 33 ♖b7 ♖f8 34 ♕d5 ♖d8 35 ♕e4 ♕d2 36 ♗d3 ♕×e1+ 37 ♕×e1 ♖e8 38 ♕e4 g6 39 ♕c6 1-0

*Round 4, August 28, 1973*

**Chaos – Chess 4.0** 1 d4 d5 2 c4 d×c4 3 ♘f3 ♘f6 4 e3 e6 5 ♗×c4 c5 6 0-0 a6 7 ♕e2 b5 8 ♗b3 ♗b7 9 ♘c3 ♘bd7 10 ♗d2 ♗d6 11 ♖fe1 c×d4 12 e×d4 0-0 13 ♗g5 ♕b6 14 ♗c2 ♗×f3 15 ♕×f3 ♕×d4 16 ♖ad1 ♕b4 17 g3 ♕×b2 18 ♖e2 ♕a3 19 ♕c6 ♗b4 20 ♖×d7 ♖ac8 21 ♖c7 ♖×c7 22 ♕×c7 ♕×c3 23 ♕b7 ♕c4 24 ♗d1 ♖d8 25 ♖e5 ♖×d1+ 26 ♔g2 ♕f1+ 27 ♔f3 ♕h1+ 28 ♔e2 ♖e1+ 29 ♔d3 ♕f1+ 30 ♔d4 ♕c4# 0-1

**Ostrich – Tech** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 ♘f6 4 0-0 ♘×e4 5 d3 ♘c5 6 ♗×c6 b×c6 7 ♘×e5 ♗e7 8 ♗e3 0-0 9 ♗×c5 ♗×c5 10 d4 ♕f6 11 a3 d5 12 ♖e1 ♖e8 13 ♘×c6 ♖×e1+ 14 ♕×e1 ♗f5 15 ♕c3 ♖e8 16 g4 ♗d7 17 d×c5 ♕×c3 18 ♘×c3 ♗×c6 19 a4 d4 20 ♘b5 ♖e5 21 ♘×d4 ♖×c5 22 b4 ♖c4 23 ♘×c6 ♖×c6 24 ♖d1 ♖d6 25 ♖×d6 c×d6 26 h4 d5 27 c3 g6 28 a5 ♔g7 29 f4 ♔f6 30 ♔f2 a6 31 ♔e3 ♔e6 32 h5 f6 33 ♔d4 ♔d6 34 h6 g5 35 ♔e3 ♔e6 36 f5+ ♔d6 37 ♔f3 d4 38 c4 ♔e5 39 b5 a×b5 40 c×b5 d3 41 a6 d2 42 ♔e2 d1(♕)+ 43 ♔×d1 ♔e4 44 a7 ♔d5 45 a8(♕)+ ♔c5 46 ♕c6+ ♔d4 47 b6 ♔e5 48 b7 ♔f4 49 b8(♕)+ ♔×g4 50 ♕e4+ ♔h3 51 ♕f3+ ♔h4 52 ♕bg3# 1-0

**Tech II – Ga Tech** 1 e4 e5 2 ♘f3 ♘c6 3 ♘c3 ♗c5 4 ♘×e5 ♗×f2+ 5 ♔×f2 ♘×e5 6 d4 ♘c6 7 ♗c4 ♕f6+ 8 ♔e3 ♕g5+ 9 ♔f3 ♕h5+ 10 ♔f2 ♘f6 11 ♕×h5 ♘×h5 12 ♘b5 ♔d8 13 ♗×f7 ♘f6 14 ♗f4 ♘×e4+ 15 ♔e3 ♘d6 16 ♘×d6 c×d6 17 ♗×d6 b6 18 ♖hf1 g5 19 ♖f6 ♗a6 20 ♖d1 g4 21 c4 ♔c8 22 ♗d5 ♔b7 23 ♗e5 ♖hf8 24 ♖d6 ♖ad8 25 ♖h6 b5 26 c×b5 ♗×b5 27 ♖×h7 ♔b6 28 ♗d6 ♖f5 29 ♗c5+ ♔b7 30 ♗e6 ♖g5 31 ♗×d7 g3 32 h4 ♖×c5 33 d×c5 ♖f8 34 ♖d6 ♖f1 35 ♗×c6+ ♔a6 36 a4 ♗×c6 37 b4 ♖e1+ 38 ♔f4 ♖e4+ 39 ♔×g3 ♖e3+ 40 ♔f2 ♖e6 41 b5+ ♔a5 42 ♖×e6 ♗d5 43 ♖×a7+ ♔b4 44 b6 ♗×e6 45 b7 ♔×c5 46 b8(♕) ♔d5 47 ♔e3 1-0

**Dartmouth – Coko IV** 1 e4 e5 2 f4 e×f4 3 ♗c4 ♘c6 4 ♕h5 ♘h6 5 ♘f3 g6 6 ♕b5 a6 7 ♕b3 ♗c5 8 d3 g5 9 ♗d5 ♗e3 10 ♗×c6 ♗×c1 11 ♗d5 c6 12 ♗c4 g4 13 ♕c3 0-0 14 ♘bd2 g×f3 15 g×f3 ♗×d2+ 16 ♔×d2 d6 17 ♖ag1+ ♘g4 18 f×g4 ♗e6 19 g5 d5 20 e×d5 ♗×d5 21 ♗×d5 ♕×d5 22 a3 f3 23 ♔e3 ♖ae8+ 24 ♔f4 f2 25 ♖c1 f5 26 ♔g3 ♖e2 27 ♕c4 ♕f7 28 ♕f4 ♖d8 29 ♖hf1 ♕h5 30 h4 b5 31 ♖×f2 ♖×f2 32 ♔×f2 ♕g4 33 ♕×g4 f×g4 34 ♔g3 ♖d4 35 b4 ♔g7 36 ♖f1 a5 37 b×a5 ♖a4 38 ♖f6 ♖×a5 39 ♖×c6 ♖×a3 40 ♔×g4 ♖a4+ 41 ♔h5 ♖d4 42 ♖c7+ ♔g8 43 ♖c8+ ♔f7 44 ♖c7+ ♔g8 45 ♖c8+ ♔f7 46 ♖c7+ ½-½

**Belle – The Fox** 1 e4 e5 2 ♘c3 ♘f6 3 ♗c4 ♘×e4 4 ♕h5 ♘d6 5 ♗b3 ♘c6 6 d4 e×d4 7 ♘d5 a6 8 ♘f3 ♘e4 9 0-0 g6 10 ♕g4 ♘c5 11 ♖e1+ ♘e6 12 ♗f4 h5 13 ♕h3 ♗d6 14 ♗g5 ♗e7 15 ♗×e7 ♘×e7 16 ♘×d4 ♘×d5 17 ♗×d5 c6 18 ♗×e6 d×e6 19 ♕d3 ♕d5 20 c4 ♕d6 21 ♖ad1 0-0 22 ♖e3 ♗d7 23 ♕c2 ♕f4 24 ♘b5 ♗e8 25 ♘d6 ♖b8 26 ♕c3 ♗d7 27 a4 h4 28 ♘f5 ♕×f5 29 ♖×d7 1-0

**USC CP – Ches** 1 e4 e5 2 d4 ♗b4+ 3 c3 ♗d6 4 d×e5 ♗×e5 5 f4 ♗d6 6 ♗c4 ♕h4+ 7 ♔f1 ♗×f4 8 ♘f3 ♕g4 9 h3 ♕g3 10 ♗×f4 ♕×f4 11 ♕d4 ♘f6 12 e5 ♕c1+ 13 ♘e1 ♕×b2 14 e×f6 ♘c6 15 ♕e3+ ♔f8 16 f×g7+ ♔×g7 17 ♕g5+ ♔f8 18 ♕c5+ d6 19 ♕f2 ♕×f2+ 20 ♔×f2 ♗f5 21 g4 ♗e4 22 ♖f1 ♔e7 23 ♔e3 d5 24 ♗b3 ♔e6 25 ♗c2 ♗×c2 26 ♘×c2 ♘e5 27 ♘ba3 ♘c4+ 28 ♘×c4 d×c4 29 ♔d4 b5 30 ♔c5 a6 31 a4 b×a4 32 ♔×c4 c5 33 ♔×c5 ♖hc8+ 34 ♔d4 ♖d8+ 35 ♔e3 ♖dc8 36 ♔d2 ♖d8+ 37 ♘d4+ ♖×d4+ 38 c×d4 ♖d8 39 ♖×a4 ♖d6 40 ♔c3 ♖c6+ 41 ♔d3 f6 42 ♖aa1 ♔d5 43 ♖f5+ ♔d6 44 ♖×f6+ ♔d5 45 ♖a5+ 1-0

*2nd Place Playoff, August 1973*

**Chaos – Tech II** 1 d4 d5 2 c4 d×c4 3 ♘f3 ♘c6 4 ♘c3 ♗d7 5 e4 ♘a5 6 ♘e5 b5 7 ♘×d7 ♕×d7 8 ♕h5 c6 9 ♗e3 ♘f6 10 ♕d1 0-0-0 11 b4 ♘b7 12 a4 e5 13 d×e5 ♗×b4 14 ♕×d7+ ♖×d7 15 ♗d2 ♖×d2 16 ♔×d2 ♘×e4+ 17 ♔e3 ♗×c3 18 ♖d1 ♘bc5 19 f4 b×a4 20 ♗×c4 ♖f8 21 ♔e2 a3 22 g3 ♔c7 23 ♖hf1 ♖b8 24 ♗×f7 ♖b2+ 25 ♔e3 a2 26 ♖a1 ♖×h2 27 ♖×a2 ♖×a2 28 ♗×a2 ♘×g3 29 ♖c1 ♘ge4 30 ♖h1 h6 31 e6 ♔d6 32 f5 ♔e5 33 e7 ♘d6 34 ♔e2 ♘ce4 35 ♖g1 ♗d4 36 ♖×g7 ♘c3+ 37 ♔d2 ♘×a2 38 ♖g6 ♘c3 39 ♖×h6 ♘ce4+ 40 ♔d3 ♘f6 31 ♖h8 c5 42 ♖a8 c4+ 43 ♔c2 ♔×f5 44 ♖h8 ♔e6 45 ♖h4 ♘f5 46 ♖f4 a5 47 ♔b1 ♔e5 48 ♖g4 c3 49 ♖g2 ♘×e7 50 ♔c2 a4 51 ♖e2+ ♔d6 52 ♔d3 ♘c6 53 ♔c4 ♘d5 54 ♔b5 a3 55 ♔a4 ♗c5 56 ♖h2 c2 57 ♖×c2 ♘b6+ 58 ♔b3 ♘d4+ 59 ♔c3 ♘a4+ 60 ♔c4 ♘×c2 61 ♔b3 ♘d4+ 62 ♔c4 a2 63 ♔d3 a1(♕) 64 ♔e4 ♕f1 65 ♔e3 ♘b3+ 66 ♔e4 ♘d2# 0-1

**Ostrich – Chaos** 1 d4 ♘f6 2 ♘f3 e6 3 e3 d5 4 ♘c3 ♗b4 5 a3 ♗×c3+ 6 b×c3 ♘e4 7 ♗d2 0-0 8 h4 ♘c6 9 c4 ♗d7 10 ♗b4 ♘×b4 11 a×b4 ♕e7 12 c5 ♖fe8 13 c4 ♗c6 14 b5 ♗d7 15 ♕a4 c6 16 b×c6 ♗×c6 17 ♕c2 b6 18 c×d5 ♗×d5 19 c×b6 ♕b4+ 20 ♘d2 ♖ec8 21 ♕d3 ♘×d2 22 ♕×d2 ♕×b6 23 h5 a5 24 h6 ♕b4 25 ♕×b4 a×b4 26 ♖b1 ♗e4 27 ♖d1 b3 28 h×g7 b2 29 f3 ♗c2 30 ♔f2 ♗×d1 31 ♗d3 ♗c2 32 ♖b1 ♗×b1 0-1

**Tech II – Ostrich** 1 e4 e5 2 ♘f3 ♘c6 3 ♘c3 ♗b4 4 d4 e×d4 5 ♘×d4 ♕h4 6 ♕d3 ♘e5 7 ♕e3 ♘g4 8 ♕g3 ♕×g3 9 h×g3 ♗×c3+ 10 b×c3 ♘8f6 11 ♗f4 ♘×e4 12 f3 c5 13 ♘b5 ♘e3 14 ♘c7+ ♔d8 15 ♗×e3 ♔×c7 16 f×e4 d6 17 ♗b5 a6 18 ♗c4 ♗e6 19 ♗×e6 f×e6 20 ♔e2 ♔c6 21 ♔d3 e5 22 ♔c4 b5+ 23 ♔b3 c4+ 24 ♔b4 a5+ 25 ♔a3 h5 26 ♖ad1 a4 27 ♖d5 ♖ad8 28 ♔b4 ♖b8 29 ♗g5 ♖he8 30 ♖×h5 ♖b7 31 ♗e3 ♖a8 32 a3 g6 33 ♖h1 ♖d8 34 ♖f1 g5 35 ♗×g5 ♖g8 36 ♖f5 ♖g6 37 g4 ♖g8 38 g3 ♖gg7 39 ♗e3 ♖×g4 40 ♖f6 ♔c7 41 ♖d×d6 ♔c8 42 ♔c5 ♖×e4 43 ♖f8+ ♔c7 44 ♗g5 ♖e2 45 ♖c6+ ♔d7 46 ♖d8# 1-0

**ACM 1974 – San Diego**

| | rate | perf | 1 | 2 | 3 | 4 | tot |
|---|---|---|---|---|---|---|---|
| 1 Ribbit | 0 | 2150 | 10+ | 7+ | 8+ | 2+ | 4 |
| 2 Chess 4.0 | 1579 | 1754 | 6+ | 5+ | 3+ | 1– | 3 |
| 3 Chaos | 1477 | 1511 | 4+ | 9+ | 2– | 6+ | 3 |
| 4 Belle | 1200 | 1363 | 3– | 10+ | 11+ | 5+ | 3 |
| 5 Duchess | 1374 | 1251 | 8+ | 2– | 9+ | 4– | 2 |
| 6 Dart 4.1 | 0 | 1327 | 2– | 11+ | 7+ | 3– | 2 |
| 7 Ostrich | 1300 | 1227 | 12+ | 1– | 6– | 9+ | 2 |
| 8 Tech II | 0 | 1110 | 5– | 12+ | 1– | – | 2 |
| 9 Chute 1 | 0 | 928 | 11+ | 3– | 5– | 7– | 1 |
| 10 Tyro | 0 | 727 | 1– | 4– | – | – | 0 |
| 11 Xenarbor | 0 | 548 | 9– | 6– | 4– | | 0 |
| 12 Kches6 | 0 | 701 | 7– | 8– | – | – | 0 |

*Round 1, November 10, 1974*

**Dart 4.1 – Chess 4.0** 1 e4 e5 2 f4 e×f4 3 ♘f3 g5 4 ♘c3 g4 5 ♘e5 ♕h4+ 6 g3 f×g3 7 ♕×g4 g2+ 8 ♕×h4 g×h1(♕) 9 ♕h5 ♗b4 10 ♘×f7 ♘f6 11 ♘d6+ ♔d8 12 ♘f7+ ♔e7 13 ♘d5+ ♘×d5 14 ♘×h8 ♕×e4+ 15 ♗e2 ♘f4 16 ♕g5+ ♔f8 17 ♕g4 ♗d6 18 ♕f3 ♕×e2+ 19 ♕×e2 ♘×e2 29 ♔×e2 ♗×h2 21 d3 ♗e5 22 ♗h6+ ♔e7 23 ♗g5+ ♔e6 24 ♖f1 ♗×h8 25 ♖f8 ♗×b2 26 ♖×c8 ♗e5 27 ♖e8+ ♔d6 28 ♖f8 h5 29 c4 c6 30 a4 b6 31 ♖f7 ♘a6 32 ♔e3 ♘c5 33 a5 ♖g8 34 ♗h4 b×a5 35 ♔d2 ♖g2+ 36 ♔e3 ♗d4+ 37 ♔×d4 ♖g4+ 38 ♔c3 ♖×h4 39 ♖f6+ ♔e5 40 ♖f3 a4 41 ♖e3+ ♔d6 42 ♔d2 a3 43 ♖g3 a2 44 ♖g1 a1(♕) 45 ♖×a1 ♘b3+ 46 ♔c3 ♘×a1 47 ♔b2 ♖h1 48 ♔c3 ♔e5 49 c5 ♖h2 50 ♔c4 a5 51 d4+ ♔e4 52 ♔c3 ♖c2# 0-1

**Ribbit – Tyro** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗a4 ♘f6 5 0-0 ♗e7 6 ♕e2 b5 7 ♗b3 0-0 8 c3 ♗c5 9 d3 d6 10 h4 h6 11 ♘a3 ♗g4 12 ♗e3 ♖b8 13 ♗×c5 d×c5 14 ♕e3 ♕d6 15 ♔h2 ♖fd8 16 ♖fd1 ♕e7 17 ♔g3 b4 18 ♘c4 b×c3 19 b×c3 ♗×f3 20 ♕×f3 ♖b7 21 ♖ab1 ♖bb8 22 ♗a4 ♖×b1 23 ♖×b1 ♘b8 24 ♔h2 ♕e6 25 a3 ♘g4+ 26 ♔h1 c6 27 ♕f5 ♔f8 28 f3 ♘f6 29 ♕×e5 ♘bd7 30 ♕×e6 f×e6 31 g4 g5 32 h×g5 h×g5 33 ♔g2 ♔e7 34 ♖b7 ♖f8 35 ♘e5 ♔d6 36 ♘×d7 ♘×d7 37 d4 c×d4 38 c×d4 e5 39 d×e5+ 1-0

**Belle – Chaos** 1 e4 c5 2 ♘f3 ♘c6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 e6 6 ♘×c6 b×c6 7 ♗f4 ♗b4 8 ♗d3 ♗b7 9 0-0 0-0 10 ♕f3 ♕a5 11 e5 ♗×c3 12 b×c3 ♘d5 13 ♗d2 ♕c5 14 ♖fe1 f5 15 e×f6 ♖×f6 16 ♕g3 ♖af8 17 ♗e3 ♘×e3 18 ♕×e3 ♕×e3 19 f×e3 e5 20 ♖f1 d5 21 ♖×f6 ♖×f6 22 ♖f1 ♖×f1+ 23 ♔×f1 ♗c8 24 ♔f2 g6 25 h4 ♔f7 26 h5 g×h5 27 ♗×h7 e4 28 g4 ♗×g4 29 ♔g3 ♔g7 30 ♗×e4 d×e4 31 ♔f4 ♗f3 32 ♔e5 h4 33 ♔d6 h3 34 ♔c5 h2 35 ♔×c6 ♔f6 36 c4 ♗g4 37 ♔b7 a5 38 c5 a4 39 c6 ♗e6 40 c7 h1(♕) 41 a3 ♕c1 42 c8(♕) ♗×c8+ 43 ♔×c8 ♕×e3 44 ♔d7 ♕×a3 45 ♔c7 ♕c5+ 46 ♔b7 ♕×c2 47 ♔b6 a3 48 ♔b7 ♕c5 49 ♔a6 a2 50 ♔b7 a1(♕) 51 ♔b8 ♕aa7# 0-1

**Tech II – Duchess** 1 e4 e5 2 ♘f3 ♘f6 3 ♘c3 ♘c6 4 d4 e×d4 5 ♘×d4 ♗b4 6 ♗g5 ♗×c3+ 7 b×c3 0-0 8 ♗d3 ♕e7 9 ♕d2 d5 10 f3 ♗d7 11 0-0-0 d×e4 12 f×e4 ♕a3+ 13 ♔b1 ♕d6 14 ♕f4 ♖fe8 15 ♕×d6 c×d6 16 ♖he1 ♗g4 17 ♖d2 ♖e5 18 ♗f4 ♖a5 19 ♗×d6 ♖e8 20 ♖e3 ♗d7 21 ♖f2 ♖a4 22 ♘×c6 ♗×c6 23 ♖fe2 b5 24 ♖f3 ♖e6 25 ♗c5 a6 26 ♖f5 g6 27 ♖f1 ♘×e4 28 ♗×e4 ♗×e4 29 ♖d2 ♗f5 30 ♔b2 h6 31 ♖d4 ♖e2 32 ♖×a4 ♖×c2+ 33 ♔b3 b×a4+ 34 ♔c4 ♗e6+ 35 ♔b4 ♖×a2 36 ♖f4 ♖×g2 37 ♖h4 ♖b2+ 38 ♔×a4 ♔g7 39 c4 ♖c2 40 ♔b3 ♖g2 41 ♗d4+ f6 42 ♖e4 ♔f7 43 h4 f5 44 ♖f4 ♔e8 45 ♔b4 g5 46 h×g5 h×g5 47 ♖f3 ♖g4 48 ♔c5 ♔e7 49 ♖c3 ♖e4 50 ♖b3 g4 51 ♖c3 f4 52 ♖b3 g3 53 ♖c3 g2 54 ♖c1 ♗d7 55 ♔d5 ♖e6 56 c5 ♗c6+ 57 ♔c4 ♖e4 58 ♖b1 ♔e6 59 ♖c1 ♗b5+ 0-1

**Kches6 – Ostrich** 1 e4 d5 2 e×d5 ♕×d5 3 c4 ♕e4+ 4 ♘e2 ♕×c4 5 b3 ♕d3 6 ♘ec3 ♕d8 7 ♗b5+ ♗d7 8 ♕f3 ♗c6 9 ♕g3 e6 10 f3 ♘h6 11 ♗c4 ♘f5 12 ♕g4 h5 13 ♕h3 ♘d4 14 ♗d3 ♘×f3+ 15 ♔f2 ♗c5+ 16 ♔e2 ♘g1+ 17 ♖×g1 ♗×g1 18 ♗e4 ♗×e4 19 ♘×e4 0-0 20 ♗a3 ♗d4 21 ♗×f8 ♗×a1 22 ♗b4 ♕d4 23

♘c5 ♕×b4 24 ♕×h5 b6 25 ♘a4 ♕e4+ 26 ♔f2 ♗d4+ 27 ♔g3 ♕×b1 28 ♕f3 f6 29 ♕g4 ♗e5+ 30 ♔f2 ♕×a2 31 ♕h5 ♗d4+ 32 ♔e1 ♕×b3 33 ♕h4 ♕×a4 34 ♕f4 a5 35 g3 b5 36 h4 f5 37 ♔e2 ♕c4+ 38 ♔e1 b4 39 ♔d1 b3 40 ♔e1 b2 41 ♔d1 b1(♕)# 0-1

**Chute 1 – Xenarbor** 1 e4 c5 2 c4 ♘c6 3 ♘f3 e5 4 d3 ♘f6 5 ♗e2 ♗e7 6 0-0 0-0 7 b3 d6 8 ♗b2 ♗e6 9 ♘g5 ♗d7 10 ♘d2 h6 11 ♘gf3 ♗e6 12 ♗c3 ♕b6 13 ♖b1 ♕a6 14 a4 ♖ac8 15 ♖a1 b6 16 ♕e1 ♖fd8 17 ♕d1 ♖f8 18 ♖b1 ♖fd8 19 ♖e1 ♖f8 20 ♔f1 ♖fd8 21 b4 ♕b7 22 b×c5 d×c5 23 ♗×e5 ♘d7 24 ♗c3 ♕a6 25 ♗b2 ♖f8 26 ♗c3 ♖fd8 27 e5 ♖f8 28 ♘e4 ♖cd8 29 ♖c1 ♖a8 30 ♕c2 ♖ad8 31 ♗d1 ♖a8 32 ♖e2 ♖ad8 33 ♕d2 ♖a8 34 ♔e1 ♖ad8 35 ♕e3 ♕c8 36 ♘d6 ♕a8 37 ♔d2 a5 38 ♕e4 g6 39 ♖e3 ♔g7 40 ♗b2 f5 41 e×f6+ ♗×f6 42 d4 ♗g8 43 ♗c2 ♘db8 44 ♕×g6+ ♔h8 45 ♕×h6+ ♗h7 46 ♕×h7# 1-0

*Round 2, November 10, 1974*

**Duchess – Chess 4.0** 1 e4 e5 2 ♘f3 ♘f6 3 d4 ♘×e4 4 ♗d3 d5 5 ♘×e5 ♗d6 6 0-0 0-0 7 c4 ♗×e5 8 d×e5 ♘c6 9 f4 ♘b4 10 c×d5 ♕×d5 11 ♗×e4 ♕×e4 12 ♘c3 ♕g6 13 ♘d5 ♘×d5 14 ♕×d5 ♕b6+ 15 ♔h1 ♗e6 16 ♕f3 ♕a6 17 b3 ♗d5 18 ♕f2 ♗×b3 19 ♗a3 ♕×a3 20 a×b3 ♕×b3 21 ♖×a7 ♖×a7 22 ♕×a7 ♕c4 23 ♖d1 ♕×f4 24 ♕×b7 ♕×e5 25 ♕c6 f5 26 ♕a4 ♕c3 27 ♕a7 c5 28 ♖f1 c4 0-1

**Ostrich – Ribbit** 1 d4 ♘f6 2 ♗g5 e6 3 ♘c3 ♗d6 4 e4 h6 5 ♗×f6 g×f6 6 ♘f3 b6 7 e5 f×e5 8 d×e5 ♗b4 9 a3 ♗×c3+ 10 b×c3 ♖g8 11 ♕d2 ♖h8 12 ♗b5 ♗a6 13 ♗×a6 ♘×a6 14 0-0 ♕e7 15 ♕d3 ♘c5 16 ♕d4 ♖g8 17 c4 0-0-0 18 ♖fd1 d6 19 h4 ♖g6 20 h5 ♖gg8 21 g3 a5 22 ♔g2 ♔b7 23 ♖ab1 d×e5 24 ♕×e5 ♖×d1 25 ♖×d1 ♖g4 26 ♖d4 ♘d7 27 ♕e3 ♖×d4 28 ♘×d4 ♕c5 29 ♕f3+ c6 30 c3 ♘e5 31 ♕f6 ♕d6 32 f4 ♘×c4 33 ♕×f7+ ♔a6 34 ♕×e6 ♕×e6 35 ♘×e6 ♘×a3 36 ♘d8 ♔b5 37 ♘f7 a4 38 ♔f2 ♔c4 39 f5 ♘b5 40 ♔e3 a3 41 ♔d2 a2 42 ♘e5+ ♔b3 43 ♘×c6 a1(♕) 44 c4 ♔×c4 45 g4 ♕c3+ 46 ♔e2 ♕d3+ 47 ♔f2 ♕d6 48 ♘b8 ♕×b8 49 ♔g2 ♕f4 0-1

**Chaos – Chute 1** 1 d4 d5 2 c4 e6 3 ♘c3 ♗b4 4 ♘f3 ♗×c3+ 5 b×c3 ♘e7 6 c×d5 ♕×d5 7 ♗f4 ♘a6 8 e3 c6 9 ♗×a6 b×a6 10 0-0 0-0 11 ♕c2 ♖d8 12 c4 ♕h5 13 ♗c7 ♖d7 14 ♗f4 f6 15 ♕e4 ♖b7 16 ♖ab1 ♖×b1 17 ♖×b1 f5 18 ♕e5 ♔h8 19 ♕d6 ♕e8 20 ♗g5 ♘g6 21 ♕d8 ♕×d8 22 ♗×d8 h5 23 ♗c7 ♗d7 24 ♖b8+ ♖×b8 25 ♗×b8 ♘e7 26 ♗×a7 ♘c8 27 ♗b8 ♔h7 28 ♗c7 g6 29 ♘e5 ♗e8 30 ♘f3 ♗d7 31 ♘e5 ♗e8 32 f3 g5 33 ♘d3 ♔h6 34 ♘c5 ♗f7 35 ♘×a6 e5 36 ♗×e5 ♗×c4 37 ♘b4 ♘a7 38 a4 h4 39 a5 g4 40 ♗b8 ♘c8 41 a6 g×f3 42 g×f3 ♗d5 43 ♘×d5 c×d5 44 ♔f2 ♔g6 45 ♔e2 ♘b6 46 h3 ♘c8 47 ♔d3 ♔f7 48 a7 ♘×a7 49 ♗×a7 ♔g8 50 ♗b6 ♔h7 51 ♗d8 ♔g6 52 ♗×h4 ♔h6 53 ♗d8 ♔h7 54 f4 ♔g6 55 h4 ♔h5 56 ♗g5 ♔g6 57 ♔c3 ♔h5 58 ♔b4 ♔g4 59 ♔c5 ♔f3 60 h5 ♔e4 61 h6 ♔f3 1-0

**Kches6 – Tech II** 1 e4 e5 2 ♘f3 ♘c6 3 ♗c4 ♘f6 4 ♘g5 d5 5 e×d5 ♘×d5 6 d4 e×d4 7 ♕f3 ♗e6 8 ♘×e6 f×e6 9 ♕e4 ♗b4+ 10 c3 d×c3 11 b×c3 ♘×c3 12 ♕g4 ♘×b1+ 13 ♔e2 ♘d4+ 14 ♔e3 ♘c2+ 15 ♔e2 ♘c3+ 16 ♔f3 ♘×a1 17 ♗g5 0-0+ 18 ♔g3 ♘e4+ 19 ♕×e4 ♕×g5+ 20 ♔h3 ♕h5+ 21 ♔g3 ♗d6+ 22 f4 ♕g5+ 23 ♔h3 ♕h6+ 24 ♔g4 ♖×f4+ 0-1

**Xenarbor – Dart 4.1** 1 d4 d5 2 c4 e6 3 ♘c3 ♘f6 4 ♗g5 ♘bd7 5 e3 c6 6 ♘f3 ♕a5 7 ♘d2 ♗b4 8 ♕c2 0-0 9 ♘b3 ♗×c3 10 b×c3 ♕b6 11 c5 ♕c7 12 ♗×f6 ♘×f6 13 0-0-0 ♗d7 14 ♗d3 b6 15 ♔b2 a5 16 ♗e2 b5 17 ♘c1 e5 18 f3 ♖fe8 19 ♘d3 e×d4 20 c×d4 ♖×e3 21 ♗f1 ♖ae8 22 ♘c1 ♔f8 23 h4 h6 24 ♔b1 ♗c8 25 ♔b2 ♕d8 26 ♔b1 ♕e7 27 ♕d2 ♗f5+ 28 ♔b2 a4 29 h5 ♗h7 30 ♔a1 a3 31 f4 ♕e4 32 ♘b3 ♕×f4 33 ♕c1 ♕g4 34 ♖h2 ♕×d1 35 ♕×d1 ♖e1 36 ♕c1 ♖×c1+ 37 ♘×c1 ♖e1 38 g4 ♖×c1# 0-1

**Tyro – Belle** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗a4 ♘f6 5 0-0 ♗e7 6 ♖e1 b5 7 ♗b3 0-0 8 d3 ♗b7 9 a3 ♗c5 10 c3 ♕e7 11 d4 e×d4 12 c×d4 ♗b6 13 e5 ♘g4 14 ♕d3 ♘a5 15 ♘g5 g6 16 ♗d1 h6 17 ♗×g4 h×g5 18 b4 ♘c4 19 f4 g×f4 20 ♖a2 ♘×e5 21 ♕d1 ♕d6 22 ♔h1 ♕×d4 23 ♖d2 ♗×g2+ 24 ♔×g2 ♕a1 25 ♗b2 ♕a2 26 ♗×e5 ♕c4 27 ♗×d7 ♗e3 28 ♖d5 c6 29 ♖d3 ♕e4+ 30 ♔h3 ♕×e5 31 ♗×c6 ♕e6+ 32 ♕g4 ♕×c6 33 ♕d7 ♕f3+ 34 ♔h4 ♕h5# 0-1

*Round 3, November 11, 1974*

**Chess 4.0 – Chaos** 1 e4 c5 2 ♘f3 ♘c6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 e6 6 ♘×c6 b×c6 7 e5 ♘d5 8 ♘×d5 c×d5 9 ♗d3 g6 10 0-0 ♗g7 11 ♗f4 0-0 12 ♕f3 ♕b6 13 c4 ♕d4 14 c×d5 ♗b7 15 ♕g3 ♗×d5 16 b3 ♖ac8 17 ♖ad1 ♕b4 18 ♖c1 ♖fd8 19 ♗e3 ♗e4 20 ♕h4 d5 21 e×d6 f5 22 ♖×c8 ♖f8 23 d7 ♕b7 24 ♖×f8+ ♗×f8 25 d8(♕) ♗×g2 26 ♕hf6 ♕b4 27 ♕e8 ♕e7 28 ♕f×e7 h5 29 ♕8×f8# 1-0

**Tech II – Ribbit** 1 e4 c5 2 ♘f3 d6 3 ♘c3 ♘c6 4 d4 ♘f6 5 ♗c4 c×d4 6 ♘×d4 ♘×d4 7 ♕×d4 e5 8 ♕d3 ♗d7 9 0-0 ♗e7 10 ♗g5 ♕b6 11 ♗×f6 ♗×f6 12 ♘d5 ♕c5 13 ♘×f6+ g×f6 14 ♗d5 ♗c8 15 ♖fe1 0-0 16 ♖e3 a5 17 ♖ae1 h5 18 ♖g3+ ♔h7 19 ♕f3 ♗g4 20 ♕×f6 ♕b4 21 ♖c3 ♕×b2 22 ♗×f7 d5 0-1

**Dart 4.1 – Ostrich** 1 e4 d5 2 ♘c3 d×e4 3 ♘×e4 ♘d7 4 ♘f3 e6 5 ♗e2 ♘h6 6 0-0 ♗e7 7 d4 0-0 8 ♗×h6 g×h6 9 c4 ♘b6 10 ♕c1 f5 11 ♘c3 ♗g5 12 ♕c2 a5 13 ♖fe1 ♗f6 14 ♖ad1 ♗d7 15 c5 ♘c8 16 ♗c4 ♕e8 17 ♕b3 ♔g7 18 ♕×b7 ♖a7 19 ♕b3 ♔f7 20 ♗×e6+ ♗×e6 21 d5 ♖a6 22 d×e6+ ♔g6 23 ♕c4 ♖c6 24 ♘e5+ ♗×e5 25 ♖×e5 ♘b6 26 ♕d4 ♘c8 27 ♖de1 ♘e7 28 ♕a4 ♖f6 29 f4 ♕a8 30 ♕d4 ♕b7 31 ♘d5 ♖c×e6 32 ♖×e6 ♘×d5 33 ♖×f6+ ♘×f6 34 ♖e6 ♕e4 35 ♕×f6+ ♔h5 36 ♖×e4 ♔g4 37 ♕g7+ 1-0

**Chute 1 – Duchess** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗×c6 d×c6 5 ♘×e5 ♕d4 6 ♘f3 ♕×e4+ 7 ♔f1 ♗e6 8 d3 ♕b4 9 a3 ♕b6 10 b3 0-0-0 11 ♗e3 ♗c5 12 d4 ♗×d4 13 ♗×d4 c5 14 ♔e2 ♕b5+ 15 c4 ♕e8 16 ♔f1 c×d4 17 ♕d3 ♗g4 18 ♘bd2 ♕d7 19 ♘e5 ♕e6 20 ♖e1 ♗h5 21 ♘ef3 ♕f6 22 ♘e4 ♕f4 23 ♘×d4 ♗g6 24 ♕e3 ♕×e3 25 ♖×e3 ♖×d4 26 ♘c5 ♘f6 27 ♖e7 ♘e4 28 ♖e5 ♖d1+ 29 ♔e2 ♘c3+ 30 ♔f3 ♖×h1 31 h3 ♖f8 32 ♔g3 ♖d8 33 ♖e7 ♖g8 34 h4 ♖d1 35 ♖e3 ♖c1 36 ♔h2 ♘b1 0-1

**Belle – Xenarbor** 1 e4 c5 2 ♘f3 d6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 a6 6 h3 e5 7 ♘de2 ♗e6 8 g4 ♘fd7 9 ♗e3 ♕h4 10 ♗g2 ♘c6 11 0-0 ♗e7 12 ♘d5 ♗d8 13 c4 ♖b8 14 ♘ec3 ♘a5 15 b3 ♘c6 16 ♕d2 0-0 17 ♖fe1 h5 18 f3 h×g4 19 h×g4 ♕h7 20 a3 ♗h4 21 ♗f2 ♗d8 22 ♖ad1 ♘a5 23 ♖b1 ♘c6 24 ♖b2 a5 25 ♘b5 ♗e7 26 ♘×e7+ ♘×e7 27 ♘×d6 ♘c6 28 ♖d1 b6 29 ♘f5 ♘f6 30 a4 ♖fd8 31 ♘d6 ♘e8 32 c5 b×c5 33 ♗×c5 ♗×b3 34 ♖a1 ♘×d6 35 ♗×d6 ♖×d6 36 ♕×d6 ♖d8 37 ♕×c6 ♖d1+ 38 ♖×d1 ♕g6 39 ♕×g6 f×g6 40 ♖×b3 1-0

*Round 4, November 12, 1974*

**Ribbit – Chess 4.0** 1 e4 c5 2 c3 d5 3 e×d5 ♕×d5 4 d4 c×d4 5 c×d4 ♘c6 6 ♘f3 ♗g4 7 ♘c3 ♕d6 8 d5 ♘b4 9 ♗b5+ ♗d7 10 ♗×d7+ ♔×d7 11 ♗e3 ♕a6 12 ♘e5+ ♔e8 13 a3 ♕d6 14 ♕a4+ ♘c6 15 d×c6 b×c6 16 ♘×c6 e5 17 ♘×a7+ ♕d7 18 ♕×d7+ ♔×d7 19 ♖d1+ ♔e6 20 0-0 ♘f6 21 b4 ♗e7 22 h3 h5 23 ♖fe1 h4 24 ♖d3 e4 25 ♗d4 ♖he8 26 ♗×f6 ♗×f6 27 ♖×e4+ ♔f5 28 ♖×e8 ♖×e8 29 g4+ h×g3 30 f×g3 ♖e1+ 31 ♔f2 ♖c1 32 g4+ ♔g6 33 ♘e4 ♗e5 34 b5 ♖c2+ 35 ♔f3 ♖h2 36 ♘f2 ♗f6 37 ♖d6 ♔h7 38 ♖d5 ♗b2 39 ♔g3 ♖×f2 40 ♔×f2 ♗×a3 41 b6 ♗c1 42 b7 ♗f4 43 ♘c6 ♗c7 44 ♖d7 ♗f4 45 ♖×f7 ♗d6 46 b8(♕) ♗×b8 47 ♘×b8 ♔g6 48 ♖f5 ♔h6 49 ♘d7 g6 50 ♖f6 ♔g5 51 ♔g3 ♔h6 52 ♘e5 ♔g7 53 g5 ♔g8 54 ♘×g6 ♔h7 55 h4 ♔g8 56 h5 ♔g7 57 h6+ ♔h7 58 ♘e5 ♔h8 59 g6 ♔g8 60 ♔g4 ♔h8 61 ♖f8# 1-0

**Chaos – Dart 4.1** 1 d4 d5 2 c4 e6 3 ♘c3 ♘f6 4 ♗g5 ♘bd7 5 e3 c6 6 c×d5 e×d5 7 ♗d3 ♗b4 8 ♘f3 ♗×c3+ 9 b×c3 0-0 10 0-0 ♖e8 11 ♕c2 ♘f8 12 ♖ab1 ♗g4 13 ♘e5 ♗c8 14 f3 c5 15 ♗b5 ♗d7 16 ♗×f6 g×f6 17 ♘×d7 ♘×d7 18 ♕f5 ♖e7 19 ♗d3 ♔f8 20 ♕×h7 ♖e6 21 ♖×b7 c×d4 22 ♗f5 ♖d6 23 ♕h8+ ♔e7 24 ♕×d8+ ♖×d8 25 e×d4 a5 26 a4 ♖c6 27 ♖e1+ ♔d6 28 ♖e3 ♖b6 29 ♖a7 ♖b2 39 ♖e1 ♔c6 31 ♖×a5 ♖g8 32 ♗×d7+ ♔×d7 33 ♖×d5+ ♔c6 34 ♖c5+ ♔d6 35 g4 ♖g5 36 ♖f5 ♖×f5 37 g×f5 ♖a2 38 h4 ♖×a4 39 h5 ♖c4 40 ♖c1 ♔e7 41 ♔h2 ♔f8 42 ♔g3 ♔g8 43 ♔f2 ♔g7 44 ♔g3 ♔h6 45 ♔h4 ♔g7 46 ♖c2 ♔h6 47 f4 ♔h7 48 ♔g4 ♔g7 49 d5 ♔f8 50 h6 ♖e4 51 d6 ♖e1 52 h7 ♔g7 53 ♖h2 ♖g1+ 1-0

**Duchess – Belle** 1 e4 e5 2 ♘f3 ♘c6 3 ♗c4 ♗c5 4 c3 ♘f6 5 d4 e×d4 6 c×d4 ♗b4+ 7 ♗d2 ♗×d2+ 8 ♘b×d2 d5 9 e×d5 ♘×d5 10 ♕b3 ♘ce7 11 0-0 0-0 12 ♖fe1 c6 13 a4 ♕b6 14 ♕×b6 a×b6 15 ♖e5 ♗d7 16 ♗d3 c5 17 b3 b5 18 d×c5 b×a4 19 ♖ae1 a×b3 20 ♘×b3 ♖fe8 21 ♘bd4 f6 22 ♖5e4 ♘c6 23 ♗c4 ♘b4 24 ♖e7 ♖×e7 25 ♖×e7 ♖a1+ 26 ♘e1 h5 27 ♘b3 ♔f8 28 ♖e2 ♖a8 29 ♘c2 h4 30 ♖d2 ♘×c2 31 ♗×d5 ♖e8 32 g3 h3 33 ♖d1 ♗c6 34 ♗×c6 b×c6 35 ♘a5 ♘d4 36 ♔f1 ♘f3 37 ♘×c6 ♔g8 38 ♖d8 ♖×d8 39 ♘×d8 ♔f8 40 ♘e6+ ♔f7 41 ♘f4 ♔e7 42 ♔e2 ♘×h2 43 ♘×h3 ♔d7 44 ♘f4 ♘g4 45 ♘d3 ♔c6 46 f3 ♘e5 47 ♘×e5+ f×e5 48 ♔e3 ♔×c5 49 ♔e4 ♔d6 50 ♔f5 ♔d7 51 ♔×e5 ♔c6 52 ♔f5 ♔b7 53 ♔g6 ♔b6 54 ♔×g7 ♔b7 55 ♔f6 ♔b6 56 f4 ♔b7 57 f5 ♔b6 58 ♔e5 ♔b7 59 f6 ♔b6 0-1

**Ostrich – Chute** 1 d4 d5 2 e3 ♘f6 3 ♘f3 ♗g4 4 ♘c3 e6 5 ♗d2 ♗d6 6 ♗d3 0-0 7 0-0 ♘c6 8 ♘b5 a6 9 ♘×d6 ♕b8 10 ♘×b7 ♕×b7 11 ♗c3 ♖ab8 12 ♕e2 ♖a8 13 h3 ♗h5 14 g4 ♗g6 15 ♗×g6 f×g6 16 ♘e5 ♘×e5 17 d×e5 ♘e4 18 ♕d3 ♖f3 19 b4 ♖af8 20 ♗e1 ♖×h3 21 c4 ♕c6 22 c×d5 ♕×d5 23 ♕×d5 e×d5 24 ♖d1 c6 25 ♖c1 ♘g5 26 f4 ♘f3+ 27 ♔g2 ♖h2+ 28 ♔×f3 ♖c8 29 e6 ♖×a2 30 e7 ♖a3 31 ♖×c6 ♖e8 32 ♗h4 d4 33 ♖e6 ♔f7 34 ♖e4 d3 35 f5 g×f5 36 g×f5 ♖a2 37 ♖d1 ♖a3 38 ♖d4 ♖b8 39 ♖d8 ♖×d8 40 e×d8(♕) 1-0

**ACM 1975 – Minneapolis**

| | rate | perf | 1 | 2 | 3 | 4 | tot |
|---|---|---|---|---|---|---|---|
| 1 Chess 4.4 | 1650 | 1956 | 12+ | 3+ | 4+ | 2+ | 4 |
| 2 Tree Frog | 0 | 1606 | 7+ | 5+ | 8+ | 1– | 3 |
| 3 Etaoin Shrdlu | 0 | 1589 | 8+ | 1– | 5+ | 9+ | 3 |
| 4 Chaos | 1600 | 1579 | 9+ | 10+ | 1– | 6+ | 3 |
| 5 Chute 1.2 | 1300 | 1331 | 11+ | 2– | 3– | 10+ | 2 |
| 6 Duchess | 1333 | 1270 | 10– | 11+ | 7+ | 4– | 2 |
| 7 Tyro | 0 | 1274 | 2– | 9+ | 6– | 12+ | 2 |
| 8 Ostrich | 1300 | 1181 | 3– | 12+ | 2– | 11= | 1½ |
| 9 Wita | 1200 | 1168 | 4– | 7– | 10+ | 3– | 1 |
| 10 Black Knight | 0 | 1135 | 6+ | 4– | 9– | 5– | 1 |
| 11 Iron Fish | 0 | 1007 | 5– | 6– | 12= | 8= | 1 |
| 12 Sortie | 0 | 911 | 1– | 8– | 11= | 7– | ½ |

*Round 1, October 19, 1975*

**Sortie – Chess 4.4** 1 ♘c3 e5 2 d4 d6 3 d×e5 d×e5 4 ♕×d8+ ♔×d8 5 ♗e3 ♘f6 6 0-0-0+ ♗d7 7 ♘d5 ♘g4 8 f3 ♘×e3 9 ♘×e3 ♔c8 10 ♘c4 ♘c6 11 ♔b1 b5 12 ♘e3 ♗c5 13 ♘d5 ♖d8 14 e3 ♘a5 15 ♗d3 c6 16 ♘c3 ♗×e3 17 ♖e1 ♗d4 18 ♗×h7 ♗×c3 19 b×c3 g6 20 ♖×e5 ♖h8 21 ♗×g6 f×g6 22 ♘e2 ♔c7 23 ♘f4 ♖h6 24 ♔a1 ♖ah8 25 h4 ♗f5 26 ♔b1 ♖×h4 27 ♖×h4 ♖×h4 28 g4 ♖h1+ 29 ♔b2 ♘c4+ 30 ♔b3 ♖b1# 0-1

**Treefrog – Tyro** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗a4 ♘f6 5 0-0 ♗e7 6 ♕e2 b5 7 ♗b3 0-0 8 c3 d6 9 d3 ♕e8 10 ♘a3 ♗d8 11 h3 ♖b8 12 ♘g5 h6 13 ♘f3 ♘h5 14 ♗e3 b4 15 c×b4 ♖×b4 16 ♖fc1 ♗e6 17 ♗×e6 f×e6 18 ♕c2 ♘d4 19 ♘×d4 e×d4 20 ♗d2 ♖b7 21 g4 ♘f4 22 ♗×f4 ♖×f4 23 ♘c4 ♗h4 24 ♖f1 ♖f7 25 a3 ♕c6 26 ♖ac1 d5 27 ♘e5 ♕×c2 28 ♖×c2 ♖f6 29 ♖d1 c5 30 e×d5 e×d5 31 ♖e1 ♖b5 32 ♘d7 ♖c6 33 ♖e5 c4 34 a4 ♖b3 35 ♖×d5 c3 36 b×c3 ♖b1+ 37 ♔g2 ♖×c3 38 ♖×c3 d×c3 39 ♖c5 ♖c1 40 ♖c8+ ♔f7 41 ♘e5+ ♔e7 42 ♘g6+ ♔d7 43 ♖a8 ♗f6 44 ♖a7+ ♔c8 45 ♖×a6 ♖d1 46 ♘f4 ♗e5 47 ♘g6 ♗c7 48 ♖c6 ♖×d3 49 ♘f4 ♖d7 50 ♖×c3 ♔b8 51 ♘e6 ♗a5 52 ♖c5 ♗d2 53 g5 ♖f7 54 g×h6 ♗×h6 55 a5 g6 56 a6 ♖f6 57 ♖b5+ ♔c8 58 a7 ♖×e6 59 a8(♕)+ 1-0

**Chaos – Wita** 1 d4 ♘f6 2 c4 b6 3 f3 ♘c6 4 ♘c3 ♗a6 5 e4 e5 6 ♗e3 e×d4 7 ♗×d4 ♘×d4 8 ♕×d4 ♗d6 9 ♘ge2 0-0 10 e5 ♗c5 11 ♕h4 ♘e8 12 ♕e4 f5 13 ♕d5+ ♖f7 14 f4 ♗b4 15 0-0-0 c6 16 ♕d3 ♕h4 17 g3 ♕h6 18 a3 ♗c5 19 b4 ♗e7 20 ♗g2 ♖d8 21 ♘d4 g6 22 ♖d2 c5 23 ♗d5 c×d4 24 ♗×f7+ ♔f8 25 ♘d5

♔×f7 26 ♘×e7 ♔×e7 27 ♕×d4 ♗b7 28 ♖hd1 ♕h5 29 c5 ♕f3 30 ♕d3 ♕×d3 31 ♖×d3 ♗e4 32 ♖d4 b×c5 33 b×c5 1-0

**Iron Fish – Chute 1.2** 1 e4 c5 2 ♗c4 ♘f6 3 e5 ♘g8 4 d4 c×d4 5 ♕×d4 ♘c6 6 ♕c5 e6 7 ♕e3 ♕b6 8 ♘d2 ♕×e3+ 9 f×e3 ♘×e5 10 ♘gf3 ♘×c4 11 ♘×c4 d5 12 ♘ce5 f6 13 ♘d3 e5 14 c3 e4 15 ♘f4 e×f3 16 g×f3 ♘e7 17 e4 d×e4 18 f×e4 ♗d7 19 ♗e3 ♗c6 20 ♔e2 ♗×e4 21 ♖hd1 ♘f5 22 ♔f2 ♗d6 23 c4 ♘×e3 24 ♔×e3 ♗×f4+ 25 ♔×f4 ♗c6 26 ♖d3 0-0 27 ♖e1 ♖ae8 28 ♖ee3 ♖×e3 29 ♖×e3 g5+ 30 ♔g4 ♗d7+ 31 ♔f3 f5 32 ♖e7 ♗c6+ 33 ♔e2 f4 34 ♔e1 f3 35 ♖e5 f2+ 36 ♔f1 ♗g2+ 37 ♔e2 f1(♕)+ 38 ♔d2 ♕f4+ 39 ♖e3 ♖d8+ 40 ♔c2 ♕×e3 41 b4 ♖d2+ 42 ♔c1 ♗e4 43 b5 ♖×a2+ 44 ♔d1 ♖×h2 45 c5 ♕×c5 46 b6 a×b6 47 ♔e1 ♗d3 48 ♔d1 g4 49 ♔e1 g3 50 ♔d1 ♕f2 51 ♔c1 g2 52 ♔d1 g1(♕)# 0-1

**Etaoin Shrdlu – Ostrich** 1 e4 e5 2 ♗c4 ♘f6 3 ♘f3 ♘×e4 4 ♘×e5 d5 5 ♗e2 f6 6 ♗b5+ ♔e7 7 ♘f3 c6 8 ♗e2 ♗g4 9 0-0 ♔f7 10 d4 ♗e7 11 ♘bd2 ♗×f3 12 ♘×f3 ♕d6 13 ♘h4 g5 14 ♘f5 ♕e6 15 ♗g4 ♔f8 16 ♘×e7 ♕×e7 17 f4 h6 18 f×g5 h×g5 19 ♖e1 ♖h4 20 g3 ♖h6 21 ♗d2 c5 22 d×c5 ♕×c5+ 23 ♗e3 ♕b5 24 ♖c1 ♕×b2 25 ♕×d5 ♘×g3 26 ♗c5+ ♔g7 27 h×g3 ♘a6 28 ♖b1 ♕×c2 29 ♕×b7+ ♔g6 30 ♕×a8 ♕×c5+ 31 ♔g2 ♖h7 32 ♕g8+ ♖g7 33 ♕e8+ ♖f7 34 ♗h5+ ♔×h5 35 ♕×f7+ ♔g4 36 ♕e6+ ♔h5 37 g4+ ♔h6 38 ♕×f6+ ♔h7 39 ♖h1+ ♔g8 40 ♖h8# 1-0

**Duchess – Black Knight** 0-1

*Round 2, October 19, 1975*

**Ostrich – Sortie** 1 e4 d5 2 e×d5 ♕×d5 3 ♘c3 ♕e6+ 4 ♘ge2 ♗d7 5 d4 ♘f6 6 ♗f4 ♘a6 7 b3 0-0-0 8 d5 ♕g4 9 ♗e5 ♘b4 10 a3 ♘a6 11 ♘f4 g5 12 ♗×a6 b×a6 13 ♕d4 g×f4 14 ♕×a7 ♕×g2 15 ♕a8# 1-0

**Tyro – Wita** 1 d4 ♘f6 2 c4 g6 3 ♘c3 ♗g7 4 e4 e6 5 ♗g5 h6 6 ♗×f6 ♗×f6 7 ♘f3 ♘c6 8 ♗e2 b6 9 0-0 0-0 10 ♕d2 h5 11 ♖ae1 ♕e7 12 ♕e3 ♕b4 13 ♖b1 ♗a6 14 a3 ♕b3 15 e5 ♗e7 16 ♘d2 ♗g5 17 f4 ♗×f4 18 ♖×f4 ♕c2 19 ♗d3 ♕×b1+ 20 ♗×b1 ♖ab8 21 d5 ♘d8 22 ♖f1 e×d5 23 ♘×d5 ♘e6 24 ♕h6 c6 25 ♘f6# 1-0

**Duchess – Iron Fish** 1 e4 d6 2 d4 ♘f6 3 ♘c3 ♗g4 4 ♘f3 ♗×f3 5 ♕×f3 ♘c6 6 ♗b5 ♕d7 7 0-0 ♕g4 8 ♕×g4 ♘×g4 9 d5 a6 10 d×c6 a×b5 11 c×b7 ♖b8 12 f3 ♘e5 13 f4 ♘g4 14 h3 ♘f6 15 e5 ♘d7 16 e×d6 ♖×b7 17 d×c7 ♖×c7 18 ♘×b5 ♖×c2 19 ♖e1 e6 20 b3 ♗c5+ 21 ♗e3 ♗×e3+ 22 ♖×e3 ♖c5 23 ♘d6+ ♔e7 24 ♖d1 ♖a8 25 b4 ♖c6 26 ♘f5+ ♔d8 27 b5 ♖c5 28 ♘d6 ♔e7 29 a3 ♖a4 30 f5 g6 31 f×e6 f×e6 32 g4 ♖f4 33 ♔h1 ♖c2 34 ♔g1 e5 35 ♔h1 h6 36 ♘e4 ♖c4 37 ♘c3 ♔e6 38 a4 ♖f2 39 a5 ♔e7 40 ♔g1 ♖f8 41 b6 ♖c5 42 ♘d5+ ♔d8 43 b7 ♖×a5 44 ♘b6 1-0

**Chute 1.2 – Treefrog** 1 e4 d5 2 e×d5 ♘f6 3 ♗b5+ ♗d7 4 ♗×d7+ ♕×d7 5 c4 c6 6 d×c6 ♘×c6 7 ♘c3 ♘e5 8 ♕e2 ♘d3+ 9 ♔f1 0-0-0 10 ♘b5 a6 11 ♘a7+ ♔b8 12 ♕e3 ♘g4 13 ♕b6 ♘g×f2 14 ♘c6+ ♕×c6 15 ♕×d8+ ♕c8 16 ♕×c8+ ♔×c8 17 ♔e2 ♘×h1 18 ♔×d3 e6 19 ♘f3 ♗d6 20 b3 ♘f2+ 21 ♔e2 ♘g4 22 h3 ♘f6 23 ♗b2 ♖e8 24 ♖f1 e5 25 a3 h6 26 ♔d1 ♘e4 27 ♖e1 ♘f2+ 28 ♔c2 e4 29 ♘g1 ♘d3 30 ♖f1 ♘×b2 31 ♔×b2 ♗e5+ 32 ♔a2 ♖d8 33 ♖f2 ♗h2 34 ♘f3 e×f3 35 g×f3 ♗f4 36 ♖g2 ♖×d2+ 37 ♖×d2 ♗×d2 38 ♔b2 a5 39 ♔c2 ♗e3 40 ♔d3 ♗c5 41 a4 f5 42 h4 0-1

**Black Knight – Chaos** 1 e4 c5 2 ♘f3 ♘c6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 e6 6 ♘f3 ♗b4 7 ♔e2 ♗×c3 8 b×c3 ♘×e4 9 ♕d3 d5 10 ♕e3 ♕b6 11 ♔d1 ♘×f2+ 12 ♔e2 ♕×e3+ 13 ♗×e3 ♘×h1 14 ♔d2 e5 15 ♗b5 e4 16 ♘d4 ♗d7 17 ♘b3 ♘e5 18 ♗×d7+ ♔×d7 19 ♗g1 ♘c4+ 20 ♔e2 e3 21 ♗×e3 ♖he8 22 ♔f1 ♘×e3+ 23 ♔g1 ♘×c2 24 ♖d1 ♖e1+ 25 ♖×e1 ♘×e1 26 ♘d4 ♖e8 27 ♘f5 ♘d3 28 g3 ♘hf2 29 a4 g5 30 h4 ♖e1+ 31 ♔g2 ♖e7 32 ♘d4 g4 33 h5 ♖e5 34 ♘c2 ♘e1+ 35 ♘×e1 ♖×e1 36 ♔×f2 ♖e7 37 a5 ♖e5 38 h6 a6 39 ♔g2 ♖e6 40 ♔f2 ♔c6 41 ♔g2 ♖×h6 42 ♔f2 ♖e6 43 ♔g1 ♔b5 44 ♔f2 ♔×a5 45 ♔g1 ♖e3 46 ♔g2 ♖×c3 47 ♔f2 h5 48 ♔g2 d4 49 ♔h2 d3 50 ♔h1 d2 51 ♔g2 d1(♕) 52 ♔f2 ♔b6 53 ♔g2 ♖c2# 0-1

**Chess 4.4 – Etaoin Shrdlu** 1 f4 d5 2 ♘f3 ♘c6 3 e3 ♗f5 4 ♗b5 ♘f6 5 ♘e5 ♕d6 6 0-0 a6 7 ♗×c6+ b×c6 8 g4 ♗d7 9 g5 ♘e4 10 ♘×d7 ♕×d7 11 d3 ♘d6 12 ♘d2 g6 13 ♕f3 ♗g7 14 c4 ♖b8 15 ♘b3 0-0 16 c×d5 c×d5 17 ♕×d5 c6 18 ♕g2 ♖b5 19 d4 ♖d8 20 ♘c5 ♕c8 21 a4 ♖b4 22 ♕e2 a5 23 b3 ♘f5 24 ♗b2 ♕c7 25 ♘a6 ♕d6 26 ♘×b4 ♕×b4 27 ♕d3 c5 28 ♗a3 ♕b6 29 ♖ac1 c×d4 30 e4 ♘d6 31 ♗c5 ♕b8 32 f5 g×f5 33 e×f5 h5 34 ♖cd1 ♗e5 35 ♕h3 ♘e4 36 ♗×e7 ♖d7 37 f6 ♖d5 38 ♔h1 ♘c3 39 ♖d3 ♕×b3 40 g6 ♕b7 41 ♕×h5 ♖d8+ 42 ♖df3 ♕×f3+ 43 ♖×f3 f×g6 44 ♕×g6+ ♔h8 45 ♖h3# 1-0

*Round 3, October 20, 1975*

**Iron Fish – Sortie** 1 e4 d5 2 e×d5 ♕×d5 3 d4 e5 4 d×e5 ♕×e5+ 5 ♕e2 ♕×e2+ 6 ♗×e2 ♗f5 7 ♗b5+ ♘c6 8 ♗×c6+ b×c6 9 ♘a3 0-0-0 10 ♗e3 ♗×a3 11 b×a3 ♗×c2 12 ♗×a7 ♖d3 13 ♗c5 ♘f6 14 ♖c1 ♖e8+ 15 ♘e2 ♗a4 16 f4 ♘e4 17 ♔f1 ♖e6 18 ♘c3 ♘×c3 19 ♖e1 ♖×e1+ 20 ♔×e1 ♔b8 21 g3 ♖d1+ 22 ♔f2 ♖×h1 23 ♔g2 ♖d1 24 f5 ♗c2 25 g4 ♔a8 26 h4 ♘×a2 27 ♗e3 ♘c3 28 ♔f3 ♘b5 29 ♗c5 ♖a1 39 h5 ♗d3 31 ♔e3 ♗c4 32 a4 ♖×a4 33 ♗d4 ♘×d4 34 ♔×d4 ♗b5+ 35 ♔e5 ♖×g4 36 h6 g×h6 37 ♔f6 ♗c4 38 ♔e7 ♔b7 39 ♔f6 ♗d5 40 ♔e5 ♖e4+ 41 ♔f6 ♖e3 42 ♔g7 h5 43 ♔×h7 ♖f3 44 ♔h8 ♖×f5 45 ♔h7 ♖f6 46 ♔g8 ♖g6+ 47 ♔f8 ♗e6 48 ♔e8 ♖g8+ 49 ♔e7 ♗d5 50 ♔f6 ♖g4 51 ♔e5 ♗e6 52 ♔f6 ♖g6+ 53 ♔e7 ♖g5 54 ♔e8 ♔b6 55 ♔f8 ♖g4 56 ♔e8 ♖g8+ 57 ♔e7 ♗d5 58 ♔d7 ♗c4 59 ♔e7 ♖g6 60 ♔f8 ♖g5 61 ♔e8 ♔b7 62 ♔f8 ♖g6 63 ♔e8 ♖g8+ 64 ♔e7 ♖g3 65 ♔d8 ♖g5 66 ♔e8 ♗e6 67 ♔e7 ♖a5 68 ♔e8 ♖a8+ 69 ♔e7 ♖g8 70 ♔f6 ♔b6 71 ♔e7 ♔c5 72 ♔f6 ♔d5 73 ♔e7 ♔e4 74 ♔f6 ♔d5 75 ♔e7 ♔e4 76 ♔f6 ♔f4 77 ♔e7 ♔e4 78 ♔f6 ♔f4 79 ♔e7 ♖b8 80 ♔f6 ♖g8 81 ♔e7 ♖c8 82 ♔f6 ♖g8 83 ♔e7 ♖c8 84 ♔f6 ♖g8 ½-½

**Wita – Black Knight** 1 e4 e6 2 d4 d5 3 ♘c3 ♘c6

4 e5 f5 5 ♗b5 ♔f7 6 ♘f3 h6 7 ♗f4 ♗e7 8 h3 ♗b4 9 0-0 ♗×c3 10 b×c3 g5 11 ♗c1 ♔f8 12 ♗×c6 b×c6 13 ♗a3+ ♘e7 14 ♕d3 a5 15 ♖fb1 ♔f7 16 ♗c5 a4 17 ♖b4 ♘g6 18 ♖e1 ♘e7 19 g4 ♖f8 20 ♘d2 ♕d7 21 a3 ♔g7 22 ♕f3 ♖a5 23 ♖b8 ♖a6 24 ♕d3 ♖f7 25 ♖eb1 f4 26 ♔g2 ♖f8 27 ♖×c8 ♕×c8 28 ♗×e7 f3+ 29 ♘×f3 ♖f4 30 ♗f6+ ♔f8 31 ♖b8 ♖a8 32 ♖×c8+ ♖×c8 33 ♕h7 ♖e4 34 ♗e7+ ♔e8 35 ♗c5 ♖e2 36 ♕e7# 1-0

**Treefrog – Ostrich** 1 e4 e5 2 ♘f3 ♘f6 3 ♘×e5 d6 4 ♘f3 ♘×e4 5 ♘c3 d5 6 ♕e2 ♗e7 7 ♘×e4 d×e4 8 ♕×e4 ♘d7 9 ♗c4 ♘f6 10 ♕e5 ♔f8 11 ♘g5 ♗d6 12 ♘×f7 ♗×e5 13 ♘×d8 ♗h3 14 ♘f7 ♗×g2 15 ♖g1 ♖e8 16 ♖×g2 ♗×b2+ 17 ♔d1 ♗×a1 18 ♗a3+ ♖e7 19 ♘×h8 ♘g8 20 ♗×g8 ♔×g8 21 ♗×e7 ♔×h8 22 f4 c6 23 ♔e2 g6 24 ♔e3 ♔g7 25 ♖g1 ♗f6 26 ♗×f6+ ♔×f6 27 c4 h6 28 c5 b5 29 c×b6 a×b6 30 ♖c1 c5 31 ♖b1 g5 32 ♖×b6+ ♔f5 33 f×g5 h×g5 34 ♖b5 ♔e6 35 ♖×c5 ♔f6 36 a4 ♔e6 37 ♖×g5 ♔e7 38 h4 ♔d7 39 a5 ♔c7 40 h5 ♔d6 41 h6 ♔e7 42 h7 ♔f7 43 h8(♕) 1-0

**Chaos – Chess 4.4** 1 d4 ♘f6 2 c4 c5 3 d5 e6 4 ♘c3 e×d5 5 c×d5 d6 6 e4 g6 7 ♗e2 ♗g7 8 ♘f3 0-0 9 0-0 ♖e8 10 ♘d2 ♘a6 11 f3 ♘c7 12 a4 b6 13 ♘c4 ♗a6 14 ♗g5 h6 15 ♗h4 g5 16 ♗f2 ♘h5 17 a5 ♘f4 18 a×b6 a×b6 19 ♗e3 ♗×c3 20 b×c3 ♗×c4 21 ♗×c4 ♖×a1 22 ♕×a1 ♘c×d5 23 ♗×d5 ♘×d5 24 ♗×c5 ♘×c3 25 ♗b4 ♘e2+ 26 ♔h1 ♘f4 27 ♖d1 ♕a8 28 ♗×d6 ♕×a1 29 ♖×a1 ♘e2 30 ♖b1 ♖c8 31 h3 ♖c6 32 ♗e5 ♔h7 33 ♖e1 ♖c2 34 ♔h2 b5 35 ♖b1 ♖c5 36 ♗d6 ♖c6 37 ♗f8 ♖c2 38 ♖a1 ♘f4 39 ♖g1 ♔g6 40 ♗b4 f5 41 e×f5+ ♔×f5 42 ♗d6 ♖b2 43 ♔h1 ♔f6 44 ♗×f4 g×f4 45 ♔h2 b4 46 ♖c1 b3 47 ♖c6+ ♔g5 48 ♖c5+ ♔g6 49 ♖c6+ ♔h5 50 ♖c5+ ♔g6 51 ♖c6+ ♔g5 52 ♖c5+ ♔f6 53 ♖b5 ♔g6 54 ♔g1 ♖b1+ 55 ♔h2 ♔g7 56 ♖b4 b2 57 ♖b6 ♔f7 58 h4 ♔g7 59 h5 ♔f7 60 ♖b8 ♔e6 61 ♖b7 ♔f5 62 ♖b5+ ♔f6 63 g3 f×g3+ 64 ♔g2 ♔e6 65 ♖b3 ♔f5 66 ♖b6 ♔g5 67 ♖b5+ ♔h4 68 ♖b8 ♔×h5 69 ♖b5+ ♔g6 70 ♖b6+ ♔f5 71 ♖b5+ ♔e6 72 ♖b6+ ♔d5 73 ♖b5+ ♔c6 74 ♖b3 h5 75 f4 h4 76 ♖b8 ♔d5 77 ♖b5+ ♔e4 78 ♖b6 ♔f5 79 ♖b5+ ♔g4 80 ♖g5+ ♔×f4 81 ♖g4+ ♔e3 82 ♖b4 ♔d3 83 ♖b5 ♔c3 84 ♖c5+ ♔b4 85 ♖c6 ♖g1+ 86 ♔h3 ♖h1+ 87 ♔g2 ♔a5 88 ♖c5+ ♔a6 89 ♖c6+ ♔b7 90 ♖e6 b1(♕) 91 ♖e7+ ♔c6 92 ♔f3 ♕f5+ 93 ♔e2 0-1

**Tyro – Duchess** 1 d4 d5 2 c4 e6 3 g3 ♘f6 4 ♘f3 ♗e7 5 ♗g2 0-0 6 0-0 d×c4 7 ♕c2 a6 8 ♕×c4 b5 9 ♕c2 ♗b7 10 ♗f4 ♗d6 11 ♘e5 ♗×g2 12 ♔×g2 ♗×e5 13 ♗×e5 ♘d5 14 ♖c1 f6 15 ♗×c7 ♘×c7 16 ♕×c7 ♕×d4 17 ♖c2 ♕d5+ 18 f3 b4 19 ♖d2 ♕b5 20 ♔h1 e5 21 a4 b×a3 22 ♘×a3 ♕c6 23 ♕a5 ♖a7 24 ♕d5+ ♔h8 25 ♘c4 ♖c8 26 ♘d6 ♕×d5 27 ♖×d5 ♖c2 28 e4 ♘c6 29 ♘f5 g6 30 ♘h4 g5 31 ♘f5 a5 32 ♖b5 ♖d7 33 b3 ♖dd2 34 h3 ♖h2+ 35 ♔g1 ♖cg2+ 36 ♔f1 ♖f2+ 37 ♔e1 ♖b2 38 ♖b8+ ♘×b8 39 h4 ♖h1# 0-1

**Etaoin Shrdlu – Chute 1.2** 1 e4 c5 2 ♗c4 ♘f6 3 e5 ♘g8 4 ♘f3 e6 5 ♘c3 ♘c6 6 0-0 ♘d4 7 ♘×d4 c×d4 8 ♘b5 ♕b6 9 ♕g4 ♘h6 10 ♕×d4 ♕×d4 11 ♘×d4 ♗e7 12 d3 ♘g4 13 ♖e1 ♗c5 14 c3 f6 15 e×f6 ♗×d4 16 c×d4 ♘×f6 17 d5 ♔f7 18 d×e6+ d×e6 19 ♗f4 ♘d5 20 ♗e5 ♘b6 21 ♖e4 ♘×c4 22 ♖×c4 ♖d8 23 ♖c7+ ♖d7 24 ♖ac1 ♖×c7 25 ♖×c7+ ♔e8 26 ♖×g7 h6 27 ♖h7 ♔f8 28 g4 ♔g8 29 ♖×h6 ♔f8 30 h4 b6 31 ♖f6+ ♔e7 32 ♖h6 ♗d7 33 ♖h7+ ♔d8 34 ♗f6+ ♔c7 35 ♗e5+ ♔c6 36 g5 ♖g8 37 f4 a6 38 h5 ♖d8 39 ♗f6 ♖f8 40 d4 ♗e8 41 h6 ♗g6 42 ♖a7 ♗d3 43 h7 ♔b5 44 ♖f7 ♖a8 45 h8(♕) ♖×h8 46 ♗×h8 ♗b1 47 ♗f6 ♗×a2 48 ♖h7 ♗b1 49 ♖h6 ♗e4 50 ♔f2 1-0

*Round 4, October 21, 1975*

**Sortie – Tyro** 1 ♘f3 d5 2 d4 ♘f6 3 ♗f4 c5 4 d×c5 ♕a5+ 5 ♘c3 e6 6 ♗×b8 ♖×b8 7 ♕d4 ♗×c5 8 ♕f4 ♖a8 9 e3 ♗b4 10 ♗b5+ ♕×b5 11 0-0-0 ♗×c3 12 b×c3 ♕c4 13 ♕×c4 d×c4 14 ♘e5 ♘e4 15 ♖d4 ♘×c3 16 ♖×c4 ♘d5 17 e4 ♘b6 18 ♖c5 ♘a4 19 ♖c4 b5 20 ♖b4 ♖b8 21 c4 ♘c3 22 ♖e1 ♘×a2+ 23 ♔b1 ♘×b4 24 f4 f6 25 ♘g4 b×c4 26 ♖c1 h5 27 ♘×f6+ g×f6 28 ♖×c4 a5 29 g3 ♖h7 30 h3 ♖d7 31 ♖c3 ♖d3 32 ♖×d3 ♘×d3+ 33 ♔a2 ♔d7 34 ♔a1 ♘f2 35 ♔a2 ♘×e4 36 g4 h×g4 37 h×g4 ♗a6 38 ♔a1 ♘c3 39 g5 ♖b1# 0-1

**Ostrich – Iron Fish** 1 e4 d5 2 e×d5 ♕×d5 3 ♘c3 ♕e5+ 4 ♘ge2 ♘f6 5 d4 ♕a5 6 ♗e3 ♗f5 7 ♘g3 ♗g4 8 ♗e2 ♗×e2 9 ♕×e2 ♘bd7 10 0-0 ♕b4 11 ♕b5 ♕×b5 12 ♘×b5 0-0-0 13 ♘×a7+ ♔b8 14 ♘b5 e5 15 d×e5 ♘×e5 16 ♗g5 ♗b4 17 ♗×f6 g×f6 18 c3 ♖d5 19 ♘×c7 ♖d2 20 c×b4 ♔×c7 21 b3 ♔c6 22 ♘e4 ♖e2 23 ♖fe1 ♖×e1+ 24 ♖×e1 f5 25 ♘g3 ♘d3 26 ♖e7 f4 27 ♘h5 f5 28 a3 ♖a8 29 a4 b5 30 ♖×h7 b×a4 31 ♖h6+ ♔c7 32 ♖h7+ ♔c6 33 ♖h6+ ♔c7 34 b×a4 ♖×a4 35 ♖h7+ ♔d6 36 ♖h6+ ♔e5 37 ♔f1 ♖a1+ 38 ♔e2 ♘×b4 39 h4 ♖c1 40 ♖b6 ♘d5 41 ♖h6 ♖c2+ 42 ♔f3 ♖c3+ 43 ♔e2 ♖c2+ 44 ♔f1 ♖c1+ 45 ♔e2 ♖c2+ ½-½

**Chaos – Duchess** 1 d4 d5 2 c4 e6 3 ♘c3 ♘f6 4 ♗g5 ♗e7 5 e3 0-0 6 ♘f3 h6 7 ♗h4 ♘e4 8 ♗×e7 ♕×e7 9 c×d5 ♘×c3 10 b×c3 e×d5 11 ♕b3 ♕d6 12 ♗d3 ♘a6 13 0-0 ♖e8 14 ♘e5 c5 15 ♕b1 ♖e7 16 ♗h7+ ♔h8 17 ♗d3 b6 18 f3 h5 19 a4 ♗b7 20 ♕c2 c×d4 21 e×d4 ♖c8 22 ♖fe1 ♘b4 23 ♕d2 ♘×d3 24 ♕×d3 ♕h6 25 a5 b×a5 26 ♖×a5 ♕b6 27 ♖aa1 ♕c7 28 ♖×a7 ♕b6 29 ♖a2 ♕c7 30 ♖c1 ♕b6 31 ♕f5 ♖×c3 32 ♕×h5+ ♔g8 33 ♖d1 ♖b3 34 ♖c2 ♖b1 35 ♖d2 ♕b3 36 ♖×b1 ♕×b1+ 37 ♔f2 ♗a6 38 ♔g3 ♕c1 39 ♖a2 ♕e1+ 40 ♖f2 ♕c3 41 ♕g4 ♗b5 42 h4 ♗e8 43 ♔h2 f6 44 ♘g6 ♖e1 45 ♖a2 ♕c4 46 ♖a7 f5 47 ♕h5 ♖h1+ 48 ♔×h1 ♕c1+ 49 ♔h2 ♗×g6 50 ♕×g6 ♕f4+ 51 ♔h3 ♕g4+ 52 f×g4 f×g4+ 53 ♔×g4 ♔f8 54 ♕f7# 1-0

**Wita – Etaoin Shrdlu** 1 c4 e5 2 ♘f3 ♘c6 3 d4 e×d4 4 ♘×d4 ♗c5 5 ♘f5 g6 6 ♘g3 ♘f6 7 ♗g5 ♗×f2+ 8 ♔×f2 ♘g4+ 9 ♔f3 ♕×g5 10 h4 ♕e3+ 11 ♔×g4 d5+ 12 ♘f5 ♗×f5# 0-1

**Chess 4.4 – Treefrog** 1 e4 d5 2 e×d5 ♘f6 3 d4 ♘×d5 4 ♘f3 ♗g4 5 ♗e2 e6 6 0-0 ♘c6 7 c4 ♘f6 8 ♘c3 ♗b4 9 d5 ♗×c3 10 d×c6 ♕×d1 11 ♖×d1 ♗b4 12 c×b7 ♖b8 13 h3 ♗f5 14 a3 ♗c5 15 g4 ♗c2 16 ♖d2 ♗b3 17 ♘d4 ♗×d4 18 ♖×d4 e5 19 ♖d2

♖×b7 20 g5 ♘e4 21 ♖d5 f6 22 ♗h5+ ♔e7 23 ♗f3 c6 24 ♖a5 ♗c2 25 b4 ♖hb8 26 ♗e3 ♘×g5 27 ♗×g5 f×g5 28 ♖×e5+ ♔d6 29 ♖ae1 ♖f7 30 ♖e6+ ♔c7 31 ♖×c6+ ♔d8 32 ♖ce6 ♖b6 33 ♖e8+ ♔d7 34 ♗d5 ♖f5 35 ♖1e7+ ♔d6 36 ♗e6 ♖c6 37 ♗×f5 ♗×f5 38 c5+ 1-0

**Chute 1.2 – Black Knight** 1 e4 e6 2 ♘c3 d5 3 e×d5 e×d5 4 ♘f3 ♗e6 5 ♘d4 ♕f6 6 ♘×e6 ♕×e6+ 7 ♗e2 d4 8 ♘b1 ♘c6 9 0-0 ♘f6 10 b3 ♗c5 11 ♗c4 ♕d7 12 ♖e1+ ♔d8 13 d3 ♗f8 14 ♗d2 a6 15 a4 ♖a7 16 ♕f3 ♘b4 17 ♘a3 c5 18 ♗g5 ♖g8 19 ♕f4 ♔c8 20 ♗×f6 ♘d5 21 ♗×d5 b5 22 ♗e4 ♗d6 23 ♗e5 b4 24 ♘c4 g5 25 ♘×d6+ ♔b8 26 ♕f6 ♖g6 27 ♘c4+ ♔c8 28 ♘b6# 1-0

### ACM 1976 – Houston

| | rate | perf | 1 | 2 | 3 | 4 | tot |
|---|---|---|---|---|---|---|---|
| 1 Chess 4.5 | 1722 | 1936 | 6+ | 5+ | 3+ | 2+ | 4 |
| 2 Chaos | 1600 | 1615 | 8+ | 7+ | 5+ | 1– | 3 |
| 3 Black Knight | 0 | 1530 | 9+ | 10+ | 1– | 7+ | 3 |
| 4 Blitz IV | 0 | 1319 | 5– | BYE | 9+ | 6+ | 3 |
| 5 Duchess | 1351 | 1394 | 4+ | 1– | 2– | 11+ | 2 |
| 6 Wita | 1250 | 1235 | 1– | 9+ | 8+ | 4– | 2 |
| 7 Chute 1.2 | 1300 | 1217 | 11+ | 2– | 10+ | 3– | 2 |
| 8 L'Excentrique | 0 | 1075 | 2– | 11+ | 6– | 10+ | 2 |
| 9 Etaoin Shrdlu | 0 | 704 | 3– | 6– | 4– | BYE | 1 |
| 10 Chesstar | 0 | 674 | BYE | 3– | 7– | 8– | 1 |
| 11 Xenarbor IV | 0 | 641 | 7– | 8– | BYE | 5– | 1 |

*Round 1, October 19, 1976*

**Wita – Chess 4.5** 1 d4 ♘f6 2 g3 e6 3 ♗g2 ♘c6 4 ♘h3 ♗b4+ 5 c3 ♗e7 6 0-0 0-0 7 ♗f4 d6 8 ♘a3 e5 9 d×e5 d×e5 10 ♗e3 ♘g4 11 ♕×d8 ♖×d8 12 ♘b5 ♘×e3 13 f×e3 ♗×h3 14 ♗×h3 ♖d2 15 ♗g4 ♖×b2 16 ♘×c7 ♖d8 17 ♖fb1 ♖dd2 18 ♗f3 ♗c5 19 ♔f2 ♘a5 20 ♖×b2 ♖×b2 21 ♗d5 ♖d2 22 ♖b1 ♗b6 23 ♔e1 ♗×e3 24 a3 b6 25 g4 ♗f4 26 ♖b4 ♖c2 27 ♗e4 ♖c1+ 28 ♔f2 ♗×h2 29 ♔g2 ♗g1 30 ♗d5 ♗c5 31 ♖e4 ♗d6 32 ♘b5 ♗b8 33 ♖a4 a6 34 ♘d4 e×d4 35 c×d4 ♖c3 36 ♔f1 ♗d6 37 g5 ♖×a3 38 ♖×a3 ♗×a3 39 ♔f2 ♗e7 40 ♔g3 ♗×g5 41 ♔g4 ♗e3 42 ♔f3 0-1

**Chaos – L'Excentrique** 1 d4 e6 2 c4 d5 3 ♘c3 d×c4 4 e4 e5 5 d5 b5 6 ♘×b5 ♗b4+ 7 ♗d2 ♗×d2+ 8 ♕×d2 a6 9 ♘c3 f5 10 ♘f3 f×e4 11 ♘×e5 ♕d6 12 ♕g5 ♕f6 13 ♘×e4 ♕×g5 14 ♘×g5 h6 15 ♘ef7 h×g5 16 ♘×h8 ♘e7 17 ♗×c4 ♘d7 18 0-0-0 ♘e5 19 ♗e2 ♗f5 20 d6 c×d6 21 ♖×d6 ♖c8+ 22 ♔d2 ♖c2+ 23 ♔d1 ♖×b2 24 ♖f1 ♖×a2 25 ♖×a6 ♖×a6 26 ♗×a6 ♔f8 27 ♖e1 ♘g4 28 f3 ♘×h2 29 ♖h1 ♘×f3 30 g×f3 g4 31 f×g4 ♗×g4+ 32 ♔d2 ♗f5 33 ♗d3 ♗×d3 34 ♔×d3 g5 35 ♖h5 ♔g7 36 ♖×g5+ ♔×h8 37 ♔e4 ♔h7 38 ♔e5 ♘c6+ 39 ♔f6 ♔h6 40 ♖d5 ♘b4 41 ♖d4 ♘c6 42 ♖h4# 1-0

**Black Knight – Etaoin Shrdlu** 1 e4 d5 2 e×d5 ♕×d5 3 ♘c3 ♕d6 4 ♗b5+ ♘c6 5 ♘ge2 ♗f5 6 ♘g3 ♕e5+ 7 ♔f1 ♗d7 8 ♕h5 ♘f6 9 ♕×e5 ♘×e5 10 d4 ♘eg4 11 ♗×d7+ ♘×d7 12 ♘b5 ♔d8 13 ♗g5 a6 14 ♘c3 f6 15 ♗f4 e5 16 d×e5 f×e5 17 ♗g5+ ♘gf6 18 ♗×f6+ g×f6 19 ♖d1 ♗c5 20 ♘f5 h5 21 ♔e2 ♖g8 22 ♘e4 ♗e7 23 h3 ♖×g2 24 ♘fg3 f5 25 ♔f1 ♖×g3 26 ♘×g3 e4 27 ♖g1 ♗f6 28 ♘×h5 ♗e7 29 ♖g8+ ♗f8 30 ♖×f8+ ♔e7 31 ♖×a8 e3 32 ♖g8 ♘f6 33 ♖g7+ ♔f8 34 ♖d8+ ♘e8 35 ♖×c7 f4 36 ♘g7 1-0

**Duchess – Blitz IV** 1 e4 e5 2 ♘f3 ♘c6 3 ♗c4 ♗c5 4 c3 ♘f6 5 d4 e×d4 6 c×d4 ♗b4+ 7 ♗d2 ♗×d2+ 8 ♘b×d2 d5 9 e×d5 ♘×d5 10 ♕b3 ♕e7+ 11 ♔f1 ♘f4 12 ♖e1 ♘e6 13 d5 ♕d6 14 d×e6 f×e6 15 ♗×e6 ♘e7 16 ♗×c8 ♕a6+ 17 ♔g1 ♖×c8 18 ♕b4 c5 19 ♕g4 ♔f8 20 ♕e4 ♖e8 21 ♕f4+ ♔g8 22 ♖×e7 ♖×e7 23 ♕b8+ ♔f7 24 ♕×h8 ♕×a2 25 b3 h6 26 ♕d8 ♕a1+ 27 ♘f1 ♕c3 28 ♕c8 g5 29 h4 ♔g6 30 h×g5 h×g5 31 ♕g8+ ♕g7 32 ♕d5 g4 33 ♘h4+ ♔h6 34 ♘f5+ ♔g6 35 ♕d6+ ♕f6 36 ♘×e7+ ♔g7 37 ♖h7+ ♔×h7 38 ♕×f6 b6 39 ♕g6+ ♔h8 40 ♕g8# 1-0

**Xenarbor IV – Chute 1.2** 1 d4 d5 2 c4 e6 3 ♘c3 ♗b4 4 c×d5 ♕×d5 5 ♕d2 ♕a5 6 ♕c2 e5 7 d5 ♕×d5 8 e4 ♕c5 9 ♗e3 ♗×c3+ 10 b×c3 ♕a5 11 ♗d3 ♘c6 12 ♘f3 ♘f6 13 0-0 ♖b8 14 a4 ♘g4 15 ♗c1 0-0 16 h3 ♘f6 17 ♘d2 ♗e6 18 ♗a3 ♖fe8 19 ♘c4 ♗×c4 20 ♗×c4 a6 21 ♕b3 ♘d8 22 ♖fe1 b5 23 ♗f1 ♕b6 24 a×b5 a×b5 25 ♗b2 ♘e6 26 ♕c2 ♖a8 27 ♖×a8 ♖×a8 28 ♖e3 ♕c5 29 ♕c1 c6 30 g3 ♖a4 31 ♗g2 ♘g5 32 f3 ♘h5 33 h4 ♘e6 34 ♔f2 ♘f6 35 ♗h3 ♕a7 36 ♗×e6 f×e6 37 ♕d2 ♕c5 38 ♕d8+ ♔f7 39 ♔e2 ♖a2 40 ♕d2 ♕c4+ 41 ♖d3 ♘h5 42 ♕c2 ♘×g3+ 43 ♔d2 c5 44 ♖d7+ ♔f6 45 ♔c1 b4 46 c×b4 ♘e2+ 47 ♔b1 ♕×c2+ 48 ♔×c2 c4 49 ♔b1 ♖a4 50 ♖b7 c3 51 h5 c×b2 52 ♔×b2 ♘f4 53 ♔b3 ♖a1 54 h6 g×h6 55 ♖×h7 ♖b1+ 56 ♔c2 ♖h1 57 ♖d7 ♖f1 58 ♖d2 ♖×f3 59 ♖h2 h5 60 b5 ♖e3 61 b6 ♖×e4 62 ♖d2 ♖b4 63 ♖d6 ♔e7 64 ♖c6 ♘d5 65 ♔c1 ♔d7 66 ♖c5 e4 67 ♖a5 ♖c4+ 68 ♔d2 e3+ 69 ♔e2 ♖c2+ 70 ♔d1 ♖d2+ 71 ♔e1 ♔c6 72 ♖a1 ♘c3 73 ♖c1 ♖d3 74 ♖×c3+ ♖×c3 75 ♔e2 ♖b3 0-1

*Round 2, October 20, 1976*

**Chute 1.2 – Chaos** 1 e4 c5 2 c4 e6 3 ♘c3 ♘c6 4 ♘f3 ♘f6 5 d3 ♗d6 6 ♗e2 0-0 7 0-0 b6 8 ♘b5 ♗b8 9 ♘c3 ♗b7 10 ♗g5 h6 11 ♗×f6 ♕×f6 12 ♕a4 ♖c8 13 ♕b5 ♗c7 14 ♘a4 ♘d4 15 ♘×d4 ♕×d4 16 ♖ac1 ♗c6 17 ♕a6 f5 18 e×f5 ♕e5 19 g3 ♕×e2 20 f×e6 ♕f3 21 ♕×c8+ ♖×c8 22 e×d7 ♕g2# 0-1

**Etaoin Shrdlu – Wita** 1 e4 c5 2 ♗c4 e6 3 ♘c3 d6 4 ♘f3 ♘f6 5 0-0 ♗e7 6 d4 0-0 7 d5 e×d5 8 e×d5 ♗g4 9 ♖e1 ♖e8 10 ♗d3 ♕a5 11 ♗e3 ♘bd7 12 h3 ♗×f3 13 ♕×f3 ♘e5 14 ♕e2 ♕b4 15 ♗b5 ♕×b2 16 ♘a4 ♕b4 17 ♖ab1 ♕a3 18 ♗×c5 d×c5 19 ♕×e5 ♗f8 20 ♗×e8 ♖×e8 21 ♕f4 ♕×a2 22 d6 ♖×e1+ 23 ♖×e1 ♕×c2 24 ♖c1 ♕e2 25 ♘×c5 ♘h5 26 ♕g4 ♕×g4 27 h×g4 ♘f4 28 ♖e1 b6 29 d7 b×c5 30 d8(♕) c4 31 ♖e4 ♘e6 32 ♕d5 c3 33 ♖c4 c2 34 ♖×c2 g5 35 ♔h2 h6 36 ♔g1 ♔g7 0-1

**Chesstar – Black Knight** 1 d4 d5 2 h4 e6 3 e4 d×e4 4 ♗g5 f6 5 ♗e3 e5 6 d×e5 ♕×d1+ 7 ♔×d1 f×e5 8 ♘c3 ♘f6 9 ♗c4 ♗g4+ 10 ♔c1 ♘c6 11 a3 0-0-0 12 b4 ♗d7 13 ♔b2 ♗e7 14 ♖b1 a5 15 ♘d5 ♘×d5 16 ♗×d5 a×b4 17 ♗×e4 b×a3+ 18 ♔a1 ♗g4 0-1

**Chess 4.5 – Duchess** 1 e4 e5 2 ♘f3 ♘c6 3 d4 e×d4 4 c3 d5 5 e×d5 ♕×d5 6 c×d4 ♗g4 7 ♗e2 0-0-0 8 ♗e3 ♗b4+ 9 ♘c3 ♘ge7 10 0-0 ♕d7 11 ♖c1 ♘d5 12 h3 ♗×f3 13 ♗×f3 ♗×c3 14 b×c3 ♘×e3 15 f×e3 ♕e7 16 ♗×c6 ♕×e3+ 17 ♔h1 b×c6 18 ♕g4+ ♕e6 19 ♕×g7 ♖hf8 20 ♕×h7 ♕g6 21 ♕h4 f5 22 ♖b1 ♕e6 23 ♕f2 ♕e4 24 ♖fe1 ♕d5 25 ♖e5 ♕d6 26 ♖×f5 ♕g6 27 ♖f1 ♖×f5 28 ♕×f5+ ♕×f5 29 ♖×f5 ♖e8 30 ♖f2 ♖e1+ 31 ♔h2 ♖c1 32 ♖f3 ♖a1 33 ♖f8+ ♔d7 34 ♖f7+ ♔d6 35 ♖f6+ ♔d7 36 ♖f2 c5 37 g4 c×d4 38 c×d4 ♔d6 39 g5 c5 40 g6 c×d4 41 g7 ♖e1 42 g8(♕) 1-0

**L'Excentrique – Xenarbor IV** 1 e3 ♘f6 2 ♗c4 e6 3 ♕f3 ♗e7 4 d4 d5 5 ♗d3 ♘c6 6 ♗b5 ♗d7 7 ♗d2 ♘e4 8 ♗×c6 ♗×c6 9 ♘c3 ♘×d2 10 ♔×d2 0-0 11 ♔d3 ♕d6 12 h4 ♗d7 13 a3 ♕a6+ 14 ♔d2 c5 15 h5 c×d4 16 e×d4 ♗g5+ 17 ♔e1 ♗h6 18 ♕d3 ♕×d3 19 c×d3 ♖ac8 20 ♘f3 f6 21 ♔e2 ♗f4 22 a4 a6 23 a5 ♔h8 24 g3 ♗h6 25 ♘h2 f5 26 f4 ♗e8 27 ♘f3 ♗d7 28 ♘e5 ♗e8 29 ♔f2 ♔g8 30 ♔g2 ♖c7 31 ♔f2 ♖f6 32 ♖ac1 ♖f8 33 ♘×d5 ♖×c1 34 ♘e7+ ♔h8 35 ♖×c1 ♗×h5 36 ♔g2 g6 37 ♔f2 ♔g7 38 ♖c7 ♔h8 39 ♖×b7 g5 40 f×g5 ♗g7 41 ♔e3 ♖e8 42 ♘7c6 ♖c8 43 ♘c4 ♖f8 44 ♔f2 f4 45 g×f4 ♖×f4+ 46 ♔e3 ♖f3+ 47 ♔e4 ♖f8 48 ♘6e5 ♗e8 49 ♘e3 ♗h5 50 b4 ♗e8 51 ♖e7 ♗b5 52 ♖×e6 ♖f2 53 ♘f3 ♗f8 54 ♖b6 ♗×b4 55 ♖b8+ ♔g7 56 ♖b7+ ♔g8 57 d5 ♗×a5 58 ♘d4 ♗e8 59 ♖b8 ♖f8 60 ♘df5 ♗c3 61 ♘h6+ ♔h8 62 ♘hf5 a5 63 ♘c4 ♔g8 64 ♘h6+ ♔h8 65 ♔e3 ♗e1 66 ♘d6 ♗f2+ 67 ♔d2 ♗h4 68 ♘hf7+ ♗×f7 69 ♖×f8+ ♗g8 70 ♘e4 ♔g7 71 ♖d8 h6 72 g×h6+ ♔h7 73 ♖d6 ♗f7 74 ♖d7 ♔g8 75 h7+ ♔×h7 76 ♖×f7+ ♔g6 77 ♖b7 ♔f5 78 d6 ♔f4 79 ♖b8 ♔e5 80 d7 ♔e6 81 ♘c5+ ♔d6 82 d4 ♗g5+ 83 ♔e2 ♔c7 84 ♖a8 ♔b6 85 ♔d3 ♗e7 86 ♖e8 ♗g5 87 ♔c2 ♔c7 88 ♖a8 ♔b6 89 ♖g8 ♗e7 90 ♖b8+ ♔c6 91 d8(♕) ♗×d8 92 ♖×d8 ♔c7 93 ♖a8 ♔c6 94 ♖g8 ♔b5 95 ♔b3 ♔c6 96 ♔c4 ♔c7 97 ♔b5 ♔d6 98 ♖g7 a4 99 ♖d7# 1-0

*Round 3, October 20, 1976*

**Blitz IV – Etaoin Shrdlu** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 ♘f6 4 0-0 ♘×e4 5 d4 e×d4 6 ♖e1 d5 7 ♘bd2 ♗g4 8 ♘×e4 d×e4 9 ♖×e4+ ♗e6 10 ♘×d4 ♕d5 11 ♗×c6+ b×c6 12 ♕e2 ♖d8 13 ♘×e6 f×e6 14 ♖×e6+ ♔f7 15 ♖e3 ♕d1+ 16 ♕e1 ♕×c2 17 ♖f3+ ♔g6 18 ♕e6+ ♔h5 19 ♖h3# 1-0

**Chesstar – Chute 1.2** 1 d4 d5 2 h4 ♘f6 3 ♕d3 e6 4 ♗g5 ♗e7 5 ♘d2 ♘c6 6 a3 h6 7 ♗f4 ♘h5 8 e3 ♗×h4 9 ♗h2 a5 10 ♗e2 g6 11 ♖d1 ♘g7 12 ♘gf3 ♘f5 13 ♘×h4 ♘×h4 14 ♗f4 ♘×g2+ 15 ♔f1 ♘×f4 16 e×f4 ♕f6 17 ♘e4 d×e4 18 ♕×e4 h5 19 c4 0-0 20 b3 h4 21 d5 ♘e7 22 d6 c×d6 23 ♕d4 ♕×d4 24 ♖×d4 ♘f5 25 ♖d3 ♖e8 26 ♖hh3 a4 27 b4 e5 28 f×e5 ♖×e5 29 ♗f3 ♘g3+ 30 ♔g2 ♗×h3+ 31 ♔×h3 ♘e4 32 ♗×e4 ♖×e4 33 f3 ♖×c4 0-1

**Chaos – Duchess** 1 d4 d5 2 c4 e6 3 ♘c3 ♘f6 4 ♗g5 ♗e7 5 e3 0-0 6 ♘f3 h6 7 ♗h4 ♘e4 8 ♗×e7 ♕×e7 9 c×d5 ♘×c3 10 b×c3 e×d5 11 ♕b3 ♕d6 12 c4 d×c4 13 ♘×c4 ♘c6 14 ♖c1 ♘a5 15 ♕c2 ♘×c4 16 ♕×c4 ♗g4 17 h3 ♗×f3 18 g×f3 ♖fc8 19 ♖g1 c6 20 f4 ♖e8 21 ♔d2 a6 22 a4 a5 23 ♖b1 ♖e7 24 h4 ♖c8 25 ♕c3 ♕d5 26 ♕b3 ♕×b3 27 ♖×b3 ♖d8 28 ♔c2 f5 29 h5 ♔h8 30 ♖gb1 ♖dd7 31 ♖g1 ♖d5 32 ♖g6 b5 33 ♖×c6 b4 34 ♖c4 ♖b7 35 ♔c1 ♖e7 36 ♖c8+ ♔h7 37 ♖c6 ♔g8 38 ♖c8+ ♔f7 39 ♖b2 ♖e8 40 ♖×e8 ♔×e8 41 ♖c2 ♔d7 42 ♔b2 ♔d6 43 ♔b3 ♔d7 44 ♖c1 ♔e8 45 ♖c6 ♔f7 46 ♖a6 ♔g8 47 ♔c4 b3 48 ♔×b3 ♔f7 49 ♔c4 ♖d8 50 ♖×a5 ♔f6 51 ♖a6+ ♔f7 52 ♔b5 ♔g8 53 a5 ♔h8 54 ♖b6 ♖d5+ 55 ♔b4 ♖d7 56 ♖b5 ♖f7 57 a6 ♖a7 58 ♔a5 ♖f7 59 ♖b7 ♖f8 60 a7 ♔h7 61 ♖b8 ♖f7 62 a8(♕) ♖a7+ 63 ♔b6 ♖×a8 64 ♖×a8 g6 65 ♖a7+ ♔g8 66 h×g6 h5 67 ♖f7 h4 68 d5 h3 69 d6 h2 70 d7 h1(♕) 71 d8(♕)# 1-0

**Black Knight – Chess 4.5** 1 e4 c5 2 ♘f3 ♘c6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 d6 6 ♗g5 e6 7 ♕d2 ♗e7 8 0-0-0 0-0 9 f4 ♘×d4 10 ♕×d4 h6 11 ♗h4 ♕a5 12 ♗c4 ♖d8 13 e5 d×e5 14 ♕×e5 ♕b4 15 ♗b3 ♗d7 16 ♖d4 ♕b6 17 h3 ♗c6 18 ♖×d8+ ♖×d8 19 ♕e2 ♕d4 20 ♖f1 ♗b4 21 ♗e1 ♗c5 22 f5 ♕e3+ 23 ♗d2 ♕×e2 24 ♘×e2 ♗×g2 25 ♖f4 ♗×h3 26 f×e6 ♗×e6 27 ♘c3 ♗×b3 28 a×b3 ♘d5 29 ♖g4 f5 30 ♖a4 ♘e3 31 ♖f4 g5 32 ♖a4 ♘f1 33 ♗e1 ♖e8 34 ♗d2 ♘×d2 35 ♔×d2 g4 36 ♔d1 g3 37 ♔d2 g2 38 ♘e2 ♖×e2+ 39 ♔×e2 g1(♕) 40 ♖c4 b6 41 ♔d2 b5 42 ♖×c5 ♕×c5 43 c3 ♕e5 44 ♔c2 f4 45 ♔b1 f3 46 b4 f2 47 ♔c1 f1(♕)+ 48 ♔c2 ♕d5 49 b3 ♕dg2# 0-1

**Wita – L'Excentrique** 1 ♘f3 b6 2 d3 d5 3 ♗f4 e6 4 g3 ♕f6 5 ♗e5 ♕f5 6 ♗×c7 a5 7 ♗×b6 e5 8 ♗c7 ♘c6 9 ♘c3 ♗b4 10 ♘d2 ♕d7 11 ♗b6 ♖b8 12 ♘a4 ♕g4 13 c3 ♗e7 14 ♗g2 ♘f6 15 0-0 0-0 16 f4 e4 17 d×e4 d×e4 18 ♘×e4 ♘×e4 19 ♗×e4 ♖×b6 20 ♘×b6 ♗c5+ 21 ♔h1 ♗×b6 22 ♗×c6 ♕e6 23 ♗d5 ♕h3 24 ♕b3 ♗e6 25 ♗×e6 ♕×e6 26 ♕×e6 f×e6 27 ♔g2 a4 28 c4 ♖c8 29 b3 a3 30 ♖ad1 ♗e3 31 ♖d3 ♗c5 32 ♖d7 ♖f8 33 h3 g5 34 f×g5 ♖e8 35 e4 e5 36 ♖b7 ♖d8 37 b4 ♗e3 38 ♖ff7 ♖d2+ 39 ♔f1 ♖×a2 40 ♖g7+ ♔f8 41 ♖bf7+ ♔e8 42 c5 ♗×g5 43 c6 ♖c2 44 c7 ♖c1+ 45 ♔g2 a2 46 c8(♕)+ ♖×c8 47 ♖a7 ♖c2+ 48 ♔f3 ♖c3+ 49 ♔g4 h5+ 50 ♔×h5 ♗h6 51 ♖a8+ ♖c8 52 ♖×c8# 1-0

*Round 4, October 21, 1976*

**L'Excentrique – Chesstar** 1 d4 e5 2 d×e5 ♘c6 3 e4 a5 4 ♕d5 h6 5 a4 ♕e7 6 h4 ♕b4+ 7 c3 ♕b6 8 ♗c4 ♗c5 9 ♕×f7+ ♔d8 10 ♕×g7 d6 11 ♕f8+ 1-0

**Blitz IV – Wita** 1 e4 c5 2 ♘f3 d6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 a6 6 ♗g5 e6 7 ♗c4 ♗e7 8 0-0 ♕a5 9 ♘f3 ♕c5 10 ♕d4 ♘c6 11 ♕×e5 d×c5 12 e5 ♘g4 13 ♗×e7 ♔×e7 14 ♖ae1 ♖d8 15 h3 ♘h6 16 ♘g5 ♖d2 17 ♗d3 ♘b4 18 ♖d1 ♖×d1 19 ♖×d1 ♗d7 20 ♗e4 ♗c6 21 ♗×h7 ♖c8 22 ♖d6 ♘f5 23 ♖d2 f6 24 ♗×f5 e×f5 25 e×f6+ g×f6 26 ♖e2+ ♔f8 27 ♘e6+ ♔f7 28 ♘×c5 ♖g8 29 a3 ♗×g2 30 a×b4 b6 31 ♔h2 ♗f3 32 ♖e3 ♗h5 33 ♘×a6 f4 34 ♖e1 ♗f3 35 ♖g1 ♖×g1 36 ♔×g1 ♔g7 37 ♘c7 f5 38 ♘7d5 ♗×d5 39 ♘×d5 f3 40 ♘×b6 ♔g6 41 ♘d7 ♔g5 42 c4 ♔h4 43 ♔h2 ♔g5 44 ♔g3 f4+ 45 ♔×f3 ♔f5 46 ♘c5 ♔g5 1-0

**Duchess – Xenarbor IV** 1 e4 c5 2 d4 c×d4 3 c3 d×c3 4 ♘×c3 ♘c6 5 ♘f3 d6 6 ♗c4 e6 7 0-0 ♘f6 8 ♕e2 ♗e7 9 ♖d1 e5 10 ♗e3 ♘g4 11 ♗c5 h5 12

♘b5 ♘d4 13 ♘b×d4 d×c5 14 ♘f5 ♕c7 15 ♘×g7+ ♔f8 16 ♘f5 ♗×f5 17 e×f5 ♖d8 18 ♖×d8+ ♗×d8 19 ♖d1 ♗f6 20 h3 ♘h6 21 ♖d5 ♘×f5 22 ♘×e5 ♘d4 23 ♕e3 ♕e7 24 ♘d7+ ♔g7 25 ♘×f6 ♕×f6 26 ♖d7 ♘f5 27 ♕×c5 ♔g6 28 ♖×b7 ♘d6 29 ♗d3+ ♔g7 30 ♖×a7 ♖h6 31 ♖d7 h4 32 b4 ♕a1+ 33 ♔h2 ♕f6 34 a4 ♔g8 35 b5 ♕f4+ 36 ♔g1 ♕f6 37 b6 ♕a1+ 38 ♗f1 ♕×a4 39 ♖×d6 ♖×d6 40 b7 ♖e6 41 b8(♕)+ ♕e8 42 ♕g5+ ♔f8 43 ♕b4+ ♖d6 44 ♕×d6+ ♕e7 45 ♕d×e7# 1-0

**Chess 4.5 – Chaos** 1 e4 c5 2 ♘f3 ♘c6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 e6 6 ♘×c6 b×c6 7 e5 ♘d5 8 ♘×d5 c×d5 9 ♗d3 g6 10 0-0 ♗g7 11 ♕e2 ♗b7 12 b3 ♕e7 13 a4 ♕b4 14 ♗a3 ♕c3 15 f4 f6 16 ♗d6 f5 17 a5 ♔f7 18 ♕f2 ♗f8 19 ♗×f8 ♖h×f8 20 ♕h4 ♕c5+ 21 ♔h1 ♔g7 22 b4 ♕×b4 23 ♖ab1 ♕×a5 24 ♖×b7 ♕a4 25 ♗b5 1-0

**Chute 1.2 – Black Knight** 1 e4 e5 2 ♘f3 f5 3 e×f5 e4 4 ♘d4 ♕f6 5 ♘b5 ♘a6 6 ♕h5+ ♔d8 7 ♘5c3 g6 8 f×g6 ♘b4 9 ♘a3 h×g6 10 ♕e2 d5 11 h3 ♗d6 12 ♘cb5 ♗f4 13 g3 ♗e5 14 ♘c3 ♘e7 15 ♕b5 ♕d6 16 d3 ♗d7 17 ♕×b7 ♗c6 18 ♕×a8+ ♗×a8 19 d×e4 ♗×c3+ 20 b×c3 ♕e6 21 c×b4 ♕×e4+ 22 ♗e3 ♕×h1 23 h4 ♘f5 24 ♗g5+ ♔d7 25 ♔d2 ♕f3 26 ♗e3 d4 27 ♗b5+ c6 28 ♗×d4 ♘×d4 29 ♗d3 ♘b3+ 30 a×b3 ♕×f2+ 31 ♔c1 ♕×g3 32 ♘c4 ♔d8 33 ♔b2 ♕×h4 34 ♖d1 ♕d4+ 35 ♔a3 ♔e7 36 ♖e1+ ♔f6 37 ♘e5 c5 38 b×c5 ♕×c5+ 39 ♔b2 ♕d4+ 40 c3 ♖h2+ 41 ♔b1 ♕×c3 42 ♘g4+ ♔g5 43 ♘×h2 ♕×e1+ 44 ♔c2 ♕f2+ 45 ♔b1 ♕d2 46 ♗×g6 ♔×g6 47 ♘f1 ♗e4+ 48 ♔a1 ♕c1+ 49 ♔a2 ♗b1+ 50 ♔a1 ♕c2 51 b4 ♕a2# 0-1

## ACM 1977 – Seattle

| | rate | perf | 1 | 2 | 3 | 4 | tot |
|---|---|---|---|---|---|---|---|
| 1 Chess 4.6 | 1935 | 1972 | 8+ | 4+ | 3+ | 2= | 3½ |
| 2 Duchess | 1825 | 1939 | 5+ | 6+ | 7+ | 1= | 3½ |
| 3 Chaos | 1750 | 1771 | 10+ | 9+ | 1– | 5+ | 3 |
| 4 Xenarbor | 1244 | 1434 | 11+ | 1– | 6+ | 8= | 2½ |
| 5 Blitz V | 1573 | 1535 | 2– | 8+ | 9+ | 3– | 2 |
| 6 Black Knight | 1196 | 1283 | 12+ | 2– | 4– | 11+ | 2 |
| 7 Ostrich | 1400 | 1304 | 9– | 12+ | 2– | 10+ | 2 |
| 8 Chute 1.2 | 1300 | 1313 | 1– | 5– | 10+ | 4= | 1½ |
| 9 8080 Chess | 0 | 1188 | 7+ | 3– | 5– | 12– | 1 |
| 10 Tyro | 0 | 1190 | 3– | 11+ | 8– | 7– | 1 |
| 11 Wita | 1275 | 1044 | 4– | 10– | 12+ | 6– | 1 |
| 12 Brute Force | 0 | 1070 | 6– | 7– | 11– | 9+ | 1 |

*Round 1, October 15, 1977*

**Chaos – Tyro** 1 d4 d5 2 c4 e6 3 ♘c3 ♘f6 4 ♗g5 h6 5 ♗×f6 g×f6 6 c×d5 e×d5 7 ♘f3 ♗b4 8 g3 0-0 9 ♗g2 ♗×c3+ 10 b×c3 ♖e8 11 ♕d2 ♔g7 12 0-0 ♗f5 13 ♕b2 b6 14 ♘h4 ♕d7 15 ♘×f5+ ♕×f5 16 c4 c6 17 c×d5 c×d5 18 ♕b5 ♖d8 19 e4 ♕d7 20 ♕×d5 ♘c6 21 ♕×d7 ♖×d7 22 e5 ♖c8 23 ♗h3 ♖cc7 24 e×f6+ ♔×f6 25 ♗×d7 ♖×d7 26 ♖ac1 ♘×d4 27 ♔g2 ♘e2 28 ♖fd1 ♘×c1 29 ♖×d7 ♘×a2 30 ♖×a7 ♘b4 31 ♔f3 ♘d5 32 ♔e2 b5 33 ♔d3 ♘e7 34 ♖b7 ♘g6 35 ♖×b5 ♘e5+ 36 ♔e4 ♘g4 37 f3 ♘×h2 38 ♖b1 ♔e6 39 ♖h1 f5+ 40 ♔f4 ♘×f3 41 ♖×h6+ ♔f7 42 ♔×f3 ♔g7 43 ♖b6 ♔f7 44 ♔f4 ♔e7 45 ♔×f5 ♔d7 46 ♔f6 ♔c7 47 ♖e6 ♔d7 48 g4 ♔c7 49 g5 ♔d7 50 g6 ♔c7 51 g7 ♔d7 52 g8(♕) ♔c7 53 ♕f7+ ♔d8 54 ♖e8# 1-0

**Black Knight – Brute Force** 1 e4 d5 2 e×d5 ♕×d5 3 ♘c3 ♕e6+ 4 ♘ge2 ♗d7 5 d4 ♘a6 6 ♗g5 h6 7 ♗e3 ♕b6 8 ♕b1 g5 9 h4 g×h4 10 d5 ♕b4 11 ♗d4 e5 12 d×e6 f6 13 a3 ♕e7 14 e×d7+ ♔d8 15 ♕a2 c5 16 0-0-0 c×d4 17 ♘×d4 ♘c5 18 ♗c4 ♕g7 19 ♘e4 ♕×g2 20 ♘×c5 ♕g5+ 21 ♖d2 ♕×d2+ 22 ♔×d2 ♗×c5 23 ♘e6+ ♔×d7 24 ♘×c5+ ♔c7 25 ♗d5 ♘e7 26 ♗×b7 ♖ad8+ 27 ♔c3 ♖b8 28 ♕e6 ♘g8 29 ♕c6+ ♔d8 30 ♕d7# 1-0

**Xenarbor – Wita** 1 d4 ♘f6 2 c4 g6 3 ♘c3 ♗g7 4 e4 e6 5 ♘f3 d6 6 g3 ♘c6 7 ♗g2 h6 8 0-0 0-0 9 d5 e×d5 10 c×d5 ♘b8 11 ♗e3 ♖e8 12 ♕c2 ♕e7 13 ♖fd1 ♘×e4 14 ♘×e4 ♕×e4 15 ♕×c7 ♗×b2 16 ♖ab1 ♘a6 17 ♕×d6 ♗g7 18 ♘d4 ♗e5 19 ♗×e4 ♗×d6 20 ♗g2 ♗f8 21 ♖dc1 ♗c5 22 a4 f6 23 a5 ♖b8 24 ♖×c5 ♘×c5 25 ♘c6 ♗f5 26 ♖b5 a6 27 ♖b6 b×c6 28 d×c6 ♖×b6 29 a×b6 ♖e5 30 c7 h5 31 h4 a5 32 ♗f1 a4 33 ♗d4 a3 34 ♗×e5 f×e5 35 ♗c4+ ♔f8 36 ♔h2 ♔e7 37 f3 ♗c8 38 f4 e×f4 39 g×f4 ♘d7 40 ♗b3 ♗b7 41 ♗a2 ♔f6 42 ♗b3 ♔g7 43 f5 g×f5 44 ♗d5 ♗c8 45 b7 a2 46 b×c8(♕) ♘f6 47 ♗×a2 f4 48 ♕f5 f3 49 ♕g5+ 1-0

**8080 Chess – Ostrich** 1 e4 e5 2 d4 ♘f6 3 d×e5 ♘×e4 4 ♘f3 ♗c5 5 ♘d4 ♘c6 6 ♘×c6 ♗×f2+ 7 ♔e2 d×c6 8 ♕×d8+ ♔×d8 9 ♗f4 ♗d4 10 c3 ♗g4+ 11 ♔d3 ♘f2+ 12 ♔×d4 ♘×h1 13 ♗c4 ♗e6 14 ♗d3 ♔c8 15 ♘d2 ♖d8+ 16 ♔e3 ♘f2 17 ♗×h7 ♘g4+ 18 ♔e2 ♖h8 19 ♗e4 ♘×h2 20 b4 ♖h4 21 g3 ♖h5 22 a4 ♗d5 23 ♗d3 ♘g4 24 c4 ♗g2 25 ♖g1 ♗h3 26 ♘f3 a5 27 b×a5 ♖×a5 28 ♘d4 ♘×e5 29 ♗×e5 ♖h×e5+ 30 ♔f3 c5 31 ♘e2 ♖×a4 32 ♘f4 ♗e6 33 ♖c1 g5 34 ♘e2 ♖a2 35 ♖c3 ♗f5 36 ♘c1 ♖d2 37 ♗e2 ♖e4 38 ♗d3 1-0

**Chess 4.6 – Chute 1.2** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 ♘ge7 4 d4 e×d4 5 ♘×d4 ♘×d4 6 ♕×d4 a6 7 ♗e2 ♘c6 8 ♕c4 ♗d6 9 ♗e3 0-0 10 ♘c3 b5 11 ♕d5 ♗b7 12 0-0 ♕e8 13 a3 ♕e6 14 f4 ♖ae8 15 ♕×e6 f×e6 16 ♖ad1 e5 17 f5 ♘e7 18 b3 ♖f6 19 b4 ♖a8 20 ♗g5 ♖f7 21 ♗h5 g6 22 ♗×e7 ♗×e7 23 f×g6 ♖×f1+ 24 ♔×f1 ♖f8+ 25 ♔e1 ♗h4+ 26 g3 ♗×g3+ 27 h×g3 h×g6 28 ♗×g6 ♖f3 29 ♖d3 ♖×d3 30 c×d3 ♔g7 31 ♗f5 d6 32 ♘d5 c6 33 ♘b6 ♔h8 34 ♘c8 d5 35 ♘d6 ♗a8 36 e×d5 c×d5 37 ♗c8 e4 38 ♗×a6 ♗c6 39 ♗×b5 ♗×b5 40 ♘×b5 e×d3 41 a4 ♔g7 42 a5 ♔h6 43 a6 ♔h5 44 a7 d2+ 45 ♔×d2 ♔g4 46 a8(♕) ♔×g3 47 ♘a3 d4 48 b5 ♔f2 49 b6 ♔g3 50 ♕e4 ♔f2 51 b7 ♔g1 52 b8(♕) ♔f2 53 ♕e1+ ♔g2 54 ♕bg3# 1-0

**Blitz V – Duchess** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗×c6 d×c6 5 d4 e×d4 6 ♕×d4 ♕×d4 7 ♘×d4 ♗d7 8 ♗e3 0-0-0 9 ♘c3 ♗d6 10 0-0-0 ♘f6 11 ♖d3 ♘g4 12 g3 ♘×e3 13 ♖×e3 ♗c5 14 ♖d3 ♗g4 15 f3 ♗h5 16 ♘ce2 ♗×d4 17 ♘×d4 c5 18 ♘f5 ♖×d3 19 c×d3 ♗×f3 20 ♖f1 ♗e2 21 ♘e7+ ♔d8 22 ♖×f7 ♗×d3 23

♘d5 ♗×e4 24 ♘×c7 ♖g8 25 ♘e6+ ♔e8 26 ♖×g7 27 ♘×g7+ ♔e7 28 ♘h5 ♔d6 29 ♘f4 c4 30 ♔d2 ♗b1 31 a3 b5 32 g4 ♔e5 33 ♘e2 ♗d3 34 h4 ♗×e2 35 ♔×e2 ♔f4 36 g5 ♔g4 37 ♔e3 ♔×h4 38 ♔f4 a5 39 ♔f5 b4 40 a×b4 a×b4 41 g6 h×g6+ 42 ♔e4 g5 43 ♔d4 g4 44 ♔×c4 g3 45 ♔×b4 ♔g5 46 ♔a5 g2 47 b4 g1(♕) 48 b5 ♕a7+ 49 ♔b4 ♔f5 50 ♔c4 ♔e5 51 ♔b4 0-1

*Round 2, October 15, 1977*

**8080 Chess – Chaos** 1 e4 c5 2 d4 c×d4 3 ♕×d4 ♘c6 4 ♕d5 ♘f6 5 ♕d3 e5 6 ♘f3 ♗c5 7 ♗g5 ♕b6 8 ♗×f6 ♗×f2+ 9 ♔d2 ♕×b2 10 ♗×g7 ♖g8 11 ♕c3 ♕×c3+ 12 ♔×c3 ♖×g7 13 ♘bd2 d6 14 ♘c4 ♗c5 15 h4 ♗e6 16 ♘b2 ♖g4 0-1

**Wita – Tyro** 1 d4 d5 2 c4 e6 3 e4 d×e4 4 ♘c3 ♗b4 5 a3 ♗×c3+ 6 b×c3 ♘f6 7 ♖b1 0-0 8 ♗e2 b6 9 ♗f4 c6 10 ♘h3 ♕e7 11 ♕a4 ♘fd7 12 0-0 ♗a6 13 ♗×b8 ♘×b8 14 ♖fd1 f5 15 ♔h1 e5 16 d×e5 ♕×e5 17 ♕b3 ♔h8 18 ♘g5 ♕f6 19 ♘h3 ♖d8 20 ♖×d8+ ♕×d8 21 ♘f4 ♕h4 22 g3 ♕f6 23 ♖d1 ♗c8 24 ♘h5 ♕e7 25 h3 g6 26 c5 g×h5 27 c×b6 a×b6 28 f4 b5 29 ♗×h5 ♗e6 30 ♕b4 ♕×b4 31 c×b4 ♗d5 32 g4 f×g4 33 ♔h2 g×h3 34 f5 ♖×a3 35 ♗g4 ♖a2+ 36 ♔×h3 ♘d7 37 ♗h5 ♘f6 38 ♔h4 ♖h2+ 39 ♔g5 ♘×h5 40 f6 ♔g8 41 ♖g1 ♔f7 42 ♖a1 ♘×f6 43 ♔f5 ♖f2+ 44 ♔g5 h5 45 ♔h6 ♖g2 46 ♖a7+ ♔e6 47 ♖a1 e3 48 ♖a6 ♔f5 49 ♖a2 ♖g6# 0-1

**Chute 1.2 – Blitz V** 1 e4 e5 2 ♘f3 ♘f6 3 ♘×e5 d6 4 ♘f3 ♘×e4 5 ♕e2 ♕e7 6 d3 ♘f6 7 ♗g5 ♘bd7 8 ♕×e7+ ♗×e7 9 ♗e2 0-0 10 0-0 h6 11 ♗d2 ♘b6 12 ♗a5 ♗e6 13 ♘c3 ♖fe8 14 ♗×b6 a×b6 15 ♘d4 ♗d5 16 ♘×d5 ♘×d5 17 ♗f3 c6 18 ♗×d5 c×d5 19 ♘f5 ♖ac8 20 ♖ae1 ♗f8 21 ♖×e8 ♖×e8 22 d4 ♖e2 23 ♘e3 ♖d2 24 ♘×d5 ♖×c2 25 ♖b1 ♖d2 26 ♘×b6 ♖×d4 27 ♘d7 ♖d2 28 ♔f1 d5 29 ♔e1 ♖c2 30 ♘×f8 ♔×f8 31 ♖d1 ♖×b2 32 a4 ♖a2 33 ♖×d5 ♖×a4 34 ♖d8+ ♔e7 35 ♖d3 b5 36 ♖e3+ ♔d6 37 ♖d3+ ♔c6 38 ♖c3+ ♔b6 39 ♖c1 ♖e4+ 40 ♔d2 b4 41 f3 ♖e5 42 ♖c4 ♖b5 43 g3 b3 44 ♖c8 ♔b7 45 ♖f8 b2 46 ♖×f7+ ♔b6 47 ♖×g7 b1(♕) 48 ♖d7 ♕a2+ 49 ♔d3 0-1

**Duchess – Black Knight** 1 e4 c5 2 d4 c×d4 3 c3 d×c3 4 ♘×c3 ♘c6 5 ♘f3 d6 6 ♗c4 e6 7 0-0 ♘f6 8 ♕e2 ♗e7 9 ♖d1 e5 10 ♗e3 ♘g4 11 ♗d2 ♗h4 12 ♗e1 ♗e7 13 ♗d2 ♗e6 14 ♗×e6 f×e6 15 ♘g5 ♗×g5 16 ♕×g4 ♗×d2 17 ♕×g7 ♖f8 18 ♕×b7 ♕b6 19 ♕×a8+ ♔e7 20 ♕×f8+ ♔×f8 21 ♖×d2 ♕c7 22 ♖ad1 ♘d4 23 f4 ♕g7 24 ♖d3 h5 25 ♔h1 a6 26 f×e5 d×e5 27 ♖f1+ ♔e8 28 ♖g3 ♕e7 29 ♖g8+ ♔d7 30 ♖a8 ♔c6 31 ♖×a6+ ♔b7 32 ♖a4 ♔c6 33 ♖a5 ♕b4 34 ♖×e5 ♕×b2 35 ♖c5+ ♔d6 36 e5+ ♔e7 37 ♖b1 ♕f2 38 ♖c7+ ♔d8 39 ♖h7 ♘c6 40 ♖h8+ ♔c7 41 ♘b5+ ♔d7 42 ♖h7+ ♔d8 43 ♘d6 ♕f8 44 ♔g1 h4 45 ♖bb7 ♘e7 46 ♖b8+ ♔c7 47 ♖×f8 h3 48 ♖×e7+ ♔c6 49 g×h3 ♔d5 50 ♘f7 ♔d4 51 ♖×e6 ♔d5 52 ♖d6+ ♔e4 53 ♖c8 ♔f3 54 ♖c4 ♔e2 55 ♖e4+ ♔f3 56 ♘g5# 1-0

**Ostrich – Brute Force** 1 e4 d5 2 e×d5 ♕×d5 3 ♘c3 ♕e5+ 4 ♘ge2 a5 5 d4 ♕f5 6 ♗e3 g5 7 ♘g3 ♕g6 8 ♗d3 f5 9 0-0 ♘f6 10 ♘b5 ♔d8 11 d5 g4 12 ♗d4 e6 13 d×e6 ♗b4 14 ♗×f5 ♔e7 15 ♗×g6 h×g6 16 ♗×f6+ ♔×f6 17 ♕d4+ ♔e7 18 ♕g7+ ♔×e6 19 ♘×c7+ ♔d6 20 ♘×a8 ♖e8 21 ♕c7+ ♔e6 22 ♖fe1+ ♗×e1 23 ♖×e1+ ♔f6 24 ♖×e8 ♗d7 25 ♕e5+ ♔f7 26 ♕e7# 1-0

**Xenarbor – Chess 4.6** 1 d4 ♘f6 2 c4 c5 3 d5 e6 4 ♘f3 e×d5 5 c×d5 d6 6 e3 ♗f5 7 ♘c3 ♘bd7 8 g3 ♘e5 9 ♘×e5 d×e5 10 ♕b3 ♕b6 11 ♕×b6 a×b6 12 ♗g2 ♗d3 13 e4 ♗d6 14 ♗e3 0-0 15 0-0-0 ♗c4 16 b3 ♗a6 17 ♘a4 ♗b5 18 ♘×b6 ♖a6 19 a4 ♗e2 20 ♖d2 ♖×b6 21 ♖×e2 ♖×b3 22 ♖c2 ♖c8 23 ♗h3 ♖a8 24 ♖c4 b5 25 ♖c2 ♖×a4 26 ♗g2 c4 27 h4 c3 28 ♖e2 ♘×e4 29 ♗×e4 ♖×e4 30 ♖a2 ♖a3 31 ♖×a3 ♗×a3+ 32 ♔c2 ♗b4 33 ♖a1 f5 34 ♖a8+ ♔f7 35 ♖b8 f4 36 ♔d3 f×e3 37 f×e3 ♖c4 38 ♖×b5 c2 39 ♔×c4 ♗e7 40 e4 c1(♕)+ 41 ♔d3 ♕f1+ 42 ♔e3 ♕×b5 43 g4 ♗×h4 44 ♔d2 ♕c4 45 d6 ♕×e4 46 g5 ♗×g5+ 47 ♔c3 h5 48 ♔b3 h4 49 ♔a2 ♕b4 50 ♔a1 ♗c1 51 ♔a2 ♕b2# 0-1

*Round 3, October 16, 1977*

**Chess 4.6 – Chaos** 1 e4 c5 2 ♘f3 ♘c6 3 ♗b5 ♘f6 4 e5 ♘d5 5 0-0 e6 6 ♗×c6 d×c6 7 d3 ♗e7 8 ♘a3 0-0 9 ♗d2 ♗d7 10 ♕e2 ♕b6 11 ♘c4 ♕d8 12 ♘d6 ♕b6 13 c4 ♕×b2 14 c×d5 c×d5 15 ♖fb1 ♕a3 16 ♖×b7 ♕a4 17 ♖c1 ♖ab8 18 ♖c7 ♖b2 19 ♖7×c5 ♖fb8 20 ♕e3 ♕×a2 21 ♖c7 ♗×d6 22 ♖×d7 ♗a3 23 ♕f4 f5 24 e×f6 ♗f8 25 f×g7 ♗e7 26 ♖×e7 h6 27 ♕f7+ ♔h7 28 g8(♕)# 1-0

**Chute 1.2 – Tyro** 1 d4 d5 2 c4 e6 3 g3 d×c4 4 ♘f3 ♘c6 5 a3 ♘f6 6 ♘c3 a6 7 e4 b5 8 ♗f4 ♘h5 9 ♗h3 ♘×f4 10 g×f4 ♕f6 11 ♕d2 ♕h6 12 ♘g5 ♘d8 13 d5 ♗e7 14 ♗g4 f5 15 e×f5 e×f5 16 ♗f3 ♗b7 17 0-0 ♕b6 18 ♖ae1 h6 19 ♕e3 ♕f6 20 ♘e6 ♖c8 21 ♘×d8 ♔×d8 22 ♗h5 g5 23 ♔h1 g×f4 24 ♕×f4 ♗d6 25 ♕f3 ♖g8 26 ♖g1 ♖×g1+ 27 ♖×g1 ♕e5 28 ♕g2 ♔d7 29 ♖d1 ♗c5 30 ♖d2 ♕e1+ 31 ♕g1 ♕×d2 32 ♕g7+ ♗e7 33 ♕e5 ♕×f2 34 ♕e6+ ♔d8 35 ♕×h6 ♕f1# 0-1

**Black Knight – Xenarbor** 1 e4 c5 2 ♘f3 ♘c6 3 ♘c3 d6 4 d4 c×d4 5 ♘×d4 ♘×d4 6 ♕×d4 ♘f6 7 e5 d×e5 8 ♕×e5 a6 9 ♗e2 ♕d6 10 ♕×d6 e×d6 11 ♗g5 ♗e6 12 ♗×f6 g×f6 13 ♗f3 0-0-0 14 0-0-0 ♗h6+ 15 ♔b1 ♗g7 16 ♖d3 f5 17 ♖hd1 ♗e5 18 ♗d5 ♗×d5 19 ♘×d5 ♗×h2 20 ♖h3 ♗e5 21 f4 h5 22 f×e5 d×e5 23 ♖f3 f4 24 g3 f×g3 25 ♖×g3 h4 26 ♖g7 f5 27 ♘b6+ ♔b8 28 ♖dd7 ♖×d7 29 ♘×d7+ ♔a7 30 ♘×e5 ♖e8 31 ♘c6+ ♔a8 32 ♘e7 f4 33 ♖g8 ♖×g8 34 c4 ♖g1+ 35 ♔c2 f3 36 ♘c8 ♔b8 37 ♘b6 f2 38 ♘d7+ ♔a7 39 ♘e5 f1(♕) 40 ♔c3 h3 41 c5 h2 42 a4 h1(♕) 43 b4 ♕a1+ 44 ♔d3 ♕×e5 45 a5 ♕he4+ 46 ♔d2 ♕5d4# 0-1

**Blitz V – 8080 Chess** 1 e4 e5 2 ♘f3 d5 3 ♘×e5 d×e4 4 ♗c4 ♘h6 5 0-0 ♘c6 6 ♘×c6 b×c6 7 ♕e2 ♗f5 8 d3 ♗c5 9 ♗×h6 g×h6 10 ♕h5 ♕f6 11 d×e4 ♕×b2 12 ♕×f7+ ♔d8 13 ♕×f5 ♗×f2+ 14 ♕×f2 ♕×a1 15 ♕h4+ ♔d7 16 ♖d1+ ♔e8 17 ♕h5+ 1-0

**Ostrich – Duchess** 1 e4 e5 2 ♘f3 ♘f6 3 ♗c4 ♘×e4

4 ♕e2 d5 5 ♘xe5 ♗e6 6 0-0 ♗d6 7 ♘f3 0-0 8 d3 dxc4 9 dxe4 ♘c6 10 ♘g5 ♘d4 11 ♕h5 h6 12 ♘xe6 fxe6 13 ♕d1 c5 14 c3 ♘c6 15 ♘a3 ♕b8 16 h4 ♘e5 17 ♗e3 ♗e7 18 f4 ♘d3 19 ♘xc4 ♘xf4 20 g3 ♘g6 21 ♕g4 ♔h7 22 h5 ♘e5 23 ♖xf8 ♘xg4 24 ♖xb8 ♖xb8 25 ♗f4 ♖d8 26 ♘e5 ♘xe5 27 ♗xe5 ♗f6 28 ♗xf6 gxf6 29 ♖f1 ♔g7 30 ♔g2 ♔f7 31 ♔f3 ♖d3+ 32 ♔g4 ♖d2 33 ♖b1 ♖f2 34 a4 ♔e7 35 b4 c4 36 ♖e1 ♖c2 37 ♖e3 b6 38 a5 bxa5 39 bxa5 ♔d6 40 ♔f4 ♖a2 41 ♖f3 ♖xa5 42 ♔e3 ♔e5 43 ♖f2 ♖a1 44 ♖c2 ♖e1+ 45 ♔d2 ♖xe4 46 ♖c1 ♖g4 47 ♖e1+ ♔d5 0-1

**Brute Force — Wita** 1 d4 ♘f6 2 ♘c3 ♘c6 3 d5 ♘xd5 4 ♕xd5 d6 5 a3 e6 6 ♕d2 ♗e7 7 b4 ♗f6 8 b5 ♘e5 9 f4 ♘g4 10 ♘f3 c6 11 bxc6 bxc6 12 h3 ♗h4+ 13 ♔d1 ♘f2+ 14 ♔e1 ♘xh1+ 15 g3 ♘xg3 16 ♕d3 ♕f6 17 ♘xh4 ♕xh4 18 ♕f3 d5 19 ♕g4 ♕xg4 20 hxg4 ♘xf1 21 ♔xf1 a6 22 ♗e3 e5 23 fxe5 ♗xg4 24 a4 h5 25 a5 h4 26 ♗a4 ♗e6 27 ♗f2 h3 28 e3 h2 29 ♗g3 h1(♕)+ 30 ♔e2 ♕c1 31 ♔d3 ♖b8 32 e4 ♖b2 33 ♖a3 ♖xc2 34 ♘a2 ♕d2# 0-1

*Round 4, October 17, 1977*

**Duchess — Chess 4.6** 1 e4 c5 2 d4 cxd4 3 c3 dxc3 4 ♘xc3 ♘c6 5 ♘f3 d6 6 ♗c4 e6 7 0-0 ♘f6 8 ♕e2 ♗e7 9 ♖d1 e5 10 ♗e3 ♘g4 11 ♗d2 ♘d4 12 ♘xd4 exd4 13 ♘b5 ♕b6 14 ♗f4 ♘e5 15 ♘xd4 ♘xc4 16 ♕xc4 ♗g4 17 ♕a4+ ♗d7 18 ♕b3 0-0 19 ♕xb6 axb6 20 a3 ♖fc8 21 ♖ac1 ♖xc1 22 ♖xc1 ♖a4 23 ♗e3 ♗d8 24 f3 ♖a5 25 ♖d1 ♗f6 26 ♖d2 ♖c5 27 ♘e2 ♖c6 28 a4 ♗e6 29 ♘d4 ♖c1+ 30 ♔f2 ♗d7 31 ♘b5 ♗xb5 32 axb5 ♖b1 33 ♗xb6 ♖xb2 34 ♖xb2 ♗xb2 35 ♔e3 ♔f8 36 ♗d4 ♗xd4+ 37 ♔xd4 ♔e8 38 ♔d5 ♔d7 39 b6 g6 40 f4 ♔e7 41 h3 f6 42 h4 ♔d7 43 g4 h6 44 h5 gxh5 45 gxh5 ♔e7 46 ♔d4 ♔e6 47 ♔c4 ♔e7 48 f5 ♔d8 49 ♔b4 ♔d7 50 ♔b5 ♔e8 51 ♔a4 ♔e7 52 ♔b4 ♔d8 53 ♔c4 ♔e8 54 ♔d4 ♔d7 55 ♔d5 ♔e7 56 ♔d4 ♔e8 57 ♔c4 ♔f7 58 ♔d3 ½- ½

**Chaos — Blitz V** 1 d4 d5 2 c4 e6 3 ♘c3 ♘f6 4 ♗g5 ♗e7 5 e3 0-0 6 ♘f3 h6 7 ♗h4 ♘e4 8 ♗xe7 ♕xe7 9 cxd5 ♘xc3 10 bxc3 exd5 11 ♕b3 ♖d8 12 ♗d3 ♘c6 13 0-0 ♖d6 14 ♖fc1 b6 15 c4 dxc4 16 ♕xc4 ♗g4 17 ♘d2 ♖ad8 18 ♗e4 ♗d7 19 ♖ab1 a5 20 ♗d5 ♘b4 21 ♗xf7+ ♕xf7 22 ♕xc7 ♗e6 23 a3 ♕xc7 24 ♖xc7 ♘d5 25 ♖cc1 ♖c8 26 ♖xc8+ ♗xc8 27 ♖c1 ♗b7 28 g3 b5 29 ♘b3 a4 30 ♘c5 ♗c6 31 e4 ♘b6 32 d5 ♗e8 33 ♘b7 ♖d7 34 ♘c5 ♖c7 35 ♖c3 ♗g6 36 f4 ♘c4 37 ♘a6 ♖e7 38 e5 ♗e4 39 d6 ♖f7 40 ♘c7 ♗c6 41 e6 ♖f8 42 e7 ♖c8 43 ♖d3 ♗d7 44 ♘a6 ♔f7 45 ♔f2 ♔e8 46 ♘c7+ ♖xc7 47 dxc7 ♔xe7 48 ♔f3 ♗f5 49 ♖d5 ♗g4+ 50 ♔xg4 ♘d6 51 ♖xd6 1-0

**Chute 1.2 — Xenarbor** 1 e4 c5 2 c4 ♘c6 3 ♘f3 e5 4 ♘c3 ♘f6 5 ♗e2 d6 6 0-0 g6 7 d3 ♗g7 8 ♗g5 ♘d4 9 ♘xd4 cxd4 10 ♘d5 h6 11 ♘xf6+ ♗xf6 12 ♗xf6 ♕xf6 13 ♕a4+ ♗d7 14 ♕a5 b6 15 ♕d5 ♖b8 16 ♖ac1 a5 17 b3 h5 18 ♗f3 h4 19 a4 h3 20 ♖fe1 ♖c8 21 ♕b7 ♕d8 22 ♕a7 ♗c6 23 gxh3 ♖a8 24 ♕xa8 ♕xa8 25 ♗g4 g5 26 ♖c2 ♕d8 27 ♖ce2 ♕e7 28 f3 ♕f6 29 ♔g2 ♕f4 30 ♗f5 0-0 31 ♗g4 f6 32 ♗e6+ ♔h8 33 ♗f5 ♖g8 34 ♗g4 ♖e8 35 ♗h5 ♖g8 36 ♗g4 ♖g7 37 ♗c8 ♖h7 38 ♗f5 ♖h4 39 ♗g4 ♖h7 40 ♗f5 ♖h6 41 ♗g4 ♖h4 42 ♗f5 ♖h5 43 ♗g4 ♖h6 44 ♗e6 ♖h7 45 ♗f5 ♖g7 46 ♗g4 ½-½

**Tyro — Ostrich** 1 d4 d5 2 c4 dxc4 3 e4 ♘c6 4 d5 ♘e5 5 f4 ♗g4 6 ♘f3 ♗xf3 7 gxf3 ♘d3+ 8 ♗xd3 cxd3 9 ♕xd3 c6 10 ♔d2 ♕d7 11 dxc6 ♕xd3+ 12 ♔xd3 0-0-0+ 13 ♔e2 bxc6 14 ♖d1 ♖d6 15 ♗e3 ♖h6 16 ♗xa7 ♖xh2+ 17 ♗f2 h5 18 b3 ♖h6 19 a4 ♖f6 20 ♔e3 ♖h3 21 b4 e5 22 fxe5 ♖f×f3+ 23 ♔e2 ♗xb4 24 ♗b6 ♘e7 25 ♖d4 ♖e3+ 26 ♔f2 ♗e1+ 27 ♔f1 ♖hf3+ 28 ♔g1 ♖e2 29 ♖d8+ ♔b7 30 ♗d4 ♖xe4 31 ♖d7+ ♔c8 32 ♖xe7 ♖g4+ 33 ♔h2 ♖xd4 34 ♔g2 ♖f2+ 35 ♔g3 ♖d3+ 36 ♔h4 ♖f5# 0-1

**Wita — Black Knight** 1 e4 c5 2 ♘f3 ♘c6 3 d4 cxd4 4 ♘xd4 ♘f6 5 ♘c3 d6 6 h3 a6 7 g4 ♘xd4 8 ♕xd4 e5 9 ♕d3 ♗e6 10 ♗g2 ♗e7 11 ♗e3 ♖c8 12 ♕d2 h5 13 g5 ♘d7 14 ♘d5 ♗xd5 15 exd5 ♖c4 16 ♕e2 b5 17 b3 ♖h4 18 f4 0-0 19 0-0 exf4 20 ♗f2 f3 21 ♕xf3 ♗xg5 22 ♗xh4 ♗xh4 23 ♕xh5 ♗f6 24 ♖ae1 ♗c3 25 a3 ♕b6+ 26 ♔h2 ♘f6 27 ♕d1 ♗xe1 28 ♖xe1 ♖c8 29 ♖e2 ♖c3 30 ♕d2 ♕a5 31 ♖e3 ♖xb3 32 ♕xa5 ♖xe3 33 a4 ♖e2 34 ♕xa6 ♖xc2 35 ♕a8+ ♔h7 36 axb5 ♖b2 37 ♕c6 ♘e4 38 ♕e8 ♘g5 39 ♕e7 ♔h6 40 b6 f5 41 ♕e3 ♔g6 42 h4 ♘e4 43 ♔g1 ♖b1+ 44 ♕e1 ♖xe1+ 45 ♔h2 ♖b1 46 ♗f1 ♖xf1 47 b7 ♖b1 48 b8(♕) ♖xb8 49 ♔g1 ♘c3 50 h5+ ♔xh5 51 ♔h2 g5 52 ♔g3 g4 53 ♔g2 f4 54 ♔f2 ♖b1 55 ♔g2 ♘e4 56 ♔h2 ♔h4 57 ♔g2 f3+ 58 ♔h2 g3# 0-1

**Brute Force — 8080 Chess** 1 d4 d5 2 ♘c3 ♘c6 3 ♗f4 ♘f6 4 ♘b5 e5 5 ♗xe5 ♗b4+ 6 c3 ♘xe5 7 dxe5 ♗xc3+ 8 ♘xc3 ♘e4 9 ♘xe4 dxe4 10 ♕a4+ ♗d7 11 ♕xe4 b5 12 g4 0-0 13 ♗g2 c6 14 h4 a5 15 0-0-0 ♖a6 16 ♕d4 f5 17 ♕xd7 ♕xd7 18 ♖xd7 fxg4 19 ♖d6 ♖xf2 20 ♗xc6 ♖f5 21 ♗d5+ ♔f8 22 ♖xa6 ♖xe5 23 e4 a4 24 ♖b6 g5 25 hxg5 ♔g7 26 ♖b7+ ♔g6 27 ♖hxh7 ♔xg5 28 ♖xb5 ♔f4 29 ♖a5 a3 30 ♖f7+ ♔e3 31 ♖xa3+ ♔d4 32 ♖g7 ♔c5 33 ♖xg4 ♔d6 34 ♖a6+ ♔c5 35 ♘f3 ♖e7 36 ♖c6+ ♔b4 37 e5+ ♔b5 38 ♘d4+ ♔a5 39 ♘b3+ 1-0

**ACM 1978 — Washington D.C.**

| | rate | perf | 1 | 2 | 3 | 4 | tot |
|---|---|---|---|---|---|---|---|
| 1 Belle | 1950 | 2244 | 8+ | 2+ | 3+ | 4+ | 4 |
| 2 Chess 4.7 | 2040 | 1961 | 9+ | 1— | 6+ | 7+ | 3 |
| 3 Chaos | 1775 | 1797 | 6+ | 7+ | 1— | 6= | 2½ |
| 4 Blitz 6.5 | 1670 | 1724 | 10+ | 6= | 11+ | 11— | 2½ |
| 5 Sargon II | 0 | 1448 | 3— | 12+ | 8= | 11+ | 2½ |
| 6 Duchess | 1850 | 1744 | 11+ | 4= | 2— | 3= | 2 |
| 7 Ostrich | 1450 | 1454 | 12+ | 3— | 9+ | 2— | 2 |
| 8 Mike | 0 | 1268 | 1— | 9= | 5= | 10= | 1½ |
| 9 Black Knight | 1174 | 1183 | 2— | 8= | 7— | 12+ | 1½ |
| 10 BS'66'76 | 0 | 1133 | 4— | 11— | 12+ | 8= | 1½ |
| 11 Awit | 1300 | 1275 | 6— | 10+ | 4— | 5— | 1 |
| 12 Brute Force | 0 | 592 | 7— | 5— | 10— | 9— | 0 |

*Round 1, December 3, 1978*

**Brute Force – Ostrich** 1 e4 c6 2 ♘f3 d5 3 ♘c3 d×e4 4 ♘×c4 e6 5 ♗c4 b5 6 ♗e2 ♗e7 7 0-0 ♗a6 8 d4 ♘f6 9 ♘×f6+ ♗×f6 10 ♘e5 0-0 11 ♗e3 ♕c7 12 f4 h6 13 ♕d3 c5 14 ♗f3 b4 15 c4 b×c3 0-1

**Chaos – Sargon** 1 d4 d5 2 c4 ♘c6 3 ♘c3 d×c4 4 ♘f3 ♗g4 5 d5 ♗×f3 6 e×f3 ♘e5 7 ♗f4 ♘d3+ 8 ♗×d3 c×d3 9 0-0 ♘f6 10 ♕b3 c6 11 ♕×b7 c×d5 12 ♕c6+ ♘d7 13 ♘×d5 ♖c8 14 ♘c7+ ♖×c7 15 ♗×c7 ♕c8 16 ♖fd1 g5 17 ♖×d3 e6 18 ♖ad1 ♗g7 19 ♖×d7 0-0 20 ♖d8 ♗×b2 21 ♖×c8 ♖×c8 22 ♕b7 ♖×c7 23 ♕×c7 ♗g7 24 ♖d8+ ♗f8 25 ♕e7 h5 26 ♖×f8+ ♔g7 27 ♖×f7+ ♔g6 28 ♕×e6# 1-0

**Chess 4.7 – Black Knight** 1 e4 c5 2 ♘f3 ♘c6 3 ♗b5 g6 4 0-0 ♗g7 5 c3 a6 6 ♗×c6 d×c6 7 d4 c×d4 8 ♘×d4 e5 9 ♘f3 ♗e6 10 ♘a3 ♕×d1 11 ♖×d1 ♖b8 12 ♘g5 ♖d8 13 ♖×d8+ ♔×d8 14 ♘×e6+ f×e6 15 ♗e3 ♘f6 16 f3 b5 17 ♖d1+ ♔c7 18 ♘c2 a5 19 c4 b×c4 20 ♘a3 c3 21 b×c3 ♗f8 22 ♘c4 ♘d7 23 ♘×a5 ♗e7 24 ♘c4 ♖b8 25 ♗a7 ♖f8 26 a4 c5 27 a5 ♖g8 28 a6 ♖h8 29 ♖b1 ♗g5 30 ♖b7+ ♔c6 31 ♘a5+ ♔d6 32 ♗b8+ ♖×b8 33 ♖×b8 ♗e3+ 34 ♔h1 ♗d2 35 a7 ♗×c3 36 ♖b7 ♗a1 37 a8(♕) ♔e7 38 ♕c8 ♔f6 39 ♕d8+ ♔g7 40 ♖×d7+ ♔h6 41 ♕h4# 1-0

**Blitz – BS'66'76** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗a4 ♘f6 5 d3 ♗b4+ 6 c3 0-0 7 c×b4 ♕e7 8 ♗×c6 d×c6 9 0-0 ♗g4 10 a3 ♖fd8 11 h3 ♗×f3 12 ♕×f3 ♕d6 13 ♖d1 c5 14 b×c5 ♕×c5 15 ♗g5 ♕c2 16 ♗×f6 g×g6 17 ♖d2 ♕c5 18 ♕×f6 ♖d6 19 ♕f5 ♕c1+ 20 ♔h2 ♖f8 21 ♕×e5 ♖c6 22 f3 ♖c5 23 ♕d4 b5 24 ♕g1 ♖e8 25 d4 ♖c2 26 ♕×c1 ♖×c1 27 ♔g3 ♖e6 28 ♔f4 h6 20 d5 ♖f6+ 30 ♔e3 ♖g6 31 e5 ♖e1+ 32 ♔f4 c5 33 h4 c4 34 d6 f6 35 d7 f×e5+ 36 ♔f5 ♔f7 37 d8(♕) 1-0

**Mike – Belle** 1 e4 e5 2 d4 e×d4 3 ♕×d4 ♘c6 4 ♕a4 ♘f6 5 ♗b5 ♗c5 6 ♘e2 ♘g4 7 f3 ♘f2 8 0-0 ♘g4+ 9 ♔h1 ♘×h2 10 ♕c4 ♕h4 11 g3 ♕h3 12 ♘f4 ♕×f1+ 13 ♕×f1 ♘×f1 14 ♗×f1 ♘d4 15 ♘d5 ♗d6 16 ♗f4 ♘×c2 17 ♗×d6 c×d6 18 ♗d3 ♘×a1 19 ♘c7+ ♔d8 20 ♘×a8 b5 21 ♘c3 ♗b7 22 ♘×b5 ♗×a8 23 ♘×d6 ♔e7 24 ♘f5+ ♔f6 25 b4 ♖c8 26 ♘d6 ♖c1+ 27 ♔h2 ♖d1 28 e5+ ♔×e5 29 ♗e4 ♖×d6 30 ♗×a8 ♖d2+ 31 ♔h3 ♖×a2 32 f4+ ♔d4 33 ♗f3 d5 34 b5 ♖b2 35 ♗h5 g6 36 ♗d1 ♘c2 37 ♗×c2 ♖×c2 38 ♔h4 f6 39 ♔h3 ♔e4 40 ♔g4 ♖h2 41 f5 g×f5# 0-1

**Awit – Duchess** 1 b3 d5 2 ♗b2 c5 3 e3 ♘f6 4 ♘f3 ♘c6 5 ♗b5 ♕b6 6 ♘a3 ♗f5 7 ♘h4 ♗g4 8 f3 ♗d7 9 g3 a6 10 ♗×c6 ♕×c6 11 0-0 b5 12 c4 d×c4 13 b×c4 g5 14 ♘g2 ♗g7 15 ♗×f6 16 ♖b1 b4 17 ♘c2 ♕a4 18 a3 b×a3 19 ♖b7 a2 20 d4 ♕×c4 21 ♕d2 ♗f5 22 ♘a1 c×d4 23 e4 ♗e6 24 ♖c1 ♕a4 25 f4 g×f4 26 ♘×f4 ♗g5 27 h4 ♗×f4 28 ♕×f4 ♖d8 29 ♕g5 ♕a3 30 ♖c6 f6 31 ♕h5+ ♗f7 32 ♕g4 ♖g8 33 ♕f4 ♕×g3+ 34 ♕×g3 ♖×g3+ 35 ♔f2 ♖e3 36 ♖cc7 ♖×e4 37 ♖c1 ♖×h4 38 ♖a7 ♖h3 39 ♖e1 e5 40 ♖×a6 ♔e7 41 ♔g2 ♖e3 42 ♖×e3 d×e3 43 ♔f3 e2 44 ♘c2 ♗c4 45 ♖a4 ♖d2 46 ♘e1 ♖d1 47 ♖a7+ ♔e6 0-1

*Round 2, December 3, 1978*

**Belle – Chess 4.7** 1 e4 ♘c6 2 d4 d5 3 ♘c3 e6 4 ♘f3 ♗b4 5 e5 ♘ge7 6 ♗d2 ♘f5 7 ♘e2 ♗e7 8 c3 0-0 9 ♘f4 f6 10 ♗d3 f×e5 11 d×e5 g5 12 g4 ♘g7 13 ♘g2 b6 14 ♕e2 ♗b7 15 ♖g1 a5 16 a4 ♔h8 17 h3 ♔g8 18 ♖h1 h6 19 h4 d4 20 h×e5 ♘b4 21 g×h6 ♘×d3+ 22 ♕×d3 d×c3 23 ♕g6 c×d2+ 24 ♘×d2 ♖f7 25 h×g7 ♖×g7 26 ♕×e6+ ♖f7 27 ♕h6 ♖g7 28 ♕h8+ ♔f7 29 e6+ ♔×e6 30 ♕×g7 ♗×g2 31 ♖h6+ ♔d7 320-0-0 ♗d5 33 ♘e4 ♔c8 34 ♖h8 ♗×e4 35 ♖d×8+ ♗×d8 36 ♕e7 ♔b7 37 ♕×e4+ ♔a7 38 ♖g8 ♖b8 39 g5 ♗e7 40 ♖×b8 ♗×g5+ 41 f4 ♗×f4+ 42 ♕×f4 ♔×b8 43 ♔d2 ♔b7 44 ♔d3 ♔c8 45 b4 a×b4 46 ♕×b4 ♔d7 47 ♕b5+ ♔d8 48 ♔e4 1-0

**Duchess – Blitz** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗a4 ♘f6 5 0-0 ♗e7 6 ♕e2 b5 7 ♗b3 0-0 8 c3 d5 9 d3 d×e4 10 d×e4 ♗b7 11 ♖d1 ♕e8 12 a4 ♘a5 13 ♗c2 ♗d6 14 b4 ♘c4 15 ♘a3 ♘×a3 16 ♖×a3 ♕c6 17 ♘g5 h6 18 ♘f3 ♘×e4 19 a×b5 ♕×b5 20 c4 ♕c6 21 c5 ♘×c5 22 ♖c3 e4 23 ♘d2 ♕b5 24 ♕×b5 a×b5 25 b×c5 ♗e5 26 ♖b3 c6 27 ♘×e4 ♖a2 28 ♗b1 ♖aa8 29 f4 ♗b8 30 ♘d6 ♖d8 31 ♖bd3 ♗a6 32 ♗a2 b4 33 ♖d4 ♗a7 34 ♘×f7 ♖×d4 35 ♖×d4 ♗×c5 36 ♘e5+ ♔h8 37 ♗b2 ♗×d4+ 38 ♗×d4 ♗b7 39 ♗e6 ♔h7 40 ♗c5 ♖a4 41 ♘d3 ♔g6 42 ♗×b4 ♗a6 43 f5+ ♔h7 44 ♘c5 ♖×b4 45 ♘×a6 ♖b2 46 ♘c5 ♖d2 47 ♘d7 ♖d4 48 ♘e5 c5 49 ♘d7 ♖d1+ 50 ♔f2 ♖d2+ 51 ♔f3 ♖c2 52 ♘f8+ ♔h8 53 ♘g6+ ♔h7 54 ♘e5 ♖c3+ 55 ♔g4 ♖c2 56 ♔g3 ♖e2 57 ♘d7 ♖e3+ 58 ♔f4 ♖e2 59 ♘f8+ ♔h8 60 ♘g6+ ♔h7 61 ♗d5 ♖f2+ 62 ♔e5 ♖e2+ 63 ♗e4 c4 64 ♘f4 ♖e1 65 ♔d4 ♔h8 66 ♗d5 ♖h1 67 h3 ♖f1 68 g3 c3 69 ♔×c3 ♖g1 70 g4 ♖a1 71 ♔d4 ♔h7 72 ♔e3 ♖a4 73 ♔f3 ♖a3+ 74 ♔g2 ♖c3 75 ♗e6 ♖a3 76 ♘g6 ♖c3 77 ♘f8+ ♔h8 78 ♗f7 ♖c2+ 79 ♔g3 ♖c7 80 ♘g6+ ♔h7 81 ♗d5 ♖c3+ 82 ♔h4 ♖c8 83 ♔h5 ♖b8 84 ♘f4 ♖b5 85 ♗f7 ♖b7 86 ♗a2 ♖b2 87 ♗e6 ♖b1 88 ♔h4 ♖a1 89 ♘g6 ♖a8 90 ♗b3 ♖b8 91 ♗f7 ♖c8 92 ♗d5 ♖c5 93 ♗a2 ♖c8 94 ♘f4 ♖a8 ½-½

**Black Knight – Mike** 1 e4 e5 2 ♘f3 d5 3 e×d5 ♕×d5 4 ♘c3 ♕a5 5 ♗b5+ ♗d7 6 ♘×e5 ♗×b5 7 ♕f3 f6 8 ♕×b7 ♗a6 9 ♕f3 ♕×e5+ 10 ♔d1 ♗e2+ 11 ♘×e2 c6 12 d4 ♕c7 13 ♗f4 ♗d6 14 ♕b3 ♗×f4 15 ♘×f4 ♕×f4 16 ♕b7 ♕×d4+ 17 ♔c1 ♕×f2 18 ♕×g7 h6 19 ♖d1 ♕f4+ 20 ♔b1 ♕×h2 21 ♖e1+ ♔d8 22 g4 ♕d2 23 ♖h1 ♕d5 24 ♖f1 ♕c4 25 ♖d1+ ♔c8 26 a4 ♖h7 27 ♕×h7 ♕×g4 28 ♖e1 ♘d7 29 ♖e8+ ♔b7 30 ♖e4 ♕d1+ 31 ♔a2 ♕d6 32 ♖g1 ♖d8 33 ♖×g8 ♕d5+ 34 b3 ♖×g8 35 ♔b1 ♖g1+ 36 ♔b2 ♕d2 37 ♖e7 ♖d1 38 ♕f5 ♔b6 39 ♖e4 a5 40 ♕f4 ♘e5 41 ♕×d2 ♖×d2 42 ♖h4 ♘f7 43 ♖f4 ♖d6 44 ♖f1 ♘e5 45 c4 c5 46 ♔c2 ♔c6 47 ♖h1 f5 48 ♖f1 ♖f6 49 ♔c3 h5 50 ♖h1 ♖h6 51 ♖f1 ♖f6 52 ♖h1 ♖h6 53 ♖f1 ♖f6 54 ♖h1 ½-½

**Sargon – Brute Force** 1 e4 e5 2 ♘f3 ♘c6 3 ♘c3 ♘f6 4 d4 ♘×d4 5 ♘×d4 e×d4 6 ♕×d4 ♗e7 7 ♗f4 0-0 8 ♘b5 c5 9 ♕a4 a6 10 ♘d6 ♘g4 11 ♗e2 h5 12 ♗×g4 h×g4 13 0-0-0 14 ♕c7 14 ♕c4 g5 15 ♗e5 ♕c6 16 ♕e2 ♗×d6 17 ♖×d6 ♕a4 18 ♕×g4 ♖e8 19 ♕×g5+ ♔f8 20 ♖h6 ♕×c2+ 21 ♔×c2 f6 22 ♕×f6+ ♔g8 23 ♕g7# 1-0

**BS'66'76 — Awit** 1 d4 ♘f6 2♗f4 d6 3 ♘c3 ♘c6 4 ♘f3 g6 5 e4 ♗g7 6 ♗b5 ♘h5 7 ♗e3 ♗d7 8 a4 0-0 9 a5 e6 10 ♕e2 a6 11 ♗d3 b6 12 ♗×a6 b×a5 13 ♗b5 ♘b4 14 ♗c4 c6 15 ♗c1 ♕c7 16 g4 ♘f6 17 g5 ♘h5 18 ♖f1 ♖fb8 19 ♗b3 c5 20 ♕c4 ♖a7 21 d×c5 a4 22 ♘e2 d×c5 23 c3 ♗b5 24 ♗f4 ♘×f4 25 ♕×b5 ♖×b5 26 ♘×f4 ♕×f4 27 ♖×a4 ♖×a4 28 ♔e2 ♕×e4+ 29 ♔d2 ♕d3+ 30 ♔e1 ♗×c3+ 31 b×c3 ♘c2+ 32 ♗×c2 ♖e4# 0-1

**Ostrich — Chaos** 1 e4 c5 2 ♘f3 ♘c6 3 ♗b5 ♘f6 4 e5 ♘d5 5 0-0 ♘c7 6 ♘c3 ♘×b5 7 ♘×b5 h6 8 c3 d5 9 e×d6 e×d6 10 ♕e2+ ♗e6 11 d4 ♗e7 12 ♗e3 0-0 13 ♖fd1 c×d4 14 c×d4 ♘b4 15 a3 ♘d5 16 ♘c3 ♕b6 17 ♔h1 ♕b3 18 ♘×d5 ♕×d5 19 h3 ♖fe8 20 b4 ♖ec8 21 ♖dc1 ♕e4 22 ♖×c8 ♖×c8 23 ♘d2 ♕c6 24 b5 ♕c2 25 ♖b1 d5 26 ♖b3 ♗×a3 27 b6 a5 28 ♔g1 ♗b4 29 ♕d3 ♕×d3 30 ♖×d3 ♖c6 31 ♘b3 ♗f5 32 ♖d2 ♗×d2 33 ♗×d2 a4 34 ♘c5 a3 35 g4 ♗c2 36 ♗c1 a2 37 ♗b2 ♖×b6 38 ♗a1 ♖b1+ 0-1

*Round 3, December 4, 1978*

**Chess 4.7 — Duchess** 1 e4 e5 2 ♘f3 ♘f6 3 d4 e×d4 4 e5 ♘e4 5 ♕×d4 d5 6 e×d6 ♘×d6 7 ♗d3 ♘c6 8 ♕f4 g6 9 0-0 ♗g7 10 ♗d2 ♕f6 11 ♕×f6 ♗×f6 12 ♘c3 ♗e6 13 ♘g5 ♗c4 14 ♘ge4 ♘×e4 15 ♘×e4 ♗×b2 16 ♗×c4 ♗×a1 17 ♖×a1 ♖d8 18 ♖e1 0-0 19 ♗h6 b6 20 ♘f6+ ♔h8 21 ♗×f8 ♖×f8 22 ♗b3 ♘d4 23 ♘d7 ♖d8 24 ♘e5 ♖f8 25 ♘×f7+ ♔g7 26 ♘g5 ♔h6 27 ♘e6 ♖e8 28 ♔f1 ♘×e6 29 ♖×e6 ♖d8 30 ♖e7 c6 31 ♖×a7 ♖d1+ 32 ♔e2 ♖d8 33 ♖c7 c5 34 f4 ♖d4 35 ♔f3 g5 36 g3 g×f4 37 g×f4 ♔g6 38 ♖c6+ ♔f5 39 ♗e6+ ♔f6 40 ♖×b6 ♔e7 41 ♗f5 ♖a4 42 ♗×h7 ♖a3+ 43 ♔g4 ♖×a2 44 f5 ♔d7 45 f6 ♖a4+ 46 ♔g5 ♖a1 47 f7 ♖a8 48 ♔f6 c4 49 ♗f5+ ♔c7 50 ♖c6+ ♔b7 51 ♔e7 ♖h8 52 ♖×c4 ♖×h2 53 f8(♕) ♖e2+ 54 ♔f7 ♖e5 55 ♕c8+ ♔b6 56 ♕c7+ ♔a6 57 ♖a4+ ♔b5 58 ♗d7# 1-0

**Awit — Blitz** 1 b3 ♘f6 2 ♗b2 ♘c6 3 e3 e6 4 ♘f3 ♗c5 5 ♗b5 0-0 6 a3 a6 7 ♗d3 d5 8 0-0 h6 9 b4 ♗b6 10 c4 ♗a7 11 ♕c2 ♗d7 12 ♘c3 a5 13 b5 ♘e7 14 ♖fb1 c5 15 ♘a4 d×c4 16 ♕×c4 b6 17 ♗×f6 g×f6 18 ♕g4+ ♔h8 19 ♕h3 ♔g7 20 ♗e4 ♖c8 21 ♕h4 ♖g8 22 ♕g3+ ♘g6 23 ♘c3 ♕e7 24 a4 ♖cd8 25 d4 ♗b8 26 ♕h3 c4 27 ♕h5 ♔h7 28 g3 f5 29 ♗c2 ♖c8 30 ♘d2 ♕g5 31 ♗d1 ♕×h5 32 ♗×h5 ♖c7 33 ♖c1 ♖gc8 34 ♖ab1 ♔g7 35 f3 ♔h7 36 e4 f×e4 37 ♘c×e4 c3 38 ♘f6+ ♔g7 39 ♘de4 ♘f8 40 ♘×d7 ♖×d7 41 ♖×c3 ♖×c3 42 ♘c3 ♖×d4 43 ♖c1 ♘d7 44 ♖d1 ♖×d1+ 45 ♘×d1 ♘c5 46 ♘c3 ♗e5 47 ♘e2 ♘×a4 48 ♘c1 ♘c3 49 f4 ♗d4+ 50 ♔g2 a4 51 ♔f3 a3 52 ♔g2 a2 53 ♘b3 ♘×b5 54 ♗e2 ♘d6 55 h4 ♘e4 56 ♗f3 ♘d2 57 ♘×d2 a1(♕) 58 ♔h3 ♕c3 59 ♔g4 ♕×d2 60 ♔h3 b5 61 ♗c6 f5 62 ♗×b5 ♗g1 63 ♗c4 ♕h2# 0-1

**Brute Force — BS'66'76** 1 e4 d5 2 e×d5 e5 3 ♘c3 f5 4 ♘f3 e4 5 ♘e5 ♗b4 6 ♗c4 ♘e7 7 ♕h5+ g6 8 ♕d1 ♘d7 9 ♘×d7 ♗×d7 10 0-0 c6 11 d×c6 ♗×c6 12 ♗e6 ♕d6 13 ♗c4 ♕d4 14 ♗e6 ♕e5 15 ♗c4 0-0-0 16 ♖e1 h5 17 ♖e3 b5 18 ♗f7 h4 19 ♕e2 ♖h7 20 ♗b3 ♗c5 21 ♖h3 ♖dh8 22 d4 ♗×d4 23 ♔f1 ♔b8 24 ♗g5 b4 25 f4 e×f3 0-1

**Black Knight — Ostrich** 1 e4 c6 2 d4 d5 3 e×d5 ♕×d5 4 ♘c3 ♕a5 5 ♘f3 e6 6 a3 ♗d6 7 ♗d2 ♕d8 8 ♘e4 ♗e7 9 ♗d3 f5 10 ♘c3 ♘f6 11 0-0 0-0 12 ♖e1 c5 13 ♗e3 c×d4 14 ♗×d4 ♘c6 15 ♗b5 ♕c7 16 ♗c4 ♖e8 17 ♖×e6 ♗×e6 18 ♗×e6+ ♔h8 19 ♘g5 ♘×d4 20 ♕×d4 ♗c5 21 ♕h4 ♕b6 22 ♗a4 ♕c7 23 ♕c4 ♗×f2+ 24 ♔f1 ♕×c4+ 25 ♗×c4 ♖ec8 26 ♗b3 ♗h4 27 ♘f3 b5 28 ♘c3 b4 29 a×b4 ♖cb8 30 ♘×h4 ♖×b4 31 ♖a4 ♖×a4 32 ♗×a4 ♖b8 33 ♗b3 ♘e4 34 ♘×e4 f×e4 35 ♘f5 ♖f8 36 ♗e6 g6 37 g4 g×f5 38 ♔e1 f×g4 39 ♗×g4 e3 40 ♔d1 ♖f1+ 41 ♔e2 ♖f2+ 42 ♔d3 ♖d2+ 43 ♔c3 ♖×h2 44 ♗f5 ♖f2 45 ♗e4 ♖f4 46 ♗d3 ♖f2 47 b4 ♖h2 48 b5 ♔g7 49 ♗e4 ♖d2 50 ♗f3 h6 51 ♗g4 ♔g6 52 ♗f3 ♖f2 53 ♗b7 ♖d2 54 ♗e4+ ♔f6 55 ♗f3 ♖f2 56 ♗c6 ♖d2 57 ♗f3 ♖f2 58 ♗e4 ♖f4 59 ♗d5 e2 60 ♔d2 ♖d4+ 61 ♔×e2 ♖×d5 62 c4 ♖d4 63 c5 ♖b4 64 ♔d3 h5 65 c6 ♔e6 66 ♔c3 ♖b1 67 ♔d4 ♔d6 68 c7 ♔×c7 69 ♔c5 h4 70 ♔c4 h3 71 ♔c5 h2 72 ♔c4 h1(♕) 73 ♔d3 ♔d6 74 ♔e3 ♖×b5 75 ♔d4 ♖b3 76 ♔c4 ♕d5# 0-1

**Mike — Sargon** 1 e4 e5 2 d4 ♘c6 3 d5 ♘b4 4 c3 ♘a6 5 f4 ♘c5 6 f×e5 ♕h4+ 7 ♔d2 ♘×e4+ 8 ♔c2 ♘f2 9 ♕e1 ♗c5 10 ♘f3 ♕h5 11 ♖g1 ♘e7 12 ♗e3 ♕g6+ 13 ♔c1 ♗×e3+ 14 ♕×e3 ♘g4 15 ♕d2 d6 16 ♗d3 ♕h5 17 ♗b5+ ♗d7 18 ♗×d7+ ♔×d7 19 e×d6 c×d6 20 h3 ♘f6 21 ♕g5 ♕×g5+ 22 ♘×g5 ♖hf8 23 c4 ♖ac8 24 ♘d2 ♘e×d5 25 ♔b1 ♘e3 26 ♘ge4 ♘×e4 27 ♘×e4 ♖×c4 28 ♘c3 f5 29 a4 d5 30 g3 ♖f6 31 ♖e1 d4 32 ♘b5 a6 33 ♘×d4 f4 34 g×f4 ♖×d4 35 ♖×e3 ♖f×f4 36 a5 ♖d1+ 37 ♔a2 ♖a4+ 38 ♖a3 ♖×a3+ 39 b×a3 ♖d2+ 40 ♔b3 ♖d3+ 41 ♔c4 ♖×h3 42 ♖d1+ ♔c6 43 a4 ♖h5 44 ♔b4 ♖h4+ 45 ♔b3 ♖h3+ 46 ♔c4 ♖h5 47 ♔b4 ♖h4+ 48 ♔b3 ♖h3+ 49 ♔c4 ♖h5 50 ♔b4 ♖h4+ ½-½

**Chaos — Belle** 1 d4 d5 2 c4 d×c4 3 ♘f3 ♘f6 4 e3 e6 5 ♗×c4 c5 6 ♕e2 a6 7 0-0 b5 8 ♗b3 ♘c6 9 ♘c3 ♗e7 10 ♖d1 c4 11 ♗c2 ♘b4 12 e4 ♘×c2 13 ♕×c2 ♗b7 14 ♗f4 b4 15 e5 ♘h5 16 ♘e2 ♘×f4 17 ♘×f4 g5 18 ♘h5 ♗×f3 19 g×f3 ♕d5 20 ♕e4 c3 21 ♕×d5 e×d5 22 ♖db1 c2 23 ♖e1 ♖c8 24 ♖ac1 ♖c4 25 ♖e2 ♖×d4 26 ♘f6+ ♗×f6 27 e×f6+ ♔d7 28 ♖e×c2 ♖f8 29 ♖c7+ ♔e6 30 ♖1c6+ ♔f5 31 ♖×a6 ♖d3 32 ♖b7 ♖×f3 33 ♖×b4 ♔e5 34 ♖g4 ♖g8 35 ♖g3 g4 36 ♔g2 ♖×f6 37 ♖a7 d4 38 ♔f1 ♖h6 39 ♖e7+ ♔f6 40 ♖e4 ♖×h2 41 ♖×d4 h5 42 ♖f4+ ♔g6 43 ♖c3 ♖h1+ 44 ♔g2 ♖b1 45 ♖b4 ♖a8 46 ♖b6+ ♔g7 47 a3 ♖a7 48 ♖cb3 ♖a4 49 ♖3b5 h4 50 ♖b7 ♖d4 51 ♖c7 ♔g6 52 ♖bb7 ♖dd1 53 ♖c8 ♖g1+ 54 ♔h2 ♖h1+ 55 ♔g2 ♖bg1# 0-1

*Round 4, December 5, 1978*

**Blitz — Belle** 1 e4 e5 2 ♘f3 ♘c6 3 ♘c3 ♘f6 4 ♗b5 ♘d4 5 ♗c4 ♗c5 6 ♘×e5 ♕e7 7 ♗×f7+ ♔f8 8 ♘g6+ h×g6 9 ♗c4 ♘×e4 10 0-0 ♖×h2 11 ♔×h2 ♕h4+ 12 ♔g1 ♘g3 13 ♕h5 g×h5 14 f×g3+ ♘f3# 0-1

**Ostrich — Chess 4.7** 1 e4 e5 2 ♘f3 ♘f6 3 ♘×e5 d6 4 ♘f3 ♘×e4 5 ♕e2 ♕e7 6 d3 ♘f6 7 ♗g5 ♘bd7 8 ♘c3 ♕×e2+ 9 ♔×e2 h6 10 ♗e3 b6 11 ♘b5 ♘d5 12 c4 a6 13 ♘×d6+ ♗×d6 14 c×d5 ♗b7 15 ♘d4 ♗×d5 16 ♔e1 ♗e5 17 f4 ♗f6 18 ♔d2 c5 19 ♘c2

♗×b2 20 ♖b1 ♗f6 21 a3 0-0 22 ♔d1 ♗a2 23 ♗×c5 ♖fc8 24 ♖×b6 ♘×b6 25 d4 ♗b3 26 ♔e1 ♘a4 27 ♘b4 ♘×c5 28 d×c5 ♖×c5 29 ♖g1 ♖c1+ 30 ♔d2 ♖d8+ 31 ♔e2 ♗d4 32 ♖h1 ♖e8+ 33 ♔f3 ♗d1+ 34 ♔g3 ♖e3+ 35 ♔f2 ♖d3+ 36 ♔e1 ♗g4# 0-1

**Duchess – Chaos** 1 e4 c5 2 d4 c×d4 3 c3 ♘f6 4 e5 ♘d5 5 ♕×d4 e6 6 ♗c4 d6 7 ♘f3 ♘c6 8 ♗b5 ♗d7 9 ♗×c6 ♗×c6 10 0-0 f6 11 e×f6 e×f6 12 ♖e1 ♘c7 13 ♕d3 ♕d7 14 ♘d4 ♖b8 15 ♘×c6 b×c6 16 ♕d4 ♘d5 17 c4 ♘c7 18 ♕×a7 ♕c8 19 b3 e5 20 ♘c3 ♗e7 21 ♗h6 ♖g8 22 ♕e3 ♖g6 23 ♖ad1 ♕g4 24 g3 c5 25 ♕d2 f5 26 ♗e3 f4 27 ♗×c5 ♕h3 28 ♘d5 ♘×d5 29 ♕×d5 ♕c8 30 ♗a7 ♖a8 31 ♗b6 f×g3 32 f×g3 ♖×a2 33 ♖f1 ♕h3 34 ♕c6+ ♕d7 35 ♕e4 ♕c8 36 ♖d5 ♕h3 37 ♗f2 ♖h6 38 ♕g2 ♕d7 39 ♔h1 ♖f6 40 ♖b5 ♕c8 41 g4 ♖f7 42 ♖b6 ♗h4 43 ♖×d6 ♖f×f2 44 ♖×f2 ♖×f2 45 ♕e4 ♗e7 46 ♖d3 ♖f1+ 47 ♔g2 ♖a1 48 ♔h3 ♖g1 49 ♖g3 ♖×g3+ 50 h×g3 ♕b8 51 ♕×h7 ♕×b3 52 ♕h5+ ♔d8 53 ♕×e5 ♕×c4 54 ♕h5 ♗f6 55 ♕g6 ♕f1+ 56 ♔h2 ♕f3 57 ♕f7 ½-½

**Brute Force – Black Knight** 1 e4 c5 2 ♘f3 ♘c6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 d6 6 ♗c4 ♗d7 7 ♘×c6 ♗×c6 8 ♕d4 e5 9 ♕e3 ♕b6 10 ♕×b6 a×b6 11 ♘d5 ♗×d5 12 ♗×d5 ♘×d5 13 e×d5 ♖a5 14 ♗e3 ♖×d5 15 c4 ♖a5 16 ♗×b6 ♖a6 17 ♗e3 ♖a4 18 b3 ♖a8 19 0-0 ♗e7 20 ♖fd1 0-0 21 ♖d5 ♖a3 22 ♖b5 ♖b8 23 ♔f1 h6 24 ♔e2 ♖ba8 25 ♖×b7 ♗h4 26 ♖d1 ♖×a2+ 27 ♔f3 ♖e8 28 ♖×d6 e4+ 29 ♔g4 ♗×f2 30 ♗f4 ♗e1 31 ♔h3 g5 32 ♗e3 ♖e2 33 ♗d4 h5 34 g4 h×g4+ 35 ♔×g4 ♗f2 36 ♔×g5 ♗×d4 37 ♖×d4 ♖g2+ 38 ♔f4 ♖×h2 39 ♖bd7 e3 40 ♔f5 e2 41 ♖g4+ ♔h7 42 ♖×f7+ ♔h8 43 ♖e4 ♖×e4 44 ♖f8+ ♔g7 45 ♖f6 ♖h5+ 46 ♔×e4 e1(♕)+ 47 ♔d4 ♕d2+ 48 ♔e4 ♔×f6 49 b4 ♖f5 50 b5 ♖f4# 0-1

**Sargon – Awit** 1 e4 c5 2 d4 c×d4 3 ♕×d4 ♘c6 4 ♕e3 ♘f6 5 ♘f3 e6 6 ♘c3 ♗b4 7 ♗b5 a6 8 ♗×c6 d×c6 9 ♗d2 ♘g4 10 ♕f4 ♘f6 11 0-0 0-0 12 ♖ad1 b5 13 ♗e3 ♕a5 14 ♗d4 ♗×c3 15 ♗×c3 ♕×a2 16 ♗×f6 g×f6 17 ♕×f6 ♕c4 18 ♕g5+ ♔h8 19 ♕f6+ ♔g8 20 ♕g5+ ♔h8 21 ♖d8 ♖×d8 22 ♕f6+ ♔g8 23 ♕g5+ ♔f8 24 ♕×d8+ ♔g7 25 ♕g5+ ♔f8 26 ♕d8+ ♔g7 27 ♕d4+ ♕×d4 28 ♘×d4 ♗b7 29 ♖e1 ♔g6 30 ♖e3 ♖d8 31 ♖d3 c5 32 ♘×e6 ♖×d3 33 ♘f4+ ♔g5 34 ♘×d3 ♗×e4 35 ♘×c5 ♗×c2 36 ♘×a6 ♗b3 37 ♘c5 ♗d5 38 g3 ♗f3 39 ♘b3 b4 40 ♘d4 ♗e4 41 f3 ♗b7 42 ♔f2 h6 43 ♔e3 ♗d5 44 ♘c2 b3 45 ♘d4 ♔g6 46 ♔d3 ♔h7 47 f4 ♔g8 48 ♔c3 ♔g7 49 ♘×b3 ♗×b3 50 ♔×b3 ♔f6 51 ♔c4 ♔e7 52 b4 ♔d7 53 ♔d5 h5 54 b5 ♔c7 55 ♔c5 ♔b7 56 b6 ♔b8 57 ♔c6 ♔c8 58 b7+ ♔b8 59 ♔b6 h4 60 g×h4 f6 61 h5 f5 62 ♔c6 ♔a7 63 ♔c7 ♔a6 64 b8(♕) ♔a5 65 ♕b3 ♔a6 66 ♕a4# 1-0

**BS'66'76 – Mike** 1 d4 d5 2 c4 d×c4 3 ♘f3 ♘f6 4 e3 ♗e6 5 ♘a3 c5 6 ♘×c4 ♘c6 7 ♕a4 c×d4 8 ♘ce5 ♗d7 9 ♘×d7 ♘×d7 10 ♘×d4 ♘×d4 11 ♕×d4 e5 12 ♕a4 ♕b6 13 ♗d2 ♕×b2 14 ♖d1 0-0-0 15 ♗a5 ½-½

**ACM 1979 – Detroit**

| | rate | perf | 1 | 2 | 3 | 4 | tot |
|---|---|---|---|---|---|---|---|
| 1 Chess 4.9 | 2040 | 2099 | 8+ | 9+ | 3+ | 2= | 3½ |
| 2 Belle | 1950 | 1982 | 5+ | 4= | 7+ | 1= | 3 |
| 3 Duchess | 1889 | 1942 | 10+ | 7+ | 1– | 4+ | 3 |
| 4 Chaos | 1775 | 1794 | 12+ | 2= | 9+ | 3– | 2½ |
| 5 L'excentrique | 0 | 1640 | 2– | 12+ | 8+ | 6= | 2½ |
| 6 Mychess | 0 | 1552 | 7– | 10+ | 11+ | 5= | 2½ |
| 7 Sargon 3 | 0 | 1614 | 6+ | 3– | 2– | 9= | 1½ |
| 8 Ostrich 80 | 1450 | 1374 | 1– | 11+ | 5– | 10= | 1½ |
| 9 Blitz 6.9 | 0 | 1516 | 11+ | 1– | 4– | 7= | 1½ |
| 10 Awit | 1325 | 1314 | 3– | 6– | 12+ | 8= | 1½ |
| 11 BS'66'76 | 0 | 1045 | 9– | 8– | 6– | 12+ | 1 |
| 12 Rufus | 0 | 644 | 4– | 5– | 10– | 11– | 0 |

*Round 1, October 28, 1979*

**L'Excentrique – Belle** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗a4 ♘f6 5 0-0 ♘×e4 6 ♕e2 ♘c5 7 ♗×c6 d×c6 8 d4 ♘e6 9 d×e5 ♘d4 10 ♘×d4 ♕×d4 11 h3 ♗e7 12 ♖d1 ♕b6 13 b3 ♗f5 14 g4 ♗e6 15 a4 0-0 16 a5 ♕b5 17 ♘c3 ♕×e2 18 ♘×e2 ♖fd8 19 ♗e3 ♗d5 20 c4 ♗f3 21 ♖×d8+ ♖×d8 22 ♘d4 ♗e4 23 f3 c5 24 f×e4 c×d4 25 ♗d2 d3 26 ♔g2 f6 27 e×f6 ♗×f6 28 ♖f1 c6 29 g5 ♗e5 30 c5 g6 31 h4 ♗d4 32 b4 ♖d7 33 ♖f3 ♗e5 34 ♖f1 ♖d8 35 ♖d1 ♔f7 36 ♖h1 ♔e6 37 h5 ♗g7 38 h6 ♗h8 39 ♖f1 ♔e7 40 ♔g3 ♗d4 41 ♔h3 ♗e5 42 ♔g4 ♖d4 43 ♔f3 ♔e6 44 ♖h1 ♗c7 45 ♖a1 ♗d8 46 ♖f1 ♖d7 47 ♔g3 ♗c7+ 48 ♔g2 ♔e5 49 ♖f4 ♔d4 50 ♖f3 ♔×e4 51 ♖f1 ♔d4 52 ♖e1 ♖f7 53 ♖e3 ♗e5 54 ♖e1 ♖f5 55 ♗e3+ ♔c3 56 b5 d2 57 ♖d1 ♔d3 58 ♗×d2 ♔e2 59 b×a6 ♔×d1 60 a7 ♖f8 61 ♗b4 ♗f4 62 ♗c3 ♗×g5 63 ♗e5 ♖a8 64 ♔f3 ♖×a7 65 ♗c7 ♗×h6 0-1

**Chess 4.9 – Ostrich 80** 1 e4 c6 2 c4 d5 3 e×d5 c×d5 4 d4 e6 5 c×d5 ♕×d5 6 ♘c3 ♕d8 7 ♘f3 ♘f6 8 ♗f4 ♘c6 9 ♗b5 ♗d7 10 0-0 a6 11 ♗d3 ♕b6 12 ♕b3 ♕×b3 13 a×b3 ♗e7 14 ♘e4 0-0 15 ♘×f6+ ♗×f6 16 ♗e4 h6 17 ♘e5 ♗×e5 18 d×e5 g5 19 ♗g3 b6 20 ♖fd1 ♖a7 21 ♖×d7 ♖×d7 22 ♗×c6 ♖d2 23 h4 ♖×b2 24 ♖×a6 ♖×b3 25 ♖a7 ♖c3 26 ♗f3 g×h4 27 ♗×h4 b5 28 ♖b7 ♖b3 29 ♔h2 ♖c8 30 ♗h5 ♖c6 31 ♖b8+ ♔h7 32 ♗f6 ♖h3+ 33 g×h3 ♖c8 34 ♖×c8 b4 35 ♖h8# 1-0

**Awit – Duchess** 1 c4 ♘f6 2 d4 e6 3 ♘f3 d5 4 ♗g5 h6 5 ♗d2 d×c4 6 ♘c3 c6 7 e3 b5 8 ♘e5 ♗d6 9 ♖c1 0-0 10 ♗e2 ♗×e5 11 d×e5 ♘fd7 12 f4 ♘c5 13 0-0 ♘d3 14 ♖c2 ♘a6 15 a4 b4 16 ♘e4 b3 17 ♖×c4 ♘×b2 18 ♕×b3 ♖b8 19 ♕c2 ♘×c4 20 ♗×c4 ♕b6 21 a5 ♕c7 22 ♖b1 ♖×b1+ 23 ♕×b1 ♖d8 24 ♘d6 ♘b8 25 ♗c3 ♗a6 26 g4 ♗×c4 27 ♘×c4 c5 28 ♔f2 ♕d7 29 h3 ♘c6 30 h4 ♕d3 31 ♕×d3 ♖×d3 32 ♗d2 ♔f8 33 ♗e1 ♔e7 34 ♔e2 ♖b3 35 g5 h×g5 36 h×g5 ♔d7 37 ♔d2 ♖b1 38 e4 ♖a1 39 ♔e2 ♘d4+ 40 ♔f2 ♘f3 41 ♔×f3 ♖×e1 42 f5 g6 43 f×g6 f×g6 44 ♔f4 ♖a1 45 ♔g3 ♖a4 46 ♘d2 c4 47 ♘×c4 ♖×c4 48 ♔f4 a6 49 ♔e3 ♖c5 50 ♔f3 ♖×e5 0-1

**Chaos – Rufus** 1 d4 f5 2 c4 e6 3 ♘f3 ♘f6 4 e3 d5 5 ♗d3 ♘c6 6 ♘c3 ♗b4 7 0-0 0-0 8 ♗d2 ♗d7 9 ♕b3

d×c4 10 ♗×c4 ♕e7 11 a3 ♗×c3 12 ♗×c3 b6 13 ♔h1 ♘e4 14 ♗b4 ♘×b4 15 a×b4 ♕d6 16 ♗d3 ♗c6 17 ♗×e4 ♗×e4 18 ♘e5 ♖fe8 19 ♕c3 a6 20 b3 b5 21 f3 ♗d5 22 ♘d3 ♖ab8 23 ♕c2 g6 24 ♘c5 ♖b6 25 e4 f×e4 26 f×e4 ♗b7 27 ♕f2 ♖c8 28 ♕f7+ ♔h8 29 ♘d7 ♗×e4 30 ♘f6 ♗×g2+ 31 ♔×g2 ♕d5+ 32 ♘×d5 e×d5 33 ♖ae1 ♖c6 34 ♖e8+ ♖×e8 35 ♕×e8+ ♔g7 36 ♕f8# 1-0

**BS'66'76 – Blitz 6.9** 0-1

**Sargon 3 – Mychess** 1 d4 d5 2 ♘f3 ♘f6 3 e3 e6 4 ♗d3 ♘c6 5 0-0 ♗d6 6 ♘c3 a6 7 ♗d2 b6 8 a3 ♗b7 9 ♖e1 a5 10 e4 d×e4 11 ♘×e4 ♗e7 12 c3 ♘×e4 13 ♗×e4 0-0 14 ♕b1 g6 15 ♘e5 ♘×e5 16 ♗×b7 ♖b8 17 ♗h6 ♖e8 18 ♗e4 ♘g4 19 ♗f4 ♗d6 20 ♗g3 c5 21 d×c5 ♗×g3 22 h×g3 ♕d2 23 ♕c2 ♕×c2 24 ♗×c2 b×c5 25 b3 e5 26 f3 ♘f6 27 ♖ad1 ♖b6 28 ♔f2 h5 29 ♔e3 g5 30 ♔d3 ♖d8+ 31 ♔c4 ♖×d1 32 ♖×d1 ♖b8 33 ♖d6 ♔g7 34 ♖a6 ♖d8 35 ♖×a5 ♘d5 36 ♗e4 ♘e3+ 37 ♔×c5 ♖b8 38 b4 ♖c8+ 39 ♗c6 ♔f6 40 ♖a7 ♘×g2 41 ♖d7 ♘e3 42 b5 ♘c2 43 a4 h4 44 g×h4 g×h4 45 ♔d5 ♖b8 46 ♖d6+ ♔f5 47 ♗d7+ ♔g5 48 ♔×e5 ♖a8 49 f4+ ♔h5 50 ♗c6 ♖g8 51 ♗f3+ ♖g4 52 ♖d8 ♔h6 53 ♗×g4 ♔g7 54 ♖d7 ♘e3 55 ♗e6 ♔f8 56 ♖×f7+ ♔g8 57 ♔f6 ♔h8 58 ♔g6 ♔g8 59 ♖f6+ ♔h8 60 ♖f8# 1-0

*Round 2, October 28, 1979*

**Belle – Chaos** 1 d4 ♘f6 2 c4 e6 3 ♘c3 ♗b4 4 e3 c5 5 ♗d3 0-0 6 a3 ♗×c3+ 7 b×c3 ♘c6 8 ♘e2 d6 9 e4 e5 10 d5 ♘a5 11 0-0 b6 12 f4 ♕e8 13 f×e5 d×e5 14 ♗g5 ♘h5 15 ♕c2 ♗a6 16 ♕a2 ♕a4 17 g4 ♘f4 18 ♘×f4 e×f4 19 ♗×f4 ♗×c4 20 ♗×c4 ♘×c4 21 ♔g2 ♘e5 22 ♕e2 ♖fe8 23 ♖fe1 ♘c4 24 ♕a2 ♘a5 25 ♔f3 f6 26 d6+ ♔h8 27 ♕d5 ♕d7 28 ♕f5 ♖ad8 29 ♕×d7 ♖×d7 30 e5 f×e5 31 ♖×e5 ♖×e5 32 ♗×e5 ♘c4 33 ♗f4 b5 34 a4 g5 35 ♗×g5 b×a4 36 ♖×a4 ♘×d6 37 ♖a5 ♖f7+ 38 ♗f4 c4 39 ♖c5 ♖f8 40 ♖c6 ♘f7 41 ♔g3 ♔g7 42 ♖×c4 a6 43 ♖c7 ♖a8 44 c4 a5 45 c5 a4 46 ♗e5+ ♔f8 47 ♗a1 a3 48 c6 ♖a6 49 ♗f6 ♖a8 50 ♖b7 a2 51 c7 ♖c8 52 ♖b8 ♘d6 53 ♔f4 ♔e8 54 ♖×c8+ ♘×c8 55 ♔g5 ♔d7 56 ♔h6 ♔×c7 57 ♔×h7 ♘d6 58 ♗e5 ♔d7 59 g5 ♘f7 60 ♗b2 ♘×g5+ 61 ♔g6 ♘f3 62 ♔f5 ♘×h2 63 ♔e4 ♔c6 64 ♔d4 ♔b5 65 ♔c3 ♔a4 66 ♔c2 ♘f3 67 ♗f6 ♔a3 68 ♗e7+ ♔a4 69 ♔b2 a1(♕)+ 70 ♔×a1 ½-½

**Blitz 6.9 – Chess 4.9** 1 e4 ♘c6 2 d4 d5 3 e5 f6 4 f4 ♗f5 5 ♘e2 e6 6 ♘g3 f×e5 7 f×e5 ♕h4 8 ♗e3 ♘h6 9 ♘c3 ♘g4 10 ♗g1 ♗e7 11 ♘b5 ♖c8 12 c4 d×c4 13 ♗×c4 a6 14 ♘c3 ♘b4 15 ♔d2 ♕g5+ 16 ♔e2 ♘×e5 17 ♗×e6 ♗×e6 18 ♕b1 ♗g4+ 19 ♔f1 0-0+ 20 ♗f2 ♖×f2+ 21 ♔g1 ♕e3 22 ♘d5 ♘f3+ 23 g×f3 ♖f1+ 24 ♔×f1 ♗h3# 0-1

**Duchess – Sargon 3** 1 e4 g6 2 d4 ♗g7 3 ♘f3 d6 4 ♘c3 ♘f6 5 ♗e2 0-0 6 0-0 ♘bd7 7 e5 d×e5 8 ♘×e5 ♘×e5 9 d×e5 ♘d7 10 f4 g5 11 f5 ♗×e5 12 ♗×g5 ♘f6 13 ♔h1 ♕d7 14 ♕e1 ♖d8 15 ♖d1 ♕e8 16 ♗d3 ♘g4 17 h3 f6 18 h×g4 ♗×c3 19 ♕×c3 f×g5 20 ♗c4+ e6 21 f×e6 ♗×e6 22 ♖de1 ♖d6 23 ♗×e6+ ♕×e6 24 ♖×e6 ♖×e6 25 ♕b3 ♖e8 26 ♖f6 c5 27 ♖×e6 ♖×e6 28 ♕×e6+ ♔g7 29 ♕e7+ ♔g6 30 ♕×b7 a5 31 ♕b6+ ♔f7 32 ♕×c5 h6 33 ♕×a5 ♔e6 34 ♕a6+ ♔e5 35 ♕×h6 ♔f4 36 ♕e6 ♔g3 37 ♕e3+ ♔×g4 38 ♕f3+ ♔h4 39 ♕h3# 1-0

**Rufus – L'Excentrique** 1 c4 f5 2 ♘c3 ♘f6 3 ♘f3 a5 4 e3 e6 5 ♗e2 ♗e7 6 0-0 0-0 7 d4 ♘e4 8 ♗d2 d5 9 c×d5 e×d5 10 ♕b3 ♘×c3 11 ♕×c3 ♗b4 12 ♕c2 ♕e7 13 ♘e5 ♗×d2 14 ♕×d2 a4 15 ♖ac1 g6 16 ♖fe1 ♔g7 17 ♗f3 c6 18 ♕d3 ♘d7 19 ♘×d7 ♗×d7 20 ♖c5 b6 21 ♖cc1 h5 22 g3 h4 23 b3 a3 24 h3 ♖h8 25 ♗g2 g5 26 g4 f×g4 27 h×g4 h3 28 ♗f3 ♕f6 29 ♕e2 ♖af8 30 ♗h1 h2+ 31 ♔f1 ♗×g4 32 ♕c2 ♗h3+ 33 ♔e2 ♕×f2+ 34 ♔d3 ♗f5+ 35 e4 ♖h3+ 36 ♗f3 ♖×f3+ 37 ♖e3 ♕×e3# 0-1

**Ostrich 80 – BS'66'76** 1 e4 c5 2 ♘f3 ♘c6 3 ♗b5 g6 4 0-0 ♗g7 5 ♘c3 a6 6 ♗×c6 d×c6 7 h3 b6 8 e5 ♗b7 9 d3 1-0

**Mychess – Awit** 1 e4 c5 2 ♘f3 d6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 a6 6 ♗e2 e5 7 ♘b3 ♗e7 8 ♗e3 0-0 9 a4 a5 10 f4 ♘c6 11 f5 ♘b4 12 0-0 d5 13 e×d5 ♘f×d5 14 ♘×d5 ♘×d5 15 ♗c5 ♗×c5+ 16 ♘×c5 ♘e3 17 ♕×d8 ♖×d8 18 ♖f2 b6 19 ♗f3 ♖a7 20 ♘d3 ♗×f5 21 ♘×e5 ♖e8 22 ♘c6 ♖d7 23 c3 ♗e4 24 ♗×e4 ♖×e4 25 ♘d4 g6 26 ♖e1 ♔g7 27 ♖fe2 ♘×g2 28 ♖×e4 ♘×e1 29 ♖×e1 ♖d6 30 ♔f2 f5 31 ♖e6 ♖×e6 32 ♘×e6+ ♔h6 33 ♘f4 ♔g5 34 ♘d5 ♔h4 35 ♘×b6 ♔h3 36 ♔g1 f4 37 b3 f3 38 ♘c4 f2+ 39 ♔×f2 ♔×h2 40 ♘×a5 g5 41 ♘c6 g4 42 a5 g3+ 43 ♔e2 g2 44 ♘e5 g1(♕) 45 ♘f3+ ♔g2 46 ♘×g1 ♔h2 47 a6 ♔×g1 48 a7 h5 49 a8(♕) h4 50 ♕g8+ ♔h2 51 ♔f2 ♔h1 52 ♕g2# 1-0

*Round 3, October 29, 1979*

**Chess 4.9 – Duchess** 1 e4 d5 2 e×d5 ♕×d5 3 ♘c3 ♕d6 4 d4 ♗f5 5 ♘f3 ♘c6 6 ♘b5 ♕d7 7 ♗f4 ♖c8 8 ♘e5 ♘×e5 9 d×e5 ♕c6 10 ♕f3 ♕×f3 11 g×f3 a6 12 ♘d4 ♗d7 13 0-0-0 g6 14 ♘b3 ♖d8 15 ♘a5 ♗c8 16 ♖×d8+ ♔×d8 17 ♗c4 ♔e8 18 ♖d1 ♗h6 19 ♗d2 ♗g7 20 ♗c3 c6 21 e6 ♗h6+ 22 ♔b1 f6 23 ♘×b7 ♗×b7 24 ♗a5 ♔f8 25 ♖d7 ♗f4 26 h3 ♗e5 27 ♖×b7 g5 28 ♗×a6 h5 29 a4 ♖h6 30 ♗c4 f5 31 ♗d2 ♖g6 32 a5 ♗d4 33 a6 ♗×f2 34 a7 ♗×a7 35 ♖×a7 g4 36 ♗c3 ♘f6 37 ♗b4 ♘g8 38 ♖a8+ ♔g7 39 ♗c3+ ♔h6 40 ♗d2+ ♔g7 41 f×g4 h×g4 42 h×g4 ♖×g4 43 ♗d3 f4 44 ♗c3+ ♔h6 45 ♖f8 c5 46 ♗e5 f3 47 ♖×f3 c4 48 ♖f4 ♖g1+ 49 ♗f1 ♔g5 50 ♖f7 ♔h6 51 ♔a2 ♖h1 52 ♗×c4 ♖e1 53 ♗g7+ ♔g5 54 ♗f8 ♖e4 55 ♖g7+ ♔f6 56 ♗d3 ♖a4+ 57 ♔b3 ♖a7 58 ♖×g8 ♔×e6 59 ♗c4+ ♔e5 60 ♖g7 ♖b7+ 61 ♔c3 1-0

**Sargon 3 – Belle** 1 ♘c3 d5 2 e4 ♘f6 3 e×d5 ♘×d5 4 ♘×d5 ♕×d5 5 ♘e2 ♘c6 6 d3 e5 7 ♘c3 ♗b4 8 ♗d2 ♗×c3 9 b×c3 ♗f5 10 c4 ♕d4 11 ♗e3 ♕c3+ 12 ♗d2 ♕a3 13 g4 ♗d7 14 ♗g2 0-0 15 0-0 f5 16 ♕b1 ♖ab8 17 g×f5 ♗×f5 18 ♕b5 a6 19 ♕d5+ ♖f7 20 ♖fb1 ♖d8 21 ♕f3 ♘d4 22 ♕d1 c6 23 ♗g5 ♖df8 24 ♗e3 ♕c3 25 ♖c1 ♕b2 26 a4 ♗g6 27 ♖ab1 ♕a3 28 ♖a1 ♕c3 29 ♗d2 ♕b2 30 ♖ab1 ♕a2 31 ♖a1 ♕b2 32 ♗e3 ♔h8 33 ♖ab1 ♕a3 34 ♖a1 ♕c3 35 ♗d2 ♕b2 36 ♖ab1 ♕a2 37 ♖a1 ♕b2 38 ♗e3 ♖e7 39 ♖ab1 ♕a3 40 ♖a1 ♕c3 41 ♗d2 ♕b2 42 ♖ab1 ♕a2 43 ♗b4 ♖ff7 44 ♖a1 ♕b2 45 ♖ab1 ♕a2 46

♗×e7 ♖×e7 47 ♖a1 ♕b2 48 ♔f1 ♖f7 49 ♖ab1 ♕a2 50 ♕e1 ♘×c2 51 ♕×e5 ♗×d3+ 52 ♔g1 ♕a3 53 ♕b8+ ♖f8 54 ♕×b7 ♘d4 55 ♖a1 ♕c5 56 ♖d1 ♗×c4 57 ♔h1 ♗d5 58 ♗×d5 ♕×d5+ 59 ♔g1 ♘e2+ 60 ♔f1 ♕f3 61 ♔e1 ♕×f2+ 62 ♔d2 ♖d8+ 63 ♔c2 ♘d4+ 64 ♔d3 ♘b5+ 65 ♕d7 ♖×d7+ 66 ♔c4 ♕c2+ 67 ♔b4 ♕c3# 0-1

**Chaos – Blitz 6.9** 1 d4 d5 2 c4 e6 3 ♘c3 ♘f6 4 ♗g5 ♗e7 5 e3 0-0 6 ♘f3 h6 7 ♗h4 b6 8 c×d5 ♘×d5 9 ♗×e7 ♕×e7 10 ♘×d5 e×d5 11 ♗d3 ♘c6 12 ♕c2 ♘b4 13 ♕c3 ♘×d3+ 14 ♕×d3 ♕b4+ 15 ♕d2 ♕b5 16 ♕c3 ♗a6 17 0-0-0 ♕a4 18 ♔b1 ♗c4 19 b3 ♕b5 20 ♘e5 ♗e2 21 ♖de1 f6 22 a4 ♕a6 23 ♘c6 ♗d3+ 24 ♔b2 ♕b7 25 ♘×a7 ♗h7 26 ♘c6 ♖fe8 27 ♔a1 ♕a6 28 f3 ♖e6 29 ♘b4 ♕a5 30 ♖c1 ♖ae8 31 ♖he1 ♔h8 32 ♖e2 f5 33 h3 f4 34 ♘a2 ♖×e3 35 ♖×e3 ♖×e3 36 ♕×a5 b×a5 37 ♖×c7 ♖×b3 38 ♘c3 ♖b4 39 ♖a7 ♖×d4 40 ♖×a5 ♗g8 41 ♔b2 ♖d2+ 42 ♔c1 ♖×g2 43 ♘×d5 ♖g1+ 44 ♔d2 ♗×d5 45 ♖×d5 ♖g3 46 ♖d3 ♖×h3 47 ♖a3 g5 48 a5 g4 49 a6 ♖h2+ 50 ♔c3 ♖e2 51 a7 ♖e3+ 52 ♔b4 ♖e8 53 a8(♕) ♖×a8 54 ♖×a8+ ♔g7 55 f×g4 f3 56 ♔c4 f2 57 ♖a1 ♔g6 58 ♔d3 ♔g5 59 ♔e2 ♔×g4 60 ♔×f2 h5 61 ♖a4+ ♔f5 62 ♔g3 1-0

**L'Excentrique – Ostrich 80** 1 e4 c6 2 d4 d5 3 ♘c3 d×e4 4 ♘×e4 ♘d7 5 ♗c4 ♘gf6 6 ♘g5 ♘d5 7 ♕h5 g6 8 ♕e2 ♘7f6 9 c3 h6 10 ♘e4 e6 11 ♘×f6+ ♘×f6 12 ♘f3 ♗e7 13 a4 ♕d6 14 ♕e5 ♕×e5+ 15 d×e5 ♘d7 16 h4 ♘b6 17 ♗d3 ♗d7 18 a5 ♘d5 19 ♗d2 0-0-0 20 0-0 c5 21 c4 ♘b4 22 ♗×b4 c×b4 23 ♗e4 ♗c6 24 ♗×c6 b×c6 25 ♖fd1 ♖×d1+ 26 ♖×d1 g5 27 h×g5 h×g5 28 a6 g4 29 ♘h2 ♖g8 30 b3 g3 31 f×g3 ♖×g3 32 ♘f3 ♖g4 33 ♔h2 ♖g6 34 g3 ♔c7 35 ♔h3 ♖g8 36 g4 ♔b6 37 ♖a1 ♖d8 38 ♔g3 ♖d3 39 ♖b1 ♔×a6 40 ♔f4 ♖d7 41 ♖h1 ♖d3 42 ♖b1 ♖d7 43 ♖h1 ♗c5 44 ♔e4 ♔b6 45 ♖h2 ♔c7 46 ♖h1 ♔b6 47 ♖h2 ♖d1 48 ♖h7 ♖b1 49 ♖×f7 ♖×b3 50 ♘d4 ♗×d4 51 ♔×d4 c5+ 52 ♔e4 ♖c3 53 ♖f2 ♖×c4+ 54 ♔f3 ♖c3+ 55 ♔f4 a5 56 g5 ♖c4+ 57 ♔f3 ♖c1 58 g6 ♖g1 59 ♖g2 ♖f1+ 60 ♔g4 ♖f8 61 ♔g5 ♖f5+ 62 ♔h6 ♖f8 63 g7 ♖b8 64 ♖f2 ♖g8 65 ♖f8 ♖×g7 66 ♔×g7 ♔c7 67 ♖a8 ♔b6 68 ♔f7 c4 69 ♔×e6 c3 70 ♖b8+ ♔a7 71 ♖c8 ♔b7 72 ♖c4 ♔a7 73 ♔f7 ♔a6 74 e6 c2 75 e7 ♔a7 76 e8(♕) c1(♕) 77 ♖×c1 1-0

**Awit – Rufus** 1 c4 e5 2 ♘c3 ♘f6 3 g3 ♗b4 4 a3 ♗c5 5 ♗g2 ♘c6 6 d3 0-0 7 ♘f3 d6 8 b4 ♗b6 9 0-0 ♗f5 10 ♘a4 ♕d7 11 ♘×b6 a×b6 12 ♗b2 ♖fe8 13 ♘d2 ♘g4 14 h3 ♘f6 15 g4 ♗g6 16 ♕b3 ♖ac8 17 b5 ♘a5 18 ♕b4 ♖b8 19 a4 c5 20 b×c6 b×c6 21 ♘b3 ♕c7 22 ♔h2 ♘×b3 23 ♕×b3 c5 24 ♕b5 ♖bc8 25 f4 e×f4 26 ♖f2 ♘d7 27 ♗h1 h5 28 ♗d5 h×g4 29 h×g4 ♘f6 30 ♗×f6 g×f6 31 ♖h1 ♖e5 32 ♔g1 ♔g7 33 ♔g2 ♖h8 34 ♖×h8 ♔×h8 35 ♕b2 ♕e7 36 ♔h2 ♕e8 37 d4 c×d4 38 ♕×d4 ♕×a4 39 ♕×f4 f5 40 ♕h6+ ♔g8 41 ♕×g6+ ♔h8 42 ♗×f7 d5 43 ♕g8# 1-0

**BS'66'76 – Mychess** 1 d4 d5 2 ♗f4 ♘f6 3 e3 ♘c6 4 ♘d2 e6 5 ♗b5 ♗d7 6 ♘e2 a5 7 ♘b3 a4 8 ♘c5 ♗×c5 9 d×c5 a3 10 b3 ♖a5 11 ♗×c6 ♗×c6 12 ♕d4 ♕e7 13 b4 ♖a8 14 f3 ♗b5 15 ♘c3 ♗c4 16 e4 c6 17 0-0-0 ♘h5 18 e×d5 c×d5 19 ♖he1 b6 20 ♗e5 b×c5 21 ♕g4 c×b4 22 ♕×h5 ♔f8 23 ♗×g7+ ♔×g7 24 ♘×d5 ♗×d5 25 ♖×d5 b3 26 a×b3 a2 27 ♖g5+ ♔f8 28 ♕h6+ ♔e8 29 ♔d2 a1(♗) 30 ♔d3 ♕d6+ 31 ♔e2 ♖a2 32 ♖d1 ♖×c2+ 33 ♔e1 ♗c3+ 0-1

*Round 4, October 30, 1979*

**Belle – Chess 4.9** 1 d4 ♘f6 2 c4 c5 3 d5 e6 4 ♘c3 e×d5 5 c×d5 d6 6 e4 g6 7 ♘f3 ♗g7 8 ♗e2 0-0 9 0-0 ♖e8 10 ♘d2 ♘a6 11 f3 ♘c7 12 a4 b6 13 ♘c4 ♗a6 14 ♗g5 h6 15 ♗h4 g5 16 ♗f2 ♘h5 17 ♘e3 ♗c8 18 ♕c2 ♘f4 19 ♗c4 ♗d7 20 ♖fd1 ♕f6 21 ♗g3 ♘h5 22 ♗e1 ♘f4 23 ♔h1 a6 24 ♗g3 b5 25 a×b5 a×b5 26 ♖×a8 ♖×a8 27 ♗f1 b4 28 ♘e2 b3 29 ♕b1 ♘h5 30 ♗f2 ♘f4 31 ♘c4 ♘×e2 32 ♗×e2 ♗b5 33 ♗g3 ♖a4 34 ♕c1 ♗f8 35 ♖d2 ♕d8 36 ♕f1 h5 37 ♔g1 h4 38 ♗f2 ♗g7 39 ♘e3 ♗×e2 40 ♕×e2 ♖a1+ 41 ♖d1 ♖a2 42 ♕d3 ♖×b2 43 ♘c4 ♖c2 44 e5 ♗×e5 45 ♘×e5 d×e5 46 ♕×b3 ♖e2 47 ♔f1 c4 48 ♕b7 ♖a2 49 ♗b6 h3 50 ♕×c7 ♕f6 51 ♕d8+ ♕×d8 52 ♗×d8 ♖×g2 53 ♖e1 c3 54 ♖×e5 c2 55 ♖e8+ ♔g7 56 ♗×g5 ♖×g5 57 ♖c8 ♖g2 58 d6 ♖×h2 59 d7 ♖d2 60 ♔g1 ♖×d7 61 ♖×c2 ♖d3 62 ♖f2 ♔f6 63 ♔h2 ½-½

**Duchess – Chaos** 1 e4 c5 2 ♘c3 ♘c6 3 g3 e6 4 ♘f3 d5 5 e×d5 e×d5 6 d4 ♗g4 7 ♗e2 ♘f6 8 ♗g5 ♗×f3 9 ♗×f3 ♕e7+ 10 ♔f1 c×d4 11 ♘×d5 ♕d8 12 ♕e2+ ♔d7 13 ♕b5 ♖b8 14 ♘×f6+ g×f6 15 ♕f5+ ♔c7 16 ♗×f6 ♗e7 17 ♗×h8 ♕×h8 18 ♕×f7 ♕g8 19 ♕×g8 ♖×g8 20 ♗e4 ♖h8 21 f4 ♘a5 22 ♔e2 ♘c4 23 ♖hb1 ♗f6 24 ♔d3 ♘b6 25 ♖e1 ♔b8 26 ♖ab1 ♘a4 27 ♗f5 h5 28 ♗g6 ♘c5+ 29 ♔c4 ♘d7 30 ♖e8+ ♖×e8 31 ♗×e8 ♘b6+ 32 ♔c5 h4 33 g×h4 ♔c8 34 h5 ♔d8 35 ♗b5 ♔c7 36 h6 a6 37 ♗d3 ♗h8 38 ♗f5 ♘a4+ 39 ♔b4 b5 40 ♔a5 ♔b7 41 ♗e4+ ♔a7 42 h7 ♘b6 43 h4 ♘c4+ 44 ♔b4 ♔b6 45 c3 a5+ 46 ♔b3 d×c3 47 ♖d1 ♘d2+ 48 ♖×d2 c×d2 49 ♔c2 b4 50 h5 ♔c5 51 ♔×d2 ♗×b2 52 h6 ♗h8 53 ♔e3 ♔c4 54 ♗c2 ♔d5 55 ♗b3+ ♔c5 56 ♔e4 ♔d6 57 ♔f5 ♔d7 58 ♔g6 ♔e7 59 f5 ♗e5 60 ♗c2 ♗f6 61 ♗e4 ♗h8 62 ♗f3 ♗e5 63 ♗c6 ♗f6 64 ♗e8 ♗e5 65 ♗b5 ♗h8 66 ♗c4 a4 67 ♗g8 ♗f6 68 ♗d5 ♗e5 69 ♔g5 ♗f6+ 70 ♔f4 ♔f8 71 ♔e4 ♔e7 72 ♗e6 a3 73 ♔d3 ♗h8 74 ♔c4 ♗c3 75 ♔c5 ♔d8 76 ♔d6 ♔e8 77 ♗b3 ♗b2 78 ♔e6 ♗h8 79 ♗a4+ ♔f8 80 ♗d1 ♔e8 81 f6 ♔f8 82 ♗b3 ♔e8 83 ♗a4+ ♔f8 84 ♗d1 ♔e8 85 ♗b3 ♔f8 86 ♗c4 ♔e8 87 ♗b5+ ♔f8 88 ♗a4 b3 89 ♗×b3 ♔e8 90 ♗c2 ♔f8 91 ♗d3 ♔e8 92 ♗b5+ ♔f8 93 ♗a6 ♔e8 94 ♗d3 ♔f8 95 ♗e2 ♔e8 96 ♗h5+ ♔f8 97 ♗g4 ♔e8 98 ♗f3 ♔f8 99 ♗e4 ♔e8 100 ♗g6+ ♔f8 101 ♗f5 ♔e8 102 ♗e4 ♔f8 103 ♗b1 ♔e8 104 ♗f5 ♔f8 105 ♗c2 ♔e8 106 ♗a4+ ♔f8 107 ♗d1 ♔e8 108 ♗e2 ♔f8 109 ♗f3 ♔e8 110 ♗c6+ ♔f8 111 ♗d5 1-0

**Mychess – L'Excentrique** 1 e4 e5 2 ♘f3 ♘f6 3 ♘×e5 d6 4 ♘f3 ♘×e4 5 d4 d5 6 ♗d3 ♘c6 7 0-0 ♗e7 8 ♖e1 ♗g4 9 ♗×e4 d×e4 10 ♖×e4 ♗×f3 11 g×f3 0-0 12 d5 ♕d7 13 ♕e2 ♕×d5 14 ♘c3 ♕d6 15 ♗f4 ♕g6+ 16 ♔h1 ♕f5 17 ♗×c7 ♕d7 18 ♖d1 ♕×c7 19 ♘d5 ♕a5 20 ♘×e7+ ♘×e7 21 ♖×e7 ♕×a2 22 ♖×b7 ♕e6 23 ♕×e6 f×e6 24 ♖dd7 ♖×f3 25 ♖×g7+ ♔f8 26 ♖×h7 ♔e8 27 ♖he7+ ♔d8 28 ♔g2 ♖f6 29 ♖ed7+ ♔e8 30 ♖e7+ ♔f8 31 ♖×a7 ♖g6+ 32 ♔f3 ♖×a7 33 ♖×a7 ♖h6 34 ♖a8+ ♔e7 35 ♔g3

♖g6+ 36 ♔f3 ♖h6 37 ♔g3 ♖g6+ 38 ♔f3 ♖h6 39 ♔g3 ♖g6+ ½-½

**Ostrich 80 – Awit** 1 e4 c5 2 ♘f3 d6 3 ♗b5+ ♘c6 4 0-0 ♗d7 5 ♘c3 ♘f6 6 d4 a6 7 ♗×c6 ♗×c6 8 ♗e3 ♕b6 9 b3 ♘×e4 10 ♘×e4 ♗×e4 11 c3 ♕c6 12 ♕e2 0-0-0 13 ♘g5 ♗g6 14 ♘f3 ♗h5 15 ♗g5 ♔d7 16 ♖ad1 ♗×f3 17 g×f3 c×d4 18 c×d4 ♕d5 19 f4 h6 20 ♗h4 g5 21 f×g5 h×g5 22 ♗g3 ♖c8 23 ♖d3 ♗g7 24 ♖fd1 e6 25 ♕e3 b5 26 f4 f6 27 f×g5 ♖c2 28 ♖1d2 ♖c1+ 29 ♗e1 ♕×g5+ 30 ♕×g5 f×g5 31 ♖e3 ♔e7 32 ♔g2 g4 33 ♖ee2 ♗f6 34 ♗g3 ♗g5 35 ♖d3 d5 36 a4 b×a4 37 b×a4 ♖a1 38 ♗e5 ♖g8 39 ♔g3 ♗h6 40 ♔h4 ♖×a4 41 ♖g2 a5 42 ♖×g4 ♖f8 43 ♔h5 ♗c1 44 ♖g7+ ♖f7 45 ♗d6+ ♔×d6 46 ♖×f7 ♖c4 47 ♔g6 ♖c2 48 h4 a4 49 ♖ff3 a3 50 ♖b3 a2 51 ♖b6+ ♔c7 52 ♖a6 ♔b7 53 ♖a4 ♔b6 54 ♖f6 ♔b5 55 ♖a8 ♗e3 56 ♖×e6 ♗×d4 57 ♖b8+ ♔a5 58 ♖a8+ ♔b5 59 ♖b8+ ♔c4 60 ♖c6+ ♔d3 61 ♖b3+ ♗c3 62 ♖a6 ♖g2+ 63 ♔h5 ♔c4 64 ♖ba3 a1(♕) 65 ♖6a4+ ♔d3 66 ♖×a1 ♗×a1 67 ♖×a1 ♖g8 68 ♖d1+ ♔e4 69 ♖e1+ ♔f5 70 ♖f1+ ♔e4 71 ♖e1+ ♔f4 72 ♖d1 ♔e5 73 ♖e1+ ♔d6 74 ♖d1 ♔e5 75 ♖e1+ ♔d6 76 ♖d1 ♔e5 ½-½

**Blitz 6.9 – Sargon** 1 e4 g6 2 d4 ♗g7 3 ♘f3 d6 4 ♗c4 ♘f6 5 ♕e2 ♘c6 6 0-0 ♗g4 7 c3 e5 8 d×e5 ♘×e5 9 ♘bd2 ♘×c4 10 ♘×c4 0-0 11 ♗g5 ♕e8 12 ♖fe1 ♕e6 13 h3 h6 14 ♗×f6 ♗×f3 15 ♕×f3 ♗×f6 16 ♘e3 ♗e5 17 ♘d5 ♖ac8 18 ♖ed1 ♖fe8 19 ♕e3 c6 20 ♘f4 ♕d7 21 ♖ac1 ♗g7 22 ♘d3 b6 23 ♕f4 ♕e6 24 ♖e1 g5 25 ♕g3 ♔h7 26 ♖cd1 c5 27 ♕h2 c4 28 ♘b4 ♗e5 29 ♕h1 ♕d7 30 ♘d5 ♕a4 31 a3 ♕c2 32 h4 g4 33 ♖b1 ♖c5 34 g3 ♗g7 35 ♕g2 ♔h8 36 f3 ♕×g2+ 37 ♔×g2 g×f3+ 38 ♔×f3 f5 39 ♖bd1 ♖e5 40 ♘f4 ♖×e4 41 ♖×e4 f×e4+ 42 ♔×e4 ♖e5+ 43 ♔f3 ♖b5 44 ♖d2 ♔h7 45 a4 ♖a5 46 ♖×d6 ♖×a4 47 ♖d7 ♔h8 48 ♘d5 ♖a2 49 g4 b5 50 g5 h×g5 51 h×g5 ♖×b2 52 ♖×a7 ♖d2 53 ♔e4 ♖e2+ 54 ♔f4 ♔h7 55 ♖d7 ♖f2+ 56 ♔g4 ♖g2+ 57 ♔h4 ♖h2+ 58 ♔g3 ♖a2 59 ♔f3 ♖a1 60 ♔e4 ♖f1 61 ♖d6 ♖e1+ 62 ♔f5 ♖f1+ 63 ♔g4 ♖c1 64 ♖d7 ♖e1 65 ♖b7 ♖g1+ 66 ♔f5 ♖f1+ 67 ♔e4 ♖e1+ 68 ♔f4 ♖f1+ 69 ♔g3 ♖f5 70 ♖×b5 ♖×g5+ 71 ♔f4 ♖g1 72 ♖b4 ♖d1 73 ♔e4 ♖e1+ 74 ♔f3 ♖f1+ 75 ♔e2 ♖f5 76 ♘e3 ♖f4 77 ♔d2 ♖f2+ 78 ♔d1 ♗×c3 79 ♖×c4 ♗g7 80 ♔e1 ½-½

**Rufus – BS'66'76** 1 c4 ♘f6 2 ♘c3 d5 3 c×d5 ♘×d5 4 d4 e6 5 e4 ♘×c3 6 b×c3 b6 7 ♘f3 ♗e7 8 ♗c4 0-0 9 0-0 c5 10 ♗f4 ♘c6 11 ♕d2 ♗b7 12 ♖fe1 c×d4 13 ♘×d4 ♘a5 14 ♗d3 ♖c8 15 ♗e5 ♖c5 16 ♘f3 ♘c6 17 ♗d4 ♖h5 18 ♖ab1 ♖a5 19 ♗c4 ♔h8 20 ♕c2 ♕c8 21 a4 ♖d8 22 ♗e3 ♘e5 23 ♘×e5 ♖×e5 24 ♗d3 ♖h5 25 ♗f4 e5 26 ♗e2 ♖h4 27 g3 ♖h3 28 ♗×e5 ♔g8 29 ♗f3 ♖h6 30 ♖ed1 ♖e8 31 ♗g2 g5 32 c4 ♕g4 33 a5 ♗c5 34 h3 ♕e6 35 a6 ♗×a6 36 ♖a1 ♗b7 37 ♖×a7 ♗c6 38 ♗c3 b5 39 ♖aa1 ♕×c4 0-1

**ACM 1980 – Nashville**

| | rate | perf | 1 | 2 | 3 | 4 | tot |
|---|---|---|---|---|---|---|---|
| 1 Belle | 2150 | 2294 | 6+ | 5+ | 2+ | 3+ | 4 |
| 2 Chaos | 1800 | 1952 | 7+ | 9+ | 1– | 5+ | 3 |
| 3 Challenger X | 0 | 1807 | 4= | 7+ | 9+ | 1– | 2½ |
| 4 BeBe | 0 | 1762 | 3= | 6+ | 5– | 9+ | 2½ |
| 5 Cray Blitz | 1850 | 1791 | 8+ | 1– | 4+ | 2– | 2 |
| 6 Mychess | 0 | 1545 | 1– | 4– | 8+ | 10+ | 2 |
| 7 Ostrich 81 | 1450 | 1372 | 2– | 3– | 10+ | 8= | 1½ |
| 8 Cube 2.0 | 0 | 1334 | 5– | 10+ | 6– | 7= | 1½ |
| 9 Awit | 1400 | 1337 | 10+ | 2– | 3– | 4– | 1 |
| 10 Clash | 1000 | 875 | 9– | 8– | 7– | 6– | 0 |

*Round 1, October 26*

**Awit – Clash** 1 b3 e5 2 ♗b2 ♘c6 3 e3 ♗c5 4 ♘f3 d6 5 ♗c4 ♘f6 6 d4 e×d4 7 e×d4 ♗b4+ 8 c3 ♕e7+ 9 ♗e2 ♗a5 10 0-0 ♘d5 11 ♗c4 ♘f4 12 ♖e1 ♗e6 13 d5 ♔d7 14 d×c6+ ♔×c6 15 ♘d4+ ♔c5 16 b4+ ♔×c4 17 ♘d2+ ♔d5 18 c4# 1-0

**Belle – Mychess** 1 e4 e5 2 ♘f3 ♘f6 3 d4 e×d4 4 e5 ♘e4 5 ♕×d4 d5 6 e×d6 ♘×d6 7 ♘c3 ♘c6 8 ♕f4 ♗f5 9 ♗b5 ♕e7+ 10 ♗e3 ♗×c2 11 ♗×c6+ b×c6 12 ♘d4 ♗e4 13 0-0 c5 14 ♘db5 ♘×b5 15 ♘×b5 g5 16 ♕g3 ♖c8 17 f3 c6 18 ♗×g5 f6 19 ♘×a7 ♖a8 20 ♖ae1 ♕×a7 21 ♗×f6 c4+ 22 ♔h1 ♗g7 23 ♗×g7 ♖g8 24 ♖×e4+ ♔f7 25 ♖fe1 ♖×g7 26 ♕f4+ ♔g8 27 ♖e8+ ♖×e8 28 ♖×e8# 1-0

**Ostrich 81 – Chaos** 1 e4 c5 2 ♘f3 ♘c6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘×c6 b×c6 6 ♘c3 e6 7 ♗e2 ♗b4 8 ♗d2 d5 9 f3 d×e4 10 ♘×e4 ♘×e4 11 f×e4 ♕h4+ 12 ♔f1 ♕f6+ 13 ♗f3 ♗a6+ 14 ♔f2 ♗c5+ 15 ♗e3 ♗×e3+ 16 ♔×e3 ♖d8 17 ♕c1 ♕g5+ 18 ♔f2 ♖d2+ 19 ♕×d2 ♕×d2+ 20 ♔g3 ♕×c2 21 b3 ♕d2 22 ♖he1 ♕g5+ 23 ♔f2 ♔e7 0-1

**Cray Blitz – Cube 2.0** 1 e4 e5 2 ♘f3 ♘c6 3 ♗c4 d6 4 ♘c3 ♘f6 5 ♘g5 d5 6 e×d5 ♘d4 7 d6 ♗e6 8 ♗×e6 f×e6 9 d×c7 ♕×c7 10 0-0 1-0

**Challenger X – Belle** 1 e4 c5 2 ♘f3 d6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 g6 6 f4 ♘bd7 7 ♗e2 ♗g7 8 0-0 0-0 9 ♗e3 a6 10 ♖c1 e6 11 ♘f3 ♘g4 12 ♕d2 ♘×e3 13 ♕×e3 ♕c7 14 ♖cd1 b5 15 ♕d2 b4 16 ♘a4 ♖b8 17 b3 d5 18 e×d5 e×d5 19 ♕×d5 ♕×f4 20 ♗c4 ♘b6 21 ♘×b6 ♕e3+ 22 ♔h1 ♕×b6 23 ♘g5 ♗b7 24 ♕d7 ♗c8 25 ♗×f7+ ♔h8 26 ♕d6 ♗f5 27 ♗c4 ♕×d6 28 ♖×d6 ♗c3 29 ♗×a6 ♗×c2 30 ♖d7 ♖×f1+ 31 ♗×f1 ♗g7 32 ♗c4 h6 33 ♘f7+ ♔h7 34 ♘d8 ♗f5 35 ♘e6 ♗×e6 36 ♗×e6 ♖a8 37 ♖d2 ♖a6 38 ♗c4 ♖a7 39 ♔g1 h5 40 ♖e2 ♗d4+ 41 ♔f1 ♔g7 42 ♖e4 ♗b2 43 ♗d3 ♗c3 44 ♖e2 ♖f7+ 45 ♖f2 ♖d7 46 ♗b5 ♖d5 47 ♗c4 ♖a5 48 g3 ♗d4 49 ♖d2 ♗c3 50 ♖e2 g5 51 ♖f2 h4 52 g4 h3 53 ♖f5 ♖×f5+ 54 g×f5 ♔f6 ½-½

*Round 2, October 26*

**Mychess – BeBe** 1 e4 c5 2 ♘f3 d6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 g6 6 ♗e3 ♗g7 7 ♗e2 0-0 8 ♕d2 ♘c6 9 0-0-0 ♗d7 10 ♔b1 ♖e8 11 h4 ♖c8 12 h5

♘×h5 13 ♗×h5 g×h5 14 ♘×c6 b×c6 15 ♗×a7 ♖a8 16 ♗d4 ♗×d4 17 ♕×d4 ♖a5 18 a4 ♖g5 19 f4 ♖×g2 20 ♖dg1 ♖×g1+ 21 ♕×g1+ ♔h8 22 ♕d4+ f6 23 ♖×h5 ♖g8 24 b3 ♖g3 25 b4 e5 26 f×e5 d×e5 27 ♕c5 ♗h3 28 ♖h4 ♕b8 29 ♘e2 ♖f3 30 a5 ♗f1 31 ♘g1 ♖g3 32 ♕e7 ♖g7 33 ♕c5 ♖×g1 34 ♕×g1 ♕×b4+ 35 ♔a2 ♗c4+ 36 ♔a1 ♕a3+ 37 ♔b1 ♗a2+ 38 ♔a1 ♗b3+ 39 ♔b1 ♕a2+ 40 ♔c1 ♕×c2# 0-1

**Cray Blitz – Belle** 1 e4 e5 2 ♘f3 ♘c6 3 ♗c4 ♘f6 4 ♘g5 d5 5 e×d5 ♘a5 6 ♗b5+ c6 7 d×c6 b×c6 8 ♗e2 h6 9 ♘f3 e4 10 ♘e5 ♗d6 11 f4 e×f3 12 ♘×f3 0-0 13 0-0 ♕c7 14 d4 c5 15 ♘c3 a6 16 d×c5 ♗×c5+ 17 ♔h1 ♗b7 18 ♕e1 ♖ae8 19 ♕h4 ♘e4 20 ♗f4 ♕c6 21 ♘×e4 ♕×e4 22 ♗d3 ♕e6 23 ♕h5 ♕b6 24 ♕f5 g6 25 ♕h3 h5 26 ♗h6 ♕×b2 27 ♗×f8 ♗×f8 28 ♕d7 ♖e7 29 ♕a4 ♕c3 30 ♕f4 ♗d5 31 ♗×a6 ♕×c2 32 ♖fc1 ♕b2 33 ♕d2 ♕×d2 34 ♘×d2 ♗g7 35 ♖ab1 ♗×a2 36 ♖b6 ♖e6 37 ♖b8+ ♔h7 38 ♗f1 ♗h6 39 ♖b2 ♗×d2 40 ♖×d2 ♘b3 41 ♖×a2 ♘×c1 42 ♖d2 ♔g7 43 g3 ♘b3 44 ♖d5 ♖c6 45 ♔g2 ♖c2+ 46 ♔h3 ♔f6 47 ♖d6+ ♔e5 48 ♖d8 ♘d4 49 ♗d3 ♖f2 50 ♖c8 g5 51 ♖h8 ♘f3 52 ♖×h5 f5 53 ♗×f5 ♔×f5 54 g4+ ♔f4 55 ♖h6 ♖×h2# 0-1

**Chaos – Awit** 1 d4 ♘f6 2 c4 g6 3 ♘c3 d5 4 c×d5 ♘×d5 5 e4 ♘×c3 6 b×c3 ♗g7 7 ♗c4 c5 8 ♘e2 0-0 9 0-0 ♘c6 10 ♗e3 c×d4 11 c×d4 ♗d7 12 ♖c1 ♖c8 13 f3 ♘a5 14 ♗d3 ♕b6 15 ♕d2 ♖×c1 16 ♖×c1 ♖b8 17 d5 ♕d8 18 ♗×a7 ♖c8 19 ♖×c8 ♗×c8 20 ♗c5 b6 21 ♗e3 ♗e5 22 ♕b4 ♗c7 23 ♕b2 ♕d6 24 ♗f4 ♕c5+ 25 ♔h1 ♘c4 26 ♕c3 ♗×f4 27 ♘×f4 b5 28 ♗e2 ♕a3 29 ♕×a3 ♘×a3 30 ♔g1 g5 31 ♘d3 ♔g7 32 ♔f2 h6 33 ♘b4 h5 34 ♔e3 f5 35 ♔d4 g4 36 ♗d3 g×f3 37 g×f3 ♔f6 38 e5+ ♔g6 39 ♘c6 ♔f7 40 ♘a7 ♗b7 41 ♗×f5 ♗a6 42 f4 ♘c4 43 ♗d3 h4 44 ♘×b5 ♗×b5 45 ♗×c4 ♗×c4 46 ♔×c4 ♔e8 47 d6 h3 48 ♔d5 e×d6 49 ♔×d6 ♔f7 50 e6+ ♔g6 1-0

**Ostrich – Challenger X** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗×c6 d×c6 5 d4 e×d4 6 ♕×d4 ♕×d4 7 ♘×d4 ♗d7 8 0-0 ♗d6 9 ♗e3 ♘f6 10 ♘c3 ♘g4 11 ♘f3 ♘×e3 12 f×e3 ♗e6 13 ♖fd1 0-0-0 14 ♘d4 ♗e5 15 ♘×e6 f×e6 16 ♖×d8+ ♔×d8 17 ♘d1 ♔c8 18 h3 ♖d8 19 ♔f1 ♖d2 20 c3 ♔d7 21 a4 ♔d6 22 ♖b1 ♗g3 23 c4 ♔c5 24 b3 ♔b4 25 a5 c5 26 e5 h5 27 e4 ♔×a5 28 ♖a1+ ♔b4 29 ♖b1 ♗×e5 30 ♘e3 ♔c3 31 ♔e1 ♔d3 32 ♘d1 ♗d4 33 b4 ♖×g2 34 ♖b3+ ♔×c4 35 ♘e3+ ♗×e3 36 ♖×e3 0-1

**Cube 2.0 – Clash** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 ♗c5 4 c3 ♕f6 5 b4 ♗b6 6 d3 ♘ge7 7 ♘a3 d5 8 ♗g5 ♕d6 9 e×d5 ♘×d5 10 ♘c4 ♕e6 11 ♗d2 e4 12 ♘g5 ♕f6 13 ♘×e4 ♕f5 14 ♘ed6+ c×d6 15 ♘×d6+ ♔e7 16 ♘×f5+ ♗×f5 17 ♕f3 ♔e6 18 ♗×c6 b×c6 19 c4 ♘e7 20 c5 ♗c7 21 ♕e3+ ♔d7 22 ♕d4+ ♘d5 23 ♕×g7 ♔c8 24 0-0-0 ♗g6 25 ♕f6 ♔b7 26 b5 ♗×h2 27 b×c6+ ♔b8 28 ♖×h2 ♘c7 29 ♕×g6 h×g6 30 ♖×h8+ ♘e8 31 ♗f4+ ♔c8 32 ♖×e8# 1-0

*Round 3, October 27*

**Clash – Ostrich 81** 1 e4 d5 2 ♘c3 ♘f6 3 ♘×d5 ♘×e4 4 ♘c3 ♘×c3 5 d×c3 ♕×d1+ 6 ♔×d1 ♗f5 7 ♗d3 e6 8 ♗f4 ♗c5 9 ♔e2 ♗×d3+ 10 c×d3 ♘a6 11 ♗e5 0-0 12 b4 f6 13 b×c5 f×e5 14 d4 e×d4 15 c×d4 e5 16 d×e5 ♘×c5 17 ♖d1 ♘e4 18 ♘f3 ♘c3+ 19 ♔e3 ♘×d1+ 20 ♖×d1 c5 21 ♖d7 b6 22 e6 ♖ae8 23 e7 ♖f7 24 ♖×a7 ♖e×e7+ 25 ♖×e7 ♖×e7+ 26 ♔d3 ♖a7 27 h3 ♖×a2 28 ♔e3 c4 29 g4 b5 30 ♘e5 ♖a3+ 31 ♔e4 ♖×h3 32 f3 c3 33 ♔d4 c2 34 ♘d3 ♖×f3 35 ♘e1 ♖f4+ 36 ♔e5 c1(♕) 37 ♘d3 ♕e3+ 38 ♔d6 ♕×d3+ 39 ♔c6 ♖×g4 40 ♔c7 ♖g6 41 ♔b8 ♕d8+ 42 ♔b7 ♖b6+ 43 ♔a7 ♕b8# 0-1

**Belle – Chaos** 1 e4 c5 2 c3 ♘f6 3 e5 ♘d5 4 d4 c×d4 5 c×d4 d6 6 ♘f3 ♘c6 7 ♗c4 ♘b6 8 ♗b5 e6 9 0-0 ♗e7 10 e×d6 ♕×d6 11 ♘c3 0-0 12 ♖e1 ♗d7 13 ♗e3 ♘d5 14 ♗a4 ♘×e3 15 f×e3 ♖fc8 16 e4 a6 17 ♗×c6 ♖×c6 18 ♘e5 ♖c7 19 ♖e2 ♗e8 20 ♘f3 ♖ac8 21 e5 ♕b4 22 d5 ♕c5+ 23 ♔h1 e×d5 24 ♘×d5 ♖d8 25 ♖d2 ♖cd7 26 b4 ♕×d5 27 ♖×d5 ♖×d5 28 ♕e1 ♗a4 29 ♕c3 ♗c6 30 a3 ♗g5 31 ♕c2 ♗f4 32 ♖e1 ♖d3 33 ♕c4 ♗h6 34 e6 ♖×a3 35 e×f7+ ♔f8 36 ♕c5+ ♔×f7 37 ♕e7+ ♔g8 38 ♕×d8+ 1-0

**Awit – Challenger X** 1 b3 d5 2 ♗b2 ♘f6 3 e3 e6 4 ♘f3 ♗d6 5 ♗e2 0-0 6 0-0 ♗d7 7 ♘a3 ♘g4 8 h3 ♕e7 9 ♘b5 ♗×b5 10 ♗×b5 ♘f6 11 ♘e5 c6 12 ♗e2 ♘bd7 13 f4 ♘e4 14 ♘×d7 ♕×d7 15 ♕e1 ♕d8 16 d3 ♘f6 17 ♗f3 ♖c8 18 a4 ♖e8 19 ♕g3 ♗c5 20 ♖fe1 ♗b4 21 ♖e2 ♗e7 22 ♖ae1 ♖f8 23 ♔h2 ♗b4 24 c3 ♗d6 25 e4 d×e4 26 d×e4 ♗c5 27 b4 ♗e7 28 ♕f2 c5 29 ♖d2 ♕c7 30 e5 c×b4 31 e×f6 ♗×f6 32 c×b4 ♕×f4+ 33 g3 ♕×b4 34 ♗×f6 g×f6 35 ♖b2 ♕×a4 36 ♗×b7 ♖c4 37 ♖a2 ♕b4 38 ♖×a7 f5 39 ♕e3 ♖c2+ 40 ♔g1 ♖b8 41 ♗f3 f4 42 g×f4 ♖c3 43 ♕e5 ♖×f3 44 ♔g2 ♕×f4 45 ♕×f4 ♖×f4 46 ♔g3 ♖c4 47 ♖e7 ♖b3+ 48 ♔h2 ♖c2+ 49 ♔g1 ♖g3+ 50 ♔f1 ♖f3+ 51 ♔g1 ♖×h3 52 ♖f1 ♖g3+ 53 ♔h1 ♖h3+ 54 ♔g1 h5 55 ♖e×f7 ♖g3+ 56 ♔h1 h4 57 ♖1f4 e5 58 ♖f8+ ♔g7 59 ♖8f7+ ♔g6 60 ♖4f6+ ♔h5 61 ♖h7+ ♔g5 62 ♖hh6 e4 63 ♖hg6+ ♔h5 64 ♖h6+ ♔g4 65 ♖c6 ♖×c6 66 ♖×c6 0-1

**BeBe – Cray Blitz** 1 e4 c5 2 ♘f3 e6 3 ♗e2 ♘c6 4 0-0 ♘f6 5 ♘c3 ♗e7 6 d4 c×d4 7 ♘×d4 0-0 8 ♗f4 ♘×d4 9 ♕×d4 d6 10 e5 d×e5 11 ♕×e5 ♕b6 12 ♖ab1 ♗d7 13 ♖fd1 ♗c5 14 ♗g3 ♖ac8 15 a3 ♖fd8 16 b4 ♗e7 17 ♗f4 ♗e8 18 ♖×d8 ♗×d8 19 ♖b3 ♗c7 20 ♕g5 ♗×f4 21 ♕×f4 ♖d8 22 ♗f3 ♗c6 23 ♗×c6 ♕×c6 24 ♔f1 b6 25 a4 ♕e8 26 a5 e5 27 ♕c4 ♖d2 28 a×b6 a×b6 29 ♖b2 e4 30 ♖a2 h6 31 ♕c7 ♕e6 32 b5 ♘g4 33 ♔e1 ♖×f2 34 h3 ♖×g2 35 h×g4 ♖g1+ 36 ♔f2 ♖×g4 37 ♖a8+ ♔h7 38 ♔e3 f5 39 ♖b8 f4+ 40 ♔d4 e3 41 ♕c8 f3+ 42 ♔d3 0-1

**Cube 2.0 – Mychess** 1 d4 ♘f6 2 c4 g6 3 ♘f3 e6 4 ♕a4 ♘c6 5 a3 a5 6 ♘c3 ♗g7 7 e4 ♘g4 8 h3 ♘h6 9 d5 ♘e7 10 d×e6 f×e6 11 ♗g5 ♘f7 12 ♗h4 0-0 13 ♖d1 ♘e5 14 ♘×e5 ♗×e5 15 ♗d3 d6 16 ♗c2 c5 17 ♕b3 ♕c7 18 ♘b5 ♕d7 19 ♕d3 ♖a6 20 b3 ♔g7 21 ♗g3 ♗×g3 22 f×g3 d5 23 e×d5 e×d5 24 c×d5 ♘×d5 25 ♘c3 ♖e6+ 26 ♘e4 ♖e5 27 ♖f1 ♕e7 28 ♖×f8 ♕×f8 29 ♔d2 ♗f5 30 ♕f3 ♕e7 31 ♔c1 ♖×e4 32 g4 ♗×g4 33 h×g4 ♕g5+ 34 ♔b1 ♖d4 35 ♖×d4 c×d4 36 ♗d3 h6 37 ♗c4 ♘c3+ 0-1

*Round 4, October 28*

**Clash — Mychess** 1 e4 e5 2 ♗c4 ♘f6 3 ♘c3 ♘c6 4 d3 ♗c5 5 ♕f3 d6 6 ♗e3 ♗g4 7 ♕g3 ♗×e3 8 ♕×e3 ♘d4 9 ♖c1 0-0 10 ♘ce2 b5 11 ♗b3 ♘×b3 12 c×b3 c5 13 ♘f3 ♕a5+ 14 ♘c3 b4 15 ♘d5 ♘×d5 16 e×d5 ♕×a2 17 ♖c4 f5 18 h3 ♗×f3 19 ♕×f3 a5 20 0-0 ♕×b2 21 ♕d1 a4 22 b×a4 ♕a2 23 ♕f3 ♕×a4 24 g4 f×g4 25 ♕×g4 ♖f6 26 ♕e4 ♖f4 27 ♕e3 ♖×c4 28 d×c4 ♕a3 29 ♖e1 ♕×e3 30 ♖×e3 ♖a3 31 ♔g2 ♖×e3 32 f×e3 b3 33 ♔f3 b2 34 h4 b1(♕) 35 e4 ♕d3+ 36 ♔f2 ♕×e4 37 h5 ♕×c4 38 ♔g3 g6 39 h×g6 h×g6 40 ♔g2 e4 41 ♔g3 ♕×d5 42 ♔h4 e3 43 ♔g3 ♕f5 44 ♔h2 e2 45 ♔g3 c4 46 ♔g2 c3 47 ♔g3 c2 48 ♔g2 d5 49 ♔g3 d4 50 ♔g2 d3 51 ♔h2 d2 52 ♔g3 c1(♖) 53 ♔g2 d1(♖) 54 ♔g3 ♖g1+ 55 ♔h4 ♕g4# 0-1

**Challenger X — Belle** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗a4 ♘f6 5 0-0 ♘×e4 6 d4 b5 7 ♗b3 d5 8 d×e5 ♗e6 9 c3 ♗c5 10 ♗e3 ♗×e3 11 f×e3 ♖b8 12 ♘bd2 ♘c5 13 ♕e1 ♘d3 14 ♕g3 0-0 15 ♖ab1 ♕e7 16 ♗c2 ♕c5 17 ♘g5 ♘c×e5 18 ♘×h7 ♖fd8 19 ♘f6+ ♔f8 20 ♗×d3 ♘×d3 21 ♘h5 g6 22 ♘f6 ♗f5 23 ♕g5 b4 24 ♘h7+ ♔e8 25 ♘f6+ ♔f8 26 ♘b3 ♕c4 27 ♘h7+ ♔e8 28 c×b4 ♖×b4 29 ♘f6+ ♔f8 30 ♘g4 ♖d6 31 ♘h6 ♕c4 32 ♖×f5 g×f5 33 ♕g8+ ♔e7 34 ♕×f7+ ♔d8 35 ♘×f5 ♖c6 36 h4 ♖c2 37 ♕f8+ ♔d7 38 ♕f7+ ♔c8 39 ♕g6 ♔b7 40 ♘a5+ ♔a8 41 ♕×a6+ ♔b8 42 ♕g6 ♘e5 43 ♕g8+ ♔a7 44 ♖f1 ♖b×b2 45 ♔h1 ♖×g2 46 ♘c6+ ♘×c6 47 ♕a8+ ♔×a8 48 a4 ♔b8 49 a5 ♔a8 50 a6 ♔a7 51 h5 ♔b6 52 a7 ♖g1+ 53 ♔×g1 ♕g2# 0-1

**Chaos — Cray Blitz** 1 d4 d5 2 c4 e6 3 ♘c3 ♘f6 4 ♗g5 ♗e7 5 e3 0-0 6 ♘f3 h6 7 ♗h4 ♘e4 8 ♗×e7 ♕×e7 9 c×d5 ♘×c3 10 b×c3 e×d5 11 ♕b3 ♖d8 12 c4 d×c4 13 ♗×c4 ♘c6 14 ♗e2 ♖d6 15 0-0 ♗e6 16 ♕c3 ♖ad8 17 ♖ab1 ♗×a2 18 ♖×b7 a5 19 ♗d3 ♗d5 20 ♖fb1 a4 21 ♕a1 ♖a8 22 ♘d2 ♖dd8 23 ♘c4 ♕e6 24 ♖×c7 ♕g4 25 ♗f1 ♘a5 26 ♘b6 ♘b3 27 ♕a2 ♗e4 28 ♘×a8 ♗×b1 29 ♕×b1 ♖×a8 30 ♗c4 ♘×d4 31 e×d4 ♕×d4 32 ♗×f7+ ♔h8 33 g3 ♖f8 34 ♕f5 ♕a1+ 35 ♔g2 ♕b2 36 ♗g6 ♕b8 37 ♕d7 ♕a8+ 38 ♔g1 ♖g8 39 ♖a7 ♕f3 40 ♖×a4 ♕c3 41 ♕d5 ♖f8 42 ♔g2 ♕b2 43 ♗f5 ♕b8 44 ♗d3 ♕b2 45 ♖f4 ♖g8 46 ♕e4 1-0

**BeBe — Awit** 1 e4 c5 2 ♘f3 d6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 a6 6 ♗e2 e5 7 ♘f3 ♘c6 8 0-0 ♗e7 9 ♗e3 ♗d7 10 ♘d5 0-0 11 ♗b6 ♕c8 12 ♘×f6+ g×f6 13 c3 f5 14 ♗d3 ♗d8 15 ♗×d8 ♕×d8 16 e×f5 d5 17 ♗c2 ♕b6 18 f6 ♕b5 19 ♕d2 ♖fc8 20 ♕h6 1-0

**Ostrich — Cube 2.0** 1 e4 c5 2 ♘f3 ♘f6 3 ♘c3 ♘c6 4 ♗e2 d6 5 0-0 ♗g4 6 h3 ♗×f3 7 ♗×f3 g6 8 ♖e1 ♗h6 9 e5 ♘×e5 10 ♗×b7 ♖b8 11 ♗a6 ♕a5 12 ♗e2 0-0 13 d3 ♗×c1 14 ♕×c1 ♘c6 15 ♖b1 ♘d4 16 a4 ♕b4 17 ♔h1 a5 18 ♕d1 ♕b7 19 b3 ♕b4 20 ♘e4 ♘×e4 21 d×e4 ♘×e2 22 ♖×e2 c4 23 ♖e3 ♖fc8 24 ♕f3 ♖b6 25 ♕g4 ♖cb8 26 ♕h4 ♕d2 27 ♖be1 ♕×c2 28 ♖1e2 ♕d1+ 29 ♖e1 ♕d4 30 b×c4 e5 31 ♖c1 ♖b4 32 ♖a3 ♖×c4 33 ♖×c4 ♕×c4 34 ♖a1 f5 35 f3 f×e4 36 f×e4 d5 37 ♕e1 ♕×e4 38 ♕×a5 ♖b1+ 39 ♖×b1 ♕×b1+ 40 ♔h2 ♕b7 41 ♕d8+ ♔g7 42 a5 ♕f7 43 a6 ♕f4+ 44 ♔h1 ♕f1+ 45 ♔h2 ♕×a6 46 ♕e7+ ♔h6 47 ♕×e5 ♕d3 48 ♕f4+ ♔g7 49 ♕e5+ ♔g8 50 ♕e8+ ♔g7 51 ♕e7+ ♔h6 52 ♕h4+ ♔g7 ½-½

**ACM 1981 — Los Angeles**

| | rate | perf | 1 | 2 | 3 | 4 | tot |
|---|---|---|---|---|---|---|---|
| 1 Belle | 2200 | 2306 | 7+ | 8+ | 2= | 3+ | 3½ |
| 2 Nuchess | 0 | 2119 | 10+ | 4+ | 1= | 5= | 3 |
| 3 Cray Blitz | 0 | 2174 | 6+ | 5+ | 11+ | 1− | 3 |
| 4 Bebe | 1800 | 1871 | 12+ | 2− | 14+ | 7+ | 3 |
| 5 Duchess | 1900 | 1998 | 15+ | 3− | 9+ | 2= | 2½ |
| 6 Philidor | 0 | 1923 | 3− | 15+ | 8+ | 9= | 2½ |
| 7 Ostrich | 0 | 1611 | 1− | 16+ | 12+ | 4− | 2 |
| 8 L'Excentrique | 0 | 1825 | 16+ | 1− | 6− | 11+ | 2 |
| 9 Ch Challenger X | 0 | 1768 | 14+ | 11= | 5− | 6= | 2 |
| 10 Mychess | 0 | 1497 | 2− | 12− | 15+ | 14+ | 2 |
| 11 Chaos | 1800 | 1680 | 13+ | 9= | 3− | 8− | 1½ |
| 12 Cube 2.1 | 0 | 1425 | 4− | 10+ | 7− | 13= | 1½ |
| 13 Schach 2.5 | 0 | 1245 | 11− | 14− | 16+ | 12= | 1½ |
| 14 Chaturanga | 0 | 1319 | 9− | 13+ | 4− | 10− | 1 |
| 15 Awit | 1400 | 1296 | 5− | 6− | 10− | 16+ | 1 |
| 16 Prodigy | 1000 | 871 | 8− | 7− | 13− | 15− | 0 |

*Round 1, November 8, 1981*

**Belle — Ostrich** 1 e4 d5 2 e×d5 ♘f6 3 d4 ♘×d5 4 c4 ♘b6 5 ♘f3 ♗g4 6 ♗e2 c6 7 ♘a3 e6 8 c5 ♘d5 9 ♗g5 f6 10 ♗d2 ♕e7 11 ♘c4 ♕d7 12 ♘h4 ♗f5 13 ♘×f5 e×f5 14 0-0 ♗e7 15 ♗d3 0-0 16 ♘e3 ♘×e3 17 f×e3 g6 18 ♗c4+ ♔h8 19 b4 b5 20 ♗b3 a6 21 a4 ♕b7 22 ♗c3 ♗d8 23 ♗e6 ♕e7 24 d5 b×a4 25 ♖e1 c×d5 26 ♗×d5 ♖a7 27 ♖×a4 ♕e8 28 ♗c4 h5 29 ♕a1 ♕c6 30 b5 ♕c8 31 b×a6 ♕c6 32 ♖d1 ♗e7 33 ♗d5 Machine failure. 1-0

**Philidor — Cray Blitz** 1 e4 e5 2 ♘f3 ♘c6 3 ♗c4 ♗c5 4 0-0 ♘f6 5 d4 e×d4 6 c3 d×c3 7 e5 c×b2 8 ♗×b2 ♘g4 9 ♗×f7+ ♔×f7 10 ♕d5+ ♔g6 11 ♕×c5 d6 12 ♕b5 ♘c×e5 13 ♗×e5 ♘×e5 14 ♘×e5+ d×e5 15 ♕×e5 c6 16 ♕e4+ ♗f5 17 ♕e5 ♖e8 18 ♕g3+ ♕g5 19 ♕d6+ ♖e6 20 ♕a3 ♖d8 21 f3 ♗d3 22 f4 ♕f6 23 f5+ ♗×f5 24 ♕g3+ ♕g5 25 ♕f2 ♕e3 26 h3 ♕×f2+ 27 ♔×f2 ♗d3 28 ♖c1 ♖e2+ 29 ♔g1 ♗e4 30 g4 ♖g2+ 31 ♔f1 ♖h2 32 ♘c3 ♖f8+ 33 ♔e1 ♗d3 34 ♘d5 ♖e2+ 35 ♔d1 ♖f1# 0-1

**Chaos — Schach 2.5** 1 d4 d5 2 c4 e6 3 ♘c3 ♘f6 4 ♗g5 ♗e7 5 e3 0-0 6 ♘f3 h6 7 ♗h4 b6 8 c×d5 ♘×d5 9 ♗×e7 ♕×e7 10 ♘×d5 e×d5 11 ♗d3 ♗e6 12 ♕c2 ♕b4+ 13 ♕c3 ♕×c3+ 14 b×c3 ♘d7 15 ♖b1 ♖ae8 16 0-0 c5 17 d×c5 ♘×c5 18 ♗b5 ♗d7 19 ♗×d7 ♘×d7 20 ♖b5 ♘f6 21 ♖fb1 ♖e4 22 ♘d4 ♖c8 23 ♖5b3 ♔f8 24 ♘b5 ♖a4 25 ♖1b2 ♔e7 26 f3 ♖cc4 27 ♘d4 ♘d7 28 ♘f5+ ♔f6 29 ♘d6 ♖c6 30 ♘b5 ♘c5 31 ♘d4 ♖c7 32 ♖b5 ♘d3 33 ♖c2 ♖e7 34 ♖e2 ♖e5 35 f4 ♖h5 36 g4 ♖h4 37 ♖g2 ♘c5 38 ♘f5 ♖h3 39 ♖d2 h5 40 ♖×d5 h×g4 41 ♖b2 ♘e4 42 ♘d4 ♖×e3 43 ♖f5+ ♔g6 44 ♖e5 ♔f6 45 ♖f5+ ♔e7 46 ♖e5+ ♔f8 47 f5 Time. 1-0

**Mychess – Nuchess** 1 e4 c5 2 ♘f3 ♘c6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 d6 6 ♗g5 e6 7 ♗e2 ♗e7 8 ♘db5 0-0 9 0-0 a6 10 ♗×f6 g×f6 11 ♘d4 ♔h8 12 ♘×c6 b×c6 13 ♕d2 ♗d7 14 ♖ad1 ♕a5 15 b3 d5 16 e×d5 ♗b4 17 ♕e3 ♗×c3 18 d×e6 ♗×e6 19 ♖d3 ♗b2 20 ♗f3 ♗f5 21 ♖d6 ♖ae8 22 ♕d2 ♕×d2 23 ♖×d2 ♗c3 24 ♖d6 ♖e6 25 ♖dd1 ♖fe8 26 ♖c1 ♖d6 27 g4 ♖g8 28 h3 h5 29 ♗e2 h×g4 30 h×g4 ♗×g4 31 ♗×g4 ♖×g4+ 32 ♔h2 f5 33 f3 ♖h6# 0-1

**Duchess – Awit** 1 e4 c5 2 d4 c×d4 3 c3 d×c3 4 ♘×c3 ♘c6 5 ♘f3 d6 6 ♗c4 e6 7 0-0 ♘f6 8 ♕e2 ♗e7 9 ♖d1 0-0 10 e5 ♘e8 11 e×d6 ♘×d6 12 ♗f4 ♘b4 13 a3 ♘c6 14 ♗×d6 ♗×d6 15 ♘e4 ♗×h2+ 16 ♘×h2 ♕e7 17 ♕h5 ♗d7 18 ♖ac1 ♖ad8 19 ♘d6 ♗e8 20 ♘×e8 ♖×d1+ 21 ♖×d1 ♖×e8 22 ♘f3 ♕c7 23 ♕h4 h6 24 ♕h2 ♕×h2+ 25 ♔×h2 ♖d8 26 ♖×d8+ ♘×d8 27 ♔g3 g5 28 ♘d4 h5 29 ♔f3 ♔f8 30 ♔e4 g4 31 ♔f4 ♔g7 32 ♔g5 a6 33 ♔×h5 b5 34 ♗b3 e5 35 ♘f5+ ♔f6 36 ♔×g4 ♘b7 37 ♘g3 ♘c5 38 ♗c2 ♔e7 39 ♔f5 ♘d7 40 b4 ♔d8 41 ♘e4 ♔e7 42 g4 ♔e8 43 ♘f6+ ♔e7 44 ♘×d7 ♔×d7 45 ♔×e5 ♔e7 46 ♗e4 ♔e8 47 ♔f6 ♔f8 48 ♗b7 1-0

**Prodigy – L'Excentrique** 1 d4 f5 2 ♗g5 g6 3 e3 h6 4 ♗h4 ♘f6 5 ♘f3 g5 6 ♗g3 e6 7 ♘bd2 ♘e4 8 ♗e5 ♖h7 9 ♘×e4 f×e4 10 ♘d2 ♖f7 11 ♘×e4 ♘c6 12 f3 d5 13 ♘g3 ♗d7 14 ♗h8 ♖h7 15 ♗e5 ♘×e5 16 d×e5 ♕e7 17 ♗d3 ♖h8 18 ♗g6+ ♔d8 19 ♕d4 c5 20 ♕c3 ♕g7 21 ♗h5 a5 22 0-0-0 b5 23 f4 b4 24 ♕d3 g×f4 25 e×f4 c4 26 ♕d4 a4 27 ♕b6+ ♔c8 28 c3 b3 29 a3 ♗e7 30 ♖d2 ♕f8 31 ♕f2 ♗c5 32 ♕f3 ♗c6 33 f5 e×f5 34 ♘×f5 ♔c7 35 ♘d4 ♕×f3 36 g×f3 ♗e8 37 ♗g4 ♔b7 38 ♔d1 h5 39 ♗e6 ♖d8 40 h4 ♖f8 41 ♘f5 ♗c6 42 ♖e1 ♔c7 43 ♖ee2 ♖de8 44 ♘g7 ♖e7 45 ♖g2 d4 46 ♖ge2 ♖×g7 47 ♔c1 d×c3 48 ♖d1 ♗×f3 49 ♗×c4 ♗×e2 50 ♗×b3 0-1

**Bebe – Cube 2.1** 1 e4 c5 2 ♘f3 ♘f6 3 ♘c3 ♘c6 4 ♗b5 ♘d4 5 ♗a4 ♕a5 6 e5 ♘g8 7 0-0 g6 8 ♖b1 ♗g7 9 ♖e1 b5 10 ♘×d4 b×a4 11 ♘db5 ♖b8 12 ♖e4 ♖×b5 13 ♖×a4 ♕b6 14 ♘×b5 ♕×b5 15 c4 ♕b6 16 ♕e2 ♘h6 17 d3 ♗b7 18 ♗×h6 ♗×h6 19 ♖a3 ♗f4 20 ♖b3 ♕c7 21 ♖e1 ♕c6 22 ♕e4 ♕×e4 23 ♖×e4 ♗×h2+ 24 ♔×h2 ♗c8 25 ♖b5 0-0 26 ♖×c5 1-0

**Chaturanga – Chess Challenger X** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗a4 ♘f6 5 0-0 ♗e7 6 ♘c3 b5 7 ♗b3 d6 8 ♖b1 ♗g4 9 g3 ♘d4 10 ♗d5 ♗×f3 11 ♗c6+ ♔f8 12 ♕e1 ♘×c2 13 ♗×a8 ♘×e1 14 ♗b7 ♕d7 15 ♖×e1 ♕h3 16 ♘d5 ♕g2# 0-1

*Round 2, November 9, 1981*

**L'Excentrique – Belle** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗a4 ♘f6 5 0-0 ♘×e4 6 ♕e2 ♘c5 7 ♗×c6 d×c6 8 d4 ♘e6 9 d×e5 ♘d4 10 ♘×d4 ♕×d4 11 h3 ♗e7 12 c3 ♕b6 13 ♖d1 ♗e6 14 b4 a5 15 ♗e3 ♕a6 16 ♕×a6 ♖×a6 17 ♗c5 a×b4 18 ♗×e7 ♔×e7 19 c×b4 ♖×a2 20 ♖×a2 ♗×a2 21 ♘c3 ♗b3 22 ♖d2 ♖f8 23 g4 f6 24 e×f6+ ♖×f6 25 ♖b2 ♗c4 26 ♔g2 ♗d5+ 27 ♘×d5+ c×d5 28 ♖d2 ♔d6 29 f3 ♖f4 30 ♖b2 ♖c4 31 h4 d4 32 ♖b3 ♔c6 33 b5+ ♔b6 34 ♔g3 ♖c5 35 ♖d3 ♖d5 36 h5 ♔×b5 37 ♖b3+ ♔a6 38 ♖a3+ ♖a5 39 ♖d3 c5 40 f4 ♖a1 41 g5 ♔b5 42 f5 ♖h1 43 f6 g×f6 44 g×f6 ♖g1+ 45 ♔f4 ♖g8 46 ♔f5 ♔c4 47 ♖d1 ♖f8 48 ♔e6 ♔c3 49 f7 ♔c2 50 ♖a1 d3 51 ♔e7 ♖×f7+ 52 ♔×f7 d2 53 ♔g7 c4 54 ♖a2+ ♔d3 55 ♖a8 d1(♕) 56 h6 c3 Adjudicated. 0-1

**Cray Blitz – Duchess** 1 e4 e5 2 ♘f3 ♘c6 3 ♗c4 ♗c5 4 c3 ♘f6 5 d4 e×d4 6 c×d4 ♗b4+ 7 ♗d2 ♗×d2+ 8 ♘b×d2 d5 9 e×d5 ♘×d5 10 ♕b3 ♘ce7 11 0-0 0-0 12 ♖fe1 c6 13 ♖ac1 a5 14 ♘e4 a4 15 ♕a3 ♘f5 16 h3 ♘b6 17 ♕c5 ♘×c4 18 ♖×c4 ♗e6 19 ♖b4 ♕c7 20 g4 ♘h6 21 ♘d6 ♖fb8 22 ♘c4 ♖a6 23 ♕d6 ♕×d6 24 ♘×d6 b5 25 a3 ♖aa8 26 ♘g5 ♗d5 27 f4 ♖f8 28 f5 ♖fd8 29 ♘b7 ♖e8 30 ♖×e8+ ♖×e8 31 ♘d6 ♖e7 32 ♘c8 ♖e1+ 33 ♔f2 ♖e8 34 ♘b6 ♗b3 35 ♘f3 ♖e7 36 ♘e5 ♖b7 37 ♘×a4 ♗×a4 38 b3 ♖c7 39 b×a4 f6 40 ♘d3 b×a4 41 ♘f4 Adjudicated. 1-0

**Chess Challenger X – Chaos** 1 e4 ♘f6 2 e5 ♘d5 3 d4 d6 4 ♘f3 d×e5 5 ♘×e5 g6 6 g3 ♗g7 7 ♗c4 f6 8 ♘f3 ♘c6 9 0-0 ♗h3 10 ♖e1 ♗g4 11 ♗b5 0-0 12 ♗×c6 b×c6 13 ♗d2 ♕b8 14 b3 f5 15 c3 f4 16 g×f4 ♘×f4 17 ♗×f4 ♖×f4 18 ♖e3 ♕b5 19 ♘bd2 ♖af8 20 h3 ♗×h3 21 ♘e5 ♗e6 22 c4 ♕b6 23 c5 ♕b5 24 a4 ♕b7 25 ♘df3 ♗×e5 26 ♘×e5 ♗d5 27 ♖a2 ♕b4 28 ♖d3 e6 29 f3 g5 30 ♖h2 a6 31 ♖h5 ♖8f5 32 ♖h3 a5 33 ♖h5 ♔g7 34 ♔h2 h6 35 ♔g1 ♖f8 36 ♖h2 ♔h7 37 ♕b1 ♔g7 38 ♕d1 ♔h7 39 ♕b1 ♔g7 40 ♕d1 ½-½

**Nuchess – Bebe** 1 f4 d5 2 ♘f3 ♘f6 3 e3 e6 4 b3 ♗d6 5 ♗e2 0-0 6 0-0 ♘c6 7 ♗b2 a6 8 ♘e5 ♗×e5 9 f×e5 ♘d7 10 ♗a3 ♖e8 11 d4 ♕g5 12 ♕d3 b5 13 ♘d2 b4 14 ♗b2 a5 15 c4 a4 16 ♖ac1 ♗a6 17 e4 a3 18 ♗a1 ♘b6 19 e×d5 e×d5 20 ♖cd1 d×c4 21 ♘×c4 ♘×c4 22 b×c4 ♘a5 23 ♖c1 c6 24 ♕f3 ♖e7 25 ♕e4 ♖ae8 26 ♗d3 ♕h6 27 ♕e1 ♖b7 28 ♖b1 ♗×c4 29 ♗×c4 ♘×c4 30 ♖×b4 ♖×b4 31 ♕×b4 ♘e3 32 ♖f3 ♘g4 33 h3 ♕c1+ 34 ♖f1 ♕e3+ 35 ♔h1 ♘f2+ 36 ♔h2 ♘e4 37 ♖f3 ♕c1 38 ♗c3 ♘g5 39 ♖f5 ♘e6 40 ♕b3 ♖d8 41 ♖f3 ♘g5 42 ♖f2 ♕e3 43 ♖f5 ♘e6 44 ♕b7 ♖f8 45 ♖f3 ♕c1 46 ♗b4 c5 47 d×c5 ♕b2 48 ♖e3 ♘×c5 49 ♕b5 ♘e6 50 ♖×a3 ♖c8 51 ♖c3 ♖a8 52 ♕c6 ♖d8 53 ♗e7 ♖b8 54 ♖c2 ♕b1 55 ♕c8+ ♘f8 56 ♕c4 ♘e6 57 ♗d6 ♖d8 58 a4 ♕b6 59 ♗e7 ♖e8 60 ♕c8 ♕b8 61 ♕×b8 ♖×b8 62 a5 ♘d4 63 ♖c4 ♘f5 64 ♗b4 ♘e3 65 ♖c6 h5 66 a6 ♖a8 67 ♗c5 ♘c4 68 a7 ♔h7 Adjudicated. 1-0

**Ostrich – Prodigy** 1 e4 e6 2 d4 d5 3 e5 c5 4 ♘f3 ♘c6 5 ♗b5 c×d4 6 0-0 ♗c5 7 ♗×c6+ b×c6 8 a3 ♗a6 9 ♖e1 ♕a5 10 ♘bd2 d3 11 c×d3 ♕b6 12 d4 ♗×d4 13 ♘×d4 ♕×d4 14 g3 0-0-0 15 ♕c2 ♗b7 16 ♘f3 ♕g4 17 ♘g5 f6 18 ♘f7 ♘e7 19 ♕c5 ♘f5 20 ♘×h8 ♖×h8 21 e×f6 ♘d4 22 f×g7 ♘f3+ 23 ♔h1 ♖d8 24 g8(♕) ♖×g8 25 ♖g1 ♘×g1 26 ♔×g1 a6 27 ♗e3 e5 28 ♗d2 ♕d4 29 ♕e7 ♕×b2 30 ♕e6+ ♔b8 31 ♕×g8+ ♗c8 32 ♖d1 ♕c2 33 ♖a1 ♕×d2 34 ♕×h7 ♕c3 35 ♖b1+ ♔a8 36 ♕c7 ♕e1+ 37 ♖×e1 ♗f5 38 ♖×e5 d4 39 ♖e8+ ♗c8 40 ♖×c8# 1-0

**Awit – Philidor** 1 b3 e5 2 ♗b2 ♘c6 3 e3 ♘f6 4 ♗b5 d6 5 ♘e2 ♗g4 6 0-0 ♗e7 7 h3 ♗h5 8 f4 ♗×e2 9 ♕×e2 e4 10 d3 0-0 11 d×e4 ♘×e4 12 ♖e1 ♗f6 13 ♗×f6 ♕×f6 14 ♘a3 ♘c3 15 ♕f3 d5 16 ♗d3 ♘b4 17

g4 ♘b×a2 18 g5 ♕e7 19 ♘b5 ♘×b5 20 ♗×b5 ♘b4 21 ♖ec1 a6 22 c3 a×b5 23 c×b4 c6 24 ♖×a8 ♖×a8 25 ♕f2 ♖e8 26 ♖c3 ♕×b4 27 ♕d2 ♕e4 28 ♖d3 ♕f3 20 ♕g2 ♕×g2+ 30 ♔×g2 b4 31 h4 ♖e4 32 ♔f3 f5 33 g×f6 g×f6 34 ♖d2 ♔f7 35 ♖h2 c5 36 ♖d2 ♔e6 37 ♖a2 ♔f5 38 ♖c2 b6 39 ♖c1 d4 40 ♖g1 ♖×e3+ 41 ♔f2 ♖h3 0-1

**Schach 2.5 – Chaturanga** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗×c6 d×c6 5 ♘c3 ♗g4 6 d3 ♗d6 7 h3 ♗×f3 8 ♕×f3 ♘e7 9 ♗e3 ♗b4 10 ♕g3 ♘g6 11 a3 ♗d6 12 ♗g5 f6 Time. 0-1

**Cube 2.1 – Mychess** 1 e4 e5 2 ♘f3 ♘f6 3 ♘×e5 d6 4 ♘f3 ♘×e4 5 ♘c3 ♘×c3 6 d×c3 ♗e7 7 ♗f4 0-0 8 ♗c4 ♘c6 9 0-0 ♗e6 10 ♗×e6 f×e6 11 ♗c1 d5 12 ♖e1 ♕d7 13 ♕e2 ♖f6 14 ♗g5 ♖g6 15 ♗×e7 ♕×e7 16 ♕b5 ♖b8 17 a4 ♕f6 18 ♔f1 ♖g4 19 ♖e3 ♕g6 20 a5 ♖×g2 21 a6 ♘d8 22 ♕d7 ♖g4 23 ♕×c7 ♘c6 24 a×b7 ♕e8 25 ♖ae1 ♖g6 26 b4 ♖d8 27 b5 ♘b8 28 ♘e5 ♖f6 29 ♖g3 g6 30 ♘g4 ♖f5 31 ♘h6+ ♔f8 32 ♘×f5 ♕×b5+ 33 ♔g2 ♕d7 34 ♕×d7 ♖×d7 35 ♘d4 ♖×b7 36 ♘×e6+ ♔g8 37 ♖e5 ♖b2 38 ♖×d5 a6 39 ♖d8+ ♔f7 40 ♘g5+ ♔g7 41 c4 h6 42 ♖b3 ♖×b3 43 c×b3 ♘c6 44 ♖d7+ ♔f6 45 ♖d6+ 1-0

*Round 3, November 10, 1981*

**Belle – Nuchess** 1 e4 e5 2 ♘f3 ♘f6 3 d4 ♘×e4 4 ♗d3 d5 5 ♘×e5 ♗d6 6 0-0 0-0 7 c4 ♗×e5 8 d×e5 ♘c6 9 f4 ♘b4 10 c×d5 ♕×d5 11 ♗×e4 ♕×e4 12 ♘c3 ♕g6 13 ♗e3 ♗e6 14 ♕f3 ♘c2 15 ♖ac1 ♘×e3 16 ♕×e3 c6 17 ♖cd1 f6 18 ♖d6 f×e5 19 ♖f3 e4 20 ♘×e4 ♖ae8 21 ♕d4 ♕f5 22 a3 a5 23 ♖g3 ♖f7 24 ♖d8 ♖×d8 25 ♕×d8+ ♖f8 26 ♕c7 g6 27 ♕e5 ♗c4 28 ♕×f5 ♖×f5 29 ♖c3 b5 30 g3 ♗d5 31 ♖e3 ♖f7 32 ♔f2 ♔g7 33 ♘c5 ♔f6 34 g4 ♔g7 35 ♔g3 ♔h6 36 ♘e6 b4 37 a4 ♖d7 38 f5 g×f5 39 g×f5 ♗h1 40 f6 ♗d5 41 ♔f4 ♗×e6 42 ♖×e6 ♖d2 43 b3 ♔g6 44 ♖×c6 ♖f2+ 45 ♔e4 ♖b2 46 ♖c5 ♔×f6 47 ♖×a5 ♖×b3 48 ♖f5+ ♔g6 49 ♖b5 ♖h3 50 ♔d5 ♖×h2 51 ♖×b4 ♖h5+ 52 ♔c4 ♖f5 53 ♖b6+ ♔f7 54 ♖b5 ♖f4+ 55 ♔b3 ♖f3+ 56 ♔c2 ♔g6 57 a5 h5 58 a6 ♖a3 59 ♖b6+ ♔f5 60 ♔b2 ♖a5 61 ♔c3 h4 62 ♔b4 ♖a1 63 ♖b5+ ♔f4 64 ♖a5 ♖b1+ 65 ♔c5 ½-½

**Chaos – Cray Blitz** 1 d4 ♘f6 2 c4 c5 3 d5 e5 4 ♘c3 ♕a5 5 e3 d6 6 ♗d2 ♕b6 7 ♕c2 ♗e7 8 ♘f3 ♘a6 9 ♘a4 ♕d8 10 ♗d3 0-0 11 0-0 h6 12 b3 ♗g4 13 h3 ♗×f3 14 g×f3 ♘b4 15 ♗×b4 c×b4 16 ♕d2 a5 17 ♔h1 ♔h8 18 ♖g1 ♖c8 19 h4 ♘h5 20 ♗f5 ♖c7 21 ♗g4 g6 22 ♗×h5 g×h5 23 e4 ♔h7 24 ♕e1 ♗×h4 25 ♕e3 ♕c8 26 ♖g2 ♗g5 27 ♕b6 ♕h3+ 28 ♔g1 ♖g8 29 ♖g3 ♗f4 30 ♖×g8 ♔×g8 31 ♖e1 ♗h2+ 32 ♔h1 f6 33 ♕c5 ♗f4+ 34 ♔g1 ♖g7# 0-1

**Duchess – Chess Challenger X** 1 e4 e5 2 ♘f3 ♘c6 3 ♗c4 ♘f6 4 ♕e2 ♗c5 5 d3 0-0 6 0-0 d6 7 c3 ♗g4 8 ♗g5 ♘a5 9 ♗b5 a6 10 ♗a4 b5 11 ♗c2 h6 12 ♗d2 ♘c6 13 b4 ♗b6 14 a4 ♗e6 15 ♗e3 ♗×e3 16 f×e3 ♘g4 17 ♘bd2 ♘e7 18 ♘h4 ♕d7 19 a×b5 a×b5 20 ♗b3 ♗×b3 21 ♘×b3 c6 22 ♘f5 h5 23 ♘a5 ♘×f5 24 ♖×f5 g6 25 ♖f3 d5 26 e×d5 c×d5 27 h3 ♘h6 28 e4 d×e4 29 ♕×e4 ♕c7 30 c4 ♘f5 31 ♔h2 b×c4 32 d×c4 ♖fe8 33 c5 ♘d4 34 ♖f6 ♕d8 35 ♖af1 ♕c7 36 ♘c4 ♘f5 37 ♖1×f5 g×f5 38 ♖×f5 ♖ad8 39 ♖×h5 f6 40 ♘d6 ♖e7 41 ♕g4+ ♖g7 42 ♕e6+ ♖f7 43 ♘×f7 ♕×f7 44 ♖h8+ ♔×h8 45 ♕×f7 ♖g8 46 c6 f5 47 c7 ♖a8 48 b5 f4 49 b6 ♖g8 50 b7 ♖×g2+ 51 ♔×g2 f3+ 52 ♔f2 e4 53 c8(♕)# 1-0

**Philidor – L'Excentrique** 1 e4 c6 2 d4 d5 3 e5 ♗f5 4 ♘e2 e6 5 ♘g3 ♗g6 6 ♗d3 h5 7 0-0 h4 8 ♘e2 ♗×d3 9 ♕×d3 ♘d7 10 ♘d2 ♗e7 11 c3 ♕c7 12 ♘f3 0-0-0 13 ♗f4 b5 14 a4 ♕b7 15 b4 b×a4 16 ♖×a4 f5 17 ♗e3 ♘b6 18 ♖a2 ♘c4 19 ♗f4 ♔d7 20 ♖a4 a6 21 g3 h3 22 ♖a2 ♖h5 23 ♖e1 ♖f8 24 ♗c1 g5 25 ♘d2 ♘×d2 26 ♗×d2 ♖a8 27 ♖ea1 ♕b5 28 ♕×b5 c×b5 29 f4 g4 30 ♖×a6 31 ♖×a6 ♗d8 32 ♖d6+ ♔e7 33 ♗e1 ♗c7 34 ♖a6 ♘h6 35 ♗d2 ♘f7 36 ♖c6 ♔d7 37 ♖a6 ♘d8 38 ♘c1 ♖h7 39 ♘d3 ♔c8 40 ♘c5 ♗b8 41 ♘×e6 ♖h6 42 ♘c7 ♖×a6 43 ♘×a6 ♘e6 44 ♗e3 ♗c7 45 ♔f1 ♔d7 46 ♘×c7 ♘×c7 47 ♔e2 ♘a8 48 ♔d3 ♘b6 49 ♗d2 ♘c4 50 ♗c1 ♔e7 51 ♗d2 ♘×d2 52 ♔×d2 ♔f7 53 ♔e3 ♔e7 54 ♔f2 ♔d7 55 ♔e3 ♔c7 56 ♔d3 ♔d7 57 ♔d2 ♔e7 58 ♔e3 ♔d7 59 ♔d3 ♔e7 60 ♔d2 ♔d7 Adjudicated. 1-0 (61 c4 d×c4 62 d5 ♔d8 63 e6 ♔e7 64 ♔c3 ♔d6 65 ♔d4 ♔e7 66 ♔e5 c3 67 d6+ ♔e8 68 ♔f6 c2 69 d7+ ♔d8 70 ♔f7 c1(♕) 71 e7+ ♔×d7 72 e8(♕)+ ♔c7 73 ♕×b5)

**Cube 2.1 – Ostrich** 1 e4 d5 2 e×d5 ♘f6 3 c4 c6 4 d4 c×d5 5 ♘c3 e6 6 ♘f3 ♗b4 7 ♕a4+ ♘c6 8 ♗g5 0-0 9 c×d5 ♕×d5 10 ♗×f6 ♗×c3+ 11 b×c3 ♕e4+ 12 ♗e2 g×f6 13 ♕c4 ♕g4 14 h3 ♕f4 15 0-0 ♗d7 16 ♖ab1 ♘d8 17 ♕b4 h6 18 ♕e7 ♗c6 19 ♕a3 b6 20 ♖fe1 h5 21 ♖b2 ♘b7 22 g3 ♕f5 23 ♔g2 e5 24 ♕b3 ♘a5 25 ♕b4 ♗e4 26 ♗d1 ♘c6 27 ♕c4 ♘a5 28 ♕b5 ♗c6 29 ♕b4 ♘b7 30 d×e5 f×e5 31 g4 h×g4 32 h×g4 ♕f6 33 ♔g3 a5 34 ♕×b6 ♕f4+ 35 ♔h4 ♔g7 36 ♔h3 ♗×f3 37 ♗×f3 ♕×f3+ 38 ♔h2 ♖h8+ 39 ♔g1 ♕h1# 0-1

**Prodigy – Schach 2.5** 1 d4 d5 2 ♗g5 h6 3 ♗h4 ♘f6 4 e3 g5 5 ♗g3 ♘e4 6 c4 ♘×g3 7 f×g3 d×c4 8 ♗×c4 e6 9 ♘c3 c5 10 ♘f3 c×d4 11 e×d4 ♕b6 12 ♕d2 ♕b4 13 ♗b5+ ♗d7 14 ♗×d7+ ♘×d7 15 ♖c1 ♘b6 16 ♖c2 ♘c4 17 ♕c1 g4 18 ♘e5 ♗g7 19 ♘×g4 ♗×d4 20 ♖f1 ♘×b2 21 ♕d2 ♖d8 22 ♖×b2 ♗×c3 23 ♘f6+ ♔e7 24 ♖×b4 ♗×d2+ 25 ♔e2 ♗×b4 26 ♔e3 ♖d2 27 a3 ♗a5 28 ♘g4 ♖×g2 29 ♖b1 ♗b6+ 30 ♔f3 ♖a2 31 ♖e1 ♖×a3+ 32 ♔g2 h5 33 ♘f2 ♖a2 34 ♖f1 ♖h6 35 h3 a5 36 h4 ♖×f2+ 37 ♖×f2 ♗×f2 38 ♔×f2 a4 39 ♔f3 a3 0-1

**Mychess – Awit** 1 e4 c5 2 ♘f3 d6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 a6 6 ♗e2 e6 7 0-0 ♗e7 8 ♗g5 ♗d7 9 ♘b3 ♗c6 10 ♗f3 0-0 11 a4 ♘bd7 12 ♘d4 ♕b6 13 ♘×c6 ♕×c6 14 e5 d5 15 e×f6 ♘×f6 16 ♖e1 ♕c5 17 ♕d2 ♖ac8 18 ♕d3 ♖fd8 19 ♕e2 ♕b4 20 ♖ab1 d4 21 ♗×f6 ♗×f6 22 ♘e4 ♗h4 23 b3 ♗e7 24 ♖bd1 g6 25 ♕d2 ♕×d2 26 ♘×d2 ♖×c2 27 ♗×b7 ♗b4 28 ♘f3 ♗×e1 29 ♖×e1 a5 30 ♔f1 ♖b2 31 ♖e5 ♖×b3 32 ♗e4 ♖a3 33 ♖×a5 ♖a2 34 ♘e5 ♖a1+ 35 ♔e2 ♖a2+ 36 ♔e1 ♔g7 37 ♖a7 ♖f8 38 ♘c6 ♖c8 39 ♗f3 ♔f6 40 a5 h5 41 ♖d7 ♖a1+ 42 ♔d2 e5 43 ♖d6+ ♔g7 44 ♖d7 ♔f6 45 ♗d5 ♔g5 46 ♖×f7 ♖g8 47 ♖e7 ♖a8 48 ♖×e5+ ♔f6 49 f3 ♖8×a5 50 ♖e6+ ♔g7 51 ♘×a5 ♖×a5 52 ♖d6 Adjudicated. 1-0

**Chaturanga – Bebe** 1 e4 c5 2 ♘f3 d6 3 ♗d3 ♘f6 4 e5 d×e5 5 ♘×e5 ♕d5 6 ♘f3 ♗f5 7 ♘c3 ♕e6+ 8 ♗e2 ♘c6 9 d3 ♘d4 10 ♘×d4 c×d4 11 ♘b5 ♕b6 12 c4 d×c3 13 ♘×c3 0-0-0 14 ♗e3 ♕×b2 15 ♘d5 ♘×d5 16 ♗d2 ♘c7 17 g4 ♗×d3 18 ♗×d3 ♖×d3 19 ♔e2 ♕d4 20 f4 ♕e4+ 21 ♔f2 ♕d4+ 22 ♔e2 ♕e4+ 23 ♔f2 ♕d5 24 ♗e3 ♖×d1 25 ♖h×d1 ♕c4 26 ♗d4 e6 27 ♗e3 ♗c5 28 ♔f3 ♕c3 29 ♖e1 ♗×e3 30 ♔e4 ♗d2 31 ♖e2 ♘e8 32 ♖×d2 ♘f6# 0-1

*Round 4, November 11, 1981*

**Cray Blitz – Belle** 1 e4 e5 2 ♘f3 ♘c6 3 ♗c4 ♘f6 4 ♘g5 d5 5 e×d5 ♘a5 6 ♗b5+ c6 7 d×c6 b×c6 8 ♕f3 ♖b8 9 ♗×c6+ ♘×c6 10 ♕×c6+ ♘d7 11 d3 ♗e7 12 ♘e4 ♗b7 13 ♕a4 ♕c7 14 ♘bc3 ♗c6 15 ♕c4 ♕c8 16 ♘d5 ♗×d5 17 ♕×d5 ♕×c2 18 0-0 f6 19 f4 ♘b6 20 ♕a5 ♕×d3 21 ♕×a7 0-0 22 ♕×e7 ♕×e4 23 ♕e6+ ♔h8 24 f×e5 f×e5 25 ♖×f8+ ♖×f8 26 h3 ♕e1+ 27 ♔h2 h6 28 ♕×b6 ♖f1 29 ♕d8+ ♔h7 30 ♕d3+ e4 31 ♕×f1 ♕×f1 32 a3 e3 33 ♗×e3 ♕×a1 34 ♗d4 h5 35 ♗c3 g5 36 ♗e5 ♕e1 37 ♗c3 ♕f2 38 ♔h1 g4 39 h×g4 h×f4 40 ♔h2 ♕h4+ 41 ♔g1 g3 42 ♔f1 ♔g6 0-1

**Nuchess – Duchess** 1 f4 ♘f6 2 e3 g6 3 b3 ♗g7 4 ♗b2 0-0 5 ♘f3 c5 6 ♗e2 b6 7 0-0 ♗b7 8 ♕e1 e6 9 ♘c3 d5 10 ♕h4 ♘e4 11 ♘g5 h6 12 ♘f3 ♕×h4 13 ♘×h4 ♘×d2 14 ♖fd1 ♘e4 15 ♘a4 ♗×b2 16 ♘×b2 ♘c3 17 ♖e1 ♘×e2+ 18 ♖×e2 d4 19 e×d4 g5 20 f×g5 h×g5 21 ♘f3 g4 22 ♘e5 c×d4 23 ♘×g4 ♔g7 24 ♖ae1 ♘c6 25 ♘d3 ♖h8 26 ♖f1 f5 27 ♘gf2 ♖ae8 28 g4 ♘d8 29 g×f5 e×f5 30 ♖×e8 ♖×e8 31 ♘h1 ♖e2 32 ♖f2 ♖×f2 33 ♘h×f2 ♘e6 34 ♔f1 ♔f6 35 ♘h3 ♗f3 36 ♔e1 ♗g2 37 ♘hf4 ♘×f4 38 ♘×f4 ♗f3 39 ♔d2 ♔e5 40 ♘d3+ ♔e4 41 ♘f2+ ♔d5 42 ♘d3 b5 43 h4 ♔e4 44 ♘c5+ ♔f4 45 a4 b×a4 46 ♘×a4 ♔e5 47 ♘c5 f4 48 ♘d3+ ♔f5 49 ♘b4 a5 50 ♘d3 ♗h5 51 ♔e1 ♗e8 52 ♘c5 ♗g6 53 ♘d3 ♗h5 54 ♘c5 ♗f7 55 ♔d2 ♗d5 56 ♘d3 ♗g8 57 ♘e1 ♔g4 58 ♔d3 f3 59 ♘×f3 ♔×f3 60 ♔×d4 ♔e2 61 ♔c5 ♔d2 62 ♔b5 ♔×c2 63 ♔×a5 ♔×b3 ½-½

**Ostrich – Bebe** 1 e4 c5 2 ♘f3 d6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 g6 6 ♗e2 ♗g7 7 0-0 0-0 8 ♗e3 ♗d7 9 a4 ♘c6 10 ♘b3 ♘b4 11 f4 ♕c7 12 a5 ♘×e4 13 ♘×e4 ♕×c2 14 ♗f3 ♗×b2 15 ♖b1 ♕×d1 16 ♗×d1 ♗g7 17 ♗f3 ♗b5 18 ♖f2 ♗c6 19 ♘g5 ♗×f3 20 ♘×f3 ♘d5 21 ♗d2 ♖ac8 22 ♘c1 ♖c7 23 ♘d3 ♖fc8 24 ♖b3 e6 25 ♗b4 ♖d7 26 ♗d2 ♖c2 27 ♘fe1 ♖a2 28 ♘c1 ♖a4 29 ♘e2 ♖a2 30 ♘c1 ♖a4 31 ♘e2 ♖e7 32 ♘c2 ♖a2 33 ♘ed4 ♖c7 34 g4 e5 35 f×e5 d×e5 36 ♘b4 ♘×b4 37 ♘b5 ♖cc2 38 ♖×b4 ♗f8 39 ♘a3 ♖×a3 40 ♖×b7 ♗c5 41 ♗e1 ♗×f2+ 42 ♗×f2 ♖a1+ 0-1

**Chess Challenger X – Philidor** 1 e4 c5 2 c3 d5 3 e×d5 ♕×d5 4 d4 e6 5 ♘f3 ♘f6 6 ♗e3 ♘e4 7 c4 ♕d8 8 ♗d3 f5 9 ♗×e4 f×e4 10 ♘e5 c×d4 11 ♗×d4 ♕g5 12 0-0 ♘d7 13 ♘×d7 ♗×d7 14 ♕e2 ♗c6 15 ♘c3 e5 16 ♗e3 ♕f5 17 ♖ad1 ♗e7 18 ♕d2 0-0 19 ♘d5 ♗d8 20 ♗c5 ♖f7 21 ♘e3 ♕g5 22 ♔h1 ♗b6 23 ♗×b6 a×b6 24 b3 ♖af8 25 ♔g1 b5 26 c5 ♖f3 27 h4 ♕×h4 28 g×f3 e×f3 29 ♕d6 ♕g5+ 30 ♔h1 ♕h4+ 31 ♔g1 ♕g5+ 32 ♔h2 ♕h4+ 33 ♔g1 ½-½

**L'Excentrique – Chaos** 1 e4 c5 2 ♘f3 ♘c6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 e6 6 ♘db5 ♗b4 7 a3 ♗×c3+ 8 ♘×c3 d5 9 e×d5 e×d5 10 ♗d3 0-0 11 0-0 h6 12 ♗f4 d4 13 ♘b5 ♘d5 14 ♗g3 ♗e6 15 ♖e1 ♕d7 16 h3 ♖ad8 17 ♕f3 ♘de7 18 a4 ♗f5 19 ♗d6 ♗×d3 20 c×d3 a6 21 ♗×e7 ♘×e7 22 ♘a3 ♖fe8 23 ♘c4 ♘d5 23 b3 ♖c8 25 ♖×e8+ ♖×e8 26 a5 ♘c3 27 ♔f1 ♕e7 28 ♕g4 ♕e2+ 29 ♕×e2 ♖×e2 30 ♘d6 ♖e7 31 ♘f5 ♖e2 32 ♘×d4 ♖d2 33 ♘f5 ♖×d3 34 f3 ♘d5 35 ♖a3 ♖c3 36 ♔f2 f6 37 g4 ♘f4 38 h4 ♘d3+ 39 ♔g3 ♘e5 40 ♘d4 ♔f7 41 ♖a2 ♖d3 42 ♖a4 ♖d2 43 ♖b4 ♔g8 44 ♘f5 ♖d7 45 ♖b6 ♘c6 46 b4 g6 47 ♘×h6+ ♔g7 48 g5 f5 49 f4 ♖d3+ 50 ♔f2 ♖d2+ 51 ♔e1 ♖d7 52 b5 a×b5 53 a6 b×a6 54 ♖×c6 ♖a7 Adjudicated. 1-0

**Schach 2.5 – Cube 2.1** 1 e4 c5 2 ♘f3 ♘f6 3 e5 ♘d5 4 ♘c3 ♘c7 5 ♗d3 d6 6 e×d6 ♕×d6 7 ♘e4 ♕d5 8 0-0 ♗g4 9 h3 ♗×f3 10 ♕×f3 ♘d7 11 c4 ♕d4 12 ♕e3 ♕×e3 13 d×e3 e6 14 ♗e2 f5 15 ♘g3 ♗e7 16 ♗d2 ♗f6 17 ♖ad1 0-0-0 18 ♖b1 ♘b8 19 ♖fd1 g6 20 b3 ♘c6 21 ♖bc1 ♖he8 22 ♗c3 ♖×d1+ 23 ♗×d1 ♗×c3 24 ♖×c3 ♖d8 25 ♗e2 h5 26 a4 h4 27 ♘f1 e5 28 f3 ♘e6 29 ♔f2 f4 30 ♘h2 f×e3+ 31 ♔×e3 ♘f4 32 ♗f1 ♖d1 33 ♔e4 ♖e1+ 34 ♖e3 ♖c1 35 ♗d3 ♘×g2 36 ♖e2 ♘f4 37 ♖e3 ♖d1 38 ♘g4 ♘×h3 39 f4 ♘×f4 40 ♘×e5 ♘×e5 41 ♗c2 ♘×c4 42 ♗×d1 ♘×e3 43 ♔×e3 ♘d5+ 44 ♔d3 ♘f6 45 ♔c4 b6 46 a5 b×a5 47 ♔×c5 h3 48 ♔b5 h2 49 ♗f3 ♘h5 50 ♔×a5 ♘g3 51 ♔b5 h1(♕) 52 ♗×h1 ♘×h1 53 ♔c5 ♘f2 54 ♔d4 g5 55 ♔e5 ♔d7 56 ♔f5 a5 57 ♔×g5 ♘e4+ 58 ♔f5 ♘c5 59 ♔e5 ♘×b3 60 ♔d5 a4 61 ♔c4 ♔d6 62 ♔b4 ♘c5 63 ♔a3 ♔c6 64 ♔b4 ♔d5 65 ♔a3 ♔c4 66 ♔b2 ♘d3+ 67 ♔a3 ♔b5 68 ♔a2 ♔b4 69 ♔b1 ♔b3 70 ♔a1 a3 71 ♔b1 a2+ 72 ♔a1 ♘b2 ½-½

**Awit – Prodigy** 1 b3 d5 2 ♗b2 ♗g4 3 ♘f3 c5 4 e3 e6 5 h3 ♗×f3 6 ♕×f3 ♘c6 7 ♘a3 e5 8 ♗b5 e4 9 ♕f5 ♘e7 10 ♕g4 h5 11 ♕g5 f6 12 ♕f4 ♘g6 13 ♕g3 ♘e5 14 ♗×e5 f×e5 15 ♕×e5+ ♔f7 16 ♕f5+ ♔g8 17 ♗×c6 b×c6 18 ♕e6+ ♔h7 19 ♕×c6 ♖c8 20 ♕a6 ♕d7 21 c4 ♖c6 22 ♕a4 d4 23 0-0 d×e3 24 d×e3 ♗d6 25 ♖ad1 h4 26 ♖d5 ♔h6 27 ♘b5 ♖b8 28 ♘×d6 ♕c7 29 ♘×e4 ♖b7 30 ♖fd1 ♖bb6 31 ♕a5 ♕f7 32 ♘×c5 ♖g6 33 ♘a4 ♕f3 34 g4 h×g3 35 ♖h5+ ♕×h5 36 ♕×h5+ ♔×h5 37 ♘×b6 g×f2+ 38 ♔×f2 a×b6 39 ♖d4 ♖f6+ 40 ♔g3 ♖c6 41 ♔f3 ♖h6 42 a4 ♔g5 43 h4+ ♔h5 44 ♖e4 ♖f6+ 45 ♔g3 ♖g6+ 46 ♔h3 ♖d6 47 c5 b×c5 48 ♖e5+ ♔h6 49 ♖×c5 ♖d3 50 ♖b5 ♖×e3+ 51 ♔g4 ♖e4+ 52 ♔g3 ♖e3+ 53 ♔f4 ♖d3 54 b4 ♖d4+ 55 ♔g3 ♖d3+ 56 ♔g4 ♖d4+ 57 ♔h3 ♖d3+ 58 ♔h2 1-0

**Mychess – Chaturanga** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗a4 d6 5 c3 ♘f6 6 ♗×c6+ b×c6 7 d4 e×d4 8 ♕×d4 c5 9 ♕d3 ♗e6 10 c4 ♗e7 11 ♘c3 ♕b8 12 0-0 ♘g4 13 ♘d5 ♘e5 14 ♘×e5 d×e5 15 ♘×e7 ♔×e7 16 ♕c3 c6 17 ♗e3 ♕d6 18 ♖ad1 ♕c7 19 ♗×c5+ ♔f6 20 ♗d6 ♕b6 21 ♗×e5+ ♔e7 22 ♗×g7 ♖hc8 23 c5 ♕b5 24 ♕f6+ ♔e8 25 ♕e5 ♕c4 26 ♕d6 ♕×f1+ 27 ♔×f1 ♗c4+ 28 ♔g1 ♗d5 29 e×d5 f5 30 ♕e6+ ♔d8 31 d×c6+ ♔c7 32 ♗e5# 1-0 This game was played with colours opposite to what was paired.

## ACM 1982 — Dallas

| | rate | perf | 1 | 2 | 3 | 4 | tot |
|---|---|---|---|---|---|---|---|
| 1 Belle | 2150 | 2120 | 9= | 4+ | 3+ | 2= | 3 |
| 2 Cray Blitz | 0 | 2053 | 8+ | 5= | 7+ | 1= | 3 |
| 3 Nuchess | 0 | 2040 | 10+ | 7+ | 1− | 5+ | 3 |
| 4 Chaos | 1800 | 1888 | 14+ | 1− | 8+ | 9+ | 3 |
| 5 Bebe | 1850 | 1907 | 11+ | 2= | 6+ | 3− | 2½ |
| 6 Advance 2.4 | 0 | 1649 | 12= | 9+ | 5− | 10+ | 2½ |
| 7 Fidelity X | 0 | 1636 | 13+ | 3− | 2− | 11+ | 2 |
| 8 Savant X | 0 | 1579 | 2− | 12+ | 4− | 13+ | 2 |
| 9 Ostrich | 0 | 1595 | 1= | 6− | 12+ | 4− | 1½ |
| 10 Schach 2.6 | 0 | 1356 | 3− | 13+ | 11= | 6− | 1½ |
| 11 Philidor | 0 | 1322 | 5− | 14+ | 10= | 7− | 1½ |
| 12 Sfinks X | 0 | 1364 | 6= | 8− | 9− | 14+ | 1½ |
| 13 Pion | 0 | 1126 | 7− | 10− | 14+ | 8− | 1 |
| 14 Chaturanga 2.0 | 1000 | 814 | 4− | 11− | 13− | 12− | 0 |

*Round 1, October 24, 1982*

**Belle — Ostrich** 1 e4 d5 2 e×d5 ♘f6 3 d4 ♘×d5 4 ♘f3 ♗g4 5 ♗e2 ♘c6 6 c4 ♘f6 7 d5 ♗×f3 8 ♗×f3 ♘e5 9 ♗g5 e6 10 0-0 ♗c5 11 ♖e1 ♘×f3+ 12 ♕×f3 ♗b4 13 ♖e4 ♕e7 14 d×e6 0-0-0 15 ♕e2 h6 16 ♗×f6 ♕×f6 17 g3 ♖he8 18 e×f7 ♖×e4 19 ♕×e4 ♕×b2 20 c5 ♗×c5 21 ♕f5+ ♖d7 22 ♕×c5 ♖×f7 23 ♕×a7 ♕×a1 24 ♕a8+ ♔d7 25 ♕×b7 ♕×a2 26 ♕b5+ c6 27 ♕b7+ ♔d6 28 ♕b4+ ♔c7 29 ♘d2 ♕a7 30 ♘e4 ♕a1+ 31 ♔g2 ♕e5 32 ♕a4 ♖e7 33 ♕a7+ ♔d8 34 ♕a8+ ♔d7 35 ♕b7+ ♕c7 36 ♕b3 ♕e5 37 ♕b7+ ♕c7 38 ♕b3 ♕e5 39 ♕b7+ ½-½

**Schach 2.6 — Nuchess** 1 e4 g6 2 d4 ♗g7 3 c4 d6 4 ♘c3 ♘f6 5 ♗e2 0-0 6 ♘f3 ♘c6 7 0-0 ♗g4 8 ♗e3 ♕d7 9 ♕b3 b6 10 h3 ♗×f3 11 ♗×f3 e5 12 ♘e2 e×d4 13 ♗×d4 ♖ae8 14 ♗×f6 ♗×f6 15 ♕b5 a6 16 ♕a4 ♗×b2 17 ♖ab1 ♗g7 18 ♗g4 f5 19 e×f5 g×f5 20 ♗h5 ♖d8 21 ♕×a6 ♖a8 22 ♕b5 ♖×a2 23 c5 ♖a5 24 ♕b3+ ♔h8 25 c×b6 ♖b8 26 b7 ♘d8 27 ♗f3 c6 28 ♕b6 ♖c5 29 ♖fd1 ♖×b7 30 ♕a6 ♖×b1 31 ♖×b1 d5 32 ♖b8 ♖c2 33 ♕a8 ♗f6 34 ♕a4 ♖d2 35 ♘f4 ♕d6 36 ♕b4 ♕×b4 37 ♖×b4 ♗e5 38 ♖a4 ♘f7 39 ♘e6 ♖b2 40 ♖a8+ ♗b8 41 g3 ♖b4 42 ♔g2 h6 43 ♗e2 ♘e5 44 f4 ♘d7 45 ♗d3 ♖b2+ 46 ♔h1 ♘f6 47 ♘d4 c5 48 ♘c6 c4 49 ♗×f5 ♘h5 50 ♔g1 ♘×g3 51 ♗g4 h5 52 ♗f3 c3 53 ♘×b8 c2 54 ♘a6+ ♔h7 55 ♖a7+ ♔g6 56 ♖c7 ♘e4 57 ♘b4 ♖b1+ 58 ♔g2 c1♕ 59 ♖×c1 ♖×c1 60 ♘×d5 ♖c2+ 61 ♔h1 ♘g3+ 62 ♔g1 ♖d2 63 ♘e7+ ♔f6 64 ♘d5+ ♔f5 65 h4 ♖a2 66 ♗g2 ♘e2+ 67 ♔f1 ♘×f4 68 ♗e4+ ♔g4 69 ♘c3 ♖b2 70 ♘d1 ♖d2 71 ♘c3 ♔×h4 72 ♘d5 0-1

**Bebe — Philidor** 1 e4 c5 2 ♘f3 ♘c6 3 d4 c×d4 4 ♘×d4 g6 5 ♘×c6 d×c6 6 ♕×d8+ ♔×d8 7 ♘c3 ♗g7 8 ♗d2 b5 9 0-0-0 ♔c7 10 ♗e2 b4 11 ♘a4 ♖b8 12 ♗c4 ♘h6 13 h3 ♗d7 14 g4 ♖hf8 15 ♘c5 ♗c8 16 ♘d3 ♖d8 17 ♗f4+ e5 18 ♘×e5 ♖×d1+ g5 20 ♗g3 f5 21 g×f5 ♖a8 22 ♘×c6+ ♔b6 23 ♘b8 a5 24 ♖d5 ♖×b8 25 ♗×b8 g4 26 f6 ♗×f6 27 ♖d6+ ♔c5 28 ♖×f6 ♔×c4 29 ♖×h6 ♔d4 30 e5 g×h3 31 e6 ♔c4 32 e7 ♗d7 33 ♖×h7 ♔b5 34 ♖×h3 ♗c6 35 ♖h5+ ♔c4 36 ♗a7 b3 37 a×b3+ ♔b4 38 ♗e3 ♗d5 39 e8♕ ♗×b3 40 ♕b5# 1-0

**Savant X — Cray Blitz** 1 e4 e5 2 ♘f3 ♘f6 3 ♘×e5 d6 4 ♘f3 ♘×e4 5 d4 d5 6 ♗d3 ♗e7 7 0-0 ♘c6 8 c4 ♘f6 9 ♘c3 d×c4 10 ♗×c4 0-0 11 ♗g5 ♗g4 12 ♗e3 ♖b8 13 d5 ♗×f3 14 ♕×f3 ♘e5 15 ♕e2 ♘×c4 16 ♕×c4 a6 17 a4 ♘e8 18 ♖fe1 ♘d6 19 ♕f4 ♖e8 20 ♖ac1 b5 21 ♖e2 f5 22 ♗c5 b4 23 ♗×b4 ♗g5 24 ♗×d6 ♗×f4 25 ♖×e8+ ♕×e8 26 ♗×f4 ♖×b2 27 ♗×c7 ♖c2 28 ♖a1 ♖×c3 29 d6 ♕×a4 30 ♖b1 ♖d3 31 ♖b8+ ♔f7 32 g3 ♖d1+ 33 ♔g2 ♕c4 34 d7 ♕f1+ 35 ♔f3 ♕h1+ 36 ♔f4 ♕e4+ 37 ♔g5 ♕g4# 0-1

**Fidelity X — Pion** 1 e4 d6 2 d4 ♘f6 3 ♘c3 g6 4 ♘f3 ♗g7 5 ♗e2 0-0 6 0-0 c6 7 a4 ♕c7 8 h3 ♘bd7 9 ♗e3 e5 10 d×e5 d×e5 11 ♕d2 ♘h5 12 ♖fd1 ♘f4 13 ♗f1 ♖e8 14 a5 ♗f8 15 ♗c4 ♘e6 16 ♘h2 ♘f6 17 a6 b5 18 ♗d3 ♗d7 19 ♗h6 b4 20 ♘e2 ♗e7 21 ♗c4 ♖ad8 22 ♕e3 ♘c5 23 ♘g3 ♗c8 24 ♖×d8 ♖×d8 25 ♕e2 ♕d6 26 c3 b×c3 27 b×c3 ♘e8 28 ♕f3 ♗e6 29 ♗e2 ♕c7 30 ♘g4 ♗×g4 31 h×g4 ♘d6 32 ♖a2 ♕b6 33 ♖a1 ♕c7 34 ♗f1 ♘b5 35 ♗c4 ♘d6 36 ♗e2 ♘e8 37 ♘f1 ♘f6 38 ♘d2 ♖d7 39 ♗c4 ♖d8 40 g5 ♗f8 41 ♗×f8 ♔×f8 42 ♕×f6 ♖×d2 43 ♖b1 ♖d8 44 ♕h8+ ♔e7 45 ♕g7 ♖f8 46 f3 ♔e8 47 ♕×h7 ♕a5 48 ♖b8+ ♔e7 49 ♖×f8 ♕a1+ 50 ♔h2 ♔d6 51 ♖×f7 ♘×a6 52 ♖×a7 ♔c5 53 ♗×a6 ♕×c3 54 ♕e7+ ♔d4 55 ♖d7+ ♔e3 56 ♖d3+ ♕×d3 57 ♗×d3 1-0

**Chaturanga 2.0 — Chaos** 1 e4 c5 2 ♘f3 ♘c6 3 ♘c3 e6 4 ♘g1 ♘f6 5 ♗b5 ♗e7 6 ♕f3 ♘d4 7 ♕d3 a6 8 ♗a4 b5 9 ♗×b5 a×b5 10 e5 ♘g4 11 h3 ♘×e5 12 ♕e4 d5 13 ♕×e5 ♗f6 14 ♕f4 ♘×c2+ 15 ♔d1 ♘×a1 16 d4 c×d4 17 ♘×b5 ♖×a2 18 ♘d6+ ♔e7 19 ♘f3 ♕×d6 20 ♕d2 ♘b3 21 ♕c2 ♕b8 22 ♔e1 ♖a1 23 ♔d1 ♕f4 24 ♘e5 ♖×c1+ 25 ♕×c1 ♕×c1+ 26 ♔e2 ♕c2+ 27 ♔f3 ♕e4+ 0-1

**Advance 2.4 — Sfinks X** 1 e4 c5 2 ♘f3 e6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 ♘c6 6 ♗e2 ♗e7 7 0-0 d6 8 ♘×c6 b×c6 9 ♗e3 e5 10 ♕d3 0-0 11 h3 a5 12 ♖ad1 ♖b8 13 ♗c1 ♗e6 14 ♕g3 ♔h8 15 ♕e3 ♕b6 16 ♕×b6 ♖×b6 17 ♗f3 ♗c4 18 ♖fe1 ♖b7 19 ♗e2 ♗e6 20 ♗a6 ♖b6 21 ♗d3 ♖fb8 22 a3 ♖6b7 23 ♘a4 ♘d7 24 ♗d2 ♖a8 25 ♗c3 f6 26 ♖e3 d5 27 e×d5 c×d5 28 ♖ee1 ♗b4 29 ♗d2 ♗×d2 30 ♖×d2 ♖ab8 31 c3 ♔g8 32 ♗c2 e4 33 c4 ♘e5 34 c×d5 ♗f5 35 ♗×e4 ♘c4 36 ♖de2 ♗×e4 37 ♖×e4 ♘×b2 38 ♘×b2 ♖×b2 39 d6 ♖d2 40 ♖e8+ ♖×e8 41 ♖×e8+ ♔f7 42 ♖e7+ ♔f8 43 ♖a7 ♖×d6 44 ♖×a5 ♖d1+ 45 ♔h2 ♖d2 46 ♔g3 ♖d3+ 47 ♔f4 ♖d7 48 ♖a8+ ♔f7 49 a4 ♔e6 50 a5 ♔d5 51 a6 ♔c5 52 ♔g4 ♔b6 53 ♔h5 ♖d2 54 ♖g8 g5 55 f3 ♖×g2 56 ♖f8 ♔×a6 57 ♖×f6+ ♔b5 58 ♖f5+ ♔c4 59 ♖×g5 ♖f2 60 ♖f5 ♔d4 61 ♖f7 ♔e3 62 ♖×h7 ♖×f3 ½-½

*Round 2, October 24, 1982*

**Nuchess — Fidelity X** 1 g3 e5 2 ♗g2 d5 3 d4 e×d4 4 ♕×d4 ♘f6 5 ♗g5 ♗e7 6 ♕a4+ ♘c6 7 ♘c3 h6 8 ♗e3 ♗b4 9 ♕b3 0-0 10 0-0-0 ♗×c3 11 ♕×c3 ♗f5 12 ♕b3 ♘a5 13 ♕b4 b6 14 ♗h3 c5 15 ♕h4 ♕d7 16 ♗×f5 ♕×f5 17 ♕f4 ♕e6 18 b3 d4 19 ♗d2 ♘c6 20 ♔b1 ♘d5 21 ♕f3 ♘db4 22 ♖c1 a5 23 a3 ♘d5 24 a4 ♘db4 25 h4 ♘e5 26 ♕b7 ♖ab8 27 ♕g2 d3 28 e×d3

♘b×d3 29 ♖d1 ♘b4 30 ♘e2 ♖bd8 31 ♗×b4 a×b4 32 ♘f4 ♕g4 33 ♘d5 ♖fe8 34 f4 ♘g6 35 ♕f2 ♖d7 36 ♖d2 ♘f8 37 ♘×b6 ♖×d2 38 ♕×d2 ♕×g3 39 h5 ♘e6 40 ♘d5 ♖d8 41 f5 ♕f3 42 ♘e7+ ♔f8 43 ♕e1 ♔×e7 44 f×e6 f×e6 45 ♖g1 ♕f7 46 ♖g6 ♖d5 47 ♕g3 ♔f8 48 ♕b8+ ♕e8 49 ♕f4+ ♖f5 50 ♕e4 ♕f7 51 ♖×e6 ♖f1+ 52 ♔a2 ♕d7 53 ♖e5 ♕d6 54 ♖e8+ ♔f7 55 ♖a8 ♖c1 56 ♕e8+ ♔f6 57 ♕g6+ ♔e5 58 ♖e8+ 1-0

**Cray Blitz – Bebe** 1 e4 c5 2 ♘f3 d6 3 d4 c×d4 4 ♘×d4 ♘f6 5 f3 e5 6 ♗b5+ ♗d7 7 ♗×d7+ ♘b×d7 8 ♘f5 d5 9 e×d5 ♕a5+ 10 ♘c3 ♘b6 11 ♘e3 ♘b×d5 12 ♘e×d5 0-0-0 13 ♕e2 ♖×d5 14 ♗d2 ♖d7 15 ♘e4 ♕a4 16 ♗c3 ♘×e4 17 ♕×e4 ♕a6 18 f4 e×f4 19 ♕e8+ ♖d8 20 ♕×f7 ♕b5 21 ♕e6+ ♔b8 22 ♕e5+ ♕×e5+ 23 ♗×e5+ ♗d6 24 ♗×d6+ ♖×d6 25 0-0 ♖f8 26 ♖ae1 g5 27 ♖e7 ♖d2 28 ♖f2 ♖d1+ 29 ♖f1 ♖d2 30 ♖f2 ♖d1+ 31 ♖f1 ♖d2 ½-½

**Chaos – Belle** 1 d4 d5 2 c4 e6 3 ♘c3 c6 4 ♘f3 d×c4 5 a4 ♗b4 6 e3 b5 7 ♗d2 a5 8 a×b5 ♗×c3 9 b×c3 c×b5 10 ♕b1 ♗a6 11 ♗e2 ♘c6 12 0-0 a4 13 ♕e4 ♘ge7 14 ♘g5 h6 15 ♘f3 0-0 16 ♖fb1 ♕d6 17 ♕g4 f5 18 ♕h5 e5 19 d×e5 ♘×e5 20 ♘d4 ♘5c6 21 ♗f3 ♕d7 22 ♖a2 ♖ab8 23 ♖ab2 ♖f6 24 ♕h3 ♖c8 25 ♕h4 ♘×d4 26 e×d4 ♖e6 27 h3 ♘d5 28 g4 ♖ce8 29 g×f5 ♖f6 30 ♗e4 ♘e7 31 ♕g4 ♘×f5 32 f3 ♖e7 33 ♔h1 ♕c8 34 h4 ♔h8 35 ♗f4 ♘×d4 36 ♕g3 ♘×f3 37 ♗×f3 ♕f5 38 ♗×h6 ♕×f3+ 39 ♕×f3 ♖×f3 40 ♗g5 ♖ef7 41 ♖c1 ♖h3+ 42 ♔g2 ♖ff3 43 ♗d2 a3 44 ♖b4 a2 45 ♖a1 ♖fg3+ 46 ♔f1 ♖h1+ 47 ♔f2 ♖×a1 48 ♔×g3 ♖g1+ 49 ♔f2 a1♕ 0-1

**Ostrich – Advance 2.4** 1 e4 c5 2 ♘f3 d6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 a6 6 ♗g5 e6 7 ♗e2 ♗e7 8 ♗e3 ♘bd7 9 0-0 ♘c5 10 f3 ♕c7 11 a4 0-0 12 ♕e1 e5 13 ♘b3 ♘×b3 14 c×b3 ♗e6 15 b4 d5 16 e×d5 ♘×d5 17 ♘×d5 ♗×d5 18 b5 ♗c5 19 ♗×c5 ♕×c5+ 20 ♔h1 a5 21 ♕g3 ♕d4 22 ♖ad1 ♕×b2 23 ♕f2 ♖ad8 24 ♖d3 ♗b3 25 ♕g1 ♖×d3 26 ♗×d3 ♗×a4 27 ♕c5 ♖d8 28 ♗c4 ♕b4 29 ♕×b4 a×b4 30 h3 ♗c2 31 g3 b3 32 ♔g2 ♖d4 33 ♖a1 ♔f8 34 ♖a8+ ♔e7 35 ♗×b3 ♗×b3 36 ♖h8 h6 37 ♖c8 ♖d2+ 38 ♔f1 ♗d5 39 f4 ♖d3 40 f×e5 ♖×g3 41 h4 ♖g4 42 ♖g8 ♗c4+ 43 ♔e1 ♗×b5 44 ♖b8 ♗c6 45 ♖g8 0-1

**Sfinks – Savant X** 1 d4 d5 2 c4 e5 3 c×d5 ♕×d5 4 e3 e×d4 5 ♘c3 ♗b4 6 ♕×d4 ♕×d4 7 e×d4 ♘f6 8 ♗f4 ♘d5 9 ♗d2 ♗e6 10 ♘×d5 ♗×d2+ 11 ♔×d2 ♗×d5 12 ♘e2 0-0 13 ♘f4 ♖d8 14 ♔c3 ♘c6 15 ♘×d5 ♖×d5 16 g3 ♖×d4 17 ♗d3 ♖ad8 18 ♖ad1 ♖4d6 19 a3 a5 20 ♖he1 ♖h6 21 ♗a6 ♖b8 22 ♗×b7 ♔f8 23 ♗a6 ♖×h2 24 ♖d2 h5 25 b3 g6 26 ♔c4 ♖d8 27 ♖de2 ♖d4+ 28 ♔b5 ♖d6 29 ♖e8+ ♔g7 30 ♖f1 ♘d4+ 31 ♔×a5 ♘×b3+ 32 ♔b4 ♘d2 33 ♖d1 ♖×f2 34 ♗e2 ♘e4 35 ♖×e4 ♖×e2 36 ♖×e2 ♖×d1 37 ♖c2 ♖d4+ 38 ♔b5 ♖d5+ 39 ♔c6 ♖a5 40 ♖c3 ♖a7 41 ♔b5 f5 42 a4 ♔f8 43 a5 ♔f7 44 a6 ♔f6 45 ♖c6+ ♔f7 46 ♔c5 ♔g7 47 ♔d4 ♔h6 48 ♔e5 ♔g5 49 ♔d5 ♔h6 50 ♔e6 ♔g7 51 ♔d7 h4 52 g×h4 f4 53 ♔c8 f3 54 ♖×c7+ ♖×c7+ 55 ♔×c7 f2 56 a7 f1♕ 57 a8♕ ♕f4+ 58 ♔d7 ♕×h4 59 ♕a7 g5 60 ♕a1+ ♔g6 61 ♕b1+ ♔h5 62 ♕h7+ ♔g4 63 ♕e4+ ♔g3 64 ♕e3+ ♔g2 65 ♕e2+ ♔h3 66 ♕f3+ ♕g3 67 ♕e4 g4 68 ♕h1+ ♕h2 69 ♕f1+ ♕g2 70 ♕c4 ♕d2+ 71 ♔e6 g3 72 ♕b3 ♕f4 73 ♕d3 ♕h6+ 74 ♔d5 ♔h2 75 ♕c2+ g2 76 ♕f2 ♕h5+ 77 ♔d4 ♔h1 78 ♕f8 0-1

**Pion – Schach 2.6** 1 ♘f3 d5 2 g3 e6 3 ♗g2 c5 4 0-0 ♘c6 5 d3 ♗d6 6 e4 ♘ge7 7 ♘bd2 0-0 8 ♖e1 ♕c7 9 c3 ♗d7 10 ♕e2 f6 11 d4 c×d4 12 ♘×d4 ♘×d4 13 c×d4 ♕c2 14 e×d5 ♘×d5 15 ♕h5 f5 16 ♕e2 ♖ac8 17 ♘f3 ♗b4 18 ♕×c2 ♖×c2 19 ♖d1 ♗b5 20 ♘e1 ♖e2 21 ♘d3 ♗×d3 22 ♗×d5 ♗c2 23 ♗×e6+ ♖×e6 24 ♗d2 ♗×d1 25 ♗×b4 ♗h5 26 d5 ♖a6 27 ♗×f8 ♔×f8 28 b4 ♖d6 29 ♖c1 ♖×d5 30 ♖c8+ ♔f7 31 ♖c7+ ♔f6 32 g4 ♗×g4 33 a4 ♗f3 34 ♖c1 ♖d4 35 b5 ♖×a4 36 h3 ♖a5 37 ♖b1 b6 38 ♔f1 ♗e4 39 ♖b3 ♖×b5 40 ♖×b5 ♗d3+ 41 ♔e1 ♗×b5 42 f4 g5 43 f×g5+ ♔×g5 44 ♔f2 ♔f4 45 ♔e1 0-1

**Philidor – Chaturanga 2.0** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗×c6 d×c6 5 0-0 ♕×d2 6 ♕×d2 a5 7 ♘×e5 a4 8 ♘c3 b6 9 ♘×c6 ♗d7 10 ♕d5 ♗×c6 11 ♕×c6+ ♔e7 12 ♘d5+ ♔d8 13 ♕×a8+ ♔d7 14 ♕×f8 1-0

*Round 3, October 25, 1982*

**Belle – Nuchess** 1 e4 e5 2 c3 ♘f6 3 d4 ♘×e4 4 d×e5 ♗c5 5 ♕g4 ♘×f2 6 ♕×g7 ♖f8 7 ♗g5 ♕×g5 8 ♕×g5 ♘×h1 9 ♘f3 ♘c6 10 ♗d3 h6 11 ♕×h6 ♖g8 12 g3 ♘f2 13 ♕h7 ♘×d3+ 14 ♕×d3 ♖h8 15 ♘bd2 ♖h5 16 0-0-0 ♘×e5 17 ♖e1 d6 1-0

**Bebe – Advance 2.4** 1 e4 c5 2 ♘f3 d6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 a6 6 ♗e2 e5 7 ♘f3 ♗e7 8 0-0 0-0 9 ♗g5 ♘bd7 10 ♕c1 h6 11 ♗×f6 ♘×f6 12 ♖d1 ♗d7 13 ♘d2 ♗c6 14 ♘c4 ♘×e4 15 ♘×e4 ♗×e4 16 ♘×e5 ♕c7 17 ♘d3 ♖ae8 18 ♕d2 ♗g5 19 f4 ♕b6+ 20 ♘f2 ♗h4 21 g3 ♗f6 22 c3 ♗c6 23 ♖ac1 d5 24 ♗h5 ♖e7 25 ♖c2 ♖fe8 26 ♗f3 d4 27 ♗×c6 d×c3 28 b×c3 b×c6 29 c4 ♕c5 30 ♔g2 ♕f5 31 g4 ♕c8 32 ♕a5 ♕b7 33 ♕c5 ♖e2 34 ♖dd2 ♖e1 35 ♘d3 ♖1e4 36 ♕f5 ♕c8 37 ♕×c8 ♖×c8 38 ♔f3 ♖ce8 39 ♖e2 ♖×e2 40 ♖×e2 ♖×e2 41 ♔×e2 a5 42 a4 ♔f8 43 ♔e3 ♔e8 44 ♘c5 ♗d8 45 ♔f3 ♗e7 46 ♘b3 ♗b4 47 h4 g6 48 ♔e4 ♔d7 49 c5 ♗e1 50 h5 g×h5 51 g×h5 ♗b4 52 ♔e5 ♔e7 53 ♔f5 ♗c3 54 ♘c1 ♔e8 55 ♘d3 ♔f8 56 ♘e5 ♗×e5 57 ♔×e5 ♔e7 58 ♔f5 ♔e8 59 ♔f6 ♔f8 60 f5 ♔e8 61 ♔g7 ♔e7 62 ♔×h6 ♔f8 63 ♔h7 f6 64 ♔g6 ♔g8 65 ♔×f6 ♔h7 66 ♔e6 ♔h6 67 f6 ♔h7 68 ♔d7 ♔g8 69 ♔e7 ♔h8 70 h6 ♔h7 71 f7 ♔×h6 72 ♔d8 ♔g7 73 ♔e8 ♔f6 74 f8♕+ 1-0

**Fidelity X – Cray Blitz** 1 e4 e5 2 ♘c3 ♘c6 3 ♗c4 ♗c5 4 ♕g4 g6 5 ♕f3 ♘f6 6 ♘ge2 d6 7 d3 ♗g4 8 ♕g3 ♗×e2 9 ♘×e2 ♘a5 10 ♗b3 0-0 11 ♗h6 ♘×b3 12 a×b3 ♖e8 13 0-0 d5 14 ♕f3 ♖e6 15 ♘c3 c6 16 e×d5 ♘×d5 17 ♘e4 ♗e7 18 c4 ♘b4 19 ♖fd1 f5 20 ♕h3 ♕d7 21 ♘c3 ♖d6 22 ♗e3 ♖×d3 23 ♖×d3 ♕×d3 24 ♖d1 ♕c2 25 ♖d2 ♕×b3 26 ♗h6 ♕×c4 27 ♖d7 ♕a6 28 ♕e3 ♗f6 29 ♘d1 ♘d5 30 ♕b3 ♖b8 31 ♘e3 f4 32 ♘f1 ♔h8 33 ♕h3 ♖e8 34 ♗×f4 h5 35 ♗d2 ♖e7 36 ♖×e7 ♘×e7 37 ♕e6 c5 38 ♕×a6 b×a6 39 ♗e3 c4 40 ♘d2 ♘d5 41 ♗×a7 c3 42 b×c3 ♘×c3 43 ♗b6 ♔g7 44 ♗a5 ♘d5 45 ♔f1 ♗e7 46 ♘c4 ♔f6 47 ♔e2 ♗c5 48 f3 ♘f4+ 49 ♔f1 ♔e6 50 g3 ♘d5 51 ♗d2 ♘e7 52 ♔e2 ♘c6 53 ♔d3 ♔d5 54

h3 ♘d4 55 ♘e3+ ♔e6 56 f4 e×f4 57 g×f4 ♘f3 58 ♗c3 ♘g1 59 h4 ♘f3 60 ♘g2 0-1

**Schach 2.6 – Philidor** 1 e4 c5 2 ♘f3 d6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 a6 6 ♗g5 ♘bd7 7 ♗e2 e5 8 ♘f5 ♕b6 9 ♕c1 ♕c6 10 ♗f3 g6 11 ♘e3 ♗e7 12 0-0 0-0 13 ♘g4 ♖e8 14 ♘h6+ ♔f8 15 h3 b5 16 a4 b4 17 ♘d5 ♘×d5 18 e×d5 ♕c4 19 ♗×e7+ ♔×e7 20 b3 ♕c3 21 ♖d1 ♘c5 22 ♘g4 e4 23 ♗e2 ♗×g4 24 ♗×g4 f5 25 ♗e2 ♔d7 26 ♖a2 e3 27 f×e3 ♖×e3 28 ♗c4 ♖c8 29 ♔f2 ♘e4+ 30 ♔g1 ♘c5 31 ♕d2 ♕×d2 32 ♖×d2 ♘e4 33 ♖d4 ♖e1+ 34 ♔h2 ♘c3 35 ♖b2 ♖a8 36 ♗d3 a5 37 ♗b5+ ♘×b5 38 a×b5 ♖b8 39 c4 b×c3 40 ♖c2 ♖e3 41 ♖c4 ♖×b5 42 ♖2×c3 ♖×c3 43 ♖×c3 ♖×d5 44 ♔g3 ♖d4 45 ♔f2 ♖d2+ 46 ♔f3 h6 47 g4 f×g4+ 48 h×g4 ♔e6 49 ♔e3 ♖a2 50 ♔f3 ♔f6 51 ♖d3 ♔e5 52 ♖e3+ ♔d5 53 ♖d3+ ♔e6 54 ♖c3 d5 55 ♖c6+ ♔e7 56 ♖c7+ ♔f6 57 ♖c6+ ♔g7 58 ♔f4 ♖a1 59 ♔e5 ½-½

**Savant X – Chaos** 1 e4 c5 2 c3 ♘f6 3 e5 ♘d5 4 d4 c×d4 5 c×d4 d6 6 ♘f3 ♘c6 7 ♗c4 ♘b6 8 ♗b5 e6 9 ♗g5 ♗e7 10 ♗×e7 ♕×e7 11 ♘c3 ♗d7 12 0-0 0-0 13 ♖e1 a6 14 ♗d3 d×e5 15 d×e5 ♕c5 16 ♕c2 h6 17 a4 ♘b4 18 ♕e2 ♘×d3 19 ♕×d3 ♖ac8 20 a5 ♘d5 21 ♖ec1 ♖fd8 22 h3 ♗c6 23 ♘×d5 ♕×d5 24 ♕×d5 ♖×d5 25 ♖c3 ♖cd8 26 ♖c4 ♖d3 27 ♘h2 ♖d1+ 28 ♖×d1 ♖×d1+ 29 ♘f1 ♗b5 30 ♖c8+ ♔h7 31 ♔h2 ♖×f1 32 ♔g3 ♗c6 33 ♖c7 ♔g6 34 h4 ♖e1 35 f4 ♖e2 36 b4 ♖×g2+ 37 ♔h3 ♖b2 38 h5+ ♔×h5 39 ♖×f7 ♖×b4 40 ♔g3 ♖b2 41 ♖e7 ♖g2+ 42 ♔h3 g5 43 f×g5 ♖×g5 44 ♖f7 ♖×e5 45 ♔g3 ♖×a5 46 ♔f4 ♖f5+ 47 ♖×f5 e×f5 48 ♔×f5 a5 0-1

**Ostrich – Sfinks X** 1 e4 c5 2 ♘f3 e6 3 c3 d5 4 ♗b5+ ♗d7 5 ♗×d7+ ♘×d7 6 e×d5 e×d5 7 0-0 ♘gf6 8 d4 ♗e7 9 ♗f4 0-0 10 d×c5 ♘×c5 11 ♕e2 ♕d7 12 ♘bd2 ♖fe8 13 b4 ♗d6 14 ♗e3 ♘ce4 15 ♖ac1 ♘g4 16 ♕d3 ♘×e3 17 f×e3 ♘f6 18 ♖fe1 ♖ac8 19 ♕d4 ♕a4 20 e4 d×e4 21 ♘×e4 ♘×e4 22 ♖×e4 ♖×e4 23 ♕×e4 ♗×b4 24 ♕×b7 ♗c5+ 25 ♔h1 ♖d8 26 ♕b3 ♕×b3 27 a×b3 ♖d3 28 ♘e1 ♖e3 29 b4 ♗e7 30 ♘f3 ♗f6 31 c4 ♖b3 32 b5 ♖b2 33 h3 ♔f8 34 h4 ♔e7 35 ♖a1 ♖c2 36 ♖×a7+ ♔d6 37 ♖×f7 ♔c5 38 ♖c7+ ♔b4 39 b6 ♗b2 40 b7 ♖c1+ 41 ♔h2 ♗f6 42 b8♕+ 1-0

**Chaturanga 2.0 – Pion** 1 e4 d6 2 d4 ♘f6 3 ♘c3 g6 4 f4 ♗g7 5 ♘f3 0-0 6 ♗c4 ♘×e4 7 ♘×e4 d5 8 ♗×d5 ♕×d5 9 ♘c5 ♘c6 10 ♘h4 ♕×d4 11 ♕×d4 ♘×d4 12 ♔d2 ♖d8 13 ♘d3 ♗f6 14 g3 ♗×h4 15 g×h4 ♗f5 16 ♖d1 ♘f3+ 17 ♔e2 ♗g4 18 ♔f2 ♘d4 19 ♖d2 ♘×c2 20 ♖×c2 ♖×d3 21 ♗e3 c6 22 ♖g1 ♗f5 23 ♖c5 ♔f8 24 ♖e5 b6 25 ♔e2 ♖ad8 26 ♖g5 e6 27 ♖g3 f6 28 ♗×b6 a×b6 29 ♖ee3 ♖d2+ 30 ♔f3 ♖×h2 31 ♖b3 ♖×h4 32 ♖×b6 ♖d3+ 33 ♔g2 ♗e4+ 34 ♔f2 ♖×f4+ 35 ♔e2 ♖×g3 36 ♖b8+ ♔g7 37 ♖b7+ ♔h6 38 ♖f7 ♗d3+ 39 ♔d2 ♖f2+ 40 ♔c3 c5 41 ♖d7 ♗b5+ 42 ♖d3 ♖×d3# 0-1

*Round 4, October 26, 1982*

**Cray Blitz – Belle** 1 e4 e5 2 ♘c3 ♘f6 3 ♗c4 ♘c6 4 d3 ♗c5 5 ♘f3 d6 6 ♗g5 ♘a5 7 ♗×f6 ♕×f6 8 ♘d5 ♕d8 9 b4 ♘×c4 10 b×c5 c6 11 d×c4 c×d5 12 c×d5 ♕a5+ 13 ♕d2 ♕×c5 14 0-0 0-0 15 ♖ab1 b6 16 ♕b4 f5 17 ♕×c5 b×c5 18 ♘d2 f×e4 19 ♘×e4 ♗a6 20 ♖fc1 ♗c4 21 ♘×d6 ♗×d5 22 c4 ♗c6 23 ♘b7 ♗×b7 24 ♖×b7 ♖f7 25 ♖b5 ♖c8 26 f3 ♔f8 27 ♖e1 ♖e7 28 ♔f2 g6 29 ♖d1 e4 30 ♖e1 e×f3 31 ♖×e7 ♔×e7 32 ♖b7+ ♔e6 33 g×f3 ♖d8 34 ♖×a7 ♖d4 35 ♖×h7 ♖×c4 36 ♖c7 ♔d5 37 a3 ♖c2+ 38 ♔g3 c4 39 ♖d7+ ♔e6 40 ♖g7 ♔f5 41 ♖c7 ♔e6 42 ♖c6+ ♔d5 43 ♖×g6 c3 44 ♖g7 ♔c4 45 h4 ♖c1 46 ♖c7+ ♔b3 47 ♖b7+ ♔a2 48 ♔f4 c2 49 ♖c7 ♔b1 50 ♖b7+ ♔a2 51 ♖c7 ♔b1 52 ♖b7+ ♔a2 ½-½

**Nuchess – Bebe** 1 c3 e5 2 d4 e×d4 3 c×d4 d5 4 ♘c3 ♗e7 5 e3 ♘f6 6 ♗d3 0-0 7 ♘f3 ♘c6 8 0-0 ♖e8 9 a3 ♗g4 10 h3 ♗×f3 11 ♕×f3 ♖c8 12 b4 a6 13 ♗d2 ♗f8 14 g4 h6 15 h4 ♘e4 16 ♘×e4 d×e4 17 ♗×e4 ♕×h4 18 ♗×c6 b×c6 19 ♔g2 ♕g5 20 ♖ac1 ♕d5 21 ♕×d5 c×d5 22 ♖c6 a5 23 b×a5 ♖e6 24 ♖×e6 f×e6 25 ♗b4 ♖b8 26 ♖c1 ♗×b4 27 ♖b1 ♔f7 28 a×b4 c6 29 a6 ♖a8 30 ♖a1 ♖a7 31 ♔g3 ♔e7 32 e4 ♔d7 33 ♔f4 ♔d6 34 f3 ♖f7+ 35 ♔e3 ♖a7 36 e×d5 e×d5 37 ♔f4 ♔e6 38 ♔g3 ♔e7 39 ♔h4 g6 40 f4 ♔f7 41 ♖a2 ♔e6 42 ♖a3 ♔f6 43 ♔g3 ♔e6 44 ♖a1 ♔f6 45 ♖a2 ♔e6 46 ♖a3 ♔f6 47 ♔f3 ♔e6 48 ♖a2 ♔f6 49 ♖a4 ♔e6 50 ♖a5 ♔f6 51 ♔g3 ♔e6 52 ♔h4 ♔f6 53 b5 c×b5 54 ♖×b5 ♖×a6 55 ♖×d5 ♖a3 56 ♖e5 ♖f3 57 g5+ h×g5+ 58 f×g5+ ♔g7 59 d5 ♖d3 60 ♔g4 ♔f8 61 ♔f4 ♖d4+ 62 ♔e3 ♖d1 63 ♔f3 ♖d4 64 ♔e2 ♔g8 65 ♔e3 ♔f8 66 ♔×d4 ♔g7 67 d6 ♔h8 68 ♖e7 ♔g8 69 d7 ♔h8 70 d8♕# 1-0

**Chaos – Ostrich** 1 d4 ♘f6 2 c4 e6 3 ♘c3 ♗b4 4 ♕c2 0-0 5 a3 ♗×c3+ 6 ♕×c3 ♘c6 7 ♘f3 d6 8 ♗g5 h6 9 ♗h4 b6 10 e3 ♗a6 11 d5 e×d5 12 c×d5 ♗×f1 13 ♖×f1 ♘b8 14 ♗×f6 ♕×f6 15 ♕×f6 g×f6 16 ♖c1 ♘a6 17 ♔e2 h5 18 ♖c4 ♔g7 19 ♖fc1 ♘c5 20 b4 ♘a6 21 g3 ♖h8 22 ♖c6 ♖hc8 23 ♘d4 ♔f8 24 ♘b5 ♖e8 25 ♘×c7 ♘×c7 26 ♖×c7 ♖e7 27 ♖×e7 ♔×e7 28 ♖c7+ ♔e8 29 ♔f3 a6 30 ♖c6 ♔d7 31 ♖×b6 ♔c7 32 ♖c6+ ♔d7 33 ♔f4 a5 34 b5 a4 35 ♖a6 ♖×a6 36 b×a6 ♔c7 37 ♔f5 ♔b6 38 ♔×f6 ♔×a6 39 ♔e7 ♔a5 40 ♔×d6 ♔b5 41 ♔e7 f5 42 d6 ♔c4 43 d7 ♔b3 44 f4 ♔×a3 45 d8♕ ♔b4 46 ♕d4+ ♔b5 47 ♕d3+ ♔b4 48 ♕×f5 a3 49 ♕×h5 a2 50 ♕h8 1-0

**Advance 2.4 – Schach 2.6** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗a4 ♘f6 5 0-0 ♗e7 6 ♖e1 b5 7 ♗b3 d6 8 c3 0-0 9 d3 ♘a5 10 ♗c2 c5 11 h3 c4 12 d×c4 ♘×c4 13 a4 ♗d7 14 b3 ♘b6 15 a×b5 ♗×b5 16 c4 ♗c6 17 ♘c3 h6 18 ♖e2 ♘bd7 19 ♗a3 ♕c7 20 ♘d5 ♘×d5 21 e×d5 ♗b7 22 ♘d4 ♘c5 23 ♘f5 ♗g5 24 b4 ♘d7 25 ♗b3 ♖fc8 26 ♗c1 ♗f6 27 ♖e4 ♘f8 28 ♖g4 ♘g6 29 c5 h5 30 ♖c4 d×c5 31 b×c5 ♕d7 32 ♘d6 ♖×c5 33 ♖×c5 ♕×d6 34 ♖ca5 e4 35 ♖1a2 ♖c8 36 ♖5a4 ♗e5 37 ♖×e4 ♗h2+ 38 ♔h1 h4 39 ♗g5 ♗f4 40 ♖c2 ♖b8 41 ♗×f4 ♘×f4 42 ♕g4 ♘×d5 43 ♕h5 g6 44 ♕×h4 ♗c6 45 ♖ec4 ♘e7 46 ♗a2 ♕d1+ 47 ♔h2 ♕d6+ 48 ♕f4 ♖d8 49 ♖b4 ♕×f4+ 50 ♖×f4 ♖f8 51 ♖c5 ♔g7 52 ♗c4 f5 53 ♖e5 ♖e8 54 ♖d4 ♔f6 55 ♖c5 ♔g5 56 ♖d6 ♗b5 57 ♗×b5 a×b5 58 ♖e5 ♘c8 59 ♖×g6+ ♔×g6 60 ♖×e8 ♘a7 61 ♖a8 ♘c6 62 ♖a6 1-0

**Philidor – Fidelity X** 1 e4 c5 2 ♘f3 ♘c6 3 ♗b5 e6 4 ♘c3 a6 5 ♗e2 d5 6 0-0 d4 7 ♘b1 ♘f6 8 d3 ♗d6

9 ♗g5 0-0 10 ♘a3 h6 11 ♗d2 e5 12 ♘c4 ♗c7 13 a4 ♗e6 14 b3 b5 15 a×b5 a×b5 16 ♖×a8 ♕×a8 17 ♘b2 ♗a5 18 ♗×a5 ♕×a5 19 ♕a1 ♖a8 20 ♕×a5 ♖×a5 21 ♖c1 ♖a2 22 ♖b1 ♘d7 23 ♘h4 ♘b4 24 ♗d1 g6 25 h3 ♔h7 26 ♘f3 f5 27 e×f5 ♗×f5 28 ♘d2 ♘d5 29 g4 ♗e6 30 ♘e4 ♘c3 31 ♘×c3 d×c3 32 ♘c4 b×c4 33 b×c4 ♔g7 34 ♔g2 ♖b2 35 ♖a1 ♔f6 36 ♖a3 ♔g5 37 ♖×c3 ♔h4 38 ♖a3 h5 39 ♖a6 ♘f8 40 ♖a8 ♘h7 41 ♖h8 ♘f6 42 ♖f8 ♘d7 43 ♖e8 ♖b6 44 ♖h8 ♘f6 45 ♗f3 ♖b2 46 c3 ♖c2 47 ♖h6 ♔g5 48 ♖h8 ♖×c3 49 g×h5 g×h5 50 ♖d8 ♗d7 51 ♗e2 ♔h4 52 ♖f8 ♗×h3+ 53 ♔h2 ♖c2 54 ♖×f6 ♖×e2 55 ♔g1 ♖e1+ 56 ♔h2 ♗f1 57 ♖d6 e4 58 d4 ♗e2 59 d×c5 ♗f3 0-1

**Pion — Savant X** 1 ♘f3 d5 2 g3 ♘f6 3 ♗g2 ♘c6 4 0-0 e5 5 d3 ♗g4 6 h3 ♗×f3 7 ♗×f3 e4 8 ♗g2 ♗c5 9 ♘d2 ♕e7 10 c4 e3 11 f×e3 ♕×e3+ 12 ♔h2 ♕e6 13 ♘b3 ♗d4 14 c×d5 ♘×d5 15 ♘×d4 ♘×d4 16 e4 ♘b6 17 ♗e3 0-0-0 18 h4 ♔b8 19 ♗h3 ♕e7 20 a4 h6 21 a5 ♘c8 22 ♕a4 ♘e6 23 ♖ad1 g5 24 ♗×e6 ♕×e6 25 ♗d4 g×h4 26 ♗×h8 h×g3+ 27 ♔×g3 ♖×h8 28 ♕d4 ♖g8+ 29 ♔h2 ♕g4 0-1

**Sfinks — Chaturanga 2.0** 1 d4 d5 2 c4 c6 3 c×d5 ♕×d5 4 ♘c3 ♕c4 5 e4 ♕b4 6 a3 ♕a5 7 ♗d3 ♘f6 8 ♘f3 ♗g4 9 0-0 ♘bd7 10 b4 ♕h5 11 ♗e2 0-0-0 12 ♗f4 ♘b6 13 h3 ♗e6 14 ♘g5 ♕h4 15 g3 ♕h6 16 ♘×e6 ♕×h3 17 ♘g5 ♕h6 18 ♘×f7 ♕h3 19 ♘×h8 e5 20 ♗×e5 ♗×b4 21 ♗×f6 ♖×d4 22 ♗×d4 h5 23 a×b4 c5 24 b×c5 ♘d7 25 ♖×a7 ♔b8 26 c6 ♕×f1+ 27 ♗×f1 ♘f8 28 ♖×b7+ ♔c8 29 ♗b6 ♘d7 30 ♕×d7# 1-0

In 1983 the World Computer Championship replaced the North American Championship at the ACM Conference in New York. The tournament table and games from this event can therefore be found in the section devoted to games from the World Championship (pages 394-400).

**ACM 1984 — San Francisco**

| | rate | perf | 1 | 2 | 3 | 4 | tot |
|---|---|---|---|---|---|---|---|
| 1 Cray Blitz | 2200 | 2459 | 2+ | 4+ | 3+ | 6+ | 4 |
| 2 Fidelity X | 1900 | 2041 | 1− | 11+ | 9+ | 10+ | 3 |
| 3 Bebe | 2100 | 2107 | 12+ | 10+ | 1− | 8+ | 3 |
| 4 Chaos | 1800 | 2046 | 11+ | 1− | 13+ | 7+ | 3 |
| 5 Belle | 2200 | 2016 | 7= | 6− | 12+ | 13+ | 2½ |
| 6 Nuchess | 2150 | 2087 | 8= | 5+ | 7= | 1− | 2 |
| 7 Phoenix | 0 | 1981 | 5= | 9+ | 6= | 4− | 2 |
| 8 Novag X | 1900 | 1896 | 6= | 12+ | 10= | 3− | 2 |
| 9 Int Software X | 0 | 1788 | 13+ | 7− | 2− | 11+ | 2 |
| 10 Schach 2.7 | 0 | 1722 | 14+ | 3− | 8= | 2− | 1½ |
| 11 Ostrich | 1750 | 1558 | 4− | 2− | 14+ | 9− | 1 |
| 12 Awit | 0 | 1600 | 3− | 8− | 5− | 14+ | 1 |
| 13 Merlin | 0 | 1506 | 9− | 14+ | 4− | 5− | 1 |
| 14 Xenarbor | 1400 | 1172 | 10− | 13− | 11− | 12− | 0 |

*Round 1, October 7, 1984*

**Cray Blitz — Fidelity X** 1 e4 c5 2 d4 c×d4 3 ♘f3 e5 4 c3 ♕a5 5 ♕b3 f6 6 ♗c4 ♘h6 7 ♗×h6 g×h6 8 ♗g8 ♔d8 9 ♕f7 ♖×g8 10 ♕×g8 ♔e8 11 ♕×h7 ♕b5 12 b3 ♕c6 13 0-0 d×c3 14 ♘×c3 b6 15 ♘d5 ♕d6 16 ♘h4 ♗a6 17 ♘f5 ♕a3 18 ♕g6+ ♔d8 19 ♕×f6+ ♔c8 20 ♖fc1+ ♗c4 21 ♖×c4+ ♕c5 22 ♕×f8+ ♔b7 23 ♖×c5 b×c5 24 ♕×c5 ♘c6 25 ♘d6+ ♔b8 26 ♕b5# 1-0

**Phoenix — Belle** 1 d4 d5 2 ♗g5 f6 3 ♗f4 ♘c6 4 ♘f3 g5 5 ♗g3 g4 6 ♘g1 h5 7 h3 e5 8 h×g4 ♗×g4 9 c3 e×d4 10 c×d4 ♘ge7 11 f3 ♗e6 12 e3 ♘f5 13 ♗f2 ♕d6 14 ♗d3 ♘b4 15 g4 ♘×d3+ 16 ♕×d3 ♕b4+ 17 ♘c3 ♘g7 18 0-0-0 0-0-0 19 ♕g6 ♕e7 20 ♗h4 ♘e8 21 g5 ♗f7 22 ♕d3 f×g5 23 ♕f5+ ♔b8 24 ♗×g5 ♕d7 25 ♕×d7 ♖×d7 26 e4 ♖g8 27 ♘h3 d×e4 28 f×e4 ♗e6 29 ♗f4 ♗g4 30 ♖d3 ♗g7 31 ♗e3 ♖f7 32 ♘g5 ♖ff8 33 ♘d5 ♗h8 34 ♔b1 ♘d6 35 ♖c1 ♘b5 35 a4 ♘×d4 37 ♗×d4 ♖×g5 38 ♘×c7 ♗e2 39 ♖a3 ♖g4 40 ♘e6 ♖fg8 41 ♗×h8 ♖×h8 42 ♘c5 h4 43 ♘d7+ ♔a8 44 ♘f6 ♖f4 45 e5 ♖d8 46 ♖e3 ♗f3 47 ♖d3 ♖h8 48 ♖dc3 ♗c6 49 b3 ♖f5 50 ♘d5 ♖×e5 51 ♘c7+ ♔b8 52 ♘a6+ b×a6 53 ♖×c6 h3 54 ♖×a6 h2 55 ♖h1 ♖g5 56 ♖c6 ♖h3 57 ♔b2 ♖gg3 58 ♖c2 ♖×b3+ 59 ♔c1 ♖a3 60 ♖h×h2 ♖×h2 61 ♖×h2 ♖×a4 ½-½

**Nuchess — Novag X** 1 c4 e5 2 ♘c3 ♘f6 3 ♘f3 ♘c6 4 g3 ♗b4 5 ♗g2 0-0 6 0-0 ♖e8 7 a3 ♗×c3 8 d×c3 d6 9 ♗e3 ♗e6 10 ♕a4 ♘g4 11 ♘g5 ♘×e3 12 ♘×e6 12 ♘×e6 ♖×e6 13 f×e3 ♕g5 14 ♗×c6 ♕×e3+ 15 ♖f2 ♖f6 16 ♖af1 ♖×f2 17 ♖×f2 b×c6 18 ♕×c6 ♖f8 19 ♕×c7 ♕c1+ 20 ♔g2 ♕×b2 21 ♕×d6 ♕×c3 22 c5 h6 23 c6 ♕d4 24 ♕×d4 e×d4 25 ♖f5 ♖c8 26 ♖c5 ♔f8 27 ♔f3 ♔e7 28 ♔e4 ♔d6 29 ♖a5 ♖c7 30 ♖a6 ♔c5 31 a4 f6 32 h4 ♔c4 33 ♖a5 g6 34 ♖a6 h5 35 ♖a5 ♖×c6 36 ♖×a7 ♖e6+ 37 ♔f3 ♖e3+ 38 ♔f2 ♔c3 39 ♖b7 ♖e6 40 a5 ♖a6 41 ♖b5 f5 42 ♖b6 ♖×a5 43 ♖×g6 ♖e5 44 ♖g5 ♔c2 45 ♔f1 ♖×e2 46 ♖×f5 d3 47 ♖c5+ ♔d1 48 ♖×h5 d2 49 ♖c5 ♖e1+ 50 ♔f2 ♖e2+ 51 ♔f1 ♖e1+ 52 ♔f2 ♖e2+ 53 ♔f1 ½-½

**Awit — Bebe** 1 c4 e5 2 ♘c3 ♘f6 3 ♘f3 ♘c6 4 d3 d6 5 g3 g6 6 ♗g2 ♗g7 7 0-0 0-0 8 ♖b1 a5 9 a3 ♗d7 10 ♗g5 h6 11 ♗d2 g5 12 ♕b3 b6 13 a4 g4 14 ♘e1 ♘d4 15 ♕a2 ♗c6 16 f3 h5 17 h4 g×h3 18 ♗×h3 ♗d7 19 ♗×d7 ♕×d7 20 ♔g2 ♖a6 21 ♖h1 ♖fa8 22 b4 a×b4 23 ♖×b4 ♘e6 24 ♖h4 ♘c5 25 ♘c2 ♘×a4 26 ♘×a4 ♖×a4 27 ♖×a4 ♕×a4 28 ♕×a4 ♖×a4 29 ♘b4 c5 30 ♘c6 ♖a2 31 ♗g5 ♖×e2+ 32 ♔f1 ♖b2 33 ♘e7+ ♔h7 34 ♗×f6 ♗×f6 35 ♖×h5+ ♔g7 36 ♘f5+ ♔g6 37 g4 ♖b1+ 38 ♔g2 ♖b2+ 39 ♔f1 ♗g7 40 f4 f6 41 ♘×d6 e×f4 42 ♘e4 ♖b1+ 43 ♔e2 ♗h6 44 ♘×c5 b×c5 45 ♖×c5 ♖g1 46 d4 ♖×g4 47 ♖b5 f3+ 48 ♔d3 ♗c1 49 ♖b8 ♖×d4+ 50 ♔×d4 f2 51 ♖g8+ ♔f5 52 ♔c3 f1♕ 0-1

**Schach 2.7 — Xenarbor** 1 e4 c5 2 ♘f3 ♘c6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 e5 6 ♘db5 d6 7 ♗g5 a6 8 ♗×f6 g×f6 9 ♘a3 b5 10 ♕d5 ♗d7 11 ♘e2 ♖g8 12 ♕d3 ♗e6 13 ♘b1 ♕a5+ 14 ♕c3 ♕×c3+ 15 ♘b×c3 ♗e7 16 0-0-0 ♔d7 17 ♘d5 ♗×d5 18 e×d5 ♘a7 19 ♘g3 ♔c7 20 ♗d3 ♖h8 21 ♘f5 ♗f8 22 ♖he1 ♖g8 23 ♗e4 h6 24 ♖d3 ♔b7 25 ♖h3 ♖c8 26 ♘×h6

♗×h6+ 27 ♖×h6 ♖h8 28 ♖×f6 ♖×h2 29 ♖×d6 ♖c7 30 ♖f6 ♖d7 31 ♗f3 a5 32 ♖×e5 ♖h1+ 33 ♔d2 ♘c8 34 ♗g4 ♖c7 35 d6 ♘×d6 36 ♖×d6 b4 37 ♗f3+ ♔b8 38 ♖b5+ ♔c8 39 ♗g4+ ♖d7 40 ♖×d7 ♖h4 41 ♖c5+ ♔b8 42 ♖e5 ♖h8 43 ♖b5+ ♔c8 44 ♖×f7+ 1-0

**Ostrich — Chaos** 1 e4 c5 2 c3 ♘f6 3 e5 ♘d5 4 ♗c4 ♘b6 5 ♗b3 ♘c6 6 ♘f3 d6 7 0-0 d×e5 8 ♕e2 f6 9 ♘h4 g6 10 ♘a3 ♗d7 11 d3 e6 12 ♘f3 ♘a5 13 ♗d2 ♘×b3 14 a×b3 ♗c6 15 ♘c4 ♘×c4 16 b×c4 ♗e7 17 ♗h6 ♔f7 18 ♘d2 g5 19 ♕h5+ ♔g8 20 ♕h3 ♕d7 21 ♖fe1 a6 22 ♘b3 ♖d8 23 ♖e3 ♔f7 24 ♕h5+ ♔g8 25 h3 ♗a4 26 ♘c1 ♕e8 27 ♕f3 ♗c6 28 ♕g4 f5 29 ♕e2 ♕g6 30 ♗×g5 ♗×g5 31 ♘b3 ♗×e3 32 f×e3 ♖×d3 33 ♘×c5 ♖×e3 34 ♕d2 ♖×h3 35 ♕d8+ ♔f7 36 ♕c7+ ♔e8 37 ♕c8+ ♔e7 38 ♕c7+ ♔e8 39 ♕c8+ ♔f7 40 ♕c7+ ♔f6 41 ♘d7+ ♗×d7 42 ♕×d7 ♕g3 43 ♖f1 ♕e3+ 44 ♖f2 ♖h4 45 ♕d2 ♕×d2 46 ♖×d2 ♖×c4 47 ♖d7 b5 48 ♖d6 a5 49 ♖b6 b4 0-1

**Merlin — Intelligent Software X** 1 e4 c5 2 c3 d5 3 e×d5 ♕×d5 4 d4 e6 5 ♘f3 ♘f6 6 ♗e2 ♘c6 7 0-0 ♗d7 8 d×c5 ♗×c5 9 ♕c2 e5 10 ♖d1 ♕e6 11 b4 ♗b6 12 ♘g5 ♕e7 13 ♘d2 h6 14 ♘ge4 ♗f5 15 b5 ♘d8 16 ♘d6+ ♕×d6 17 ♕×f5 ♕e6 18 ♕d3 e4 19 ♕g3 e3 20 f×e3 ♕×e3+ 21 ♕×e3+ ♕×e3+ 22 ♔h1 ♗×d2 23 ♗×d2 ♘e4 24 ♗e1 0-0 25 ♖d4 ♘c5 26 ♖ad1 ♘de6 27 ♖d5 a5 28 ♗f3 ♖fc8 29 ♔g1 a4 30 ♗g3 a3 31 ♖d6 ♘g5 32 ♗g4 ♖e8 33 ♖6d5 ♘ge4 34 ♗f4 ♘×c3 35 ♗f3 ♘×d1 36 ♖×d1 ♖a5 37 b6 ♖a6 38 ♗c7 ♖e6 39 ♖d8+ ♔h7 40 ♗d5 ♖e1+ 41 ♔f2 ♖e7 42 ♔f3 ♘e6 43 ♗e4+ g6 44 ♗×b7 ♘×c7 45 ♗×a6 ♘×a6 46 ♖a8 ♘c5 47 ♖×a3 ♔g7 48 ♖a5 ♘b7 49 ♖b5 f5 50 ♖b3 ♔f6 51 a3 g5 52 ♖e3 ♖×e3+ 53 ♔×e3 ♔e5 54 h4 adjudicated 0-1

*Round 2, October 7, 1984*

**Chaos — Cray Blitz** 1 d4 d5 2 c4 e6 3 ♘c3 ♘f6 4 ♗g5 ♘bd7 5 e3 ♗b4 6 c×d5 e×d5 7 ♗d3 c5 8 ♘ge2 c4 9 ♗c2 h6 10 ♗h4 ♕b6 11 0-0 0-0 12 h3 ♖e8 13 ♗a4 ♗×c3 14 b×c3 ♘e4 15 ♕c2 ♕e6 16 ♘f4 ♕d6 17 f3 ♘g5 18 ♖fe1 a6 19 ♕d2 b5 20 ♗c2 ♘f6 21 ♖ab1 ♗d7 22 ♖b4 a5 23 ♖b2 b4 24 ♔h1 ♖ac8 25 ♗g3 b×c3 26 ♕×c3 ♕a6 27 ♖eb1 a4 28 ♗h4 ♖c7 29 ♖b6 ♕a8 30 ♕b4 ♖cc8 31 ♕c3 ♘gh7 32 ♗g3 a3 33 ♘e2 ♗c6 34 ♗d6 ♘d7 35 ♖6b4 ♖e6 36 ♗f4 ♘hf8 37 ♗f5 ♖ee8 38 ♗d6 ♘e6 39 ♔g1 g6 40 ♗×e6 ♖×e6 41 ♗f4 g5 42 ♗g3 ♖ce8 43 ♗f2 ♘f6 44 ♘g3 ♔f8 45 ♘f5 ♘h5 46 g4 ♘g7 47 e4 ♘×f5 48 g×f5 ♖d6 49 ♗g3 ♖d7 50 ♖b8 ♕a4 51 e×d5 ♗×d5 52 f6 ♖dd8 53 ♖8b4 ♕a7 54 ♖b6 ♖e2 55 ♖b8 ♖ee8 56 ♖8b6 ♔g8 57 ♗e5 ♕d7 58 ♔h2 ♖a8 59 ♔g2 ♕f5 60 ♖1b5 ♗e6 61 ♔g1 ♕d3 62 d5 ♕d1+ 63 ♔g2 ♕e2+ 64 ♔g3 ♗d7 65 d6 ♗×b5 66 ♖×b5 ♕×a2 0-1

**Bebe — Schach 2.7** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗a4 ♘f6 5 0-0 ♗e7 6 ♖e1 b5 7 ♗b3 d6 8 c3 0-0 9 h3 ♘b8 10 d3 ♘bd7 11 ♘bd2 ♗b7 12 ♘f1 ♘c5 13 ♗c2 a5 14 ♘g3 c6 15 d4 ♘cd7 16 ♘f5 ♖e8 17 ♗g5 ♗f8 18 d×e5 ♘×e5 19 ♘×e5 ♖×e5 20 ♕f3 ♕b6 21 ♘h6+ g×h6 22 ♗×f6 ♖g5 23 ♗×g5 h×g5 24 ♕f5 ♗g7 25 ♖ad1 g4 26 ♕×g4 ♕c7 27 e5 d5 28 ♕f5 h6 29 e6 f×e6 30 ♖×e6 ♖c8 31 ♕h7+ ♔f8 32 ♖de1 1-0

**Belle — Nuchess** 1 e4 c5 2 c3 e6 3 d4 c×d4 4 c×d4 d5 5 ♗d3 d×e4 6 ♗×e4 ♘f6 7 ♗c2 ♘c6 8 ♘f3 b6 9 0-0 ♗a6 10 ♖e1 ♗b4 11 ♘c3 0-0 12 a3 ♗e7 13 ♗g5 ♕d6 14 b4 ♗c4 15 ♘e5 b5 16 ♗×f6 ♗×f6 17 ♘e4 ♗×e5 18 ♘×d6 ♗×d6 19 ♗d3 ♘e7 20 ♗×c4 b×c4 21 ♖c1 ♖ac8 22 ♕e2 c3 23 ♕a6 ♗f4 24 ♖c2 ♖a8 25 ♕b7 ♘d5 26 ♖ee2 ♗d2 27 ♕a6 g6 28 ♖e4 ♖fc8 29 ♔h1 ♖f8 30 ♖h4 ♗g5 31 ♖h3 ♗e7 32 ♕a4 ♗f6 33 ♖d3 ♖fd8 34 b5 ♖db8 35 ♖c1 a6 36 b×a6 ♖b6 37 a7 ♖b7 38 ♖a1 ♖a×a7 39 ♕d1 ♖b2 40 f3 c2 41 ♕f1 ♗g5 42 g3 ♖ab7 43 f4 ♖b1 44 ♖f3 ♗f6 45 f5 ♗×d4 46 ♖×b1 c×b1♕ 0-1

**Novag X — Awit** 1 e4 c5 2 ♘f3 d6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 a6 6 ♗e2 e6 7 0-0 ♗e7 8 f4 0-0 9 ♔h1 ♗d7 10 e5 d×e5 11 f×e5 ♘e8 12 ♗f3 ♕c7 13 ♕e2 ♗c5 14 ♘b3 ♗b4 15 ♖f2 f5 16 ♗d2 ♘c6 17 ♘d5 ♕×e5 18 ♘×b4 ♕×b2 19 ♘×c6 b×c6 20 ♗×c6 ♗×c6 21 ♕×e6+ ♔h8 22 ♕×c6 ♘f6 23 ♗c3 ♖ac8 24 ♕×c8 ♕×a1+ 25 ♘×a1 ♖×c8 26 ♗×f6 ♖b8 27 ♘b3 g×f6 28 ♖×f5 ♖e8 29 ♔g1 ♖c8 30 ♖×f6 ♖×c2 31 ♖×a6 ♖b2 32 ♖a7 h5 33 ♔f1 h4 34 ♔g1 ♖b1+ 35 ♔f2 ♔g8 36 ♘d2 ♖b2 37 ♔e3 ♔f8 38 a4 ♖a2 39 a5 ♖a3+ 40 ♔f2 ♔e8 41 a6 ♔d8 42 ♘c4 ♖a2+ 43 ♔f3 h3 44 ♘e3 ♔c8 45 ♔g3 ♔b8 46 ♖b7+ ♔a8 47 ♔×h3 ♖a3 48 ♖e7 ♖×a6 49 g4 ♖h6+ 50 ♔g3 ♖g6 1-0

**Phoenix — Intelligent Software X** 1 d4 d5 2 ♗g5 ♘c6 3 e3 h6 4 ♗h4 ♘f6 5 c4 ♗g4 6 f3 ♗f5 7 c×d5 ♕×d5 8 ♘c3 ♕e6 9 e4 ♘×e4 10 f×e4 ♗×e4 11 ♘ge2 0-0-0 12 ♕b3 ♕g4 13 ♗f2 e6 14 ♘g3 ♗b4 15 ♗e2 ♗×c3+ 16 b×c3 ♕f4 17 0-0 ♗d5 18 ♕a3 ♖d7 19 c4 ♗×g2 20 ♔×g2 ♕d6 21 ♕×d6 ♖×d6 22 ♘e4 ♖dd8 23 ♖ad1 f5 24 ♘c3 b6 25 d5 e×d5 26 c×d5 ♘e7 27 ♗a6+ ♔b8 28 ♗g3 ♘c8 29 ♖×f5 ♘d6 30 ♗×d6 ♖×d6 31 ♖f7 ♖g6+ 32 ♔f3 ♖f6+ 33 ♖×f6 g×f6 34 ♖e1 h5 35 ♖e7 ♖g8 36 ♘b5 ♖c8 37 ♗×c8 ♔×c8 38 ♖×c7+ ♔d8 39 d6 ♔e8 40 ♘d4 1-0

**Xenarbor — Merlin** 1 d4 d5 2 c4 c6 3 ♘f3 ♘f6 4 ♘c3 d×c4 5 a4 ♗f5 6 ♘e5 ♘bd7 7 ♘×c4 ♕c7 8 g3 e5 9 d×e5 ♘×e5 10 ♗f4 ♖d8 11 ♕×d8+ ♕×d8 12 ♗×e5 ♗c2 13 ♗×f6 ♕×f6 14 ♘e3 ♗b3 15 ♗h3 ♗b4 16 ♘ed1 ♗×d1 17 ♖×d1 ♗×c3+ 18 b×c3 ♕×c3+ 19 ♔f1 ♕b4 20 ♗d7+ ♔e7 21 ♗f5 ♕×a4 22 ♔g2 ♔f6 23 ♗b1 ♔e6 24 h4 a5 25 h5 ♕g4 26 f3 ♕c4 27 ♔f2 b5 28 e3 ♕c3 29 h6 ♕b2+ 30 ♔g1 ♕e2 31 h×g7 ♕×d1+ 32 ♔f2 ♕d2+ 33 ♔f1 ♖g8 34 ♖h6+ f6 35 ♗f5+ ♔×f5 36 e4+ ♔e5 37 ♖h5+ ♔d4 38 e5 ♔e3 39 ♔g1 ♕f2+ time forfeit 0-1

**Fidelity X — Ostrich** 1 e4 d5 2 e×d5 ♘f6 3 d4 ♘×d5 4 c4 ♘b6 ♘f3 ♗g4 6 ♗e2 ♘c6 7 d5 ♗×f3 8 ♗×f3 ♘e5 9 b3 g6 10 ♗b2 ♗g7 11 ♘c3 0-0 12 ♗e2 ♘ed7 13 0-0 ♗×c3 14 ♗×c3 e5 15 ♗b4 ♖e8 16 c5 ♘c8 17 ♕c2 ♘f6 18 ♗b5 ♖e7 19 c6 ♘d6 20 c×b7 ♖b8 21 ♗c6 ♕f8 22 ♗c5 ♕d8 23 ♗×a7 ♖×b7 24 ♗×b7 ♘×b7 25 ♕c6 ♘d6 26 a4 ♘de8 27 ♖ad1 ♖d7 28 ♖fe1 e4 29 ♗c5 ♕b8 30 f3 e×f3 31 ♖d3 ♕d8 32 g×f3 ♘g7 33 ♖e5 ♘ge8 34 ♔f2 h5 35 ♔e1 ♘h7 36

♖d2 ♘ef6 37 ♖de2 ♔g7 38 ♗e7 ♕b8 39 b4 h4 40 h3 ♕a7 41 ♗c5 ♕b8 42 a5 ♕c8 43 ♗e7 ♕g8 44 ♖d2 ♕h8 45 ♖d4 ♕c8 46 ♖e2 g5 47 f4 ♔g6 48 f×g5 ♔×g5 49 ♖g4+ ♔h5 1-0

*Round 3, October 8, 1984*

**Cray Blitz – Bebe** 1 e4 c5 2 d4 c×d4 3 ♘f3 ♘c6 4 ♘×d4 ♘f6 5 ♘c3 d6 6 f4 ♕b6 7 ♘b3 e5 8 ♕e2 ♗g4 9 ♕b5 ♕×b5 10 ♗×b5 a6 11 ♗×c6+ b×c6 12 0-0 ♖b8 13 ♘a5 ♗d7 14 ♖d1 ♘g4 15 h3 ♘f6 16 ♘c4 ♖b4 17 b3 d5 18 e×d5 ♗c5+ 19 ♔h2 ♗d4 20 d×c6 ♗×h3 21 ♘d6+ ♔e7 22 ♔×h3 ♔×d6 23 ♗a3 a5 24 ♗×b4+ a×b4 25 ♘b5+ ♔×c6 26 ♘×d4+ e×d4 27 ♖×d4 ♘d5 28 a3 b×a3 29 c4 ♘b4 30 ♖×a3 ♖e8 31 ♖a7 ♖e3+ 32 ♔h4 ♔c5 33 ♖d8 ♖×b3 34 ♖c8+ ♔d4 35 ♖×f7 g6 36 ♖×h7 ♖e3 37 g3 1-0

**Nuchess – Phoenix** 1 f4 d5 2 ♘f3 ♗g4 3 e3 ♘d7 4 ♗e2 e6 5 0-0 ♗d6 6 c4 ♘gf6 7 ♘c3 d×c4 8 ♗×c4 ♘b6 9 ♗e2 0-0 10 ♕b3 ♘fd5 11 d4 ♘×c3 12 b×c3 c5 13 ♗d2 ♕c7 14 a4 c4 15 ♕b5 a6 16 ♕b2 ♘d5 17 h3 ♗f5 18 ♘e5 ♗×e5 19 f×e5 f6 20 ♕a2 f×e5 21 ♕×c4 ♕b6 22 a5 ♕d6 23 ♕b3 e×d4 24 c×d4 ♗e4 25 ♗d3 ♗×d3 26 ♕×d3 ♖ad8 27 ♖×f8+ ♖×f8 28 ♖f1 ♘f6 29 ♕b3 ♕d5 30 ♕b2 ♖c8 31 ♖c1 ♖c6 32 ♕a3 e5 33 ♕b4 ♖×c1+ 34 ♗×c1 e×d4 35 e×d4 ♕b5 36 ♗a3 ♕×b4 37 ♗×b4 ♔f7 38 ♔f2 g5 39 ♔e2 ♘d5 40 ♗d2 h6 41 ♔d3 ♔e6 42 ♔c4 ♔d6 43 h4 g×h4 44 ♗×h6 b5+ 45 a×b6 ♘×b6+ 46 ♔d3 a5 47 ♗g5 a4 48 ♗×h4 ♘d5 49 ♗g5 a3 50 ♔c2 ♘b4+ 51 ♔b1 ♔d5 52 ♗e7 a2+ 53 ♔b2 ♘c2 54 ♔×a2 ♘e3 55 g3 ½-½

**Schach 2.7 – Novag X** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗×c6 d×c6 5 ♘c3 f6 6 d4 e×d4 7 ♕×d4 ♗d6 8 0-0 ♘e7 9 ♕c4 b5 10 ♕b3 c5 11 ♘d5 c4 12 ♕c3 0-0 13 ♖d1 ♔h8 14 ♗f4 ♘×d5 15 ♖×d5 ♗b7 16 ♖d4 ♕e7 17 ♗×d6 c×d6 18 ♖ad1 ♗×e4 19 ♖×d6 ♖ad8 20 ♖6d4 ♗b7 21 ♖e1 ♕f7 22 ♕a5 ♖×d4 23 ♘×d4 ♕d5 24 ♘f3 ♕f5 25 ♕a3 ♔g8 26 ♘d4 ♕g4 27 ♘f3 b4 28 ♕e3 ♗×f3 29 ♕×f3 ♕×f3 30 g×f3 a5 31 ♖e4 ♖c8 32 c3 ♔f7 33 ♔f1 g6 34 ♔g2 b×c3 35 b×c3 h6 36 a4 ♖c6 37 f4 ♖c5 38 h4 ♖c6 39 f3 ♖c7 40 ♔f2 ♖c8 41 ♔e2 ♖c7 ½-½

**Merlin – Chaos** 1 e4 c5 2 c3 ♘f6 3 e5 ♘d5 4 d4 c×d4 4 ♘f3 ♘c6 6 c×d4 d6 7 ♗c4 ♘b6 8 ♗b5 e6 9 0-0 ♗e7 10 ♘c3 d5 11 ♕c2 ♗d7 12 ♗f4 ♖c8 13 ♖ac1 ♘×e5 14 ♗×e5 ♗×b5 15 ♗×g7 ♗×f1 16 ♖×f1 ♖g8 17 ♕×h7 ♔d7 18 ♘e5+ ♔c7 19 ♘b5+ ♔b8 20 ♘×f7 ♖×g7 21 ♕×g7 21 ♕×g7 ♕d7 22 ♕g3+ ♔a8 23 ♘c3 ♗b4 24 ♘e5 ♕c7 25 ♕h3 ♗×c3 26 ♖c1 ♗×d4 27 ♖×c7 ♖×c7 28 ♘d3 e5 29 ♕g3 ♖c8 30 ♕g7 ♗×b2 31 ♔f1 ♗d4 32 ♘×e5 ♖c1+ 33 ♔e2 a6 34 ♕g8+ ♖c8 35 ♕e6 ♗×e5 36 ♕×e5 ♖c2+ 37 ♔f3 ♖×a2 38 ♕d6 ♔a7 39 h4 ♖c2 40 g4 a5 41 h5 ♖c3+ 42 ♔f4 a4 43 f3 a3 44 ♕e5 ♖c4+ 45 ♔f5 ♖a4 46 ♕a1 ♘c4 47 h6 ♘e3+ 48 ♔e6 ♘c2 49 ♕a2 ♖a6+ 50 ♔d7 ♘b4 51 ♕f2+ ♔a8 52 ♕c5 a2 53 ♕×b4 a1♕ 54 ♕f8+ ♔a7 55 h7 ♕a4+ 56 ♔d8 ♕a5+ 57 ♔e8 ♕c7 58 ♕f7 ♕e5+ 59 ♔f8 ♖f6 60 h8♘ ♕g5 61 ♕×f6 ♕×f6+ 62 ♘f7 ♕×f3 63 g5 ♕f5 64 ♔g8 d4 0-1

**Intelligent Software X – Fidelity X** 1 ♘f3 c5 2 g3 ♘c6 3 ♘c3 d5 4 d3 d4 5 ♘e4 e5 6 ♗g2 f5 7 ♘ed2 ♘f6 8 ♘c4 e4 9 ♘fe5 ♘×e5 10 ♘×e5 ♕a5+ 11 ♗d2 ♕b6 12 ♘c4 ♕a6 13 e3 ♗e6 14 ♕e2 ♗e7 15 0-0 0-0 16 a4 ♕c6 17 d×e4 f×e4 18 f4 ♕a6 19 b3 ♗d6 20 ♖fb1 ♗f5 21 e×d4 c×d4 22 a5 ♗c5 23 ♔h1 e3 24 ♗e1 ♗d7 25 ♕d3 ♖ad8 26 ♘e5 ♗b5 27 ♕f5 ♘d5 28 ♕h3 ♖fe8 29 b4 ♗f8 30 ♗f3 ♘f6 31 c4 ♗×c4 32 ♘×c4 ♕×c4 33 ♗×b7 d3 34 b5 d2 35 ♗c6 d×e1♘ 36 ♖×e1 ♖e7 37 ♖ac1 ♕b4 38 a6 ♖d2 39 ♕f5 ♕b2 40 ♕h3 ♘g4 41 ♗d5+ ♔h8 42 ♔g1 ♖×h2 43 ♕g2 ♕f2+ 44 ♕×f2 e×f2+ 45 ♔f1 f×e1♕+ 46 ♖×e1 ♖f2+ 47 ♔g1 ♖×e1# 0-1

**Awit – Belle** 1 c4 e6 2 ♘c3 d5 3 c×d5 e×d5 4 d4 ♗e7 5 e3 ♘f6 6 ♗d3 0-0 7 ♘f3 b6 8 0-0 ♗b7 9 ♖e1 ♘e4 10 ♘e5 f5 11 ♕b3 a5 12 a3 ♘×c3 13 ♕×c3 ♗b4 14 a×b4 a×b4 15 ♖×a8 b×c3 16 ♖a3 c×b2 17 ♗×b2 ♘d7 18 ♖a7 ♘×e5 19 d×e5 ♕b8 20 ♖ea1 c6 21 h3 g6 22 ♖7a4 ♖c8 23 ♗c3 b5 24 ♖b4 ♕c7 25 g4 f×g4 26 h×g4 ♖a8 27 ♖×a8+ ♗×a8 28 f4 ♕a7 29 ♗d2 ♕a2 30 ♗e1 ♕a3 31 ♗×b5 c×b5 32 ♖×b5 ♕×e3+ 33 ♗f2 ♕c1+ 34 ♔h2 ♕×f4+ 35 ♗g3 ♕d2+ 36 ♔h3 ♕h6+ 37 ♔g2 d4+ 38 ♔f1 ♕c1+ 39 ♔e2 ♕c4+ 40 ♔f2 ♕×b5 0-1

**Ostrich – Xenarbor** 1 e4 c5 2 c3 ♘f6 3 e5 ♘d5 4 ♗c4 e6 5 ♗×d5 e×d5 6 d4 ♕b6 7 d×c5 ♗×c5 8 ♕d2 ♕e6 9 ♘f3 h6 10 b4 ♗e7 11 0-0 0-0 12 ♖d1 ♘c6 13 ♕×d5 ♕×d5 14 ♖×d5 f5 15 b5 ♘d8 16 ♗a3 ♗×a3 17 ♘×a3 ♘e6 18 ♘c4 ♘c7 19 ♖c5 ♘e8 20 ♘d6 ♘×d6 21 e×d6 a6 22 b6 ♖f6 23 ♖e1 ♔f8 24 ♖d5 a5 25 ♖ee5 g6 26 a4 f4 27 ♘d2 ♔g7 28 ♘c4 f3 29 g3 ♖e6 30 ♖×e6 d×e6 31 ♖c5 ♔f7 32 ♘e5+ ♔e8 33 ♖×c8+ ♖×c8 34 d7+ ♔d8 35 d×c8♕+ ♔×c8 36 h4 g5 37 h×g5 h×g5 38 ♔h2 ♔d8 39 g4 ♔e7 40 ♘×f3 ♔f6 41 ♔g3 e5 42 ♘d2 ♔e6 43 ♘b3 ♔d5 44 ♘×a5 ♔c5 45 ♘×b7+ ♔×b6 46 ♘d6 ♔a5 47 ♘e4 ♔×a4 48 ♘×g5 ♔b3 49 ♘e4 ♔b2 50 g5 ♔b3 51 g6 ♔c4 52 ♔g4 ♔d3 53 ♔f5 1-0

*Round 4, October 9, 1984*

**Nuchess – Cray Blitz** 1 c4 e5 2 ♘c3 ♗b4 3 a3 ♗×c3 4 d×c3 ♘e7 5 g3 d5 6 c×d5 ♕×d5 7 ♕×d5 ♘×d5 8 ♗g2 ♘b6 9 a4 0-0 10 a5 ♘c4 11 ♖a4 ♘d6 12 a6 ♘d7 13 ♗e3 ♘b6 14 ♖h4 ♖d8 15 a×b7 ♗×b7 16 ♗×b7 ♘×b7 17 ♘f3 ♖d5 18 c4 ♖a5 19 0-0 ♖a2 20 ♖d1 ♖×b2 21 c5 ♘c8 22 ♖d7 f6 23 ♖g4 g6 24 ♖h4 h5 25 ♖×c7 ♘d8 26 ♖a4 ♖b7 27 ♖×b7 ♘×b7 28 ♖a6 ♔f7 29 ♘d2 ♘d8 30 ♘e4 f5 31 ♘g5+ ♔g7 32 ♘f3 ♘f7 33 ♘×e5 34 ♗d4 ♔g8 35 ♗×e5 ♘e7 36 e3 ♔f7 37 ♖f6+ ♔g8 38 ♔g2 ♖c8 39 ♔f3 ♖e8 40 ♖a6 ♖a8 41 ♔f4 ♔f7 42 ♔g5 ♖g8 43 ♖f6+ ♔e8 44 ♗d6 ♘c8 45 ♖×g6 ♖×g6+ 46 ♔×g6 ♘×d6 47 c×d6 a5 48 g4 h×g4 49 ♔×f5 a4 50 e4 a3 51 ♔×g4 a2 52 e5 a1♕ 53 f4 ♕g1+ 54 ♔f5 ♕×h2 55 e6 ♕c2+ 0-1

**Bebe – Novag X** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗a4 ♘f6 5 0-0 b5 6 ♗b3 d6 7 ♘g5 d5 8 e×d5 ♘d4 9 d6 ♘×b3 10 d×c7 ♕×c7 11 a×b3 h6 12 ♘f3 e4 13 ♖e1 ♗e7 14 ♘c3 e×f3 15 ♘×b5 ♕c5 16 ♕×f3 ♕×b5 17 ♕×a8 ♕b7 18 ♕×b7 ♗×b7 19 d4 ♔d7 20 c4 ♗b4 21 ♖e2 ♖d8 22 d5 ♘×d5 23 c×d5 ♗×d5 24 ♖×a6

♗×b3 25 ♖e4 ♗c5 26 ♗e3 ♔c7 27 ♖a1 ♗d5 28 ♗f4+ ♔c6 29 ♖ea4 ♗e6 30 ♗e5 ♖d2 31 ♗×g7 ♗×f2+ 32 ♔h1 ♗d5 33 ♗c3 ♖c2 34 ♖a6+ ♔c7 35 ♖6a5 ♔c6 36 ♖a6+ ♔c7 37 ♖6a5 ♔c6 38 ♖a8 ♗b6 39 ♖g8 f5 40 ♖g6+ ♔b5 41 ♖d1 ♗e4 42 ♗d2 ♖×b2 43 ♗×h6 f4 44 ♖g5+ ♔c6 45 ♖c1+ ♔d6 46 ♗g7 ♖f2 47 ♖d1+ ♔c6 48 ♗e5 ♗e3 49 ♖g4 ♗c2 50 ♖a1 ♔d5 51 ♗b8 f3 52 g×f3 ♖×f3 53 ♖a5+ ♔e6 54 h4 ♗f5 55 ♖ga4 ♔f7 56 ♔g2 ♖f2+ 57 ♔g3 ♖f1 58 ♗f4 ♗f2+ 59 ♔g2 ♗d3 60 ♖d5 ♗e2 61 ♖d7+ ♔g6 62 h5+ ♔×h5 63 ♖d2 ♗b6 64 ♖×e2 ♖g1+ 65 ♔h2 ♖b1 66 ♖g2 ♖b5 67 ♖a6 ♖f5 68 ♔g3 ♖f6 1-0

**Chaos — Phoenix** 1 d4 c5 2 d5 e5 3 e4 d6 4 c4 g6 5 ♘c3 ♗g7 6 ♗d3 ♘e7 7 ♘ge2 0-0 8 ♗d2 f5 9 f3 ♘a6 10 ♗g5 ♘b4 11 ♗b1 h6 12 ♗h4 g5 13 ♗f2 f×e4 14 ♗×e4 ♗f5 15 0-0 ♕d7 16 a3 ♘a6 17 ♕b3 ♘c7 18 ♗×f5 ♘×f5 19 ♕×b7 ♖fb8 20 ♕c6 ♕×c6 21 d×c6 ♖×b2 22 ♖ab1 ♖ab8 23 ♘g3 ♖×b1 24 ♖×b1 ♖×b1+ 25 ♘×b1 ♘e7 26 ♘e4 ♘e8 27 ♘×d6 ♘×d6 28 ♗×c5 ♘ec8 29 ♘d2 ♔f7 30 ♔f2 ♔e6 31 ♔e3 ♘×c4+ 32 ♘×c4 ♔d5 33 ♗b4 ♔×c4 34 ♔e4 ♔b5 35 c7 ♔b6 36 ♔f5 a5 37 ♗c3 ♔×c7 38 ♔g6 ♗f8 39 ♗×a5+ ♔c6 40 ♗b4 ♗×b4 41 a×b4 ♘e7+ 42 ♔×h6 g4 43 ♔g5 g×f3 44 g×f3 ♔b5 45 ♔f6 ♘d5+ 46 ♔×e5 ♘×b4 47 h4 ♘d3+ 48 ♔f5 ♘c5 49 f4 ♔c6 50 h5 ♘d7 51 h6 ♘f8 52 ♔f6 ♔d7 53 ♔f7 ♘h7 54 ♔g7 ♔e7 55 ♔×h7 ♔f7 56 f5 ♔f6 57 ♔g8 1-0

**Fidelity X — Schach 2.7** 1 e4 e5 2 ♗c4 ♘f6 3 d3 c6 4 ♘f3 d5 5 ♗b3 ♗d6 6 ♘c3 ♗e6 7 ♗g5 ♕a5 8 0-0 ♘bd7 9 e×d5 ♘×d5 10 ♘e4 ♗c7 11 ♗d2 ♕a6 12 ♘fg5 0-0 13 ♘×e6 f×e6 14 ♕g4 ♖ae8 15 ♕h3 ♗d8 16 a4 ♘c7 17 ♘d6 ♖e7 18 ♗c4 ♕b6 19 ♘c8 ♕×b2 20 ♖a2 ♕d4 21 c3 ♕c5 22 ♗e3 ♕a5 23 ♘×e7+ ♗×e7 24 ♗×e6+ ♘×e6 25 ♕×e6+ ♖f7 26 ♕×d7 ♗f8 27 ♕h3 ♕×c3 28 ♗×a7 ♕b3 29 ♖aa1 ♕b4 30 ♕e6 ♕a5 31 ♗b8 ♕d8 32 ♗×e5 ♕×d3 33 ♖ab1 ♕a6 34 ♖b3 c5 35 ♖b6 ♕a8 36 ♖fb1 ♗e7 37 ♖×b7 ♕d8 38 ♖×e7 ♕×e7 39 ♖b8+ 1-0

**Belle — Merlin** 1 e4 e5 2 ♗c4 ♘f6 3 d4 e×d4 4 ♘f3 d5 5 e×d5 ♗b4+ 6 c3 ♕e7+ 7 ♗e2 d×c3 8 b×c3 ♗c5 9 0-0 0-0 10 c4 ♗g4 11 ♘c3 ♘a6 12 h3 ♗f5 13 ♘d4 ♗g6 14 ♗e3 ♘b4 15 a3 ♘a6 16 ♕c1 ♖ad8 17 ♖d1 ♗b6 18 ♗d3 ♗×d3 19 ♖×d3 ♘e4 20 ♘f5 ♕e5 21 ♗×b6 a×b6 22 ♘×e4 ♕×f5 23 ♕e1 ♖a8 24 ♖b1 ♖fb8 25 ♖e3 ♖c8 26 ♖bb3 ♕f4 27 ♖g3 ♕e5 28 ♖be3 ♕d4 29 ♕d2 ♕×d2 30 ♘×d2 ♘c5 31 ♖e7 ♔f8 32 ♖e1 ♖a4 33 ♖c3 ♖ca8 34 ♘b1 f6 35 f3 f5 36 g4 f4 37 ♔g2 ♔f7 38 h4 ♖e8 39 ♖×e8 ♔×e8 40 ♘d2 ♔e7 41 ♘e4 ♘×e4 42 f×e4 ♔d6 43 ♔f2 ♔e5 44 ♔f3 ♖a8 45 ♖d3 ♖a5 46 ♖b3 g6 47 g5 ♖a4 48 ♖c3 ♖a8 49 ♖b3 ♖d8 50 ♖b2 ♖e8 51 ♖b4 ♖h8 52 ♖b5 ♖f8 53 ♖b1 ♖c8 54 ♖b2 ♖d8 55 ♖b4 ♖c8 56 ♖a4 c6 57 d×c6 b×c6 58 ♖a7 ♖h8 59 a4 ♔d4 60 ♖b7 ♔c5 61 ♔×f4 ♖f8+ 62 ♔e3 ♖h8 63 ♖c7 h6 64 ♖g7 h×g5 65 h×g5 ♖a8 66 ♖×g6 ♖×a4 67 ♖f6 ♖×c4 68 ♖f5+ ♔d6 69 g6 ♖c3+ 70 ♔f4 ♖c1 71 g7 ♖f1+ 72 ♔g5 ♖g1+ 73 ♔f6 ♖×g7 74 ♔×g7 ♔e6 75 ♖f6+ 1-0

**Intelligent Software X — Ostrich** 1 ♘f3 e6 2 d4 d5 3 ♗f4 ♗d6 4 ♗×d6 ♕×d6 5 ♘c3 ♘c6 6 e3 a6 7 ♗e2 ♘f6 8 a3 0-0 9 ♕d3 ♖b8 10 0-0 ♗d7 11 ♖fd1 b5 12 ♔f1 ♖be8 13 h3 e5 14 d×e5 ♘×e5 15 ♘×e5 ♖×e5 16 f4 ♖e6 17 ♘×d5 ♘×d5 18 ♕×d5 ♖×e3 19 ♕×d6 c×d6 20 ♖×d6 ♖fe8 21 ♗d3 ♗c8 22 a4 b×a4 23 ♖×a4 ♖3e6 24 ♖×e6 ♗×e6 25 ♖×a6 g6 26 c3 ♖b8 27 b4 ♖c8 28 ♖a3 ♗c4 29 ♗×c4 ♖×c4 30 g3 ♔g7 31 ♔e2 ♖e4+ 32 ♔d3 ♖e1 33 c4 ♖g1 34 ♔d4 ♖g2 35 b5 ♖b2 36 ♔e5 ♖c2 37 ♔d5 ♖b2 38 g4 ♔f6 39 ♖a6+ ♔g7 40 b6 ♖b3 41 h4 h5 42 g5 ♖d3+ 43 ♔c5 1-0

**Xenarbor — Awit** 1 d4 ♘f6 2 c4 g6 3 ♘c3 d5 4 ♗f4 ♗g7 5 e3 c6 6 c×d5 ♘×d5 7 ♘×d5 ♕×d5 8 ♘f3 ♗f5 9 ♗e2 0-0 10 h3 ♖d8 11 ♗c7 ♖c8 12 ♗×b8 ♖c×b8 13 0-0 ♖d8 14 ♕b3 ♕×b3 15 a×b3 ♗f6 16 ♖fc1 a5 17 ♘d2 ♗g5 18 ♖c5 ♖d5 19 ♖×d5 c×d5 20 b4 ♖c8 21 f4 ♗h4 22 ♘f3 ♗g3 23 ♗b5 a×b4 24 ♘e1 ♗e4 25 ♗d7 ♖c7 26 ♗g4 ♗×e1 27 ♖×e1 h5 28 ♗f3 ♗×f3 29 g×f3 ♖c2 30 b3 ♖b2 31 ♖c1 ♖×b3 32 ♖c7 ♖×e3 33 ♖×b7 ♖×f3 34 ♖b8+ ♔g7 35 ♖×b4 ♖×h3 36 ♔g2 ♖d3 37 ♔f2 ♔f6 38 ♔e2 ♖a3 39 ♖b2 ♔f5 40 ♔f1 ♖a4 41 ♖e2 ♖×d4 42 ♖e5+ ♔g4 43 ♖×e7 ♖×f4+ 44 ♔g1 f5 45 ♖e2 g5 46 ♖d2 d4 47 ♖c2 h4 48 ♔h2 ♔f3 49 ♔g1 ♔e3 50 ♖c7 d3 51 ♖c3 ♔e2 0-1

**ACM 1985 — Denver**

| | rate | perf | 1 | 2 | 3 | 4 | tot |
|---|---|---|---|---|---|---|---|
| 1 Hightech | 2200 | 2486 | 7+ | 4+ | 2+ | 5+ | 4 |
| 2 Bebe | 2100 | 2224 | 9+ | 5+ | 1— | 4+ | 3 |
| 3 Intelligent | 0 | 2005 | 6+ | 9+ | 5— | 7= | 2½ |
| 4 Phoenix | 0 | 1967 | 8+ | 1— | 7+ | 2— | 2 |
| 5 Cray Blitz | 2200 | 2045 | 10+ | 2— | 3+ | 1— | 2 |
| 6 Chaos | 1800 | 1790 | 3— | 7— | 8+ | 10+ | 2 |
| 7 Lachex | 0 | 1885 | 1— | 6+ | 4— | 3 | 1½ |
| 8 Spock | 0 | 1676 | 4— | 10+ | 6— | 9= | 1½ |
| 9 Ostrich | 1750 | 1633 | 2— | 3— | 10= | 8= | 1 |
| 10 Awit | 1600 | 1502 | 5— | 8— | 9= | 6— | ½ |

*Round 1, October 13, 1985*

**Awit — Cray Blitz** 1 c4 ♘f6 2 ♘c3 c6 3 ♘f3 d5 4 e3 ♗e6 5 d4 d×c4 6 ♘e5 b5 7 e4 b4 8 ♘a4 ♘×e4 9 ♘×c4 ♘d6 10 ♘×d6+ e×d6 11 ♗e2 ♗e7 12 0-0 0-0 13 ♗d2 a5 14 ♗d3 ♖e8 15 ♕h5 h6 16 ♖fe1 ♗f6 17 ♖e4 d5 18 ♖f4 ♗g5 19 ♖e1 ♗×f4 20 ♗×f4 ♗c8 21 ♖×e8 ♕×e8 22 ♗d2 ♖a7 23 ♘c5 ♘d7 24 ♘×d7 ♕×d7 25 ♕e5 ♕d8 26 ♕b8 ♕c7 27 ♕×c7 ♖×c7 28 h4 c5 29 ♗f4 ♖c6 30 ♗e5 c×d4 31 ♔f1 ♖c1+ 32 ♔e2 ♗g4+ 33 f3 ♗d7 34 ♗×d4 ♖a1 35 h5 a4 36 g4 a3 37 ♔d2 ♖×a2 38 ♔c2 ♖a1 39 b×a3 ♗a4+ 40 ♔b2 ♖d1 0-1

**Hitech — Lachex** 1 e4 e5 2 ♘f3 ♘c6 3 d4 e×d4 4 ♘×d4 ♗c5 5 ♗e3 ♕f6 6 c3 ♘ge7 7 ♕d2 0-0 8 f4 d6 9 ♘b5 ♗×e3 10 ♕×e3 ♗g4 11 ♘×c7 ♖ac8 12 ♘b5 ♘d5 13 ♕g3 ♕×f4 14 ♕×f4 ♘×f4 15 h3 ♗e6 16 ♘×d6 ♖b8 17 g3 ♘h5 18 g4 ♘g3 19 ♖h2 ♖fd8 20 ♖d2 ♘×f1 21 ♔×f1 ♖d7 22 ♘f5 ♖bd8 23 ♖×d7 ♖×d7 24 ♔g2 ♖d3 25 b3 g6 26 ♘g3 f6 27 ♘h1 ♔g7 28 ♘f2 ♖e3 29 c4 ♘b4 30 ♘a3 ♖e2 31 ♔f3 ♖×a2 32 ♖×a2 ♘×a2 33 ♘d3 ♘c3 34 ♘f4 ♗d7 35 ♘c2 g5

36 ♘d5 ♘×d5 37 c×d5 h5 38 g×h5 ♗×h3 39 ♘d4 ♔h6 40 ♘f5+ ♗×f5 41 e×f5 ♔g7 42 h6+ ♔f8 43 d6 a6 44 h7 ♔g7 45 d7 ♔×h7 46 d8♕ ♔g7 47 ♕e7+ ♔g8 48 ♔g4 a5 49 ♔h5 b6 50 ♔g6 g4 51 ♕e8# 1-0

**Ostrich – Bebe** 1 e4 c5 2 c3 d5 3 e×d5 ♕×d5 4 d4 e6 5 ♘f3 ♘c6 6 d×c5 ♕×d1+ 7 ♔×d1 ♗×c5 8 ♗e3 ♗×e3 9 f×e3 ♘f6 10 ♗b5 0-0 11 ♗×c6 b×c6 12 ♔c1 ♘g4 13 ♖e1 ♖d8 14 h3 ♘f2 15 ♔c2 ♗a6 16 b3 ♗d3+ 17 ♔c1 ♗e4 18 ♖f1 ♘d3+ 19 ♔c2 ♘b4+ 20 ♔b2 ♘c2 21 ♘g5 ♗×g2 22 ♖f2 ♘×e3 23 ♖e2 ♖d3 24 ♘d2 h6 25 ♘ge4 ♗×h3 26 ♔b1 ♖ad8 27 ♘f2 ♗f5 28 ♘de4 ♖d2 29 ♖×d2 ♖×d2 30 ♘h3 ♗×e4+ 0-1

**Chaos – Intelligent** 1 d4 ♘f6 2 c4 g6 3 ♘c3 d5 4 c×d5 ♘×d5 5 e4 ♘×c3 6 b×c3 ♗g7 7 ♗c4 0-0 8 ♘e2 ♘c6 9 ♖b1 a6 10 0-0 e6 11 ♗a3 ♖e8 12 ♕a4 ♕h4 13 f3 ♗h6 14 f4 ♕g4 15 g3 ♖b8 16 ♕×c6 b×c6 17 ♖×b8 a5 18 ♖fb1 ♕f3 19 e5 ♕e3+ 20 ♔g2 ♕e4+ 21 ♔f2 ♗f8 22 ♗×f8 ♖×f8 23 a3 c5 24 ♖1b2 c×d4 25 ♘×d4 ♗d7 26 ♗e2 ♖×b8 27 ♖×b8+ ♔g7 28 ♖d8 ♗a4 29 ♗f3 ♕b1 30 ♘e2 ♕b2 31 ♖c8 ♕b6+ 32 ♔g2 ♗c6 33 ♖d8 ♗×f3+ 34 ♔×f3 ♕c5 35 ♖c8 h5 36 ♔g2 ♕e3 37 ♔f1 ♕f3+ 38 ♔e1 ♕b7 39 ♖d8 c5 40 ♔f2 a4 41 ♔e3 ♕b3 42 ♖d7 ♕×a3 43 ♖c7 ♕b2 44 ♖c8 a3 45 ♖a8 a2 46 ♖a7 a1♖ 47 ♖×a1 ♕×a1 48 ♔d3 ♕h1 0-1

**Spock – Phoenix** 1 e4 e6 2 d4 d5 3 ♘c3 ♘c6 4 e×d5 e×d5 5 ♕e2+ ♗e6 6 ♕b5 a6 7 ♕d3 ♕f6 8 ♗e3 ♘b4 9 ♕d2 ♗f5 10 0-0-0 ♗×c2 11 ♖e1 ♕f5 12 a3 ♘d3+ 13 ♗×d3 ♗×d3 14 ♗f4+ ♘e7 15 ♗×c7 ♖c8 16 ♖e5 ♕g4 17 ♘×d5 f6 18 ♖e3 ♕×g2 19 ♕×d3 ♕×h1 20 ♔d2 ♕×d5 21 ♗b6 ♔f7 22 ♖e1 ♕a2 23 ♕b1 ♕×b1 24 ♖×b1 ♘d5 25 ♗a7 ♖a8 26 ♗c5 ♗×c5 27 d×c5 ♖hd8 28 ♘f3 ♖ac8 29 b4 ♘×b4+ 30 ♔e2 ♖e8+ 31 ♔d1 ♘d3 32 ♖×b7+ ♔f8 33 ♔d2 ♘×c5 34 ♖b2 ♖cd8+ 35 ♔c2 ♖e2+ 36 ♔b1 ♖d1+ 37 ♔a2 ♖×b2+ 38 ♔×b2 ♖f1 39 ♘d4 ♖×f2+ 40 ♔c3 ♖×h2 41 ♔c4 0-1

*Round 2, October 13, 1985*

**Cray Blitz – Bebe** 1 e4 c5 2 ♘f3 d6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 g6 6 ♗g5 ♗g7 7 ♕d2 ♘c6 8 0-0-0 0-0 9 ♘b3 ♖e8 10 ♗c4 ♘g4 11 h3 ♘ge5 12 ♗b5 a6 13 ♗e2 a5 14 ♗b5 ♗e6 15 ♘d5 a4 16 ♘d4 ♗d7 17 ♘×c6 b×c6 18 ♘×e7+ ♖×e7 19 ♗×e7 ♕×e7 20 ♗e2 ♕e6 21 ♔b1 ♖b8 22 b3 a×b3 23 c×b3 ♗e8 24 ♔c2 ♘d7 25 f3 ♖a8 26 ♔c1 ♘c5 27 ♕c2 ♕f6 28 ♗c4 ♕a1+ 29 ♔d2 ♕×a2 30 ♕×a2 ♖×a2+ 31 ♔c1 d5 32 e×d5 c×d5 33 ♗×d5 ♗b5 34 ♖he1 ♘d3+ 35 ♖×d3 ♗×d3 36 ♖e8+ ♗f8 37 g4 ♔g7 38 ♖e3 ♗a3+ 39 ♔d1 ♖a1+ 40 ♔d2 ♗f1 41 ♔c3 ♖c1+ 42 ♔d2 ♖c5 43 ♔e1 ♗×h3 44 ♗c4 h5 45 g×h5 g×h5 46 ♔f2 h4 47 ♖d3 ♗f5 48 ♖d4 h3 49 ♖h4 ♖c7 50 ♖h5 0-1

**Phoenix – Hitech** 1 d4 d5 2 ♗g5 ♘f6 3 ♗×f6 e×f6 4 e3 ♗f5 5 c4 ♗×b1 6 ♕×b1 ♗b4+ 7 ♔d1 ♗e7 8 c×d5 ♕×d5 9 ♘f3 ♘d7 10 ♗d3 h6 11 ♗e4 ♕b5 12 ♕c2 c6 13 ♗d3 ♕b6 14 ♘d2 ♕c7 15 ♖c1 a5 16 ♗c4 0-0 17 ♕f5 a4 18 ♗d3 g6 19 ♕g4 f5 20 ♕g3 ♕d8 21 ♖f1 a3 22 b3 ♗f6 23 ♘c4 ♗h4 24 ♕f4 ♗g5 25 ♕d6 c5 d5 ♖a6 27 ♕g3 ♘f6 28 ♘d2 ♖d6 29 ♖×c5 ♘×d5 30 ♖c8 ♕×c8 31 ♕×d6 ♘c3+ 32 ♔e1 ♖d8 33 ♕×a3 ♕d7 34 f3 ♕×d3 0-1

**Intelligent – Ostrich** 1 ♘f3 d6 2 d4 ♘f6 3 ♘c3 d5 4 ♗f4 ♘h5 5 e3 ♘×f4 6 e×f4 g6 7 ♗b5+ c6 8 ♗a4 ♕d6 9 ♘e5 ♗h6 10 g3 0-0 11 ♕e2 f6 12 ♘d3 ♗h3 13 ♗b3 ♘d7 14 ♘e4 ♕c7 15 ♘ec5 ♘×c5 16 ♘×c5 ♔h8 17 ♖d1 b5 18 ♘e6 ♗×e6 19 ♕×e6 ♕c8 20 ♕×c8 ♖f×c8 21 0-0 e6 22 ♖fe1 ♖e8 23 a4 ♔g8 24 ♖e2 b×a4 25 ♗×a4 ♖ac8 26 ♖de1 ♔f7 27 c3 ♖c7 28 b4 ♗f8 29 ♗c2 ♗d6 30 ♗d3 ♖ce7 31 ♖a2 ♗b8 32 ♖a6 ♖c8 33 ♖ea1 ♔g8 34 ♖6a3 ♖d8 35 ♖a4 ♖d6 36 h4 ♖d8 37 ♖a6 ♖d6 38 ♖6a5 ♔h8 39 ♗e2 ♖d8 40 ♖a6 ♖c7 41 h5 ♔g8 42 h×g6 h×g6 43 ♔g2 ♖dd7 44 ♗d3 f5 45 ♖e1 ♔f7 46 b5 ♖d6 47 ♖b1 ♔g8 48 b×c6 ♖d8 49 c4 ♔f7 50 c×d5 e×d5 51 ♗e2 ♖e7 52 ♗f3 ♗c7 53 ♖×a7 ♔f6 54 ♖bb7 ♖c8 55 ♗×d5 ♖h7 56 ♗c4 ♖g7 57 f3 ♖h7 58 g4 f×g4 59 f×g4 ♖g7 60 g5+ ♔e7 61 ♖×c7+ 1-0

**Lachex – Chaos** 1 e4 c5 2 d4 c×d4 3 c3 ♘f6 4 e5 ♘d5 5 ♘f3 ♘c6 6 c×d4 d6 7 ♗c4 ♘b6 8 ♗b5 e6 9 0-0 ♗e7 10 ♕c2 ♗d7 11 e×d6 ♗×d6 12 h3 a6 13 ♗×c6 ♗×c6 14 ♘c3 0-0 15 ♕d3 f6 16 ♗d2 ♕d7 17 ♖fe1 ♗c7 18 ♘e4 ♕d5 19 ♗b4 ♖fd8 20 b3 ♕f5 21 g4 ♕g6 22 ♗c5 ♘d7 23 ♗e7 ♖dc8 24 ♗b4 ♕h6 25 ♔g2 ♕g6 26 ♘h4 ♕h6 27 ♘f3 ♕g6 28 ♘h4 ♕f7 29 ♘f3 e5 30 ♖ac1 a5 31 ♗d2 e×d4 32 ♘×d4 ♘e5 33 ♕e2 ♗d5 34 ♘b5 b6 35 ♘×c7 ♖×c7 36 ♔g3 ♖×c1 37 ♖×c1 ♕e6 38 ♖e1 ♖e8 39 ♘c3 ♗b7 40 f4 ♖d8 41 f×e5 ♕×e5+ 42 ♕×e5 f×e5 43 ♗e3 ♖f8 44 ♔h4 ♖f3 45 ♘b5 ♖f6 46 ♗g5 ♖e6 47 ♘d4 ♖e8 48 ♗f4 e4 49 ♗c7 a4 50 ♗×b6 a×b3 51 a×b3 ♗d5 52 b4 e3 53 ♘f5 e2 54 ♗c5 ♖e5 55 ♘e7+ ♖×e7 56 ♗×e7 ♗c4 57 ♔g3 ♔f7 58 ♗c5 ♗b5 59 ♖a1 ♔e6 60 ♔f2 ♔d5 61 ♖a7 g5 1-0

**Awit – Spock** 1 c4 e6 2 ♘c3 d5 3 c×d5 e×d5 4 d4 ♗b4 5 e3 ♗×c3+ 6 b×c3 ♘e7 7 ♖b1 0-0 8 ♗a3 ♖e8 9 ♗d3 c6 10 ♕h5 ♘g6 11 ♘f3 ♘d7 12 ♖b3 ♘f6 13 ♕g5 h6 14 ♕g3 ♘h5 15 ♕d6 ♘gf4 16 ♗f1 ♕f6 17 ♕×f6 ♘×f6 18 ♘e5 ♘g6 19 ♘×g6 f×g6 20 ♗d3 ♖e6 21 ♗×g6 ♘g4 22 ♗f5 ♖f6 23 ♗×g4 ♗×g4 24 ♖×b7 ♗c8 25 ♖b3 ♗a6 26 ♗c5 ♗d3 27 f3 ♖e8 28 ♔d2 ♗c4 29 ♖b7 ♖fe6 30 ♖e1 ♖g6 31 ♖g1 ♖ge6 32 ♖e1 ♖g6 33 ♖g1 ♖ge6 34 ♖×a7 ♖×e3 35 a4 ♖d3+ 36 ♔c1 ♖×c3+ 37 ♔b2 ♖ce3 38 a5 ♖b8+ 39 ♔c1 ♖eb3 40 ♖e1 ♖b1+ 41 ♔d2 ♖8b2+ 42 ♔c3 ♖b3+ 43 ♔d2 ♖d3+ 44 ♔c2 ♖×e1 45 ♖a8+ ♔h7 46 ♖c8 ♖e2+ 47 ♔c1 ♗b3 48 ♖h8+ 0-1

*Round 3, October 14, 1985*

**Bebe – Hitech** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗a4 ♘f6 5 0-0 b5 6 ♗b3 ♗b7 7 ♖e1 ♗c5 8 c3 d6 9 d4 ♗b6 10 a4 h5 11 a×b5 a×b5 12 ♖×a8 ♕×a8 13 ♘a3 e×d4 14 c×d4 ♗a6 15 e5 d×e5 16 d×e5 ♘g4 17 ♗×f7+ ♔e7 18 ♔f1 b4+ 19 ♘c4 ♖d8 20 ♕c2 ♔×f7 21 ♕f5+ ♘f6 22 ♕c2 b3 23 ♕e2 ♘d4 24 ♘×d4 ♖×d4 25 ♔g1 ♗×c4 26 ♕f3 ♕×f3 27 g×f3 0-1

**Cray Blitz – Intelligent** 1 e4 c6 2 d4 d5 3 e5 ♗f5 4 ♗d3 ♗×d3 5 ♕×d3 e6 6 ♘c3 ♘a6 7 ♘f3 ♗e7 8 ♗f4 g5 9 ♗e3 g4 10 ♘g1 ♘b4 11 ♕e2 h5 12 h3 f5

13 e×f6 ♘×f6 14 a3 ♘a6 15 h×g4 ♕b6 16 0-0-0 ♘×g4 17 ♖×h5 ♖×h5 18 ♕×g4 ♖f5 19 ♕g8+ ♖f8 20 ♕×e6 ♖f6 21 ♕g8+ ♖f8 22 ♕g6+ ♔d7 23 ♕g4+ ♔d8 24 ♘f3 ♖b8 25 ♘e5 ♕c7 26 ♗h6 ♕c8 27 ♗×f8 ♗×f8 28 ♕f4 ♗e7 29 ♖e1 ♗×a3 30 b×a3 ♕e6 31 ♘f7+ ♔d7 32 ♖×e6 ♔×e6 33 ♘e5 ♔d6 34 ♕f7 ♖g8 35 ♘c4+ d×c4 36 ♘e4 # 1-0

**Lachex – Phoenix** 1 e4 e6 2 d4 d5 3 ♘c3 ♘c6 4 e×d5 e×d5 5 ♘f3 ♗f5 6 ♗b5 ♗b4 7 0-0 ♘ge7 8 a3 ♗×c3 9 b×c3 0-0 10 ♘h4 ♗e6 11 ♗f4 ♖e8 12 ♕h5 a6 13 ♗d3 ♘g6 14 g3 ♕d7 15 f3 ♘a5 16 ♘×g6 h×g6 17 ♕h4 ♕c6 18 ♗d2 ♘c4 19 ♗×c4 ♕×c4 20 ♖ab1 b5 21 ♕f4 ♗f5 22 ♖b2 ♗h3 23 ♖f2 ♖e2 24 g4 ♖ae8 25 ♖b1 ♖×f2 26 ♔×f2 ♕e2+ 27 ♔g3 ♕g2+ 28 ♔h4 f6 29 ♕×c7 g5+ 30 ♗×g5 f×g5+ 31 ♔×g5 ♕×f3 32 ♕f4 ♗×g4 33 ♕×f3 ♗×f3 34 a4 b×a4 35 ♖b6 ♗e2 36 ♔g6 ♖c8 37 c4 ♗×c4 38 ♖b7 ♖c6+ 39 ♔g5 ♗b5 40 ♖d7 a3 41 ♖e7 a2 42 ♖e8+ ♔f7 43 ♖e1 0-1

**Spock – Chaos** 1 d4 ♘f6 2 c4 e6 3 ♘f3 b6 4 ♗f4 ♗b4+ 5 ♗d2 ♘c6 6 ♗×b4 ♘×b4 7 ♕d2 a5 8 ♘c3 ♗b7 9 0-0-0 ♘e4 10 ♘×e4 ♗×e4 11 b3 ♕f6 12 ♔b2 h6 13 ♕e3 ♕f5 14 g4 ♕×g4 15 ♖g1 ♕f5 16 ♘h4 ♕h7 17 ♕g3 g5 18 ♕×c7 g×h4 19 a3 ♘c6 20 f3 ♗g6 21 e4 ♖b8 22 ♕f4 ♕g7 23 e5 a4 24 b×a4 ♖a8 25 ♗d3 h3 26 ♗b1 ♖×a4 27 ♗c2 ♖×c4 28 ♗b3 d5 29 ♗×c4 d×c4 30 ♖g3 ♘e7 31 ♕c1 b5 32 ♖×h3 ♘d5 33 ♔a1 ♔e7 34 ♕b2 b4 35 ♕c1 c3 36 ♖g3 b3 37 ♖g2 ♖b8 38 ♖dd2 c×d2 39 ♕c5+ ♔e8 40 ♕c6+ ♔d8 41 ♕d6+ ♔c8 42 ♕c6+ ♘c7 43 ♖×d2 ♗c2 44 ♔b2 ♕g1 45 ♕b7+ ♖×b7 46 a4 ♕a1+ 47 ♔×a1 b2+ 48 ♔a2 b1♕+ 49 ♔a3 ♖b3# 0-1

**Ostrich – Awit 1** 1 e4 c5 2 c3 ♘f6 3 e5 ♘d5 4 d4 c×d4 5 c×d4 ♘c6 6 ♘f3 d6 7 ♗c4 ♘b6 8 ♗b5 e6 9 0-0 ♗d7 10 ♘c3 d×e5 11 d×e5 ♗c5 12 ♗×c6 ♗×c6 13 ♕×d8+ ♖×d8 14 ♗g5 ♖d7 15 ♖fb1 h6 16 ♗h4 ♘c4 17 b3 ♗×f3 18 b×c4 ♗c6 19 ♘e2 ♖d2 20 ♖d1 ♖×d1+ ♖×d1 g5 22 ♗g3 0-0 23 h3 f6 24 e×f6 ♖×f6 25 h4 ♔f7 26 h5 e5 27 ♘c1 e4 28 ♘b3 ♖f5 29 ♘×c5 ♖×c5 30 ♖d6 ♔g7 31 ♖g6+ ♔h7 32 ♗d6 ♖×c4 33 ♗e5 ♖c1+ 34 ♔h2 a6 35 ♖g7+ ♔h8 36 ♖×g5+ ♔h7 37 ♖g7+ ♔h8 38 ♖×b7+ ♔g8 39 ♖g7+ ♔f8 40 ♖c7 ♔g8 41 ♗f4 ♖c5 42 g4 ♖c2 43 ♔g1 a5 44 a3 ♗a4 45 ♖b7 ♖c6 46 ♖b8+ ♔f7 47 ♖b7+ ♔g8 48 ♖b8+ ♔f7 49 ♖b7+ ♔g8 ½-½

*Round 4, October 15, 1985*

**Hitech – Cray Blitz** 1 e4 e5 2 ♘f3 ♘c6 3 d4 e×d4 4 ♘×d4 ♘f6 5 ♘×c6 b×c6 6 ♗d3 d5 7 ♕e2 ♗g4 8 f3 ♗e6 9 e×d5 ♘×d5 10 ♗f5 ♕h4+ 11 ♔f1 ♕f6 12 ♗×e6 ♕×e6 13 c4 ♕×e2+ 14 ♔×e2 ♘b6 15 b3 ♗d6 16 ♘c3 0-0 17 ♗e3 ♗e5 18 ♖ac1 ♖fe8 19 ♔f2 ♖ad8 20 f4 ♗f6 21 ♖hd1 ♖×d1 22 ♘×d1 h5 23 ♘c3 ♔h7 24 ♘e2 ♖d8 25 ♔f3 ♔g6 26 ♘g3 h4 27 f5+ ♔h7 28 ♘e4 ♗e7 29 ♔g4 ♖e8 30 ♗f2 ♘d7 31 ♔h3 a6 32 ♖d1 ♘f6 33 ♖e1 ♘×e4 34 ♖×e4 ♔g8 35 ♗d4 ♔f8 36 c5 f6 37 ♖×h4 ♖d8 38 ♖h8+ ♔f7 39 ♖×d8 ♗×d8 40 ♔g4 ♗e7 41 h4 ♔e8 42 ♔f4 ♗d8 43 g4 ♗e7 44 ♔e4 ♗d8 45 ♗e3 ♗e7 46 a4 ♔d8 47 ♔d4 ♔c8 48 g5 f×g5 49 h×g5 ♔d8 50 ♔e5 ♔d7 51 f6 ♗f8 52 a5 g6 53 ♗d4 1-0

**Bebe – Phoenix** 1 e4 e6 2 d4 d5 3 ♘d2 ♘f6 4 e5 ♘g8 5 ♘gf3 c5 6 d×c5 ♗×c5 7 ♗b5+ ♗d7 8 ♗×d7+ ♘×d7 9 0-0 ♗b6 10 c4 ♘e7 11 c×d5 e×d5 12 ♕b3 ♖c8 13 ♖d1 0-0 14 ♘f1 ♘c5 15 ♕b5 a6 16 ♕e2 ♘e6 17 ♘g3 f6 18 e×f6 ♖×f6 19 a3 ♕d7 20 ♘e4 ♖f5 21 ♕d2 ♕c6 22 ♘g3 ♖f6 23 ♕e1 ♕c2 24 ♗d2 ♖c4 25 ♖ac1 ♕a4 26 ♘h5 ♖g6 27 ♖×c4 d×c4 28 ♘e5 ♘d4 29 ♘×g6 h×g6 30 ♘×g7 ♕c2 31 ♘e6 ♘b3 32 ♘g5 ♘d4 33 ♖c1 ♕d3 34 ♕×e7 1-0

**Intelligent – Lachex** 1 ♘f3 d5 2 c4 c6 3 b3 ♘f6 4 ♗b2 d×c4 5 b×c4 e6 6 ♗d4 ♗e7 7 ♕a4 0-0 8 ♘c3 c5 9 ♗×f6 ♗×f6 10 ♕b5 ♗×c3 11 d×c3 ♕f6 12 ♖c1 b6 13 e3 ♗b7 14 ♗e2 ♗c6 15 ♕b1 ♖d8 16 ♗d3 g6 17 ♗e4 ♕g7 18 ♘g5 h6 19 ♘f3 f5 20 ♗×c6 ♘×c6 21 0-0 ♘e5 22 ♘×e5 ♕×e5 23 ♖fd1 ♖×d1+ 24 ♖×d1 ♕×c3 25 ♕b5 ♕c2 26 ♕b3 ♕e2 27 ♕a4 ♖f8 28 g3 ♖f7 29 ♖d8+ ♔g7 30 ♕b3 e5 31 ♕c3 ♖e7 32 a3 ♔f7 33 ♖d5 ♕f3 34 a4 ♕e4 35 ♕b3 h5 36 a5 h4 37 a×b6 a×b6 38 ♕d3 h×g3 39 h×g3 ♕×d3 40 ♖×d3 e4 41 ♖d6 ♖e6 42 ♖d7+ ♖e7 43 ♖d6 ♖e6 44 ♖d7+ ♖e7 45 ♖d6 ½-½

**Ostrich – Spock** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗c6 d×c6 5 d4 e×d4 6 ♕×d4 ♕×d4 7 ♘×d4 ♘f6 8 0-0 ♗c5 9 c3 ♘×e4 10 ♖e1 f5 11 f3 0-0 12 f×e4 ♖d8 13 e×f5 ♗×f5 14 ♗e3 ♗×d4 15 ♗×d4 ♖d7 16 ♖c1 ♖e8 17 ♔f2 ♖de7 18 ♔f3 ♖f7 19 ♘d2 ♗e4+ 20 ♔e2 ♗×g2+ 21 ♔d3 ♗h3 22 ♖e1 ♗f5+ 23 ♔c4 ♗e6+ 24 ♔d3 ♗f5+ 25 ♔c4 ♗e6+ 26 ♔d3 ♖f4 27 ♗e5 ♖d8+ 28 ♔e3 ♖f7 29 ♘f3 ♗c4 30 ♘d2 ♖d3+ 31 ♔e2 ♖×c3+ 32 ♔d1 ♖d3 33 ♗c3 ♖f2 34 ♖g1 ♖f7 35 ♔c2 b5 36 b3 b4 37 ♗×b4 ♗b5 38 a4 ♖d4 39 a×b5 c×b5 40 ♗c3 ♖d6 41 ♖af1 ♖e7 42 ♗b4 c5 43 ♗×c5 ♖c6 44 b4 ♖e8 45 ♔d1 ♖d8 46 ♖f3 ♖cc8 47 ♖g4 ♖d5 48 ♔e1 ♖cd8 49 ♗e3 ♖d3 50 ♖e4 ♖c3 51 ♘b1 ♖b3 52 ♘d2 ♖b2 53 ♖e6 ♖×b4 54 ♖×a6 ♖h4 55 ♘f1 ♖e8 56 ♖a7 ♖he4 57 ♖g3 g6 58 ♖h3 h5 59 ♖a6 ♖g4 60 ♖g3 ♖×g3 61 h×g3 ♔f7 62 ♖b6 ♖e5 63 ♖b8 g5 64 ♖c8 b4 65 ♖b8 ♖e4 66 ♖b7+ ♔e6 67 ♔d2 ♖c4 68 ♗×g5 ♖d4+ 69 ♔c2 ♖d5 70 ♗f4 ♖d4 71 ♗e3 ♖e4 72 ♔d3 ♖g4 73 ♗d4 h4 74 g×h4 ♖f4 75 ♘e3 ♖×h4 76 ♖×b4 ♔f7 77 ♔e2 ♖e4 78 ♔e1 ♔g8 79 ♔d2 ♖h4 80 ♔e1 ♖e4 81 ♖c4 ♖e8 82 ♖c7 ♖e4 83 ♗c5 ♖e5 84 ♔d1 ♖g5 85 ♔e1 ♖e5 86 ♔d1 ♖g5 87 ♔e1 ♖e5 88 ♔d1 ½-½

**Chaos – Awit** 1 d4 ♘f6 2 c4 g6 3 ♘c3 d5 4 c×d5 ♘×d5 5 e4 ♘×c3 6 b×c3 c5 7 ♗c4 ♗g7 8 ♘e2 ♗g4 9 ♕b3 c×d4 10 ♗×f7+ ♔f8 11 f3 d3 12 f×g4 d×e2 13 ♔×e2 ♕c8 14 ♗e6 ♕×c3 15 ♕×b7 ♕c6 16 ♖f1+ ♔e8 17 ♗f7+ ♔d8 18 ♖d1+ ♕d7 19 ♗f4 ♗d4 20 ♖×d4 ♕d6 21 ♖×d6+ ♘d7 22 ♖×d7# 1-0

**ACM 1986 — Dallas**

| | rate | perf | 1 | 2 | 3 | 4 | 5 | tot |
|---|---|---|---|---|---|---|---|---|
| 1 Belle | 2200 | 2507 | 15+ | 2+ | 6+ | 4+ | 7+ | 5 |
| 2 Lachex | 0 | 2249 | 9+ | 1− | 13+ | 10+ | 4+ | 4 |
| 3 Novag | 0 | 1978 | 7− | 16+ | 12= | 8+ | 6+ | 3½ |
| 4 Bebe | 2100 | 2076 | 11+ | 12+ | 8+ | 1− | 2− | 3 |
| 5 Phoenix | 2000 | 1873 | 8− | 9− | 15+ | 13+ | 10+ | 3 |
| 6 Mephisto | 0 | 1910 | 13+ | 10= | 1− | 9+ | 3− | 2½ |
| 7 Challenger | 0 | 1967 | 3+ | 8− | 9= | 14+ | 1− | 2½ |
| 8 Recom | 0 | 1960 | 5+ | 7+ | 4− | 3− | 11= | 2½ |
| 9 Cyrus | 0 | 1968 | 2− | 5+ | 7= | 6− | 14+ | 2½ |
| 10 Fidelity | 0 | 1862 | 16+ | 6= | 14+ | 2− | 5− | 2½ |
| 11 Chiptest | 0 | 1761 | 4− | 14− | 16+ | 12+ | 8= | 2½ |
| 12 Merlin | 0 | 1676 | 14= | 4− | 3= | 11− | 15+ | 2 |
| 13 Vaxchess | 0 | 1463 | 6− | 15+ | 2− | 5− | 16+ | 2 |
| 14 Ostrich | 1750 | 1696 | 12= | 11+ | 10− | 7− | 9− | 1½ |
| 15 Waycool | 1000 | 967 | 1− | 13− | 5− | 16= | 12− | ½ |
| 16 Rex | 1000 | 969 | 10− | 3− | 11− | 15= | 13− | ½ |

*Round 1, November 2, 1986*

**Bebe — Chiptest** 1 e4 e6 2 d4 d5 3 ♘d2 d×e4 4 ♘×e4 ♕d5 5 ♘c3 ♗b4 6 ♘ge2 ♕d8 7 a3 ♗d6 8 g3 ♘f6 9 ♗g2 ♘c6 10 0-0 ♗d7 11 ♗g5 h6 12 ♗×f6 ♕×f6 13 ♘e4 ♕f5 14 d5 e×d5 15 ♘×d6+ c×d6 16 ♘f4 0-0-0 17 ♖e1 d4 18 ♗×c6 ♗×c6 19 ♕×d4 g5 20 ♕d3 ♕f6 21 ♘d5 ♕×b2 22 ♖ab1 ♕g7 23 ♘e7+ ♔c7 24 ♘×c6 ♖d7 25 ♕b5 b6 26 ♘×a7 ♕d4 27 ♕c6+ ♔b8 28 ♕×d7 ♕d5 29 ♘c6+ ♕×c6 30 ♕×c6 d5 31 ♖×b6+ ♔a7 32 ♕b7# 1-0

**Waycool — Belle** 1 e4 e5 2 ♘f3 ♘c6 3 c3 ♘f6 4 d4 ♘×e4 5 d5 ♘b8 6 ♘×e5 ♗c5 7 ♕g4 0-0 8 ♕×e4 d6 9 ♗d3 f5 10 ♕c4 b5 11 ♕×b5 ♕e7 12 0-0 d×e5 13 ♗g5 ♕d6 14 ♕b3 ♔h8 15 ♘d2 ♘d7 16 ♗c4 h6 17 ♗e3 ♗×e3 18 f×e3 ♖b8 19 ♕a3 ♕b6 20 ♖f3 ♕×b2 21 ♕×b2 22 ♖f2 ♖d8 23 ♘f3 ♖×f2 24 ♔×f2 e4 25 ♘d4 ♘e5 26 ♗b3 ♘g4+ 27 ♔e2 ♘×h2 28 c4 ♗a6 29 ♘b5 ♖d7 30 ♖d1 ♗×b5 31 c×b5 ♘g4 32 ♖d4 g5 33 ♖a4 ♘f6 34 ♖×a7 ♘×d5 35 a4 ♘c3+ 36 ♔f1 ♖d3 37 ♗c2 ♖d2 38 ♗b3 ♖b2 39 ♖a8+ ♔g7 40 ♖g8+ ♔h7 41 ♗e6 ♘d1 42 ♔g1 ♘×e3 43 g3 ♖g2+ 44 ♔h1 ♖×g3 45 ♖c8 ♔g6 46 ♖×c7 ♔f6 47 ♖c6 ♔e5 48 b6 ♖h3+ 49 ♔g1 ♖f3 50 b7 ♖f1+ 51 ♔h2 ♖b1 52 ♗c8 g4 53 ♔g3 f4+ 54 ♔f2 g3+ 55 ♔e2 ♖b2+ 56 ♔e1 g2 57 ♖g6 ♔d4 0-1

**Phoenix — Recom** 1 d4 c6 2 e4 d5 3 e5 ♗f5 4 ♘c3 e6 5 g4 ♗g6 6 ♗e3 ♗b4 7 ♗g2 h5 8 g5 c5 9 d×c5 ♘d7 10 c6 ♗×c3+ 11 b×c3 b×c6 12 ♘f3 ♕a5 13 ♔d2 ♘e7 14 ♕b1 c5 15 ♕e1 d4 16 ♘×d4 c×d4 17 ♗×a8 ♕×c3+ 18 ♔e2 ♕c4+ 19 ♔f3 ♕c8 20 ♗e4 ♘×e5+ 21 ♔g3 ♗×e4 0-1

**Novag — Challenger** 1 a3 a5 2 e4 ♘c6 3 d4 d6 4 ♗b5 ♗d7 5 ♘f3 ♘f6 6 ♘bd2 e5 7 c3 e×d4 8 c×d4 d5 9 e5 ♘h5 10 ♘b1 ♘e7 11 ♗×d7+ ♕×d7 12 ♘c3 ♘c6 13 0-0 h6 14 h3 g6 15 ♕b3 0-0-0 16 ♖e1 ♘g7 17 ♗f4 ♗e7 18 ♗d2 ♘e6 19 ♕b5 ♘c×d4 20 ♘×d4 ♘×d4 21 ♕×a5 ♔b8 22 ♖ad1 ♘b3 23 ♕b5 ♕×b5 24 ♘×b5 ♘×d2 25 ♖×d2 c6 26 ♘d4 c5 27 ♘f3 d4 28 e6 f6 29 ♘h4 ♖hg8 30 g3 d3 31 ♖ed1 c4 32 ♘g2 ♖ge8 33 ♘h4 ♗d6 34 ♘×g6 ♖×e6 35 ♘h4 ♖de8 36 ♘f5 ♖e1+ 37 ♔g2 ♖×d1 38 ♖×d1 ♗c5 39 b4 c×b3 40 ♖×d3 ♖e2 41 ♔f3 b2 42 ♖d8+ ♔a7 43 ♖d1 ♖×f2+ 44 ♔e4 ♖e2+ 45 ♔f4 ♖c2 46 ♖b1 ♖c4+ 47 ♔f3 ♗×a3 48 ♘d6 ♗×d6 49 ♖×b2 ♖c3+ 50 ♔e4 ♗×g3 51 ♔f5 ♗e5 52 ♖a2+ ♔b8 53 h4 b5 54 ♔g6 ♗f4 55 ♔×f6 b4 56 ♖b2 b3 57 ♔e6 ♗c1 58 ♖b1 ♖e3+ 59 ♔f7 b2 60 ♔f6 0-1

**Mephisto — Vaxchess** 1 c4 e5 2 ♘c3 ♘c6 3 ♘f3 ♘f6 4 e4 ♗c5 5 ♘×e5 ♘×e5 6 d4 ♗×d4 7 ♕×d4 d6 8 ♗e2 c5 9 ♕d2 ♗e6 10 b3 0-0 11 0-0 ♘c6 12 ♖d1 ♘d4 13 ♕d3 ♖e8 14 ♗e3 ♘×e2+ 15 ♘×e2 ♕e7 16 ♘c3 ♘g4 17 ♗f4 ♘e5 18 ♕×d6 ♕×d6 19 ♖×d6 ♘g6 20 ♗e3 ♖ac8 21 ♖ad1 ♘e5 22 h3 f6 23 f4 ♘f7 24 ♖6d3 b6 25 f5 ♘e5 26 f×e6 ♘×d3 27 ♖×d3 ♖×e6 28 ♖d7 a6 29 ♔f2 ♖cc6 30 ♗f4 g5 31 ♗c7 h5 32 ♔f3 ♔f8 33 g3 ♔e8 ♖h7 ♔f8 35 ♘d5 b5 36 h4 g×h4 37 g×h4 b×c4 38 b×c4 ♔g8 39 ♖d7 ♔f8 40 ♘f4 ♔e8 41 ♖d8+ ♔f7 42 ♘×e6 ♔×e6 43 ♗f4 a5 44 ♖d5 ♔f7 45 ♖×h5 ♔e6 46 ♗e3 ♖b6 47 ♖×c5 1-0

**Ostrich — Merlin** 1 e4 e5 2 ♘f3 ♘f6 3 ♘×e5 d6 4 ♘f3 ♘×e4 5 ♕e2 ♕e7 6 d3 ♘f6 7 ♗g5 ♕×e2+ 8 ♗×e2 ♗e7 9 ♘c3 c6 10 ♘d4 ♘a6 11 0-0 ♘c5 12 b4 ♘e6 13 ♘×e6 ♗×e6 14 ♗f3 ♔d7 15 b5 ♘d5 16 ♗×d5 ♗×g5 17 b×c6+ b×c6 18 ♗×e6+ f×e6 19 ♖ab1 ♖ab8 20 ♘e4 ♗f4 21 g3 ♗e5 22 ♔g2 ♖b2 23 ♖×b2 ♗×b2 24 ♖e1 ♖b8 25 ♘g5 ♗c3 26 ♖e2 e5 27 ♘×h7 ♖b2 28 a4 ♔c8 29 ♘g5 ♔d7 30 f4 e×f4 31 ♘h7 f×g3 32 ♘f8+ ♔c7 33 h×g3 ♔b6 34 ♔f3 ♔a5 35 ♘g6 ♖a2 36 ♘e7 ♔b6 37 ♖e4 ♗f6 38 ♘f5 d5 39 ♖b4+ ♔a5 40 ♖b7 a6 41 ♘×g7 ♖×c2 42 ♘e6 ♗e5 43 g4 ♔×a4 44 g5 ♖c1 45 g6 ♖g1 46 d4 ♗d6 47 g7 ♖g3+ 48 ♔e2 ♖g2+ 49 ♔e3 ♖g6 50 ♘f8 ♖g3+ 51 ♔f2 ♖×g7 52 ♖×g7 ♗×f8 53 ♖c7 ♔b5 54 ♖b7+ ♔c4 55 ♖c7 ♔×d4 56 ♖×c6 a5 ½-½

**Lachex — Cyrus** 1 e4 c5 2 c3 d5 3 e×d5 ♕×d5 4 d4 e6 5 ♘f3 ♘f6 6 ♗b5+ ♘c6 7 0-0 c×d4 8 c4 ♕d6 9 ♘×d4 ♗d7 10 ♘×c6 ♗×c6 11 ♘c3 ♕c7 12 ♕f3 ♗d6 13 ♗×c6+ b×c6 14 ♘b5 ♗×h2+ 15 ♔h1 ♕c8 16 ♘×a7 ♖×a7 17 ♔×h2 0-0 18 ♔g1 ♕b7 19 a4 ♖a5 20 ♗d2 ♖f5 21 ♕a3 ♖b8 22 b4 ♘e4 23 ♗e1 ♕c7 24 a5 ♕f4 25 a6 ♘d2 26 a7 ♖a8 27 ♗×d2 ♗×d2 28 ♖ad1 ♕g5 29 b5 c×b5 30 ♕d6 ♖c8 31 ♕b8 ♕×g2+ 1-0

**Rex — Fidelity** 1 e4 c5 2 c3 ♘c6 3 d4 c×d4 4 c×d4 d5 5 e×d5 ♕×d5 6 ♘f3 e5 7 ♘c3 ♗b4 8 ♕d2 ♗×c3 9 b×c3 e×d4 10 c×d4 ♘ge7 11 ♗e2 ♗f5 12 0-0 0-0 13 ♖d1 b5 14 ♕b2 a6 15 ♗f4 h6 16 ♖ac1 ♗e6 17 a3 ♕b3 18 ♕×b3 ♗×b3 19 ♖d3 ♗c4 20 ♖d2 ♘a5 21 ♖cd1 ♘d5 22 ♗d6 ♖fe8 23 ♗d3 ♗×d3 24 ♖×d3 ♘c4 25 ♗c5 ♘b2 26 ♖3d2 ♘×d1 27 ♖×d1 f6 28 ♖d2 ♖e4 29 h4 g6 30 ♔h2 ♖ae8 31 ♗d6 ♔f7 32 ♗b4 ♘×b4 33 a×b4 ♖e2 34 ♔g3 h5 35 ♖×e2 ♖×e2 36 d5 ♖e4 37 ♔h3 ♖×b4 38 ♔g3 ♖g4+ 39 ♔h2 ♔e7 40 ♔g1 ♔d6 41 g3 ♔×d5 42 ♔f1 a5 43 ♔e2 a4 0-1

*Round 2, November 2, 1986*

**Vaxchess – Waycool** 1 d4 ♘f6 2 c4 e6 3 ♘c3 ♗b4 4 e3 b6 5 ♘ge2 ♗a6 6 a3 ♗e7 7 ♘f4 d5 8 c×d5 ♗×f1 9 ♔×f1 e×d5 10 b3 0-0 11 a4 ♗b4 12 ♕d3 g5 13 ♘fe2 ♘c6 14 ♕f5 h6 15 h4 ♘a5 16 h×g5 ♘e4 17 ♘×d5 ♘×g5 18 ♘×b4 c5 19 ♘d3 ♘×b3 20 ♖b1 ♘×c1 21 ♖×c1 c×d4 22 ♘×d4 ♔g7 23 ♕e5+ f6 24 ♘f5+ ♔g8 25 ♘×h6+ ♔h8 26 ♕b5 ♘h7 27 ♘f5 a6 28 ♕b3 ♕d7 29 ♘d4 ♕b7 30 ♖c6 ♖ab8 31 ♘f4 ♖fc8 32 ♘f5 ♖g8 33 ♖×f6 a5 34 ♖f7 1-0

**Rex – Novag** 1 e4 c5 2 c3 e6 3 d4 d5 4 e×d5 e×d5 5 ♘f3 ♗d6 6 ♗e3 c4 7 b3 c×b3 8 ♗b5+ ♘c6 9 a×b3 ♗d7 10 0-0 ♘f6 11 ♘a3 0-0 12 ♖e1 a6 13 ♗d3 ♖e8 14 ♘c2 ♕c7 15 h3 ♘a5 16 ♘e5 ♘×b3 17 ♘×d7 ♘×d7 18 ♖a2 ♕×c3 19 ♗f5 ♘f6 20 ♕b1 b5 21 ♔f1 ♘e4 22 ♔g1 ♕c4 23 ♕b2 a5 24 ♘a3 ♕c3 25 ♘c2 b4 26 h4 ♕×b2 27 ♖×b2 ♘bd2 28 ♗d7 ♘c4 29 ♗×e8 ♘×b2 30 ♗c6 ♖c8 31 ♗×d5 ♘f6 32 ♗b3 a4 33 ♗×f7+ ♔×f7 34 ♖e2 b3 35 ♖e1 a3 0-1

**Challenger – Recom** 1 e4 c6 2 d4 d5 3 e×d5 c×d5 4 ♗d3 ♘c6 5 c3 ♘f6 6 ♗f4 g6 7 ♘f3 ♗g7 8 ♘bd2 ♘h5 9 ♗e3 0-0 10 0-0 f5 11 ♘b3 ♕d6 12 ♖e1 f4 13 ♗d2 ♗g4 14 ♗e2 ♖ae8 15 ♘e5 ♗×e2 16 ♕×e2 e6 17 ♘f3 e5 18 d×e5 ♘×e5 19 ♘×e5 ♗×e5 20 ♕d3 ♘f6 21 ♘d4 ♘e4 22 a4 b6 23 ♘b5 ♕c5 24 ♖e2 a6 25 ♘d4 ♘×d2 26 b4 ♕c4 27 ♖×d2 ♗f6 28 b5 a5 29 ♕f3 ♖e4 30 ♕d3 ♖c8 31 ♕×c4 ♖×c4 32 f3 ♖e×d4 33 c×d4 ♗×d4+ 34 ♖×d4 ♖×d4 35 ♔f1 ♖d2 36 ♖e1 ♔f7 37 ♖c1 ♖a2 38 ♖c6 ♖×a4 39 ♖×b6 ♖b4 40 ♖b7+ ♔e6 41 g3 f×g3 42 h×g3 h5 43 b6 d4 44 ♔e2 ♔d5 45 ♖d7+ ♔c5 46 b7 a4 47 ♖c7+ ♔d5 48 ♖g7 a3 49 ♖×g6 ♖×b7 50 ♖h6 ♔c4 51 ♖c6+ ♔b3 52 ♖c1 a2 53 ♔d3 ♔a3 54 ♖a1 ♖b1 55 ♖×a2+ ♔×a2 56 ♔×d4 ♖g1 57 ♔d3 0-1

**Merlin – Bebe** 1 e4 c5 2 c3 d5 3 e×d5 ♕×d5 4 d4 e6 5 ♘f3 ♘c6 6 ♗e2 ♘f6 7 0-0 c×d4 8 c×d4 ♗e7 9 ♘c3 ♕a5 10 ♗f4 0-0 11 ♘e5 ♖d8 12 ♘×c6 b×c6 13 ♗e5 ♘d7 14 ♗f4 ♘f8 15 ♗e3 e5 16 ♗f3 e×d4 17 ♗×d4 ♗f6 18 ♗×c6 ♗×d4 19 ♗×a8 ♗×c3 20 ♕c2 ♗f6 21 ♗e4 ♗e6 22 ♖fd1 ♖b8 23 ♕d2 ♕e5 24 ♕d3 ♖×b2 25 ♗d5 ♕f4 26 ♕g3 ♕×g3 27 h×g3 ♗×d5 28 ♖×d5 ♖c2 29 ♖ad1 ♖×a2 30 ♖c1 a5 31 ♔f1 a4 32 ♖c8 a3 33 ♖a8 ♗c3 34 ♖b5 ♗d4 35 ♖f5 h6 36 g4 g6 37 ♖f3 ♗c5 38 ♖c8 ♖a1+ 39 ♔e2 ♗b4 40 ♖e3 a2 41 ♖a8 ♖e1+ 0-1

**Belle – Lachex** 1 e4 e5 2 ♗c4 ♘f6 3 d4 e×d4 4 ♘f3 ♗b4+ 5 c3 d×c3 6 b×c3 ♗c5 7 e5 ♕e7 8 ♗e3 ♘e4 9 ♕d3 ♗×e3 10 ♕×e3 ♘c5 11 0-0 0-0 12 ♖d1 ♘c6 13 ♗d5 b6 14 ♘d4 ♗b7 15 ♗×c6 d×c6 16 ♘f5 ♕e6 17 ♘d4 ♕g6 18 f4 ♕e4 19 ♕g3 ♘e6 20 ♘×e6 f×e6 21 ♘d7 ♖f7 22 ♖×f7 ♔×f7 23 ♘d2 ♕e2 24 ♘f3 ♕e3+ 25 ♔h1 ♔g8 26 ♖d1 ♔h8 27 ♖d7 ♖g8 28 ♖×c7 ♗a8 29 ♘g1 ♕×g3 30 h×g3 c5 31 ♖×a7 h6 32 ♖a6 ♖b8 33 ♔h2 h5 34 ♘f3 ♗e4 35 ♘g5 ♗d5 36 a4 ♗c4 37 ♖a7 ♔g8 38 a5 b×a5 39 ♖×a5 ♖c8 40 ♘e4 ♗d5 41 ♖×c5 ♖×c5 42 ♘×c5 ♔f7 43 ♔h3 ♔e7 44 ♘d3 ♔d7 45 ♔h4 g6 46 ♔g5 ♔e7 47 ♘b4 ♗e4 48 c4 ♔d7 49 c5 ♔c7 50 ♔f6 ♗f5 51 ♔e7 ♗g4 52 ♘c2 ♔c6 53 ♘d4+ ♔×c5 54 ♘×e6+ 1-0

**Fidelity – Mephisto** 1 d4 d5 2 c4 d×c4 3 ♘f3 c6 4 e4 b5 5 ♗f4 ♘f6 6 ♘bd2 ♗g4 7 ♗e3 e6 8 ♗e2 ♘bd7 9 0-0 ♕c7 10 h3 ♗h5 11 ♕c2 ♗b4 12 e5 ♗g6 13 ♕c1 ♘d5 14 ♘h4 ♗d3 15 ♗×d3 ♘×e3 16 f×e3 c×d3 17 ♘hf3 c5 18 a3 ♗×d2 19 ♕×d2 c4 20 ♖ac1 ♕b7 21 b3 ♖c8 22 ♕b4 ♕c6 23 a4 c3 24 a×b5 ♕e4 25 ♖×c3 ♕×e3+ 26 ♔h2 ♖×c3 27 ♕×c3 0-0 28 ♖d1 ♘b6 29 ♖×d3 ♕e2 30 ♕d2 ♕f1 31 ♕d1 ♕×d1 32 ♖×d1 ♖d8 33 ♖a1 ♖d7 34 ♖c1 f6 35 ♖c5 ♔f7 36 ♔g3 ♘d5 37 ♔h4 ♖b7 38 ♘d2 ♘f4 39 e×f6 g×f6 40 g3 ♘e2 41 ♘f3 ♖d7 42 g4 ♘×d4 43 ♘×d4 ♖×d4 44 ♖c7+ ♔g6 45 ♖×a7 ♖b4 46 ♖b7 e5 47 ♖b8 ♖×b3 48 b6 ♔g7 49 ♖b7+ ♔h6 50 ♖b8 ♔g6 51 b7 ♔g7 52 g5 f5 53 ♖e8 ♖b4+ 54 ♔h5 ♖×b7 55 ♖×e5 ♖f7 56 ♖e3 ♖d7 57 ♖f3 ♖f7 58 ♖f4 ♖f8 59 ♖f2 ♖f7 60 ♖f3 ♖f8 61 h4 ½-½

**Chiptest – Ostrich** 1 f4 d5 2 e3 ♘f6 3 ♘f3 ♗g4 4 b3 ♘bd7 5 ♗b2 ♘e4 6 ♗e2 ♘×f3 7 ♗×f3 e6 8 ♗×e4 d×e4 9 0-0 ♕h4 10 g4 0-0-0 11 ♘c3 f5 12 g×f5 e×f5 13 ♔h1 ♘c5 14 ♘b5 a6 15 ♘d4 ♕h3 16 ♖g1 ♖d5 17 ♘×f5 ♖×f5 18 ♗×g7 ♗×g7 19 ♖×g7 ♕f3+ 20 ♕×f3+ e×f3 21 ♖×c7+ ♔×c7 22 ♖f1 ♘e4 23 d3 ♘d2 24 ♖f2 ♖e8 25 ♖×d2 ♖×e3 26 ♔g1 ♖×f4 27 ♔f2 ♖e5 28 c3 ♖h5 29 ♔f1 f2 0-1

**Cyrus – Phoenix** 1 e4 e6 2 d4 d5 3 ♘c3 ♘c6 4 e5 ♗b4 5 ♗e2 f6 6 ♘f3 ♘ge7 7 e×f6 g×f6 8 ♗d2 ♘f5 9 a3 ♗f8 10 ♗d3 ♘c×d4 11 ♘×d4 ♘×d4 12 ♕h5+ ♔d7 13 0-0 f5 14 ♖fe1 h6 15 ♕g6 c5 16 ♘b5 ♔c6 17 ♘×d4+ c×d4 18 ♗×f5 ♗d6 19 ♗×e6 ♗×e6 20 ♖×e6 ♔c7 21 ♗×h6 ♕d7 22 f4 ♖ag8 23 ♕f6 ♗c5 ♕e5+ ♔c8 25 ♔h1 ♖e8 26 f5 a5 1-0

*Round 3, November 3, 1986*

**Recom – Bebe** 1 d4 ♘f6 2 g3 d5 3 ♗g2 e6 4 c4 d×c4 5 ♕a4+ ♘bd7 6 ♕×c4 c5 7 ♘f3 ♘b6 8 ♕d3 c×d4 9 0-0 ♗c5 10 ♖d1 0-0 11 e3 ♕d5 12 e×d4 ♖d8 13 ♗g5 ♗e7 14 ♘e5 ♕a5 15 ♘c3 h6 16 ♗e3 h5 17 a3 ♘bd5 18 ♘×d5 ♘×d5 19 b4 ♕a4 20 ♗f3 h4 21 ♗d2 h×g3 22 f×g3 b6 23 ♗h5 f5 24 ♗f7+ ♔f8 25 ♕e2 ♗f6 26 ♕h5 ♗d7 27 ♖e1 ♗×e5 28 d×e5 ♗e8 29 ♗×e8 ♕×e8 30 ♕h7 ♔f7 31 ♕h4 ♕b5 32 a4 ♕d3 33 ♕h5+ ♔g8 34 ♖ad1 ♕d4+ 35 ♔h1 ♘×b4 36 ♗×b4 ♕×b4 37 ♕g6 ♖e8 38 ♖a1 ♖ad8 39 ♕h5 ♖e7 40 ♖ad1 ♖ed7 41 ♖a1 ♕b3 42 ♔g2 ♖d2+ 43 ♔g1 ♕d5 44 ♕h3 ♖c8 45 ♖ac1 ♕d4+ 46 ♔h1 ♕e4+ 0-1

**Lachex – Vaxchess** 1 e4 c5 2 c3 d5 3 e×d5 ♕×d5 4 d4 e6 5 ♘f3 ♘f6 6 ♗b5+ ♗d7 7 ♗×d7+ ♘b×d7 8 0-0 c×d4 9 c×d4 ♗b4 10 ♕a4 a5 11 ♘c3 ♗×c3 12 b×c3 b5 13 ♕a3 e5 14 ♗g5 e4 15 ♗×f6 ♘×f6 16 ♘e5 a4 17 ♖ab1 h5 18 ♕b4 ♕×a2 19 c4 b×c4 20 ♕b5+ ♔e7 21 ♕b7+ ♔e6 22 ♕×f7+ ♔f5 23 ♘×c4 ♔f4 24 g3+ ♔g4 25 ♕×g7+ 1-0

**Chiptest – Rex** 1 f4 d5 2 e3 ♘f6 3 ♘f3 e6 4 b3 ♘bd7 5 ♗e2 ♗c5 6 ♗b2 0-0 7 0-0 ♘e4 8 ♘c3 ♘b6 9 ♖e1 ♗d7 10 ♘×e4 d×e4 11 ♘e5 ♘d5 12 ♖f1 b5 13 c4 b×c4 14 ♘×c4 ♖b8 15 ♘e5 ♗b5 16 ♗×b5 ♖×b5 17 ♕c2 ♘f6 18 a4 ♖b6 19 ♕×c5 ♕×d2 20 ♘c4 ♕c2 21 ♗a3 ♖bb8 22 ♕×c7 ♖fc8 23 ♕×a7 ♖×b3 24 ♖ac1 ♕e2 25 ♘a5 ♖bc3 26 ♖×c3 ♖e8 27 ♘c4 ♘d5 28 ♘e5 ♔h8 29 ♕×f7 ♖b8 30 ♕×e6

♕×f1+ 31 ♔×f1 ♘×e3+ 32 ♖×e3 h6 33 ♘f7+ ♔h7 34 ♘g5+ ♔h8 35 ♗b2 h5 36 ♕h6+ ♔g8 37 ♕×g7 # 1-0

**Ostrich — Fidelity** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗×c6 d×c6 5 d4 e×d4 6 ♕×d4 ♕×d4 7 ♘×d4 ♗d7 8 ♘c3 0-0-0 9 ♗e3 ♖e8 10 0-0 ♘f6 11 f3 h5 12 ♗g5 ♘h7 13 ♗h4 ♗d6 14 ♖ad1 g5 15 ♗f2 ♘f6 16 ♗e3 ♖hg8 17 ♖f2 g4 18 ♔h1 ♗×h2 19 ♖fd2 ♗d6 20 ♗f2 g×f3 21 g×f3 h4 22 ♖e2 ♘h5 23 e5 ♗×e5 25 ♗×h4 ♖h8 25 ♖×e5 ♖×e5 26 ♔h2 c5 27 f4 ♖e3 28 ♔g1 ♘×f4 29 ♗f6 ♖g8+ 30 ♔f2 ♖gg3 31 ♔f1 c×d4 32 ♗×d4 ♖gf3+ 33 ♔g1 ♖e8 0-1

**Mephisto — Belle** 1 d4 d5 2 c4 e6 3 ♘c3 c6 4 ♕b3 d×c4 5 ♕×c4 b5 6 ♕d3 ♗a6 7 ♘e4 b4 8 ♕d2 ♕d5 9 ♘g3 ♘f6 10 ♘f3 c5 11 e3 ♗×f1 12 ♖×f1 ♘c6 13 b3 c×d4 14 e×d4 ♖d8 15 ♗b2 a5 16 ♕c2 ♖c8 17 ♖c1 ♖c7 18 ♕d3 ♗d6 19 ♖c4 0-0 20 ♖g1 ♖fc8 21 ♕e2 ♗f4 22 ♘f1 ♕f5 23 ♘3d2 ♘d5 24 ♘e4 ♕g6 25 g3 ♘b6 26 ♖c5 ♘d7 27 ♖c4 ♗h6 28 ♕c2 ♘b6 29 ♖c5 ♘d5 30 ♕d3 ♘f4 31 ♕f3 ♘h3 32 ♖g2 ♘g5 33 ♘×g5 ♗×g5 34 ♘e3 ♗e7 35 ♖c4 ♕b1+ 36 ♘d1 ♕×a2 37 d5 e×d5 38 ♖g4 g6 39 ♕d3 ♖d8 40 ♖f4 ♗g5 41 ♖g4 ♗c1 42 f4 ♖e8+ 43 ♔f1 ♖ce7 44 ♗e5 ♘×e5 45 f×e5 ♕a1 46 ♖f2 ♖×e5 47 ♔g2 f5 48 ♖×f5 49 ♕×f5 ♕e5 50 ♕×e5 ♖×e5 51 ♔f3 ♔f7 52 ♖d4 ♔e6 53 g4 ♗g5 54 ♖d3 h5 55 ♘f2 ♔d6 56 g×h5 g×h5 57 ♘h3 ♗h6 58 ♘g1 ♖e4 59 ♘e2 ♔e5 60 ♘g3 ♖f4+ 61 ♔g2 h4 62 ♘e2 ♖e4 63 ♔f2 ♗f8 64 ♔f3 ♗c5 65 ♖d2 ♖e3+ 66 ♔g4 ♖×b3 67 ♘f4 d4 68 ♘d3+ ♔d5 69 ♔×h4 ♗d6 70 ♘b2 ♖e3 71 ♔g4 ♖e4+ 72 ♔h5 ♖e5+ 73 ♔g4 ♖e3 74 h3 ♖g3+ 75 ♔h4 b3 76 ♖f2 ♖e3 77 ♖f1 ♗a3 78 ♖f5+ ♔e6 79 ♖f2 ♗×b2 80 ♖×b2 a4 81 ♖g2 a3 82 ♖g5 a2 0-1

**Phoenix — Waycool** 1 d4 ♘f6 2 ♗g5 ♘e4 3 ♗h4 g5 4 f3 g×h4 5 f×e4 c5 6 e3 ♕b6 7 ♘f3 ♕×b2 8 ♘bd2 ♕c3 9 ♗d3 ♗h6 10 0-0 ♗×e3+ 11 ♔h1 ♗×d2 12 ♘×d2 c×d4 13 ♘b3 0-0 14 ♕g4+ ♔h8 15 ♕×h4 f6 16 e5 ♖f7 17 e×f6 e×f6 18 ♖ae1 ♕×d3 19 ♖×f6 1-0

**Cyrus — Challenger** 1 c4 ♘f6 2 ♘c3 e6 3 e4 d5 4 e5 d4 5 e×f6 d×c3 6 b×c3 ♕×f6 7 d4 c5 8 ♘f3 c×d4 9 ♗g5 ♕f5 10 c×d4 ♘c6 11 c5 ♗d7 12 ♗d3 ♕d5 13 ♕b3 b6 14 ♖c1 f6 15 ♕×d5 e×d5 16 ♗d2 b×c5 17 d×c5 ♖b8 18 0-0 ♗g4 19 ♗f4 ♖b2 20 ♖fe1+ ♗e7 21 ♖e2 ♖b4 22 ♗d6 ♔f7 23 ♗×e7 ♗×f3 24 g×f3 ♘×e7 25 c6 g6 26 c7 ♘c8 27 ♖d2 ♔e6 28 ♖dc2 ♔d7 29 ♖c5 ♔d6 30 ♖c6+ ♔e7 31 ♔g2 ♖b2 32 ♖1c5 ♖b6 33 ♖c2 ♖×c6 34 ♖×c6 ♖e8 35 ♖c1 ♔d6 36 ♗b5 ♖g8 37 ♔g3 d4 38 ♖c6+ ♔e7 39 ♗c4 ♖f8 40 ♗a6 ♔d7 41 ♖c1 ♔d6 42 ♔f4 ♖e8 43 h4 ♔d5 44 h5 ♔d6 45 h×g6 h×g6 46 ♔g4 ♖h8 47 f4 ♔d7 48 ♗b5+ ♔e7 49 f5 g×f5+ 50 ♔f4 ♖h4+ 51 ♔g3 ♖g4+ 52 ♔f3 ♖g7 53 ♗a6 ♔d7 54 ♔e2 ♖g2 55 a3 ♖h2 56 ♔d3 ♖×f2 57 ♔×d4 ♖h2 58 ♗b5+ ♔d6 59 ♖c6+ ♔e7 60 ♗d3 ♖h5 61 ♔e3 ♖h3+ 62 ♔d2 ♖h2+ 63 ♔e1 ♔d7 64 ♖c3 ♔e6 65 ♗a6 ♖h8 66 ♖d3 ♔e7 67 ♔f2 ♖f8 68 ♖c3 ♔d7 69 ♗b5+ ♔d6 70 ♖c6+ ♔d5 71 ♖c1 ♔d6 72 ♗a6 ♔e6 73 ♖d1 ♔e7 74 ♔e3 ♖h8 75 ♗b7 ♘b6 76 ♖c1 ♘c8 77 ♗g2 ♖h4 78 ♔f2 a5 79 ♖c5 a4 80 ♖c3 ♔d6 81 ♗b7 ♔d7 82 ♔g3 ♖h8 83 ♔f4 ♘d6 84 ♗d5 ♖c8 85 ♗c6+ ♔×c7 86 ♗×a4+ ♔b6 87 ♖×c8 ♘×c8 88 ♔×f5 ♘d6+ 89 ♔×f6 ♘c4 90 ♗c2 ♘×a3 ½-½

**Novag — Merlin** 1 e4 e5 2 f4 ♗c5 3 ♘f3 d6 4 c3 ♘f6 5 f×e5 d×e5 6 ♘×e5 ♕e7 7 d4 ♗d6 8 ♘f3 ♘×e4 9 ♗e2 0-0 10 0-0 c5 11 ♗d3 ♘d7 12 ♖e1 f5 13 ♗c4+ ♔h8 14 ♗e3 ♘b6 15 ♘bd2 c×d4 16 c×d4 ♘×c4 17 ♘×c4 ♗b4 18 ♗d2 ♗e6 19 ♖c1 ♗×d2 20 ♘f×d2 b5 21 ♘×e4 f×e4 22 d5 b×c4 23 d×e6 ♕×e6 24 ♕d4 ♖fe8 25 ♖×c4 e3 26 ♖c7 ♕e5 27 ♕×e5 ♖×e5 28 ♖f7 ♔g8 29 ♖d7 a5 30 ♖e2 ♖c8 31 g3 ♖c6 32 ♔g2 ♖b6 33 h3 ♔f8 34 g4 h5 35 ♔g3 h×g4 36 h×g4 g5 37 ♖d3 ♖×b2 38 ♖×b2 e2 39 ♖b1 e1♕+ 40 ♖×e1 ♖×e1 41 ♔f3 ♖e5 42 ♖d7 ♔e8 43 ♖c7 ♔f8 44 a3 ♔e8 45 ♖b7 ♔f8 46 ♖d7 ♔e8 47 ♖c7 ♔d8 48 ♖b7 ♔e8 49 ♖a7 ½-½

*Round 4, November 4, 1986*

**Vaxchess — Phoenix** 1 d4 c5 2 d5 e5 3 e4 d6 4 ♘c3 g6 5 ♘f3 ♗g7 6 ♗g5 ♗f6 7 ♗e3 ♘e7 8 ♗b5+ ♘d7 9 0-0 0-0 10 ♗h6 ♗g7 11 ♗×g7 ♔×g7 12 ♕d3 ♘b6 13 a4 f5 14 ♖fd1 f×e4 15 ♕×e4 a6 16 ♗f1 ♖f4 17 ♕d3 ♗f5 18 ♕d2 ♗g4 19 ♗e2 ♖b4 20 b3 h6 21 a5 ♘d7 22 h3 ♗×f3 23 ♗×f3 ♖d4 24 ♕e2 ♘f6 25 ♖d3 ♕d7 26 ♖e1 ♕c7 27 ♖a1 c4 28 b×c4 ♖×c4 29 ♖a3 ♖c5 30 ♕e3 ♖×a5 31 ♖×a5 ♕×a5 32 ♕d2 ♕a1+ 33 ♔h2 ♘f5 34 g4 ♘d4 35 ♘e2 ♘×f3+ 36 ♖×f3 a5 37 ♖c3 ♖f8 38 f4 ♘e4 39 ♖c7+ ♖f7 40 ♖×f7+ ♔×f7 41 ♕e3 ♕a4 42 f×e5 ♕×c2 43 e6+ ♔e7 44 ♔g1 ♕d1+ 45 ♔h2 ♕×d5 46 ♕×h6 ♕×e6 47 ♕g7+ ♕f7 48 ♕d4 ♕f3 49 ♕g7+ ♔d8 50 ♕g8+ ♔c7 51 ♕g7+ ♔b8 52 ♕h8+ ♔a7 53 ♕d4+ b6 54 ♕c4 ♕f2+ 55 ♔h1 ♕×e2 56 ♕×e2 ♘g3+ 57 ♔g2 ♘×e2 58 ♔f3 a4 59 ♔×e2 a3 60 ♔e3 0-1

**Mehpisto — Cyrus** 1 d5 d5 2 c4 c6 3 c×d5 c×d5 4 ♘f3 ♘f6 5 ♘c3 ♘c6 6 ♗f4 ♗f5 7 e3 e6 8 ♗d3 ♗×d3 9 ♕×d3 ♘h5 10 ♗g5 f6 11 ♗h4 g5 12 ♗g3 g4 13 ♘d2 ♘×g3 14 h×g3 f5 15 a3 h5 16 ♘e2 ♕g5 17 ♘f4 ♖h6 18 ♕b5 0-0-0 19 ♖c1 ♗d6 20 ♘b3 a6 21 ♖×c6+ b×c6 22 ♕×c6+ ♔b8 23 ♘a5 ♕g7 24 ♕b6+ ♔c8 25 ♕×a6+ ♔d7 26 ♕b7+ ♗c7 27 ♕c6+ ♔c8 28 ♘×e6 ♖×e6 29 ♕×e6+ ♖d7 30 ♕a6+ ♔d8 31 ♘c6+ ♔e8 32 ♕c8+ ♗d8 33 ♘e5 1-0

**Recom — Novag** 1 d4 ♘f6 2 g3 e6 3 ♗g2 d5 4 c4 c5 5 c×d5 e×d5 6 ♘f3 ♘c6 7 ♘c3 ♗e7 8 0-0 0-0 9 ♗g5 c×d4 10 ♘×d4 h6 11 ♗e3 ♗b4 12 ♕b3 ♗×c3 13 ♕×c3 ♗g4 14 ♖fd1 ♖c8 15 ♘×c6 ♖×c6 16 ♕d3 ♕e7 17 h3 ♗h5 18 ♖d2 ♘e4 19 ♗×e4 d×e4 20 ♕d5 ♗g6 21 ♗×a7 ♖fc8 22 ♖ad1 b6 23 ♕d7 ♕b4 24 ♖d6 ♖6c7 25 ♕g4 h5 26 ♕g5 ♖×a7 27 ♕e5 ♖a5 28 a3 ♖×e5 29 a×b4 ♖c2 30 ♖×b6 ♖×e2 31 ♖b8+ ♔h7 32 ♖dd8 ♗f5 33 h4 ♖×b2 34 ♔f1 e3 0-1

**Fidelity — Lachex** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗a4 ♘f6 5 d4 e×d4 6 0-0 ♘×e4 7 ♔h1 ♗c5 8 ♘bd2 ♘×d2 9 ♗×d2 0-0 10 ♗f4 ♖e8 11 ♘g5 ♕f6 12 ♕f3 d5 13 ♕g3 h6 14 ♘f3 ♗d6 15 ♗×d6 c×d6 16 ♖ad1 ♖e4 17 ♘d2 ♖f4 18 ♖de1 ♗e6 19 ♕a3 ♖×f2 20 ♕×d6 ♕e7 21 ♕×e7 ♖×f1+ 22 ♖×f1 ♘×e7 23 ♘f3 b5 24 ♗b3 a5 25 a4 b×a4 26 ♗×a4 ♘f5 27 ♖d1 ♖b8 28 ♖b1 ♖b4 29 b3 ♘e3 30 ♖c1 ♔f8 31 ♘e5 ♗f5 32 ♘c6 ♗×c2 33 ♘×b4 a×b4 34 ♖e1 ♗e4 35

♖e2 d3 36 ♖d2 ♘xg2 37 ♔g1 ♘e1 38 ♔f1 ♘f3 39 ♖a2 d2 40 ♔e2 f5 41 ♔d1 d4 42 ♗b5 ♗d5 43 ♗c4 ♗xc4 4 b×c4 g5 45 h3 ♔e8 46 ♖b2 g4 47 h×g4 f×g4 48 ♖×b4 g3 49 ♖b8+ ♔d7 50 ♖b1 g2 51 ♔e2 g1♕ 52 ♖×g1 ♘xg1+ 53 ♔xd2 h5 0-1

**Bebe – Belle** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗a4 ♘f6 5 0-0 ♘xe4 6 ♖e1 ♘c5 7 ♘xe5 ♗e7 8 ♗×c6 d×c6 9 d4 ♘e6 10 ♗e3 0-0 11 ♘c3 f6 12 ♘d3 ♖e8 13 ♕h5 g6 14 ♕a5 ♘xd4 15 ♗xd4 ♕xd4 16 ♕xc7 ♕d7 17 ♕xd7 ♗xd7 18 ♖e2 ♗d8 19 ♘c5 ♖xe2 20 ♘xe2 ♗c8 21 ♖d1 ♗e7 22 ♘d7 ♖a7 23 h3 ♔f7 24 b3 b5 25 ♘b6 ♗e6 26 ♘d4 ♖b7 27 ♘xe6 ♔xe6 28 ♖e1+ ♔f7 29 ♘c8 ♗c5 30 ♖d1 ♔e6 31 c3 a5 32 a3 ♗xa3 33 ♖a1 b4 34 c×b4 ♗xb4 35 ♖d1 ♗c5 36 ♖c1 ♖b5 37 g3 ♔d7 0-1

**Challenger – Ostrich** 1 e4 c5 2 ♘f3 ♘c6 3 ♗b5 g6 4 0-0 ♗g7 5 c3 ♘f6 6 ♖e1 0-0 7 d4 c×d4 8 c×d4 a6 9 ♗d3 d6 10 ♘c3 ♗g4 11 ♗e3 ♗×f3 12 ♕×f3 ♘d7 13 d5 ♘b4 14 ♕e2 ♗xc3 15 b×c3 ♘xd3 16 ♕×d3 ♘e5 17 ♕d4 ♕d7 18 ♗f4 f6 19 ♖ab1 ♖f7 20 ♖b6 ♖c8 21 ♖eb1 ♕g4 22 ♗g3 ♖c4 23 ♕d2 ♕xe4 24 ♖×b7 ♘d7 25 ♖e1 ♕f5 26 f3 ♖c5 27 ♖d1 ♖c4 28 ♔h1 a5 29 ♖a7 ♖a4 30 ♕b2 ♘c5 31 ♕b8+ ♖f8 32 ♕c7 ♕c2 33 ♕xe7 ♕×d1+ 34 ♗e1 ♕xe1+ 35 ♕xe1 ♖xa2 36 h3 a4 37 ♕e7 1-0

**Merlin – Chiptest** 1 e4 e6 2 d4 d5 3 e5 b6 4 c3 ♕d7 5 ♗e3 ♘c6 6 ♘d2 f6 7 ♗d3 f×e5 8 d×e5 ♘xe5 9 ♕h5+ ♘f7 10 ♘gf3 ♘f6 11 ♕h4 e5 12 ♘g5 e4 13 ♘×f7 ♕×f7 14 ♗b5+ ♗d7 15 ♗×d7+ ♕×d7 16 ♗g5 ♗e7 17 ♗×f6 ♗×f6 18 ♕h5+ g6 19 ♕e2 0-0 20♕e3 ♕a4 21 0-0 ♕c2 22 ♖ab1 ♖ae8 23 ♘b3 ♖e5 24 ♖fc1 ♕d3 25 ♕×d3 e×d3 26 ♘d4 c5 27 ♘b5 ♖e2 28 ♖e1 ♖f7 ♖×e2 d×e2 30 ♖e1 ♖e7 31 a4 d4 32 c×d4 c×d4 33 ♘a3 d3 34 ♘b1 ♗g5 35 f4 ♗×f4 36 g3 ♗e3+ 37 ♔g2 ♖f7 38 ♘c3 ♖f2+ 39 ♔h3 d2 40 ♖×e2 ♖×e2 41 a5 b×a5 0-1

**Waycool – Rex** 1 e4 ♘f6 2 e5 ♘d5 3 d4 d6 4 c4 ♘b6 5 e×d6 ♘8d7 6 d×e7 ♕×e7+ 7 ♕e2 c5 8 ♘f3 c×d4 9 ♕×e7+ ♗×e7 10 b3 ♗b4+ 11 ♗d2 ♗c5 12 b4 ♗d6 13 a4 0-0 14 a5 ♖e8+ 15 ♔d1 ♘xc4 16 ♗×c4 ♗e5 17 ♖e1 ♔f8 18 ♘g5 h6 19 ♘×f7 ♗×h2 20 ♖×e8+ ♔×e8 21 ♘a3 ♘e5 22 ♘d6+ ♔e7 23 ♘×c8+ ♖×c8 24 ♗d5 ♖c7 25 ♘b5 ♖d7 26 ♗a2 a6 27 ♘a3 ♘g4 28 ♗g5+ h×g5 29 ♔e1 ♗g1 30 f3 ♘e3 31 ♔d2 ♗f2 32 g4 g6 33 ♖h1 ♔f6 34 ♖h8 d3 35 ♖f8+ ♔e5 36 ♘c4+ ♔d4 37 ♘b6 ♘f1+ 38 ♔d1 ♖d6 39 ♖g8 ♘e3+ 40 ♔d2 ♘c2 41 ♘c8 ♗e1+ 42 ♔d1 ♗×b4 43 ♖e8 ♘e3+ 44 ♖×e3 ♔×e3 45 ♘×d6 ♗×d6 46 ♗d5 ♗b4 47 ♗e4 ½-½

*Round 5, November 5, 1986*

**Chiptest – Recom** 1 d4 c5 2 d5 e5 3 e4 d6 4 ♘c3 g6 5 ♘f3 ♗g7 6 ♗g5 ♗f6 7 ♗e3 ♘e7 8 ♗b5+ ♘d7 9 0-0 0-0 10 ♗h6 ♗g7 11 ♗×g7 ♔×g7 12 ♕d3 ♘b6 13 a4 f5 14 ♖fd1 f×e4 15 ♕×e4 a6 16 ♗f1 ♖f4 17 ♕d3 ♗f5 18 ♕d2 ♗g4 19 ♗e2 ♖b4 20 b3 h6 21 a5 ♘d7 22 h3 ♗×h3 23 ♗×f3 ♖d4 24 ♕e2 ♘f6 25 ♖d3 ♕d7 26 ♖e1 ♕c7 27 ♖a1 c4 28 b×c4 ♖×c4 29 ♖a3 ♖c5 30 ♖e3 ♖×a5 31 ♖×a5 ♕×a5 32 ♕d2 ♕a1+ 33 ♔h2 ♘f5 34 g4 ♘d4 35 ♘e2 ♘×f3+ 36 ♖×f3 a5 37 ♖c3 ♖f8 38 f4 ♘e4 39 ♖c7+ ♖f7 40 ♖×f7+ ♔×f7 41 ♕e3 ♕a4 42 f×e5 ♕×c2 43 e6+ ♔e7 44 ♔g1 ♕d1+ 45 ♔h2 ♕×d5 46 ♕×h6 ♕×e6 47 ♕g7+ ♕f7 48 ♕d4 ♕f3 49 ♕g7+ ♔d8 50 ♕g8+ ♔c7 51 ♕g7+ ♔b8 52 ♕h8+ ♔a7 53 ♕d4+ b6 54 ♕c4 ♕f2+ 55 ♔h1 ♕×e2 56 ♕×e2 ♘g3+57 ♔g2 ♘xe2 58 ♔f3 a4 59 ♔xe2 a3 60 ♔e3 ½-½

**Phoenix – Fidelity** 1 d4 ♘f6 2 ♗g5 d5 3 ♗×f6 e×f6 4 e3 ♗d6 5 c4 d×c4 6 ♗×c4 c5 7 d×c5 ♕a5+ 8 ♘d2 ♗×c5 9 ♘gf3 ♘c6 10 0-0 ♘e5 11 ♘×e5 f×e5 12 ♘b3 ♕c7 13 ♗b5+ ♔e7 14 ♖c1 b6 15 ♘×c5 b×c5 16 b4 ♕d6 17 ♕×d6+ 18 ♖fd1+ ♔e7 19 ♖×c5 ♗e6 20 ♖×e5 ♔f6 21 f4 ♗×a2 22 ♖d6+ ♗e6 23 ♔f2 ♖hc8 24 ♖c5 ♖ab8 25 ♗d3 g6 26 e4 ♔e7 27 e5 f6 28 ♖a6 ♖×b4 29 ♖×e6+ ♔×e6 30 ♖×c8 ♖×f4+ 31 ♔e3 f×e5 32 ♖c6+ ♔d5 33 ♖a6 ♖f7 34 ♗e4+ ♔c5 35 ♖c6+ ♔b4 36 ♖e6 a5 37 ♖×e5 ♖c7 38 ♗f3 ♖d7 39 ♖e4+ ♔b3 40 ♖d4 ♖×d4 41 ♔×d4 a4 42 ♗d1+ ♔b4 43 ♔d3 h6 44 ♔c2 a3 45 ♔b1 ♔c3 46 ♔a2 h5 47 ♗b3 g5 48 ♔×a3 h4 49 h3 ♔d3 50 ♔b4 ♔d4 51 ♔b5 ♔e4 52 ♗c2+ ♔d5 53 ♗d1 ♔d6 54 ♗f3 ♔e6 55 ♔c6 ♔e5 56 ♗g4 ♔f4 57 ♔d6 ♔g3 58 ♗f3 1-0

**Cyrus – Ostrich** 1 e4 c5 2 c3 b6 3 d4 ♗b7 4 ♗d3 c×d4 5 c×d4 ♘f6 6 ♘c3 e6 7 ♗e3 ♗b4 8 f3 ♗×c3+ 9 b×c3 ♕e7 10 ♘e2 d5 11 e5 ♘fd7 12 ♕a4 ♕h4+ 13 g3 ♕h5 14 0-0 g5 15 f4 h6 16 ♖ad1 a6 17 ♕a3 ♘c6 18 f5 e×f5 19 ♖×f5 ♘e7 20 ♖f2 ♕h3 21 ♖df1 ♖f8 22 ♖f3 ♗c6 23 ♔h1 a5 24 ♕d6 b5 25 ♕c7 a4 26 a3 ♕g4 27 ♕d6 ♕h3 28 ♖3f2 ♕h5 29 ♖f6 ♕h3 30 ♘g1 ♕g4 31 ♖×h6 ♘b8 32 ♘h3 ♘c8 33 ♕c5 ♘e7 34 ♗×g5 ♕d7 35 ♗×e7 ♘a6 36 ♕×c6 ♕×c6 37 ♖×c6 ♔×e7 38 ♗×b5 ♘b8 39 ♖c5 ♖a7 40 ♖×d5 ♖b7 41 c4 1-0

**Belle – Challenger** 1 e4 c5 2 ♘f3 ♘c6 3 d4 c×d4 4 ♘×d4 g6 5 c4 ♗g7 6 ♗e3 ♘f6 7 ♘c3 ♘g4 8 ♕×g4 ♘×d4 9 ♕d1 e5 10 ♘b5 0-0 11 ♕d2 ♕h4 12 ♗d3 d5 13 c×d5 ♘×b5 14 ♗×b5 ♕×e4 15 0-0 ♖d8 16 d6 ♗e6 17 ♖ad1 ♗×a2 18 ♗g5 f6 19 ♗e3 ♗b3 20 ♕d3 ♕×d3 21 ♖×d3 ♗e6 22 ♖a1 a6 23 d7 ♖f8 24 ♗a4 e4 25 ♖d2 b5 26 ♗×b5 a×b5 27 ♖×a8 ♖×a8 28 d8♕+ ♖×d8 20 ♖×d8+ ♔f7 30 ♖b8 ♗d7 31 ♗d4 h5 32 h4 ♗e8 ♖b7+ ♔f8 34 ♔h2 ♗h8 35 ♔g3 ♗g7 36 ♗c3 e3 37 f×e3 ♔g8 38 e4 ♗c6 39 ♖b6 ♗×e4 40 ♗×f6 ♗×f6 41 ♖×f6 ♔g7 42 ♖b6 ♗d3 43 b3 ♗e2 44 ♖d6 ♔f7 45 ♔f4 ♗g4 46 ♖d5 b4 47 ♖b5 ♗e6 48 ♖×b4 ♔f6 49 g3 ♗f7 50 ♖b6+ ♔e7 51 ♔e5 1-0

**Merlin – Waycool** 1 e4 c5 2 c3 ♘f6 3 e5 ♘d5 4 d4 c×d4 5 ♘f3 ♘c6 6 c×d4 d6 7 ♗c4 ♘b6 8 ♗b5 d5 9 ♘c3 ♗g4 10 h3 ♗×f3 11 ♕×f3 e6 12 0-0 ♗e7 13 ♕g3 ♔f8 14 ♘e2 ♖c8 15 ♗f4 ♘b4 16 ♗d2 ♘c2 17 ♖ac1 a6 18 ♗d3 ♘b4 19 ♖×c8 ♕×c8 20 ♖c1 ♘c4 21 b3 ♘×a2 22 ♖c2 b5 23 ♖×a2 ♘×d2 24 ♖×d2 ♔g8 25 ♖c2 ♕a8 26 ♘f4 ♕d8 27 ♘h5 g6 28 ♖c6 ♕a8 29 ♖c7 ♕d8 30 ♖a7 ♔f8 31 ♘f6 ♗×f6 32 e×f6 ♕×f6 33 ♕d6+ ♔g7 34 ♕c5 ♕f4 35 ♖×a6 e5 36 d×e5 ♕×e5 37 ♗×b5 ♖b8 38 b4 ♕e1+ 39 ♔h2 ♕e4 40 ♖a7 ♕e5+ 41 g3 ♕f6 42 ♔g2 ♕b6 43 ♕×b6 ♖×b6 44 ♖a5 d4 45 ♔f3 f5 46 ♔e2 h5 47 ♔d3 ♖d6 48 ♖a6 ♖d8 49 ♖c4 g5 50 ♖a7+ ♔f6 51 ♖f7+ ♔g6 52 b5 h4 53 b6 h×g3 54 f×g3 f4 1-0

**Lachex – Bebe** 1 e4 c5 2 c3 d5 3 e×d5 ♕×d5 4 d4

e6 5 ♘f3 ♘c6 6 ♘a3 ♗d7 7 ♗c4 ♕e4+ 8 ♕e2 ♕×e2+ 9 ♗×e2 c×d4 10 ♘b5 0-0-0 11 c×d4 ♗b4+ 12 ♘c3 ♗e8 13 ♗e3 ♘ge7 14 0-0 ♘g6 15 ♖ad1 ♔b8 16 a3 ♗d6 17 d5 e×d5 18 ♖×d5 ♘f4 19 ♖×d6 ♘×e2+ 20 ♘×e2 ♖×d6 21 ♗f4 ♔c8 22 ♗×d6 ♗d7 23 ♖c1 ♗g4 24 ♘ed4 ♗×f3 25 g×f3 ♔d7 26 ♘×c6 b×c6 27 ♖d1 ♔c8 28 ♗f4 ♖d8 29 ♖×d8+ ♔×d8 30 b4 a6 31 ♔f1 ♔e7 32 ♔e2 ♔e6 33 ♔d3 ♔d5 34 ♗e3 g6 35 a4 h5 36 h4 f6 37 f4 f5 38 ♗b6 ♔e6 39 ♔c4 ♔d6 40 f3 1-0

**Novag – Mephisto** 1 e4 c6 2 d4 d5 3 e×d5 c×d5 4 c4 ♘f6 5 ♘c3 e6 6 ♘f3 ♗b4 7 ♗d2 0-0 8 ♗d3 d×c4 9 ♗×c4 ♘c6 10 a3 ♗e7 11 ♗f4 ♗d6 12 ♗×d6 ♕×d6 13 0-0 ♖d8 14 ♘b5 ♕e7 15 ♖c1 a6 16 ♘c3 ♕d6 17 ♘e2 b6 18 ♕c2 ♗b7 19 ♖fe1 ♘a5 20 ♘e5 ♖ac8 21 ♕a4 ♘×c4 22 ♘×c4 ♕d5 23 ♘e3 ♕g5 24 ♖×c8 ♖×c8 25 ♖c1 ♖×c1+ 26 ♘×c1 ♕f4 27 ♘e2 ♕d6 28 ♕c4 ♔f8 29 h3 h6 30 ♕d3 h5 31 ♘c3 b5 32 b4 ♔e7 33 ♕e2 g6 34 ♕d2 ♕c7 35 ♕c2 ♕c6 36 ♕d3 ♔f8 37 ♔f1 ♕c7 38 ♔g1 ♕d6 39 ♕d2 ♔g7 40 ♕d3 1-0

**Rex – Vaxchess** 1 e4 c5 2 c3 d5 3 e×d5 ♕×d5 4 d4 e6 5 ♘f3 ♘f6 6 ♘a3 ♘c6 7 ♘b5 ♗d6 8 d×c5 ♕×d1+ 9 ♕×d1 ♗b8 10 ♗e3 0-0 11 ♔c1 a6 12 ♘bd4 ♗d7 13 ♗c4 ♘×d4 14 c×d4 ♗c6 15 h3 ♗×f3 16 g×f3 ♖d8 17 ♖e1 ♔f8 18 h4 ♗h2 19 h5 b5 20 c×b6 ♖ac8 21 b3 h6 22 ♔b2 ♗d6 23 ♗×a6 ♖c6 24 ♗c4 ♖×b6 25 ♖h1 ♖c6 26 ♖ac1 ♖a8 27 ♔b1 ♖bb8 28 ♗d2 ♖a4 29 ♖c2 ♖a3 30 f4 ♘e4 31 ♗c1 ♖a5 32 ♗e3 ♘f6 33 ♗e2 ♘d5 34 ♖h4 ♗b4 35 ♗c4 ♗e7 36 ♖h1 ♘f6 37 ♗e2 ♗d6 38 ♖c6 ♗e7 39 ♖hc1 ♖d8 40 ♖c8 ♘×h5 41 ♖cc7 ♘f6 42 ♗c4 ♘e8 43 ♖×d8 ♗×d8 44 ♖c8 ♔e7 45 ♗d2 ♖h5 46 a4 ♗b6 47 ♗e3 ♘d6 48 ♖c6 ♗a5 49 ♖c5 ♖×c5 50 d×c5 ♘×c4 51 b×c4 h5 52 ♔b2 h4 53 f3 h3 54 ♗g1 ♗d2 55 ♗h2 ♔d7 56 f5 e×f5 57 ♔b3 ♗e3 58 ♔b4 ♔c6 59 ♔c3 ♗×c5 60 a5 g5 61 ♔b3 g4 62 f×g4 ♗d6 63 ♗g1 f×g4 64 a6 h2 65 ♗×h2 ♗×h2 0-1

### ACM 1987 – Dallas

| | rate | perf | 1 | 2 | 3 | 4 | tot |
|---|---|---|---|---|---|---|---|
| 1 Chiptest-m | 2350 | 2584 | 5+ | 4+ | 2+ | 3+ | 4 |
| 2 Cray Blitz | 2200 | 2275 | 7+ | 6+ | 1– | 5+ | 3 |
| 3 Sun Phoenix | 0 | 2253 | 10+ | 8+ | 6+ | 1– | 3 |
| 4 Lachex | 0 | 1981 | 12+ | 1– | 1= | 7+ | 2½ |
| 5 Cyrus 68K | 0 | 1994 | 1– | 9+ | 8+ | 2– | 2 |
| 6 Bebe | 2100 | 2000 | 11+ | 2– | 3– | 12+ | 2 |
| 7 Novag X | 0 | 1827 | 2– | 13+ | 12+ | 4– | 2 |
| 8 Belle | 0 | 1790 | 9+ | 3– | 5– | Bye | 2 |
| 9 Waycool | 0 | 1658 | 8– | 5– | 13+ | 11+ | 2 |
| 10 GNU Chess | 0 | 1298 | 3– | 11– | Bye | 13+ | 2 |
| 11 BF | 0 | 1620 | 6– | 10+ | 4= | 9– | 1½ |
| 12 Ostrich | 1750 | 1537 | 4– | Bye | 7– | 6– | 1 |
| 13 Greco | 1000 | 911 | Bye | 7– | 9– | 10– | 1 |

*Round 1, October 25, 1987*

**Cray Blitz – Novag X** 1 e4 e5 2 ♘f3 ♘c6 3 ♗c4 ♗c5 4 d3 ♘f6 5 ♘c3 d6 6 ♗g5 ♘a5 7 ♗×f6 g×f6 8 ♗b5+ c6 9 ♗a4 b5 10 ♗b3 ♘×b3 11 c×b3 ♖g8 12 0-0 ♕b6 13 ♘h4 ♖g4 14 ♘f3 ♖g6 15 ♘h4 ♖h6 16 ♘f5 ♗×f5 17 e×f5 0-0-0 18 ♕g4 d5 19 ♕g7 ♗f8 20 ♕×f7 ♔b8 21 ♖ac1 ♕a5 22 a3 ♕b6 23 h3 ♕b7 24 ♕g8 ♖e8 25 ♕g4 ♗d6 26 ♘e2 ♖c8 27 b4 ♕f7 28 ♘g3 ♖g8 29 ♕f3 ♕b7 30 ♖c3 ♖h4 31 ♖fc1 ♖c8 32 ♘h5 ♗e7 33 g4 d4 34 ♖×c6 ♖×c6 35 ♖×c6 ♖×h3 36 ♕d5 a6 37 ♘×f6 ♗×f6 38 f3 ♗g7 39 ♕d6+ ♔a7 40 ♖c7 e4 41 d×e4 ♖×f3 42 ♖×b7+ ♔×b7 43 ♕d7+ ♔b8 44 ♕×g7 d3 45 ♕g8+ 1-0

**Waycool – Belle** 1 e4 e5 2 ♘f3 ♘c6 3 c3 ♘f6 4 d4 ♘×e4 5 d5 ♘b8 6 ♘×e5 ♗c5 7 ♕g4 0-0 8 ♕×e4 d6 9 ♗d3 f5 10 ♕c4 b5 11 ♕×b5 ♕e7 12 0-0 d×e5 13 ♗g5 ♕d6 14 ♕b3 ♔h8 15 ♘d2 ♘d7 16 ♖ad1 ♕g6 17 ♗e3 ♖b8 18 ♕×b8 ♘×b8 19 ♗×c5 ♖f6 20 ♗×a7 ♘d7 21 ♗b5 ♖a6 22 ♗×a6 ♕×a6 23 ♗e3 f4 24 ♗×f4 e×f4 25 ♖fe1 ♕g6 26 ♖e7 ♘f6 27 ♘f3 ♗h3 28 ♘h4 ♕g4 29 f3 ♕×h4 30 g×h3 ♕×h3 31 ♖f1 ♕c8 32 c4 c6 33 b4 c×d5 34 c5 ♕a8 35 ♖f2 d4 36 ♖ee2 d3 37 ♖b2 ♘d7 38 ♔f1 ♘e5 39 ♖b3 ♕d5 40 ♖b1 ♘×f3 41 a4 ♘g5 0-1

**Chipest-m – Cyrus 68K** 1 e4 c5 2 c3 ♘f6 3 e5 ♘d5 4 d4 c×d4 5 c×d4 d6 6 ♘f3 ♘c6 7 ♗e2 g6 8 e×d6 ♕×d6 9 ♘c3 ♘×c3 10 b×c3 ♗g7 11 0-0 0-0 12 ♕b3 ♘a5 13 ♕b5 b6 14 ♗e3 ♕c7 15 ♖ac1 ♗d7 16 ♕b4 ♗e6 17 c4 ♘c6 18 ♕d2 ♗g4 19 c5 b×c5 20 ♖×c5 ♕b7 21 ♖fc1 ♖ac8 22 d5 ♗×f3 23 g×f3 ♘e5 24 ♖×c8 ♖×c8 25 ♖×c8+ ♕×c8 26 ♗×a7 ♕h3 27 ♕d1 ♕h4 28 ♗e3 ♘d7 29 ♕c1 ♗e5 30 ♕c8+ ♘f8 31 h3 ♗g7 32 ♗c4 ♕f6 33 ♔g2 ♕e5 34 a4 ♕b2 35 a5 ♗d4 36 ♗×d4 ♕×d4 37 a6 ♕b6 38 d6 e×d6 39 ♕e8 d5 40 ♗×d5 ♕c7 41 ♕×f7+ 1-0

**BP – Bebe** 1 e4 c5 2 ♘f3 d6 3 d4 c×d4 4 ♘×d4 ♘f6 5 f3 e5 6 ♗b5+ ♗d7 7 ♗×d7+ ♘b×d7 8 ♘f5 d5 9 e×d5 ♕a5+ 10 ♘c3 ♘b6 11 ♕e2 0-0-0 12 ♗d2 ♘b×d5 13 0-0-0 ♕c7 14 ♘b5 ♕c5 15 b4 ♕b6 16 ♕c4+ ♔b8 17 ♗e3 ♘×e3 18 ♖×d8+ ♕×d8 19 ♘×e3 a6 20 ♘c3 ♕d4 21 ♘cd1 ♗×b4 22 ♕×d4 e×d4 23 ♘f5 ♗c5 24 g3 g6 25 ♘h6 ♖f8 26 ♘f2 d3 27 ♘d1 ♖c8 28 c3 ♖c7 29 ♔d2 ♗f8 30 ♘g4 ♘×g4 31 f×g4 ♖d7 32 g5 ♗e7 33 h4 f6 34 g×f6 ♗×f6 35 ♘f2 ♖d5 36 ♖e1 ♖a5 37 ♖e8+ ♔c7 38 ♔×d3 ♖×a2 39 ♘e4 ♗d8 40 ♖g8 ♖a5 41 ♖g7+ ♔c6 42 ♖×h7 ♖d5+ 43 ♔e2 ♖d7 44 ♖×d7 ♔×d7 45 ♘c5+ ♔c8 46 c4 a5 47 ♔e3 ♗e7 48 ♔d4 b6 49 ♘a4 ♗f6+ 50 ♔d5 ♔c7 0-1 (time)

**Sun Phoenix – GNU Chess** 1 d4 d5 2 ♗g5 ♘c6 3 e3 f6 4 ♗h4 e5 5 d×e5 ♘×e5 6 ♘f3 ♗g4 7 ♗e2 ♗b4+ 8 c3 ♗×f3 9 ♗×f3 ♘×f3+ 10 ♕×f3 ♗c5 11 c4 ♗b4+ 12 ♘c3 ♗×c3+ 13 b×c3 ♘e7 14 ♖d1 c6 15 c×d5 ♘×d5 16 e4 ♕e7 17 0-0 ♘b6 18 e5 0-0 19 e×f6 ♕f7 20 ♕g3 g6 21 f4 ♕×a2 22 f5 ♕f7 23 ♗g5 ♖fc8 24 ♖d4 ♖e8 25 ♕h3 ♖e2 26 ♖h4 ♖×g2+ 27 ♔h1 ♖×h2+ 28 ♔×h2 1-0

**Ostrich – Lachex** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗×c6 d×c6 5 d4 e×d4 6 ♕×d4 ♕×d4 7 ♘×d4 ♗d7 8 ♘c3 ♗b4 9 0-0 ♗×c3 10 b×c3 c5 11 ♘b3 c4 12 ♘d4 ♘f6 13 ♗a3 0-0-0 14 ♗e7 ♘×e4 15 ♗×d8 ♖×d8 16 ♘e2 ♗a4 17 f3 ♘f6 18 ♖fc1 ♖d2 19 ♘d4 ♘d5 20 ♖d1 ♖×d1+ 21 ♖×d1 ♘×c3 22 ♖a1 b5 23

♔f2 b4 24 ♔e3 g6 25 a3 b3 26 c×b3 c×b3 27 ♘×b3 ♗×b3 28 ♔d3 ♘b5 29 a4 ♘d6 30 ♔c3 ♗e6 31 ♔d4 ♘b7 32 ♔c3 a5 33 ♔d4 c5+ 34 ♔c3 f5 35 ♖b1 ♗d5 36 ♖d1 ♗c6 37 ♔b3 ♔c7 38 ♖e1 ♔d6 39 ♖e3 ♘d8 40 ♔c4 ♘e6 41 ♖d3+ ♘d4 42 ♖a3 g5 43 h3 ♗d5+ 44 ♔c3 ♘e2+ 45 ♔d2 ♘f4 46 g4 ♘×h3 47 g×f5 ♘g1 48 ♔e3 ♘×f3 49 f6 ♘e5 50 ♔e2 h5 51 ♖h3 h4 52 ♖c3 g4 53 ♔e3 h3 54 ♖c2 0-1

*Round 2, October 25, 1987*

**Bebe — Cray Blitz** 1 e4 c5 2 ♘f3 e6 3 d4 c×d4 4 ♘×d4 ♘c6 5 ♘c3 a6 6 ♗c4 ♗b4 7 ♘×c6 b×c6 8 0-0 ♘e7 9 ♕d4 0-0 10 ♗e3 d5 11 ♖fd1 ♕a5 12 e×d5 ♘f5 13 ♕f4 ♗×c3 14 d×e6 ♗×b2 15 ♗d2 ♕c5 16 e×f7+ ♔h8 17 ♖ab1 ♗e5 18 ♕e4 ♘d6 19 ♕e2 ♕×c4 20 ♕×e5 ♘×f7 21 ♕b2 ♗f5 22 ♗c3 ♕g4 23 f3 ♕g6 24 ♖d2 ♔g8 25 ♕b3 ♗e6 26 ♕a3 ♖fe8 27 f4 ♗d5 28 ♖f2 ♘d6 29 ♗b4 ♘e4 30 ♖f3 ♘f6 31 f5 ♕f7 32 ♖e3 ♖×e3 33 ♕×e3 ♗×a2 34 ♖a1 ♘d5 35 ♕d4 ♘×b4 36 ♕×b4 ♕a7+ 37♔f1 ♗e6 38 g4 ♕e3 39 ♖a3 ♕c1+ 40 ♔f2 ♕×c2+ 41 ♔g3 ♗d5 42 ♕f4 ♕g2+ 43 ♔h4 c5 44 ♖c3 ♖c8 0-1

**Belle — Sun Phoenix** 1 e4 e6 2 d4 d5 3 ♘d2 ♘f6 4 e5 ♘g8 5 ♗d3 c5 6 c3 ♘c6 7♘e2 ♗d7 8 0-0 ♕b6 9 ♘f3 f6 10 d×c5 ♗×c5 11 ♕c2 f×e5 12 ♗×h7 0-0-0 13 ♗g5 ♘f6 14 b4 ♗d6 15 ♗f6 g×f6 16 ♕g6 e4 17 ♘fd4 ♕c7 18 h3 ♘×d4 19 ♘×d4 e5 20 ♘f5 ♗×f5 21 ♕×f5+ ♔b8 22 ♗g6 ♖dg8 23 ♖ad1 ♖h6 24 ♖×d5 ♖h×g6 25 g4 ♕×c3 26 ♔g2 ♕×b4 27 ♖fd1 ♗c7 28 ♖c1 ♗b6 29 ♖c2 ♖h6 30 ♕e6 ♖gh8 31 g5 ♖×h3 32 ♕×h3 ♖×h3 33 ♔×h3 e3 34 f×e3 ♕e4 35 ♖dd2 ♕×e3+ 36 ♔h2 ♕f4+ 37 ♔h3 f×g5 38 ♔g2 g4 39 ♖f2 ♕h6 40 ♖c3 ♗×f2 0-1

**Lachex — Chiptest-m** 1 e4 d5 2 e×d5 3 ♘c3 ♕a5 4 d4 c6 5 ♗c4 ♘f6 6 ♗d2 ♕b6 7 ♘f3 ♕×b2 8 ♘e5 e6 9 ♖b1 ♕a3 10 ♖b3 ♕d6 11 ♗f4 ♕d8 12 0-0 ♗e7 13 ♖e1 0-0 14 ♗d3 c5 15 d×c5 ♗×c5 16 ♕f3 ♘bd7 17 ♘c4 ♘b6 18 ♘×b6 a×b6 19 ♗e5 ♗d4 20 ♗×d4 ♕×d4 21 ♕e3 ♕×e3 22 ♖×e3 ♘d7 23 ♖b4 ♖a5 24 ♗b5 ♘f6 25 ♗e2 ♘d5 26 ♘×d5 e×d5 27 a3 ♖c5 28 ♗d3 ♖c6 29 ♖e7 ♖d8 30 ♖f4 f6 31 ♖b4 ♔f8 32 ♖e1 h6 33 ♖d4 ♗e6 34 ♖a4 ♔f7 35 ♔h1 f5 36 f4 d4 37 ♖e5 ♔f6 38 ♖e1 g5 39 f×g5+ h×g5 40 ♖b1 ♖dd6 41 ♖f1 ♖c5 42 ♖a8 g4 43 ♖h8 ♔g6 44 ♖b8♖c7 45 ♖h8 ♖d5 46 ♖e1 ♗f7 47 h3 g3 48 a4 ♔f6 49 ♖b1 ♖e7 50 ♖h6+ ♔g5 51 ♖h×b6 ♗e8 52 ♖6b4 ♗c6 53 ♖f1 ♖ee5 54 ♖d1 f4 55 ♖db1 ♖d6 56 ♗b5 ♖e3 57 ♗×c6 b×c6 58 ♖d1 ♖e2 59 ♖c1 ♖de6 60 ♖bb1 ♖6e3 61 a5 ♖a3 0-1

**Novag X — Greco** 1 e4 e5 2 ♘f3 ♘f6 3 ♘×e5 d6 4 ♘f3 ♘×e4 5 d4 d5 6 ♗d3 ♗d6 7 c4 ♗b4+ 8 ♘bd2 ♗×d2+ 9 ♗×d2 ♕e7 10 0-0 ♘×d2 11 ♕×d2 d×c4 12 ♖fe1 ♗e6 13 ♗×c4 ♘c6 14 d5 0-0-0 15 ♕c3 b5 16 ♗×b5 ♘b4 17 d×e6 f×e6 18 ♘e5 ♖hf8 19 a3 a6 20 a×b4 ♖d6 21 ♖×a6 c6 22 ♗×c6 1-0

**Cyrus 68K - Waycool** 1 d4 ♘f6 2♘c3 c5 3 d×c5 ♕a5 4 ♘f3 e6 5 ♗d2 ♗×c5 6 e3 ♕d8 7 ♗d3 d5 8 0-0 ♘c6 9 e4 d×e4 10 ♘×e4 ♗e7 11 ♕e2 ♗d7 12 ♘×f6+ ♗×f6 13 c3 ♗e7 14 ♗f4 0-0 15 ♖fd1 f5 16 ♗c4 ♖f6 17 ♖d2 ♕e8 18 ♘g5 ♘d8 19 ♖ad1 ♖g6 20 ♖×d7 ♗×g5 21 ♗×g5 ♖×g5 22 ♕e5 ♖c8 23 ♗b3 ♔h8 24 ♕e3 ♖h5 25 ♕×a7 e5 26 ♕b6 ♘e6 27 ♕×e6 ♕×e6 28 ♗×e6 g6 29 ♖×b7 1-0

**GNU Chess — BP** 1 d4 d5 2 ♘f3 ♘f6 3 ♗f4 c5 4 e3 e6 5 ♘c3 c4 6 a3 ♗d6 7 ♗×d6 ♕×d6 8 b3 c×b3 9 c×b3 ♗d7 10 ♗d3 0-0 11 ♘e5 ♖c8 12 ♕d2 ♘c6 13 ♘×d7 ♕×d7 14 0-0 e5 15 d×e5 ♘×e5 16 ♖ac1 a5 17 ♖fd1 ♕e6 18 ♘b5 ♘×d3 19 ♕×d3 ♖×c1 20 ♖×c1 ♖c8 21 ♖d1 ♖c5 22 ♘d4 ♕e4 23 ♕×e4 d×e4 24 ♘f5 h5 25 ♘d6 b6 26 b4 ♖c3 27 ♖a1 g5 28 h3 a4 29 ♘b5 ♖d3 30 ♖c1 ♘d5 31 ♖c8+ ♔g7 32 ♖a8 ♖d1+ 33 ♔h2 ♖d2 34 ♘d6 ♖×f2 35 ♘×e4 ♖e2 36 ♘g3 ♖a2 37 ♖×a4 ♘×e3 38 ♔h1 ♖a1+ 39 ♔h2 h4 40 ♘h5+ ♔g6 41 g4 f5 42 b5 f×g4 43 h×g4 ♖b1 44 ♖e4 ♘×g4+ 45 ♖×g4 ♔×h5 46 ♖b4 ♖×b4 47 a×b4 ♔g4 48 ♔g2 h3+ 49 ♔h2 ♔h4 50 ♔h1 g4 51 ♔g1 ♔g3 52 ♔h1 ♔f2 53 ♔h2 g3+ 54 ♔×h3 g2 55 ♔g4 g1♕+ 0-1

*Round 3, October 26, 1987*

**Cray Blitz - Chiptest-m** 1 e4 d5 2 e×d5 ♕×d5 3 ♘c3 ♕a5 4 d4 c6 5 ♘f3 ♘f6 6 ♗c4 ♗g4 7 h3 ♗h5 8 ♕e2 ♘bd7 9 ♗d2 ♕c7 10 g4 ♗g6 11 0-0-0 0-0-0 12 ♘g5 e5 13 ♗×f7 e×d4 14 ♘a4 ♘e5 15 ♗×g6 ♘×g6 16 ♘e6 ♖e8 17 ♖he1 ♕d6 18 g5 ♘d7 19 ♕g4 b5 20 ♘ac5 ♘ge5 21 ♘×f8 ♖h×f8 22 ♘e4 ♕d5 23 ♕g2 ♖e6 24 ♔b1 ♘f3 25 ♕g4 ♘×e1 26 ♖×e1 ♘e5 27 ♕d1 ♘f3 0-1

**Sun Phoenix — Bebe** 1 d4 ♘f6 2 ♗g5 ♘e4 3 ♗h4 d5 4 f3 ♘d6 5 e3 g6 6 ♘c3 ♗h6 7 e4 d×e4 8 f×e4 ♗g7 9♕d2 0-0 10 0-0-0 ♘c6 11 ♘f3 ♗g4 12 ♕e3 a5 13 h3 ♗d7 14 ♗e2 a4 15 a3 b6 16 ♗g3 ♗e8 17 d5 ♘a5 18 e5 ♘f5 19 ♕f2 ♘×g3 20 ♕×g3 ♘b7 21 ♘d4 ♘c5 22 ♕e3 ♕c8 23 ♗g4 ♕b8 24 ♖de1 ♕b7 25 ♗f3 ♖c8 26 ♔b1 ♕a6 27 ♘a2 ♖d8 28 ♘b4 ♕c8 29 ♘bc6 ♗×c6 30 ♘×c6 ♖de8 31 h4 h632 h5 g5 33 ♖hf1 ♕f5 34 ♕d4 ♕c8 35 ♗g4 ♕a6 36 ♗e2 ♕b7 37 ♗b5 ♕a8 38 ♖f3 e6 39 ♗c4 e×d5 40 ♗×d5 ♘e6 41 ♕c4 ♕c8 42 ♘a7 ♕d7 43 g3 c5 44 ♘b5 ♖e7 45 ♘d6 ♔h8 46 ♖×f7 ♖f×f7 47 ♘×f7+ ♖×f7 48 ♗×e6 ♕d2 49 ♕e4 ♖a7 50 ♗c4 ♕d4 51 ♕×d4 c×d4 52 ♖e4 ♖e7 53 e6 ♗f6 54♗b5 ♔g7 55 ♗d7 ♔f8 56 ♔c1 ♔g7 57 ♔d2 ♔f8 58 ♔e2 ♔g8 59 ♔f3 ♔f8 60 ♔g4 ♔g8 61 ♔f5 ♗h8 62 ♔g6 ♗g7 63 ♗×a4 ♔h8 64 b4 ♔g8 65 ♗d7 ♔h8 66 ♖e1 ♗f8 67 ♔f6 1-0

**Cyrus 68K - Belle** 1 d4 d5 2♘c3 f5 3 ♗g5 ♘f6 4 e3 e6 5 ♗×f6 ♕×f6 6 ♘f3 ♗d7 7 ♘b5 ♗b4+ 8 c3 ♗d6 9 ♘×d6+ c×d6 10 ♕b3 ♗c6 11 ♗b5 ♕e7 12 ♗d3 0-0 13 ♖d1 ♗e8 14 ♖c1 a5 15 0-0 ♗c6 16 ♖fd1 a4 17 ♕b4 ♗e8 18 b3 a×b3 19 a×b3 ♘c6 20 ♕b6 ♖a6 21 ♗×a6 b×a6 22 ♕×a6 ♕d7 23 ♖d2 ♕f7 24 ♕a3 f4 25 ♕×d6 ♕e7 26 ♕×e7 ♘×e7 27 e×f4 1-0

**BP — Lachex** 1 e4 e5 2 f4 e×f4 3 ♘f3 g5 4 h4 g4 5 ♘e5 h5 6 ♗c4 ♖h7 7 d4 ♗h6 8 ♘c3 ♘c6 9 ♘×f7 ♖×f7 10 ♗×f7+ ♔×f7 11 ♗×f4 ♗×f4 12 0-0 ♕f6 13 ♘d5 ♕×d4+ 14 ♕×d4 ♘×d4 15 ♘×f4 ♔g7 16 ♘×h5+ ♔h6 17 ♘g3 ♔h7 18 c3 ♘c2 19 ♖f7+ ♔g6 20 ♖af1 ♘e3 21 h5+ ♔h6 22 ♖e1 ♘c4 23 ♘f5+ ♔×h5 24 ♖h7+ ♘h6 25 ♖×h6+ ♔g5 26 ♖h8 b6 27 ♖g8+ ♔f6 28 ♘h6 ♔e6 29 b3 ♘e5 30 ♘×g4 ♗b7

31 ♖×a8 ♗×a8 32 ♘f2 a5 33 a4 ♗c6 34 ♔h2 d6 35 ♔g3 ♘d7 36 ♘d3 b5 37 a×b5 ♗×b5 38 c4 ♗c6 39 e5 d×e5 40 ♘×e5 ♘×e5 41 ♔f4 ♔d6 42 ♖×e5 ♗×g2 43 ♖×a5 c5 44 ♖a1 ♗b7 45 ♖d1+ ♔c7 46 b4 c×b4 47 ♖b1 ♔c6 48 ♖×b4 ♗c8 49 ♖a4 ♔c5 50 ♔e5 ♗h3 51 ♖a3 ♗f1 52 ♖f3 ♗×c4 53 ♖c3 ♔b5 54 ♔d4 ♗e2 55 ♖c5+ ♔b6 56 ♔d5 ♗f3+ 57 ♔d6 ♗a8 58 ♖f5 ♗b7 59 ♖f8 ♔b5 60 ♖b8 ♔b6 61 ♔d7 ♔a6 62 ♔c7 ♗f3 63 ♔d6 ½-½

**Ostrich – Novag X** 1 e4 c5 2 c3 ♘f6 3 e5 ♘d5 4 d4 c×d4 5 c×d4 d6 6 ♘f3 ♘c6 7 ♗c4 e6 8 0-0 ♗e7 9 ♕e2 0-0 10 ♘c3 ♘×c3 11 b×c3 d×e5 12 ♘×e5 ♘×e5 13 ♕×e5 ♗d6 14 ♕e2 ♗d7 15 f4 ♖c8 16 ♖f3 ♕c7 17 ♗d3 ♖fd8 18 ♖b1 ♗c6 19 ♗e4 b6 20 ♗×c6 ♕×c6 21 ♖b5 h6 22 f5 ♖e8 23 f×e6 ♖×e6 24 ♕f1 ♖ce8 25 ♖d2 ♖e2 26 ♖b2 ♕c7 27 ♖h3 ♕e7 28 d5 ♗c5+ 29 ♔h1 ♖f2 30 ♕g1 0-1

Greco – Waycool 1 f4 e5 2 f×e5 d6 3 e×d6 ♗×d6 4 ♘f3 g5 5 d4 g4 6 ♘g5 f5 7 ♕d3 h6 8 ♕e3+ ♕e7 9 ♕×e7+ ♘×e7 10 h4 ♘bc6 11 c3 ♗d7 12 d5 ♘×d5 13 e4 ♘f4 14 g3 ♘h5 15 e×f5 ♘×g3 16 ♖g1 h×g5 17 ♗×g5 ♗×f5 18 ♗c4 ♘e4 19 ♔d1 ♘×g5 20 h×g5 ♖h5 21 ♖e1+ ♘e5 22 ♘d2 0-0-0 23 ♗d5 ♘d3 24 ♖e2 ♗e5 25 ♗g2 ♘×b2+ 26 ♔c1 ♗×c3 27 ♗e4 ♗×e4 28 ♖×e4 ♖×d2 0-1

*Round 4, October 27, 1987*

**Chiptest-m – Sun Phoenix** 1 e4 c6 2 d4 d5 3 ♘c3 g6 4 h3 ♗g7 5 ♘f3 ♘f6 6 e5 ♘e4 7 ♘×e4 d×e4 8 ♘g5 c5 9 d×c5 ♕a5+ 10 c3 ♕×c5 11 ♕d4 ♕×e5 12 ♕×e5 ♗×e5 13 ♗c4 0-0 14 0-0 ♗d7 15 ♖d1 ♗a4 16 ♖e1 ♘d7 17 ♗d5 ♘c5 18 ♘×e4 ♘d3 19 ♖e2 ♗c6 20 ♗×c6 ♘×c1 21 ♖×c1 b×c6 22 ♘c5 ♗f4 23 ♖ce1 ♖fb8 24 g3 ♗d6 25 ♘e4 ♗c7 26 f4 ♖d8 27 ♔g2 ♖d5 28 c4 ♖d4 29 c5 ♗a5 30 ♖f1 ♖ad8 31 ♖ff2 ♗b4 32 a3 ♗a5 33 ♘g5 ♖e8 34 f5 ♖d3 35 f×g6 f×g6 36 ♘f3 ♗c7 37 ♖e6 ♗b8 38 ♖×c6 ♖e3 39 g4 e5 40 ♘g5 ♖b3 41 ♖e6 ♖×e6 42 ♘×e6 h5 43 c6 e4 44 c7 ♗×c7 45 ♘×c7 1-0

**Cray Blitz – Cyrus 68K** 1 e4 c5 2 ♘f3 ♘c6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 d6 6 ♗g5 e6 7 ♗e2 ♗e7 8 0-0 0-0 9 ♗e3 ♘e5 10 f4 ♘g6 11 ♕d2 d5 12 e×d5 ♘×d5 13 ♘×d5 ♕×d5 14 ♗f3 ♕d7 15 ♖ad1 ♕a4 16 b3 ♕a6 17 ♗e2 ♕d6 18 ♘b5 ♕b8 19 ♕c3 ♗f6 20 ♕c4 ♗d8 21 ♖d2 a6 22 ♘d6 ♗c7 23 ♖fd1 ♖d8 24 ♕b4 a5 25 ♕c5 b6 26 ♕a3 b5 27 ♗×b5 ♘f4 28 ♗×f4 ♕b6+ 29 ♔h1 ♗a6 30 ♗×a6 ♖×a6 31 ♘×f7 ♖×d2 32 ♗×d2 1-0

**Novag X – Lachex** 1 e4 e5 2 ♘f3 ♘c6 3 d4 e×d4 4 ♘×d4 ♘f6 5 ♘×c6 b×c6 6 ♘d2 ♗c5 7 ♗d3 0-0 8 0-0 ♕e7 9 ♖e1 d6 10 ♘b3 ♘g4 11 ♘×c5 ♕e5 12 h3 ♕h2+ 13 ♔f1 ♕h1+ 14 ♔e2 ♕×g2 15 ♗e3 ♘×e3 16 ♔×e3 ♕g5+ 17 ♔e2 ♕×c5 18 ♖h1 f5 19 e×f5 ♗×f5 20 ♗×f5 ♖×f5 21 f3 ♖e8+ 22 ♔f1 ♕e3 23 ♔g2 ♖g5+ 24 ♔f1 ♖g3 25 h4 ♖×f3+ 26 ♕×f3 ♕×f3+ 27 ♔g1 ♖e4 28 ♖h3 0-1

**Waycool – BP** 1 e4 e5 2 ♘f3 ♘f6 3 ♘×e5 d6 4 ♘f3 ♘×e4 5 d4 d5 6 ♗d3 ♗e7 7 0-0 ♘c6 8 ♖e1 ♗g4 9 c4 ♘f6 10 c×d5 ♘×d5 11 ♘c3 0-0 12 ♗e4 ♗e6 13 ♕c2 h6 14 ♘×d5 ♗×d5 15 ♗f4 ♖e8 16 ♗×d5 ♕×d5 17 ♗×c7 ♖ec8 18 ♗f4 ♘×d4 19 ♕e4 ♘×f3+ 20 g×f3 ♕×e4 21 f×e4 ♖c2 22 ♖ab1 g5 23 ♗g3 h5 24 ♖ed1 ♖e8 25 ♖dc1 ♖×c1+ 26 ♖×c1 ♗f6 27 ♖c7 b5 28 e5 ♗g7 29 h3 h4 30 ♗h2 ♗×e5 31 ♗×e5 ♖×e5 32 ♖×a7 ♖e2 33 b3 f6 34 ♔g2 ♖b2 35 ♖a5 ♔f7 36 ♔f3 ♖d2 37 ♖×b5 ♖×a2 38 ♖d5 ♔e6 39 ♖c5 ♖b2 40 ♖c3 ♔d6 41 ♔g2 f5 42 ♖d3+ ♔e5 43 ♔f3 ♖c2 44 ♖e3+ ♔d4 45 b4 ♖b2 46 ♖e6 ♖×b4 47 ♖g6 ♖b3+ 48 ♔g2 f4 49 ♖×g5 f3+ 50 ♔h2 ♖b2 51 ♔g1 ♖b1+ 52 ♔h2 ♖f1 53 ♖d5+ 1-0 (crash)

**Bebe – Ostrich** 1 e4 c5 2 ♘f3 ♘c6 3 d4 c×d4 4 ♘×d4 e5 5 ♘b5 a6 6 ♘d6+ ♗×d6 7 ♕×d6 ♕f6 8 ♕d1 ♕g6 9 ♘c3 ♘ge7 10 h4 h5 11 ♗g5 d5 12 e×d5 ♘b4 13 ♗×e7 ♔×e7 14 ♗d3 ♕×g2 15 ♗e4 ♕g4 16 a3 ♕×d1+ 17 ♔×d1 f5 18 ♗f3 ♘×c2 19 ♔×c2 e4 20 ♗e2 ♗d7 21 ♖hg1 ♖h7 22 ♖g6 ♔f7 23 ♖b6 ♖a7 24 ♖g1 ♔e7 25 ♖g5 a5 26 ♔d2 ♔f8 27 ♗b5 1-0

**Greco – GNU Chess** 1 d4 d5 2 c4 e6 3 ♕a4+ ♘c6 4 ♘c3 ♗b4 5 c5 ♗d7 6 ♕d1 ♘f6 7 ♗f4 ♘e4 8 ♕d3 0-0 9 ♘f3 a6 10 ♖c1 f6 11 h4 e5 12 d×e5 ♗f5 13 ♕d1 ♗×c3+ 14 b×c3 f×e5 15 ♘×e5 ♘×e5 16 ♗×e5 ♘×f2 17 ♔×f2 ♗c2+ 18 ♔e3 ♗×d1 19 ♖×d1 ♖e8 20 ♔f4 ♖×e5 21 ♔×e5 ♕f6+ 0-1

# 9.2 Games from the World Computer Championships

Following the ever increasing success of the ACM tournaments in 1970, 1971, 1972 and 1973, it was decided to inaugurate an event of World Championship status. The original suggestion was my own, made in response to a comment by Ben Mittman during the 1973 ACM tournament in Atlanta: "This is great fun! What can we do next?"

The original concept, which has been retained ever since, was to emulate the world of human chess, in which the World Championship match took place every third year. The first World Computer Championship took place as part of the 1974 conference of the International Federation for Information Processing (IFIP), in Stockholm. Since then it has alternated between one side of the Atlantic and the other: Toronto in 1977, Linz in 1980, New York in 1983, and Cologne in 1986.

The World Championship contests have provided, among other things, an ideal forum in which to match the talents of chess programmers in the Eastern bloc with those in the West. When the first event took place in 1974, very little was known about computer chess in the USSR, even though chess itself had been the most popular game there for decades. The 1974 World Computer Championship was dominated by the presence of the Russian program Kaissa, which won the event after much excitement. Three years later, in Toronto, Kaissa finished in a tie for second place. In 1980 the program was tied for sixth. The reason for the decline in Kaissa's fortunes is not difficult to find. Programmers in the West are fortunate in having access to computer time almost without limit. Since the late 1970s, the popularity and availability of home computers has brought chess programming within the reach of almost every pocket. But in the East there is a comparative dearth of computers, and so computer time is not readily available, though in the mid 1980s home computers appear to be reaching Eastern Europe in ever increasing numbers and so this handicap is being slowly reduced.

**First World Computer Championship, Stockholm, Sweden, August 5th - 8th 1974**

| | 1 | 2 | 3 | 4 | total |
|---|---|---|---|---|---|
| 1 Kaissa (USSR) | 7+ | 5+ | 4+ | 6+ | 4 |
| 2 Chess 4.0 (USA) | 3+ | 4– | 6+ | 5+ | 3* |
| 3 Ribbit (Canada) | 2– | 10+ | 13+ | 9+ | 3 |
| 4 Chaos (USA) | 10+ | 2+ | 1– | 8+ | 3 |
| 5 Tech II (USA) | 8+ | 1– | 7+ | 2– | 2 |
| 6 Ostrich (USA) | 11+ | 9+ | 2– | 1– | 2 |
| 7 Frantz (Austria) | 1– | 13+ | 5– | 12+ | 2 |
| 8 Master (England) | 5– | 12+ | 11+ | 4– | 2 |
| 9 Beal (England) | bye | 6– | 12+ | 3– | 2 |
| 10 Freedom (Norway) | 4– | 3– | bye | 11= | 1½ |
| 11 Tell (Switzerland) | 6– | bye | 8– | 10= | 1½ |
| 12 A16CHS (England) | 13+ | 8– | 9– | 7– | 1 |
| 13 Papa (Hungary) | 12– | 7– | 3– | bye | 1 |

* Chess 4.0 and Ribbit finished ahead of Chaos on tie-break. These two programs played off for second place, and the playoff game was won by Chess 4.0.

*Round 1, August 5, 1974*

**Tell – The Ostrich** 1 e4 e5 2 ♘f3 d5 3 ♘c3 d4 4 ♘b5 a6 5 ♘a3 ♗×a3 6 ba ♘c6 7 ♖b1 ♘f6 8 d3 0-0 9 ♗d2 ♕d6 10 ♖b3 b5 11 ♘g5 h6 12 ♘f3 ♗e6 13 c4 dc 14 ♖×c3 ♘d4 15 ♕b1 c5 16 ♗e3 ♘g4 17 ♗d2 f5 18 h3 fe! 19 hg ef 20 ♖h4 ♕d5 21 g3 ♖ac8 22 ♖h5 ♗×g4 23 ♖h4 h5 24 ♗h3 ♕e6 25 ♗×g4 hg 26 ♗e3 ♘f5 27 ♖h5 ♘×e3 28 fe c4! 29 ♔f2 ♖fd8 30 dc ♖d2+ 31 ♔e1 ♖e2+ 32 ♔f1 bc 33 a4 e4 34 a3 ♕d6 35 ♖f5 ♕×g3 36 ♖f8+ ♖×f8 37 ♕b3 ♕f2 mate 0-1

**Kaissa – Frantz** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 d6 4 d4 ed 5 ♕×d4 ♘e7 6 0-0 f6 7 ♗f4 ♗e6 8 ♘c3 ♕c8 9 ♖ad1 ♗f7 10 ♕b4? a6 11 ♗×c6+ ♘×c6 12 ♕a4

b5 13 ♕a3 d5 14 b4 ♗×b4 15 ♕b2 de 16 ♖fe1 f5? 17 ♘e5 ♘×e5 18 ♗×e5 ♗×c3 19 ♗×c3? ♖g8 20 f3! ♕b7 21 ♗×g7 ♕b6+ 22 ♗d4 ♕g6 23 g3 0-0-0 24 fe fe 25 ♗f6 ♖d5! 26 ♖×d5 ♗×d5 27 ♕e5 ♕f7 28 ♖d1 ♗×a2?? 29 ♕×e4 ♔b8 30 ♗e5 ♖e8 31 ♕c6 ♕g6 32 ♕×c7+ ♔h8 33 ♖d7 ♕f5 34 ♕c6 mate 1-0

**Papa – A16CHS** 1 ♘f3 ♘f6 2 d4 ♘e4 3 d5 e6 4 ♗f4 ♗b4+ 5 c3 ♗a5 6 ♕d4 ed?? 7 ♕×d5 ♘c6 8 ♗×c7?? ♗×c7 9 ♕×e4+ ♘e7 10 e3 d5 11 ♗b5+ ♔f8 12 ♕a4 a6 13 ♘d4 ♕d6 14 h3 h5 15 ♘d2 g5 16 h4 f6 17 ♖c1 ♘g6 18 hg ♗g4 19 ♗d7?? ♗×d7 20 ♕d1 ♗g4 21 f3 ♗d7 22 e4 ♘f4 White lost on time 0-1

**Chess 4.0 – Ribbit** 1 e4 c5 2 ♘f3 d6 3 d4 cd 4 ♘×d4 ♘f6 5 ♘c3 a6 6 ♗e2 e5 7 ♘b3 ♘c6 8 0-0 ♗e7 9 ♗g5 ♗e6 10 ♕d2 0-0 11 ♖ad1 ♖c8 12 ♗×f6 gf 13 ♔h1 ♕d7 14 ♘d5 ♗d8 15 ♕e3 f5 16 ♕g3+ ♔h8 17 f3 ♖g8 18 ♕f2 fe? 19 fe ♖g7 20 c4 ♕e8 21 ♘e3 ♕d7 22 ♘c5 ♕e7 23 ♘×e6 ♕×e6 24 ♘f5 ♖g8 25 ♘×d6 ♖c7 26 ♘×f7+ ♔g7 27 ♕g3+ ♕g6 28 ♕×g6+ hg 29 ♘×d8 ♘×d8 30 ♖d6 ♘f7 31 ♖d5 ♖f8 32 c5 ♖c6 33 ♖d7 ♖×c5 34 ♖×b7 a5 35 b3! ♖c6 36 ♖a7 ♖f6 37 ♖×f6 ♔×f6 38 ♖a6+ ♔g7 39 ♖×a5 Black lost on time 1-0

**Freedom – Chaos** 1 d4 ♘f6 2 c4 e6 3 c5 d6! 4 cd ♕×d6 5 ♘c3 ♘c6 6 e3 ♗d7 7 ♗b5 ♗e7 8 ♘h3? 0-0 9 ♔f1? e5! 10 ♗d2 ed 11 ed ♗×h3 12 gh ♘×d4 13 ♗c4?? ♕c6 14 ♗d5 ♘×d5 15 ♘×d5 ♕×d5 16 ♖g1 ♘f3?? 17 ♖×g7+ ! ♔h8 18 ♔e2?? ♕e4+ 19 ♔f1 ♔×g7 20 ♗c3+ f6 21 ♔g2 ♘h4+ 22 ♔f1 ♕g2+ 23♔e2 ♕e4+ 24 ♔f1 ♕h1+ 25 ♔e2 ♕f3+ 26 ♔f1 ♕×d1+ 27 ♖×d1 ♗d6 28 ♖d5 ♗×h2 White lost one time 0-1

**Master – Tech II** 1 e4 e5 2 f4 ef 3 ♘f3 ♘f6 4 e5 ♘d5 5 ♘c3 ♘×c3 6 dc g5 7 ♕d5 ♘c6 8 ♗c4 ♕e7 9 ♔d1 g4 10 ♗×f4? gf 11 gf d6? 12 ed cd 13 ♖e1 ♗e6 14 ♕h5 ♗×c4 15 ♖×e7+ ♗×e7 16 b3 ♗f1? 17 ♔d2 ♗a6 18 ♖e1 ♖d8 19 ♖e4 ♗f1 20 ♗g5 ♘e5 21 ♗e3?? a5 22 ♗b6 ♖c8 23 ♖×e5? de 24 ♕×e5 ♖g8 25 ♗c7 ♖g5 26 ♕f4 ♔d7 27 ♗b6 ♔e6 28 c4 ♖c6 29 ♗d4 h6 30 ♔c1 a4 31 ♔b1 ab 32 ab f6 33 h4 ♖g6 34 ♔b2 ♗d6 35 ♕e4+ ♔f7 36 ♕d5+ ♔e7 37 ♕e4+ ♗e5 38 ♗×e5?? fe 39 ♕×e5+ ♔d7 40 ♕f5+ ♖ce6 41 ♕d5+ ♔c7 42 ♕c5+ ♖c6? 43 ♕e7+ ♔b6 44 ♕d8+ ♔a6? 45 ♕a8+ ♔b6 46 ♕e8 ♔c5? 47 ♕e3+ ♔d6 48 ♕e8 ♔c7 49 ♕e5+ ♔d7 50 ♕d5+ ♔e7 51 ♕d4 ♗e2 52 ♕e4+ ♖ge6 53 ♕g4 ♔d6 54 ♕g3+ ♔c5 55 ♕f2+ ♔d6 56 ♕g3+ ♔d7 57 ♕g4 ♖c5 58 c3 ♔d6 59 ♕f4+ ♖ce5 60 c5+ ♔×c5 61 b4+ ♔d5 62 ♕f8 ♖c6 63 ♕f7+ ♔d6 64 ♕f6+ ♖e6 65 ♕d8+ ♔e5 66 ♕d4+ ♔f5 67 ♕g4+ ♔f6 68 h5 ♔e5? 69 ♕d4+ ♔f5 70 ♕g4+ ♔f6 71 ♔c2 ♔e5? 72 ♕g3+ ♔d5 73 ♕f4 ♖c4 74 ♕f8 ♖c7 75 ♕d8+ ♔c6 76 ♕f8 ♗c4 77 ♕c5+ ♔d7 78 ♕f5 ♔d6 79 ♕f4+ ♖e5 80 ♕d4+? ♗d5 81 ♕f4 ♖f7! 82 ♕×h6+ ♔d7 83 ♕d2 ♖×f3 84 c4 ♖fe3! 85 ♕d1 ♖e2+ 86 ♔b3 b5! 87 h6 ♖5e3+ 88 ♕d3 ♖×d3 mate 0-1

*Round 2, August 6, 1974*

**The Ostrich – Beal** 1 ♘f3 ♘f6 2 d4 ♘c6 3 d5 ♘b4 4 ♘c3 d6 5 a3 ♘a6 6 e4 e5 7 ♗g5 ♘c5 8 b4 ♘cd7 9 ♗b5 ♗e7 10 ♗×f6 gf? 11 0-0 0-0 12♕d3 ♔g7? 13 ♘h4 ♖e8 14 f4 c5 15 ♘f5+ ♔g6 16 ♕g3+ ♔h5 17 ♗e2 mate 1-0

**Tech II – Kaissa** 1 e4 d5 2 ed ♘f6 3 ♗b5+ ♗d7 4 ♗c4 ♗g4 5 f3 ♗c8 6 ♘c3 ♘bd7 7 ♘e4 ♘e5 8 ♘×f6+ ef 9 ♕e2 ♕e7 10 ♗b5+ c6 11 dc bc 12 ♗a4 ♗a6! 13 ♕e4?? 0-0-0? 14 ♘e2 ♗×e2 15 ♔×e2 ♕d7 16 d3 ♖e8 17 ♗e3 ♗d6 18 c3? ♗b8 19 ♗c2 ♘g6 20 ♕b4 ♘f4+ 21 ♔f2 ♖×e3 22 ♔×e3 ♘d5+ 23 ♔e2 ♘×b4 24 cb ♕d4 25 ♖ab1 ♖e8+ 26 ♔f1 ♕e3 27 d4 ♕e2+ 28 ♔g1 ♕×c2 29 ♔f1 ♕×b1+ 30 ♔f2 ♕×b2+ 31 ♔f1 ♕e2+ 32 ♔g1 ♕d1+ 33 ♔f2 ♖e2 mate

**Chaos – Chess 4.0** 1 d4 d5 2 c4 dc 3 ♘f3 ♘f6 4 e3 e6 5 ♗×c4 c5 6 ♕e2 a6 7 0-0 b5 8 ♗b3 ♗b7 9 ♖d1 ♘bd7 10 ♘c3 ♗d6 11 e4 cd 12 ♘×d4 ♕b8 13 g3 b4 14 ♘a4 ♗×e4 15 f3 ♗g6? 16 ♘×e6!! fe 17 ♕×e6+ ♗e7 18 ♖e1 ♕d8 19 ♗f4 ♔f8 20 ♖ad1 ♖a7 21 ♖c1 ♘g8 22 ♖cd1 a5 23 ♗d6 ♗×d6 24 ♕×d6+ ♘e7 25 ♘c5 ♗f5 26 g4 ♕e8 27 ♗a4 b3 28 gf ba 29 ♗×d7 a1=♕ 30 ♖×a1 ♖a6 31 ♘×a6 ♕d8 32 ♔f2 ♔f7 33 ♕e6+ ♔f8 34 ♕×e7+ ♕×e7 35 ♖×e7 ♔×e7 36 ♘c5 ♖b8 37 ♖×a5 ♖×b2+ 38 ♔g3 g6 39 fg hg 40 ♖a6 ♖c2 41 ♖e6+ ♔f8 42 ♖e5 ♖c1 43 ♖g5 ♔f7 44 ♗e6+ ♔f6 45 h4 ♖×c5 46 ♖×c5 ♔×e6 47 ♖g5 ♔f6 48 ♔g4 ♔f7 49 ♖c5 ♔e6 50 ♔g5 ♔d6 51 ♖a5 ♔c6 52 f4 ♔b6 53♖a1 ♔c5 54 ♖d1 ♔b4 55 ♔×g6 ♔c3 56 ♖d8 ♔b4 57 ♖c8 ♔b5 58 h5 ♔b6 59 ♖c1 ♔b5 60 h6 ♔a4 61 ♖b1 ♔a3 62 f5 ♔a2 63 ♖b8 ♔a3 64 f6 ♔a4 65 ♖b7 ♔a5 66 ♖b8 ♔a4 67 ♖b1 ♔a3 68 ♖b7 ♔a4 69 ♖b8 ♔a5 70 ♔g7 ♔a4 71 ♖b7 ♔a5 72 ♖b2 ♔a4 73 ♖b8 ♔a5 74 ♔g8 ♔a4 75 h7 ♔a5 76 h8 = ♕♔a4 77 ♕h4+ ♔a5 78 ♕b4+ ♔a6 79 ♕a4 mate 1-0

**A16CHS – Master** 1 e4 c5 2 ♘f3 ♘f6 3 e5 ♘d5 4 c4 ♘c7 5 d4 cd 6 g4? d6 7 ♘×d4 de 8 ♘f5 ♕×d1+ 9 ♔×d1 ♗×f5 10 ef ♘c6 11 ♗d3?? 0-0-0 12 ♔d2 ♘b4 13 ♖d1 ♖×d3+ 14 ♔e2 ♖×d1 15 ♗e3?? ♖h1 16 ♗×a7 ♘c2 17 ♘c3 ♘×a1 18 b4 e6 19 a3 ♘c2 20 ♘b5 ♘×b5 21 cb ef 22 h4 ♘×a3 23 b6 ♗×b4 24 ♔d3 ♖d8+ 25 ♔e3 ♘c4+ 26 ♔f3 ♖×h4 27 ♔g3 ♖g4+ 28 ♔h3 ♖d3+ 29 ♔h2 ♖a3 30 ♔h1 ♖h3 mate 0-1

**Frantz – Papa** 1 d4 e6 2 e4 ♗b4+? 3 c3 ♗f8 4 f3 ♗d6 5 ♘e2 f5 6 ♘d2 ♕h4+ 7 g3 ♗×g3+ 8 ♘×g3 f4 9 ♖g1 fg 10 ♖×g3 ♘c6 11 ♗c4 a6 12 ♕e2 b5 13 ♗b3 ♘f6 14 d5 ♘e5 15 f4 ♘h5 16 fe ♘×g3 17 hg ♕×g3+ 18 ♕f2 ♕×f2+ 19 ♔×f2 ♗b7 20 de ♖f8+ 21 ♔g3 de 22 ♗×e6 ♖d8 23 ♗g4 ♖d3+ 24 ♔g2 ♖f4 25 ♗e6 ♗c6 26 ♗g8 ♖f5 27 ♗×h7 ♖g5+ 28 ♔f2 b4 29 ♘f3 ♖g3?? 30 ♔×g3 Black lost on time 1-0

**Ribbit – Freedom** 1 e4 e5 2 ♘f3 ♘f6 3 ♘×e5 ♘×e4? 4 ♕e2 d5 5 d3 ♗b4+ ? 6 c3 ♗d6 7 ♘f3 ♘c6 8 de d4 9 cd 0-0 10 d5 ♘a5 11 ♗d2 c6?? 12 b4 ♖e8 13 ba cd 14 e5 ♗e7 15 ♕b5 b6 16 ab ab 17 ♘c3 ♗a6 18 ♕×d5 ♗×f1 19 ♕×d8 ♖×d8 20 ♖×f1 ♖d3 21 ♖b1 ♗c5 22 ♔e2 ♖dd8 23 ♗e3 ♖a3 24 ♖b3 ♖a5 25 ♗×c5 bc 26 ♖fb1 Black lost on time 1-0

*Round 3, August 7, 1974*

**Kaissa — Chaos** 1 e4 c5 2 ♘f3 ♘c6 3 c3 d5 4 ed ♕×d5 5 d4 ♗g4 6 ♗e2 e6 7 0-0 ♘f6 8 ♗e3 cd 9 ♗×d4 e5 10 h3 ed 11 hg ♗d6 12 cd ♘×g4 13 ♘c3 ♕h5 14 g3 ♔d7?? 15 ♘h4 f5 16 d5 ♘ce5 17 ♕c2 ♖hf8 18 ♗d3 ♘×d3 19 ♕×d3 ♖ae8 20 ♘b5 f4 21 ♘×d6 ♔×d6 22 ♕a3+ ♔c7 23 ♕×a7 ♕f7 24 ♖fc1+ ♔d6 25 ♕c5+ ♔e5 26 d6+ ♔e6 27 ♖e1+ ♘e3 28 gf ♕d7 29 f5+ ♔f6 30 ♖×e3 ♖d8 31 ♖e7 ♕a4 32 ♕e5+ ♔g5 33 ♘f3+ ♔g4 34 ♖×g7+ ♔h5 35 ♕h2+ ♕h4 36 ♕×h4 mate 1-0

**Chess 4.0 — The Ostrich** 1 e4 e6 2 d4 d5 3 ♘c3 de 4 ♘×e4 ♗e7 5 ♗f4 f5? 6 ♘c5? c6? 7 ♗e5 ♗×c5?? 8 ♗×g7 ♗×d4 9 ♕×d4 ♕×d4 10 ♗×d4 ♘f6 11 ♗×f6 0-0 12 ♗c3 b5 13 0-0-0 a5 14 ♗e2 b4 15 ♗e5 ♘d7 16 ♘f3 ♘×e5 17 ♘×e5 c5 18 ♘d7 ♗×d7 19 ♖×d7 ♖f7 20 ♖hd1 f4 21 ♖×f7 ♔×f7 22 ♖d7+ ♔f8 23 ♖×h7 ♖e8 24 ♖c7 c4 25 ♖×c4 e5 26 ♖c5 e4 27 ♖×a5 ♖e7 28 ♖b5 e3 29 ♖×b4 ef 30 ♖×f4+ ♖f7 31 ♖×f7+ ♔×f7 32 ♔d2 ♔e6 33 g4 ♔d5 34 ♔e3 f1= ♕35 ♗×f1 ♔d6 36 b4 ♔d7 37 h4 ♔c7 38 ♔e4 ♔d7 39 a4 ♔e7 40 b5 ♔f6 41 b6 ♔e7 42 b7 ♔d7 43 b8= ♕ ♔c6 44 a5 ♔d7 45 a6 ♔c6 46 a7 ♔c5 47 a8=♘! ♔c648 ♕c7 mate 1-0

**Beal — A16CHS** 1 g3 g6 2 ♗h3 c5 3 ♗g2 ♕b6 4 c3 ♕c7 5 ♔f1 ♗g7 6 ♗f3 g5 7 e4 ♘c6 8 d4 cd 9 ♗×g5 d6 10 cd h6 11 ♗e3 ♕b6 12 ♘c3 ♕a6+ 13 ♘ge2 ♗h3+ 14 ♔e1 b5 15 ♘d5 ♕a5+ 16 ♘ec3 b4 17 ♘b5 ♔f8 18 ♕d3 a6 19 ♘bc7 ♖a7 20 ♖c1 ♗d7 21 e5 f6?? 22 e6 ♘b8 23 ♕g6 ♕a4 24 ♕f7 mate 1-0

**Frantz — Tech II** 1 d4 d5 2 ♘f3 ♘f6 3 ♗f4 ♘c6 4 ♘c3 ♗f5 5 ♘b5 ♖c8 6 e3 e6 7 ♗d3 ♗b4+ 8 c3 ♗×d3 9 ♕×d3 ♗e7 10 ♘e5 ♘×e5 11 de ♘h5 12 ♘×a7 ♘×f4 13 ef ♖a8 14 ♕b5+ ♔f8 15 ♕×b7 ♖b8 16 ♕c6 ♖b6 17 ♕a4 ♖×b2 18 ♘c6?? ♕e8 19 ♖d1 ♖b6 20 c4 ♖×c6 21 cd ed 22 ♕a7 ♗b4+ 23 ♔e2 ♖c2+ 24 ♔f1 ♕b5+ 25 ♔g1 ♕c5 26 ♕a8+ ♔e7 27 ♕f8+ ♖×f8 28 g3 ♖×f2 29 ♖d3 ♖d2+ 30 ♔f1 ♕c1 mate 0-1

**Master — Tell** 1 e4 e5 2 ♘f3 ♘c6 3 ♗c4 ♗c5 4 c3 d6 5 d4 ed 6 cd ♗b4+ 7 ♔f1 h5?? 8 ♕b3?? ♗g4?? 9 ♗×f7+ ♔f8 10 ♗×g8 ♖×g8 11 ♕d5 ♗×f3 12 gf ♖h8 13 ♘c3 ♗×c3 14 bc a5 15 ♖b1 ♕c8 16 ♗f4 ♕h3+ 17 ♔e2 ♘d8 18 ♗×d6+ cd 19 ♕×d6+ ♔f7 20 ♖hg1 ♖g8 21 ♖b5 g6 22 ♕c7+ ♔e8 23 ♖e5+ ♘e6 24 ♕×b7 ♖d8 25 ♖g3 ♕×h2 26 ♖×e6+ ♔f8 27 ♕e7 mate 1-0

**Papa — Ribbit**1 ♘f3 d5 2 d4 ♘c6 3 ♘e5 ♘×e5 4 de e6 5 e4?! de 6 ♗b5+ ♗d7 7 ♗×d7+ ♕×d7 8 ♕×d7+ ♔×d7 9 0-0 f5 10 ♖d1+ ♔c6 11 ♗e3 ♗c5? 12 ♗×c5 ♔×c5 13 ♖d7 ♘h6 14 ♖×c7+ ♔b6 15 ♖×g7 ♘g4 16 f4 ef 17 gf ♘×e5 18 ♘d2? ♖ag8 19 ♘c4+ ?? ♘×c4 20 ♖g6?? hg 21 ♖d1 ♖d8 22 ♖×d8 ♖×d8 23 a3 ♘×b2 24 a4 ♘×a4 25 c3 ♘×c3 26 f4 ♘d5 27 ♔f2 ♘×f4 28 ♔f3 e5 29 h4 ♖d4 30 ♔g3 ♘h5+ 31 ♔f2 ♖×h4 0-1 (46 moves)

*Round 4, August 8, 1974*

**The Ostrich — Kaissa** 1 ♘f3 e6 2 d4 ♘f6 3 ♗g5 d5 4 e3 ♗e7 5 ♘c3 ♗b4?! 6 ♗×f6?! ♗×c3+ ?! 7 bc ♕×f6 8 ♗d3 c5 9 0-0 0-0 10 ♕d2 ♘c6 11 dc?! ♕e7 12 c4! dc 13 ♗×c4 ♕×c5 14 ♕d3 ♖d8 15 ♕e4 b5? 16 ♗d3 f5? 17 ♕h4 e5 18 e4 f4 19 ♖fe1 ♗b7?? 20 ♘g5 h6 21 ♘e6 ♕b6 22 ♘×d8 ♖×d8 23 a4 b4 24 ♗c4+ ♔h8 25 ♖ad1 ♘d4 26 ♖c1 ♗c6 27 c3! bc 28 ♖×c3 ♗×a4 29 ♕e7 ♘c6 30 ♕f7 ♕c5 31 ♖d3 ♘d4 32 ♗d5 ♗b5? 33 ♖h3 ♘e2+ 34 ♔h1 ♕×f2 35 ♖d1?? ♕b6 36 ♖b1 ♖c8 37 ♗e6! ♖d8 38 ♕g6 ♕b7 39 ♕f5?? ♕c7 40 ♖h4?? ♘d4 41 ♕h3 ♘×e6 42 ♕×e6 ♗d3 43 ♖g1 ♗c4? 44 ♕f5 ♗e2 45 ♖a1 a5 46 ♕g6 a4 47 ♖e1 ♗c4 48 ♖a1 a3 49 ♖b1 ♕d6 50 ♕×d6 ♖×d6 51 ♖h3 a2 52 ♖c1 ♖d4 53 ♖hc3 ♖×e4 54 ♖a1 ♖d4 55 ♖×c4 ♖×c4 56 g3 f3 57 h3 ♖c2 58 ♖d1 ♖d2 59 ♖c1 e4 60 g4 e3 61 ♔g1 e2 62 ♔f2 ♖d1 63 ♖c8+ ♔h7 64 ♔×f3 e1=♕ 65 ♖c2 ♖d3+ 66 ♔f4 g5+ 67 ♔f5 ♖f3 mate 0-1

**Tech II — Chess 4.0** 1 e4 c5 2 ♘f3 ♘c6 3 ♘c3 ♘f6 4 d4 cd 5 ♘×d4 d6 6 ♗c4 e6 7 0-0 a6 8 ♗e3 d5 9 ♘×c6? bc 10 ed cd 11 ♕f3 ♗d6 12 ♗d3 0-0 13 ♖fe1 ♕c7 14 g3 ♗b7 15 ♗d4?? e5 16 ♗×e5 ♗×e5 17 ♗f5 g6 18 ♗d3 ♗×c3 19 bc ♕×c3 20 ♕f4 ♖fe8 21 ♖×e8+ ♖×e8 22 ♖b1 ♖e1+ 23 ♖×e1 ♕×e1+ 24 ♔g2 ♕c3 25 ♔f3 d4+ 26 ♔e2 ♘d7 27 ♗e4 ♗×e4 28 ♕×e4 ♘f6 29 ♕d3 ♔g7 30 ♕×c3 bc 31 a3 ♘d5 32 f3 ♔f6 33 ♔d3 ♔e5 34 h3 g5 35 a4 h5 36 a5 f5 37 g4 ♘f4+ 38 ♔×c3 ♘×h3 39 gh ♘g1 40 h6 ♔f6 41 ♔d4 ♘×f3+ 42♔c5 f4 43 h7 ♔g7 44 h8=♕+ ♔×h8 45 ♔d5 ♘e5 46 ♔e4 ♘c4 47 c3 ♔g7 48 ♔f3 ♔f6 49 ♔e4 ♔e6 White lost on time 0-1

**Ribbit — Master** 1 e4 c5 2 c3 d5 3 ed ♕×d5 4 d4 ♘c6 5 dc ♕×d1+ 6 ♔×d1 ♗g4+ 7 ♗e2 0-0-0+ 8 ♗d2 ♘f6 9 ♗×g4+ ♘×g4 10 ♔e2 e6 11 b4 ♗e7 12 ♘f3 f5 13 ♘a3 ♖d5 14 h3 ♘ge5 15 ♘×e5 ♘×e5 16 ♗e3 ♖hd8 17 ♖ad1 a6 18 ♖×d5 ♖×d5 19 ♖e1 ♔d7?? 20 g3?? ♗f6?? 21 f4 b5 22 fe ♗×e5 23 ♖d1 ♖×d1 24 ♔×d1 ♗×g3 25 c4 bc 26 ♘×c4 e5 27 ♔e2 ♔e6 28 ♗d2 e4 29 ♘e3 ♗e5 30 ♘c4 ♗d4 31 c6 g5 32 ♗×g5 ♗c3 33 c7 ♔d7 34 ♘b6+ ♔×c7 35 ♘d5+ ♔b7 36 ♘×c3 ♔c7 37 ♘d5+ ♔c6 38 ♘e3 ♔b5 39 ♘×f5 ♔×b4 40 ♔e3 ♔a3 41 ♔×e4 ♔×a2 42 ♔d3 ♔b3 43 ♗d2 ♔a3 44 ♔c4 ♔a2 45 ♗c3 h5 46 ♘g7 h4 47 ♘f5 ♔a3 48 ♘×h4 ♔a2 49 ♘g6 ♔b1 50 ♔b3 ♔c1 51 ♘f4 ♔b1 52 ♘d3 a5 53 ♗×a5 ♔a1 54 h4 ♔b1 55 h5 ♔a1 56 h6 ♔b1 57 h7 ♔a1 58 h8=♕+ ♔b1 59 ♕b2 mate 1-0

**Chaos — Beal** 1 d4 ♘f6 2 c4 ♘c6 3 ♘c3 e5 4 d5 ♘d4 5 e3 ♘f5 6 e4 ♘d4 7 ♘f3 ♘×f3+ 8 gf ♗c5 9 ♗e3 ♗×e3 10 fe 0-0 11 d6 cd 12 ♕×d6 ♕b6? 13 ♕×b6 ab 14 ♖d1 ♖a5 15 ♗e2 ♖c5 16 ♖g1 ♖e8 17 ♔f2 ♖e6 18 b4 ♖cc6 19 ♘d5 h5 20 a4 h4 21 h3 ♘h5 22 ♖g4 ♘g3 23 ♗d3 f5 24 ef ♖h6 25 f6 e4 26 ♘e7+ ♔f8 27 fg+ ♔×e7 28 g8=♕ ed 29 b5 d5 30 bc ♗×g4 31 ♕g7+ ♔e8 32 ♕×h6 ♗c8 33 ♕×h4 ♘f5 34 ♕h8+ ♔e7 35 ♕×c8 bc 36 ♕×f5 dc ♕e4+ ♔f6 38 ♕×c6+ ♔e5 39 f4+ ♔f5 40 ♖×d3 cd 41 ♔f3 d2 42 e4 mate 1-0

**A16CHS — Frantz** 1 e4 e5 2 ♘f3 ♘c6 3 ♗c4 ♗c5 4 ♘c3 ♘f6 5 d3 0-0 6 ♗e3 ♗×e3 7 fe d6 8 h4 ♕e7 9 ♘d5 ♘×d5 10 ed ♘d8 11 a4 f5 12 g3 ♖f6 13 c3 a5 14 b4 ab 15 cb ♗d7 16 ♗b3 ♖a6 17 b5 ♖b6 18 ♗c4

♗e8 19 ♕b3 ♕d7 20 d4?? ♘f7 21 ♖b1 f4 22 ef ♕g4 23 de ♕×g3+ 24 ♔d2 ♕×f4+ 25 ♔e2 ♕e4+ 26 ♔f2 ♖×f3+ 27 ♕×f3 ♕×c4 28 e6 ♕×a4?? 29 ef+ ♗×f7 30 ♕b3 ♕×b3 31 ♖×b3 ♗×d5 32 ♖e1?? ♗×b3 33 ♖c1 ♖×b5 34 ♖b1 c6 35 ♖a1 g5 36 h5 g4 36 h5 g4 37 h6 b6 38 ♖a8+ ♔f7 39 ♖h8 ♗c2 40 ♖d8 ♔e6 41 ♖e8+ ♔f6 42 ♖f8+ ♔e7 43 ♖b8 ♖b3 44 ♖b7+ ♔e6 White lost on time 0-1

**Freedom – Tell** 1 d4 d5 2 c4 dc 3 ♘c3 ♘c6 4 d5 ♘e5 5 f4 ♘d7 6 e4 ♘b6 7 ♗e3 ♕d6 8 e5 ♕b4 9 e6 fe 10 de ♗×e6 11 g3? ♕×b2 12 ♖c1 ♖d8 13 ♕e2 ♕a3 14 ♘b1?? ♕a5+ 15 ♘c3 ♗d5 16 ♘f3 ♘a4 17 ♗d4 ♗×f3 18 ♕×f3 ♖×d4 19 ♕×b7 ♘×c3 20 ♕c6+ ♖d7 21 ♕a8+ ♖d8 22 ♕c6+ ♖d7 23 ♕a8+ ♖d8? 24 ♕c6+ ♖d7 ½-½

*Play-off for Second Place*

**Ribbit – Chess 4.0** 1 e4 e5 2 ♘f3 ♘f6 3 ♘×e5 d6 4 ♘f3 ♘×e4 5 d4 d5 6 ♗d3 ♘c6 7 ♕e2 ♗f5 8 ♘c3 ♗b4 9 a3? ♗×c3+ 10 bc 0-0 11 ♗b2 ♖e8 12 0-0 ♘×c3 13 ♕d2 ♗×d3 14 ♕×d3 ♘e2+ 15 ♔h1 ♕f6 16 ♖fe1 ♘f4 17 ♕b3 ♖×e1+ 18 ♖×e1 b5? 19 g3 ♘h3 20 ♔g2 ♕f5 21 ♕×b5 ♕d7 22 ♘e5 ♘f4+ 23 gf ♘×e5 24 ♕×d7 ♘×d7 25 ♗c3 ♘f6 26 ♖e7 ♖c8 27 h4? ♔f8 28 ♖e5 g6 29 ♗b4+ ♔g7 30 ♖e7 a6 31 ♗c3 c5 32 dc ♖×c5 33 ♗×f6+ ♔×f6 34 ♖e5 h6 35 c4 ♖×c4 36 ♖×d5 ♖×f4 37 ♔g3 ♖a4 38 ♖d3 ♔e5 39 ♖c3 f5 40 f3 ♔d5 41 f4 ♔e4 42 ♖b3 a5 43 ♖c3 h5 44 ♖f3 ♖c4 45 ♖f1 ♖c3+ 46 ♔g2 ♖×a3 47 ♖g1 a4 48 ♖f1 ♖e3 49 ♖f2 a3 Adjudicated a win for Black.

*Exhibition Game (9th August 1974)*

**Chess 4.0 – Kaissa** 1 e4 d5 2 ed ♘f6 3 d4 ♘×d5 4 ♘f3 g6 5 ♗e2 ♗g7 6 0-0 0-0 7 ♖e1 ♗f5 8 ♘h4 e5? 9 ♘×f5 gf 10 de ♘b4 11 ♕×d8 ♖×d8 12 ♗g5 ♖d7 13 ♘a3 ♗×e5 14 c3 ♘4c6 15 ♘c4 a5 16 ♗f3 f6 17 ♗h6 a4 18 ♖ad1 ♖×d1 19 ♖×d1 ♔h8? 20 ♗×c6 ♘×c6 21 f4 b5 22 fe bc 23 ef ♖d8 23 ef ♖d8 24 ♖f1 ♔g8 25 ♖×f5 ♖d1+ 26 ♔f2 ♘d8 27 ♗f4 c6 28 ♔f3 ♖f1+ 29 ♔e4 ♖a1 30 a3 ♖e1+ 31 ♗e3 ♖e2 32 ♖f2 ♖e1 33 ♖d2 ♘e6 34 ♖d6 ♘c5+ 35 ♔f3 ♘d3 36 ♗d4 c5 37 ♗e3 ♔f7 38 ♖d7+ ♔g6 39 ♖g7+ ♔×f6 40 ♖×h7 ♘e5+ 41 ♔f4 ♘d3+ 42 ♔e4 ♘×b2 43 g4?? ♘d1 44 g5+ ♔g6 45 ♖h6+ ♔g7 46 ♔d5 ♖×e3 47 ♔×c4 ♖×c3 48 ♔b5 ♖×a3 49 h4 ♖h3 50 ♔×c5 ♘b2 51 h5 a3 52 ♖g6+ ♔f7 53 ♖f6+ ♔g8 54 ♖g6+ ♔f7 55 ♖f6+ ♔e7 56 h6 ♘a4+ 57 ♔b4 a2 58 ♖f1 ♘c3 59 ♔b3 a1=♕ 60 ♖×a1 ♘e4+ 61 ♔c4 ♘×g5 62 ♖a6 ♘f7 63 ♖a7+ ♔e6 64 ♖a6+ ♔f5 65 ♔d4 ♘×h6 Adjudicated ½-½

---

**Second World Computer Championship Toronto, Canada, August 7th - 9th 1977**

| | | 1 | 2 | 3 | 4 | total |
|---|---|---|---|---|---|---|
| 1 | Chess 4.6 (USA) | 11+ | 9+ | 2+ | 4+ | 4 |
| 2 | Duchess (USA) | 3+ | 14+ | 1– | 9+ | 3 |
| 3 | Kaissa (USSR) | 2– | 16+ | 5+ | 6+ | 3 |
| 4 | Belle (USA) | 6= | 12+ | 7+ | 1– | 2½ |
| 5 | Chaos (USA) | 15+ | 7= | 3– | 12+ | 2½ |
| 6 | Black Knight (USA) | 4= | 8= | 14+ | 3– | 2 |
| 7 | Dark Horse (Sweden) | 16+ | 5= | 4– | 10= | 2 |
| 8 | Elsa (West Germany) | 12= | 6= | 9– | 15+ | 2 |
| 9 | Master (England) | 13+ | 1– | 8+ | 2– | 2 |
| 10 | Wita (Canada) | 14– | 11+ | 15= | 7= | 2 |
| 11 | BCP (England)) | 1– | 10– | 16+ | 14= | 1½ |
| 12 | Blitz V (USA) | 8= | 4– | 13+ | 5– | 1½ |
| 13 | Chute 1.2 (Canada) | 9– | 15= | 12– | 16+ | 1½ |
| 14 | Ostrich (Canada) | 10+ | 2– | 6– | 11= | 1½ |
| 15 | BS 6676 (Netherlands) | 5– | 13= | 10= | 8– | 1 |
| 16 | Tell (Switzerland) | 7– | 3– | 11– | 13– | 0 |

*Round 1, August 7, 1977*

**Chess 4.6 – BCP** 1 e4 e5 2 ♘f3 ♘c6 3 d4 ed 4 c3 ♕e7 5 cd ♕×e4+ 6 ♗e2 d5 7 ♘c3 ♗b4 8 0-0 ♗×c3 9 ♗d3 ♕e7 10 bc ♘f6 11 ♕b3 ♘e4 12 ♖e1 ♕e6 13 ♘g5 ♕d7 14 f3 f5 15 fe fe 16 ♗×e4 de 17 ♖×e4+ ♘e7 18 ♕f7+ ♔d8 19 ♕×g7 ♕e8 20 ♘f7+ ♔d7 21 ♘×h8 ♔d6 22 ♖×e7 ♕×e7 23 ♗a3+ ♔c6 24 ♕×e7 h5 25 ♕c5+ ♔d7 26 ♖e1 a6 27 ♕d5 mate 1-0

**Duchess – Kaissa** 1 e4 d5 2 ed ♘f6 3 d4 ♘×d5 4 ♘f3 g6 5 ♗e2 ♗g7 6 c4 ♘b6 7 ♘c3 0-0 8 ♗e3 ♗g4 9 c5 ♘d5 10 0-0 e6 11 ♕b3 b6 12 ♘×d5 ed 13 ♗g5 ♕d7 14 h3 ♗f5 15 ♕c3 ♖e8 16 ♖fe1 ♗e4 17 ♘d2 ♕f5 18 ♗e3 ♕e6 19 ♘×e4 de 20 cb cb 21 ♖ec1 ♘d7 22 ♗g4 ♕d5 23 ♕c6 ♘f6 24 ♗e2 ♖ad8 25 ♕a4 ♖e7 26 ♗b5 ♕f5 27 ♖c2 ♘d5 28 ♖ac1 ♗f6 29 ♕b3 a5 30 g4 ♕e6 31 ♖c6 a4 32 ♕×a4 ♖d6 33

♖×d6 ♕×d6 34 ♕a8+ ♖e8 (34 . . . ♔g7 35 ♕f8+!!) 35 ♕×e8+ ♔g7 36 g5 ♗d8 37 ♗c4 ♕e7 38 ♕×e7 ♘×e7 39 ♗f4 ♘f5 40 ♗d5 ♔f8 41 ♖c8 ♔e7 42 ♖c4 ♘g7 43 ♗×e4 ♘e6 44 ♗e3 ♘c7 45 d5 ♘b5 46 ♗f3 ♔d7 47 a4 ♘d6 48 ♖c6 ♘f5 1-0

**Chute — Master** 1 e4 d6 2 ♘c3 g6 3 ♗c4 ♗e6 4 ♗×e6 fe 5 ♘f3 ♘c6 6 ♕e2 ♗g7 7 0-0 ♕d7 8 a3 0-0-0 9 b3 d5 10 ed ed 11 ♖a2 d4 12 ♘d1 d3 13 cd ♕d5 14 ♗b2 e5 15 ♖e1 ♕×b3 16 ♘c3 ♘d4 17 ♕e4 ♘×f3+ 18 ♕×f3 ♘h6 19 ♕e3 ♖d4 20 ♗a1 ♘g4 21 ♕h3 ♔b8 22 ♖e4 h5 23 ♕g3 ♕e6 24 h3 ♘h6 25 f4 ♘f5 26 ♕f3 ♘d6 27 ♖×d4 ed 28 ♘d5 c6 29 ♘b4 ♕e1+ 30 ♕f1 ♖e8 31 ♕×e1 ♖×e1+ 32 ♔f2 ♖h1 33 a4 ♔c7 34 ♘c2 c5 35 ♘e1 ♘f5 36 ♖c2 ♔d6 37 ♖b2 b6 38 ♖b1 h4 39 ♖c1 ♘g3 40 ♔f3 ♖f1+ 41 ♔g4 ♘e2 42 ♖b1 ♗f6 43 g3 hg 44 h4 g2 45 ♗×d4 Adjudicated 0-1

**Chaos — BS66/76** 1 d4 d5 2 c4 e5 3 de d4 4 ♘f3 ♘c6 5 ♘bd2 ♗e6 6 g3 h5 7 ♗g2 d3 8 0-0 ♖b8 9 ed ♕×d3 10 ♕b3 ♕d8 11 ♘e4 ♘d4 12 ♘×d4 ♕×d4 13 ♗e3 ♖d8 14 ♗×d4 ♖×d4 15 ♕×b7 ♗×c4 16 ♖fd1 ♗d3 17 ♕b8+ ♔d7 18 ♕×f8 c5 19 ♖×d3 ♔c7 20 ♕×c5+ ♔b8 21 ♕f8+ ♔c7 22 ♖×d4 ♘e7 23 ♕×e7+ ♔b6 24 ♕b4+ Black lost on time 1-0

**Belle — Black Knight** 1 e4 c5 2 ♘f3 ♘c6 3 d4 cd 4 ♘×d4 ♘f6 5 ♘c3 d6 6 ♗b5 ♗d7 7 0-0 ♘×d4 8 ♕×d4 ♗×b5 9 ♘×b5 e5 10 ♕a4 ♘d7 11 ♖d1 a6 12 ♘d6+ ♗×d6 13 ♖×d6 ♕c7 14 ♖d3 b5 15 ♕b3 0-0 16 ♗g5 ♘c5 17 ♕c3 ♖a7 18 ♗d8 ♖×d8 19 ♕×c5 ♖×d3 20 ♕×c7 ♖×c7 21 cd ♖c2 22 b3 a5 23 f3 f6 24 a4 b4 25 f4 ♖c3 26 fe fe 27 ♖b1 ♖×d3 28 ♔f2 g5 29 ♖b2 h5 30 ♔e2 ♖c3 31 ♖b1 ♖c2+ 32 ♔f1 ♖d2 33 ♖c1 ♖b2 34 ♖c8+ ♔g7 35 ♖c7+ ♔f6 36 ♖c6+ ♔f7 37 ♖c5 ♖b1+ 38 ♔f2 Adjudicated ½-½

**Ostrich — Wita** 1 e4 c5 2 ♘f3 d6 3 ♗c4 ♘f6 4 ♘c3 e6 5 0-0 ♗e7 6 d3 ♗d7 7 ♗e3 ♘c6 8 ♗b5 0-0 9 ♘g5 h6 10 ♗×c6 ♗×c6 11 ♘h3 ♔h8 12 ♕f3 ♘d7 13 ♖fb1 ♕a5 14 ♕g4 ♗f6 15 ♗d2 ♗d4 16 ♕h4 ♕d8 17 ♕×d8 ♖a×d8 18 ♘e2 f5 19 ♘×d4 cd 20 ♘f4 ♖de8 21 ♘e2 e5 22 ef ♖×f5 23 f4 ♘c5 24 ♗b4 g6 25 fe ♖fxe5 26 ♘×d4 ♗d5 27 c4 ♘×d3 28 ♗×d6 ♖e4 29 ♘b5 ♗×c4 30 ♘×a7 ♖a8 31 b3 ♗a6 32 b4 ♖×a7 33 b5 b6 39 ba ♖×a6 35 ♗c7 ♘b4 36 a3 ♘c2 37 ♖a2 ♖e1+ 38 ♖×e1 ♘×e1 39 ♔f2 ♘d3+ 40 ♔f3 ♘b4 41 ♖b2 ♘d5 42 ♗d6 b5 43 ♗c5 ♖f6+ 44 ♔e4 ♘c7 45 ♖f2 ♘e8 46 ♖×f6 ♘×f6+ 47 ♔d3 ♔g7 48 ♗d4 ♔f7 49 ♗×f6 ♔×f6 50 ♔d4 ♔e7 51 ♔c5 ♔d8 52 ♔×b5 ♔c7 53 a4 g5 54 a5 ♔b8 55 ♔b6 ♔c8 56 a6 ♔b8 57 g4 ♔a8 58 a7 h5 59 gh g4 60 h6 g3 61 h7 1-0

**Elsa — Blitz V** 1 d4 d5 2 c4 e6 3 ♘c3 ♘f6 4 ♗g5 ♗e7 5 ♘f3 0-0 6 ♕b3 dc 7 ♕×c4 ♘bd7 8 0-0-0 ♘b6 9 ♕d3 ♗d7 10 ♘e5 ♖c8 11 ♕f3 ♘fd5 12 ♗×e7 ♕×e7 13 ♘×d7 ♕×d7 14 e4 ♘×c3 15 ♕×c3 ♕a4 16 a3 ♖fd8 17 ♕c5 ♖d6 18 ♗b5 ♕a5 19 f4 a6 20 ♗e8 ♕×c5 21 dc ♖×d1+ 22 ♖×d1 ♖×e8 23 cb cb 24 b4 h6 25 ♔b2 ♖e7 26 ♖d6 b5 27 g4 ♖c7 28 ♔b3 ♔f8 29 e5 ♖c4 30 ♖d8+ ♔e7 31 ♖b8 ♖c7 32 h4 g6 33 g5 h5 34 ♖a8 ♔d7 35 ♖b8 ♔c6 36 ♖d8 ♖d7 37 ♖×d7 ♔×d7 38 ♔c3 ♔c6 39 ♔d4 b6 40 ♔e4 a5 41 ♔d4 a4 42 ♔e4 ♔d7 43 ♔d4 ♔c6 44 ♔e4 ♔d7 45 ♔d4 ♔e8 Adjudicated ½-½

**Tell — Dark Horse** 1 e4 ♘f6 2 ♘c3 ♘c6 3 d4 d6 4 ♗b5 ♗d7 5 ♗g5 e5 6 ♗×f6 ♕×f6 7 ♘d5 ♕d8 8 ♘f3 ed 9 ♘×d4 ♗e7 10 ♗×c6 ♗×c6 11 ♘×c6 bc 12 ♘b4 ♕d7 13 ♕d4 ♗f6 14 ♕e3 ♗×b2 15 ♖b1 ♗f6 16 ♘a6 0-0 17 ♖b7 ♖fc8 18 ♖×a7 ♕g4 19 0-0 ♖×a7 20 ♕×a7 ♕×e4 21 ♕b7 ♖f8 22 ♕×c7 ♗e5 23 ♖d1 ♕e2 24 ♖b1 ♕×a6 25 ♖b8 g6 26 ♖×f8+ ♔×f8 27 g3 ♕c4 28 ♕d8+ ♔g7 29 f4 ♗d4+ 30 ♔g2 ♕×c2+ 31 ♔f3 ♕f2+ 32 ♔g4 ♕×h2 33 ♕a5 f5+ 34 ♕×f5 gf+ 35 ♔×f5 ♕×g3 36 ♔e4 c5 37 a4 ♕g2+ 38 ♔d3 ♕f3+ 39 ♔c4 ♕×f4 40 a5 ♕f1+ 41 ♔d5 ♗e5 42 a6 ♕f3+ 43 ♔c4 ♕e2+ 44 ♔d5 ♕×a6 45 ♔e4 ♕e2+ 46 ♔d5 ♕f3+ 47 ♔c4 ♗d4 48 ♔b5 ♕b7+ 49 ♔c4 h6 50 ♔d3 ♕b2 51 ♔c4 ♗e5 52 ♔d5 Adjudicated 0-1

*Round 2, August 7, 1977*

**Kaissa — Tell** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 ♘f6 4 0-0 ♘×e4 5 d4 ♘×d4 6 ♘×d4 ed 7 ♕×d4 ♘c5 8 ♖e1+ ♘e6 9 ♘c3 c5 10 ♕e5 h5 11 ♘d5 ♗e7 12 ♗g5 ♕a5 13 ♗×e7 a6 14 ♕×g7 ♖h6 15 ♕g8+ ♘f8 16 ♕×f8 mate 1-0

**Master — Chess 4.6** 1 ♘f3 d5 2 c4 dc 3 ♘a3 e6 4 ♕a4+ ♗d7 5 ♕×c4 ♘c6 6 e3 ♘f6 7 ♗d3 a6 8 ♘c2 b5 9 ♕f4 ♗d6 10 ♕g5 0-0 11 0-0 e5 12 ♕h4 ♕e7 13 ♗e4 ♘×e4 14 ♕×e4 ♔h8 15 g4 ♘a5 16 g5 ♗c6 17 ♕g4 ♗×f3 18 ♕×f3 ♕×g5+ 19 ♕g2 ♕×g2+ 20 ♔×g2 e4 21 f3 ef+ 22 ♖×f3 f5 23 b3 ♘c6 24 ♗b2 ♘e5 25 ♗×e5 ♗×e5 26 ♖af1 g6 27 h3 c5 28 d4 cd 29 ed ♗d6 30 ♘e3 ♖fe8 31 ♖1f2 ♖ad8 32 ♖d2 ♗b4 33 ♖d3 ♗f8 34 d5 ♖e5 35 a4 ba 36 ba ♗c5 37 d6 ♗×d6 38 ♘c4 ♖e6 39 ♘×d6 ♖e×d6 40 ♖×d6 ♖×d6 41 ♖c3 ♖d4 0-1

**Dark Horse — Chaos** 1 ♘c3 e5 2 ♘f3 ♘c6 3 e4 ♗b4 4 ♘d5 ♘f6 5 ♘×b4 ♘×b4 6 c3 ♘c6 7 ♕e2 0-0 8 d4 d6 9 b3 ♗g4 10 d5 ♘e7 11 g3 c6 12 dc bc 13 ♗g5 ♕a5 14 ♕e3 ♘fd5 15 ed ♘×d5 16 ♕d2 ♗×f3 17 b4 ♕a3 18 ♖g1 ♕×c3 19 ♕×c3 ♘×c3 20 ♗e7 ♖fe8 21 ♗×d6 ♘e4 22 ♗c5 ♘×c5 23 bc ♖ad8 24 ♗e2 ♗×e2 25 ♔×e2 ♖d5 26 ♖ac1 ♖ed8 27 ♖gd1 ♖×d1 28 ♖×d1 ♖e8 29 ♖d6 ♖c8 30 ♔f3 a5 31 ♔e4 f6 32 ♔f3 a4 33 ♖d7 ♖a8 34 ♖d6 ♖a6 35 ♔g2 h6 36 h4 ♔f7 37 h5 ♔g8 38 f3 ♔f8 39 ♖d8+ ♔e7 40 ♖d6 ♔f8 41 ♖d8+ ♔e7 42 ♖d6 ♔e8 43 a3 ♔e7 44 ♔f2 ♔f8 45 ♖d8+ ♔e7 46 ♖d6 ♔f8 47 ♖d8+ ♔f7 48 ♖d7+ ♔g8 49 ♖d8+ ♔h7 50 ♖c8 g5 51 hg+ ♔×g6 52 ♖g8+ ♔f7 53 ♖c8 ♔e6 54 ♖e8+ ♔f7 55 ♖c8 ♔g6 56 ♖g8+ ♔h5 57 g4+ ♔h4 58 ♖g6 h5 59 gh ♔×h5 60 ♖×f6 ♔g5 61 ♖e6 ♔f5 62 ♖h6 ♔g5 63 ♖d6 ♔f4 64 ♖g6 ♔f5 65 ♖g4 ♖a5 66 ♖c4 ♔e6 67 ♔e2 ♔d5 68 ♖e4 ♖×c5 69 ♖×a4 ♖c3 70 ♖a7 c5 Adjudicated ½-½

**Black Knight — Elsa** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 d6 4 d4 ed 5 ♕×d4 ♗d7 6 ♗×c6 ♗×c6 7 ♘c3 ♕f6 8 ♗e3 ♗e7 9 ♘d5 ♕×d4 10 ♘×d4 ♗×d5 11 ed 0-0-0 12 ♘b5 a6 13 ♘a7+ ♔b8 14 0-0-0 ♘f6 15 f3 ♖fe8 16 ♖fe1 ♗f8 17 ♖d2 ♖×e3 18 ♖×e3 ♔×a7 19

♖d1 ♖d7 20 c4 b5 21 cb ab 22 ♖b3 ♔a6 23 ♖a3+ ♔b6 24 ♖b3 g6 25 a4 ♗h6+ 26 ♔b1 ♔a5 27 ♖×b5+ ♔×a4 28 ♖b7 ♖e7 29 ♖d3 ♖e1+ 30 ♔a2 ♖e3 31 ♖d4+ ♔a5 32 ♖×c7 ♗g5 33 h4 ♘e8 34 ♖×f7 ♗f6 35 ♖e4 ♖×e4 36 fe ♗×h4 37 ♖×h7 Adjudicated ½-½

**Duchess – Ostrich** 1 e4 e5 2 ♘f3 ♘f6 3 d4 ♗e7 4 de ♘×e4 5 ♗d3 d5 6 ed ♘×d6 7 0-0 0-0 8 ♖e1 ♗g4 9 c3 ♔h8 10 ♗f4 ♘c6 11 ♘bd2 g5 12 ♗e3 ♘f5 13 ♕e2 ♗×f3 14 ♘×f3 ♘×e3 15 ♕×e3 g4 16 ♗f5 ♖g8 17 ♘e5 ♘×e5 18 ♕×e5+ ♗f6 19 ♕f4 ♗g5 20 ♕e4 ♗d2 21 ♖ad1 ♕g5 22 ♖e2 ♗f4 23 ♖d7 ♖ae8 24 ♕×e8 ♖×e8 25 ♖×e8+ ♔g7 26 ♗e6 a5 27 ♖×f7+ ♔h6 28 ♖ee7 ♕g6 29 ♗f5 ♕g8 30 ♗e4 ♕d8 31 ♖e6+ ♔g5 32 ♖f5+ ♔h4 33 ♖h6+ 1-0

**Blitz V – Belle** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 ♘f6 4 0-0 ♘×e4 5 ♖e1 ♘d6 6 ♘×e5 ♗e7 7 ♗×c6 dc 8 ♘c3 0-0 9 ♕h5 ♖e8 10 d4 ♗e6 11 ♗f4 ♘b5 12 ♘×b5 cb 13 ♘f3 ♕d7 14 ♖e3 a5 15 h3 a4 16 ♗e5 ♕c6 17 c3 h6 18 a3 ♖ad8 19 ♘d2 ♗g5 20 ♖e2 g6 21 ♕f3 ♕×f3 22 ♘×f3 ♗c4 23 ♖ee1 c6 24 ♘×g5 hg 25 ♗f6 ♖c8 26 ♗×g5 ♖a8 27 ♖e7 ♖×e7 28 ♗×e7 ♖e8 29 ♖e1 ♗b3 30 ♖e5 ♗a2 31 f4 ♗e6 32 ♗f6 ♖f8 33 h4 ♗g4 34 ♖e7 ♗c8 35 g3 ♔h7 36 ♔g2 ♔g8 37 ♔f3 b6 38 ♔e4 ♗f5+ 39 ♔e3 ♖c8 40 ♔f3 ♖b8 41 ♖e5 ♗c2 42 d5 cd 43 ♖×d5 b4 44 ab ♖e8 45 ♖d6 ♗e4+ 46 ♔e3 b5 47 ♖d4 ♗c6+ 48 ♗e5 ♗g2 49 ♔f2 ♗c6 50 ♔e2 ♗g2 51 ♔e3 f6 52 ♖d6 fe 53 ♖×g6+ ♔f7 54 f5 ♗h3 55 g4 ♖g8 56 ♖×g8 ♔×g8 57 ♔f3 e4+ 58 ♔f4 e3 59 ♔×e3 ♗×g4 60 ♔e4 ♔g7 61 ♔f4 ♗e2 62 ♔g5 ♗d1 63 f6+ ♔h7 64 c4 bc 65 b5 c3 66 bc a3 67 f7 ♔g7 68 f8=♕+ ♔×f8 0-1

**BCP – WITA** 1 e4 c5 2 ♘c3 d6 3 d4 ♘f6 4 ♗e3 e6 5 ♘f3 ♗e7 6 e5 de 7 ♗b5+ ♗d7 8 de ♘g4 9 0-0 ♘c6 10 ♗×c6 ♗×c6 11 ♕×d8+ ♖×d8 12 ♗f4 0-0 13 h3 ♗×f3 14 gf ♘h6 15 ♗×h6 gh 16 ♘e4 f5 17 ef ♗×f6 18 ♘×c5 ♖d2 19 ♘×e6 ♖e8 20 ♘c5 ♖×c2 21 ♘×b7 ♖c7 22 ♘d6 ♖f8 23 ♖ab1 ♗e5 24 ♖fd1 ♖g7+ 25 ♔f1 ♖×f3 26 ♘c4 ♗f6 27 ♖d6 ♗h4 28 ♔e2 ♖×f2+ 29 ♔e3 ♖c2 30 ♔d4 ♖c7 31 ♖g1+ ♔f7 32 ♘e3 ♖d2+ 33 ♔e4 ♖×d6 34 ♘f5 ♖c4+ 35 ♔e5 ♖e6+ 36 ♔d5 ♖b4 37 ♖g7+ ♔f6 38 ♖g4 ♖×g4 39 hg ♖e5+ 40 ♔c4 ♖×f5 41 gf h5 42 a4 ♔×f5 43 a5 h6 44 a6 ♗f6 45 b4 h4 46 b5 h3 47 b6 h2 48 ba h1=♕ 49 ♔b3 h5 50 ♔a4 ♕c6+ 51 ♔a3 ♕a8 52 ♔b4 ♗d4 53 ♔c4 ♔e4 54 ♔b5 h4 55 ♔c4 h3 55 ♔b3 h2 57 ♔c4 ♕d5+ 58 ♔b4 h1=♕ 59 ♔a3 ♕a1+ 60 ♔b4 ♕aa5 mate 0-1

**BS 66/76 – Chute** 1 d4 d5 2 e4 de 3 ♘c3 ♘f6 4 f3 ef 5 ♕×f3 ♕×d4 6 ♗e3 ♕b4 7 0-0-0 ♗g4 8 ♘b5 ♘a6 9 ♕×b7 ♗×d1 10 ♘d6+ ♕×d6 11 ♗b5+ c6 12 ♗×c6+ ♘d7 13 ♕×a8+ ♘ab8 14 ♗b5 a6 15 ♗×d7+ ♔×d7 16 ♕b7+ ♔e6 17 ♕e4+ ♔d7 18 ♗f4 e5 19 ♗×e5 ♕h6+ 20 ♔×d1 ♗d6 21 ♘f3 ♖c8 22 ♕b7+ ♗c7 23 ♕d5+ ♔e8 24 ♗×c7 ♖×c7 25 ♖e1+ ♖e7 26 ♖×e7+ ♔×e7 27 ♕e5+ ♕e6 28 ♕×b8 ♕d5+ 29 ♔c1 ♕×a2 30 ♕b7+ ♔f6 31 ♕c6+ ♕e6 32 ♕c3+ ♔g6 33 ♘h4+ ♔h5 34 ♕f3+ ♔×h4 35 ♕f4+ ♔h5 36 g4+ ♕×g4 37 ♕×f7+ ♔h6 38 ♕f1 ♕g5+ 39 ♔g1 ♕a5 40 ♕h3+ ♕h5 41 ♕e6+ ♕g6 42 ♕e3+ ♕g5 43 ♕e6+ ♕f6 44 ♕h3+ ♔g6 45 ♕g4+ ♔f7 46 ♕c4+ ♔g6 47 ♕e4+ ♔h6 48 b4 ♕f1+ 49 ♔a2 ♕f7+ 50 ♔a3 ♕f6 51 ♕e3+ ♔g6 52 ♕d3+ ♔h6 53 ♕h3+ ♔g6 54 ♔b3 ♕f7+ 55 ♔a3 ♕c4 56 ♕g3+ ♔f7 57 ♕f3+ ♔g6 58 ♕g3+ ♔f7 ½-½

*Round 3, August 8, 1977*

**Chaos – Kaissa** 1 d4 ♘f6 2 c4 c5 3 d5 d6 4 ♘c3 g6 5 e4 ♗g7 6 ♗e2 0-0 7 ♗g5 h6 8 ♗e3 ♕b6 9 ♕d2 ♘g4 10 ♗×g4 ♗×g4 11 f3 ♗d7 12 ♗×h6 ♗×h6 13 ♕×h6 ♕×b2 14 ♖b1 ♕×c3+ 15 ♕d2 ♕×d2+ 16 ♔×d2 ♗c8 17 ♘e2 ♔g7 18 h4 ♖h8 19 ♘f4 ♘d7 20 h5 g5 21 ♘d3 ♖b8 22 f4 ♘f6 23 e5 ♘e4+ 24 ♔e3 ♘g3 25 ♖h2 gf+ 26 ♔×f4 ♘×h5+ 27 ♔f3 ♗f5 28 ♖d1 ♗×d3 29 ♖×d3 de 30 ♖h3 ♘f6 31 ♖g3+ ♔f8 32 ♖e3 b5 33 ♖×e5 bc 34 ♖f5 c3 35 d6 ed 36 ♔e3 ♖e8+ 37 ♔d3 c4+ 38 ♔×c4 ♖c8+ 39 ♔b3 c2 40 ♖×f6 c1=♕ 41 ♖gf3 ♖b8+ 42 ♔a4 ♕c4+ 43 ♔a5 ♕b5 mate

**Chess 4.6 – Duchess** 1 e4 e5 2 ♘f3 ♘f6 3 d4 ed 4 e5 ♘e4 5 ♕×d4 d5 6 ed ♘×d6 7 ♗d3 ♘c6 8 ♕f4 g6 9 0-0 ♗g7 10 ♗d2 ♕f6 11 ♕×f6 ♗×f6 12 ♘c3 0-0 13 ♘d5 ♗×b2 14 ♖ab1 ♗g7 15 ♘×c7 ♖b8 16 ♗f4 ♖d8 17 ♘b5 ♗f8 18 ♗g5 ♖d7 19 ♖fe1 b6 20 ♘c3 f5 21 ♘d5 ♔h8 22 ♗f4 ♗b7 23 ♘g5 ♖c8 24 ♘f6 ♖dc7 25 ♘e6 ♖f7 26 ♘×f8 ♖f×f8 27 ♘×h7 ♔×h7 28 ♗×d6 ♖fd8 29 ♗f4 ♖d4 30 ♗g5 ♖g4 31 f4 ♘a5 32 ♖e7+ ♔g8 33 g3 ♔f8 34 ♖be1 ♗f3 35 ♗h6+ ♔g8 36 ♖1e3 ♗e4 37 ♗×e4 fe 38 h3 ♖×f4 39 ♗×f4 ♖×c2 40 ♖3×e4 ♖b2 41 ♖e8+ ♔f7 42 ♖4e7+ ♔f6 43 ♗e5+ ♔g5 44 ♗×b2 ♘c4 45 ♗c1+ ♔f6 46 ♖e6+ ♔g7 47 ♗g5 ♘d6 48 ♖×d6 ♔f7 49 ♖e7+ ♔g8 50 ♖d8 mate 1-0

**Elsa – Master** 1 e4 d6 2 d4 ♘f6 3 ♘c3 g6 4 ♘f3 ♗g7 5 ♗e2 0-0 6 0-0 b6 7 ♗g5 ♗b7 8 e5 ♘e4 9 ♘×e4 ♗×e4 10 ♗d3 ♗b7 11 ♕e2 de 12 de f6 13 ♗c4+ ♔h8 14 ♖ad1 ♕c8 15 ♗h4 ♕g4 16 h3 ♕h5 17 g4 ♕h6 18 ♖d3 fe 19 ♗×e7 ♕×h3 20 ♗×f8 ♕×g4+ 21 ♔h1 ♕×c4 22 ♗×g7+ ♔×g7 23 ♕×e5+ ♔h6 24 ♕e3+ ♔g7 25 ♕e7+ ♕f7 26 ♕e5+ ♕f6 27 ♕×c7+ ♕f7 28 ♕e5+ ♕f6 29 ♕×f6+ ♔×f6 30 ♖fd1 h5 31 c4 g5 32 a4 g4 33 ♖d6+ ♔e7 34 ♔g1 ♗×f3 35 ♖1d2 h4 36 a5 ♘c6 37 ab ab 38 ♖d7+ ♔f8 39 ♖d1 ♗×d1 40 ♖×d1 ♘e5 41 b3 ♘f3+ 42 ♔g2 ♔e7 43 ♖b1 ♖a2 44 ♖c1 ♖b2 45 ♖c3 ♔f6 46 ♖c1 ♖×b3 47 c5 bc 48 ♖×c5 ♖b1 49 ♖f5+ ♔e6 50 ♖×f3 gf+ 51 ♔×f3 ♖g1 52 ♔e4 h3 53 f4 ♖f1 54 ♔d4 h2 55 ♔d3 h1=♕56 f5+ ♔×f5 57 ♔d4 ♖c1 58 ♔d3 ♕e1 59 ♔d4 ♕e4 mate 0-1

**Ostrich – Black Knight** 1 e4 c5 2 ♘f3 ♘c6 3 ♗c4 ♘f6 4 ♘c3 ♘×e4 5 ♗×f7+ ♔×f7 6 ♘×e4 d5 7 ♘×c5 e5 8 ♘b3 ♗f5 9 0-0 ♗e7 10 d3 d4 11 ♖e1 ♕c7 12 ♕e2 ♗d6 13 ♗d2 ♕b6 14 c3 ♖he8 15 cd ed 0-1 Computer break-down.

**Chute – Blitz V** 1 e4 e5 2 ♘f3 ♘f6 3 ♘×e5 d6 4 ♘f3 ♘×e4 5 ♕e2 ♕e7 6 d3 ♘f6 7 ♕×e7+ ♗×e7 8 ♘a3 ♗g4 9 h3 ♗×f3 10 gf 0-0 11 ♘b5 ♘a6 12 ♗e3 c6 13 ♘×a7 ♘c5 14 ♗×c5 dc 15 ♘×c6 bc 16 a4 ♖ae8 17 0-0-0 ♗d6 18 b3 ♖e5 19 ♔b1 ♖fe8 20 ♔b2 h6 21 ♖g1 ♖e1 22 ♖×e1 ♖×e1 23 c3 ♗h2 24 ♖h1 ♗e5 25 ♔c2 ♘d5 26 c4 ♘b4+ 27 ♔d2 ♖b1 28 ♔e3

♖×b3 29 ♔e4 ♗d4 30 ♖h2 ♖a3 31 ♖g2 ♖a1 32 ♗e2 ♖e1 33 f4 ♖×e2+ 34 ♔f3 ♖d2 35 a5 ♘×d3 36 ♖g3 ♖×f2+ 37 ♔e4 ♘b2 38 ♖b3 ♘×c4 39 ♖b8+ ♔h7 40 a6 ♖×f4+ 41 ♔×f4 ♗e5+ 42 ♔f5 ♗×b8 43 ♔e4 ♘d6+ 44 ♔e3 f5 45 h4 g5 46 hg hg 47 ♔e2 g4 48 ♔e3 c4 49 ♔d4 g3 50 ♔e3 c3 51 ♔d3 g2 52 a7 ♗×a7 53 ♔×c3 g1=♕ 54 ♔b3 ♕b1+ 55 ♔a4 ♕b2 56 ♔a5 ♕a3 mate 0-1

**Tell – BCP** 1 e4 e5 2 ♘f3 ♘c6 3 ♗c4 ♘f6 4 ♘c3 ♗e7 5 0-0 0-0 6 ♗d5 d6 7 ♗×c6 bc 8 d4 ♗g4 9 de ♗×f3 10 ♕×f3 de 11 ♖d1 ♗d6 12 ♗e3 a5 13 ♗c5 ♕b8 14 ♗×d6 cd 15 ♖ab1 a4 16 b4 ♖c8 17 ♕f5 ♖d8 18 b5 cb 19 ♖×b5 ♕c8 20 ♕×c8 ♖d×c8 21 ♖d3 ♖a6 22 ♖b4 a3 23 ♖b7 h6 24 f3 ♖ac6 25 ♘d5 ♘×d5 26 ed ♖a6 27 ♖d2 ♖a4 28 g4 e4 29 fe ♖×e4 30 h3 ♖e1+ 31 ♔g2 ♖a1 32 ♖f2 ♖×a2 33 ♖b×f7 ♖a×c2 34 ♖7f3 ♖×f2+ 35 ♖×f2 g5 36 ♖f6 a2 37 ♖g6+ ♔f7 38 ♖g7+ ♔×g7 39 h4 gh 40 ♔h3 a1=♕ 41 ♔g2 ♕d4 42 ♔h3 ♕×d5 43 ♔h2 ♖c2+ 44 ♔h3 ♕h1 mate 0-1

**Wita – BS 66/76** 1 d4 d5 2 c4 e5 3 de d4 4 e4 f5 5 ♗d3 ♘c6 6 ♘f3 ♗b4+ 7 ♔e2 fe 8 ♗×e4 ♗g4 9 ♕a4 d3+ 10 ♔f1 ♕d7 11 ♗×c6 ♕×c6 12 ♕×b4 ♗×f3 13 gf ♕×f3 14 ♖g1 d2 15 ♗×d2 ♕d1+ 16 ♗e1 ♕d3+ 17 ♔g2 ♕g6+ 18 ♔h1 ♕e4+ 19 ♖g2 a5 20 ♕c3 ♕f5 21 ♖×g7 0-0-0 22 ♕×a5 ♕e4+ 23 f3 ♕×f3+ 24 ♔g1 ♕e3+ 25 ♔h1 ♕e4+ 26 ♔g1 ♕e3+ 27 ♔h1 ♕e4+ 28 ♔g1 ♕e3+ ½-½

**Belle – Dark Horse** 1 e4 ♘f6 2 e5 ♘d5 3 d4 e6 4 ♗d3 ♘c6 5 ♘f3 ♗e7 6 0-0 0-0 7 c3 d6 8 ♖e1 de 9 de b6 10 ♕c2 g6 11 ♗h6 ♘db4 12 cb ♘×b4 13 ♕b3 ♘×d3 14 ♗×f8 ♗×f8 15 ♖d1 ♗a6 16 ♘e1 ♘×e1 17 ♖×d8 ♖×d8 18 ♕a4 b5 19 ♕×a6 ♘c2 20 ♘c3 ♘×a1 21 ♕×b5 ♖d2 22 ♕f1 ♘c2 23 ♕c1 ♗h6 24 ♘e4 ♖d5 25 ♘f6+ ♔g7 26 ♕×c2 ♖×e5 27 ♘e8+ ♔f8 28 ♕c3 ♖d5 29 ♕h8+ ♔e7 30 g3 c6 31 ♘f6 ♖d1+ 32 ♔g2 ♗g5 33 ♘×h7 ♗h6 34 ♕b8 ♗g7 35 ♕×a7+ ♖d7 36 ♕c5+ ♖d6 37 ♕g5+ ♔d7 38 ♘f6+ ♗×f6 39 ♕×f6 ♔e8 40 a4 ♖d5 41 ♕c3 c5 42 a5 ♖f5 43 a6 f6 44 a7 ♖d5 45 a8=♕+ ♔e7 46 ♕b7+ ♖d7 47 ♕×c5+ ♔e8 48 ♕bc8+ ♖d8 49 ♕×e6 mate 1-0

*Round 4, August 9, 1977*

**Master – Duchess** 1 ♘f3 ♘f6 2 c4 d5 3 cd ♘×d5 4 g3 c6 5 ♗g2 e6 6 0-0 ♗e7 7 d4 0-0 8 e4 ♘f6 9 ♘c3 b6 10 ♘e5 ♗a6 11 ♖e1 ♕c8 12 ♗e3 ♘bd7 13 ♘×d7 ♕×d7 14 e5 ♘d5 15 ♕a4 ♗b5 16 ♘×b5 cb 17 ♕b3 ♖fd8 18 ♗d2 ♖ac8 19 ♖ac1 a5 20 ♖×c8 ♖×c8 21 ♖c1 ♖×c1+ 22 ♗×c1 a4 23 ♕d1 ♘b4 24 ♗e3 ♘×a2 25 d5 ed 26 ♕×d5 ♕×d5 27 ♗×d5 ♘b4 28 ♗e4 ♗c5 29 ♔g2 ♗×e3 30 fe h6 31 ♔f3 ♔f8 32 h4 ♔e7 33 h5 ♔e6 34 ♔f4 ♘d5+ 35 ♗×d5+ ♔×d5 36 g4 b4 37 ♔f5 a3 38 e4+ ♔c5 39 ba ba 40 e6 fe+ 41 ♔e5 a2 42 ♔×e6 a1=♕ 43 g5 hg 44 e5 ♕d4 45 ♔f5 ♕f4+ 46 ♔e6 g4 47 ♔e7 g3 0-1

**B5 66/76 – Elsa** 1 d4 ♘f6 2 c4 g6 3 ♘f3 ♗g7 4 ♘c3 0-0 5 ♗f4 ♘h5 6 e3 ♘×f4 7 ef c5 8 dc ♗×c3+ 9 bc ♕c7 10 ♕d4 ♘a6 11 c6 dc 12 h3 ♖d8 13 ♕e3 ♕d6 14 ♕e5 ♗f5 15 g4 ♗e6 16 g5 ♘c5 17 ♖b1 ♕d3 18 ♗×d3 ♘×d3+ 19 ♔e2 ♘×e5 20 ♘×e5 f6 21 ♘×g6 ♗×c4+ 22 ♔e3 hg 23 gf ef 24 ♖×b7 ♖d3+ 25 ♔e2 ♖e8+ 26 ♖e7 ♖×e7+ 27 ♔f1 ♖g3 mate

**BCP – Ostrich** 1 e4 e5 2 ♗c4 ♘f6 3 d4 d6 4 ♘f3 ♗g4 5 de ♗×f3 6 gf de 7 ♗×f7+ ♔e7 8 ♕×d8+ ♔×d8 9 0-0 ♘bd7 10 ♗g5 h6 11 ♗×f6+ ♘×f6 12 ♘c3 ♗b4 13 ♘d5 ♘×d5 14 ♗×d5 c6 15 ♗e6 ♔e8 16 a3 ♗e7 17 ♖fd1 ♖d8 18 ♖×d8+ ♗×d8 19 ♖d1 ♖f8 20 ♔g2 ♗b6 21 ♗f5 ♖f6 22 ♗c8 ♖f7 23 ♗f5 ♔f8 24 ♖d7 ♖×d7 25 ♗×d7 ♗d4 26 c3 ♗c5 27 b4 ♗d6 28 c4 c5 29 b5 ♔e7 30 ♗f5 ♔f6 31 h4 g6 32 ♗d7 h5 33 a4 g5 34 hg+ ♔×g5 35 a5 h4 36 b6 ab 37 ab ♗e7 38 ♗e6 ♗d8 39 ♗d5 ♔f4 40 ♗×b7 h3+ 41 ♔×h3 ♔×f3 42 ♗d5 ♗×b6 43 ♔h2 ♔×f2 44 ♔h3 ♔f3 45 ♔h2 ♗d8 46 ♔h3 ♗g5 47 ♔h2 ♗f4+ 48 ♔h3 ♗g3 49 ♗c6 ♗e1 50 ♗d5 ½-½

**Blitz V – Chaos** 1 e4 c5 2 ♘f3 ♘c6 3 d4 cd 4 ♘×d4 ♘f6 5 ♘c3 e6 6 ♗e2 ♗b4 7 0-0 ♗×c3 8 bc ♘×e4 9 ♘×c6 dc 10 ♕×d8+ ♔×d8 11 ♗d3 ♘c5 12 ♗e3 ♘×d3 13 ♖fd1 e5 14 cd ♔c7 15 ♖ab1 ♗f5 16 ♖b3 b6 17 ♖d2 f6 18 h3 ♖ad8 19 d4 ♔c8 20 de ♖×d2 21 ♗×d2 fe 22 ♗g5 h6 23 ♗h4 b5 24 a3 a5 25 f3 ♔d7 26 ♔h2 ♔e6 27 ♖b2 ♖f8 28 ♖e2 ♗d3 29 ♖d2 ♗f1 30 ♔g1 ♗c4 31 ♗d8 a4 32 ♗c7 ♔f5 33 g4+ ♔g5 34 ♗×e5 ♖×f3 35 ♖d6 ♗d5 36 ♔h2 ♔h4 37 ♗×g7 ♖×h3+ 38 ♔g1 h5 39 ♗e5 ♔×g4 40 ♗d4 ♖g3+ 41 ♔f2 h4 42 ♖g6+ ♔f4 43 ♖h6 ♖g2+ 44 ♔f1 ♔g3 45 ♖g6+ ♔h3 46 ♖d6 ♗c4+ 47 ♔e1 ♖e2+ 48 ♔d1 ♖e6 49 ♖d7 ♔h2 50 ♔d2 ♖e2+ 51 ♔d1 h3 52 ♖d6 ♖e6 53 ♖d7 ♗b3+ 54 ♔d2 ♔h1 55 ♗e3 ♗c2 56 ♔×c2 ♖×e3 57 ♖d6 ♔g2 58 ♖×c6 h2 59 ♖g6+ ♖g3 60 ♖h6 h1=♕ 61 ♖×h1 ♔×h1 62 ♔d2 ♔g1 63 ♔c2 ♖g4 64 ♔d2 ♔f1 65 ♔d1 ♖c4 66 ♔d2 ♔f2 67 ♔d3 ♔e1 68 ♔c2 ♔e2 69 ♔b2 ♔d2 70 ♔b1 ♔×c3 71 ♔c1 ♖c6 72 ♔d1 ♖e6 73 ♔c1 ♖e1 mate 0-1

**Tell – Chute** 1 e4 c5 2 ♗c4 d6 3 ♕h5 e6 4 ♗b5+ ♘d7 5 ♘c3 ♘gf6 6 ♕h4 a6 7 ♗e2 d5 8 ed ed 9 ♘f3 b5 10 ♘×d5 ♘×d5 11 ♕e4+ ♗e7 12 ♕×d5 ♘b6 13 ♕c6+ ♗d7 14 ♕e4 0-0 15 0-0 f5 16 ♕e5 ♖f6 17 ♗d3 ♗d6 18 ♕c3 ♘d5 19 ♕b3 c4 20 ♗×c4 bc 21 ♕×c4 ♗e6 22 ♘g5 ♖h6 23 h4 ♘e3 24 ♘×e6 ♗h2+ 25 ♔h1 ♘×c4 26 ♘×d8 ♖×h4 27 ♘e6 ♗f4+ 28 ♔g1 ♗h2+ 29 ♔h1 ♗e5+ 30 ♔g1 ♖e8 31 ♘c5 ♗h2+ 32 ♔h1 ♗d6+ 33 ♔g1 ♗×c5 34 d3 ♘e5 35 ♗g5 ♖g4 36 ♗c1 ♘f3+ 37 ♔h1 ♖h4 mate

**Belle – Chess 4.6** 1 e4 ♘c6 2 ♘f3 e6 3 d4 d5 4 ♘c3 ♗b4 5 e5 ♘ge7 6 a3 ♗×c3+ 7 bc ♘a5 8 ♗b5+ ♗d7 9 ♗d3 ♖c8 10 ♘g5 h6 11 ♘f3 c5 12 dc ♖×c5 13 ♗e3 ♖×c3 14 ♗×a7 ♘c4 15 0-0 ♖×a3 16 ♖×a3 ♘×a3 17 ♗c5 ♕a5 18 ♗d6 ♘c4 19 ♕a1 ♘c6 20 ♕×a5 ♘6×a5 21 ♖a1 ♗c8 22 c3 ♘c6 23 ♖a4 ♘×d6 24 ed ♔d7 25 ♖g4 g5 26 ♗c2 ♔×d6 27 ♖a4 b5 28 ♖a1 b4 29 cb ♘×b4 30 ♗b1 ♗d7 31 ♔h1 f5 32 ♘d4 ♖c8 33 ♘e2 ♗b5 34 ♘g1 ♖c1 35 ♖a5 ♖×b1 36 f3 ♗f1 37 h4 ♖b2 38 hg ♗×g2+ 39 ♔h2 hg 40 ♖a4 ♗×f3+ 41 ♔g3 ♗h5 42 ♔h3 f4 43 ♖a8 ♗g6 44 ♔g4 ♖g2+ 45 ♔h3 ♖×g1 46 ♔h2 ♖g4 47 ♖d8+ ♔e5 48 ♖g8 ♗e4 49 ♖g7 ♗f3 50 ♖h7 ♘d3 51 ♖h3 ♖g2+ 52 ♔h1 ♘f2 mate 0-1

**Wita – Dark Horse** 1 c4 ♘c6 2 g3 e5 3 ♗g2 ♕f6

4 ♘c3 ♗c5 5 ♘f3 d6 6 d3 ♕d8 7 ♗e3 ♗xe3 8 fe ♘f6 9 0-0 ♗e6 10 ♘g5 ♔d7 11 ♕a4 a5 12 ♕b5 ♖b8 13 ♘a4 g6 14 a3 ♖e8 15 ♖ac1 h6 16 ♘xe6 fe 17 ♘c3 ♕e7 18 h3 g5 19 b3 g4 20 d4 gh 21 ♗xh3 ♕g7 22 ♔h1 ♕xg3 23 ♖f3 ♕g6 24 ♖g1 ♕c2 25 ♖xf6 ♕xc3 26 d5 ♕xe3 27 dc+ ♔d8 28 ♗g4 ♕g5 29 ♖f2 b6 30 ♗h3 ♕e3 31 ♖f3 ♕xe2 32 ♖f6 ♕e4+ 33 ♔h2 ♕c2+ 34 ♖g2 ♕e4 35 ♖xe6 ♕f4+ 36 ♔h1 ♕f1+ 37 ♔h2 ♖xe6 38 ♖g8+ ♔e7 39 ♗xe6 ♕f2+ 40 ♔h3 ♕f3+ 41 ♔h2 ♕h5+ 42 ♔g3 ♖xg8+ 43 ♗xg8 ♕g5+ 44 ♔f2 ♕xg8 45 b4 ab 46 ab ♕f8+ 47 ♔g2 ♔d8 48 ♕d5 h5 49 b5 h4 50 ♔h3 ♕h6 51 ♕g8+ ♔e7 52 ♕b8 ♕e3+ ½-½

**Black Knight – Kaissa** 1 e4 d5 2 ed ♘f6 3 d4 ♘xd5 4 c4 ♘b6 5 ♘f3 g6 6 ♘c3 ♗g7 7 h3 0-0 8 ♗e3 ♘c6 9 ♕d2 e5 10 d5 ♘e7 11 g4 f5 12 0-0-0 e4 13 ♘h2 ♗d7 14 ♗e2 ♗a4 15 ♘xa4 ♘xa4 16 ♗d4 ♗xd4 17 ♕xd4 ♕d6 18 ♖d2 c5 19 ♕e3 ♖ae8 20 ♕g5 fg 21 hg ♖xf2 22 ♗d1 ♖xd2 23 ♗xa4 ♖xh2 24 ♖xh2 ♕xh2 25 ♗xe8 ♕a1+ 26 ♔c2 ♕g2+ 27 ♕d2 ♕xg4 28 ♕f2 ♘f5 29 ♕xc5 ♕g2+ 30 ♔b3 ♕g3+ 31 ♔a4 a6 32 ♕c8 ♘d6 33 ♕e6+ ♔f8 34 ♗d7 ♕f4 35 c5 e3+ 36 ♔a3 ♘c4+ 37 ♔b3 ♘a5+ 38 ♔a3 ♘c4+ 39 ♔b3 ♘e5 40 ♕e8+ ♔g7 41 ♕e7+ ♔h6 42 ♕e6 ♕c4+ 43 ♔a3 ♕xc5+ 44 ♔b3 ♕c4+ 45 ♔a3 ♕d3+ 46 b3 ♘c4+ 47 ♔b4 e2 48 ♔c5 ♘d6 49 ♔b6 ♘e4 50 ♔xb7 ♘c5+ 51 ♔b6 ♘xe6 52 de e1=♕ 53 ♔b7 ♕a5 54 b4 ♕dd5+ 55 ♗c6 ♕xb4+ 56 ♔c7 ♕dd6+ 57 ♔c8 ♕bb8 mate 0-1

*Exhibition Game (10th August 1977)*

**Kaissa – Chess 4.5** 1 e4 ♘c6 2 ♘f3 e6 3 d4 d5 4 ♗d3 de 5 ♗xe4 ♗d7 6 0-0 ♘f6 7 ♖e1 ♘xe4 8 ♖xe4 ♗e7 9 c4 f5 10 ♖e1 0-0 11 ♘c3 f4 12 ♕d3 ♕e8 13 g3 fg 14 hg ♕f7 15 ♗f4 g5 16 d5 ed 17 ♘xd5 gf 18 ♘xe7+ ♘xe7 19 ♕xd7 ♘g6 20 ♕xf7+ ♖xf7 21 g4 ♖d7 22 ♖ad1 ♖ad8 23 ♖xd7 ♖xd7 24 ♔g2 ♔g7 25 ♘g5 ♖d2 26 ♖b1 ♖c2 27 b3 ♘e5 28 ♖h1 ♖xa2 29 ♖h4 ♘d3 30 ♘h3 ♖b2 31 g5 ♔g8 32 ♘xf4 ♖xf2+ 33 ♔g3 ♖xf4 34 ♖xf4 ♘xf4 35 ♔xf4 ♔f7 36 b4 ♔e6 37 ♔e4 c6 38 ♔f4 ♔d6 39 ♔e4 c5 40 bc+ ♔xc5 41 ♔d3 a5 42 ♔c3 a4 43 ♔d3 ♔b4 44 ♔c2 ♔xc4 0-1

---

**Third World Computer Championship Linz, Austria, September 25th-29th 1980**

| | | rate | perf | 1 | 2 | 3 | 4 | total |
|---|---|---|---|---|---|---|---|---|
| 1 | Belle (USA) | 2150 | 2189 | 8+ | 6= | 4+ | 3+ | 3½ |
| 2 | Chaos (USA) | 1800 | 2025 | 17+ | 4= | 6+ | 7+ | 3½ |
| 3 | Duchess (USA) | 1900 | 1945 | 18+ | 12+ | 7+ | 1– | 3 |
| 4 | L'Excentrique (Canada) | 0 | 2030 | 5+ | 2= | 1– | 12+ | 2½ |
| 5 | Chess 4.9 (USA) | 2050 | 1903 | 4– | 9= | 11+ | 10+ | 2½ |
| 6 | Nuchess (USA) | 0 | 1759 | 16+ | 1= | 2– | 9= | 2 |
| 7 | Kaissa (USSR) | 0 | 1631 | 14+ | 13+ | 3– | 2– | 2 |
| 8 | BCP (England) | 0 | 1584 | 1– | 10= | 14= | 15+ | 2 |
| 9 | BeBe (USA) | 0 | 1781 | 10= | 5= | 12= | 6= | 2 |
| 10 | Schach 2.3 (West Germany) | 0 | 1706 | 9= | 8= | 13+ | 5– | 2 |
| 11 | Awit (Canada) | 1400 | 1249 | 13– | 17+ | 5– | 18+ | 2 |
| 12 | Master (England) | 0 | 1643 | 15+ | 3– | 9= | 4– | 1½ |
| 13 | Ostrich (Canada) | 1450 | 1431 | 11+ | 7– | 10– | 14= | 1½ |
| 14 | Mychess (USA) | 0 | 1380 | 7– | 15= | 8= | 13= | 1½ |
| 15 | Parwell (West Germany) | 0 | 1256 | 12– | 14= | 16+ | 8– | 1½ |
| 16 | Advance 1.0 (England) | 0 | 754 | 6– | 18= | 15– | 17+ | 1½ |
| 17 | Dark Horse (Sweden) | 0 | 636 | 2– | 11– | 18+ | 16– | 1 |
| 18 | CSC (USA) | 0 | 505 | 3– | 16= | 17– | 11– | ½ |

*Round 1, September 25, 1980*

**L'Excentrique – Chess 4.9** 1 e4 e5 2 ♘f3 ♘f6 3 d4 ♘xe4 4 ♗d3 d5 5 ♘xe5 ♗d6 6 c4 ♗xe5 7 dxe5 ♘c6 8 ♗c2 ♘xe5 9 ♕xd5 ♕xd5 10 cxd5 ♗f5 11 0-0 c6 12 f4 ♘g4 13 h3 ♘gf6 14 g4 ♘d2 15 ♖e1+ ♔f8 16 ♗d1 ♗xb1 17 ♗xd2 ♗d3 18 ♖e3 ♗c4 19 ♗b4+ ♔g8 20 d6 ♘d5 21 ♖e4 ♗d3 22 ♖d4 ♘xb4 23 ♖xb4 b6 24 ♖d4 ♗b5 25 a4 c5 26 ♖d2 ♗c6 27 a5 ♖d8 28 a6 h5 29 gxh5 ♖h6 30 ♗g4 ♖hxd6 31 ♖xd6 ♖xd6 36 ♔f2 ♖d2+ 33 ♔e3 ♖xb2 34 ♖d1 ♖b3+ 35 ♔f2 ♗e4 36 ♖d8+ ♔h7 37 ♖f8 f6 38 ♖f7 ♖a3 39 ♖xa7 f5 40 ♗e2 ♖xh3 41 ♖d7 ♖h2+ 42 ♔e3 ♖h1 43 ♖d8 ♖h3+ 44 ♔d2 ♖b3 45 ♗c4 ♖b4 46 ♗f7 ♖d4+ 47 ♖xd4 cxd4 48 a7 ♔h6 49 ♗g6 ♗g2 50 ♔d3 b5 51 ♗xf5 g5 52 ♗e4 ♗f1+ 53 ♔xd4 ♔xh5 54 f5 g4 55 a8(♕) ♔h4 56 ♔e3 ♔h3 57 ♔f2 1-0

**Schach 2.3 – Belle** 1 e4 c5 2 ♘f3 d6 3 d4 cxd4 4 ♘xd4 ♘f6 5 ♘c3 g6 6 ♗e2 ♗g7 7 0-0 0-0 8 ♗e3 ♗d7 9 ♕d3 ♘a6 10 ♘b3 ♘b4 11 ♕d2 ♘g4 12 ♗g5

♖c8 13 h3 ♘f6 14 ♗e3 a5 15 ♘d4 e5 16 ♘db5 d5 17 e×d5 ♗×b5 18 ♗×b5 ♘b×d5 19 ♘×d5 ♕×d5 20 ♕e2 e4 21 c3 ♕e6 22 ♖ad1 ♘d5 23 a3 ♘×e3 24 ♕×e3 ♖fd8 25 ♖×d8+ ♖×d8 26 ♖e1 ♕b3 27 ♕b6 ♖d1 28 ♖×d1 ♕×d1+ 29 ♔h2 ♗f8 30 ♗c4 ♕d6+ 31 ♕×d6 ♗×d6+ 32 ♔g1 ♗f4 33 ♗d5 b6 34 b4 a×b4 35 c×b4 e3 36 ♔f1 e×f2 37 ♔×f2 b5 38 ♗c6 ♗c1 39 a4 b×a4 40 ♗×a4 ♔f8 41 b5 ♗a3 42 ♗b3 ♔e7 43 ♗d5 f6 44 ♔g3 ♗c5 45 ♔f4 ♔e8 ½-½

**Advance 1.0 – Nuchess** 1 e4 d5 2 e×d5 ♕×d5 3 ♘c3 ♕d6 4 d4 ♘f6 5 ♘b5 ♕b6 6 ♗f4 ♘d5 7 ♗g3 c6 8 c4 a6 9 c×d5 a×b5 10 d×c6 b×c6 11 ♘f3 g6 12 ♗e5 ♕a5+ 13 ♕d2 ♕×d2+ 14 ♘×d2 f6 15 ♗×b8 ♖×b8 16 0-0-0 ♗e6 17 b3 ♖a8 18 ♔b2 ♗h6 19 ♖e1 ♗d5 20 ♘b1 ♔f7 21 f3 b4 22 ♗d3 f5 23 a4 ♗g7 24 ♔c2 ♗×d4 25 ♘d2 ♖hd8 26 ♖d1 e6 27 h3 c5 28 ♖hf1 ♖ab8 29 a5 ♖a8 30 a6 ♗c3 31 ♖f2 ♖d6 32 h4 ♖ad8 33 ♖e2 h5 34 f4 ♗f6 35 g3 ♗×b3+ 36 ♘×b3 c4 37 ♗×c4 ♖×d1 38 ♘d2 ♖g1 39 ♖×e6 ♔g7 40 ♘f1 ♖c8 41 ♔b3 ♖×c4 42 ♔×c4 ♖×f1 43 ♔×b4 ♗d4 44 ♖e7+ ♔h6 45 ♔c4 ♗f2 46 ♖e8 ♖a1 47 ♖a8 ♗×g3 48 ♔d3 ♗×f4 49 a7 ♔g7 50 ♔e2 ♗g3 51 ♖d8 ♖×a7 adjudicated 0-1

**Master – Parwell** 1 ♘f3 ♘f6 2 g3 ♘a6 3 ♘c3 ♘c5 4 d4 ♘ce4 5 ♘×e4 ♘×e4 6 ♘g5 ♘×g5 7 ♗×g5 f6 8 ♗d2 e6 9 ♗g2 ♗e7 10 e4 0-0 11 0-0 d6 12 ♗c3 c6 13 a4 b6 14 d5 ♗a6 15 ♖e1 ♗c8 16 a5 b5 17 d×c6 ♕c7 18 ♗b4 ♖b8 19 ♖e3 ♖a8 20 ♖c3 g6 21 a6 ♕d8 22 ♗a5 ♕e8 23 ♗c7 b4 24 ♖c4 ♗×a6 25 ♖×a6 ♕c8 26 ♖a1 ♕×c7 27 ♖a6 ♖fb8 28 ♕g4 e5 29 ♕e6+ ♔h8 30 ♕f7 ♖f8 31 ♕d5 ♖fb8 32 ♕e6 ♖b5 33 ♕d7 ♕×d7 34 c×d7 ♖c5 35 ♖×c5 d×c5 36 ♖e6 b3 37 ♖×e7 b×c2 38 ♖e8+ ♔g7 39 ♖×a8 c1(♕)+ 40 ♗f1 1-0

**Mychess – Kaissa** 1 d4 ♘f6 2 c4 c5 3 d5 d6 4 ♘c3 g6 5 ♘f3 ♗g7 6 e4 0-0 7 ♗g5 h6 8 ♗×f6 ♗×f6 9 ♕d2 h5 10 0-0-0 ♗g4 11 ♗e2 ♘d7 12 h3 ♗×f3 13 ♗×f3 ♕a5 14 g4 ♗d4 15 ♔b1 ♘e5 16 ♗e2 ♗×f2 17 g×h5 ♗d4 18 h×g6 f×g6 19 ♘b5 ♕×d2 20 ♖×d2 ♗e3 21 ♖dd1 ♖f2 22 ♘c3 ♖af8 23 h4 ♗d4 24 ♖de1 ♖2f4 25 ♖c1 ♗×c3 26 ♖×c3 ♖×e4 27 ♖c2 ♖f2 28 ♗d1 ♖×c2 29 ♔×c2 ♖×c4+ 30 ♔d2 ♖d4+ 31 ♔e3 ♘c4+ 32 ♔e2 ♘×b2 33 ♗c2 c4 34 ♖b1 c3 35 ♖c1 ♖d2+ 36 ♔f3 ♘d3 37 ♗×d3 ♖×d3+ 38 ♔e4 ♖h3 39 ♔d4 b5 40 ♖e1 c2 41 ♖g1 ♔f7 42 ♖c1 ♖×h4+ 43 ♔d3 ♖h5 44 ♔d4 ♖h2 45 ♔d3 ♔f6 46 ♖e1 a6 47 a3 g5 48 ♖e6+ ♔f5 49 ♖e1 g4 50 ♔c3 g3 51 ♔b2 g2 52 ♖g1 ♖h1 0-1

**Chaos – Dark Horse** 1 d4 d5 2 c4 e6 3 ♘c3 c5 4 c×d5 e×d5 5 ♘f3 ♘f6 6 ♗g5 ♗e6 7 ♗×f6 g×f6 8 e4 d×e4 9 ♘×e4 c×d4 10 ♗b5+ ♗d7 11 ♗×d7+ ♘×d7 12 ♕×d4 ♕b6 13 ♕×b6 a×b6 14 0-0 ♖g8 15 ♖fd1 ♖g6 16 ♘d6+ ♗×d6 17 ♖×d6 f5 18 ♖e1+ ♔d8 19 ♖d5 ♖a5 20 ♖ed1 ♖×d5 21 ♖×d5 ♖c6 22 h3 ♖c5 23 ♖×c5 b×c5 24 ♘g5 ♘e5 25 ♘×h7 ♘d3 26 ♘g5 f6 27 ♘f7+ ♔e7 28 ♘h6 ♘×b2 29 ♘×f5+ ♔e6 30 ♘e3 ♘d3 31 a4 ♔d7 32 g3 ♘b2 33 a5 ♘d3 34 h4 ♘e5 35 ♔f1 ♘c6 36 ♘c4 ♔e6 37 h5 ♔d5 38 h6 ♘e5 39 ♘×e5 b6 40 a×b6 ♔×e5 41 b7 ♔e6 1-0

**Duchess – CSC** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗a4 ♘f6 5 0-0 ♘×e4 6 d4 b5 7 ♗b3 d5 8 d×e5 ♗e6 9 c3 ♗c5 10 ♕e2 0-0 11 ♗e3 ♗×e3 12 ♕×e3 ♘a5 13 ♘bd2 ♘×b3 14 a×b3 ♘×d2 15 ♕×d2 c5 16 ♕e3 ♕c8 17 ♖fc1 f6 18 b4 c×b4 19 c×b4 ♕d7 20 ♘d4 f×e5 21 ♕×e5 ♗g4 22 f3 ♖ae8 23 ♕g3 ♗×f3 24 ♖c7 ♕g4 25 ♘×f3 ♕×g3 26 h×g3 ♖e4 27 ♖×a6 ♖×b4 28 ♖aa7 ♖g4 29 ♔h2 h5 30 ♖d7 ♖f5 31 ♖d8+ ♔h7 32 ♖aa8 g5 33 ♖d7+ ♔g6 34 ♖a6+ ♖f6 35 ♖dd6 ♖×d6 36 ♖×d6+ ♔g7 37 ♖×d5 ♖b4 38 ♖×g5+ ♔h6 39 ♖c5 ♖×b2 40 ♘d4 b4 41 ♖b5 ♖b1 42 ♘c6 b3 43 ♘a5 ♖e1 44 ♘×b3 ♖e2 45 ♘d4 ♖e8 46 ♖b6+ ♔h7 47 ♘f5 ♖h8 48 ♖h6+ ♔g8 49 ♖f6 ♖h7 50 ♖d6 adjudicated 1-0

**Belle – BCP** 1 e4 c5 2 c3 ♘f6 3 e5 ♘d5 4 d4 c×d4 5 c×d4 d6 6 ♘f3 ♘c6 7 ♗c4 e6 8 ♗b3 d×e5 9 d×e5 ♗b4+ 10 ♗d2 0-0 11 0-0 b5 12 ♗×d5 ♕×d5 13 ♗×b4 ♕×d1 14 ♖×d1 ♘×b4 15 ♘c3 ♗a6 16 a4 b×a4 17 ♖×a4 ♖fb8 18 ♖d4 ♘c6 19 ♖×a6 ♘×d4 20 ♘×d4 ♖b4 21 ♘c6 ♖×b2 22 f4 ♖b7 23 ♔f2 h5 24 ♔e3 ♖b2 25 ♘e2 ♖b3+ 26 ♔e4 ♖b2 27 ♔f3 ♖b3+ 28 ♔f2 ♖b7 29 h3 ♖d7 30 ♘ed4 ♖b7 31 g4 h×g4 32 h×g4 ♖b2+ 33 ♔e3 ♖b7 34 f5 e×f5 35 ♘×f5 ♖d7 36 ♘d6 ♔h8 37 ♘b5 ♖d1 38 ♘c×a7 ♖ad8 39 ♖d6 ♖8×d6 40 e×d6 f6 41 ♘c6 g5 42 ♘bd4 ♖×d4 43 ♘×d4 ♔g7 44 d7 1-0

**Ostrich – Awit** 1 e4 d5 2 e×d5 ♘f6 3 ♗b5+ ♗d7 4 ♗c4 g6 5 ♘e2 ♗g7 6 ♘bc3 0-0 7 d4 ♖e8 8 ♗e3 ♘g4 9 ♕d2 ♘×e3 10 f×e3 ♘a6 11 0-0-0 ♖f8 12 ♖he1 ♗g4 13 h3 ♗d7 14 ♔b1 ♗f6 15 ♔a1 ♔g7 16 ♔b1 ♘b4 17 a3 ♘a6 18 ♗×a6 b×a6 19 e4 c6 20 e5 ♗h4 21 g3 c×d5 22 g×h4 ♗×h3 23 ♕g5 f6 24 e×f6+ e×f6 25 ♕×d5 a5 26 ♕×d8 ♖a×d8 27 d5 ♖d6 28 ♘e4 ♖b6 29 c4 ♖fb8 30 ♖d2 ♖d8 31 c5 ♖b3 32 ♘d4 ♖b7 33 ♘c6 ♖e8 34 ♘×a5 ♖be7 35 ♘c6 ♖×e4 36 ♖×e4 ♖×e4 37 ♘×a7 ♖×h4 38 c6 ♗f5+ 39 ♔c1 ♖c4+ 40 ♔d1 ♗g4+ 41 ♔e1 ♖e4+ 42 ♔f2 ♖e7 43 d6 ♖×a7 44 d7 ♖×d7 45 ♖×d7+ ♔f8 46 ♖d8+ ♔e7 47 c7 ♗d7 48 b4 h5 49 b5 f5 50 b6 g5 51 b7 ♗c6 1-0

*Round 2, September 26, 1980*

**Duchess – Master** 1 e4 d6 2 d4 ♘f6 3 ♘c3 g6 4 ♘f3 ♗g7 5 ♗e2 0-0 6 0-0 b6 7 e5 ♘fd7 8 ♗f4 ♘c6 9 ♗b5 ♗b7 10 ♗×c6 ♗×c6 11 d5 ♗b7 12 ♖e1 d×e5 13 ♘×e5 ♘×e5 14 ♗×e5 f6 15 ♗f4 g5 16 ♗c1 ♖e8 17 ♖e6 f5 18 ♗×g5 ♗×c3 19 b×c3 ♕×d5 20 ♕×d5 ♗×d5 21 ♖e5 e6 22 g3 ♔g7 23 a3 h6 24 ♗h4 ♗f3 25 ♖e3 ♗c6 26 ♖ae1 ♗d5 27 f3 c5 28 ♔g2 a5 29 ♖b1 ♖ab8 30 ♖a1 c4 31 ♖d1 ♔g6 32 ♖d4 a4 33 ♖e1 b5 34 ♖d2 ♖f8 35 ♖b1 ♖f7 36 ♖d4 ♖d7 37 h3 ♖b6 38 ♖e1 ♖bd6 39 g4 f×g4 40 ♖×g4+ ♔h7 41 ♗g3 ♖b6 42 ♗f4 h5 43 ♖h4 ♖g7+ 44 ♔f2 ♔g8 45 ♖×h5 ♖f7 46 ♖g5+ ♔f8 47 ♔g3 ♖f6 48 h4 ♖b7 49 h5 ♖h7 50 ♔g4 ♔f7 51 ♖e3 ♗c6 52 ♖g6 ♗d7 53 ♖×f6+ ♔×f6 54 ♗g5+ ♔g7 55 ♖e1 e5+ 56 ♔h4 ♗e8 adjudicated 1-0

**BCP – Schach 2.3** 1 e4 e5 2 ♘f3 ♘c6 3 d4 e×d4 4 ♘×d4 ♗c5 5 ♘b3 ♗b6 6 ♘c3 ♘f6 7 ♗b5 0-0 8 0-0 ♘e5 9 ♗g5 c6 10 ♗e2 d6 11 ♕c1 ♗e6 12 ♔h1 h6 13 ♗h4 g5 14 ♗g3 c5 15 f4 g×f4 16 ♕×f4 ♗×b3 17

a×b3 ♘fd7 18 ♕×h6 f6 19 ♗c4+ ♖f7 20 ♗×e5 ♘×e5 21 ♖×f6 ♕×f6 22 ♕×f6 ♘×c4 23 ♕g6+ ♖g7 24 ♕e6+ ♔h7 25 b×c4 ♖f8 26 ♕×d6 ♖f2 27 ♕d3 ♖f×g2 28 ♕h3+ ♔g8 29 ♕×g2 ♖×g2 30 ♔×g2 ♔f7 31 ♘d5 ♔e6 32 ♘×b6 a×b6 33 ♖a7 ♔e5 34 ♔f3 ♔d4 35 ♖×b7 ♔e5 36 c3 ♔d6 37 ♖×b6+ ♔e5 38 b4 c×b4 39 ♖b5+ ♔e6 40 ♖b6+ ♔e5 41 ♖b5+ ♔e6 42 ♖b6+ ♔e5 ½-½

**CSC – Advance 1.0** 1 e4 e6 2 d4 d5 3 ♘c3 ♗b4 4 e5 c5 5 a3 ♗×c3+ 6 b×c3 ♘e7 7 d×c5 ♕a5 8 ♕d4 ♘bc6 9 ♕e3 d4 10 ♕g3 ♘f5 11 ♕d3 ♕×c5 12 ♘f3 d×c3 13 ♕b5 ♕d5 14 ♕×d5 e×d5 15 ♗g5 h6 16 ♗c1 ♘fd4 17 ♘×d4 ♘×d4 18 ♗d3 ♗f5 19 ♗×f5 ♘×f5 20 f4 ♘d4 21 ♔d1 0-0 22 ♖b1 b6 23 ♗e3 ♘f5 24 ♗f2 ♖ac8 25 ♔e2 ♖c4 26 ♖b4 ♖×b4 27 a×b4 ♖c8 28 ♖a1 ♖c4 29 ♔f3 g5 30 g3 ♖×b4 31 ♖×a7 g×f4 32 ♖c7 f×g3 33 h×g3 ♘d4+ 34 ♗×d4 ♖×d4 35 ♖×c3 ♖e4 36 ♖c6 ♖×e5 37 ♖×b6 ♔g7 38 c3 h5 39 ♖a6 f6 40 ♖a7+ ♔g6 41 ♖d7 ♖g5 42 ♔f2 ♔h6 43 ♖f7 ♖f5+ 44 ♔e3 ♔g6 45 ♖d7 ♖g5 46 ♔f2 ♖e5 47 ♔f3 ♖f5+ 48 ♔g2 ♖g5 49 ♔f2 ♖e5 50 ♔f3 ♖f5+ 51 ♔g2 ♖g5 52 ♔f2 ♖e5 ½-½

**Nuchess – Belle** 1 f4 e5 2 e4 e×f4 3 ♘f3 d5 4 e×d5 ♘f6 5 ♗b5+ c6 6 d×c6 ♘×c6 7 d4 ♗d6 8 0-0 0-0 9 ♘c3 ♗g4 10 ♗c4 ♗b4 11 ♘e2 ♗d6 12 ♘×f4 ♗×f3 13 ♖×f3 ♗c5 14 c3 ♘e5 15 ♗e2 ♘×f3+ 16 ♗×f3 ♗d6 17 ♗×b7 ♖b8 18 ♕f3 ♖e8 19 ♗c6 ♖e1+ 20 ♔f2 ♖×c1 21 ♖×c1 ♖×b2+ 22 ♔g1 ♕c7 23 g3 ♗×f4 24 g×f4 ♖×a2 25 ♖b1 ♕d6 26 ♗b5 ♕b8 27 ♗d3 ♕e8 28 ♔h1 a5 29 ♗b5 ♕d8 30 ♗c4 ♖c2 31 ♖b7 ♕a8 32 ♗×f7+ ♔h8 33 ♗d5 ♕e8 34 ♖b1 ♘×d5 35 ♕×d5 ♖×c3 36 ♕e5 ♕a8+ 37 d5 ♖c8 38 ♖g1 ♕b7 39 f5 ♕f7 40 ♖b1 a4 41 ♖b8 ♖×b8 42 ♕×b8+ ♕g8 43 ♕e5 a3 44 f6 h6 45 f×g7+ ♕×g7 46 ♕e8+ ♔h7 47 ♕e4+ ♔h8 48 ♕e8+ ♔h7 49 ♕e4+ ♔h8 50 ♕e8+ ½-½

**Parwell – Mychess** 1 f4 e5 2 f×e5 d6 3 e×d6 ♗×d6 4 e3 ♕h4+ 5 ♔e2 ♗×h2 6 ♘f3 ♗g4 7 c3 ♘c6 8 ♕c2 0-0-0 9 ♕b3 f6 10 e4 h5 11 d3 ♘ge7 12 ♗e3 ♕g3 13 ♕a3 a6 14 ♗f2 ♕f4 15 ♗e3 ♕g3 16 ♗f2 ♕e5 17 ♔e1 ♗×f3 18 g×f3 ♕f4 19 ♘d2 h4 20 ♔d1 h3 21 ♕b3 f5 22 ♗c5 ♔b8 23 d4 a5 24 ♗×e7 ♘×e7 25 ♕f7 ♘c6 26 ♕×f5 ♕×f5 27 e×f5 ♗f4 28 f6 g×f6 29 ♘e4 f5 30 ♘c5 ♘×d4 31 c×d4 ♖×d4+ 32 ♘d3 h2 33 ♔c2 c6 34 ♗e2 ♗g3 35 ♔b1 ♖e8 36 ♘c1 ♖d5 37 a3 ♖d2 38 ♔a2 ♖d×e2 39 ♘×e2 ♖×e2 40 ♖hd1 ♗e5 41 ♖ab1 ♖e3 42 ♖d8+ ♔a7 43 f4 ♗×f4 44 ♖f8 ♖h3 45 ♖×f5 h1(♕) 46 ♖×h1 ♖×h1 47 ♖×f4 ♖h3 48 ♖f7 ♔b6 49 b3 ♖h2+ 50 ♔b1 c5 51 ♖g7 ♔c6 52 ♖g6+ ♔d5 53 ♖b6 ♖h1+ 54 ♔c2 ♖h2+ 55 ♔b1 ♖h1+ 56 ♔b2 ♖h2+ 57 ♔b1 ♖h7 58 ♖b5 ♔c6 59 ♖×a5 ½-½

**Kaissa – Ostrich** 1 e4 d5 2 e×d5 ♘f6 3 d4 ♘×d5 4 c4 ♘b6 5 ♘c3 e5 6 d×e5 ♕×d1+ 7 ♔×d1 ♗g4+ 8 f3 ♗f5 9 ♗e3 ♗b4 10 ♘d5 ♘×d5 11 c×d5 ♘d7 12 g4 ♗g6 13 ♖c1 0-0-0 14 a3 ♗a5 15 b4 ♗b6 16 ♗×b6 a×b6 17 d6 ♘×e5 18 ♖×c7+ ♔b8 19 ♖e7 ♖×d6+ 20 ♔e1 ♖d5 21 ♗g2 ♖hd8 22 ♘e2 ♘c4 23 a4 ♘b2 24 f4 ♖5d7 25 ♖×d7 ♖×d7 26 f5 ♗×f5 27 g×f5 ♘×a4 28 ♖g1 ♘b2 29 ♗e4 g6 30 ♖g3 ♖e7 31 ♗c2 ♖c7 32 ♖c3 g×f5 33 ♖×c7 ♔×c7 34 ♗×f5 h6 35 ♘c3 ♘c4 36 ♔e2 ♘d6 37 ♗d3 ♔d7 38 ♔f3 ♔c6 39 ♔g4 b5 40 ♔h5 ♘c4 41 ♔×h6 ♘d6 42 ♔g5 ♔b6 43 ♔f6 ♔c6 44 h4 ♔b6 45 h5 ♔c6 46 h6 ♘e8+ 47 ♔e7 1-0

**L'Excentrique – Chaos** 1 e4 ♘f6 2 e5 ♘d5 3 d4 d6 4 ♘f3 d×e5 5 ♘×e5 g6 6 c4 ♘b6 7 c5 ♘6d7 8 ♗c4 ♘×e5 9 d×e5 ♕×d1+ 10 ♔×d1 ♘d7 11 e6 f×e6 12 ♗e3 ♘e5 13 ♗d4 ♗g7 14 ♖e1 ♘×c4 15 ♗×g7 ♖g8 16 ♗c3 b6 17 b3 ♘a5 18 b4 ♘b7 19 c×b6 c×b6 20 ♗e5 ♔f7 21 ♖e3 ♘d6 22 g4 ♗b7 23 ♘d2 ♖ac8 24 ♖h3 h5 25 a4 ♘c4 26 ♘×c4 ♖×c4 27 ♔d2 ♖d8+ 28 ♖d3 ♖×d3+ 29 ♔×d3 ♖×g4 30 f4 ♖g2 31 h4 ♖h2 32 ♖c1 a6 33 ♖c7 ♗d5 34 ♗d6 ♖×h4 35 ♖×e7+ ♔g8 36 ♖e8+ ♔g7 37 ♖e7+ ♔f6 38 ♖a7 ♖h3+ 39 ♔d4 ♖a3 40 ♖×a6 h4 41 ♖a7 ♔f5 42 ♖f7+ ♔g4 43 ♖g7 ♖×a4 44 ♖×g6+ ♔f3 45 ♖h6 ♖a2 46 ♖×h4 ♖d2+ 47 ♔c3 ♖g2 48 b5 ♖a2 49 ♖h3+ ♔g4 50 ♖h8 ♖a5 51 ♔b4 ♖a1 52 ♖g8+ ♔f3 53 ♖c8 ♖b1+ 54 ♔a4 ♗b3+ 55 ♔a3 ♗d1 56 ♖c3+ ♔e4 57 ♔a2 ♖×b5 58 ♗b8 ♖a5+ 59 ♔b2 ♔f5 60 ♔c1 ♖d5 61 ♖g3 ♗g4 62 ♖e3 b5 63 ♖e5+ ♖×e5 ½-½

**Dark Horse – Awit** 1 e4 c5 2 ♘f3 d6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 a6 6 ♗e3 g6 7 ♗e2 ♗d7 8 0-0 ♗g7 9 a4 0-0 10 a5 ♕c7 11 ♕d3 ♖c8 12 ♖fd1 ♘c6 13 ♘×c6 ♕×c6 14 ♖a4 ♗g4 15 f3 ♗d7 16 ♖c4 ♕×c4 17 ♕×c4 ♖×c4 18 ♗×c4 ♖c8 19 ♗d5 ♘×d5 20 ♘×d5 ♖×c2 21 ♘×e7+ ♔f8 22 ♗g5 f6 23 ♘×g6+ h×g6 24 ♖×d6 ♔e7 25 ♗f4 g5 26 ♗g3 ♖×b2 27 ♖d1 b5 28 a×b6 a5 29 ♗d6+ ♔e8 30 ♗c5 ♔d8 31 g3 a4 32 ♔h1 ♖b3 33 f4 ♔c8 34 f×g5 f×g5 35 ♖d6 ♗f8 36 ♖d5 ♗×c5 37 ♖×c5+ ♔b8 38 ♖×g5 ♗c6 39 ♖g8+ ♔b7 40 ♖g4 a3 41 ♖h4 a2 42 ♖h7+ ♔×b6 0-1

**Chess 4.9 – BeBe** 1 e4 c5 2 ♘f3 d6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 g6 6 ♗e2 ♗g7 7 0-0 0-0 8 ♗g5 ♗d7 9 ♘d5 ♘×d5 10 e×d5 ♕b6 11 ♘b3 ♗×b2 12 ♗×e7 ♖e8 13 ♖b1 ♖×e7 14 ♖×b2 ♘a6 15 ♖b1 ♘b4 16 ♘c1 ♕c5 17 c4 ♖ae8 18 ♕d2 a5 19 a3 ♘a6 20 ♖×b7 ♕×a3 21 ♕f4 ♗f5 22 ♖×e7 23 c5 ♘×c5 24 ♕×d6 ♖e8 25 ♗b5 ♖f8 26 ♘b3 ♕×b3 27 ♕×c5 ♖c8 28 ♗c6 ♕b4 29 ♕a7 ♕a3 30 ♖e1 ♕c3 31 ♕e3 ♕b4 32 ♖a1 ♕b2 33 ♕e1 ♕b4 34 ♕e5 ♖b8 35 h3 ♖c8 36 ♕e3 ♗c2 37 ♕a3 ♕×a3 38 ♖×a3 a4 39 f4 f5 40 ♔f2 ♔f7 41 ♔e3 ♔f6 42 ♔d4 ♖b8 43 d6 ♗e4 44 ♗×e4 ♖b4+ 45 ♔d5 ♖×e4 46 g3 h6 47 ♖d3 ♔f7 48 d7 ♔e7 49 ♔c6 adjudicated ½-½

*Round 3, September 27, 1980*

**BeBe – Master** 1 e4 d6 2 d4 ♘f6 3 ♘c3 g6 4 f4 ♗g7 5 ♘f3 c5 6 d×c5 ♕a5 7 ♗b5+ ♗d7 8 ♕d3 ♘×e4 9 ♕×e4 ♗×b5 10 ♕×b7 ♗×c3+ 11 b×c3 ♗c6 12 ♕c8+ ♕d8 13 ♕×d8+ ♔×d8 14 ♗a3 ♘d7 15 c×d6 e×d6 16 0-0-0 ♗×f3 17 g×f3 ♖e8 18 ♖×d6 ♖e6 19 ♖d5 ♔e8 20 f5 ♖e2 21 f×g6 h×g6 22 ♗b4 ♖c8 23 ♖hd1 ♘b6 24 ♖5d2 ♖×d2 25 ♖×d2 ♘c4 26 ♖e2+ ♔d8 27 f4 a5 28 ♗f8 ♖c7 29 h3 ♔c8 30 h4 ♔b7 31 ♖e8 ♔c6 32 a4 ♔b7 33 ♗e7 ♖c6 34 ♗d8 f5 35 ♖f8 ♖d6 36 ♔b1 ♔c6 37 ♔a2 ♔d7 38 ♔b3 ♘d2+ 39 ♔a2 ♘c4 40 ♔b3 ♘d2+ 41 ♔a2 ♘c4 ½-½

**Kaissa – Duchess** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6

4 ♗a4 d6 5 0-0 ♘f6 6 c3 ♗d7 7 d4 ♗e7 8 ♖e1 0-0 9 ♘bd2 ♖e8 10 ♘f1 h6 11 ♘g3 ♗f8 12 a3 ♘a5 13 ♗c2 c6 14 ♗e3 ♘c4 15 b3 ♘xe3 16 ♖xe3 b6 17 h3 ♕c7 18 ♖e1 c5 19 dxe5 dxe5 20 ♕e2 ♗b5 21 ♗d3 ♗xd3 22 ♕xd3 ♖ed8 23 ♕e3 b5 24 ♖ed1 c4 25 b4 a5 26 ♖xd8 ♖xd8 27 ♕e1 ♖d3 28 ♔h1 g6 29 bxa5 ♗c5 30 a4 bxa4 31 a6 ♕a7 32 ♖a2 ♘d7 33 ♕a1 a3 34 ♕c1 ♗xf2 35 ♘f1 ♗c5 36 ♘e1 ♖d6 37 ♕xh6 ♖d1 38 ♕h4 ♕xa6 39 ♕d8+ ♔g7 40 ♕h4 ♕f6 41 ♕xf6+ ♘xf6 42 ♘f3 ♖xf1+ 43 ♔h2 ♘xe4 44 ♖e2 ♖h1+ 45 ♔xh1 ♘g3+ 46 ♔h2 ♘xe2 47 ♘e1 a2 0-1

**Belle – L'Excentrique** 1 e4 c6 2 d4 d5 3 ♘c3 dxe4 4 ♘xe4 ♘d7 5 ♗c4 ♘df6 6 ♘g5 e6 7 ♘e2 h6 8 ♘f3 ♗d6 9 0-0 b5 10 ♗d3 ♘e7 11 a4 bxa4 12 ♖xa4 0-0 13 ♗e3 ♘g4 14 ♗f4 ♗xf4 15 ♘xf4 ♕d6 16 ♘h5 g6 17 ♘g3 f5 18 ♕a1 a6 19 ♖e1 ♘f6 20 ♗xa6 ♗xa6 21 ♖xa6 ♖xa6 22 ♕xa6 f4 23 ♘e4 ♘xe4 24 ♖xe4 g5 25 c4 ♖b8 26 c5 ♕d5 27 ♖e2 g4 28 ♕a7 ♖e8 29 ♖e5 ♕d8 30 ♘e1 ♘f5 31 ♕a6 ♘xd4 32 ♕d3 ♔g7 33 ♕e4 ♖f8 34 ♔h1 ♖f5 35 ♘d3 ♔f6 36 ♘xf4 g3 37 f3 gxh2 38 ♖xe6+ ♘xe6 39 ♕xe6+ ♔g7 40 ♕xf5 ♔g8 41 ♕g6+ ♔h8 42 ♕f7 ♕g5 43 ♘g6+ ♕xg6 44 ♕xg6 h5 45 f4 h4 46 f5 h3 47 f6 hxg2+ 48 ♕xg2 1-0

**Advance 1.0 – Parwell** 1 e4 e6 2 d4 d5 3 ♘c3 ♗b4 4 e5 ♘e7 5 ♗d2 0-0 6 a3 ♗xc3 7 ♗xc3 ♘f5 8 ♘f3 ♘h4 9 ♘xh4 ♕xh4 10 ♗d3 ♗d7 11 0-0 ♘c6 12 g3 ♕h3 13 ♕d2 ♕h5 14 ♖ae1 ♖ac8 15 f3 ♖b8 16 b3 a6 17 ♕g2 a5 18 h3 b5 19 b4 a4 20 f4 ♘e7 21 g4 ♕h6 22 f5 exf5 23 gxf5 ♖be8 24 f6 ♗xh3 25 ♕xg7+ ♕xg7+ 26 fxg7 ♔xg7 27 ♗xb5 c6 28 ♖f3 cxb5 29 ♖xh3 ♔h8 30 ♔h1 f6 31 e6 ♘f5 32 ♖f3 ♘d6 33 e7 ♖f7 34 ♖e6 ♘c4 35 ♗e1 ♖fxe7 36 ♖xe7 ♖xe7 37 ♗h4 ♔g8 38 ♖g3+ ♔f7 39 ♖f3 ♖e3 40 ♖xf6+ ♔g8 41 ♗f2 ♖c3 42 ♔g1 ♘xa3 43 ♖f5 ♖xc2 44 ♖xd5 ♖c1+ 45 ♔h2 ♖b1 46 ♖d8+ ♔g7 47 ♖d7+ ♔g6 48 ♖d6+ ♔h5 49 ♖d7 h6 50 ♖d6 ♖b2 51 ♔g2 ♖xb4 52 ♗e3 ♘c2 53 ♖xh6+ ♔g4 54 ♗f2 ♖b3 55 d5 ♖d3 56 d6 b4 57 ♖g6+ ♔f5 58 ♖h6 a3 59 ♖h5+ ♔g6 60 ♖a5 b3 61 d7 a2 62 ♖a6+ ♔f5 63 ♖a5+ ♔e6 64 ♖a6+ ♔xd7 65 ♖a7+ ♔c6 66 ♖a6+ ♔b5 67 ♖a8 a1(♕) 68 ♖xa1 ♘xa1 adjudicated 0-1

**Dark Horse – CSC** 1 e4 e5 2 ♘f3 ♘c6 3 d4 exd4 4 ♘xd4 ♕h4 5 ♘b5 ♕xe4+ 6 ♗e2 ♗b4+ 7 ♗d2 ♗xd2+ 8 ♘xd2 ♕f4 9 g3 ♕e5 10 ♘f3 ♕xb2 11 ♘xc7+ ♔d8 12 ♘xa8 ♘b4 13 ♗d3 ♘xa2 14 0-0 ♘h6 15 ♖e1 f6 16 ♕e2 ♕b4 17 ♖xa2 a6 18 ♗xh7 d5 19 ♖a4 ♕c5 20 ♗d3 b5 21 ♖aa1 ♗b7 22 ♕e3 ♕xe3 23 fxe3 ♔d7 24 ♘b6+ ♔c7 25 ♘a4 bxa4 26 ♖xa4 ♖a8 27 ♖b1 ♗c8 28 ♖bb4 ♗b7 29 ♘d4 ♗c8 30 e4 ♗d7 31 ♖a5 dxe4 32 ♗xe4 ♖a7 33 ♗d3 ♗c8 34 ♘b5+ ♔b6 35 ♘xa7+ ♔xa7 36 ♗e4 f5 37 ♖c5 fxe4 38 ♖c7+ ♔a8 39 ♖xc8+ ♔a7 40 ♖c7+ ♔a8 41 ♖xe4 ♔b8 42 ♖xg7 ♘f7 43 ♖xf7 a5 44 ♖e8# 1-0

**Schach 2.3 – Ostrich** 1 e4 d5 2 exd5 ♘f6 3 d4 ♘xd5 4 ♘f3 ♗g4 5 h3 ♗xf3 6 ♕xf3 ♘c6 7 c3 e6 8 ♘d2 ♗d6 9 ♗d3 0-0 10 0-0 ♕h4 11 ♘c4 ♖ae8 12 ♘xd6 cxd6 13 ♗d2 a5 14 b3 ♔h8 15 ♖ae1 ♔g8 16 ♖e2 ♖e7 17 ♖ee1 ♖a8 18 ♗e4 ♘c7 19 ♗xc6 bxc6 20 ♕xc6 d5 21 g3 ♕xh3 22 ♗f4 e5 23 ♖xe5 ♕d7 24 ♕b7 ♖xe5 25 ♗xe5 ♖c8 26 c4 dxc4 27 bxc4 ♕d8 28 c5 ♘e6 29 c6 f6 30 ♗d6 ♘xd4 31 c7 ♕d7 32 ♕d5+ ♘e6 33 ♖e1 ♔f7 34 ♖e3 g5 35 f4 gxf4 36 gxf4 ♖g8+ 37 ♔f2 a4 38 f5 ♖e8 39 ♖xe6 ♖xe6 40 fxe6+ ♕xe6 41 ♕xe6+ 1-0

**Awit – Chess 4.9** 1 b3 ♘f6 2 ♗b2 d5 3 e3 ♗d7 4 ♘f3 e6 5 ♗e2 ♘c6 6 0-0 ♗e7 7 ♘e5 0-0 8 ♘xd7 ♕xd7 9 f4 ♘e4 10 ♘c3 ♘xc3 11 ♗xc3 d4 12 ♗b2 ♗c5 13 ♗c1 ♖ad8 14 ♖b1 f5 15 ♔f2 ♘b4 16 a3 dxe3+ 17 dxe3 ♕c6 18 ♗d3 ♘d5 19 ♕f3 a5 20 ♗d2 e5 21 fxe5 ♕e6 22 b4 axb4 23 axb4 f4 24 ♗xh7+ ♔xh7 25 ♕h5+ ♔g8 26 bxc5 fxe3+ 27 ♔e2 ♕a6+ 0-1

**Mychess – BCP** 1 d4 ♘f6 2 c4 g6 3 ♘c3 d5 4 cxd5 ♘xd5 5 e4 ♘xc3 6 bxc3 ♗g7 7 ♗c4 c5 8 ♘f3 ♘c6 9 ♗d5 cxd4 10 cxd4 0-0 11 ♖b1 ♘xd4 12 ♘xd4 e6 13 ♗xb7 ♗xb7 14 ♖xb7 ♕xd4 15 ♕xd4 ♗xd4 16 ♔e2 ♖fc8 17 ♗e3 ♖c2+ 18 ♔d3 ♖b2 19 ♖b1 ♖xb1 20 ♖xb1 ♗xe3 21 ♔xe3 ♖c8 22 ♖b7 ♖c3+ 23 ♔d2 ♖c4 24 ♔d3 ♖c1 25 ♖xa7 ♖g1 26 g3 ♖g2 27 h4 ♖xf2 28 ♔e3 ♖c2 29 a3 ♔g7 30 e5 g5 31 hxg5 ♖c3+ 32 ♔f4 ♖c2 33 g4 ♖c4+ 34 ♔g3 ♔g6 35 ♔h4 ♖c5 36 a4 ♔g7 37 ♖b7 ♖c4 38 ♖a7 ♖c5 39 ♖b7 ♖c4 40 a5 ♖c5 41 a6 ♖xe5 42 a7 ♖a5 43 ♖d7 e5 44 ♖e7 e4 45 ♖xe4 ♖xa7 46 ♖e2 ♖a5 47 ♖c2 f5 48 gxf6+ ♔xf6 49 g5+ ♔e5 50 ♔h5 ♖a3 ½-½

**Chaos – Nuchess** 1 d4 d5 2 c4 dxc4 3 ♘f3 ♘f6 4 e3 e6 5 ♗xc4 c5 6 ♕e2 a6 7 0-0 b5 8 ♗b3 ♗b7 9 ♖d1 ♘bd7 10 ♘c3 ♗d6 11 e4 cxd4 12 ♘xd4 ♕b8 13 g3 b4 14 ♘a4 ♗xe4 15 f3 ♗b7 16 ♘xe6 fxe6 17 ♕xe6+ ♔d8 18 ♖xd6 ♕c7 19 ♗e3 ♖e8 20 ♗b6 ♖xe6 21 ♗xe6 ♗xf3 22 ♖c1 ♖a7 23 ♖xc7 ♖xc7 24 ♗xc7+ ♔xc7 25 ♖xa6 ♗c6 26 ♖a7+ ♔b8 27 ♖a5 ♔c7 28 b3 ♔d6 29 ♗xd7 ♗xd7 30 ♘b6 ♔c6 31 ♘xd7 ♘xd7 32 ♔f2 ♘c5 33 ♔e3 ♔b6 34 ♖a8 ♘e6 35 ♔e4 ♔c5 36 ♖a5+ ♔b6 37 ♖e5 ♘c5+ 38 ♔d4 ♘b7 39 ♖e7 ♘d6 40 ♔e5 ♘b5 41 ♖xg7 h5 42 ♖f7 ♘c3 43 ♖f2 ♔c5 44 ♖d2 ♘b5 1-0

*Round 4, September 28, 1980*

**Chaos – Kaissa** 1 d4 ♘f6 2 c4 c5 3 d5 d6 4 ♘c3 g6 5 e4 ♗g7 6 ♗e2 0-0 7 ♗g5 h6 8 ♗f4 ♕b6 9 ♕d2 g5 10 ♗g3 ♘h7 11 ♘f3 g4 12 ♘h4 ♘g5 13 ♘f5 ♗xf5 14 exf5 ♕a6 15 ♕e3 ♖e8 16 ♗h4 ♗f6 17 0-0 ♘d7 18 ♖fe1 ♕b6 19 ♖ab1 ♕a6 20 ♕f4 ♗e5 21 ♕e3 ♗f6 22 ♕f4 ♗e5 23 ♕d2 ♗f6 24 ♕e3 ♘e5 25 ♕g3 ♖f8 26 h3 ♘xc4 27 ♕xg4 b5 28 ♘xb5 ♘e5 29 ♕g3 ♕b6 30 ♘c3 ♘d7 31 f4 c4+ 32 ♔h1 ♖fb8 33 fxg5 ♗e5 34 ♕g4 hxg5 35 ♕xg5+ ♗g7 36 ♗xc4 ♘e5 37 ♗b5 a6 38 ♗e2 ♖a7 39 ♕g3 ♔f8 40 ♕f2 ♖ab7 41 ♕xb6 ♖xb6 42 ♗f2 ♖xb2 43 ♖xb2 ♖xb2 44 ♖b1 ♖xb1+ 45 ♘xb1 a5 46 ♘d2 ♗h6 47 ♘c4 ♘xc4 48 ♗xc4 ♗g7 49 g4 a4 50 ♔g2 a3 51 ♔f3 ♗b2 1-0

**Chess 4.9 – Schach 2.3** 1 e4 e5 2 ♘f3 ♘c6 3 d4 exd4 4 c3 dxc3 5 ♗c4 ♘f6 6 ♘xc3 ♗d6 7 ♕e2 0-0 8 0-0 ♘e5 9 ♗d2 c6 10 ♗b3 b5 11 ♘xe5 ♗xe5 12 f4 ♗d4+ 13 ♔h1 ♖e8 14 e5 ♗xc3 15 ♗xc3 d5 16 a4 d4 17 ♗d2 bxa4 18 ♖xa4 c5 19 ♗a5 ♕d7 20 ♕c2

♗a6 21 ♖fa1 d3 22 ♕×c5 ♖ac8 23 ♕g1 ♘h5 24 ♗d2 ♗b7 25 ♖×a7 ♖c7 26 ♗d1 ♕h3 27 ♕f2 ♕f5 28 ♗f3 ♖b8 29 ♗×b7 ♘×f4 30 ♗×f4 d2 31 ♕×d2 g5 32 ♗×g5 ♖c×b7 33 ♖×b7 ♖×b7 34 ♖a8+ ♔g7 1-0

**Ostrich – Mychess** 1 e4 c5 2 ♘f3 d6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 ♘c6 6 ♗b5 ♗d7 7 0-0 ♘×d4 8 ♕×d4 e5 9 ♗×d7+ ♕×d7 10 ♕d1 ♗e7 11 ♗e3 ♘g4 12 a4 ♘×e3 13 f×e3 0-0 14 ♔h1 ♖ac8 15 ♕e1 f6 16 ♕g3 f5 17 e×f5 ♖×f5 18 ♖×f5 ♕×f5 19 e4 ♕e6 20 ♖d1 b6 21 ♖d5 ♕h6 22 ♕g4 ♖f8 23 ♖d1 ♖f2 24 ♕c8+ ♗f8 25 ♘b5 ♕g6 26 ♕c4+ ♔h8 27 ♖g1 a5 28 ♕c6 ♔g8 29 ♘a3 ♖f4 30 ♖e1 ♗e7 31 ♕×b6 ♖×e4 32 ♕×a5 ♖×e1+ 33 ♕×e1 e4 34 a5 d5 35 b4 e3 36 c3 ♕d3 37 ♘b1 e2 38 ♘d2 ♗f6 39 a6 ♗×c3 40 a7 ♕a6 41 ♕×e2 ♕a1+ 42 ♘f1 ♗×b4 43 ♕e6+ ♔f8 44 ♕c8+ ♔e7 45 ♕b7+ ♔e6 46 ♕c8+ ♔f6 47 ♕c6+ ♔e5 48 ♕e8+ ♔f6 49 ♕c6+ ♔e5 50 ♕e8+ ♔f6 51 ♕c6+ ♔e5 ½-½

**Master – L'Excentrique** 1 ♘f3 f5 2 g3 e6 3 ♗g2 ♘f6 4 d3 ♗e7 5 0-0 0-0 6 ♘bd2 d5 7 ♘b3 b6 8 c3 ♘c6 9 ♘bd4 ♘×d4 10 ♘×d4 ♗d7 11 ♗f4 c5 12 ♘c2 a5 13 f3 ♘h5 14 ♗e5 ♗a4 15 b3 ♗e8 16 f4 a4 17 e4 a×b3 18 a×b3 d×e4 19 d×e4 ♖×a1 20 ♘×a1 ♕×d1 21 ♖×d1 f×e4 22 ♗×e4 g5 23 f×g5 ♗×g5 24 ♖e1 ♖f7 25 ♔g2 ♗f6 26 ♗f3 ♘g7 27 g4 ♗×e5 28 ♖×e5 ♖d7 29 ♗e4 ♖d2+ 30 ♔f3 ♖×h2 31 ♔f4 ♗g6 32 ♗×g6 h×g6 33 ♖e1 ♖h3 34 c4 ♖d3 35 ♖e5 ♔f7 36 ♘c2 ♖×b3 37 ♘e3 ♖d3 38 ♘c2 ♖d2 39 ♘e3 ♖d4+ 40 ♔g5 ♘e8 41 ♘c2 ♖×c4 42 ♘e3 0-1

**BCP – Parwell** 1 e4 e5 2 ♘f3 ♘c6 3 d4 e×d4 4 ♘×d4 ♗c5 5 ♘b3 ♗b6 6 ♘c3 ♘f6 7 ♗b5 0-0 8 0-0 ♘e5 9 ♗g5 c6 10 ♗e2 d6 11 ♕c1 ♗e6 12 ♔h1 h6 13 ♗h4 ♘g6 14 ♗×f6 ♕×f6 15 f4 ♘h4 16 f5 ♗×b3 17 g3 d5 18 a×b3 d×e4 19 g×h4 ♕×h4 20 ♖f4 ♕h3 21 ♘×e4 ♖fe8 22 ♘c3 ♗c7 23 ♖f2 ♗d6 24 ♖g2 ♗e5 25 ♗g4 ♕h4 26 ♕d2 ♗×h2 27 ♖×h2 ♕×g4 28 ♖g1 ♕f3+ 29 ♕g2 ♖e3 30 ♖×h6 ♕×g2+ 31 ♖×g2 a6 32 ♖h4 ♖d8 33 ♖gh2 ♖e1+ 34 ♔g2 ♖d2+ 35 ♔g3 ♖×h2 36 ♖×h2 b6 37 ♔f4 ♖f1+ 38 ♔e4 ♖e1+ 39 ♘e2 ♖b1 40 ♘d4 c5 adjudicated 1-0

**Nuchess – BeBe** 1 ♘c3 d5 2 d4 ♗f5 3 f3 ♘f6 4 g4 ♗g6 5 g5 ♘h5 6 e4 e6 7 e×d5 e×d5 8 ♕e2+ ♗e7 9 ♕b5+ ♘c6 10 ♘×d5 0-0 11 c3 ♗×g5 12 ♗×g5 ♕×g5 13 ♘h3 ♖fe8+ 14 ♗e2 ♖×e2+ 15 ♕×e2 ♕×d5 16 0-0 ♖d8 17 b4 ♕f5 18 ♖fe1 ♘f6 19 ♔g2 ♗h5 20 b5 ♘a5 21 ♘g1 ♖c8 22 ♔h1 ♖d8 23 ♔g2 ♖c8 24 ♔h1 ♖d8 25 ♔g2 ½-½

**CSC – Awit** 1 d4 ♘f6 2 c4 g6 3 ♘c3 d5 4 ♘f3 ♗g7 5 ♕b3 d×c4 6 ♕×c4 0-0 7 e4 ♘c6 8 e5 ♘d7 9 ♘d5 ♘b6 10 ♘×b6 a×b6 11 ♕c3 ♗g4 12 ♗e2 ♗×f3 13 ♗×f3 ♘×d4 14 ♗×b7 ♖a7 15 ♗e4 ♗×e5 16 ♗h6 ♖e8 17 ♕c4 ♕d6 18 f4 b5 19 ♕d3 f5 20 f×e5 f×e4 21 ♕×e4 ♕b4+ 22 ♔f1 ♕×b2 23 ♖b1 ♕×a2 24 ♖c1 ♖d8 25 ♖g1 c6 26 ♔e1 ♕b2 27 ♖b1 ♕c3+ 28 ♔f1 ♖a2 29 ♖c1 ♖f8+ 30 ♗×f8 ♕×c1+ 31 ♕e1 ♕f4+ 32 ♕f2 ♕×f2# 0-1

**Advance 1.0 – Dark Horse** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 ♘f6 4 0-0 ♗e7 5 ♖e1 ♗d6 6 d4 ♘×d4 7 ♘×d4 a6 8 ♗c4 b5 9 ♗d5 ♘×d5 10 e×d5 0-0 11 ♘f5 ♗b7 12 ♗h6 g×h6 13 ♕g4+ 1-0

**Duchess – Belle** 1 e4 e5 2 ♘f3 ♘c6 3 ♗c4 ♘f6 4 d4 e×d4 5 0-0 ♘×e4 6 ♖e1 d5 7 ♗×d5 ♕×d5 8 ♘c3 ♕a5 9 ♘×e4 ♗e6 10 ♗d2 ♗b4 11 ♘×d4 ♘×d4 12 c3 ♗e7 13 c×d4 ♕d5 14 ♖c1 c6 15 ♗g5 ♗×g5 16 ♖c5 ♕×a2 17 ♘×g5 0-0-0 18 ♘×e6 f×e6 19 ♕d2 ♖d5 20 ♖cc1 ♖b5 21 ♖e2 ♖d8 22 ♕c3 ♖d6 23 ♕c2 g6 24 ♖d1 ♖bd5 25 ♖e4 e5 26 ♖e3 ♖×d4 27 ♖de1 ♖d2 28 ♕c5 ♕×b2 29 ♖×e5 ♕b6 30 ♖e8+ ♖d8 31 ♖×d8+ ♖×d8 32 ♕e7 ♕a5 33 ♖b1 ♖d7 34 ♕e3 ♕b6 35 ♕e8+ ♕d8 36 ♕e6 ♕e7 37 ♕g8+ ♖d8 38 ♕c4 h5 39 f4 ♖e8 40 ♖a1 ♕e3+ 41 ♔f1 a6 42 ♖d1 g5 43 f5 ♕e5 44 ♕d3 ♖f8 45 ♕d7+ ♔b8 46 ♔g1 ♕c5+ 47 ♔h1 ♕×f5 48 ♕d6+ ♔c8 49 h3 g4 50 h×g4 h×g4 51 ♖e1 ♖d8 52 ♕h6 ♕f2 0-1

*Playoff Game, September 29th, 1980*

The following are notes written by David Levy while the game was in progress.

| | **Belle** | **Chaos** |
|---|---|---|
| **1** | **e4** | **♘f6** |
| **2** | **e5** | **♘d5** |
| **3** | **d4** | **d6** |
| **4** | **♘f3** | **d×e5** |
| **5** | **♘xe5** | **g6** |
| **6** | **g3** | |

More usual is 6 ♗f1-c4. The text move is rarely played and now CHAOS was out of its openings book.

| | | |
|---|---|---|
| **6** | **...** | **♗f5** |
| **7** | **c4** | **♘b4** |

Black must be careful. If 7 . . . ♗f5-e4?, White wins a piece with 8 f2-f3.

| | | |
|---|---|---|
| **8** | **♕a4+** | **♘4c6** |

CHAOS had probably been under the same illusion as I, but in fact it could have played 8 . . . ♘b8-c6 quite safely, as 9 ♘e5×c6 ♘b4×c6 10 d4-d5 can be met by 10 . . . ♗f5-d7 11 d5×c6 ♗d7×c6, forking White's queen and rook.

Another possibility after 8 . . . ♘b8-c6 is 9 d4-d5 ♗f5-c2!, when 10 b2-b3 ♗f8-g7 11 ♗c1-b2 can be met by 11 . . . 0-0, and if 12 d5×c6?? ♕d8-d1 is mate! White may also try 10 ♕a4-b5, but then 10 . . . a7-a6 11 ♕b5×b7 ♘c6×e5 12 ♕b7×b4 ♘e5-f3+ 13 ♔e1-e2 ♘f3-d4+ gives Black good play for the pawn.

The conclusion is that Black's 7th move is probably playable.

| | | |
|---|---|---|
| **9** | **d5** | **♗c2** |
| **10** | **♕b5** | **♕d6?** |

In view of Black's difficulties it would have been better to try 10 . . . a7-a6 11 ♕b5×b7 ♘c6×e5 12 ♕b7×a8 ♘e5-f3+ 13 ♔e1-e2 ♘f3-d4+, when it is not clear who stands better.

| | | |
|---|---|---|
| **11** | **♘xc6** | **♘xc6** |

And here Black could try 11 . . . b7×c6 12 ♕b5-b7 ♕d6-e5+ 13 ♔e1-d2 ♗c2-f5. The complications following 14 ♕b7×a8 ♗f8-h6+ are extremely unclear.

| | | |
|---|---|---|
| **12** | **♘c3** | |

White cannot yet capture on c6 because of

mate on d1.

**12 ... ♗g7**
**13 ♕xb7 0-0**
**14 ♕xc6 ♕b4**
**15 ♔d2 ♗e4**
**16 ♖g1 ♖fb8**
**17 ♗h3 ♗h6+**
**18 f4**

Black's counter threats have now been repulsed and White can play to consolidate its material advantage.

**18 ... ♕a5**
**19 ♖e1 f5**
**20 ♕e6+ ♔f8**
**21 b3 ♗g7**
**22 ♗b2 ♗d4**
**23 g4 ♖b6**
**24 ♕d7 ♖d6**
**25 ♕a4 ♕b6**

Black's only hope is to keep the queens on and play for an attack against the white king.

**26 ♗a3 ♗xc3+**
**27 ♔xc3 ♖dd8**
**28 ♖ad1 ♕f2**

Although White is a piece ahead his task is not easy. The exposed king gives rise to all sorts of tactical possibilities.

**29 gxf5 ♕c2+**
**30 ♔d4 gxf5**
**31 ♕c6 ♕f2+**
**32 ♔e5 ♔g8**
**33 ♖g1+**

Black's King now comes under attack.

**33 ... ♔h8**
**34 ♗xe7**

With this move BELLE announced that it had found a forced mate. The threat is 35 ♗e7-f6 or 35 ♕c6-f6.

**34 ... ♕g2**
**35 ♕f6+ ♔g8**
**36 ♗xg2 ♖xd5+**
**37 ♔e6 h6**
**38 ♕xh6 ♖c5+**
**39 fxe5 ♖f8**
**40 ♗f3 mate**
**1-0**

---

**4th World Computer Championship, New York, U.S.A. October 22nd - 25th 1983**

| | | rate | perf | 1 | 2 | 3 | 4 | 5 | total |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Cray Blitz (USA) | 0 | 2418 | 16+ | 10+ | 8+ | 4= | 6+ | 4½ |
| 2 | Bebe (USA) | 1900 | 2072 | 11– | 18+ | 7+ | 10+ | 4+ | 4 |
| 3 | Awit (Canada) | 0 | 1854 | 14+ | 8– | 12+ | 18+ | 13+ | 4 |
| 4 | Nuchess (USA) | 2150 | 2192 | 18+ | 11+ | 6+ | 1= | 2– | 3½ |
| 5 | Chaos (USA) | 1850 | 1957 | 7= | 21+ | 17+ | 6– | 8+ | 3½ |
| 6 | Belle (USA) | 2203 | 2087 | 13+ | 9+ | 4– | 5+ | 1– | 3 |
| 7 | Schach 2.7 (West Germany) | 0 | 1860 | 5= | 12+ | 2– | 11+ | 10= | 3 |
| 8 | Advance 3.0 (England) | 0 | 1920 | 22+ | 3+ | 1– | 9+ | 5– | 3 |
| 9 | Mephisto X (West Germany) | 0 | 1712 | 19+ | 6– | 15+ | 8– | 14+ | 3 |
| 10 | Fidelity X (USA) | 1850 | 1775 | 15+ | 1– | 19+ | 2– | 7= | 2½ |
| 11 | Merlin (Austria) | 0 | 1791 | 2+ | 4– | 13= | 7– | 17+ | 2½ |
| 12 | Novag X (USA) | 0 | 1419 | 20= | 7– | 3– | 21+ | 16+ | 3½ |
| 13 | Phoenix (Canada) | 0 | 1652 | 6– | 22+ | 11= | 20+ | 3– | 2½ |
| 14 | Ostrich (Canada) | 0 | 1348 | 3– | 15– | 18+ | 19+ | 9– | 2 |
| 15 | Pion (Netherlands) | 0 | 1349 | 10– | 14+ | 9– | 16– | 20+ | 2 |
| 16 | BCP (England) | 0 | 1260 | 1– | 19– | 22+ | 15+ | 12– | 2 |
| 17 | Patsoc 2.0 (USA) | 0 | 1291 | 21= | 20+ | 5– | 3– | 11– | 1½ |
| 18 | Philidor X (England) | 0 | 1196 | 4– | 2– | 14– | 22+ | 19= | 1½ |
| 19 | Conchess X (Sweden) | 0 | 1247 | 9– | 16+ | 10– | 14– | 18= | 1½ |
| 20 | Bobby (USA) | 0 | 1186 | 12= | 17– | 21+ | 13– | 15– | 1½ |
| 21 | Shy (Finland) | 0 | 1118 | 17= | 5– | 20– | 12– | 22+ | 1½ |
| 22 | Sfinks (USA) | 1000 | 776 | 8– | 13– | 16– | 18– | 21– | 0 |

*Round 1, October 22, 1983*

**Phoenix – Belle** 1 d4 d5 2 ♗g5 f6 3 ♗f4 ♘c6 4 ♘f3 g5 5 ♗g3 h5 6 ♕d3 h4 7 ♕g6+ ♔d7 8 ♘xh4 gxh4 9 ♕g4+ e6 10 ♗xh4 f5 11 ♗xd8 fxg4 12 ♗g5 ♘xd4 13 ♔d1 ♗d6 14 e3 ♖h5 15 h4 gxh3 16 ♖xh3 ♖xh3 17 gxh3 ♘f3 18 h4 ♗e5 19 ♘c3 ♗xc3 20 bxc3 e5 21 ♗e2 ♘g1 22 f4 exf4 23 ♗g4+ ♔d6 24 ♗xf4+ ♔c6 25 ♗xc8 ♖xc8 26 ♖b1 ♘e7 27 ♖b4 ♘h3 28 ♗e5 ♘f5 29 h5 ♘xe3+ 30 ♔c1 ♘c4 0-1

**Cray Blitz – BCP** 1 e4 c5 2 ♘f3 ♘f6 3 e5 ♘d5

4 ♘c3 e6 5 ♘×d5 e×d5 6 d4 ♘c6 7 d×c5 ♗×c5 8 ♕×d5 ♕b6 9 ♕d2 0-0 10 ♗c4 ♖e8 11 0-0 ♘×e5 12 ♘×e5 ♖×e5 13 ♕f4 ♕f6 14 ♕×f6 g×f6 15 ♔h1 d5 16 f4 ♖h5 17 ♗e2 ♖h4 18 ♗f3 d4 19 g3 ♖h3 20 f5 ♔g7 21 ♔g2 ♖h6 22 ♗×h6+ ♔×h6 23 ♗d5 ♔g7 24 ♖ad1 a5 25 ♔h1 ♖a6 26 ♗e4 b5 27 ♖fe1 ♗d7 28 ♖d2 ♗c6 29 ♗×c6 ♖×c6 30 ♖e8 ♗b6 31 ♖b8 b4 32 ♖b7 ♔f8 33 ♖e2 ♗c7 34 g4 ♖c5 35 ♖a7 ♗b6 36 ♖a6 ♖c6 37 ♖d2 ♖d6 38 ♖d3 ♔g7 39 c3 ♔g8 40 a4 ♔g7 41 c×b4 a×b4 42 a5 ♗c5 43 ♖×d6 ♗×d6 44 ♖×d4 ♗c5 45 ♖d5 ♗e3 46 ♖d3 ♗c5 47 ♖d7 ♗e3 48 a6 h5 49 g×h5 ♔f8 50 ♖d3 ♗c5 51 ♖g3 ♔e8 52 h6 ♗d6 53 a7 ♔e7 54 ♖d3 ♗c7 55 a8♕ ♗d6 56 h7 b3 57 ♕b7+ ♔e8 58 h8♕+ ♗f8 59 ♕e4# 1-0

**Schach 2.7 — Chaos** 1 d4 ♘f6 2 c4 e6 3 ♘f3 b6 4 g3 ♗b7 5 ♗g2 ♗e7 6 0-0 0-0 7 ♘c3 ♘e4 8 ♕c2 ♘×c3 9 ♕×c3 ♗e4 10 ♗f4 c6 11 ♖fd1 d5 12 c×d5 ♕×d5 13 ♘e1 ♗×g2 14 ♘×g2 ♘d7 15 f3 ♕b5 16 e4 c5 17 d5 e×d5 18 ♖×d5 ♘f6 19 ♖d3 ♖fd8 20 ♖ad1 ♖×d3 21 ♖×d3 c4 22 ♖d1 ♗c5+ 23 ♗e3 ♗b4 24 ♕c2 h6 25 h4 ♖c8 26 ♔f2 ♗c5 27 ♕e2 ♕a5 28 ♗×c5 ♕×c5+ 29 ♘e3 b5 30 ♕d2 b4 31 ♕e2 c3 32 ♕c2 a5 33 ♖e1 ♖d8 34 ♖c1 ♕d4 35 b×c3 ♕d2+ 36 ♕×d2 ♖×d2+ 37 ♔e1 ♖×a2 38 c×b4 a×b4 39 ♘d5 b3 40 ♘×f6+ g×f6 41 ♖b1 b2 42 ♔d2 ♔g7 43 ♔c2 ♖a3 44 ♖×b2 ♖×f3 45 ♖b3 ♖×b3 46 ♔×b3 ♔g6 47 ♔c4 ♔h5 48 ♔c5 ♔g4 49 ♔d6 f5 50 e×f5 ♔×f5 51 ♔e7 ♔g6 ½-½

**Nuchess — Philidor X** 1 f4 d5 2 ♘f3 ♘f6 3 e3 g6 4 ♗e2 ♗g7 5 0-0 0-0 6 d3 c5 7 ♕e1 ♘c6 8 c3 b6 9 ♘a3 e6 10 ♘e5 ♘×e5 11 f×e5 ♘e8 12 d4 ♗d7 13 ♗d2 ♗a4 14 ♕g3 ♗c6 15 b4 ♖b8 16 ♘b5 ♗×b5 17 ♗×b5 ♕e7 18 ♖ab1 ♕c7 19 b×c5 b×c5 20 ♗d3 ♖b6 21 ♗c2 h6 22 ♕h4 g5 23 ♕h5 ♕c6 24 h4 g×h4 25 ♕×h4 ♖×b1 26 ♖×b1 ♕a6 27 a4 ♕c4 28 ♕e1 ♕a6 29 ♗c1 ♕c4 30 ♗a3 ♕a2 31 ♕c1 a6 32 ♗×c5 ♕×a4 33 ♕d2 ♘d6 34 e×d6 1-0

**Merlin — Bebe** 1 e4 c5 2 c3 d5 3 e×d5 ♕×d5 4 d4 e6 5 ♘f3 ♘c6 6 ♗e2 ♘f6 7 d×c5 ♗×c5 8 ♕×d5 ♘×d5 9 ♘bd2 0-0 10 0-0 ♖d8 11 ♘e4 ♗e7 12 ♖e1 e5 13 ♗d2 ♘c7 14 ♗g5 ♗×g5 15 ♘e×g5 h6 16 ♘e4 ♗f5 17 ♘c5 b6 18 ♘b7 ♖d7 19 ♖ad1 ♖b8 20 ♖×d7 ♗×d7 21 ♘d6 ♗e6 22 ♗c4 ♖d8 23 ♗×e6 ♘×e6 24 ♖d1 ♔h7 25 ♔f1 ♔g6 26 ♘h4+ ♔h5 27 ♘hf5 f6 28 f3 ♔g6 29 h4 ♘f4 30 g3 ♘e6 31 ♔f2 a5 32 ♔e3 ♖d7 33 ♔e4 ♘cd8 34 ♖h1 h5 35 ♖h2 ♘c7 36 ♖f2 ♘de6 37 ♖d2 ♘c5+ 38 ♔e3 a4 39 c4 ♘7e6 40 ♘c8 ♖×d2 41 ♔×d2 ♘d7 42 ♘fe7+ ♔f7 43 ♘d5 ♘d4 44 ♘c×b6 ♘×f3+ 45 ♔e3 ♘×b6 46 ♘×b6 a3 47 b×a3 ♘d4 48 a4 ♔e6 49 a5 ♘c2+ 50 ♔e2 ♘b4 51 a3 ♘a6 52 ♘d5 ♘c5 53 ♘b4 ♔d6 54 ♔e3 ♔c7 55 ♘d3 ♘b3 56 a6 ♔b6 57 ♔e4 ♘d2+ 58 ♔f5 ♘×c4 59 ♔g6 ♔×a6 60 ♔×g7 f5 61 ♔g6 e4 62 ♘b4+ ♔a5 63 ♔×f5 e3 64 ♔f4 e2 65 ♘d3 ♘b2 66 ♘e1 ♔a4 67 ♔g5 ♔×a3 68 g4 h×g4 69 ♔×g4 ♘a4 70 h5 ♘b6 71 h6 ♘d7 72 ♔f3 ♘f8 73 ♔×e2 ♘h7 74 ♔e3 ♔b3 75 ♔f4 ♔b2 76 ♔f5 ♔c3 77 ♘f3 ♔d3 1-0

**Advance 3.0 — Sfinks X** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗a4 ♘f6 5 0-0 ♗e7 6 ♖e1 b5 7 ♗b3 0-0 8 c3 d5 9 e×d5 ♘×d5 10 a4 b×a4 11 ♗×a4 ♗b7 12 ♗×c6 ♗×c6 13 ♘×e5 ♗b7 14 d4 ♕d6 15 ♘d2 ♕e6 16 ♘df3 ♖ae8 17 c4 ♘b4 18 ♕b3 c5 19 ♗d2 ♕c8 20 d×c5 ♘c6 21 ♘×c6 ♕×c6 22 ♕b6 ♗×c5 23 ♕×c6 ♖×e1+ 24 ♘×e1 ♗×c6 25 b4 ♗e7 26 ♗c3 ♗b7 27 b5 ♖a8 28 ♖d1 ♖e8 29 h3 a×b5 30 c×b5 ♗d8 31 ♖d7 ♗c8 32 ♖d6 ♗f6 33 ♗×f6 ♖×e1+ 34 ♔h2 g×f6 35 ♖d8+ ♔g7 36 ♖×c8 ♖b1 37 ♖b8 ♖b2 38 ♔g3 ♖b3+ 39 f3 ♔g6 40 b6 ♔f5 41 ♖b7 h5 42 ♔h4 ♔g6 43 f4 ♔f5 44 g3 ♔e6 45 ♖c7 ♖×b6 46 ♔×h5 ♖b5+ 47 ♔h4 ♖d5 48 ♖a7 ♖d8 49 ♖a6+ ♖d6 50 ♖×d6+ ♔×d6 51 ♔g4 ♔e6 52 h4 f5+ 53 ♔g5 ♔e7 54 ♔h6 ♔f8 55 ♔h7 f6 56 ♔g6 ♔g8 57 ♔×f6 ♔h7 58 ♔×f5 1-0

**Pion — Fidelity X** 1 ♘f3 ♘f6 2 g3 g6 3 ♗g2 ♗g7 4 c4 0-0 5 d4 d5 6 c×d5 ♘×d5 7 0-0 ♘c6 8 ♖e1 ♘b6 9 e3 ♕d6 10 ♕e2 ♗g4 11 ♖d1 e5 12 d×e5 ♕×e5 13 h3 ♗×f3 14 ♗×f3 ♘a4 15 ♗×c6 b×c6 16 ♕c2 ♘×b2 17 ♖d4 c5 18 ♖e4 ♕d5 19 ♗×b2 ♗×b2 20 ♕×b2 ♕×e4 21 ♘c3 ♕e5 22 ♖c1 ♖fb8 23 ♕e2 ♖b4 24 a3 ♖b3 25 ♘a4 ♖×a3 26 ♘×c5 a5 27 ♕f3 ♖d8 28 ♘e4 f5 29 ♘c5 ♖a1 30 ♖×a1 ♕×a1+ 31 ♔g2 ♕e5 32 ♕c6 ♕d5+ 33 ♕×d5+ ♖×d5 34 ♘e6 a4 35 ♘×c7 a3 36 ♘×d5 a2 37 ♘f6+ ♔g7 38 ♘e8+ ♔f7 39 ♘d6+ ♔e7 40 ♘c8+ ♔e6 41 ♘b6 a1♕ 42 e4 ♕b1 43 e×f5+ ♔×f5 44 ♘d7 ♕b7+ 0-1

**Mephisto X — Conchess X** 1 e4 c5 2 ♘f3 a6 3 ♗c4 ♘f6 4 ♘c3 d6 5 d4 c×d4 6 ♘×d4 ♕c7 7 ♕d3 ♘bd7 8 f4 e6 9 0-0 b5 10 ♗b3 b4 11 ♘a4 ♗b7 12 ♖e1 ♘×e4 13 ♖×e4 ♗×e4 14 ♕×e4 ♖c8 15 ♗d2 ♘f6 16 ♕e2 ♕b7 17 ♗c4 ♖a8 18 f5 e5 19 ♘f3 ♗e7 20 ♗e3 e4 21 ♘g5 d5 22 ♗b3 h6 23 ♘h3 0-0 24 ♖d1 ♖ad8 25 ♘f4 ♗d6 26 ♘h5 ♘×h5 27 ♕×h5 ♗e5 28 f6 ♗×f6 29 ♗×d5 ♕b5 30 b3 ♖d7 31 ♘c5 ♖e7 32 ♕g4 ♗g5 33 ♗d4 ♖d8 34 ♗c4 ♕b6 35 c3 b×c3 36 ♘a4 ♕c6 37 ♗×c3 ♗e3+ 38 ♔h1 ♖×d1+ 39 ♕×d1 ♗f4 40 ♕d8+ 1-0

**Awit — Ostrich** 1 c4 e5 2 ♘c3 d6 3 ♘f3 f5 4 d3 ♗e7 5 g3 ♘c6 6 ♗g2 ♘f6 7 0-0 0-0 8 ♗d2 e4 9 ♘h4 ♘d7 10 d×e4 ♗×h4 11 g×h4 ♕×h4 12 e×f5 ♖×f5 13 f4 ♘e7 14 e3 ♘b6 15 ♗e1 ♕f6 16 ♘b5 ♘×c4 17 ♘×c7 ♖b8 18 ♕a4 ♕f7 19 ♕×a7 ♘c6 20 ♗×c6 ♕×c7 21 e4 ♖f6 22 ♗c3 ♖g6+ 23 ♔f2 ♔h8 24 ♖g1 ♗g4 25 f5 ♖f8 26 ♕×b7 27 ♗×b7 ♗h5 28 a4 ♘e5 29 a5 ♖×g1 30 ♖×g1 g6 31 ♔e3 g×f5 32 ♖g5 f4+ 33 ♔f2 ♗g6 34 ♗×e5+ d×e5 35 a6 ♖d8 36 a7 ♗×e4 37 ♗×e4 ♖d2+ 38 ♔f3 ♖d8 39 ♖×e5 h6 40 b4 ♔g7 41 b5 ♔g8 42 b6 ♖c8 43 b7 ♖c3+ 44 ♔×f4 ♔g7 45 b8♕ 1-0

**Novag X — Bobby** 1 e4 e5 2 f4 ♘f6 3 f×e5 ♘×e4 4 ♘f3 ♘g5 5 d4 ♘×f3+ 6 ♕×f3 ♕h4+ 7 ♔f2 ♕×f2+ 8 ♔×f2 ♘c6 9 ♗e3 d6 10 ♗b5 d×e5 11 ♗×c6+ b×c6 12 d×e5 ♗f5 13 ♖c1 ♗e7 14 ♘c3 0-0 15 ♘e2 ♖fd8 16 ♘d4 ♗e4 17 e6 ♖ab8 18 e×f7+ ♔×f7 19 ♘b3 ♗f6 20 c3 a6 21 ♗f4 ♖b7 22 ♔g1 ♖e8 23 ♖f1 ♗d5 24 ♖f2 ♔g8 25 ♘c5 ♖a7 26 ♗g3 ♗g5 27 ♔h1 ♗e3 28 ♖e2 ♗c4 29 ♖ee1 ♗f7 30 ♘d3 ♗d2 31 ♖×e8+ ♗×e8 32 h3 ♗g6 33 ♖d1 ♗g5 34 h4 ♗h5 35 ♖e1 ♗f6 36 ♖e6 ♗f7 37 ♖×c6 ♗d5 38 ♖×c7 ♖×c7 39 ♗×c7 ♗×a2 40 g3 ♗d5+ 41 ♔g1 ♔f7 42 ♘c5 ♗c4 43 ♘d7 ♗d5 44 ♘×f6 ♔×f6 45 b4 ♔f5 46 ♔f2 ♔e4 ½-½

**Shy – Patsoc 2.0** 1 e4 e5 2 ♗c4 ♘f6 3 ♘f3 ♘c6 4 d4 e×d4 5 0-0 ♘×e4 6 ♖e1 d5 7 ♗×d5 ♕×d5 8 ♘c3 ♕h5 9 ♘×e4 ♗e6 10 ♘×d4 ♕×d1 11 ♖×d1 ♘×d4 12 ♖×d4 ♖d8 13 ♖×d8+ ♔×d8 14 ♗e3 b6 15 ♘g5 ♔d7 16 ♘×e6 ♔×e6 17 ♖d1 ♗d6 18 ♖e1 ♔d5 19 ♖d1+ ♔c6 20 ♖d4 ♗e5 21 ♖c4+ ♔d6 22 c3 ♖e8 23 ♖a4 a5 24 ♖h4 h6 25 ♖h5 a4 26 ♗d2 ♔e6 27 f4 ♗d6 28 ♔h1 f5 29 c4 ♔f6 30 ♖h4 ♖e4 31 b3 ♔g6 32 ♗c3 a×b3 33 a×b3 ♖e3 34 ♖h3 ♖×h3 35 g×h3 ♗×f4 36 h4 h5 37 b4 ♗d6 38 b5 ♔f7 39 ♔g2 ♗e7 40 ♗e5 ♗c5 41 ♗×c7 ♔e6 42 ♗f4 g6 43 ♔f3 ♔f7 44 ♔g2 ♗d4 45 ♔g3 ♔e7 46 ♔f3 ♔d7 47 ♔g3 ♔c8 48 ♗d2 ♔c7 49 ♔f4 ♗f6 50 ♔g3 ♗e5+ 51 ♔g2 ♗d6 52 ♗g5 ♔b7 53 ♗e3 ½-½

*Round 2, October 22, 1983*

**Belle – Mephisto X** 1 e4 c5 2 c3 e6 3 d4 d5 4 e×d5 e×d5 5 ♘f3 ♗d6 6 d×c5 ♗×c5 7 ♗e2 ♘f6 8 0-0 ♗f5 9 ♘d4 ♕c8 10 ♗b5+ ♘c6 11 ♖e1+ ♔f8 12 ♘×f5 ♕×f5 13 ♗e3 ♗×e3 14 ♖×e3 ♖e8 15 ♖×e8+ ♘×e8 16 ♘a3 a6 17 ♗d3 ♕g5 18 ♕b3 b5 19 ♘c2 ♘c7 20 a4 ♘e5 21 ♗e2 ♘c4 22 a×b5 a×b5 23 ♘d4 ♔g8 24 ♗×c4 d×c4 25 ♕×b5 ♕×b5 26 ♘×b5 ♘e8 27 ♘d6 ♘c7 28 ♖a7 ♘e6 29 ♘×f7 g6 30 ♘×h8 ♔×h8 31 ♖a4 ♔g7 1-0

**Fidelity X – Cray Blitz** 1 c4 e5 2 ♘c3 ♘c6 3 ♘f3 f5 4 d4 e4 5 ♘g5 ♗b4 6 d5 ♗×c3+ 7 b×c3 ♘a5 8 ♕d4 ♕f6 9 c5 ♕×d4 10 c×d4 ♘f6 11 d6 c×d6 12 c×d6 ♘c6 13 e3 ♘b4 14 ♔e2 b6 15 ♗d2 ♗a6+ 16 ♔d1 ♘d3 17 ♗×d3 e×d3 18 f3 0-0 19 ♗e1 ♖fe8 20 ♗d2 ♖ac8 21 ♖c1 ♖×c1+ 22 ♔×c1 ♖c8+ 23 ♔b1 ♖c2 24 ♖d1 ♗c4 25 a3 h6 26 ♘h3 ♗a2+ 27 ♔a1 ♗b3 28 ♔b1 ♖a2 29 ♗c1 ♖×g2 30 ♖d2 ♖×d2 31 ♗×d2 ♗d5 32 ♘f2 ♗×f3 33 ♘×d3 ♗e4 34 ♔c2 ♘g4 35 ♔c3 ♗×d3 36 ♔×d3 ♘×h2 37 ♗e1 g5 38 ♔e2 ♔f7 39 ♗g3 ♘g4 40 ♔f3 ♘f6 41 ♗e5 h5 42 ♔g2 ♘g4 43 ♔f3 ♘×e5+ 44 d×e5 b5 45 ♔f2 a5 46 e4 f4 47 ♔e2 g4 48 ♔e1 h4 49 ♔f2 h3 50 ♔g1 g3 51 ♔f1 f3 0-1

**Merlin – Nuchess** 1 e4 e5 2 ♘f3 ♘f6 3 ♘×e5 d6 4 ♘f3 ♘×e4 5 ♕e2 ♕e7 6 d3 ♘f6 7 ♗g5 ♘bd7 8 ♘c3 ♕×e2+ 9 ♗×e2 h6 10 ♗d2 g6 11 0-0 ♘b6 12 ♘b5 ♘fd5 13 c4 c6 14 ♘bd4 ♗e7 15 ♗f4 ♘f5 16 ♖fe1 ♗e7 17 ♘×f5 ♗×f5 18 ♘d4 ♗d7 19 ♗f3 ♔f8 20 h3 ♖e8 21 a4 ♘c8 22 ♘b3 g5 23 ♗h2 ♗f6 24 ♘a5 ♗×b2 25 ♖×e8+ ♔×e8 26 ♖e1+ ♗e6 27 ♘×b7 ♔d7 28 c5 d5 29 ♖b1 ♗c3 30 ♘d6 ♘×d6 31 ♗×d6 a5 32 ♖b7+ ♔d8 33 ♖a7 ♖e8 34 ♗c7+ ♔c8 35 ♗×a5 ♔b8 36 ♖a6 ♗×a5 37 ♖×a5 ♔b7 38 g4 ♗d7 39 ♔f1 h5 40 ♔g2 h×g4 41 ♗×g4 ♗×g4 42 h×g4 ♖e2 43 ♔f3 ♖e1 44 ♔g2 ♖d1 45 ♔f3 ♖×d3+ 46 ♔e2 ♖h3 47 ♔d2 ♖h2 48 ♔e1 ♖g2 49 f3 ♖c2 50 ♔d1 ♖f2 51 ♔c1 0-1

**Awit – Advance 3.0** 1 c4 e5 2 ♘c3 d6 3 ♘f3 ♘c6 4 d4 e×d4 5 ♘×d4 ♘×d4 6 ♕×d4 ♘f6 7 ♗f4 ♗e6 8 0-0-0 ♗e7 9 ♘d5 ♗×d5 10 c×d5 ♔d7 11 e4 c5 12 ♕e3 ♕a4 13 e5 d×e5 14 ♕×e5 ♘g4 15 ♕c7 ♘×f2 16 ♖e1 0-0 17 ♖g1 ♗d8 18 ♕d6 ♗a5 19 ♗e2 ♖fe8 20 ♗e5 ♗d2+ 21 ♔×d2 ♘e4+ 22 ♔c1 ♘×d6 23 ♗×d6 ♕×a2 24 ♗f3 ♕a1+ 25 ♔d2 ♕×b2+ 26 ♔d3 c4+ 27 ♔×c4 ♖ac8+ 28 ♗c5 ♕c2+ 29 ♔b4 ♖e3 30 ♖×e3 ♕×c5+ 31 ♔a4 ♕d4+ 0-1

**Chaos – Shy** 1 d4 ♘f6 2 c4 e5 3 d×e5 ♘g4 4 ♗f4 ♘c6 5 ♘f3 ♗b4+ 6 ♘bd2 ♕e7 7 a3 ♘g×e5 8 ♘×e5 ♘×e5 9 e3 ♗×d2+ 10 ♕×d2 0-0 11 ♕d5 d6 12 c5 ♖e8 13 c×d6 ♕×d6 14 ♕×d6 c×d6 15 ♖d1 ♗e6 16 ♖×d6 ♗b3 17 ♗b5 ♖ec8 18 0-0 ♘g6 19 ♗g3 ♖c2 20 ♖d7 ♖×b2 21 ♖×b7 ♗e6 22 ♖d1 ♗d5 23 ♖d7 ♖×b5 24 ♖7×d5 ♖×d5 25 ♖×d5 ♘e7 26 ♖d7 ♖c8 27 h3 ♘c6 28 f3 a5 29 ♖d5 ♖a8 30 e4 ♘e7 31 ♖d7 ♘c6 32 ♔f2 ♖e8 33 ♔e3 ♘b8 34 ♖a7 ♘c6 35 ♖a6 ♘e7 36 ♖×a5 ♘c6 37 ♖c5 ♘e7 38 a4 ♖a8 39 ♖c7 ♖e8 40 ♔d2 ♘g6 41 ♖b7 ♖a8 42 ♔e3 ♖a6 43 ♖b8+ ♘f8 44 ♖d8 ♖×a4 45 ♗d6 g6 46 ♗×f8 h5 47 ♗c5+ ♔h7 48 ♔f4 ♖a5 49 ♖c8 ♔h6 50 ♖c7 ♔g7 51 e5 ♖a4+ 52 ♔e3 ♖a6 53 ♔d4 h4 54 f4 ♖a4+ 55 ♔e3 ♖a1 56 ♔f3 ♔g8 57 ♗e7 ♖f1+ 58 ♔g4 ♖f2 59 ♖c8+ ♔h7 60 ♗f6 ♖×g2+ 61 ♔×h4 1-0

**Patsoc 2.0 – Bobby** 1 d4 ♘f6 2 ♘c3 e6 3 ♘f3 ♗b4 4 a3 ♗×c3+ 5 b×c3 ♘e4 6 c4 c5 7 e3 ♕a5+ 8 ♗d2 ♘×d2 9 ♕×d2 ♕b6 10 ♕c3 c×d4 11 e×d4 ♘c6 12 c5 ♕d8 13 ♗d3 ♘e7 14 ♕b2 ♕a5+ 15 ♔f1 ♘d5 16 ♔g1 f5 17 ♖e1 0-0 18 ♗b5 ♘f4 19 g3 ♘g6 20 ♔g2 f4 21 c3 ♕c7 22 ♕b1 f×g3 23 h×g3 b6 24 ♕e4 ♖b8 25 c6 d5 26 ♕e3 ♖f5 27 ♘g5 ♕e7 28 ♘×h7 e5 29 d×e5 ♖×e5 30 ♕c1 ♗f5 31 ♖×e5 ♘×e5 32 ♘g5 ♖c8 33 ♖e1 ♕f6 34 ♕f4 ♘g6 35 ♕d2 ♕d6 36 ♗c4 ♘e7 37 ♖×e7 1-0

**Schach 2.7 – Novag X** 1 d4 ♘f6 2 c4 e6 3 ♘f3 b6 4 g3 ♗a6 5 ♗d2 ♗e7 6 ♗g2 c6 7 0-0 d5 8 ♘e5 ♘fd7 9 c×d5 c×d5 10 ♘×d7 ♘×d7 11 ♘c3 ♖c8 12 ♖c1 0-0 13 h3 ♘f6 14 a4 ♕d7 15 ♗g5 ♖c6 16 ♗×f6 ♗×f6 17 ♕d2 ♖fc8 18 ♘a2 ♖×c1 19 ♖×c1 ♖×c1+ 20 ♖×c1 ♕d6 21 ♕c3 e5 22 d×e5 ♗×e5 23 ♕f3 ♕c5 24 ♘d3 ♗×d3 25 ♕×d3 d4 26 ♕e4 a5 27 ♕a8+ ♔f8 28 ♕c6 ♕d8 29 ♗e4 g6 30 ♗d5 ♔g7 31 ♕b5 h6 32 ♔f1 h5 33 ♗×f7 ♕f6 34 ♕e8 ♕f5 35 ♗×g6 ♕×h3+ 36 ♔e1 ♗×g3 37 ♕f7+ ♔h6 38 ♗d3 ♗×f2+ 39 ♕×f2 ♕e6 40 ♕×d4 ♔g5 41 e3 ♕f6 42 ♗b5 h4 43 ♗d7 ♕×d4 44 e×d4 ♔f4 45 ♗e6 1-0

**Bebe – Philidor X** 1 e4 c5 2 ♘f3 d6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 e6 6 ♗e2 a6 7 0-0 ♕c7 8 ♕d3 ♘bd7 9 ♕c4 ♕d8 10 ♗g5 ♘c5 11 ♘f3 ♗e7 12 e5 d×e5 13 ♘×e5 ♗d6 14 ♕d4 ♗×e5 15 ♕×e5 ♘cd7 16 ♕d6 ♕a5 17 ♘e4 ♕f5 18 ♗d3 ♕g4 19 f3 ♕h5 20 h4 ♘e5 21 ♗×f6 ♘c6 22 ♗×g7 1-0

**Ostrich – Pion** 1 e4 d6 2 d3 g6 3 ♗e2 ♘f6 4 ♘f3 ♗g7 5 0-0 0-0 6 ♘c3 ♖e8 7 ♗g5 c5 8 e5 d×e5 9 ♘×e5 ♘d5 10 d4 ♘×c3 11 b×c3 c×d4 12 c×d4 f6 13 ♗c4+ e6 14 ♗h4 g5 15 ♗×g5 f×g5 16 ♗b5 ♗d7 17 ♘×d7 ♘×d7 18 ♕g4 ♖c8 19 ♕e4 a6 20 ♗d3 ♘f8 21 ♕×b7 a5 22 ♖ad1 ♗×d4 23 ♕e4 ♕b6 24 h4 g×h4 25 ♕g4+ ♗g7 26 ♕×h4 ♕b2 27 ♕a4 ♕b6 28 ♗b5 ♖ed8 29 ♖c1 ♗b2 30 ♖cd1 ♖×d1 31 ♕g4+ ♗g7 32 ♕×d1 ♖c3 33 ♕e2 ♔f7 34 ♗d3 ♔f6 35 ♕h5 ♕b4 36 g4 e5 37 g5+ ♔e7 38 f3 ♕a4 39 ♗×h7 ♕e8 40 ♕h4 ♘×h7 41 ♕×h7 ♕f8 42 ♕e4 ♔e6 43 ♖f2 ♕f5 44 ♕a8 ♕×g5+ 45 ♖g2 ♕c1+ 46 ♔h2 ♕f4+ 47 ♔g1 ♕d4+ 48 ♔h2 49 ♕h4+ 49 ♔g1 ♕e1+ 50 ♔h2 ♖c7 51 ♕e8+ ♖e7 52 ♕c6+ ♔f7 53 ♕g6+ ♔f8 54 ♕d3 ♔e8 55 ♕h7 ♗f8 56 ♕h5+ ♖f7

57 f4 e4 58 ♖e2 ♕b4 59 ♕g6 ♕c3 60 ♕×e4+ ♖e7 61 ♕g6+ ♔d7 62 ♕g4+ ♔d8 63 ♖×e7 ♔×e7 64 ♕g5+ ♔e8 65 ♕f5 ♗g7 66 ♕e4+ ♔d7 67 ♔g2 ♗d4 68 f5 ♔d6 0-1

**Sfinks X – Phoenix** 1 e4 e6 2 d4 d5 3 e×d5 e×d5 4 ♗d3 ♘c6 5 ♘f3 ♗g4 6 ♗e3 ♘b4 7 ♗e2 ♗f5 8 ♘a3 ♘c6 9 0-0 ♗×a3 10 b×a3 ♘f6 11 ♖b1 ♖b8 12 ♗d3 ♗×d3 13 ♕×d3 0-0 14 c4 ♖e8 15 ♖fe1 d×c4 16 ♕×c4 ♕d5 17 ♕×d5 ♘×d5 18 ♗d2 ♖×e1+ 19 ♖×e1 ♘ce7 20 ♗c1 ♖e8 21 ♗b2 b6 22 ♔f1 f6 23 ♔e2 ♘f4+ 24 ♔f1 ♘d3 25 ♖e2 ♘×b2 26 ♖×b2 c6 27 ♔e2 ♘d5+ 28 ♔f1 b5 29 h3 ♖e7 30 ♔g1 ♖e6 31 ♔h2 ♔f7 32 ♔g3 ♖e4 33 ♖c2 ♘e7 34 ♔h2 ♔e8 35 ♔g1 ♔d7 36 ♔f1 ♘d5 37 ♔g1 ♔c7 38 ♔h2 ♔b6 39 ♔g3 ♘f4 40 ♖d2 ♔a5 41 ♖c2 ♘e2+ 42 ♔h2 ♔b6 43 ♖d2 ♘c3 44 ♖b2 ♖e2 45 ♘d2 ♔c7 46 f3 ♘d1 0-1

**BCP – Conchess X** 1 e4 e6 2 d4 d5 3 e5 c5 4 c3 ♘c6 5 ♘f3 ♕b6 6 ♗d3 c×d4 7 c×d4 ♘×d4 8 ♘×d4 ♗c5 9 ♕a4+ ♗d7 10 ♗b5 0-0-0 11 ♗×d7+ ♖×d7 12 ♘f3 ♗×f2+ 13 ♔e2 ♘e7 14 ♘c3 ♘c6 15 ♕g4 d4 16 ♕×g7 d3+ 17 ♔f1 ♖hd8 18 ♕×h7 ♔b8 19 ♘d1 ♗c5 20 ♗h6 f6 21 ♗g7 d2 22 g4 ♕a6+ 23 ♔g2 ♕e2+ 24 ♔g3 ♖d3 25 ♘e3 ♖×e3 26 ♖hf1 d1♕ 27 ♖ax d1 ♖×d1 28 ♕h8+ ♔c7 29 e×f6 ♖×f3+ 30 ♖×f3 ♗d6+ 31 ♔h4 ♕×f3 32 ♕e8 ♕f2+ 33 ♔h5 ♕×h2+ 34 ♔g6 ♘e5+ 35 ♔g5 ♕d2+ 36 ♔h4 ♘f3+ 37 ♔h3 ♖h1# 0-1

*Round 3, October 23, 1983*

**Nuchess – Belle** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗a4 ♘f6 5 d4 e×d4 6 0-0 ♗e7 7 e5 ♘e4 8 ♘×d4 0-0 9 ♘f5 d5 10 e×d6 ♗×f5 11 d×e7 ♘×e7 12 ♗e3 ♘d5 13 ♕f3 ♘×e3 14 f×e3 ♗g6 15 ♕f4 b5 16 ♗b3 c5 17 c4 ♕f6 18 ♕×f6 ♘×f6 19 ♖c1 b4 20 ♘d2 ♖fe8 21 ♖e1 ♖ad8 22 ♘f1 ♗d3 23 ♖ad1 ♘g4 24 ♗a4 ♖f8 25 ♘d2 ♘e5 26 ♗b3 ♖d6 27 ♘f3 ♘×f3+ 28 g×f3 f5 29 ♖d2 ♖e8 30 ♔f2 f4 31 e×f4 ♖×e1 32 ♔×e1 ♖d4 33 ♔f2 ♔f7 34 ♔e3 ♗×c4 35 ♖×d4 c×d4+ 36 ♔×d4 ♗×b3 37 a×b3 ♔f6 38 ♔e4 g6 39 h4 ♔e6 40 f5+ g×f5+ 41 ♔d4 ♔d6 42 f4 ♔e6 43 ♔c5 a5 44 h5 ♔f7 45 ♔d5 ♔f6 46 ♔d6 ♔f7 47 ♔e5 ♔e8 48 ♔×f5 1-0

**Cray Blitz – Advance 3.0** 1 e4 c5 2 ♘f3 d6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 a6 6 ♗g5 e6 7 f4 ♕b6 8 ♘b3 ♕e3+ 9 ♕e2 ♕×e2+ 10 ♗×e2 ♗e7 11 0-0 0-0 12 ♖ad1 ♖d8 13 e5 ♘bd7 14 e×f6 g×f6 15 ♗h6 ♔h8 16 ♗h5 ♔g8 17 ♖f3 ♗f8 18 ♖g3+ ♔h8 19 ♘e4 ♗×h6 20 ♘×d6 ♖f8 21 ♘×f7+ ♖×f7 22 ♗×f7 ♗g7 23 ♗×e6 ♘f8 24 ♖d8 ♗×e6 25 ♖×a8 ♗g8 26 ♖b8 f5 27 c3 b5 28 ♘c5 ♘g6 29 ♖e3 ♗f8 30 ♘×a6 ♗×a2 31 b4 ♔g7 32 g3 ♗c4 33 ♘c7 ♗e7 34 ♘e6+ ♔f7 35 ♘d4 ♗d6 36 ♖b7+ ♔g8 37 ♘×f5 ♗d3 38 ♘×d6 1-0

**Patsoc 2.0 – Chaos** 1 d4 ♘f6 2 ♘c3 d5 3 ♗g5 ♗f5 4 f3 c6 5 g4 ♗d7 6 e3 e6 7 ♕d3 h6 8 ♗h4 c5 9 ♘ge2 ♘c6 10 a3 ♗e7 11 ♗g2 ♕b6 12 d×c5 ♗×c5 13 ♗×f6 g×f6 14 ♘a4 ♕a5+ 15 ♕c3 ♗×e3 16 ♕×a5 ♘×a5 17 ♘ac3 ♘c4 18 ♘d1 ♗c5 19 ♘f4 ♖c8 20 ♘h5 ♔e7 21 a4 f5 22 g×f5 e×f5 23 f4 ♖ce8 24 ♗f3 ♖hg8 25 b3 ♔d8+ 26 ♗e2 ♖g2 27 ♘g3 ♘e3 28 ♖c1 ♘g4 29 ♔f1 ♖×h2 30 ♖×h2 ♘×h2+ 0-1

**Bebe – Schach 2.7** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗a4 ♘f6 5 0-0 ♗e7 6 ♖e1 b5 7 ♗b3 d6 8 c3 0-0 9 h3 ♘b8 10 d3 ♘bd7 11 ♘bd2 ♗b7 12 ♘f1 ♘c5 13 ♗c2 a5 14 ♘g3 c6 15 d4 ♘cd7 16 ♘f5 ♖e8 17 a4 ♗f8 18 a×b5 c×b5 19 ♕e2 ♕b6 20 ♗e3 ♕a6 21 d5 ♖ab8 22 b4 a4 23 c4 ♗a8 24 c×b5 ♕×b5 25 ♕×b5 ♖×b5 26 ♖×a4 ♖c8 27 ♗d3 ♖bb8 28 ♖ea1 ♗b7 29 ♗b5 ♖c2 30 ♖a5 ♖c3 31 ♗d2 ♖c7 32 ♘g3 ♖bc8 33 ♖a7 ♘b8 34 ♗e3 ♗e7 35 ♗b6 ♖c1+ 36 ♖×c1 ♖×c1+ 37 ♔h2 ♗×d5 38 ♖×e7 ♗e6 39 ♗c7 ♘c6 40 ♗×c6 ♖×c6 41 b5 ♖c5 42 ♗×d6 ♖×b5 43 ♗×e5 ♖a5 44 ♗×f6 g×f6 45 ♖e8+ ♔g7 46 ♘d4 h5 47 ♘×e6+ f×e6 48 ♖×e6 ♔f7 49 ♖c6 ♔g6 50 f4 h4 1-0

**Conchess X – Fidelity X** 1 c4 c5 2 ♘c3 ♘c6 3 ♘f3 g6 4 d3 ♘d4 5 ♗f4 d6 6 ♘×d4 c×d4 7 ♘d5 e6 8 ♘b4 ♗g7 9 h3 ♗d7 10 ♕d2 e5 11 ♗g3 h5 12 ♘d5 ♘f6 13 ♗h4 ♗c6 14 ♗×f6 ♗×f6 15 ♖d1 ♗g5 16 ♕b4 a5 17 ♕b3 a4 18 ♕b4 a3 19 b3 ♖a7 20 ♕b6 ♕×b6 21 ♘×b6 ♗d8 22 c5 d×c5 23 ♘c4 b5 24 ♘d2 ♗a5 25 ♖c1 ♗c3 26 ♔d1 f5 27 ♔c2 ♖hh7 28 ♖d1 h4 29 ♖g1 ♗d5 30 ♘f3 e4 31 ♘g5 ♖h5 32 f4 c4 33 d×c4 b×c4 34 b×c4 ♗×c4 35 e3 ♗×f1 36 ♖g×f1 d×e3 37 ♔×c3 e2 38 ♖g1 e×d1♕ 39 ♖×d1 ♖b7 40 ♖d6 ♖b2 41 ♖d2 e3 42 ♖d3 e2 43 ♖e3+ ♔d7 44 ♘f7 ♖×a2 45 ♘e5+ ♔c7 46 ♘×g6 ♖b2 47 ♘e5 ♖h6 48 ♘c4 ♖c6 49 ♖e7+ ♔c8 50 ♖e5 ♖×c4+ 0-1

**Mephisto X – Pion** 1 e4 d6 2 ♘f3 ♘f6 3 ♘c3 g6 4 d4 ♗g7 5 ♗b5+ c6 6 ♗e2 0-0 7 0-0 ♘bd7 8 a4 ♖e8 9 ♘g5 ♕c7 10 f4 ♖d8 11 ♗c4 d5 12 e×d5 c×d5 13 ♗b3 ♘b6 14 ♘b5 ♕b8 15 ♕e2 ♖e8 16 ♖e1 ♗f5 17 a5 ♘bd7 18 ♗d2 ♗g4 19 ♕e3 ♕d8 20 h3 ♗f5 21 g4 a6 22 ♘c3 ♗×g4 23 h×g4 ♘×g4 24 ♕h3 ♗×d4+ 25 ♔g2 ♘df6 26 ♗×d5 ♖f8 27 ♘e6 ♘f2 28 ♘×d8 ♘×h3 29 ♗×f7+ ♔h8 30 ♔×h3 ♖a×d8 31 ♗b3 ♗c5 32 ♗e3 ♗b4 33 ♗b6 ♖de8 34 ♗d4 ♖d8 35 ♖ad1 h5 36 ♖e5 ♗×c3 37 b×c3 ♖d7 38 ♖de1 ♖c7 39 ♖×e7 ♖×e7 40 ♖×e7 b6 41 a×b6 g5 42 b7 g4+ 43 ♔h4 a5 44 ♗×f6+ ♖×f6 45 b8♖+ ♖f8 46 ♖×f8# 1-0

**Phoenix – Merlin** 1 d4 d5 2 ♗g5 ♘f6 3 ♗×f6 e×f6 4 e3 ♗f5 5 c4 ♗b4+ 6 ♘c3 ♘c6 7 ♕b3 ♕e7 8 ♕d1 0-0-0 9 c×d5 ♖×d5 10 ♘ge2 ♖hd8 11 ♖c1 ♖b5 12 ♘g3 ♖×d4 13 ♗×b5 ♖×d1+ 14 ♖×d1 ♗×c3+ 15 b×c3 ♗g6 16 ♗×c6 b×c6 17 ♖d2 ♕a3 18 ♘e2 ♗e4 19 0-0 ♗d5 20 ♖c1 ♔b7 21 c4 ♗e6 22 ♖b1+ ♔a6 23 ♘d4 ♗×c4 24 ♘×c6 ♕a4 25 ♘b8+ ♔a5 26 ♖b7 a6 27 ♖×c7 ♕b5 28 ♖d1 ♗×a2 29 ♖c8 ♔a4 30 ♖c2 ♔b3 31 ♖dd2 ♕f5 32 ♘c6 ♕g6 33 f3 ♗b1 34 ♖b2+ ♔c4 35 ♘e7 ♕h6 36 f4 ♗e4 37 ♖d6 ♕h4 38 ♖d4+ ♔c3 39 ♖e2 ♕g4 40 ♖ed2 ♕e6 41 ♘d5+ ♗×d5 42 ♖2d3+ ♔b2 43 ♖×d5 ♔c2 44 ♔f2 h6 45 ♖d6 ♕g4 46 ♖d2+ ♔b3 47 ♖×a6 ♕h4+ 48 ♔f3 ♕×h2 49 ♖b6+ ♔c3 50 ♖d5 ♔c4 51 e4 ♕g1 52 ♖c6+ ♔b4 53 ♖c7 ♕f1+ 54 ♔e3 ♕e1+ 55 ♔f3 ♕f1+ 56 ♔e3 ½-½

**Novag X – Awit** 1 d4 ♘f6 2 c4 g6 3 ♘c3 d5 4 c×d5 ♘×d5 5 e4 ♘×c3 6 b×c3 c5 7 ♘f3 ♗g7 8 ♗c4 ♘c6

9 ♗d5 c×d4 10 c×d4 0-0 11 ♗e3 ♕a5+ 12 ♗d2 ♕a6 13 ♗c3 ♗g4 14 h3 ♗×f3 15 g×f3 e5 16 d×e5 ♘×e5 17 ♗×e5 ♗×e5 18 ♖c1 ♗f4 19 ♖c5 b6 20 ♖c2 ♖ac8 21 ♖g1 ♖×c2 22 ♕×c2 ♖c8 23 ♗c6 b5 24 ♔f1 ♖×c6 25 ♕b2 ♖c1+ 26 ♔g2 ♖×g1+ 27 ♔×g1 f6 28 ♕b3+ ♔g7 29 ♕d5 ♕a3 30 ♔g2 a6 31 ♕d7+ ♔h6 32 ♕f7 ♕b2 33 ♕f8+ ♔h5 34 ♕f7 a5 35 ♕×h7+ ♗h6 36 ♕d7 ♗d2 37 f4 ♔h6 38 ♕d8 ♗×f4 39 ♔f3 ♔g5 40 ♕×a5 ♕e5 41 ♕b6 ♕c3+ 42 ♔g2 ♕c4 43 ♔g1 ♕×e4 44 ♕×b5+ ♔h4 45 ♕d7 f5 46 ♕h7+ ♔g5 47 ♕f7 ♗e5 48 a3 ♕f4 49 ♕e7+ ♗f6 50 ♕e1 ♔h4 51 ♔g2 ♕a4 52 ♕b4+ ♕×b4 53 a×b4 ♗d4 54 b5 ♗c5 55 f4 ♗d4 56 ♔h2 ♗e3 57 ♔g2 ♗×f4 58 b6 g5 59 b7 ♗b8 0-1

**Bobby — Shy** 1 e4 e5 2 ♘c3 ♘f6 3 f4 d5 4 f×e5 ♘×e4 5 ♘f3 ♗e7 6 d4 0-0 7 ♗d3 f5 8 e×f6 ♗×f6 9 0-0 ♘c6 10 ♘×e4 d×e4 11 ♗×e4 ♗×d4+ 12 ♘×d4 ♕×d4+ 13 ♕×d4 ♖×f1+ 14 ♔×f1 ♘×d4 15 ♗f4 c6 16 ♖d1 ♘e6 17 ♗e3 ♔h8 18 ♔e2 ♘f8 19 ♖d6 ♗e6 20 b3 ♗g4+ 21 ♔d3 ♘d7 22 ♗d4 ♘f8 23 h3 ♗e6 24 g4 ♗f7 25 a3 ♘e6 26 ♖d7 ♘×d4 27 ♔×d4 ♗g6 28 ♗×g6 h×g6 29 ♖×b7 a5 30 ♔c5 a4 31 b4 ♖f8 32 ♔×c6 ♖f3 33 ♔b5 ♖×a3 34 ♖c7 ♖a1 35 c4 a3 36 ♖a7 g5 37 ♔c5 ♔h7 38 b5 a2 39 b6 ♔h6 40 b7 ♖b1 41 ♔c6 a1♕ 42 ♖×a1 ♖×a1 43 b8♕ ♔h7 44 c5 ♖a3 45 ♕h2 ♔g8 46 ♔d5 ♖d3+ 47 ♔c4 ♖d1 48 c6 ♖c1+ 49 ♔d5 ♖d1+ 50 ♔e6 ♖d4 51 ♕e5 ♖c4 52 c7 ♖c1 53 ♔f5 1-0

**Philidor X — Ostrich** 1 e4 d5 2 e×d5 ♘f6 3 d4 ♘×d5 4 ♘f3 ♗g4 5 c4 ♘b6 6 c5 ♘d5 7 ♕b3 ♗c8 8 ♗b5+ c6 9 ♗d3 e6 10 0-0 b6 11 ♗g5 f6 12 c×b6 ♕×b6 13 ♕×b6 a×b6 14 ♗d2 ♗a6 15 ♗×a6 ♘×a6 16 ♖e1 ♘ac7 17 ♘c3 ♘×c3 18 b×c3 ♗d6 19 ♖ab1 b5 20 ♖b2 0-0 21 ♖e4 ♖fe8 22 ♖e1 ♘d5 23 ♖a1 e5 24 d×e5 ♗×e5 25 ♖b3 ♖e7 26 a3 ♖f7 27 ♘×e5 f×e5 28 ♖e1 ♖af8 29 f3 ♖f5 30 ♖e4 ♘f6 31 ♖eb4 e4 32 f4 ♖e8 33 c4 b×c4 34 ♖×c4 ♖b5 35 ♗b4 e3 36 a4 ♖×b4 37 ♖b×b4 e2 38 ♖b1 e1♕+ 39 ♖×e1 ♖×e1+ 40 ♔f2 ♖e4 41 ♖×e4 ♘×e4+ 42 ♔f3 ♘c5 43 a5 ♔f7 44 h4 ♔e6 45 ♔f2 ♔d5 46 g4 ♔c4 47 ♔e2 ♔b5 48 ♔f2 ♔b4 49 ♔e2 h6 50 ♔d2 ♘b3+ 51 ♔c2 ♘×a5 52 ♔b2 ♘c4+ 53 ♔c2 0-1

**Sfinks — BCP** 1 e4 c5 2 ♘f3 ♘f6 3 ♘c3 d5 4 e×d5 ♘×d5 5 ♘×d5 ♕×d5 6 c3 e5 7 d4 c×d4 8 c×d4 ♗b4+ 9 ♗d2 ♗d6 10 d×e5 ♗×e5 11 ♘×e5 ♕×e5+ 12 ♕e2 ♘c6 13 ♕×e5+ ♘×e5 14 ♗c3 f6 15 ♖c1 0-0 16 f4 ♘c6 17 ♗c4+ ♔h8 18 ♔f2 ♗f5 19 ♖hd1 ♖fd8 20 ♔f3 h5 21 ♔g3 a6 22 ♔h4 ♗e4 23 ♔×h5 ♗×g2 24 ♖×d8+ ♖×d8 25 ♔g6 ♗e4+ 26 ♔f7 ♔h7 27 h3 ♔h6 28 ♗e1 ♗h7 29 ♔e6 ♔g6 30 f5+ ♔h6 31 ♗d2+ ♖×d2 0-1

*Round 4, October 24, 1983*

**Nuchess — Cray Blitz** 1 f4 d5 2 ♘f3 ♘f6 3 e3 ♗g4 4 b3 ♘bd7 5 ♗b2 e6 6 ♗d3 ♗d6 7 h3 ♗×f3 8 ♕×f3 e5 9 ♗e2 0-0 10 0-0 e×f4 11 e×f4 ♖e8 12 ♘c3 c6 13 ♕d3 ♘c5 14 ♕f3 d4 15 ♘b1 ♖e4 16 g3 ♕b6 17 ♕f2 ♘a4 18 ♗a3 ♗×a3 19 ♘×a3 ♘c5 20 ♗f3 ♖e7 21 ♘c4 ♕d8 22 ♖fe1 ♘fe4 23 ♕g2 d3 24 c3 ♘d6 25 ♘e5 f6 26 ♘×c6 b×c6 27 ♗×c6 ♘cb7 28 ♕d5+ ♖f7 29 ♕f3 ♔h8 30 ♗d5 ♖d7 31 ♔g2 ♖b8 32 b4 f5 33 ♗e6 ♖e7 34 ♖e5 ♘f7 35 ♖e3 ♘fd6 36 ♗d5 ♘e4 37 ♖×d3 ♖d7 38 c4 ♘f6 39 ♖e1 ♘×d5 40 ♖×d5 ♖×d5 41 ♕×d5 ♕×d5+ 42 c×d5 ♘d6 43 ♖e6 ♘e4 44 ♖e7 ♖a8 45 d3 ♘c3 46 d6 ♘d5 47 ♖e5 ♘f6 48 ♖×f5 ♖d8 49 ♖a5 ♖d7 50 d4 ♘e4 51 ♔f3 ♘×d6 52 g4 ♔g8 53 ♖d5 ♔f8 54 a4 ♔e8 55 ♖h5 h6 56 ♖c5 ♖e7 57 d5 ♘e4 58 ♖c6 ♔d8 59 d6 ♘d2+ 60 ♔f2 ♖e4 61 ♖c7 ♖×b4 62 ♖×g7 ♖×a4 63 ♖h7 ♘e4+ 64 ♔f3 ♘×d6 65 ♖×h6 ♔e7 66 ♖h7+ ♔e6 67 f5+ ♔d5 68 h4 ♘c4 69 ♖d7+ ♔c5 70 ♖g7 ♘e5+ 71 ♔g3 a5 72 h5 ♔d6 73 h6 ♖a3+ 74 ♔g2 ♖a2+ 75 ♔g3 ♖a3+ 76 ♔g2 ♖a2+ 77 ♔g3 ♖a3+ ½-½

**Chaos — Belle** 1 d4 d5 2 c4 e6 3 ♘c3 c6 4 c×d5 e×d5 5 ♘f3 ♗d6 6 e4 d×e4 7 ♘×e4 ♗b4+ 8 ♗d2 ♗×d2+ 9 ♕×d2 ♘e7 10 ♗c4 b5 11 ♗e2 a6 12 0-0 0-0 13 ♖fc1 ♗e6 14 ♘fg5 ♗f5 15 ♘g3 ♗g6 16 ♖c3 ♖a7 17 ♖ac1 ♖e8 18 ♗f3 ♖d7 19 ♖d1 ♘f5 20 ♘×f5 ♗×f5 21 g4 ♗e6 22 ♘×e6 ♖×e6 23 ♖e3 ♖×e3 24 f×e3 c5 25 ♕c3 c×d4 26 e×d4 ♖c7 27 ♕a5 ♕c8 28 d5 ♖c2 29 b4 ♘d7 30 d6 ♕c3 31 ♕d8+ ♘f8 32 ♕a8 ♕e3+ 33 ♔h1 ♕f4 34 ♗g2 ♕×g4 35 ♖f1 ♕e6 36 ♕×a6 ♖×a2 37 ♕b7 ♖d2 38 ♗h3 ♕c4 39 ♕e7 ♕c6+ 40 ♔g1 f6 41 d7 ♕b6+ 42 ♔h1 ♕b8 43 ♗g2 ♖×d7 44 ♕e2 ♖d4 45 ♕a2+ ♔h8 46 ♕b3 ♕d6 47 ♖b1 f5 48 ♕f3 f4 49 ♕b3 ♖d3 50 ♕c2 f3 51 ♗×f3 0-1

**Advance 3.0 — Mephisto X** 1 e4 c5 2 ♘f3 e6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 ♘c6 6 ♘×c6 b×c6 7 e5 ♘d5 8 ♘e4 ♕c7 9 f4 ♕b6 10 ♗d3 ♗e7 11 c4 ♗b4+ 12 ♔e2 ♘e7 13 a3 ♗c5 14 ♘×c5 ♕×c5 15 ♗e3 ♕a5 16 b4 ♕d8 17 ♗e4 0-0 18 ♕d6 ♘f5 19 ♗×f5 e×f5 20 ♕d3 g6 21 ♖ad1 ♕h4 22 ♗c5 ♖d8 23 ♕g3 ♕h5+ 24 ♔f2 ♗a6 25 ♖d4 ♕h6 26 a4 ♗c8 27 b5 c×b5 28 a×b5 a6 29 ♕f3 ♖b8 30 b6 ♗b7 31 ♕g3 ♖dc8 32 ♗d6 ♖a8 33 ♖b1 ♕g7 34 ♔g1 f6 35 ♖bd1 ♖c6 36 ♕h3 ♗c8 37 ♕b3 ♗b7 38 c5+ ♕f7 39 ♕×f7+ ♔×f7 40 e×f6 ♗c8 41 ♗e7 ♖e6 42 ♖b4 ♖b8 43 ♗d6 ♖b7 44 ♗e5 ♖c6 45 ♗d6 ♔×f6 46 ♖a4 ♔g7 47 ♖×a6 h6 48 ♖da1 ♔f6 49 ♖a7 ♖×d6 50 ♖×b7 ♖c6 1-0

**Fidelity X — Bebe** 1 d4 ♘f6 2 c4 e5 3 d×e5 ♘g4 4 ♗f4 ♘c6 5 ♘f3 ♗c5 6 e3 ♕e7 7 ♕d5 ♗b4+ 8 ♘c3 ♗×c3+ 9 b×c3 ♕a3 10 ♕d2 ♕a5 11 ♖b1 a6 12 ♗d3 ♘g×e5 13 ♘×e5 ♘×e5 14 ♗×e5 ♕×e5 15 ♕c2 h5 16 0-0 0-0 17 ♗h7+ ♔h8 18 ♗e4 ♖a7 19 f4 ♕c5 20 ♕d3 ♖e8 21 f5 ♕e7 22 ♖f4 ♔g8 23 ♕d5 ♕a3 24 ♕d2 d6 25 ♖b3 ♕a5 26 ♖b2 ♖e5 27 ♕d3 ♕a3 28 ♖ff2 b6 29 ♖b1 ♕c5 30 f6 ♗e6 31 ♖b4 b5 32 ♗h7+ ♔h8 33 f×g7+ ♔×g7 34 c×b5 ♖×e3 35 ♕d4+ ♕×d4 36 c×d4 ♔×h7 37 b×a6 ♖×a6 38 a4 ♖a3 39 ♖b7 ♖6×a4 40 ♖×c7 ♖×d4 41 ♖b7 ♖d1+ 42 ♖f1 ♖×f1+ 43 ♔×f1 ♖a1+ 44 ♔f2 ♖a2+ 45 ♔f1 ♗c4+ 46 ♔e1 ♖×g2 47 ♖c7 ♗e6 48 h4 ♖h2 49 ♖a7 ♖×h4 50 ♔f2 ♖c4 51 ♖a6 d5 0-1

**Awit — Patsoc 2.0** 1 c4 e5 2 ♘c3 ♘f6 3 g3 ♗b4 4 ♗g2 0-0 5 ♘h3 ♘c6 6 b3 d6 7 0-0 ♗c5 8 ♗b2 ♗g4 9 ♘a4 ♘d4 10 ♖e1 h6 11 ♘×c5 d×c5 12 f4 ♗×h3 13 ♗×h3 e×f4 14 g×f4 ♘e4 15 e3 ♘e6 16 d3 ♘f6 17 ♗g2 c6 18 a3 ♕e7 19 ♕d2 ♘h5 20 f5 ♘g5 21 b4 ♖ae8 22 ♖ad1 a6 23 d4 ♘e4 24 ♕e2 ♕g5 25 d×c5

♕×f5 26 a4 ♕g5 27 ♖d7 a5 28 ♖×b7 a×b4 29 a5 ♕×c5 30 a6 ♘d6 31 ♗d4 ♕×d4 32 e×d4 ♖×e2 33 ♖×e2 ♘×b7 34 a×b7 c5 35 d×c5 ♖d8 36 ♖e1 ♘f4 37 c6 ♔h7 38 c7 ♖d2 39 c8♕ ♖×g2+ 40 ♔h1 ♖g5 41 b8♕ b3 42 ♕×f4 ♖g6 43 ♕×f7 h5 44 ♕cf5 ♔h8 45 ♕5×g6 h4 46 ♖e8# 1-0

**Bobby – Phoenix** 1 e4 e6 2 c3 d5 3 e×d5 e×d5 4 d4 ♗d6 5 ♗d3 ♘f6 6 ♘e2 0-0 7 ♘a3 ♗×a3 8 b×a3 c5 9 d×c5 ♘bd7 10 ♗e3 ♕a5 11 ♗f5 ♘×c5 12 ♗×c5 ♕×c5 13 ♗×c8 ♖a×c8 14 ♕b3 ♖fe8 15 ♖d1 ♕×c3+ 16 ♔f1 ♕e5 17 ♘g3 ♖c3 18 ♕a4 ♕e7 19 ♕b4 ♖×a3 20 ♕b2 ♖d3 21 ♕c2 ♖×d1+ 22 ♕×d1 ♕e5 23 a4 d4 24 f3 d3 25 ♔f2 ♘g4+ 26 f×g4 ♕f4+ 27 ♔g1 d2 0-1

**Merlin – Schach 2.7** 1 e4 e5 2 ♘f3 ♘c6 3 ♗c4 ♗c5 4 c3 ♘f6 5 d3 d6 6 0-0 0-0 7 ♘bd2 h6 8 b4 ♗b6 9 ♕b3 a5 10 b5 a4 11 ♕c2 ♘e7 12 ♖b1 ♖e8 13 a3 d5 14 ♗a2 ♘g6 15 e×d5 ♘×d5 16 ♖e1 ♘df4 17 d4 e×d4 18 ♖×e8+ ♕×e8 19 c×d4 ♘h4 20 ♘e5 ♘e2+ 21 ♔f1 ♗f5 22 ♕c4 ♗×b1 23 ♘×b1 ♘×c1 24 ♕×c1 ♕×b5+ 25 ♗c4 ♕a5 26 ♕c2 ♗×d4 27 ♘×f7 ♔f8 28 h3 ♕c5 29 g3 b5 30 g×h4 ♕×c4+ 31 ♕×c4 b×c4 32 ♘h8 b8 33 ♘g6+ ♔e8 34 ♘c3 ♗×c3 35 ♘f4 c5 36 ♘d5 ♗b2 37 ♘e3 c3 38 ♘c2 ♖d8 39 ♔e1 ♖d2 40 ♘e3 ♗×a3 41 h5 ♗b4 42 ♔f1 c2 43 ♘×c2 ♖×c2 0-1

**Ostrich – Conchess X** 1 e4 c5 2 c3 ♘f6 3 e5 ♘d5 4 ♗c4 ♘b6 5 ♗b3 c4 6 ♗c2 d6 7 ♘f3 d×e5 8 ♘×e5 ♕d5 9 ♕e2 ♕×g2 10 ♗e4 ♕g5 11 ♘×c4 ♘×c4 12 ♕×c4 ♘d7 13 ♘a3 ♕e5 14 0-0 ♕g5+ 15 ♗g2 e6 16 ♘b5 ♘e5 17 ♘c7+ ♔d8 18 ♕e2 ♔×c7 19 d4 ♕g4 20 ♕×e5+ ♗d6 21 ♕a5+ ♔b8 22 ♗e3 ♗d7 23 h3 ♕g6 24 c4 ♖c8 25 ♖ac1 ♗c6 26 d5 e×d5 27 c×d5 b6 28 ♕d2 ♗b5 29 ♖×c8+ ♔×c8 30 ♖c1+ ♔d7 31 ♕d4 ♖e8 32 a4 ♗e2 33 ♖c6 ♕b1+ 34 ♗c1 f6 35 h4 ♖e7 36 a5 ♕g6 37 a×b6 ♗d3 38 ♗f4 ♗×f4 39 ♕×f4 a×b6 40 ♕b8 ♖e1+ 41 ♔h2 ♗a6 42 ♖c7+ ♔d6 43 ♖c8+ ♔e7 44 ♕c7# 1-0

**Pion – BCP** 1 ♘f3 d5 2 g3 ♘f6 3 ♗g2 c5 4 0-0 ♘c6 5 d3 e5 6 a3 ♗d6 7 c4 0-0 8 ♘c3 ♗e6 9 ♘g5 d4 10 ♘×e6 f×e6 11 ♘a4 ♔h8 12 b3 ♕d7 13 ♗g5 ♘g4 14 e4 ♘h6 15 ♗×h6 g×h6 16 ♕d2 ♕g7 17 ♖ae1 ♖f7 18 ♕e2 ♖g8 19 ♖b1 b6 20 ♕d2 h5 21 ♗h3 ♕f6 22 ♗g2 ♖g6 23 ♕e2 ♕g5 24 f4 e×f4 25 b4 ♖fg7 26 b×c5 b×c5 27 e5 ♕×e5 28 ♕d2 f×g3 29 ♔h1 ♖c7 30 ♖fe1 ♕f6 31 h×g3 ♖×g3 32 ♘b2 ♕h4+ 33 ♔g1 ♖cg7 34 ♔f1 ♖f7+ 35 ♔g1 ♖g6 36 ♖e5 ♗×e5 37 ♕f4 ♕×f4 38 ♖e1 ♕g3 39 ♖e2 ♕h2# 0-1

**Shy – Novag X** 1 e4 e5 2 ♗c4 ♘f6 3 ♘f3 ♘×e4 4 ♘c3 ♘×c3 5 d×c3 f6 6 ♕d5 ♕e7 7 ♕d1 d6 8 ♘d2 ♗e6 9 ♗×e6 ♕×e6 10 c4 ♗e7 11 ♕h5+ g6 12 ♕f3 ♘c6 13 0-0 0-0 14 ♖b1 f5 15 ♖d1 e4 16 ♕b3 b6 17 ♕a4 ♘e5 18 ♖e1 c6 19 ♕a6 ♗g5 20 ♘×e4 ♗×c1 21 ♖b×c1 f×e4 22 ♖×e4 ♕f5 23 ♖e2 ♘f3+ 24 ♔f1 ♘×h2+ 25 ♔e1 ♖ae8 26 ♔d1 ♖×e2 0-1

**Philidor X – Sfinks X** 1 e4 e6 2 d4 d5 3 ♘d2 ♘c6 4 ♘gf3 ♗b4 5 e5 f6 6 ♗b5 ♗d7 7 ♗d3 f×e5 8 d×e5 ♗c5 9 0-0 ♘ge7 10 ♘g5 ♘f5 11 ♘b3 ♗e7 12 ♗×f5 e×f5 13 e6 0-0 14 e×d7 ♗×g5 15 ♕×d5+ ♖f7 16 ♘c5 ♗×c1 17 ♖a×c1 ♕e7 18 ♖fe1 ♕f8 19 ♘×b7 ♘e7 20 d8♕ ♘×d5 21 ♕×d5 ♖b8 22 ♘a5 ♖×b2 23 ♘c4 ♖×a2 24 ♘e5 1-0

*Round 5, October 25, 1983*

**Belle – Cray Blitz** 1 e4 c5 2 c3 d5 3 e×d5 ♕×d5 4 ♘f3 e6 5 d4 ♘f6 6 ♗d3 ♘c6 7 0-0 ♗e7 8 ♗e3 0-0 9 d×c5 ♖d8 10 ♘d4 ♗×c5 11 c4 ♕d6 12 ♘×c6 b×c6 13 ♗×c5 ♕×d3 14 ♕a4 ♘e4 15 ♗b6 ♖d7 16 ♗a5 ♗b7 17 ♘c3 ♘c5 18 ♕b4 ♕d4 19 ♖ad1 ♘d3 20 ♕a4 ♕g4 21 c5 ♕f5 22 b4 ♘b2 23 ♖×d7 ♘×a4 24 ♘×a4 ♕c2 25 ♖×b7 ♕×a4 26 ♖a1 e5 27 f3 ♕c2 28 ♖c7 ♕d3 29 ♖f1 ♕d5 30 a3 g5 31 ♖e7 f6 32 ♖c7 h5 33 h3 ♔h8 34 ♔h2 a6 35 ♖e1 ♖e8 36 ♖e4 f5 37 ♖e2 g4 38 h×g4 f×g4 39 f×g4 h×g4 40 ♖f2 e4 41 ♖ff7 ♕e5+ 42 g3 e3 43 ♖h7+ ♔g8 44 b5 c×b5 45 ♗e1 ♕b2+ 46 ♔g1 ♕a1 47 ♔g2 ♕f6 48 ♔h2 ♖d8 49 ♖hd7 ♖f8 50 ♖d6 ♕b2+ 51 ♔g1 ♕b1 52 ♔h2 ♕c2+ 53 ♔g1 ♕f5 0-1

**Bebe – Nuchess** 1 e4 e5 2 ♘f3 ♘f6 3 d4 ♘×e4 4 ♗d3 d5 5 ♘×e5 ♗d6 6 0-0 0-0 7 ♗×e4 d×e4 8 ♘c3 ♕e7 9 ♕h5 g6 10 ♕e2 ♗×e5 11 d×e5 ♕×e5 12 ♕×e4 ♕×e4 13 ♘×e4 ♗f5 14 ♖e1 ♔g7 15 ♗f4 ♘a6 16 ♗e5+ ♔h6 17 ♖e2 ♖ae8 18 ♘f6 ♖e6 19 g4 ♗×c2 20 g5+ ♔g7 21 ♘d7+ ♖×e5 22 ♘×e5 ♗f5 23 ♖ae1 ♘b4 24 a3 ♘d5 25 ♖d2 c6 26 ♘c4 f6 27 ♘d6 f×g5 28 ♘×b7 ♗g4 29 b3 h5 30 ♘a5 ♗d7 31 ♘c4 ♗c8 32 b4 ♗b7 33 ♘a5 ♗a8 34 ♖c2 ♖f3 35 ♖e8 ♘b6 36 ♘c4 ♖f8 37 ♖×f8 ♔×f8 38 ♘e5 ♔g7 39 ♘×c6 a6 40 ♘b8 a5 41 ♖c7+ ♔h6 42 b×a5 ♘a4 43 ♘d7 ♗f3 44 a6 g4 45 a7 ♔g5 46 ♖c8 ♔h6 47 ♘e5 ♗d5 48 ♖d8 ♗b7 49 ♘f7+ ♔g7 50 ♘d6 ♗f3 51 ♖d7+ ♔g8 52 ♘c4 ♗c6 53 ♘e5 ♗e4 54 ♖d4 ♘b6 55 ♖×e4 ♘a8 56 ♖c4 ♔g7 57 ♖c8 ♘b6 58 ♖b8 ♘a8 59 ♖×a8 ♔f6 60 ♖e8 ♔g5 61 a8♕♔f5 62 ♕a6 ♔g5 63 ♕×g6+ ♔h4 64 ♘×g4 h×g4 65 ♖h8# 1-0

**Chaos – Advance 3.0** 1 d4 d5 2 c4 d×c4 3 ♘f3 ♘f6 4 e3 e6 5 ♗×c4 c5 6 ♕e2 a6 7 0-0 ♘c6 8 ♘c3 ♕c7 9 ♗d2 c×d4 10 e×d4 ♗e7 11 ♗b3 ♕d8 12 ♕e3 h6 13 ♖ac1 0-0 14 ♖fe1 ♘a5 15 ♗c2 ♘c4 16 ♕e2 b5 17 a4 ♗b7 18 a×b5 ♘×b2 19 ♖b1 ♗×f3 20 g×f3 ♗a3 21 b×a6 ♕×d4 22 ♘b5 ♕c5 23 ♘×a3 ♕×a3 24 ♗c1 ♖×a6 25 ♗×b2 ♕a5 26 ♗e5 ♖c6 27 ♖b5 ♕a6 28 ♗d3 ♘h5 29 ♖b8 ♕a3 30 ♗h7+ ♔×h7 31 ♕e4+ f5 32 ♕×c6 ♖×b8 33 ♗×b8 ♕b4 34 ♕×e6 ♕c5 35 ♗e5 ♕f8 36 ♖c1 ♘f4 37 ♗×f4 ♔h8 38 ♗e5 ♕a3 39 ♕×h6+ ♔g8 40 ♕×g7# 1-0

**Phoenix – Awit** 1 d4 ♘f6 2 ♗g5 ♘e4 3 ♗h4 d5 4 f3 ♘f6 5 ♘d2 g6 6 e4 ♘bd7 7 e5 ♘h5 8 c4 ♗h6 9 c×d5 ♘b6 10 g4 ♘f4 11 g5 ♗g7 12 ♘e4 ♘b×d5 13 ♗g3 h6 14 ♕d2 h×g5 15 ♘×g5 ♘h5 16 ♘1h3 f6 17 e×f6 e×f6 18 ♘e4 ♕e7 19 ♕e2 ♗×h3 20 ♗×h3 ♘hf4 21 ♘×f6+ ♗×f6 22 ♕×e7+ ♔×e7 23 ♗f1 ♗×d4 24 0-0-0 ♗e3+ 25 ♔b1 ♖ad8 26 ♗d3 c6 27 ♖he1 ♔f6 28 ♗e4 g5 29 ♖h1 ♖h3 30 ♖he1 ♘h5 31 ♗×d5 ♖×d5 32 ♖×d5 c×d5 33 ♖×e3 ♘×g3 34 h×g3 ♖×g3 35 ♖b3 g4 36 ♖×b7 ♖×f3 37 ♖×a7 g3 38 ♖a6+ ♔g5 39 ♖a8 ♖f1+ 40 ♔c2 ♖f2+ 41 ♔c3 d4+ 42 ♔×d4 ♖×b2 43 ♔c3 ♖b6 44 ♖d8 ♔f5 45 ♖d2 ♔e4 46 a4 ♖c6+ 47 ♔b4 ♔f3 48 ♖d3+ ♔f4 49 ♖d4+ ♔e3 50 ♖g4 ♔f2 51 ♖f4+ ♔e2 52 ♖f7 ♖c8 53 ♖e7+ ♔f2 54 ♖f7+ ♔g1 55 a5 ♖h8 56 a6 g2 57

a7 ♔h2 58 a8♕ 0-1

**Schach 2.7 – Fidelity X** 1 d4 d5 2 c4 d×c4 3 ♘f3 ♘f6 4 e3 e6 5 ♗×c4 c5 6 0-0 a6 7 ♕e2 b5 8 ♗b3 ♗b7 9 ♖d1 ♘bd7 10 ♘c3 ♕b8 11 d5 e×d5 12 ♘×d5 c4 13 ♘×f6+ ♘×f6 14 ♗c2 ♗c5 15 b3 c3 16 a3 0-0 17 ♖d3 ♘e4 18 ♘d4 ♗d6 19 ♕h5 ♖c8 20 f3 ♘f6 21 ♕h3 ♖c5 22 ♘f5 ♗×h2+ 23 ♕×h2 ♕×h2+ 24 ♔×h2 ♖×f5 25 ♖×c3 ♖h5+ 26 ♔g1 ♖d8 27 ♗b2 ♖d2 28 ♖d1 ♖×d1+ 29 ♗×d1 ♖d5 30 ♖c1 ♖d6 31 g3 h5 32 e4 ♘d7 33 ♗c3 f6 34 ♗b4 ♖d4 35 ♗e2 ♘e5 36 ♗c3 ♖d7 37 ♗×e5 f×e5 38 ♖c5 ♖e7 39 a4 b×a4 40 b×a4 ♔h7 41 ♗c4 g5 42 ♔f2 h4 43 ♔g2 h×g3 44 ♔×g3 ♔g7 45 ♔g4 ♔f6 46 a5 ♖e8 47 ♖c7 ♖e7 48 ♖×e7 ♔×e7 49 ♔×g5 ♔d6 50 ♗f1 ♔c5 51 ♔f5 ♔d6 52 ♔f6 ♗c8 53 ♗e2 ♗b7 54 ♗d3 ♗c8 55 ♗c4 ♗b7 56 ♗e6 ♗c6 ½-½

**Mephisto X – Ostrich** 1 e4 d5 2 e×d5 ♘f6 3 d4 ♘×d5 4 c4 ♘b6 5 ♘f3 ♗g4 6 ♗e2 ♘c6 7 d5 ♗×f3 8 ♗×f3 ♘e5 9 b3 g6 10 ♗b2 ♗g7 11 ♘c3 0-0 12 ♗e2 ♘ed7 13 f4 ♗×c3+ 14 ♗×c3 a5 15 ♕d2 a4 16 0-0 a×b3 17 a×b3 ♖×a1 18 ♖×a1 ♖e8 19 ♕b2 e6 20 d×e6 ♖×e6 21 ♖a7 ♕b8 22 ♖a5 ♖e4 23 ♗f3 ♖e3 24 ♕d2 ♖e7 25 ♕d4 f6 26 c5 ♘c8 27 ♗g4 b6 28 ♕d5+ ♔g7 29 ♖a8 c6 30 ♖×b8 c×d5 31 ♗×d7 ♖×d7 32 ♖×c8 b×c5 33 ♖×c5 d4 34 ♗d2 d3 35 ♔f2 ♖e7 36 ♔f3 ♖e2 37 ♗c1 ♖a2 38 ♖c7+ ♔g8 39 ♖d7 ♖c2 40 ♗e3 ♖c3 41 ♗d4 ♖×b3 42 ♗×f6 ♖a3 43 ♗e5 d2+ 44 ♔e2 d1♕+ 45 ♖×d1 h5 46 ♖d6 ♖a2+ 47 ♔f1 ♔f7 48 h4 ♖c2 49 ♖f6+ ♔e7 50 ♖×g6 ♔f7 1-0

**BCP – Novag X** 1 e4 e6 2 d4 d5 3 e5 c5 4 c3 ♘c6 5 ♘f3 ♕b6 6 ♗e2 ♗e7 7 0-0 ♖b8 8 ♘a3 c×d4 9 c×d4 ♗×a3 10 b×a3 ♘ge7 11 ♗g5 0-0 12 ♖b1 ♕a5 13 ♕b3 h6 14 ♗f4 ♘g6 15 ♗d2 ♕d8 16 ♕a4 ♘ge7 17 ♖fc1 ♗d7 18 ♔h1 ♘f5 19 ♗b5 a6 20 ♗×c6 ♗×c6 21 ♕b3 ♖c8 22 a4 ♕d7 23 ♕b4 ♖c7 24 ♖c5 b6 25 ♕×b6 ♖b7 26 ♖×c6 ♖×b6 27 ♖b×b6 ♘e7 28 ♖d6 ♕×a4 29 ♖×a6 ♕d1+ 30 ♗e1 ♘f5 31 ♖a3 ♘×d4 32 ♖e3 ♘c2 33 ♖b6 ♘×e3 34 f×e3 ♕c2 35 ♗b4 ♖a8 36 ♗d2 ♕d1+ 37 ♗e1 ♖×a2 38 ♖b8+ ♔h7 39 h4 ♖e2 40 ♔h2 ♖×e1 41 ♘×e1 ♕×e1 42 ♖b3 ♕×h4+ 43 ♔g1 ♔g6 44 ♖d3 ♕e1+ 45 ♔h2 h5 46 ♖b3 h4 47 ♖a3 ♔g5 48 ♖b3 ♕g3+ 49 ♔h1 d4 50 ♖b7 d×e3 51 ♖b2 ♕f2 52 ♖×f2 e×f2 53 ♔h2 f1♕ 0-1

**Patsoc 2.0 – Merlin** 1 d4 d5 2 ♘c3 ♘f6 3 ♗g5 ♗f5 4 f3 c6 5 g4 ♗g6 6 e3 h5 7 g×h5 ♖×h5 8 h4 ♕b6 9 ♘a4 ♕a5+ 10 c3 ♘h7 11 ♘h3 e5 12 ♕b3 b5 13 ♘c5 ♗×c5 14 d×c5 f6 15 ♗f4 e×f4 16 ♘×f4 ♖h6 17 ♘e6 ♘a6 18 h5 ♘g5 19 ♘×g7+ ♔f8 20 ♗g2 ♘×c5 21 ♕d1 ♘d3+ 22 ♔f1 ♘×b2 23 ♕c1 ♗d3+ 24 ♔g1 ♘a4 25 ♕d2 ♕×c3 26 ♖d1 ♗c2 27 ♕×c3 ♘×c3 28 ♖d2 ♗h7 29 f4 ♘ge4 30 ♘e6+ ♔e7 31 ♗×e4 ♗×e4 32 ♘d4 c5 33 ♖h3 c×d4 34 e×d4 b4 35 ♔f1 ♖ah8 36 a3 ♖×h5 37 ♖×h5 ♖×h5 38 a×b4 ♖h1+ 39 ♔f2 ♖h2+ 40 ♔e1 ♖h1+ 41 ♔f2 ♔e6 42 ♔g3 ♖b1 43 ♖h2 ♖×b4 44 f5+ ♗×f5 45 ♖h8 ♖×d4 46 ♖d8 ♖d3+ 0-1

**Pion – Bobby** 1 ♘f3 ♘f6 2 g3 e6 3 ♗g2 ♘c6 4 0-0 d5 5 d4 ♗d6 6 ♘c3 ♗d7 7 ♘b5 ♗e7 8 ♗f4 ♖c8 9 ♘g5 ♘h5 10 ♘h3 ♘×f4 11 ♘×f4 a6 12 ♘c3 g5 13 ♘h5 f5 14 e3 0-0 15 ♖c1 ♕e8 16 a3 ♔h8 17 ♖e1 ♘a5 18 b4 ♘c4 19 ♖a1 ♖a8 20 ♗f1 ♘b2 21 ♕c1 ♘a4 22 ♘×a4 ♗×a4 23 ♕d1 a5 24 b×a5 ♖×a5 25 ♗d3 ♗c6 26 c4 ♖×a3 27 ♖×a3 ♗×a3 28 ♕e2 d×c4 29 ♗×c4 b5 30 ♖a1 ♗d6 31 ♗b3 ♕g6 32 ♖c1 ♗e8 33 ♗d1 ♕h6 34 ♖b1 e5 35 d×e5 ♗×e5 36 f4 ♗×h5 37 ♕×h5 ♕b6 38 ♕e2 g×f4 39 ♖×b5 f×g3 40 ♖×b6 g×h2+ 41 ♔h1 ♖g8 42 ♖b8 ♖×b8 43 ♕h5 ♖g8 44 ♗g4 ♖a8 45 e4 c5 46 e×f5 c4 47 f6 ♖a1+ 48 ♗d1 ♗×f6 49 ♕d5 h6 50 ♔×h2 ♖a2+ 51 ♔g3 ♖a3+ 52 ♗f3 ♖c3 53 ♕f5 ♔g7 54 ♕g4+ ♔f7 55 ♕d7+ ♔f8 56 ♕d6+ ♔g7 57 ♕c5 ♗g5 58 ♕a7+ ♔g6 59 ♕b6+ ♗f6 60 ♕e6 ♔g7 61 ♕f5 ♗g5 62 ♕d7+ ♔f8 63 ♕d6+ ♔f7 64 ♔g4 1-0

**Conchess X – Philidor X** 1 e4 c5 2 ♘e2 ♘c6 3 d4 c×d4 4 ♘×d4 g6 5 c4 ♘f6 6 ♘×c6 d×c6 7 ♕×d8+ ♔×d8 8 ♗d3 e5 9 ♗g5 ♗g7 10 0-0 ♔c7 11 f4 e×f4 12 ♗×f4+ ♔d7 13 ♘c3 ♖e8 14 ♗g3 ♔e7 15 ♗e5 ♗e6 16 c5 ♖ad8 17 ♖ad1 ♘h5 18 ♗c7 ♗d4+ 19 ♔h1 ♖d7 20 ♗d6+ ♔d8 21 ♗b1 ♗×c3 22 b×c3 ♗c4 23 ♖f2 g5 24 ♖d4 ♗e6 25 ♗b8 a6 26 ♗d6 h6 27 ♗c2 f6 28 ♗d1 g4 29 ♗b3 ♔c8 30 ♖f1 ♖f7 31 ♗×e6+ ♖×e6 32 ♖b1 ♘g7 33 ♖c4 ♘e8 34 ♗f4 ♖fe7 35 ♖bb4 a5 36 ♖a4 h5 37 ♔g1 ♘g7 38 ♖d4 ♖d7 39 ♖d6 ♔d8 40 g3 ♖ee7 41 ♖×f6 ♖d1+ 42 ♔f2 ♖b1 43 ♖f8+ ♖e8 44 ♖×e8+ ♔×e8 45 ♖×a5 ♖b2+ 46 ♔g1 ♖b1+ 47 ♔f2 ♖b2+ 48 ♔e3 ♖×h2 49 ♖a8+ ♔e7 50 ♗g5+ ♔d7 51 ♖d8+ ♔c7 52 ♖d2 ♖h3 53 ♗f4+ ♔c8 54 ♖d6 ♖h2 55 ♖h6 ♔d8 56 ♗e5 ♘e8 57 ♖h7 ♔c8 58 a3 ♖a2 59 ♖×h5 ♖×a3 60 ♖h4 ♖a5 61 ♔d4 ♖a2 62 ♖×g4 ♖a4+ 63 c4 ♖a1 64 ♖g8 ♔d7 65 g4 ♖d1+ 66 ♔e3 ♖e1+ 67 ♔d3 ♖d1+ 68 ♔e2 ♖c1 69 ♔d3 ♖d1+ 70 ♔e3 ♖e1+ 71 ♔d3 ♖d1+ ½-½

**Sfinks X – Shy** 1 d4 ♘f6 2 c4 g6 3 ♘c3 d5 4 c×d5 ♘×d5 5 ♘×d5 ♕×d5 6 ♘f3 ♗g4 7 ♗f4 ♗g7 8 ♗×c7 ♗×f3 9 g×f3 ♕×d4 10 ♕×d4 ♗×d4 11 ♖d1 ♘c6 12 ♖d2 ♖c8 13 ♗f4 h6 14 ♗h3 ♖d8 15 ♗g2 e5 16 ♗g3 f5 17 e3 ♗c5 18 ♔e2 ♖×d2+ 19 ♔×d2 g5 20 ♖c1 ♗b4+ 21 ♔c2 f4 22 e×f4 e×f4 23 a3 f×g3 24 a×b4 g×h2 25 ♔c3 ♘e7 26 ♖h1 ♘d5+ 27 ♔c4 ♘f4 28 ♖×h2 h5 29 ♔d4 ♔e7 30 ♔e4 ♖d8 31 ♗f1 ♔f6 32 ♗c4 a6 33 b5 ♖e8+ 34 ♔d4 a×b5 35 ♗×b5 ♖e5 36 ♗c4 b5 37 ♗f1 b4 38 ♔c4 ♖e1 39 ♖h1 ♖b1 40 b3 ♖b2 41 ♖h2 ♖b1 42 ♗g2 ♖b2 43 ♗f1 h4 44 ♔×b4 ♖c2 45 ♗c4 ♔f5 46 ♗b5 ♘d5+ 47 ♔a3 ♔f4 48 ♗d3 ♖d2 49 ♗e4 ♘c3 50 ♖g2 ♘×e4 51 f×e4 g4 52 b4♔f3 53 ♖g1 ♖×f2 54 e5 g3 55 e6 ♖e2 56 ♖f1+ ♔g4 57 e7 ♖×e7 58 ♖d1 ♖e3+ 59 ♔a4 ♖e4 60 ♖d7 h3 61 ♖g7+ ♔f3 62 ♖c7 h2 63 ♖f7+ ♔g2 0-1

**Fifth World Computer Chess Championship, Köln, West Germany, June 11th - 15th 1986**

| | rate | perf | 1 | 2 | 3 | 4 | 5 | total |
|---|---|---|---|---|---|---|---|---|
| 1 Cray Blitz (USA) | 2200 | 2290 | 20+ | 6– | 3+ | 14+ | 2+ | 4 |
| 2 Hitech (USA) | 2270 | 2303 | 18+ | 14+ | 7+ | 5+ | 1– | 4 |
| 3 Bebe (USA) | 2100 | 2215 | 16+ | 15+ | 1– | 11+ | 5+ | 4 |
| 4 Sun Phoenix (Canada) | 0 | 2318 | 5– | 11+ | 18+ | 7+ | 6+ | 4 |
| 5 Rebel (Netherlands) | 0 | 2235 | 4+ | 12+ | 6+ | 2– | 3– | 3 |
| 6 Bobby (West Germany) | 0 | 2188 | 19+ | 1+ | 5– | 8+ | 4– | 3 |
| 7 Plymate (Sweden) | 0 | 2102 | 21+ | 8+ | 2– | 4– | 12+ | 3 |
| 8 Mephisto (England) | 0 | 1973 | 9+ | 7– | 17+ | 6– | 14+ | 3 |
| 9 Dutch (Netherlands) | 0 | 1828 | 8– | 19+ | 11= | 15= | 13+ | 3 |
| 10 Nona (Netherlands) | 0 | 1552 | 14– | 18– | 21+ | 22+ | 15+ | 3 |
| 11 Advance 68K (England) | 0 | 1855 | 17+ | 4– | 9= | 3– | 19+ | 2½ |
| 12 Lachex (USA) | 0 | 1840 | 13+ | 5– | 16= | 18+ | 7– | 2½ |
| 13 Ostrich (Canada) | 1750 | 1689 | 12– | 20+ | 15= | 16+ | 9– | 2½ |
| 14 Schach 2.7 (West Germany) | 0 | 1716 | 10+ | 2– | 22+ | 1– | 8– | 2 |
| 15 Cyrus 68K (England) | 0 | 1572 | 22+ | 3– | 13= | 9= | 10– | 2 |
| 16 Vaxchess (England) | 0 | 1561 | 3– | 23+ | 12= | 13– | 17= | 2 |
| 17 Chat (West Germany) | 0 | 1533 | 11– | 21+ | 8– | 19= | 16= | 2 |
| 18 BCP (England) | 0 | 1645 | 2– | 10+ | 4– | 12– | 20= | 1 |
| 19 Enterprise (Denmark) | 0 | 1591 | 6– | 9– | 20+ | 17= | 11– | 1½ |
| 20 Awit (Canada) | 1600 | 1476 | 1– | 13– | 19– | 21+ | 18= | 1½ |
| 21 Rex (USA) | 0 | 1157 | 7– | 17– | 10– | 20– | 22+ | 1 |
| 22 Shess (Netherlands) | 1000 | 855 | 15– | bye | 14– | 10– | 21– | 1 |
| 23 Kempelen (Hungary) | 0 | 767 | bye | 16– | | | | 0 |

*Round 1, June 11, 1986*

**Rex – Plymate** 1 e4 e5 2 d4 exd4 3 ♕xd4 ♘c6 4 ♕a4 ♘f6 5 ♗g5 ♗c5 6 ♗e2 h6 7 ♗f4 0-0 8 ♘c3 ♖e8 9 ♘f3 ♗b4 10 0-0 ♗xc3 11 bxc3 ♖xe4 12 ♗xc7 ♕xc7 13 ♕b5 ♘e5 14 ♘xe5 ♖xe5 15 ♕d3 ♘d5 16 ♗f3 ♘xc3 17 ♗g4 d5 18 ♗xc8 ♖xc8 19 ♖fe1 ♖xe1+20 ♖xe1 ♘xa2 21 ♕b5 ♘c3 22 ♖e8+ ♖xe8 23 ♕xe8+ ♔h7 24 h4 a5 25 h5 a4 26 g3 ♘e4 27 ♕xa4 ♕c5 0-1

**Vaxchess – Bebe** 1 e4 c5 2 ♘f3 d6 3 d4 cxd4 4 ♘xd4 ♘f6 5 ♘c3 g6 6 ♗e2 ♗g7 7 ♗e3 ♘c6 8 0-0 0-0 9 ♘xc6 bxc6 10 a4 ♕a5 11 ♕d3 ♖b8 12 b3 ♘g4 13 ♗d2 ♕e5 14 ♕h3 h5 15 ♗c4 ♘xf2 16 ♕f3 ♘g4 17 ♗xf7+ ♔h8 18 ♕g3 ♕d4+ 19 ♔h1 ♕xd2 20 ♘b1 ♕e3 21 ♖a2 ♖xf7 22 ♕xe3 ♖xf1+ 23 ♕g1 ♘f2# 0-1

**Ostrich – Lachex** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗xc6 dxc6 5 d4 exd4 6 ♕xd4 ♕xd4 7 ♘xd4 ♘f6 8 0-0 ♗c5 9 c3 0-0 10 f3 ♗d6 11 ♗g5 c5 12 ♗xf6 gxf6 13 ♘e2 ♗e6 14 ♘d2 ♖fd8 15 ♖fd1 ♔h8 16 a4 b5 17 axb5 axb5 18 ♖xa8 ♖xa8 19 g3 f5 20 ♔f2 ♖a2 21 ♖b1 ♗d7 22 ♔e3 ♖a8 23 ♖e1 ♔g8 24 f4 ♖e8 25 ♔d3 fxe4+ 26 ♘xe4 ♗f5 27 ♘c1 ♗f8 28 b3 c4+ 29 bxc4 bxc4+ 30 ♔xc4 ♗xe4 31 ♖d1 c6 32 ♖d2 ♗d5+ 33 ♔d3 ♗c5 34 ♖e2 ♖b8 35 c4 ♗e6 36 ♔c3 ♗b4+ 37 ♔c2 ♗xc4 38 ♖e5 f6 39 ♖f5 ♖d8 40 ♖xf6 ♖d2+ 41 ♔b1 ♗c3 0-1

**Cray Blitz – Awit** 1 e4 e5 2 ♘f3 ♘f6 3 ♘xe5 d6 4 ♘f3 ♘xe4 5 ♕e2 ♕e7 6 d3 ♘f6 7 ♗g5 ♕xe2+ 8 ♗xe2 ♗e7 9 ♘c3 c6 10 0-0-0 ♗e6 11 ♖de1 h6 12 ♗f4 0-0 13 ♘d4 ♗d7 14 ♗f3 ♖e8 15 ♘cb5 ♘a6 16 ♘xd6 ♗xd6 17 ♗xd6 ♔h8 18 ♔d2 ♘g4 19 ♘b3 ♘xf2 20 ♖xe8+ ♖xe8 21 ♖f1 ♘g4 22 ♘a5 ♗c8 23 ♘xb7 ♘xh2 24 ♗xh2 ♗xb7 25 ♗g1 ♘c7 26 ♗xa7 ♘e6 27 ♗h5 c5 28 g3 g6 29 ♖xf7 ♗e4 30 ♗g4 ♗d5 31 c4 ♘g5 32 ♖c7 ♗e6 33 ♗e2 ♖d8 34 ♔c3 ♖a8 35 ♗xc5 ♗h3 36 ♗d4+ ♔g8 37 ♖g7+ ♔f8 38 ♖xg6 ♔f7 39 ♖xh6 ♖a5 40 ♗f6 ♘e6 1-0

**Advance 68K – Chat** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗a4 ♘f6 5 0-0 ♗e7 6 ♖e1 b5 7 ♗b3 0-0 8 c3 d6 9 h3 ♘a5 10 ♗c2 c5 11 d4 ♘d7 12 ♘bd2 cxd4 13 cxd4 ♘c6 14 ♘b3 a5 15 ♗d3 ♕b6 16 ♗e3 exd4 17 ♘fxd4 ♘c5 18 ♘xb5 ♗e6 19 ♗e2 ♗xb3 20 axb3 ♘b4 21 ♗c4 ♕c6 22 ♗d4 ♔h8 23 ♖a3 ♘xe4 24 ♗xg7+ ♔xg7+ 25 ♕g4+ ♗g5 26 ♘d4 h5 27 ♘xc6 hxg4 28 ♘xb4 axb4 29 ♖xa8 ♖xa8 30 ♖xe4 ♖a1+ 31 ♔h2 f5 32 ♖e6 gxh3 33 ♔xh3 ♖d1 34 ♔g3 d5 35 ♖d6 d4 36 f4 ♗e7 37 ♖d5 ♔f6 38 ♖e5 ♗d8 39 ♖b5 ♗e7 40 ♖b6+ ♔g7 41 ♗e6 ♖d3+ 42 ♔h2 ♖e3 43 ♗xf5 ♖xb3 44 ♖g6+ ♔f7 45 ♗e6+ ♔xg6 46 ♗xb3 d3 47 ♗e6 ♗f6 48 b3 d2 49 ♗g4 ♗d4 50 ♗f3 ♔f5 51 g3 ♗f6 52 ♔g1 ♗g7 53 ♔f2 ♗d4+ 54 ♔f1 ♗f6 55 ♗d1 ♗d4 56 ♔e2 ♗c3 57 ♔e3 ♔g6 58 ♗f3 ♔f5 59 g4+ ♔g6 60 ♔d3 ♔f6 61 ♔c4 ♔g6 62 ♔d5 ♗b2 63 ♔e4 ♗g7 64 ♔d3 ♗c3 65 ♔e2 ♔f6 66 ♗d5 ♔g6 67 ♗e6 ♔f6 68 ♗f5 ♔f7 69 g5 ♔g7 70 ♗d3 ♔f7 71 f5 ♔g7 72 f6+ ♔f7 73 ♗c2 ♗e5 74 ♔f3 ♔e6 75 ♔e4 ♗g3 76 ♗d1 ♗f2 77 ♗g4+ 1-0 (adjudicated)

**BCP – Hitech** 1 e4 e5 2 ♘f3 ♘c6 3 d4 exd4 4 ♘xd4 ♘f6 5 ♘xc6 bxc6 6 e5 ♕e7 7 ♕e2 ♘d5 8 c4 ♗a6

9 ♘d2 ♘b4 10 ♘f3 c5 11 ♗f4 ♗b7 12 a3 ♘c6 13 0-0-0 h6 14 ♔b1 g5 15 ♗g3 ♗g7 16 ♕e4 ♘a5 17 ♕e3 0-0-0 18 ♔a1 ♖de8 19 h4 g4 20 ♘h2 h5 21 f3 g×f3 22 ♘×f3 f6 23 ♕c3 ♘c6 24 e×f6 ♗×f6 25 ♕d2 ♕g7 26 ♗f2 d6 27 ♖e1 ♘e5 28 ♘×e5 ♗×e5 29 ♗e3 ♖hf8 30 ♖b1 ♖e6 31 ♗g5 ♗d4 32 ♕d1 ♖f2 33 ♕b3 ♗e4 34 ♗d3 ♗×d3 35 ♕×d3 ♗×b2+ 36 ♔a2 ♗c1+ 37 ♔b3 ♗×g5 38 h×g5 ♕×g5 39 ♕h3 ♕g4 40 ♕×g4 h×g4 41 ♖h8+ ♔d7 42 ♖a8 ♖e3+ 43 ♔a4 ♖c2 44 ♖×a7 ♖×c4+ 45 ♔b5 ♖c2 46 ♖f1 ♖b2+ 47 ♔a4 ♔c6 48 ♖f4 ♖a2 49 ♖a6+ ♔b7 50 ♖×d6 c×d6 0-1

**Enterprise – Bobby** 1 d4 ♘f6 2 c4 d6 3 ♘f3 ♗f5 4 ♘c3 ♘bd7 5 g3 e5 6 ♗g2 c6 7 0-0 h6 8 c5 e4 9 ♘d2 d×c5 10 ♘d×e4 ♗×e4 11 ♗×e4 c×d4 12 ♕×d4 ♗c5 13 ♕d2 ♘×e4 14 ♘×e4 ♕e7 15 ♘×c5 ♘×c5 16 ♖d1 0-0 17 ♕c2 ♘a6 18 ♗f4 ♖fd8 19 ♕c4 ♕f6 20 ♕b3 ♘c5 21 ♕e3 b6 22 ♗e5 ♕f5 23 ♗d6 ♘e4 24 ♗f4 ♖e8 25 ♖d4 g5 26 ♗c7 ♖ac8 27 g4 ♕×g4+ 28 ♗g3 c5 29 ♖a4 b5 30 f3 ♘×g3 31 ♕×e8+ ♖×e8 32 ♖×g4 ♘×e2+ 33 ♔f1 f5 34 ♖g2 ♘d4 35 f4 ♘e6 36 f×g5 h×g5 37 h4 g4 38 h5 ♘f4 39 ♖c2 ♘d3 40 ♖d1 ♖e3 41 h6 ♔h7 42 ♖h2 f4 43 b3 c4 44 b×c4 b×c4 45 ♖h4 c3 46 ♖h2 g3 47 ♖c2 f3 48 ♖b1 ♖e6 0-1

**Mephisto – Dutch** 1 c4 d6 2 d4 ♘f6 3 ♘c3 g6 4 e4 ♗g7 5 ♘f3 0-0 6 ♗e2 e5 7 d×e5 d×e5 8 ♕×d8 ♖×d8 9 ♗g5 ♖e8 10 ♘d5 ♘×d5 11 c×d5 c6 12 ♗c4 c×d5 13 ♗×d5 ♘d7 14 ♖c1 h6 15 ♗h4 ♘b6 16 ♗b3 ♗e6 17 ♗×e6 ♖×e6 18 ♗g3 ♖c8 19 0-0 ♖c4 20 ♖fe1 ♖×c1 21 ♖×c1 f6 22 ♘h4 ♔h7 23 f3 ♖d6 24 ♗e1 h5 25 ♗a5 ♘a4 26 b3 ♘b2 27 ♖c2 ♘d3 28 g3 b6 29 ♗d2 ♘c5 30 ♗e3 ♘e6 31 ♔f2 ♗h6 32 ♔e2 ♗g5 33 ♘g2 ♗h6 34 f4 ♘d4+ 35 ♗×d4 ♖×d4 36 ♖c7+ ♗g7 37 ♖×a7 ♖×e4+ 38 ♔f3 ♖d4 39 ♔e3 ♖d1 40 f×e5 f×e5 41 ♖b7 ♖c1 42 ♖×b6 ♖c2 43 ♘h4 ♖×a2 44 ♘f3 ♖b2 45 ♘g5+ ♔h6 46 ♘f7+ ♔h7 47 h4 ♖g2 48 ♘g5+ ♔h6 49 ♔f3 ♖d2 50 ♖b7 ♖d3+ 51 ♔f2 ♖d2+ 52 ♔e3 ♖d1 53 ♘f7+ ♔h7 54 ♘×e5 ♖e1+ 55 ♔f4 ♖×e5 56 ♖×g7+ ♔×g7 57 ♔×e5 ♔f7 58 b4 ♔e7 59 b5 ♔d7 60 ♔f6 ♔d6 61 ♔×g6 1-0

**Schach 2.7 – Nona** 1 d4 ♘f6 2 c4 e5 3 d×e5 ♘g4 4 ♘f3 ♗c5 5 e3 ♘c6 6 a3 ♘g×e5 7 ♘×e5 ♘×e5 8 ♗e2 d6 9 b4 ♗b6 10 ♗b2 f6 11 0-0 ♗f5 12 a4 a5 13 ♗d4 a×b4 14 ♗×b6 c×b6 15 ♕b3 ♔d7 16 ♕×b4 ♔c8 17 ♖d1 ♘f7 18 ♕b2 ♘e5 19 ♘c3 ♖e8 20 ♘b5 ♘f7 21 ♖d4 ♔b8 22 ♗h5 g6 23 ♗f3 ♕e7 24 ♖ad1 ♖d8 25 ♖1d2 h5 26 ♕b3 ♘g5 27 ♗d5 ♖a5 28 ♖b2 ♖a6 29 ♖d1 ♗g4 30 ♖db1 1-0 (time)

**Rebel – Sun Phoenix** 1 d4 c5 2 d5 e5 3 e4 d6 4 ♗d2 ♘f6 5 ♘c3 ♗d7 6 ♗e2 ♕b6 7 ♖b1 ♘a6 8 ♗e3 0-0-0 9 b4 ♘b8 10 ♘f3 ♘g4 11 b×c5 ♕c7 12 ♗d2 ♕×c5 13 0-0 ♗e7 14 ♘b5 f5 15 ♖b3 ♗×b5 16 ♗×b5 f×e4 17 ♖c3 e×f3 18 ♕×f3 ♘f6 19 ♖×c5+ d×c5 20 ♕f5+ ♘bd7 21 c4 a6 22 ♗×a6 ♘×d5 23 c×d5 1-0

**Cyrus 68K – Shess** 1 e4 e5 2 ♘c3 ♘f6 3 ♘f3 ♘c6 4 a3 ♗d6 5 ♗c4 0-0 6 d3 a6 7 ♗e3 b5 8 ♗b3 ♕e7 9 a4 ♖b8 10 0-0 h5 11 ♕d2 ♖b7 12 a×b5 a×b5 13 ♖fc1 ♖e8 14 ♘d5 ♘×d5 15 e×d5 ♘d8 16 ♖a8 e4 17 d×e4 f5 18 e5 f4 19 ♗×f4 ♘f7 20 e×d6 ♘×d6 21 ♖e1 ♕d8 22 ♖×e8+ ♕×e8 23 ♗×d6 c×d6 24 ♕c3 b4 25 ♕c4 ♔f7 26 ♖×c8 1-0

*Round 2, June 12, 1986*

**Hitech – Schach 2.7** 1 e4 c5 2 ♘f3 d6 3 ♗c4 e6 4 d4 c×d4 5 ♘×d4 ♘f6 6 ♘c3 ♗e7 7 ♗e3 ♘bd7 8 ♕d2 ♘e5 9 ♗e2 0-0 10 h3 ♗d7 11 ♘f3 ♘×f3+ 12 g×f3 ♕a5 13 0-0-0 ♖ac8 14 ♖hg1 ♖fe8 15 ♗h6 g6 16 ♗g5 ♕c5 17 ♕f4 ♘h5 18 ♕h4 f6 19 ♗e3 ♕a5 20 ♗b5 ♗×b5 21 ♕×h5 g5 22 ♗×g5 f×g5 23 ♖×g5+ ♔h8 24 ♖dg1 1-0

**Lachex – Rebel** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗×c6 d×c6 5 d4 e×d4 6 ♕×d4 ♕×d4 7 ♘×d4 c5 8 ♘f3 ♗g4 9 ♗f4 ♗×f3 10 g×f3 0-0-0 11 0-0 ♗d6 12 e5 ♗e7 13 ♘c3 f6 14 ♖ad1 f×e5 15 ♗×e5 ♘f6 16 ♔h1 ♖hf8 17 ♖de1 ♖f7 18 ♖e3 ♖d2 19 ♖e2 ♖×e2 20 ♘×e2 ♗d6 21 ♗×d6 c×d6 22 ♘c3 d5 23 ♔g2 ♖e7 24 ♖d1 d4 25 ♘e4 b6 26 ♘×f6 g×f6 27 c3 d×c3 28 b×c3 ♔c7 29 ♖d5 ♖e2 30 ♖f5 ♖×a2 31 ♖×f6 ♖c2 32 ♖f7+ ♔c6 33 ♖×h7 ♖×c3 34 ♖h6+ ♔b5 35 f4 ♖d3 36 f5 a5 37 f6 ♖d7 38 ♖h3 ♔b4 39 ♖e3 ♖f7 40 ♖e4+ c4 41 ♖e6 b5 42 ♖e4 ♖×f6 43 h4 a4 44 h5 ♖h6 45 f4 ♖×h5 46 ♖e2 c3 47 f5 0-1

**Dutch – Enterprise** 1 ♘f3 d5 2 g3 c6 3 ♗g2 ♗g4 4 0-0 ♘d7 5 d3 e5 6 ♘bd2 ♗c5 7 h3 ♗f5 8 c3 f6 9 d4 ♗b6 10 d×e5 f×e5 11 c4 ♘e7 12 ♘h4 ♗e6 13 c×d5 c×d5 14 ♘b3 0-0 15 ♗g5 ♘f6 16 ♖c1 ♖c8 17 ♘f3 ♖×c1 18 ♗×c1 ♘e4 19 e3 ♕c7 20 ♕e2 ♖c8 21 ♘bd2 ♗f5 22 ♖d1 ♘×d2 23 ♘×d2 ♕c2 24 ♔h1 ♕a4 25 a3 ♗d3 26 ♕h5 ♕c2 27 b3 ♗a5 28 b4 ♗b6 29 ♕g4 ♗f5 30 ♕e2 ♖c3 31 ♔h2 ♗g6 32 ♕g4 ♖c6 33 ♔g1 ♗f5 34 ♕h5 g6 35 ♕e2 ♖c3 36 ♕f1 a5 37 b×a5 ♗×a5 38 g4 ♗d3 39 ♕e1 ♕a2 40 e4 d×e4 41 a4 ♖×c1 42 ♖×c1 ♗×d2 43 ♖a1 ♕c2 44 ♕d1 ♕×d1+ 45 ♖×d1 e3 46 f×e3 ♗×e3+ 47 ♔h1 e4 48 ♗f1 ♗×f1 49 ♖×f1 ♗c5 50 ♖c1 b6 51 a5 e3 52 ♔g2 ♗d4 53 a6 b5 54 a7 ♗×a7 55 ♖c7 ♘d5 56 ♖×a7 b4 57 ♔f3 b3 58 ♖b7 h5 59 g×h5 g×h5 60 h4 ♔f8 61 ♖b5 b2 62 ♖×b2 ♔e7 63 ♖g2 ♔d6 64 ♖g5 ♔c5 65 ♖×g5 ♔d4 66 ♖h8 ♘e7 67 ♖d8+ ♔c5 68 h5 ♘f5 69 ♖f8 ♘h6 70 ♔×e3 ♔d5 71 ♖a8 ♘g4+ 72 ♔f4 ♘f6 73 ♔f5 ♘×h5 74 ♖g8 1-0

**Awit – Ostrich** 1 c4 e5 2 ♘c3 d6 3 g3 g6 4 ♗g2 ♗g7 5 d3 ♘e7 6 ♘f3 0-0 7 e4 ♗g4 8 ♗e3 ♗×f3 9 ♕×f3 ♘bc6 10 ♕g4 h5 11 ♕h4 ♖e8 12 0-0-0 a5 13 f4 e×f4 14 ♕×f4 ♗×c3 15 b×c3 ♕d7 16 ♖hf1 ♕e6 17 ♕g5 ♘e5 18 ♔c2 ♘g4 19 ♗g1 f6 20 ♕b5 b6 21 ♖b1 ♖ad8 22 d4 f5 23 ♖fe1 c6 24 e×f5 ♕×e1 25 ♖×e1 c×b5 26 h3 ♘f6 27 ♖e6 ♔f7 28 c×b5 ♘×f5 29 d5 ♖×e6 30 d×e6+ ♔×e6 31 c4 ♖b8 32 ♗f2 d5 33 g4 ♘d6 34 ♗×d5+ ♘×d5 35 c×d5+ ♔×d5 36 g×g5 g×h5 37 a4 ♘e4 38 ♗g1 ♔c4 39 ♗e3 ♘c3 40 ♗d2 ♘×a4 0-1

**Plymate – Mephisto** 1 e4 c6 2 d4 d5 3 ♘c3 d×e4 4 ♘×e4 ♗f5 5 ♘g3 ♗g6 6 ♗c4 e6 7 ♘1e2 ♘f6 8 ♘f4 ♗d6 9 ♘×g6 h×g6 10 0-0 b5 11 ♗e2 ♘e4 12 ♗g4 ♕h4 13 ♗h3 ♘g5 14 ♗×g5 ♕×g5 15 ♕d3 ♘d7 16 ♘e4 ♕d5 17 a4 ♗c7 18 a×b5 c×b5 19 ♘c3 ♕d6

20 f4 b4 21 ♘b5 ♕b6 22 ♕c4 ♔d8 23 ♔h1 a6 24 ♘×c7 ♕×c7 25 ♕b3 ♕c6 26 ♕×b4 ♖×h3 27 ♕a5+ ♔e8 28 d5 ♕×c2 29 g×h3 ♕e4+ 30 ♔g1 ♕d4+ 31 ♖f2 e×d5 32 ♕c7 ♕e3 33 ♕d6 ♖b8 34 b4 ♕e4 35 ♕×a6 ♕×b4 36 ♕c6 ♕d4 37 ♖e1+ ♔d8 38 ♕c2 ♘f6 39 ♖c1 ♕b6 40 ♔h1 ♘e4 41 ♖ff1 ♖b7 42 ♕c8+ ♔e7 43 ♖fe1 ♖d7 44 ♕h8 ♕f6 45 ♖f1 ♖d8 46 ♕h7 ♔d7 47 ♖cd1 ♘c3 48 ♖d3 ♖a8 49 ♖c1 ♖c8 50 ♖e1 d4 51 ♖g3 ♖a8 52 ♖g2 ♘d5 53 f5 ♘e3 54 ♖d2 ♕c6+ 55 ♔g1 ♕c3 56 ♖de2 ♖a1 57 ♕h4 g×f5 58 ♖×a1 ♕×a1+ 59 ♖e1 ♕b2 60 ♕g3 g6 61 ♕f3 f6 62 ♕a8 f4 63 ♖a1 ♕b5 64 ♕e4 ♕g5+ 65 ♔h1 ♕d5 66 ♕×d5+ ♘×d5 67 ♖d1 ♔e6 68 ♖×d4 ♔e5 69 ♖a4 ♘c3 70 ♖a3 ♘d5 71 ♔g1 ♔e4 72 ♔f2 g5 73 h4 ♘e3 74 ♖a4+ ♔f5 75 h5 ♘g4+ 76 ♔g2 ♘e5 77 ♖a7 ♘g4 78 h3 ♘h6 79 ♖h7 ♘g8 80 h6 ♔g6 81 ♖g7+ ♔×h6 82 ♖×g8 ♔h7 83 ♖c8 f5 84 ♖f8 ♔g7 85 ♖×f5 ♔g6 86 ♖d5 1-0

**Kempelen – Vaxchess** 1 e4 ♘f6 2 d4 ♘c6 3 ♗d3 d6 4 ♗g5 ♘×d4 0-1 (time)

**Sun Phoenix – Advance 68K** 1 d4 ♘f6 2 ♗g5 ♘e4 3 ♗h4 d5 4 f3 ♘d6 5 e3 ♘f5 6 ♗f2 ♕d6 7 ♘c3 e6 8 e4 d×e4 9 f×e4 ♘h6 10 ♕d2 ♕b4 11 0-0-0 ♗d7 12 ♕g5 a6 13 a3 ♕e7 14 ♕f4 ♕f6 15 ♕×f6 g×f6 16 ♗h4 ♘g4 17 ♗e2 ♗h6+ 18 ♔b1 ♘e3 19 ♖e1 ♘×g2 20 ♗×f6 ♘×e1 21 ♗×h8 ♘c6 22 ♗f6 ♖b8 23 ♘h3 ♗d2 24 ♖g1 ♘e7 25 ♗g5 ♗×g5 26 ♘×g5 ♘×c2 27 ♔×c2 h6 28 ♘h7 ♖d8 29 ♗h5 ♗c8 30 ♘e2 ♘c6 31 ♘f6+ ♔e7 32 ♘g8+ ♔e8 33 ♘×h6 ♘×d4+ 34 ♘×d4 ♖×d4 35 ♖g8+ ♔d7 36 ♘×f7 c5 37 ♖d8+ ♔c7 38 ♖×d4 c×d4 39 ♔d3 b5 40 h4 ♔d7 41 ♔×d4 ♔e7 42 ♔e5 a5 43 ♘g5 a4 44 ♗g4 ♗d7 45 h5 ♔f8 46 h6 ♔g8 47 ♗×e6+ ♗×e6 48 ♔×e6 b4 49 ♔f6 b3 50 e5 ♔h8 51 e6 1-0

**Chat – Rex** 1 d4 ♘f6 2 c4 c5 3 d5 b5 4 c×b5 a6 5 b×a6 ♗×a6 6 ♘c3 e5 7 e4 ♗×f1 8 ♔×f1 ♗d6 9 ♘f3 0-0 10 ♗g5 h6 11 ♗e3 ♕b6 12 ♕e2 ♘a6 13 a4 ♖fb8 14 ♖a2 ♘b4 15 ♖a1 g5 16 h4 g4 17 ♘d2 ♘c2 18 ♘c4 ♕b3 19 ♖d1 ♘×e3+ 20 f×e3 ♗c7 21 g3 ♖b4 22 d6 ♕×c4 23 d×c7 ♕×e2+ 24 ♔×e2 ♖×b2+ 25 ♖d2 ♖×d2+ 26 ♔×d2 ♖c8 27 ♖f1 ♔g7 28 ♖f5 d6 29 ♘b5 ♘×e4+ 30 ♔d3 ♘×g3 31 ♖×f7+ ♔×f7 32 ♘×d6+ ♔e7 33 ♘×c8+ ♔d7 34 ♘e7 ♔×c7 35 a5 e4+ 36 ♔c4 ♔d6 37 ♘c8+ ♔e6 38 a6 ♔d7 39 a7 ♔×c8 40 a8♕+ ♔c7 41 ♕f8 ♔d7 42 ♕f2 ♘h1 43 ♕e1 ♔d6 44 ♕×h1 ♔e5 45 ♔×c5 h5 46 ♕g2 ♔f5 47 ♔d6 ♔f6 48 ♕×e4 ♔g7 49 ♕e5+ ♔g8 50 ♕×h5 g3 51 ♕g6+ ♔h8 52 e4 g2 53 e5 g1♕ 54 ♕×g1 ♔h7 55 e6 ♔h8 1-0

**Nona – BCP** 1 d4 ♘f6 2 c4 g6 3 ♘c3 d5 4 ♘f3 ♗g7 5 ♕b3 e6 6 c×d5 ♘×d5 7 e4 ♘×c3 8 ♕×c3 0-0 9 ♗f4 ♘c6 10 ♖d1 ♗d7 11 ♗c4 ♘×d4 12 ♘×d4 c5 13 ♗e3 c×d4 14 ♗×d4 ♗×d4 15 ♖×d4 ♕c7 16 0-0 ♖ac8 17 f4 ♕b6 18 ♔h1 ♖fd8 19 ♖fd1 ♗a4 20 ♖×d8+ ♖×d8 21 ♖×d8+ ♕×d8 22 ♗e2 ♗c6 23 ♕e3 ♕a5 24 ♗c4 ♕a4 25 ♕d3 b5 26 ♕d8+ ♔g7 27 ♗f1 ♕×e4 28 ♕d2 e5 29 f×e5 ♕×e5 30 ♕f2 ♔g8 31 ♕×a7 ♕×b2 32 ♔g1 ♕e5 33 ♕b6 ♕e4 34 ♕d8+ ♕e8 35 ♕×e8+ ♗×e8 36 a3 ♗c6 37 ♔f2 h5 38 ♔e3 g5 39 ♔d4 f5 40 ♔c5 ♗e4 41 ♔×b5 f4 42 a4 ♔f8 43 a5 ♔e8 44 a6 ♔d7 45 a7 ♔d6 46 ♔a6 ♔c5 47 ♔a5 ♗d5 48 ♗e2 ♗×g2 49 ♗b5 ♗d5 50 ♗d3 f3 51 ♗f1 f2 52 ♗e2 ♗e4 53 ♗b5 ♗f3 54 ♗d3 ♔d4 55 ♗b5 ♔c3 56 ♗a6 ♗d5 57 ♗f1 ♗e4 58 ♔b5 ♔d4 59 ♔a4 ♔c5 60 ♔a5 h4 61 h3 ♗d5 62 ♔a6 ♔d4 63 ♔b6 ♗e4 64 ♔b5 ♔c3 65 ♔a4 ♗d5 66 ♔b5 ♗h1 67 ♔a4 ♗e4 68 ♔a3 ♗d5 69 ♔a4 ♔d4 70 ♔b5 ♗e4 71 ♔b6 ♔c3 72 ♔b5 ♔c2 73 ♔c4 ♔d2 74 ♔c5 ♔e1 75 ♗c4 f1♕ 0-1

**Bebe – Cyrus 68K** 1 e4 ♘c6 2 d4 e6 3 d5 e×d5 4 e×d5 ♘e5 5 ♘c3 ♗b4 6 ♕d4 ♕e7 7 ♗e3 ♘f6 8 0-0-0 0-0 9 ♗e2 ♕d6 10 ♘b5 ♕×d5 11 ♕×d5 ♘×d5 12 ♖×d5 ♘c6 13 ♘×c7 ♖b8 14 ♘f3 f6 15 a3 ♗a5 16 ♖×a5 ♘×a5 17 ♗×a7 ♘c6 18 ♗×b8 ♘×b8 19 ♖d1 1-0

**Bobby – Cray Blitz** 1 e4 e5 2 ♘c3 ♘f6 3 f4 d5 4 f×e5 ♘×e4 5 ♘f3 ♗e7 6 d4 0-0 7 ♗d3 f5 8 e×f6 ♗×f6 9 0-0 ♘c6 10 ♘×e4 d×e4 11 ♗×e4 ♘×d4 12 c3 ♘×f3+ 13 ♗×f3 ♕×d1 14 ♖×d1 ♖e8 15 ♗f4 c6 16 g4 ♖d8 17 ♔g2 ♗e6 18 a4 ♖f8 19 ♗d6 ♖f7 20 ♖e1 ♗d7 21 ♖ad1 ♗h4 22 ♖e2 ♔h8 23 c4 ♔g8 24 ♖d3 ♖d8 25 b4 ♖c8 26 ♖ee3 ♗g5 27 ♖e1 ♖d8 28 ♗c5 b6 29 ♗d6 ♗h4 30 ♖e5 ♗f6 31 ♖e2 ♗g5 32 h3 ♖c8 33 a5 b×a5 34 ♖e5 ♗d8 35 b×a5 ♗f6 36 ♖e1 ♗d8 37 a6 ♗h4 38 ♖e2 ♖d8 39 ♗c5 ♗g5 40 ♖b2 ♖c8 41 ♖b7 ♗e6 42 ♖d6 ♗×c4 43 ♖×c6 ♖d8 44 ♖×f7 ♔×f7 45 ♗×a7 ♖d2+ 46 ♔g3 ♗d5 47 ♖c7+ ♔g8 48 ♗×d5+ ♖×d5 49 ♗b6 ♖d8 50 ♖b7 ♖d3+ 51 ♔g2 ♖d2+ 52 ♔f3 ♖d3+ 53 ♔e4 ♖×h3 1-0

*Round 3, June 13, 1986*

**Plymate – Hitech** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗a4 ♘f6 5 0-0 b5 6 ♗b3 ♗b7 7 d3 ♗c5 8 ♘c3 0-0 9 ♗g5 h6 10 ♗h4 ♗e7 11 ♕e2 ♘d4 12 ♘×d4 e×d4 13 ♘b1 d5 14 ♘d2 d×e4 15 d×e4 c5 16 ♗×f6 ♗×f6 17 e5 ♕e8 18 ♖ae1 ♕c6 19 ♕f3 ♕×f3 20 ♘×f3 ♗e7 21 a3 c4 22 ♗a2 ♗c5 23 ♘h4 ♖fe8 24 ♘f5 ♖ad8 25 ♔h1 ♗c8 26 ♘g3 d3 27 c×d3 c×d3 28 f4 d2 29 ♖b1 ♗e3 30 ♗b3 ♗e6 31 ♗c2 ♖e7 32 ♗d1 ♖c7 33 f5 ♗c4 34 ♗e2 ♖e7 35 e6 f×e6 36 f×e6 ♖×e6 37 ♖bd1 ♗g5 38 ♗×c4 b×c4 39 ♘f5 ♗f6 40 ♘g3 ♗×b2 0-1

**Rebel – Bobby** 1 d4 ♘f6 2 c4 d6 3 ♘f3 ♗f5 4 ♘c3 ♘bd7 5 g3 e5 6 ♗g2 c6 7 0-0 h6 8 d5 c×d5 9 ♘×d5 ♗e7 10 ♘h4 ♗e4 11 ♗×e4 ♘×e4 12 ♘×e7 ♕×e7 13 f3 ♘ec5 14 b4 ♘a6 15 ♘f5 ♕f6 16 ♘×d6+ ♔e7 17 c5 ♘×b4 18 ♗e3 b6 19 f4 ♕e6 20 ♕d2 ♘a6 21 f×e5 ♘×e5 22 ♘f5+ ♔f8 23 ♕d6+ ♕×d6 24 c×d6 f6 25 ♗d4 ♘d7 26 e4 ♘ac5 27 ♘e7 ♘×e4 28 ♘g6+ ♔g8 29 ♘×h8 ♔×h8 30 ♖ad1 ♖d8 31 ♖f3 ♔g8 32 ♖a3 a5 33 ♖b3 a4 34 ♖b4 ♘f8 35 ♗×b6 ♖×d6 36 ♖×d6 ♘×d6 37 ♗c5 1-0

**Mephisto – Chat** 1 c4 e5 2 ♘c3 ♘f6 3 ♘f3 ♘c6 4 e3 d5 5 c×d5 ♘×d5 6 ♗b5 ♘×c3 7 b×c3 ♗d7 8 d4 e4 9 ♘d2 ♕g5 10 ♗f1 ♕g6 11 ♕c2 f5 12 ♖b1 b6 13 g3 ♗e7 14 ♗e2 0-0 15 ♕b3+ ♗e6 16 ♗c4 ♗×c4 17 ♕×c4+ ♔h8 18 0-0 ♖ac8 19 a4 ♘a5 20 ♕a6 ♕c6 21 ♗b2 ♕×a4 22 ♕×a7 ♕c2 23 ♖fd1 h6 24 ♕a6 ♖a8 25 ♕b5 ♕d3 26 ♕×d3 e×d3 27 ♗c1 ♘c6 28 ♘f3 f4 29 e×f4 ♖a2 30 ♖×d3 ♖fa8 31 d5 ♘d8 32 ♗e3

♘b7 33 ♗d4 ♘d6 34 ♖e3 ♘f5 35 ♖e6 ♘×d4 36 ♘×d4 ♗c5 37 ♖c6 ♗×d4 38 c×d4 ♖8a7 39 d6 ♖a1 40 d7 ♖×b1+ 41 ♔g2 ♖a8 42 ♖×c7 ♖g8 43 ♖c8 ♔h7 44 ♖×g8 ♔×g8 45 d8♕+ ♔h7 46 d5 b5 47 d6 b4 48 ♕c8 ♖a1 49 d7 ♖d1 50 ♕c2+ g6 51 ♕×d1 g5 52 d8♕ g×f4 53 ♕e7+ 1-0

**Cyrus 68K – Ostrich** 1 e4 c5 2 ♘f3 ♘c6 3 ♘c3 ♘f6 4 d4 ♘×d4 5 ♘×d4 c×d4 6 ♕×d4 d6 7 ♕b4 ♕c7 8 ♗e3 a5 9 ♕a4+ ♗d7 10 ♗b5 0-0-0 11 ♗×d7+ ♘×d7 12 0-0-0 e6 13 ♕a3 d5 14 ♕b3 d×e4 15 ♘×e4 ♕c6 16 ♕c3 ♕×c3 17 ♘×c3 ♗b4 18 ♘e4 f5 19 ♘d6+ ♗×d6 20 ♖×d6 ♖he8 21 ♗d2 ♔c7 22 ♖d3 ♖a8 23 ♖g3 g6 24 ♖h3 h5 25 ♖g3 ♖g8 26 ♖c3+ ♔d6 27 ♗g5 ♘c5 28 ♖d1+ ♔c6 29 ♗e3 b6 30 ♖e1 ♔d6 31 ♗×c5+ b×c5 32 ♖ce3 ♖ae8 33 ♖a3 ♖a8 34 ♖d3+ ♔e7 35 ♖d5 c4 36 ♖de5 ♖gd8 37 ♖×e6+ ♔f7 38 ♖e7+ ♔f8 39 c3 ♖e8 40 ♖×e8+ ♖×e8 41 ♖d1 ♖e2 42 ♖d8+ ♔e7 43 ♖d2 ♖e1+ 44 ♔c2 a4 45 f4 h4 46 ♖f2 ♔d6 47 h3 ♔d5 48 ♔d2 ♖a1 49 ♖e2 ♖×a2 50 ♔c1 a3 51 ♖d2+ ♔c6 52 b×a3 ♖×a3 53 ♔b2 ♖a7 54 ♖e2 ♔d6 55 ♔c2 ♖a5 56 ♔b1 ♖a3 57 ♖e3 ♖a7 58 ♔b2 ♖a8 59 ♖e2 ♖a6 60 ♖e5 ♖a4 61 ♖e8 ♔d5 62 ♖e7 ♖a6 63 ♖e5+ ♔c6 64 ♖e6+ ♔b5 65 ♖×a6 ♔×a6 66 ♔a2 ♔b5 67 ♔b2 ♔a4 68 ♔a2 ♔a5 69 ♔b2 ½-½

**Cray Blitz – Bebe** 1 e4 c5 2 ♘f3 d6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 g6 6 ♗e2 ♗g7 7 0-0 0-0 8 ♗g5 ♘c6 9 ♘b3 ♗e6 10 f4 ♘d7 11 ♕d3 h6 12 ♗h4 ♗×b3 13 a×b3 a6 14 ♗f2 ♘b4 15 ♕d2 ♕c7 16 ♖a4 a5 17 ♖d1 ♘c5 18 ♖aa1 ♕c6 19 ♗f3 ♕c7 20 f5 ♔h7 21 f×g6+ f×g6 22 ♗e3 g5 23 ♕e2 ♘d7 24 ♗g4 ♔h8 25 ♗f5 ♘f6 26 ♖f1 ♕c6 27 ♖f3 ♘g8 28 ♕d2 ♗e5 29 ♗×g5 ♗g7 30 ♖h3 ♕c5+ 31 ♗e3 ♖×f5 32 ♗×c5 ♖×c5 33 ♖f1 ♖f8 34 ♖×f8 ♗×f8 35 g4 ♔h7 36 g5 ♔g6 37 g×h6 ♗×h6 38 ♕d1 ♘f6 39 ♔g2 ♖e5 40 ♔h1 ♖g5 41 ♕f1 ♔g7 42 ♕e2 ♖c5 43 ♕f2 b6 44 ♕g1+ ♔h7 45 ♘e2 ♘h5 46 ♕g4 1-0

**Advance 68K – Dutch** 1 e4 d6 2 d4 ♘f6 3 ♘c3 g6 4 f4 ♗g7 5 ♘f3 0-0 6 ♗d3 ♘c6 7 0-0 e5 8 d×e5 d×e5 9 f5 g×f5 10 e×f5 e4 11 ♘×e4 ♗×f5 12 ♘×f6+ ♕×f6 13 ♘g5 ♕d4+ 14 ♖f2 ♗×d3 15 ♘f3 ♗×c2 16 ♕×c2 ♕d6 17 ♕b3 ♕b4 18 ♕c2 ♖ae8 19 ♗g5 ♘d4 20 ♕×c7 ♘×f3+ 21 ♖×f3 ♗e5 22 ♕d7 ♗d4+ 23 ♔h1 ♖e1+ 24 ♖f1 ♖×a1 25 ♖×a1 ♗g7 26 ♕d2 ♕×b2 27 ♕×b2 ♗×b2 28 ♖b1 ♖e8 29 ♗d2 ♗d4 30 ♖b4 ♖d8 31 ♖a4 h5 32 g3 b5 33 ♖a6 ♗e5 34 ♗e3 h4 35 g×h4 ♗d4 36 ♗×d4 ♖×d4 37 ♖×a7 ♖×h4 38 ♖b7 b4 39 ♔g2 ♖g4+ 40 ♔f2 f6 41 h3 ♖e4 42 ♖b8+ ♔g7 43 ♔f3 f5 44 ♖b6 ♖d4 45 ♔e3 ♖h4 46 ♔f3 ♖c4 47 ♔g3 ♖d4 48 ♖a6 ♖d3+ 49 ♔h4 ♖e3 50 ♖b6 ♖e4+ 51 ♔g5 f4 52 ♖g6+ ♔h7 53 ♖f6 ♖e2 54 ♖f7+ ♔g8 55 ♖×f4 ♖g2+ 56 ♔h5 ♔g7 57 ♖×b4 ♖×a2 58 ♖b7+ ♔f6 59 h4 ♖a4 60 ♖d7 ♖f4 61 ♖a7 ♖c4 62 ♖h7 ♖d4 63 ♖h6+ ♔f5 64 ♖b6 ♖e4 65 ♖b7 ♔f4 66 ♖g7 ♔f5 67 ♖g5+ ♔f6 68 ♖g4 ♖e8 69 ♖f4+ ♔e5 70 ♔g4 ♖g8+ ♔f3 ♖g1 72 ♖e4+ ♔f5 73 ♖b4 ♖d1 74 ♖b5+ ♔g6 75 h5+ ♔h6 76 ♔g2 ♖e1 77 ♖a5 ♖e2+ 78 ♔g3 ♖e3+ 79 ♔h4 ♖e4+ 80 ♔h3 ♖e3+ 81 ♔g4 ♖e4+ 82 ♔f3 ♖h4 ½-½ (adjudicated)

**BCP – Sun Phoenix** 1 e4 e6 2 d4 d5 3 e5 c5 4 c3 ♕b6 5 ♘f3 ♗d7 6 ♗d3 c×d4 7 ♘×d4 ♘c6 8 ♘×c6 ♗×c6 9 0-0 0-0-0 10 ♘d2 f6 11 ♕g4 ♖e8 12 ♖e1 ♘h6 13 ♕h3 ♗c5 14 ♕g3 14 ♘g4 15 ♕×g4 ♗×f2+ 16 ♔f1 ♗×e1 17 ♔×e1 f×e5 18 ♕×g7 ♕e3+ 19 ♗e2 ♖hg8 20 ♕f7 ♗b5 21 ♕f2 ♕×e2+ 22 ♕×e2 ♗×e2 23 ♔×e2 ♖×g2+ 24 ♔e3 ♖×h2 25 a4 ♖f8 26 a5 h5 27 ♖a4 ♖h1 28 ♖a1 h4 29 ♖b1 ♖e1+ 30 ♔d3 h3 31 ♘f1 ♖f×f1 0-1

**Vaxchess – Lachex** 1 d4 d5 2 c4 e6 3 ♘c3 ♘f6 4 ♗g5 ♘bd7 5 e3 c6 6 ♘f3 ♕a5 7 c×d5 ♘×d5 8 ♕d2 ♗b4 9 ♖c1 h6 10 ♗f4 0-0 11 ♗g3 c5 12 ♗d3 f5 13 0-0 c×d4 14 e×d4 f4 15 ♕e1 ♘7f6 16 ♗h4 g5 17 ♗×g5 h×g5 18 ♕e5 ♘×c3 19 b×c3 ♗×c3 20 ♕×a5 ♗×a5 21 ♖c5 ♗b4 22 ♖e5 g4 23 ♖g5+ ♔f7 24 ♘e5+ ♔e8 25 ♖c1 ♔d8 26 ♖g7 ♗a3 27 ♖b1 f3 28 ♖b3 ♗c1 29 g×f3 g×f3 30 ♘f7+ ♔e7 31 ♗g6 ♘d5 32 ♘e5+ ♔f6 33 ♖h7 a5 34 ♖×f3+ ♗f4 35 h4 ♖a6 36 ♔f1 ♖c6 37 ♗e4 ♖c1+ 38 ♔e2 ♖g1 39 ♗×d5 ♔f5 40 ♖h5+ ♔f6 41 ♗e4 ♔g7 42 ♘d3 ♗d6 43 ♖×f8 ♔×f8 44 ♖g5 ♖×g5 45 h×g5 b6 46 g6 ♗a6 47 a4 e5 48 d×e5 ♗×e5 49 ♔e3 ♗c7 50 f3 ♗c8 51 ♘f4 ♗×f4+ 52 ♔×f4 ♗d7 53 ♗c2 ♔g7 54 ♔g5 b5 55 a×b5 ♗×b5 56 f4 ♗c4 57 f5 ♔g8 58 f6 ♔f8 59 ♔f4 ♔e8 60 ♔e5 ♔f8 61 ♔d4 ♗a2 62 g7+ ♔f7 63 ♗h7 ♔×f6 64 g8♗ ♗×g8 ½-½

**Enterprise – Awit** 1 d4 ♘f6 2 c4 g6 3 ♘c3 d5 4 c×d5 ♘×d5 5 e4 ♘×c3 6 b×c3 c5 7 ♗c4 ♗g7 8 ♘e2 0-0 9 0-0 ♘c6 10 d5 ♘a5 11 ♗b5 a6 12 ♗a4 b5 13 ♗c2 ♕d6 14 a4 b4 15 ♗f4 ♕b6 16 ♖b1 ♗g4 17 c×b4 c×b4 18 ♗e3 ♕d6 19 f3 ♗d7 20 ♗d4 ♖fc8 21 e5 ♕c7 22 ♗e4 ♖ab8 23 e6 f×e6 24 d×e6 ♗c6 25 ♗×g7 ♗×e4 26 f×e4 ♕c4 27 ♗e5 ♖d8 28 ♘d4 ♖bc8 29 ♖c1 ♕×c1 30 ♕×c1 ♖×c1 31 ♖×c1 b3 32 ♖b1 ♘c4 33 ♘c6 ♖d3 34 ♘×e7+ ♔f8 35 ♘c6 ♘d2 36 ♖e1 a5 37 ♘×a5 ♔e7 38 ♘b7 ♔×e6 39 ♘c5+ ♔×e5 40 ♘×d3+ ♔d4 41 ♘b2 ♘×e4 42 a5 ♘c5 43 ♖a1 ♔c3 44 ♖c1+ ♔b4 45 ♔f2 g5 46 ♔e3 ♔b5 47 ♖a1 ♔a6 48 ♘d3 ♘b7 49 ♘b4+ ♔b5 50 a6 ♘a5 51 ♖×a5+ ♔×a5 52 a7 ♔×b4 53 a8♕ h6 54 ♕b7+ ♔c3 55 ♕c6+ ♔b2 56 ♕×h6 ♔a2 57 ♕a6+ ♔b1 58 ♕g6+ ♔a1 59 ♕×g5 b2 60 ♕a5+ ♔b1 61 ♔d3 ♔c1 62 ♕e1# 1-0

**Rex – Nona** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗a4 d6 5 ♗×c6+ b×c6 6 d4 e×d4 7 ♕×d4 ♘f6 8 ♕a4 ♗d7 9 0-0 c5 10 ♕a5 ♘×e4 11 ♘a3 ♗e7 12 ♖e1 f5 13 ♗f4 0-0 14 ♘c4 ♗b5 15 ♘cd2 ♗f6 16 ♘×e4 f×e4 17 ♖×e4 ♗×b2 18 ♖d1 ♗c6 19 ♗g5 ♗f6 20 ♗×f6 ♖×f6 21 ♖e3 ♗×f3 22 g×f3 ♖g6+ 23 ♔f1 ♖h6 24 ♔g1 ♕g5+ 25 ♔f1 ♖×h2 26 ♔e2 ♕h4 27 ♖f1 ♕c4+ 28 ♔e1 ♕×c2 29 ♕d2 ♕b1+ 30 ♔e2 ♕b5+ 31 ♔e1 ♖b8 32 ♕d5+ ♔h8 33 ♕d2 ♕c4 34 ♖d3 ♖h5 35 f4 ♖d5 36 ♖×d5 ♖b1+ 37 ♕d1 ♕e4+ 38 ♔d2 ♕×d5+ 39 ♔c2 ♕×a2+ 40 ♔c3 ♕a3+ 41 ♔d2 ♖b2+ 42 ♕c2 ♕a5+ 43 ♔d1 ♕a1+ 44 ♔d2 ♖×c2+ 45 ♔×c2 ♕×f1 46 ♔c3 h5 47 f5 h4 48 f4 h3 49 f6 g×f6 0-1

**Schach 2.7 – Shess** 1 d4 d5 2 c4 e6 3 ♘c3 ♘f6 4 ♗g5 ♗e7 5 e3 0-0 6 ♘f3 h6 7 ♗h4 b6 8 c×d5 ♘×d5 9 ♗×e7 ♕×e7 10 ♘×d5 e×d5 11 ♖c1 ♗e6 12 ♕a4 c5 13 ♗e2 ♘d7 14 0-0 ♔h8 15 ♕d1 ♖ae8 16 ♗b5 ♖d8 17 ♖e1 g5 18 ♕a4 g4 19 ♘d2 ♘f6 20

♖c3 ♗f5 21 d×c5 b×c5 22 ♕a3 ♖c8 23 ♖ec1 ♕e5 24 ♖×c5 ♖fd8 25 ♖×c8 ♗×c8 26 ♘b3 a6 27 ♗d3 ♗b7 28 ♘d4 ♖b8 29 ♕a5 ♘e4 30 ♕c7 f6 31 ♕d7 h5 32 ♖c7 ♗c6 33 ♕h7# 1-0

*Round 4, June 14, 1986*

**Awit – Rex** 1 b3 e5 2 ♗b2 ♘c6 3 e3 d5 4 ♗b5 ♕g5 5 ♘f3 ♕×g2 6 ♖g1 ♕h3 7 ♗×e5 ♗g4 8 ♖g3 ♗×f3 9 ♕×f3 ♕h6 10 ♕×d5 ♘ge7 11 ♗×c6+ b×c6 12 ♕f3 ♘d5 13 ♕g4 ♕×h2 14 ♗×g7 ♗×g7 15 ♕×g7 ♔e7 16 ♕e5+ ♔d7 17 ♘a3 ♘e7 18 ♕d4+ ♔e6 19 ♕e4+ ♔d7 20 ♖g7 ♕h5 21 ♘c4 ♔e8 22 ♘e5 f5 23 ♕g2 ♕h6 24 ♘×c6 ♘×c6 25 ♖×c7 ♔f8 26 ♖×c6 ♕g7 27 ♖f6+ ♕×f6 28 ♕×a8+ ♔f7 29 ♕×a7+ ♔f8 30 d4 ♔e8 31 ♔e2 ♔f8 32 c4 h6 33 a4 ♖g8 34 a5 ♖g7 35 ♕c5+ ♖e7 36 a6 f4 37 a7 f3+ 38 ♔f1 ♔g8 39 a8♕+ ♔h7 40 ♕h5 ♖f7 41 ♖a6 ♔g7 42 ♖×f6 ♖×f6 43 ♕e5 h5 44 ♕×f3 1-0

**Lachex – BCP** 1 e4 c5 2 c3 ♘f6 3 e5 ♘d5 4 d4 c×d4 5 c×d4 d6 6 ♘f3 ♘c6 7 ♘a3 e6 8 ♗g5 ♗e7 9 ♗×e7 ♘d×e7 10 ♘b5 d×e5 11 d×e5 0-0 12 ♕×d8 ♖×d8 13 ♘c7 ♖b8 14 ♗b5 ♘g6 15 0-0 ♘c×e5 16 ♘×e5 ♘×e5 17 ♖fe1 ♘g4 18 ♖ad1 ♖×d1 19 ♖×d1 ♔f8 20 f3 ♘e3 21 ♖d3 ♘f5 22 g4 ♘h4 23 ♔f2 e5 24 ♖d5 ♘g6 25 ♖d8+ ♔e7 26 ♖d1 ♗e6 27 ♘×e6 ♔×e6 28 ♔e3 ♘f4 29 ♔e4 h6 30 ♗c4+ ♔f6 31 ♖d7 ♘e6 32 ♔e3 a5 33 a4 g6 34 h4 b6 35 b3 g5 36 h5 ♘d8 37 ♖d6+ ♔g7 38 ♗d5 f6 39 ♔e4 ♘f7 40 ♖d7 ♖f8 41 ♔f5 41 e4 42 f×e4 b5 43 a×b5 a4 44 b×a4 ♔h8 45 ♗×f7 ♖c8 46 ♔g6 ♖g8+ 47 ♗×g8 f5 48 ♖d8 f×e4 49 ♗h7# 1-0

**Chat – Enterprise** 1 d4 ♘f6 2 c4 e6 3 ♘c3 ♗b4 4 e3 0-0 5 ♗d3 c5 6 ♘f3 ♘c6 7 ♗d2 c×d4 8 e×d4 d5 9 c5 ♗×c3 10 ♗×c3 ♘e4 11 ♕c2 f5 12 0-0 ♕c7 13 ♔h1 ♖f6 14 b4 ♖h6 15 b5 ♘e7 16 ♗b4 ♘g6 17 c6 ♘f4 18 c×b7 ♕×b7 19 g3 ♘×d3 20 ♕×d3 a6 21 a4 a×b5 22 ♕×b5 ♕×b5 23 a×b5 ♖×a1 24 ♖×a1 ♘×f2+ 25 ♔g2 ♘d3 26 ♗d2 ♖f6 27 ♖b1 e5 28 ♘×e5 ♘×e5 29 d×e5 ♖f7 30 ♖c1 ♖f8 31 ♖c7 d4 32 ♖e7 d3 33 b6 ♗a6 34 ♖a7 ♗b5 35 ♔f3 ♗c6+ 36 ♔f4 ♖b8 37 ♗a5 ♖d8 38 ♗d2 g6 39 ♔e3 ♖c8 40 ♗c3 ♗e4 41 ♖c7 ♖b8 42 e6 ♔f8 43 ♗g7+ ♔e8 44 ♗d4 h6 45 ♖g7 d2 46 ♖g8+ ♔e7 47 ♖g7+ ♔×e6 48 ♔×d2 ♔d5 49 ♗f2 g5 50 ♖h7 f4 51 ♖×h6 ♖d8 52 ♔e1 ♗f3 53 ♖h7 ♖e8+ 54 ♔f1 ♗e2+ 55 ♔g1 ♔e6 56 ♖g7 ♔f6 57 ♗d4+ ♔f5 58 ♔f2 ♗d1 59 ♖f7+ ♔e6 60 ♖c7 ♔d6 61 ♗c5+ ♔d5 62 ♗e7 f×g3+ 63 ♔×g3 ♖b8 64 b7 ♗a4 65 ♖c5+ ♔e6 66 ♖c7 ♔d5 67 ♖c5+ ♔e6 68 ♖c7 ♔d5 ½-½

**Bobby – Mephisto** 1 e4 ♘f6 2 e5 ♘d5 3 d4 d6 4 c4 ♘b6 5 e×d6 e×d6 6 ♘c3 ♗e7 7 h3 0-0 8 ♘f3 ♘c6 9 ♗e2 ♗f5 10 0-0 ♕d7 11 ♗f4 ♖ae8 12 a4 ♗f6 13 a5 ♘c8 14 a6 b6 15 g4 ♗×g4 16 h×g4 ♕×g4+ 17 ♗g3 ♘×d4 18 ♘×d4 ♕×d4 19 ♕c2 h6 20 ♖fd1 ♕c5 21 ♘b5 ♗e5 22 ♖d5 ♕c6 23 ♗×e5 ♖×e5 24 ♖×e5 d×e5 25 ♗g4 ♘d6 26 ♘×a7 ♕c5 27 ♗d7 e4 28 ♘b5 ♘×b5 29 ♗×b5 ♕g5+ 30 ♔f1 ♕h4 31 ♖d1 ♕h1+ 32 ♔e2 ♕f3+ 33 ♔e1 ♕h1+ 34 ♔d2 ♕g2 35 ♔c1 ♕g6 36 ♕b3 c6 37 ♗a4 c5 38 ♕g3 ♖a8 39 ♕×g6 f×g6 40 ♗b5 g5 41 ♖d7 h5 42 a7 e3 43 f×e3 ♖f8 44 ♗c6 h4 45 a8♕ ♖×a8 46 ♗×a8 h3 47 ♖d5 1-0

**Hitech – Rebel** 1 e4 e5 2 f4 e×f4 3 ♗e2 ♕h4+ 4 ♔f1 ♘f6 5 ♘c3 ♗b4 6 e5 ♗×c3 7 d×c3 ♘g8 8 ♘f3 ♕h6 9 ♕d4 g5 10 h4 ♘c6 11 ♕e4 ♕g6 12 ♘×g5 ♕×e4 13 ♘×e4 f3 14 g×f3 ♘×e5 15 ♗f4 d6 16 ♖e1 ♗d7 17 ♗c4 ♔f8 18 ♗×e5 d×e5 19 ♘c5 ♗c6 20 ♖×e5 ♖d8 21 ♔f2 ♘f6 22 ♖f5 ♖d2+ 23 ♔e3 ♖d6 24 ♘e4 ♗×e4 25 f×e4 ♖g8 26 e5 ♖c6 27 e×f6 1-0

**Bebe – Advance 68K** 1 e4 c5 2 ♘f3 d6 3 d4 c×d4 4 ♘×d4 ♘f6 5 ♘c3 g6 6 ♗e2 ♗g7 7 0-0 ♘c6 8 ♘b3 0-0 9 ♗e3 ♗d7 10 f4 ♗e6 11 f5 g×f5 12 e×f5 ♗d7 13 ♕d2 ♕c7 14 ♗g5 ♘b4 15 ♖f2 h6 16 ♗×h6 ♗×h6 17 ♕×h6 ♕b6 18 ♖d1 ♘×c2 19 ♕g5+ ♔h7 20 ♖d3 ♕×f2+ 21 ♔×f2 ♘e4+ 22 ♘×e4 ♘e3 23 ♖×e3 ♗×f5 24 ♕×f5+ ♔h6 25 ♖g3 ♖ad8 26 ♕h3# 1-0

**Sun Phoenix – Plymate** 1 d4 d5 2 ♗g5 ♘f6 3 ♗×f6 e×f6 4 e3 c6 5 c4 ♗b4+ 6 ♘c3 0-0 7 c×d5 c×d5 8 ♕b3 ♘c6 9 ♗d3 ♗g4 10 ♘ge2 ♗×e2 11 ♗×e2 ♗×c3+ 12 ♕×c3 ♖c8 13 ♗g4 ♖c7 14 ♖c1 ♖e8 15 0-0 ♖ee7 16 ♕d2 ♕d6 17 f3 b5 18 ♖c5 a6 19 ♖fc1 ♘d8 20 ♕a5 ♖b7 21 ♗c8 ♖a7 22 ♕c3 a5 23 ♕b3 ♖ec7 24 ♕×b5 ♔f8 25 ♖1c3 ♕e7 26 ♔f1 ♖×c5 27 ♕×c5 ♔g8 28 ♕×d5 ♕e8 29 ♕c5 ♖a8 30 ♕b6 a4 31 d5 ♕e7 32 d6 ♕f8 33 ♗g4 a3 34 b×a3 ♕e8 35 d7 ♕e7 36 ♖c8 ♕×d7 1-0

**Ostrich – Vaxchess** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗×c6 d×c6 5 d4 e×d4 6 ♕×d4 ♕×d4 7 ♘×d4 ♗d6 8 ♘c3 ♘e7 9 0-0 h5 10 ♖e1 ♗g4 11 e5 ♗b4 12 f3 0-0-0 13 ♗e3 ♗×c3 14 b×c3 ♗d7 15 ♖ab1 ♘g6 16 ♗g5 ♖df8 17 ♖bd1 ♖e8 18 f4 c5 19 ♘f3 ♗g4 20 ♖f1 h4 21 h3 ♗f5 22 ♖f2 ♗e4 23 ♘h2 b6 24 ♖fd2 ♘e7 25 ♖d7 ♘f5 26 ♖×f7 1-0

**Shess – Nona** 1 e4 e5 2 ♘f3 ♘f6 3 d4 e×d4 4 e5 ♘e4 5 ♕×d4 d5 6 e×d6 ♘×d6 7 ♘c3 ♘c6 8 ♕f4 ♗f5 9 ♗d3 ♗×d3 10 c×d3 ♗e7 11 0-0 0-0 12 ♗e3 ♗f6 13 h4 ♘e7 14 h5 ♘ef5 15 g4 ♗×c3 16 b×c3 ♘×e3 17 f×e3 ♘b5 18 c4 ♘c3 19 ♕f5 g6 20 ♕e5 ♕×d3 21 c5 ♖ae8 22 ♕×c7 ♕×e3+ 23 ♔g2 ♕e6 24 ♔g3 ♘e4+ 25 ♔h4 ♘f2 26 ♖×f2 ♕f6+ 27 ♔g3 ♕×a1 28 ♕×b7 ♕c3 29 ♕×a7 ♖a8 30 ♕e7 ♖fe8 31 ♕d6 ♖ad8 32 ♖e2 ♖f8 33 ♕e5 ♖d3 34 ♕e4 f5 35 ♕e6+ ♔h8 0-1 (time)

**Dutch – Cyrus 68K** 1 ♘f3 d5 2 g3 ♘c6 3 d4 ♘f6 4 ♗g2 ♘e4 5 0-0 h6 6 ♕d3 e6 7 ♘bd2 ♘d6 8 ♖d1 a6 9 c4 ♘f5 10 c×d5 e×d5 11 e3 ♗e7 12 ♘e5 ♘×e5 13 d×e5 c6 14 e4 d×e4 15 ♕×d8+ ♗×d8 16 ♘×e4 0-0 17 b3 ♗b6 18 ♗b2 ♗e6 19 ♖ac1 ♖fd8 20 h3 ♔h7 21 h4 ♘e7 22 ♘d6 ♖db8 23 ♗e4+ ♔g8 24 ♗f3 ♘d5 24 a3 a5 26 ♘×b7 ♖×b7 27 ♖×c6 ♖d8 28 ♖d6 ♖×d6 29 e×d6 ♗d8 30 ♗×d5 ♗×d5 31 ♖×d5 ♖×b3 32 ♖d2 a4 33 d7 ♔h7 34 ♔g2 ♔g8 35 ♖e2 ♔h7 36 ♖e8 ♖×b2 37 ♖×d8 ♖d2 38 ♔f3 ♔g6 39 ♖a8 ♖×d7 40 ♖×a4 ♖d3+ 41 ♔e2 ♖b3 42 ♖a6+ ♔f5 43 a4 ♖b1 44 ♖a7 ♖b2+ 45 ♔e3 ♖b3+ 46 ♔d4 ♔g6 47 ♖a6+ ♔h7 48 ♔e4 ♖b2 49 f3 ♖b4+ 50 ♔e3 ♖b3+ 51 ♔f4 ♖b4+ 52 ♔e5 ♖b3 53 a5 ♖×f3 54 g4 ♖h3 55 h5 ♖c3 56 ♖a7 f6+ 57 ♔e4 ♖c4+ 58 ♔f5 ♖c5+ 59 ♔f4 ♖c4+ 60 ♔g3 ♖c3+ 61 ♔f2 ♖c2+ 62 ♔f3 ♖c3+ 63 ♔g2 ♖c2+ 64 ♔h3

♖c3+ 65 ♔h2 ♖c4 66 ♔h3 ½-½

**Schach 2.7 – Cray Blitz** 1 d4 ♘f6 2 c4 e6 3 ♘f3 d5 4 ♘c3 ♗e7 5 ♗g5 0-0 6 e3 ♘e4 7 ♗×e7 ♕×e7 8 ♘×e4 d×e4 9 ♘d2 e5 10 d×e5 ♕×e5 11 ♕c2 ♘a6 12 0-0-0 ♗g4 13 f3 e×f3 14 g×f3 ♗h5 15 ♗d3 f5 16 ♖he1 ♖ad8 17 ♗f1 ♕×h2 18 c5 ♕e5 19 c6 ♔h8 20 c×b7 ♘c5 21 ♘c4 ♕g3 22 ♕×f5 ♖×d1+ 23 ♖×d1 ♕×f3 24 ♕×f3 ♗×f3 25 ♖d4 ♗×b7 26 ♗e2 g6 27 ♘e5 ♖e8 28 ♘g4 ♖e4 29 ♔d2 h5 30 ♘f6 ♖×d4+ 31 e×d4 ♘e4+ 32 ♘×e4 ♗×e4 33 ♔e3 ♗d5 34 ♔f4 ♗×a2 35 ♗d3 ♔g7 36 ♔g5 ♗f7 37 ♗e4 a5 38 ♗c6 ♗g8 39 ♗e4 ♗h7 40 ♗d3 h4 41 ♗f1 ♗g8 42 ♗h3 ♗d5 43 ♗g4 ♗e4 44 ♔×h4 ♔f6 45 ♗d7 ♗f5 46 ♗c6 g5+ 47 ♔g3 0-1 (adjudicated)

*Round 5, June 15, 1986*

**Cray Blitz – Hitech** 1 d4 d5 2 c4 d×c4 3 ♘f3 ♘f6 4 e3 e6 5 ♗×c4 c5 6 ♕e2 a6 7 d×c5 ♗×c5 8 0-0 b5 9 ♖d1 ♕e7 10 ♗d3 e5 11 e4 ♘c6 12 ♘c3 ♗g4 13 ♗e3 ♖d8 14 h3 ♗×e3 15 ♕×e3 ♗×f3 16 ♕×f3 ♘d4 17 ♕g3 0-0 18 a4 b4 19 ♘d5 ♘×d5 20 e×d5 ♖×d5 21 ♗×a6 b3 22 ♕e3 ♖fd8 23 ♗c4 ♘c2 24 ♕e2 ♖c5 25 ♖×d8+ ♕×d8 26 ♖b1 ♘d4 27 ♕f1 ♕d7 28 ♖a1 ♕c6 29 ♗b5 ♘×b5 30 a×b5 ♕b7 31 ♖a3 g6 32 ♖×b3 ♕d5 33 ♖b4 ♖c2 34 b3 ♕d2 35 ♖c4 ♖b2 36 ♖e4 ♕d5 37 ♕c4 ♕d1+ 38 ♔h2 ♖×f2 39 ♖×e5 ♕d6 40 ♕c8+ ♔g7 41 ♕c5 ♕d2 42 ♖g5 ♖e2 43 ♖g4 43 ♕a2 44 ♕c3+ ♔g8 45 b6 ♕a8 46 ♕c7 ♕f8 47 b7 ♖e8 48 ♖c4 ♔g7 49 ♖c6 ♖b8 50 ♕c8 ♖×c8 51 b×c8♕ ♕b4 52 ♕c7 ♕×b3 53 ♕e5+ ♔h6 54 ♕f4+ ♔g7 55 ♕d4+ ♔h6 56 ♖b6 ♕c2 57 ♕f4+ ♔g7 58 ♕f6+ ♔h6 59 ♕×f7 ♕c8 60 ♖d6 1-0

**Bobby – Sun Phoenix** 1 e4 e6 2 b3 d5 3 ♗b2 d×e4 4 ♘c3 ♘f6 5 ♕e2 ♗e7 6 0-0-0 ♕d4 7 ♖e1 0-0 8 ♘×e4 ♕×e4 9 ♕×e4 ♘×e4 10 ♖×e4 ♘d7 11 ♘f3 ♗c5 12 d4 ♘f6 13 ♖h4 ♗e7 14 ♔b1 e5 15 h3 e4 16 ♘e5 ♗e6 17 ♗c4 ♗×c4 18 b×c4 c5 19 d5 e3 20 ♖f4 e×f2 21 ♖×f2 ♘e4 22 ♖f4 ♘d2+ 23 ♔a1 ♗d6 24 ♖e1 ♖ae8 25 ♖e2 f5 26 ♖h4 ♘e4 27 ♘d3 ♖e7 28 ♖e3 g6 29 ♘c1 ♖fe8 30 a4 h5 31 ♘b3 ♔h7 32 ♘a5 ♔h6 33 ♖f3 ♘g3 34 ♖b3 b6 35 ♘c6 ♖e1+ 36 ♔a2 g5 37 ♘×a7 ♖a8 38 ♘b5 ♖×a4+ 39 ♗a3 ♗e5 40 ♘d4 0-1

**Plymate – Lachex** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗a4 ♘f6 5 0-0 ♘×e4 6 d4 b5 7 ♗b3 d5 8 d×e5 ♗e6 9 ♗e3 ♘c5 10 ♕d2 ♘×b3 11 c×b3 h6 12 ♕c3 ♗d7 13 ♕c2 ♘b4 14 ♕d2 ♗f5 15 ♖c1 ♘d3 16 ♖d1 c5 17 ♕c2 g6 18 ♕e2 c4 19 ♘d4 ♗c8 20 b×c4 b×c4 21 b3 a5 22 ♘c3 ♗b7 23 e6 ♗b4 24 e×f7+ ♔×f7 25 ♘cb5 ♕d7 26 ♖ab1 ♔g8 27 ♕c2 ♕f7 28 a3 ♗f8 29 ♕e2 ♖h7 30 b×c4 d×c4 31 ♘c2 ♖a6 32 ♕d2 ♗c6 33 ♘cd4 ♗e8 34 ♕a2 g5 35 ♘c3 ♘e5 36 ♘de2 ♕e6 37 ♖d5 ♖f7 38 ♗d4 ♘g4 39 ♕×c4 ♗×a3 40 ♖b8 ♖c6 41 ♕d3 ♖f8 42 ♖b7 ♖f7 43 ♖×f7 ♗×f7 44 ♖d8+ ♗f8 45 ♕b5 ♖d6 46 ♖×f8+ ♔×f8 47 ♕b8+ ♔e7 48 ♗c5 ♗e8 49 ♕c7+ ♗d7 50 ♘b5 ♔f6 51 ♕d8+ ♔g6 52 ♘×d6 ♕d5 53 ♘c3 ♕e6 54 ♘de4 a4 55 ♗d4 a3 56 ♕f8 ♗c6 57 ♕g7+ ♔h5 58 f3 a2 59 ♘g3+ ♔h4 60 ♘f5+ ♕×f5 61 ♗f2+ ♔h5 62 f×g4+ ♔×g4 63 h3+ ♔h5 64 g4+ 1-0

**Rebel – Bebe** 1 d4 ♘f6 2 c4 c5 3 d5 e6 4 ♘c3 e×d5 5 c×d5 d6 6 e4 g6 7 ♗f4 a6 8 ♘f3 ♗g4 9 ♗e2 ♕b6 10 ♕d2 ♗g7 11 0-0 0-0 12 h3 ♗×f3 13 ♗×f3 ♘bd7 14 ♖ad1 ♖fe8 15 b3 ♘e5 16 ♗e2 ♕b4 17 ♕c2 ♖e7 18 ♗g3 ♖ae8 19 ♖fe1 g5 20 ♖f1 ♔h8 21 ♖c1 h5 22 f4 g×f4 23 ♖×f4 ♘g6 24 ♖f5 ♘×e4 25 ♘×e4 ♕×e4 26 ♕×e4 ♖×e4 27 ♗×h5 ♘e7 28 ♖×f7 ♗d4+ 29 ♔h1 ♘×d5 30 ♖×b7 ♖d8 31 ♗f3 ♖e3 32 ♗h4 ♘f6 33 ♖f7 ♖e6 34 ♗d5 ♘×d5 35 ♗×d8 ♘b4 36 a3 ♔g8 37 ♖cf1 ♘c2 38 ♖f8+ ♔g7 39 a4 d5 40 h4 ♘e3 41 ♖1f7+ ♔g6 42 ♖c7 ♘d1 43 ♖g8+ ♔f5 44 ♖f7+ ♔e4 45 g4 ♔d3 46 h5 ♖e1+ 47 ♔g2 ♘e3+ 48 ♔g3 ♗e5+ 49 ♔h4 ♖h1+ 50 ♔g5 ♖g1 51 ♔g6 ♖×g4+ 52 ♗g5 ♖b4 53 ♗×e3 ♔×e3 54 ♖e8 ♖g4+ 55 ♔h7 ♖e4 56 ♖a7 d4 57 ♖×a6 d3 58 ♖g6 d2 59 ♖g1 ♔f2 60 ♖eg8 ♖e1 61 ♖1g2+ ♔e3 62 ♖×d2 ♔×d2 63 ♖c8 ♗d4 64 ♖b8 ♖e6 65 ♖b7 ♔c2 66 b4 c4 67 b5 c3 68 ♖d7 ♔d3 69 b6 c2 70 b7 c1♕ 0-1

**Dutch – Ostrich** 1 ♘f3 d5 2 g3 ♘f6 3 ♗g2 ♗g4 4 c4 ♘bd7 5 c×d5 ♘×d5 6 d4 ♗×f3 7 ♗×f3 e6 8 0-0 ♗d6 9 e4 ♘5b6 10 e5 ♗b4 11 ♗×b7 ♖b8 12 ♗g2 0-0 13 ♕b3 ♗e7 14 ♖d1 c5 15 d×c5 ♗×c5 16 ♘c3 ♕c7 17 ♘b5 ♕×e5 18 ♗f4 ♕f5 19 ♗×b8 ♗×f2+ 20 ♔h1 ♖×b8 21 ♘×a7 ♘d5 22 ♘b5 ♘c5 23 ♕a3 ♘e3 24 ♘d4 ♕g4 25 ♗f3 ♕g5 26 ♖dc1 ♘d7 27 ♕d6 ♖d8 28 ♘c6 ♘f5 29 ♕d3 ♘×g3+ 30 h×g3 ♕×g3 31 ♗e4 ♕×d3 32 ♗×d3 ♖e8 33 ♔g2 ♗e3 34 ♖e1 ♗f4 35 b4 g5 36 a4 g4 37 ♖f1 e5 38 ♖ae1 ♖e6 39 b5 ♘f6 40 ♗c4 ♖e8 41 b6 ♔f8 42 ♖d1 ♘h5 43 b7 e4 44 ♖d7 1-0

**Nona – Cyrus 68K** 1 d4 e6 2 e4 d5 3 ♘c3 ♗b4 4 e5 ♘e7 5 ♗d2 ♘f5 6 ♘f3 ♗e7 7 ♗d3 ♘c6 8 ♗e3 ♘×e3 9 f×e3 ♘b4 10 ♗e2 c5 11 0-0 ♘c6 12 ♘a4 c4 13 ♘c3 ♔f8 14 e4 ♔e8 15 ♖f2 h6 16 e×d5 e×d5 17 ♕d2 ♗f5 18 ♘d1 ♘b4 19 ♘e3 ♗e4 20 b3 c×b3 21 a×b3 h5 22 ♘g5 ♗×g5 23 ♕×b4 ♕e7 24 ♗b5+ ♔f8 25 ♕×e7+ ♔×e7 26 ♖e1 ♔e6 27 c4 ♖af8 28 c×d5+ ♗×d5 29 ♘×d5 ♔×d5 30 e6 ♗e7 31 ♖×f7 ♖×f7 32 e×f7 ♗b4 3 ♖d1 ♖f8 34 ♗e2 h4 35 ♗c4+ ♔d6 36 ♖f1 ♗c3 37 ♖f4 ♖h8 38 f8♕+ 1-0

**Vaxchess – Chat** 1 d4 ♘f6 2 c4 g6 3 ♘c3 ♗g7 4 e4 d6 5 ♗e2 0-0 6 ♗e3 ♘c6 7 ♘f3 ♘g4 8 ♗g5 h6 9 ♗h4 ♗d7 10 h3 ♘f6 11 d5 ♘e5 12 ♕b3 ♘×f3+ 13 ♗×f3 b6 14 ♖d1 e5 15 d×e6 ♗c6 16 e5 ♗×f3 17 e×f7+ ♖×f7 18 g×f3 ♕e8 19 f4 d×e5 20 f5 g×f5 21 ♖g1 ♕e6 22 ♗×f6 ♕×f6 23 c5 ♖d8 24 ♖×d8+ ♕×d8 25 ♖g6 ♕h4 26 c×b6 c×b6 27 ♕d5 ♕×h3 28 ♖g3 ♕h4 29 ♕a8+ ♔h7 30 ♕d5 ♖c7 31 ♕f3 f4 32 ♕e4+ ♔h8 33 ♕a8+ ♔h7 34 ♕e4+ ♔h8 35 ♕a8+ ♔h7 36 ♕e4+ ½-½

**Mephisto – Schach 2.7** 1 d4 d5 2 c4 e6 3 g3 d×c4 4 ♕a4+ ♗d7 5 ♕×c4 c5 6 d×c5 ♗c6 7 ♘f3 ♘d7 8 ♗g5 ♕c7 9 ♕c3 f6 10 ♗e3 ♘e7 11 ♗g2 ♘d5 12 ♕d2 0-0-0 13 b4 ♘e5 14 ♗d4 ♘×f3+ 15 ♗×f3 e5 16 ♗b2 ♘f4 17 ♕e3 ♘g2+ 18 ♗×g2 ♗×g2 19 ♖g1 ♗c6 20 ♘d2 ♕d7 21 f3 ♕h3 22 ♕f2 g6 23 e4 ♗h6 24 ♘c4 ♕e6 25 ♘d6+ ♔b8 26 b5 ♗e8 27 ♗a3 ♗f7 28 b6 a6 29 ♖g2 ♗e8 30 ♕c2 ♗c6 31 ♕b3 ♕h3 32 ♖e2 ♖d7 33 ♖d1 ♖hd8 34 ♗b4 ♗b5 35 ♖f2 ♕h5 36 a4 ♗c6 37 ♖d3 ♗g7 38 ♗c3 ♗h6 39 ♕e6 ♗g7 40 a5 ♗b5 41 ♖d1 ♗a4 42 ♖a1 ♗c6 43 ♖a3 ♕g5

44 ♗d2 ♕h5 45 ♖d3 ♗b5 46 ♖b3 ♗a4 47 ♖c3 ♗c6 48 ♖c1 f5 49 ♖c2 f×e4 50 f×e4 ♗a4 51 ♖c1 ♗c6 52 h4 ♗h6 53 ♗×h6 ♕×h6 54 ♖c3 ♕g7 55 ♖cf3 ♖e7 56 ♕b3 ♖ed7 57 ♖f7 ♕h6 58 ♕c3 ♕h5 59 ♖×d7 ♗×d7 60 ♖f7 ♗c6 61 ♕d2 ♕g4 62 ♕g5 ♕×g5 63 h×g5 ♖h8 64 ♔d2 ♔a8 1-0

**Advance 68K – Enterprise** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗a4 ♘f6 5 0-0 ♘×e4 6 d4 e×d4 7 ♖e1 d5 8 ♘×d4 ♗d6 9 ♘×c6 ♗×h2+ 10 ♔h1 ♕h4 11 ♖×e4+ d×e4 12 ♕d8+ ♕×d8 13 ♘×d8+ ♔×d8 14 ♔×h2 ♗e6 15 ♗f4 b5 16 ♗b3 ♗×b3 17 a×b3 c6 18 ♘c3 f5 19 ♘e2 g6 20 ♘d4 ♔d7 21 ♖d1 ♔c8 22 ♘×c6 ♔b7 23 ♖d6 ♖hg8 24 ♘a5+ ♔c8 25 ♖c6+ ♔d8 26 ♖c7 g5 27 ♗d6 ♖g6 28 ♘c6+ ♔e8 29 ♖e7+ ♔f8 30 ♖a7+ ♖×d6 31 ♖×a8+ ♔g7 32 ♖a7+ ♔f8 33 ♖×a6 ♖d2 34 ♘b4 ♖×f2 35 ♖b6 e3 36 ♖×b5 h6 37 ♖b8+ ♔f7 38 ♖b7+ ♔f8 39 ♖h7 e2 40 ♘d3 ♖f1 41 ♖×h6 ♔f7 42 ♔g3 e1♕+ 43 ♘×e1 ♖×e1 44 b4 f4+ 45 ♔f3 ♖f1+ 46 ♔e2 ♖c1 47 ♖c6 ♖g1 48 ♔f3 ♖b1 49 b3 ♖g1 50 b5 ♖e1 51 ♔g4 ♖e2 52 ♔×g5 ♖e5+ 53 ♔×f4 ♖×b5 54 ♖c7+ ♔e6 55 g4 ♖b4+ 56 ♔f3 ♖b5 57 ♖c6+ ♔d5 58 ♖c4 ♔d6 59 ♖d4+ ♔e5 60 ♖e4+ ♔d6 61 b4 ♖b7 62 c3 ♖f7+ 63 ♔e3 ♖f1 64 c4 ♖e1+ 65 ♔d4 ♖d1+ 66 ♔c3 ♖c1+ 67 ♔b2 ♖g1 68 ♔b3 ♖g3+ 69 ♔a4 ♔c6 70 ♖d4 ♖g2 71 c5 ♖g1 72 ♖d6+ ♔c7 73 ♖g6 ♖g2 74 ♔b5 ♔d7 75 c6+ ♔c7 76 ♖g7+ ♔d6 77 ♔b6 1-0

**BCP – Awit** 1 e4 e5 2 ♘f3 ♘c6 3 d4 e×d4 4 ♘×d4 ♕f6 5 ♗e3 ♘h6 6 ♘×c6 ♕×c6 7 ♗d3 b6 8 0-0 ♗c5 9 ♘c3 0-0 10 ♕h5 ♗×e3 11 f×e3 ♗b7 12 ♔h1 ♕e6 13 ♕g5 c5 14 b3 ♖fe8 15 ♗c4 ♕e5 16 ♕×e5 ♖×e5 17 ♖ad1 ♗c6 18 ♖d6 ♖f8 19 h3 g6 20 ♗d5 ♖c8 21 ♗×d6 d×c6 22 ♖ff6 ♔g7 23 ♖×c6 ♖×c6 24 ♖×c6 ♘g8 25 ♖c7 ♘f6 26 ♖×a7 ♘×e4 27 ♖d1 ♖d5 28 ♘b2 ♘g3+ 29 ♔g1 ♖d2 30 ♘c4 ♖×c2 31 ♘×b6 ♘f5 32 ♘c4 ♘h4 33 g4 ♘f3+ 34 ♔f1 g5 35 a3 ♘d2+ 36 ♘×d2 ♖×d2 37 ♖c7 ♖d1+ 38 ♔e2 ♖c1 39 ♔d2 ♖b1 40 ♖×c5 ♖×b3 41 ♖×g5+ ♔f6 42 ♖f5+ ♔e6 43 a4 ♖a3 44 a5 ♖b3 45 h4 f6 46 ♖f4 ♖a3 47 ♖f5 ♖b3 48 a6 ♖a3 49 ♖h5 ♖×a6 50 ♖×h7 ♖a2+ 51 ♔d3 ♖f2 52 ♖c7 ♖h2 53 ♖c6+ ♔f7 54 h5 ♖h4 55 ♖c4 ♔g7 56 ♖b4 ♔h6 57 ♔e2 ♔g5 58 ♔f3 ♖h3+ 59 ♔f2 f5 60 g×f5 ♔×f5 61 ♖f4+ ♔g5 62 ♔e2 ♔×h5 63 ♔d3 ♔g5 64 ♖b4 ♔f5 65 ♖c4 ♖h7 66 e4+ ♔e5 67 ♖c5+ ♔d6 68 ♖f5 ♖h2 69 e5+ ♔d5 70 ♔e3 ♖h4 71 ♔f3 ♖h3+ 72 ♔f4 ♖h4+ 73 ♔g5 ♖h8 74 ♔g4 ♔e6 75 ♖g5 ♖f8 76 ♖h5 ♖f1 77 ♔g3 ♖f5 78 ♖×f5 ♔×f5 79 e6 ♔×e6 ½-½

**Shess – Rex** 1 e4 e5 2 ♘f3 f5 3 ♘×e5 ♕f6 4 ♘c4 f×e4 5 ♘c3 ♕f7 6 ♘×e4 d5 7 ♘e5 ♕f5 8 ♗b5+ c6 9 ♗e2 ♕×e4 10 ♘f3 ♗f5 11 c3 ♕c2 12 ♕×c2 ♗×c2 13 0-0 ♗d6 14 d4 ♘f6 15 ♗e3 0-0 16 ♖ae1 ♘bd7 17 ♔h1 b5 18 b3 ♖fe8 19 c4 b×c4 20 b×c4 d×c4 21 ♗×c4+ ♔h8 22 ♘g5 ♗g6 23 f4 ♘d5 24 ♗×d5 c×d5 25 f5 ♗b4 26 f×g6 ♗×e1 27 ♘f7+ ♔g8 28 g×h7+ ♔×h7 29 ♖×e1 ♔g8 30 ♘g5 ♖ac8 31 ♔g1 ♖c2 32 a3 ♖c3 33 ♔f2 ♖×a3 34 h4 ♘f6 35 ♔f3 ♖b3 36 g3 ♘h5 37 ♘h3 ♘×g3 38 ♘f4 ♘f5 39 ♘×d5 ♖d3 40 h5 ♘×d4+ 41 ♔f4 ♘e6+ 42 ♔g4 ♖×d5 43 ♗×a7 ♖a8 44 ♖×e6 ♖×a7 45 ♖e8+ ♔h7 46 ♖e4 ♖aa5 47 ♔f3 ♖×h5 48 ♔e3 ♖a3+ 49 ♔f4 ♖h4+ 50 ♔e5 ♖a5+ 51 ♔d4 ♖a4+ 52 ♔d5 ♖h×e4 53 ♔c5 ♖a5+ 54 ♔d6 ♖d4+ 55 ♔e6 ♔g8 56 ♔e7 ♖e5# 0-1

# 9.3 Blitz Games between Computers and Human Players

Blitz (or speed) chess usually narrows the difference in playing strength between computer programs and superior human players. This is because chess played at high speed tends to be much more tactical in nature than serious tournament games, and computers play better tactically (relative to humans) than they do strategically. For one thing, human players are far more likely than computer programs to overlook shallow combinations or even leave pieces en prise.

The rules for blitz games between computer programs and humans usually give the human player 5 minutes for all of his moves, while the program is constrained to play at an average of 5 seconds per move. If the program fails to deliver mate by move 60 it loses on time. Most if not all of the games in this section were played under these conditions. There is one game which I would very much have liked to include in this collection but which is not available for publication. The West German Grandmaster Robert Hübner once played two blitz games against the Northwestern University program, and lost the first game (he won the second). At the time, Hübner was rated in the world's top 10.

**Chess 4.6 — Stean (GM)**
**London, September 1977**

1 e4 b6 2 d4 ♗b7 3 ♘c3 c5 4 dc bc 5 ♗e3 d6 6 ♗b5+ ♘d7 7 ♘f3 e6 8 0-0 a6 9 ♗xd7+ ♕xd7 10 ♕d3 ♘e7 11 ♖ad1 ♖d8 12 ♕c4 ♘g6 13 ♖fe1 ♗e7 14 ♕b3 ♕c6 15 ♔h1 0-0 16 ♗g5 ♗a8 17 ♗xe7 ♘xe7 18 a4 ♖b8 19 ♕a2 ♖b4 20 b3 f5 21 ♘g5 fe 22 ♘cxe4! ♖xf2 23 ♖xd6! ♕xd6 24 ♘xd6 ♖xg2 25 ♘ge4 ♖g4 26 c4! ♘f5 27 h3 ♘g3+ 28 ♔h2 ♖xe4 ♕f2! h6 30 ♘xe4 ♘xe4 31 ♕f3 ♖b8 32 ♖xe4 ♖f8 33 ♕g4 ♗xe4 34 ♕xe6+ ♔h8 35 ♕xe4 ♖f6 36 ♕e5 ♖b6 37 ♕xc5 ♖xb3 38 ♕c8+ ♔h7 39 ♕xa6 1-0

**Chess 4.6 — Berliner (IM)**
**Pittsburgh, March 1977**

1 f4 g6 2 ♘c3 ♗g7 3 e4 c5 4 ♘f3 ♘c6 5 ♗c4 e6 6 0-0 ♘ge7 7 e5 d6 8 ♘e4 de 9 fe ♘xe5 10 ♗b5+ ♘7c6 11 d3 0-0 12 ♘xe5 ♘xe5 13 ♗e3 a6 14 ♗a4 b5 15 ♗xb5!? ab 16 ♗xc5 ♖e8 17 d4 ♘c6 18 c3 e5 19 ♘d6 ed 20 ♕f3 ♘e5?? 21 ♕xa8 d3 22 ♘xe8 ♕xe8 23 a4 h5 24 ab ♘g4 25 ♖a7 ♕e5 26 ♕xc8+ ♔h7 27 ♕c7 ♕e2 28 ♕xf7 1-0

**Chess 4.6 — Day (IM)**
**Toronto, August 1977**

1 e4 c5 2 ♘f3 ♘c6 3 ♗b5 d6 4 0-0 ♗d7 5 c3 ♘f6 6 ♖e1 a6 7 ♗a4 b5 8 ♗c2 ♗g4 9 h3 ♗xf3 10 ♕xf3 g6 11 d3 ♗g7 12 ♗f4 ♘d7 13 ♘d2 0-0 14 ♗b3 ♖c8 15 a4 ♘ce5 16 ♕e2 ♕b6 17 ab ab 18 ♗e3 ♘c6 19 ♘f3 ♕b7 20 ♘g5 ♘ce5 21 f4 c4 22 ♖a7! ♕c6 23 fe cb 24 d4! d5 25 ed ♕xd5 26 ♖ea1 f6 27 ef ef 28 ♕g4! f5 29 ♕f3 ♕xf3 30 ♘xf3 ♘f6 31 ♖1a3 ♘d5 32 ♗f2 b4 33 ♖xb3 bc 34 ♖bb7 ♗h6 35 bc ♖xc3 36 ♖xh7 ♖c1+ 37 ♔h2 ♗f4+ 38 ♗g3 ♗xg3+ 39 ♔xg3 ♖f6 40 ♘e5 ♖c3+ 41 ♔h2 ♘f4 42 ♖ag7+ ♔f8 43 ♘d7+ ♔e8 44 ♖g8+ ♖f8 45 ♖xf8 mate

**Chess 4.6 — Vranesić (IM)**
**London, September 1977**

1 e4 c5 2 ♘f3 e6 3 d4 cd 4 ♘xd4 ♘f6 5 ♘c3 ♗b4 6 f3 0-0 7 ♗f4 a6 8 a3 ♗a5 9 ♗d6 ♖e8 10 b4 ♗c7 11 ♘b3 ♗xd6 12 ♕xd6 ♕e7 13 ♕xe7 ♖xe7 14 e5 ♘d5 15 ♔d2 ♘xc3 16 ♔xc3 ♘c6 17 ♖e1 b5 18 ♗d3 ♖b8 19 ♔b2 d5? 20 ed ♖d7 21 ♖d1 ♘e5 22 ♘c5 ♖d8 23 f4 ♘g4 24 d7! ♗b7 25 ♖d2 ♗d5 26 ♘xa6 ♖b6 27 ♘c5 ♘f6 28 a4! ba 29 c4 ♗a8 30 ♔c3 g6 31 ♗c2 a3 32 ♔b3 ♗c6 33 ♔xa3 ♔f8 34 ♗a4 ♗xa4 35 ♔xa4 ♔e7 36 ♖hd1 ♖a8+ 37 ♔b3 ♖d8 38 b5 h6 39 ♔b4 ♖bb8 40 g3 ♖a8 41 b6 ♖ab8 42 ♔b5 g5 43 fg hg 44 ♖d6 ♘g4 45 ♘a6 ♖a8 46 b7 ♖a7 47 b8=♕♖xb8+ 48 ♘xb8 ♖b7+ 49 ♖b6 ♖xd7 50 ♖xd7+ ♔f6 51 c5 ♘e5 52 c6 ♔g6 53 c7 ♘xd7 54 ♘xd7 ♔f5 55 c8=♕ ♔g6 56 ♕g8+ ♔f5 57 ♕h7+ ♔g4 58 ♘e5 mate

**Chess 4.6 — Vranesić (IM)**
**London, September 1977**

1 e4 c5 2 ♘f3 e6 3 d4 cd 4 ♘xd4 ♘c6 5 ♘xc6 bc 6 ♗e3 ♘f6 7 ♘c3 d5 8 e5 ♘d7 9 f4 ♗e7 10 ♗e2 0-0 11 0-0 a5 12 ♗d3 f5 13 ef ♘xf6 14 ♕f3 ♖b8 15 ♗d4 c5 16 ♗e5 c4 17 ♗xb8 ♕b6+ 18 ♔h1 ♕xb8 19 ♗e2 ♕xb2 20 ♖fc1 ♕b6 21 ♖ab1 ♕c7 22 ♕g3 ♗c5 23 ♖e1 ♗d7 24 ♖f1 ♗d6 25 ♘b5 ♗xb5 26 ♖xb5 ♘e4 27 ♕h3 ♖f6 28 ♗g4 ♕d7? 29 ♖xd5 ♕e7 30 ♖xa5 h6 31 ♕e3 ♘c5 32 ♗e2 ♕c7 33 ♖a8+ ♔h7 34 g3 e5 35 fe ♖xf1+ 36 ♗xf1 ♗xe5 37 a4 ♗f6 38 ♗g2 ♘d7 39 ♖a7 ♕d6 40 ♕e4+ g6 41 ♕d5 Black lost on time 1-0

**Chess 4.7 – R. Byrne (GM)**

1 e4 c5 2 ♘f3 d6 3 d4 cd 4 ♘×d4 ♘f6 5 ♘c3 a6 6 ♗e2 e5 7 ♘b3 ♗e7 8 0-0 0-0 9 ♗e3 ♘c6 10 f4 b5 11 fe ♘×e5 12 ♕d4 ♗b7 13 a4 ♘c6 14 ♕d2 b4 15 ♘d5 ♘×e4 16 ♕d3 ♘c5 17 ♕c4 ♘×b3 18 ♗b6 ♕d7 19 ♕×b3 ♖ac8 20 ♗d3 g6 21 ♘×b4 ♘×b4 22 ♕×b4 ♕c6 23 ♖f2 ♖ce8 24 ♖af1 d5 25 ♕a5 ♗d6 26 c3 f5 27 ♖f3 f4 28 ♔h1 ♖e4 29 b4 ♗e5 30 b5 ab 31 ab ♕e6 32 ♕a7 ♖f7 33 ♗f2 g5 34 ♕a1 g4 35 ♕a2 ♖c4 36 ♗×c4 dc 37 ♖d3 g3 38 ♖d8+ ♔g7 39 ♖c1 f3 40 ♗×g3 ♗×g3 41 ♕a7 f2 42 ♕d4+ ♗e5 43 ♕d1 ♕h3 44 ♖g8+ ♔×g8 45 ♕d8+ ♖f8 46 ♕g5+ ♗g7 47 ♖f1 ♕d3 48 ♕×g7+ ♔×g7 49 ♖×f2 ♕d1+ 50 ♖f1 ♖×f1 mate 0-1

**Belle – Kopec**
**San Francisco 1984**

1 e4 e6 2 d4 c5 3 c3 d5 4 ed ♕×d5 5 ♘f3 ♘c6 6 ♘a3 cd 7 ♘b5 ♕d8 8 ♗f4 e5 9 ♘×e5 ♕e7 10 cd f6 11 ♕h5+ g6 12 ♘d6+ ♕×d6 13 ♘×g6 ♕e6+ 14 ♘e5+ ♔d8 15 0-0-0 fe 16 d5 ♕f5 17 dc+ ♔c7 18 ♗×e5+ ♔×c6 19 ♕e8+ ♗d7 20 ♕×d7+ ♕×d7 21 ♗b5+ ♔×b5 22 ♖×d7 ♔c6 23 ♖hd1 ♘f6 24 ♗×f6 ♖g8 25 g3 ♖g6 26 ♗e5 ♗c5 27 ♖c7+ ♔b6 28 ♖dd7 ♖b8 29 ♖×c5 1-0

*February 21st 1978*

**Short – Chess 4.6**

1 d4 ♘f6 2 c4 c5 3 d5 e6 4 ♘c3 ed 5 cd d6 6 e4 g6 7 f4 ♗g7 8 ♗b5+ ♗d7 9 e5 ♘h5 10 ♘f3 0-0 11 0-0 ♗g4 12 ed ♗d4+ 13 ♔h1 ♕×d6 14 ♕d2 ♗×f3 15 ♖×f3 ♘d7 16 ♗c4 ♘b6 17 b3 ♕d7 18 ♗b2 ♖fe8 19 f5 ♗f6 20 fg hg 21 ♖af1 ♕g4 22 ♖×f6 ♘×f6 23 ♖×f6 ♘×c4 24 bc ♕d4 25 ♘b1 ♕×b2 0-1

**Short – Chess 4.6**

1 e4 e5 2 ♘f3 ♘f6 3 ♘c3 ♗b4 4 ♘×e5 0-0 5 ♘d3 ♗×c3 6 dc ♘×e4 7 ♗e2 d6 8 0-0 ♗f5 9 f3 ♘f6 10 ♘f4 ♘c6 11 ♗d3 ♗×d3 12 cd ♖e8 13 d4 ♘d7 14 b3 ♔h8 15 ♗b2 ♕g5 16 ♕d2 ♘f6 17 ♖ad1 ♘e7 18 c4 ♘g6 19 ♘×g6+ ♕×g6 20 d5 ♘d7 21 ♕d4 f6 22 ♖de1 ♕c2 23 ♕c3 ♕g6 24 ♖e6 ♘c5 25 ♖ee1 ♘d3 26 ♖d1 ♖e2 27 ♖d2 ♕×g2 mate

**Chess 4.6 – Short**

1 f3 e5 2 ♘c3 d5 3 d4 ed 4 ♕×d4 ♗e6 5 ♕a4+ ♘d7 6 ♗e3 c5 7 ♗f4 a6 8 ♗g3 b5 9 ♕f4 ♘gf6 10 e3 b4 11 ♘ce2 ♗e7 12 ♕c7 0-0 13 ♕×d8 ♖f×d8 14 a3 a5 15 ab ab 16 ♖×a8 ♖×a8 17 ♘c1 ♖a1 18 ♔d2 c4 19 ♘ge2 ♘c5 20 ♘d4 c3+ 21 bc bc+ 22 ♔d1 ♘a4 23 ♗b5 ♗a3 24 ♘b3 ♘b2+ 25 ♔e2 ♘c4 26 ♘×a1 1-0

**Chess 4.5 – Edward Lasker**

1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 f5 4 ♘c3 ♘d4 5 ef ♘f6 6 ♘×d4 ed 7 ♘e2 c6 8 ♗c4 d5 9 ♗d3 ♗d6 10 ♘×d4 0-0 11 0-0 c5 12 ♘e6 ♗×e6 13 fe c4 14 ♗f5 g6 15 ♗h3 ♘e4 16 d3 ♘×f2 17 ♖×f2 ♕b6 18 d4 ♗×h2+ 19 ♔f1 ♖×f2+ 20 ♔×f2 ♖f8+ 21 ♔e2 c3 22 bc ♕c7 23 ♗a3 ♖f4 24 ♕d3 ♖e4+ 25 ♔d1 ♗g3 26 ♕f3 ♖e1+ 27 ♔d2 ♕b8 28 ♕f7+ ♔h8 29 ♗f8 ♕f4+ 30 ♕×f4 1-0

# 9.4 Simultaneous Games between Computer Programs and Human Players

The first game in this section was played in a simultaneous exhibition on May 6th 1978, in Minnesota, in which three times U.S. Champion Walter Browne took on 44 opponents at the same time. The next two games were played in Paris on November 24th 1977, when Chess 4.6 played simultaneously against ten celebrities.

**Browne – Chess 4.6** 1 d4 ♘f6 2 c4 c5 3 ♘f3 cd 4 ♘×d4 e5 5 ♘b5 ♗c5 6 ♘1c3 0-0 7 e3 d6 8 ♗e2 a6 9 ♘a3 ♘c6 10 ♘c2 ♗f5 11 0-0 ♕d7 12 b3 ♔h8 13 ♗b2 ♖g8 14 ♘a4 ♗a7 15 ♗a3 h6 16 ♖c1 ♖ad8 17 ♘b4 ♘×b4 18 ♗×b4 ♕c7 19 ♕e1 ♗c5 20 ♗f3 ♗d3 21 ♗×c5 dc 22 ♗e2 ♗f5 23 f3 e4 24 f4 ♗d7 25 ♘c3 ♕a5 26 ♕h4 ♗c6 27 ♖c2 b5 28 g4 b4 29 ♘d1 ♖d6 30 ♘f2 ♖gd8 31 ♖d1 ♖×d1+ 32 ♗×d1 ♖d6 33 ♕g3 ♕d8 34 ♖c1 ♖d2 35 g5 hg 36 fg ♘h7 37 g6 fg 38 ♕×g6 ♕h4 39 ♕f5 ♗d7 40 ♕f4 ♕×f4 41 ef e3 42 ♘e4 e2 43 ♗×e2 ♖×e2 44 ♘×c5 ♗c8 45 ♖d1 ♖e8 46 a3 ba 47 ♖a1 g5 48 fg ♖e5 49 b4 a5 50 ♘d3 ♖×g5+ 51 ♔f2 ab 52 ♘×b4 ♖a5 53 ♔e3 ♗e6 54 ♔d4 ♘g5 55 ♘c2 a2 56 ♘b4 ♖a4 57 ♔c5 ♘e4+ 58 ♔b5 ♗d7+ 59 ♘c6 ♘c3+ 60 ♔c5 ♗×c6 61 ♔×c6 ♖×c4+ 62 ♔d6 ♖d4+ 63 ♔e5 ♖d1 0-1

**Roger Vadim – Chess 4.6** 1 d4 d5 2 e3 ♘f6 3 h3 ♘c6 4 ♘f3 ♗f5 5 ♗d3 ♘e4 6 c3 e6 7 0-0 ♗d6 8 ♕e2 0-0 9 a3 ♕f6 10 ♖d1 h6 11 c4 dc 12 ♗×c4 ♘g5 13 d5 ♘×f3+ 14 ♕×f3 ♘e5 15 ♕e2 ♘×c4 16 ♕×c4 ♕g6 17 ♗d2 ♗e4 18 ♕f1 ♗c2 19 ♖c1 ♗d3 20 ♕e1 ed 21 ♘c3 c6 22 ♘a4 ♗f5 23 ♔h1 ♗×h3 24 gh ♕e4+ 25 ♔g1 ♕×a4 26 b4 f6 27 ♕e2 a5 28 ♕g4 ♗e5 29 e4 ♗×a1 30 ♗×h6 f5 31 ef ♕×a3 32 ♖e1 ♕×b4 33 ♖e7 ♕×e7 34 resigns 0-1

**Chess 4.6 – M. Chandon Moet** 1 e4 c6 2 c4 d5 3 ed cd 4 d4 ♘f6 5 ♘c3 g6 6 ♕b3 ♗g7 7 cd 0-0 8 ♗e2 ♘bd7 9 ♗f3 ♘b6 10 ♗g5 ♖e8 11 ♘ge2 e6 12 de ♖×e6 13 0-0 ♖d6 14 ♕b4 ♗f8 15 ♘e4 ♘bd5 16 ♘×f6+ ♘×f6 17 ♗×b7 ♖b8 18 f3 ♖×b7 19 ♕d2 ♕b6 20 ♖ac1 ♗d7 21 b3 ♘e8 22 ♗h6 ♗g7 23 ♖fd1 ♕b4 24 ♗f4 ♕×d2 25 ♖×d2 ♖×c6 26 ♖×c6 ♗×c6 27 d5 ♗d7 28 ♘d4 ♘c7 29 ♘e2 ♘b5 30 ♖d3 ♗f5 31 ♖e3 h5 32 ♖e8+ ♔h7 33 ♖b8 ♘d4 34 ♘g3 ♖d7 35 d6 h4 36 ♘×f5 ♘×f5 37 ♖b4 ♘×d6 38 ♗×d6 ♖×d6 39 ♖×h4+ ♔g8 40 ♖e4 ♖d1+ 41 ♔f2 ♖d2+ 42 ♖e2 ♗d4+ 43 ♔e1 ♖×e2+ 44 ♔×e2 ♔g7 45 ♔d3 ♗g1 46 h3 ♔f6 47 ♔e4 ♗h2 48 f4 ♔e6 49 g4 f6 50 f5+ gf+ 51 gf+ ♔f7 52 h4 ♔g7 53 ♔f3 ♗d6 54 ♔g4 ♔h6 55 h5 a5 56 ♔h4 ♗b4 57 ♔g4 ♗e1 58 a3 ♗d2 59 ♔h4 ♗c1 60 b4 a4 61 b5 ♗×a3 62 b6 ♗d6 63 ♔g4 a3 64 ♔f3 a2 65 ♔e4 a1=♕ 66 ♔d5 ♕a5+ 67 ♔×d6 ♕×b6+ 68 ♔d5 ♔×h5 69 ♔c4 ♔g5 70 ♔d3 ♔×f5 71 ♔c3 ♔e4 72 ♔c4 ♕d4+ 73 ♔b5 ♔d5 74 ♔a6 ♔c6 75 ♔a5 f5 76 ♔a6 ♕b6 checkmate 0-1

# 9.5 Games Played by Computer Programs in Human Tournaments

As chess programs became stronger and stronger it was inevitable that one day they would be admitted to tournaments in which up to now only humans (or near humans) had taken part. At first there was a lot of criticism of the idea, and in the USA an organisation was spawned with the aim of preventing computers from participating alongside human players. With time this mood has mellowed somewhat, and by the mid-1980s it was not at all uncommon to see a chess program playing in important human events.

The greatest scope for computers to play in human competitions comes from the large number of Swiss system tournaments that take place every year wherever chess is popular. Computer programs have won state championships in the USA (Chess 4.5 won the 1977 Minnesota Open, and Cray Blitz the 1981 Mississippi State Championship). They have also competed in official tournaments ratified by FIDE (the World Chess Federation) – a commercially available chess computer took part in a women's "zonal" tournament, which is the first eliminating contest for the women's World Championship, and other programs have played in the Asian Cities Team Championship, the Commonwealth Championship and on many occasions the U.S. Open.

The games in this section are a small selection of those available from human chess tournaments.

**Northwestern University winter quarter tournament, 1974**

**Friedman – Chess 4.0** 1 d4 d5 2 c4 dc 3 ♘f3 ♘f6 4 e3 e6 5 ♗xc4 c5 6 0-0 a6 7 ♕e2 b5 8 ♗b3 ♗b7 9 ♖d1 ♘bd7 10 ♘c3 ♗d6 11 e4 cd 12 ♘xd4 ♕b8 13 ♘f3 b4 14 ♘a4 ♗xe4 15 h3 0-0 16 ♗e3 ♕b5 17 ♕xb5 ab 18 ♘b6 ♘xb6 19 ♗xb6 ♗d5 20 ♗c2 ♖fc8 21 ♖d2 ♗f4 22 ♗e3 ♗xe3 23 fe ♗xa2 24 ♖ad1 ♘d5 25 ♔f2 ♖xc2?! 26 ♖xc2 ♗c4 27 ♖dc1 f5 28 ♘e5 b3 29 ♘xc4 bxc4 30 ♘a3 b4 31 ♘xc2 ♖c8 32 b3 ♘c3 33 ♔f3 ♘a2 34 ♖d1 ♖xc2 35 ♖d6 ♔f7 36 g3 ♘c1 37 ♖d4 ♘xb3 38 ♖xb4 ♘d2+ 39 ♔f4 ♔f6 40 ♖b6 g5 mate 0-1

**Paul Masson Class "B" tournament, July 1976**

**Chess 4.5 – Chu (1784)** 1 e4 e6 2 d4 b6 3 ♘f3 ♗b7 4 ♘c3 ♗b4 5 ♗d3 ♘f6 6 ♗g5 h6 7 ♗xf6 ♕xf6 8 0-0 g5 9 ♕d2 ♘c6 10 a3 ♗e7 11 ♘b5 ♗d8 12 e5 ♕g7 13 ♕e3 a6 14 ♘c3 g4 15 ♘e1 ♗g5 16 f4 ♗e7 17 f5 ef 18 ♗xf5 ♘xd4 19 ♗xd7+ ♔xd7 20 ♕xd4+ ♔c8 21 ♘d3 ♔b8 22 ♕d7 ♕g5 23 ♖xf7 ♕e3+ 24 ♔h1 ♗d8 25 ♕xg4 ♔a7 26 e6 ♗g5 27 ♖e1 ♖hg8 28 ♖xe3 ♗xe3 29 ♖g7 ♖af8 30 h4 ♖f1+ 1-0

**Wesley (1742) – Chess 4.5** 1 e4 ♘c6 2 d4 d5 3 e5 f6 4 f4 ♗f5 5 ♘f3 e6 6 a3 ♘h6 7 ♗d3 fe 8 fe ♗e4 9 0-0 ♗xf3 10 ♖xf3 ♘xd4 11 ♗g6+ hg 12 ♕xd4 c6 13 ♕d3 ♘f5 14 g4 ♗c5+ 15 ♔f1 ♘h4 16 b4 ♘xf3 17 ♕xg6+ ♔d7 18 bc ♕f8 19 ♗f4 ♖xh2 20 ♘c3 ♘xe5 21 ♕g5 ♘f7 22 ♕g6 ♖h1+ 23 ♔g2 ♖xa1 24 ♘e2 ♖xa3 25 ♘d4 ♖e8 26 g5 ♘h8 27 ♘xe6 ♖xe6 28 ♕h7 ♕xf4 29 ♕xg7+ ♖e7 30 ♕xh8 ♖e2+ 31 ♔h1 ♕f1 mate 0-1

**Buljan (1751) – Chess 4.5** 1 d4 d5 2 e3 ♘f6 3 ♗d3 c5 4 c3 ♘c6 5 f4 ♗g4 6 ♘f3 e6 7 ♘bd2 ♗d6 8 0-0 c4 9 ♗c2 0-0 10 ♕e1 ♕c7 11 ♘e5 ♔h8 12 ♕h4 ♗f5 13 ♗xf5 ef 14 ♖f3 ♘e7 15 ♖h3 h6 16 ♘df3 a5 17 ♘xf7+ ♖xf7 18 ♘g5 ♘g6 19 ♘xf7+ ♕xf7 20 ♕g5 ♘e4 21 ♕h5 ♖e8 22 ♖f3 a4 23 h3 ♘f6 24 ♕xf5 ♘h4 25 ♕c2 ♘xf3+ 26 gf a3 27 ba ♖a8 28 ♖b1 ♗xa3 29 ♗xa3 ♖xa3 30 ♖b6 ♕e8 31 ♖xb7 ♕xe3+ 32 ♔g2 ♖xc3 33 ♖b8+ ♘g8 34 ♕f2 ♕xf4 35 ♖b2 ♖d3 36 a4 ♕xd4 37 ♕xd4 ♖xd4 38 ♔f2 ♖d1 39 ♖a2 d4 40 a5 ♖h1 41 ♔g2 ♖e1 42 a6 ♖e8 43 ♖a4 d3 44 ♖xc4 d2 45 ♖d4 ♖e6 46 a7 ♖a6 47 ♖d7 ♖xa7 48 ♖xd2 ♖a4 49 ♖d8 ♔h7 50 ♖d6 ♘f6 51 ♖b6 ♘h5 52 ♖c6 ♖a2+ 53 ♔g1 ♘f4 54 h4 ♖g2+ 55 ♔h1 ♖g3 56 ♖c3 ♔g6 57 ♔h2 ♖h3+ 58 ♔g1 ♖xh4 59 ♔f2 ♖h2+ 60 ♔f1 h5 61 ♖c4 ♘d3 62 ♔g1 ♖h3 63 ♔g2 ♖xf3 64 ♖c6+ ♖f6 65 ♖c3 ♘f4+ 66 ♔h2 h4 67 ♖c5 ♔h6 68 ♖a5 g5 69 ♖b5 g4 70 ♖b8 g3+ 71 ♔g1 ♖c6 72 ♖b1 h3 73 ♖a1 ♖f6 74 ♖d1 ♘e2+ 75 ♔h1 ♖g6 76 ♖d6 g2+ 77 ♔h2 g1=♕+ 78 ♔xh3 ♕h1 mate 0-1

**Chess 4.5 – Regan (1693)** 1 e4 c5 2 ♘f3 ♘c6 3 d4 cd 4 ♘xd4 g6 5 ♘c3 ♗g7 6 ♗e3 ♘f6 7 ♗c4 ♕a5 8 0-0 0-0 9 ♘b3 ♕c7 10 f3 d6 11 ♘b5 ♕b8 12 ♕e2 a6 13 ♘c3 ♗d7 14 a4 ♘b4 15 ♗b6 ♕c8 16 ♘a5 ♘c6 17 ♘xc6 ♕xc6 18 a5 ♖ac8 19 ♗d5 ♘xd5 20 ed ♕c4 21 ♕xe7 ♗b5 22 ♕xd6 ♗d4+ 23 ♗xd4 ♕xd4+ 24 ♖f2 ♖c6 25 ♕a3 ♖c4 26 ♖d1 ♖e8 27 ♖xd4 ♖e1+ 28 ♖f1 ♖xf1+ 29 ♔xf1 ♖xc3+ 30 ♔f2 ♖xa3 31 ba 1-0

**Chess 4.5 – Arnold (1704)** 1 e4 c5 2 ♘f3 d6 3 d4 cd 4 ♘xd4 ♘f6 5 ♘c3 a6 6 ♗e2 e5 7 ♘b3 ♗e7 8 0-0 g5 9 ♗xg5 ♖g8 10 f4 ef 11 ♗xf4 ♗h3 12 ♗g3 ♗g4 13 ♘d5 h5 14 ♗xg4 ♘xg4 15 ♕f3 f6 16 h3 h4 17 ♗xh4 ♘e5 18 ♕h5+ ♔d7 19 ♗xf6 ♘bc6 20 ♖ad1 ♔c8 21 ♘xe7+ ♘xe7 22 ♗xe5 1-0

**Minnesota Open Championship, February 1977**

**Stenberg (1969) – Chess 4.5** 1 d4 ♘f6 2 ♘f3 d5 3 c4 dc 4 ♘c3 a6 5 a4 ♘c6 6 ♗g5 ♗g4 7 e3 ♘a5 8 h3 ♗e6 9 ♗×f6 gf 10 d5 ♗d7 11 ♘d4 ♗g7 12 ♕h5 ♔f8 13 ♖d1 ♕e8 14 ♘f5 b5 15 ♖d4 ♗×f5 16 ♕×f5 b4 17 ♘d1 ♕×a4 18 ♗×c4 ♘×c4 19 ♖×c4 ♖d8 20 ♖×c7 h5 21 ♖c8 ♔e8 22 0-0 ♕d7 23 ♖×d8+ ♔×d8 24 ♕×d7+ ♔×d7 25 e4 ♖c8 26 ♘e3 f5 27 b3 fe 28 ♘c4 ♖g8 29 ♖d1 f5 30 d6 e6 31 ♘b6+ ♔c6 32 ♘c4 ♖d8 33 ♘a5+ ♔d7 34 ♘c4 ♗c3 35 ♔f1 ♖g8 36 ♘d2 h4 37 ♘c4 f4 38 ♘d2 ♗×d2 39 ♖×d2 f3 40 gf ef 41 ♖d3 ♖f8 42 ♔e1 ♖f4 43 ♔d2 ♖f5 44 ♔c2 ♖c5+ 45 ♔b2 ♖g5 46 ♔c2 ♖f5 47 ♖d4 ♖c5+ 48 ♔b2 a5 49 ♖d3 ♖e5 50 ♖×f3 ♖e2+ 51 ♔c1 ♔×d6 52 ♔d1 ♖a2 53 ♖d3+ ♔c5 54 ♔e1 e5 55 f4 ef 56 ♖f3 ♔d4 57 ♖×f4+ ♔e3 0-1

**Chess 4.5 – Fenner (2016)** 1 e4 c5 2 ♘f3 e6 3 d4 cd 4 ♘×d4 a6 5 c4 ♘f6 6 ♗d3 ♕c7 7 0-0 ♗c5 8 ♘b3 ♗a7 9 ♘c3 ♘c6 10 ♗g5 ♘e5 11 ♗×f6 gf 12 ♕e2 d6 13 ♔h1 ♗d7 14 f4 ♘×d3 15 ♕×d3 0-0-0 16 ♖ad1 ♗c6 17 f5 ♗b8 18 g3 h5 19 fe h4 20 ♖×f6 hg 21 ♕×g3 ♖dg8 22 ef ♕×f7 23 ♖×f7 ♖×g3 24 ♘d5 ♗e8 25 ♘b6+ ♔d8 26 ♖×b7 ♗c6 27 ♖×b8+ ♔c7 28 ♖c8+ ♖×c8 29 hg ♗×e4+ 30 ♔g1 ♖h8 31 ♘d5+ ♔c6 32 ♘a5+ 1-0

**Chess 4.5 – Ronning (1965)** 1 f4 d5 2 ♘f3 c5 3 e3 ♘c6 4 ♗b5 a6 5 ♗×c6+ bc 6 0-0 g6 7 ♘c3 ♗g7 8 d4 cd 9 ed ♗g4 10 ♗e3 ♘h6 11 ♕d2 ♗×f3 12 ♖×f3 0-0 13 ♖af1 e6 14 ♘e2 ♖b8 15 b4 ♘f5 16 ♗f2 ♘d6 17 ♗e1 ♖e8 18 ♕d3 ♕b6 19 a4 f6 20 h3 e5 21 fe fe 22 a5 ♕a7 23 ♕c3 e4 24 ♖f4 ♖bc8 25 ♗f2 ♘c4 26 ♗e1 ♗h6 27 ♖h4 ♗g5 28 ♖g4 ♗e3+ 29 ♔h2 ♕e7 30 ♗h4 ♕d6+ 31 ♗g3 ♕e6 32 ♗f4 ♖f8 33 ♕e1 ♖f5 34 ♘g3 ♖×f4 35 ♖g×f4 ♗×f4 36 ♖×f4 ♕d6 37 ♖f2 ♖f8 38 c3 h5 39 ♖×f8+ ♔×f8 40 ♔g1 ♕f6 41 ♘e2 h4 42 ♘c1 ♕f4 43 ♘b3 ♘e3 44 ♘c5 ♔f7 45 ♘×a6 ♔g8 46 ♘b8 ♘f5 47 ♘×c6 e3 48 a6 ♘g3 49 a7 e2 50 a8=♕+ ♔h7 51 ♕aa1 1-0

**Morris (2175) – Chess 4.5** 1 e4 c5 2 ♘f3 ♘c6 3 d4 cd 4 ♘×d4 ♘f6 5 ♘c3 d6 6 ♗c4 e6 7 ♗e3 ♘e5 8 ♗e2 d5 9 ed ♘×d5 10 ♗b5+ ♗d7 11 ♘×d5 ed 12 ♕e2 ♗e7 13 0-0 0-0 14 ♗f4 ♘g6 15 ♗g3 ♗f6 16 c3 ♖e8 17 ♕d3 h5 18 h3 h4 19 ♗h2 ♗e5 20 ♗×d7 ♕×d7 21 ♖fe1 ♗×h2+ 22 ♔×h2 ♘f4 23 ♕f3 g5 24 ♔g1 f5 25 ♘e2 ♘×e2+ 26 ♖×e2 g4 27 ♕d3 ♖×e2 28 ♕×e2 gh 29 gh ♖e8 30 ♕f3 ♕e6 31 ♖d1 ♕e4 32 ♕×e4 de 33 ♖d7 ♖b8 34 ♔f1 a6 35 ♔e2 b5 36 ♖d6 ♔f7 37 ♖×a6 ♔g7 38 ♖d6 b4 39 cb ♖×b4 40 b3 ♖b7 41 ♖d2 ♔f6 42 ♖b2 ♖c7 43 b4 ♖c3 44 b5 f4 45 b6 f3+ 46 ♔d2 ♖d3+ 47 ♔e1 ♖d8 48 b7 ♖b8 49 a4 e3 50 fe ♔e5 51 a5 f2+ 52 ♔×f2 ♖f8+ 53 ♔e2 ♖g8 54 b8=♕+ ♖×b8 55 ♖×b8 ♔f5 56 a6 ♔e5 57 a7 ♔d5 58 ♔d3 ♔e6 59 a8=♕ ♔e7 60 ♕a6 ♔d7 61 ♖b7+ ♔e8 62 ♕a8 mate

**Armagosi (1947) – Chess 4.5** 1 d4 ♘f6 2 ♘f3 d5 3 e3 g6 4 c4 ♗g7 5 ♘c3 0-0 6 ♕c2 ♘c6 7 cd ♘×d5 8 ♘×d5 ♕×d5 9 ♗c4 ♕f5 10 ♗d3 ♕g4 11 0-0 ♘b4 12 ♕b3 ♘×d3 13 ♕×d3 ♗f5 14 ♕b5 ♗e4 15 h3 a6 16 ♕c5 ♕d7 17 ♘e5 ♕d6 18 ♕×d6 ed 19 ♘c4 ♗d3 20 b3 ♗×f1 21 ♔×f1 f5 22 ♗a3 f4 23 ♖d1 b5 24 ♘×d6 cd 25 ♗×d6 fe 26 ♗×f8 ♖×f8 27 f3 ♖c8 28 a4 ♖c2 29 ♖d3 ♗h6 30 ab e2+ 31 ♔e1 ♗f4 0-1

**Chess 4.5 – Johnson (1954)** 1 e4 e6 2 d4 d5 3 ♘c3 ♗b4 4 e5 c5 5 a3 ♗×c3+ 6 bc ♕c7 7 ♕g4 f6 8 ♗b5+ ♔f8 9 ef ♘×f6 10 ♕f4 ♕×f4 11 ♗×f4 c4 12 ♖b1 ♘e4 13 ♘e2 ♔e7 14 f3 ♘d6 15 ♗a4 ♘a6 16 0-0 ♘c7 17 ♗g5+ ♔f7 18 g4 ♖b8 19 ♗f4 ♖d8 20 ♗c6 b5 21 a4 ♖b6 22 ♗×b5 ♘c×b5 23 ab ♘×b5 24 ♔g2 ♖d7 25 ♔h1 ♖db7 26 ♖a1 a6 27 ♖a5 ♘a7 28 ♖aa1 ♘c6 29 ♗c1 a5 30 ♗a3 ♗d7 31 ♘f4 ♖b5 32 ♖ae1 h6 33 h4 g6 34 h5 gh 35 ♘×h5 e5 36 de d4 37 ♘f6 ♘×e5 38 ♘e4 ♗c6 39 cd ♔g8 40 de 1-0

**Exhibition game, German television, 1979**

This game was played in Hamburg, where I sat in a soundproof booth in front of a giant industrial robot arm. The arm was connected, via a satellite, to the latest version of the Northwestern University program Chess 4.8, running on a CDC Cyber computer in Minneapolis.

I was under orders to play interesting chess, and so I opened with a very risky variation of the King's Gambit. The advantage swayed back and forth, and just when I had victory in my grasp I got careless and allowed the program to "swindle" me and get away with a draw. The game lasted for about 10 hours, without a break except for a few communications failures of fairly short duration.

**Levy – Chess 4.8** 1 e4 e5 2 f4 ef 3 ♘f3 g5 4 d4 g4 5 ♗×f4 gf 6 ♕×f3 ♘c6 7 d5 ♕f6 8 d×c ♕×b2 9 ♗c4 ♕×a1 10 ♗×f7+ ♔d8 11 0-0 ♕g7 12 ♗d5 ♗c5+ 13 ♗e3 ♗×e3+ 14 ♕×e3 d×c 15 ♖f7 ♕h6 16 ♕d4 c×d 17 ♕×h8 ♕b6+ 18 ♔f1 ♕×b1+ 19 ♔f2 ♕×c2+ 20 ♔g3 ♕d3+ 21 ♖f3 ♕×e4 22 ♕×g8+ ♔d7 23 ♕g7+ ♔c6 24 ♖c3+ ♔b5 25 ♖b3+ ♔a4 26 ♕c3 ♕g4+ 27 ♔f2 ♕c4 28 ♖a3+ ♔b5 29 ♕a5+ ♔c6 30 ♖c3 ♗e6 31 ♕a4+ ♔d6 32 ♖×c4 d×c 33 ♕b4+ ♔c6 34 ♕a4+ b5 35 ♕a6+ ♔d7 36 ♕×b5+ ♔d6 37 ♕b4+ c5 38 ♕d2+ ♔c7 39 ♕h6 ♗g8 40 ♕g7+ ♔c6 41 g4 a6 42 ♕f6+ ♔b5 43 ♕d6 ♔b4 44 ♕b6+ ♔a3 45 ♕c6 ♖f8+ 46 ♔e3 ♖b8 47 ♕×a6+ ♔b2 48 ♕d6 ♖a8 49 ♕d2+ ♔a3 50 h4 ♖a6 51 g5 ♖a8 52 h5 ♖e8+ 53 ♔f4 ♖a8 54 ♔e5 ♖a6 55 g6 h×g 56 h×g ♖a8 57 ♔f6 ♖a4 58 ♔g7 ♖a8 59 ♕g2 ♖d8 60 ♕c6 ♖d3 61 ♕a6+ ♔b4 62 ♔×g8 ♖a3 63 ♕b6+ ♔c3 64 g7 ♖×a2 65 ♔f7 ♖f2+ 66 ♔e7 ♖g2 67 ♕f6+ ♔c2 68 ♕f5+ ♔b2 69 ♔f7 c3 70 ♕e5 c4 71 ♕b5+ ♔c1 72 ♕×c4 ♖×g7+ 73 ♔×g7 c2 74 ♔f6 ♔d2 75 ♕d4+ ♔c1 76 ♔e5 ♔b1 77 ♕b4+ ♔a2 78 ♕c3 ♔b1 79 ♕b3+ ♔a1 80 ♕a4+ ♔b2 81 ♕d4+ ♔b1 82 ♕d3 ♔b2 83 ♕b5+ ♔c3 84 ♕c5+ ♔b2 85 ♕b6+ ♔a1 86 ♕g1+ ♔b2 87 ♕b6+ ♔a1 88 ♕a7+ ♔b1 89 ♕b7+ ♔a1 ½-½

**United States Amateur Team Championship 1980**
**Somerset, New Jersey, February 16-18**

*Introduction*

The U. S. Team competition is a yearly phenomenon where teams of four (with possibly one alternate) compete in a six round swiss tournament. This year, the team Future Schach, composed of four computers, entered. Boards are assigned according to USCF rating. The team members were 1. Chess 4.9 (2040) 2. Duchess (1849/11) 3. Belle (1747) 4. Chaos (unr) and Sargon (unr) served as standby alternate in case any of the regular computers could not make a game.

*Round 1*

**Chess 4.9 – Larry Miller (1740)** 1 e4 e5 2 ♘f3 ♘f6 3 d4 exd4 4 e5 ♘e4 5 ♕xd4 d5 6 exd6 ♘xd6 7 ♗d3 ♘c6 8 ♕f4 ♗e7 9 0-0 0-0 10 ♘c3 ♗e6 11 ♗e3 h6 12 ♕g3 ♔h8 13 ♘d4 ♘xd4 14 ♗xd4 ♗f6 15 ♗c5 ♗xc3 16 bxc3 b6 17 ♗d4 ♕g5 18 ♕xg5 hxg5 19 ♖fe1 ♖fe8 20 ♗e3 f6 21 ♗g6 ♗f7 22 ♗xf7 ♘xf7 23 ♖ad1 ♖ad8 24 ♖xd8 ♖xd8 25 f4 gxf4 26 ♗xf4 ♖d7 27 h4 ♘d6 28 ♔h2 ♔g8 29 ♔g3 ♔f7 30 ♔f3 g6 31 ♖d1 f5 32 g4 fxg4+ 33 ♔xg4 ♖e7 34 ♗xd6 cxd6 35 ♖xd6 ♖e4+ 36 ♔g5 ♖e5+ 37 ♔f4 ♖a5 38 ♖d7+ ♔f6 39 ♔e4 ♖xa2 40 ♖d6+ ♔f7 41 ♔d3 ♖a5 42 c4 ♔g7 43 ♖c6 ♔h6 44 ♖c7 ♔h5 45 ♖h7+ ♔g4 46 ♖g7 ♔h5 47 ♔c3 ♔h6 48 ♖f7 ♔h5 49 ♖h7+ ♔g4 50 ♔b3 ♔f4 51 ♖d7 ♔e5 52 ♖c7 ♔f5 53 ♔c3 ♔g4 54 ♖g7 ♔h5 55 ♔b3 ♔h6 56 ♖d7 g5 57 hxg5+ ♔xg5 58 ♖d5+ ♖xd5 59 cxd5 ♔f6 60 ♔b4 ♔e7 61 ♔b5 ♔d6 62 c4 a5 63 ♔xb6 a4 64 c5+ ♔xd5 65 c6 a3 66 c7 ½-½

**Richard Di Sciascio (1582) – Duchess** 1 e4 e5 2 ♘f3 ♘c6 3 d4 exd4 4 ♘xd4 ♗c5 5 ♘xc6 ♕f6 6 ♕e2 ♕xc6 7 ♘c3 ♘f6 8 ♗g5 ♗d4 9 ♗d2 0-0 10 f3 d5 11 exd5 ♘xd5 12 ♘xd5 ♕xd5 13 c3 ♗c5 14 ♕e4 ♕xe4+ 15 fxe4 ♖e8 16 ♗d3 ♗f5 17 0-0-0 ♗xe4 18 ♗xe4 ♖xe4 19 g3 ♖ae8 20 ♗f4 c6 21 ♖d7 b5 22 ♖hd1 f5 23 ♖c7 ♖4e6 24 h4 a5 25 a3 a4 26 ♖dd7 ♗e7 27 ♗g5 ♗f8 28 ♗f4 ♖8e7 29 ♖xe7 ♗xe7 30 ♔d2 ♗d6 31 ♗xd6 ♖xd6+ 32 ♔e3 g6 33 ♔f4 ♖e6 (Somewhere here, Black declines a draw offer, even though it is in time trouble due to machine failures.) 34 h5 ♔h8 35 h6 ♖e4+ 36 ♔g5 ♔g8 37 ♔f6 ♖e8 38 ♖xc6 Black lost on time. 1-0

**Belle – Jamie Soto (1582)** 1 e4 c5 2 c3 d6 3 d4 cxd4 4 cxd4 e6 5 ♘c3 ♘f6 6 ♘f3 ♗e7 7 ♗g5 h6 8 ♗xf6 ♗xf6 9 e5 dxe5 10 dxe5 ♕xd1+ 11 ♖xd1 ♗d8 12 ♗b5+ ♗d7 13 0-0 a6 14 ♗xd7+ ♘xd7 15 ♘e4 ♗c7 16 ♖c1 ♗xe5 17 ♘xe5 ♘xe5 18 ♘d6+ ♔e7 19 ♘xb7 ♖ac8 20 ♘c5 ♖c6 21 ♖fe1 f6 22 ♖ed1 ♖hc8 23 b4 ♖8c7 24 ♖c3 ♘f7 25 ♖a3 ♖a7 26 ♖dd3 ♘d6 27 f4 ♘c4 28 ♖ac3 ♘b6 29 ♖g3 ♔f7 30 ♖ed3 ♘d5 31 f5 ♘f4 32 fxe6+ ♘xe6 33 ♘xe6 ♖xe6 34 h3 ♖ae7 35 ♔h2 ♖b7 36 ♖d8 f5 37 a3 g5 38 ♖gd3 f4 39 ♖3d5 ♔g6 40 ♖a8 ♔h5 41 ♖a5 ♖bb6 42 ♖a4 ♖ed6 43 ♖a5 ♖e6 44 ♖a7 ♖ed6 45 ♖e7 ½-½

**James Kennedy (1330) – Chaos** 1 ♘f3 ♘f6 2 g3 d5 3 ♗g2 c5 4 b3 e6 5 0-0 ♘c6 6 ♗b2 ♗d7 7 d4 cxd4 8 ♘xd4 ♘xd4 9 ♗xd4 ♕c7 10 ♗xf6 gxf6 11 e4 ♗b5 12 ♖e1 dxe4 13 ♗xe4 ♗b4 14 c3 ♗a5 15 ♕f3 0-0-0 16 ♕xf6 ♔b8? 17 b4 ♗b6 18 a4 ♗c6 19 a5 ♗xf2+ 20 ♔xf2 ♗b5 21 ♔g1 h5 22 ♗g2 h4 23 gxh4? ♖dg8 24 ♔f2?? ♖h5 25 ♘a3 ♕xh2 26 ♘xb5 ♕xg2+ 27 ♔e3 ♖g3+ 28 ♔d4 ♕d5# 0-1

*Round 2*

**Chess 4.9 – Larry D. Evans (2393)** 1 e4 c5 2 ♘f3 d6 3 d4 cxd4 4 ♘xd4 ♘f6 5 ♘c3 a6 6 ♗e2 e6 7 0-0 ♗e7 8 ♗e3 0-0 9 f4 ♘c6 10 e5 dxe5 11 ♘xc6 bxc6 12 ♕xd8 ♖xd8 13 fxe5 ♘d5 14 ♘xd5 cxd5 15 ♗b6 ♖d7 16 ♖ad1 ♖b8 17 ♗d4 ♖c7 18 ♗d3 ♖b4 19 ♗c3 ♖a4 20 ♖a1 ♗b7 21 b3 ♖g4 22 h3 ♖xg2+ !? 23 ♔xg2 ♖xc3 24 ♖ae1 a5 25 ♔g3 ♗g5 26 h4 ♗h6 27 ♖f3 ♖c7 28 a4 d4 29 ♗e4 ♗xe4 30 ♖xe4 ♗e3 31 ♖fxe3! dxe3 32 c4 ♔f8 33 ♖xe3 ♔e7 34 ♔f4 f6 35 ♖g3 ♔f7 36 ♖d3 ♖c5 37 ♖d7+ ♔g6 38 h5+ ♔xh5 39 exf6 gxf6 40 ♖xh7+ ♔g6 41 ♖b7 ♖h5 42 ♔e4 ♖h4+ 43 ♔d3 ♖h3+ 44 ♔c2 ♖h2+ 45 ♔c3 ♖h3+ 46 ♔c2 ♖h2+ 47 ♔c3 f5 48 ♖b6 ♔f6 49 ♖a6 f4 50 ♖xa5 f3 51 ♖a8 ♔f7 52 ♖d8 e5 53 ♖d1 ♔e6 54 c5 e4 55 c6 ♖h8 56 c7 ♖c8 57 ♔d4 ♖xc7 58 ♔xe4 ♖b7 59 ♔xf3 ♖xb3+ 60 ♔e4 ♖b4+ 61 ♖d4 ♖xd4+ 62 ♔xd4 ½-½

**Sharon Evans (1490) – Duchess** 1 ♘f3 ♘f6 2 b3 d5 3 ♗b2 ♗f5 4 g3 c5 5 ♗g2 ♘c6 6 0-0 ♕c7 7 d3 e5 8 ♘bd2 0-0-0 9 c4 d4 10 ♘g5 ♗g6 11 ♘ge4 ♘d7 12 ♗a3 ♗e7 13 ♘f3 ♔b8 14 ♕d2 f5 15 ♘eg5 ♕d6 16 ♖ab1? e4 17 dxe4 fxe4 18 ♘xd4 ♘xd4 0-1

**Belle – Steve Goldstein (1243)** 1 e4 e5 2 ♘f3 ♘c6 3 d4 exd4 4 ♘xd4 ♘xd4 4 ♕xd4 d6 6 ♘c3 ♘f6 7 ♗g5 ♗e7 8 0-0-0 ♗e6 9 ♕a4+ ♘d7 10 ♗xe7 ♕xe7 11 ♘d5 ♗xd5 12 exd5 0-0 13 ♗b5 ♘c5 14 ♕f4 ♕e4 15 ♕xe4 ♘xe4 16 ♖d4 f5 17 f4 a6 18 ♗d7? ♘c5 19 ♗e6+ ♘xe6 20 dxe6 ♖ae8 21 ♖e1 ♖f6 22 ♖b4 b5 23 e7 ♖f7 24 a4 ♖fxe7 25 ♖xe7 ♖xe7 26 axb5 axb5 27 ♖xb5 g6 28 ♔d2 ♔f7 29 c4 ♖e4 30 ♔d3 ♔e7 31 g3 ♖e1 32 c5 ♖h1 33 ♖b7 ♔d7 34 cxd6 ♔xd6 35 ♔d4 ♖xh2 36 ♔e3 h6 37 ♖b8 g5 38 ♖d8+ ♔e7 39 ♖d5 gxf4+ 40 gxf4 ♖xb2 41 ♖xf5 ♖b6 42 ♖e5+ ♖e6 43 ♔d4 ♖xe5 44 ♔xe5 h5 45 ♔f5 c5 46 ♔g6 ♔f8 47 ♔f5 h4 48 ♔g4 c4 0-1

**Steve Matthaey (1076) – Chaos** 1 e4 c5 2 c3 ♘f6 3 e5 ♘d5 4 d4 cxd4 5 cxd4 d6 6 ♘f3 ♘c6 7 ♘c3 dxe5 8 dxe5 ♘db4 9 a3 ♕xd1+ 10 ♔xd1 ♘a6 11 b4 ♘c7 12 h3 ♗e6 13 ♗d3 ♘d5 14 ♘e4 ♗f5 15 ♘c5 ♗xd3 16 ♘xd3 ♖d8 17 ♖e1 ♘dxb4! 18 axb4 ♘xb4 19 ♔e2 ♘xd3 20 ♖d1 ♘xc1+ 21 ♖dxc1 a6 22 ♖c7 ♖b8 23 ♘d4 ♖g8? 24 ♘b3 e6 25 ♘a5 b6 26 ♘b7 ♖a8 27 ♘d6+ ♔d8 28 ♖xf7 ♗xd6 29 exd6 a5 30 ♖c1 ½-½

*Round 3*

**Tim Hall (1962) – Chess 4.9** 1 f4 d5 2 b3 ♘f6

3 ♗b2 ♘c6 4 e3 ♗f5 5 ♘f3 e6 6 ♗b5 ♗c5 7 ♘e5 ♕d6 8 d4 ♗b4+ 9 c3 ♗a5 10 ♗a3 ♗×c3+ 11 ♔f1? ♗b4 12 ♘×c6 ♗×a3 13 ♘a5+ c6 14 ♘×b7 ♕d7 15 ♘×a3 c×b5 16 ♘c5 ♕c6 17 ♖c1 0-0 18 h3 b4 19 ♘c2 ♘e4 20 ♕e1 ♘×c5 21 d×c5 ♕a6+ 22 ♔g1 ♕×a2 23 ♘d4 ♗e4 24 ♖h2 ♕a5 25 ♘c6 ♕a6 26 ♘e7+ ♔h8 27 c6 ♖fe8 28 ♕×b4 ♕d3 29 ♖c3? ♕d2 30 ♖c4 ♕×e3+ 31 ♔h1 d×c4 0-1

**Duchess – Brian Katz (1713)** 1 e4 c5 2 ♘c3 d6 3 g3 g6 4 d4 c×d4 5 ♕×d4 ♘f6 6 ♘f3 a6 7 ♘d5 ♗g7 8 ♕b6 ♘×d5 9 ♕×d8+ ♔×d8 10 e×d5 h6 11 c3 b5 12 ♗g2 ♗b7 13 ♘h4 ♘d7 14 0-0 ♗f6 15 a4 ♗×h4 16 g×h4 ♘f6 17 a×b5 a×b5 18 ♖×a8+ ♗×a8 19 ♖d1 ♔c7 20 ♗f4 ♘h5 21 ♗e3 ♘g7 22 ♗f1 ♘f5 23 ♗×b5 ♘×e3 24 f×e3 ♖b8 25 c4 ♗b7 26 b4 ♔b6 27 ♔f2 ♖a8 28 ♗d7 f6 29 ♖d4 ♔c7 30 ♗e6 ♖a2+ 31 ♔g3 ♖c2 32 ♖g4 g5 33 h×g5 h×g5 34 h4 g×h4+ 35 ♖×h4 ♗c8 36 ♗×c8 ♔×c8 37 ♖f4 ♔d7 38 ♔f3 ♔d8 39 b5 ♔c7 40 ♔g4 ♖g2+ 41 ♔f5 ♖g5+ 42 ♔e4 ♖e5+ 43 ♔d4 ♔b6 44 ♖e4 ♔c7 45 ♖×e5 d×e5+ 46 ♔e4 ♔b6 47 ♔f5 ♔c5 48 ♔e6 ♔b6 49 ♔×e7 f5 50 d6 f4 51 d7 1-0

**Ron Dittman (1655) – Belle** 1 e4 e5 2 ♗c4 ♘f6 3 d3 c6 4 ♘c3 d5 5 e×d5 c×d5 6 ♗b3 ♗b4 7 ♗d2 d4 8 ♘ce2 ♗×d2+ 9 ♕×d2 ♘c6 10 ♘f3 ♗g4 11 ♘g5 0-0 12 0-0 ♕a5 13 ♕e1 ♕c5 14 a3 ♖ad8 15 f3 ♗d7 16 ♕d2 ♕a5 17 ♕e1 ♕×e1 18 ♖a×e1 ♘a5 19 ♗a2 h6 20 ♘e4 ♘×e4 21 f×e4 ♗e6 22 ♗×e6 f×e6 23 ♖×f8+ ♔×f8 24 ♖d1 ♘c6 25 ♔f2 ♔e7 26 c4 ♖f8+ 27 ♔e1 ♔d6 28 b4 h5 29 ♘c1 a5 30 c5+ ♔c7 31 b5 ♘e7 32 a4 ♘g6 33 g3 h4 34 ♔e2 h×g3 35 h×g3 ♖h8 36 ♖f1 ♖f8 37 ♖×f8 ♘×f8 38 ♔f3 b6 39 c6 ♘h7 40 ♘b3 ♔d8 41 ♘d2 ♘g5+ 42 ♔g4 ♘f7 43 ♘f3 ♔e7 44 ♘g5 ♘d6 45 ♔h5 ♔f6 46 ♘f3 ♘e8 47 ♘h2 ♔e7 48 ♘f3 ♔f6 49 ♘e1 ♘c7 50 ♘c2 ♔e7 51 ♔g6 ♘e8 52 ♘a3 ♘d6 53 ♔×g7 ♘e8+ 54 ♔g6 ♘d6 55 ♔g5 ♘f7+ 56 ♔g4 ♘d6 1-0

**Chaos – Gerard Kirzner (1562)** 1 d4 d5 2 c4 c6 3 ♘c3 ♘f6 4 e3 e6 5 ♘f3 ♗b4 6 a3 ♗×c3+ 7 b×c3 0-0 8 ♗d3 ♘bd7 9 ♗d2 a6 10 0-0 ♕e7 11 c×d5 c×d5 12 c4 ♕d8 13 c×d5 ♘×d5 14 ♕c2 f5 15 ♖ab1 ♘7f6 16 ♖fc1 b5 17 a4 b×a4 18 ♕×a4 ♗d7 19 ♕a2 ♕e8 20 ♗×a6 ♗c8? 21 ♖×c8! ♕×c8 22 ♗×c8 ♖×a2 23 ♗×e6+ ♔h8 24 ♗×f5 ♖e8 25 ♖b7 g6 26 ♗b1 ♖a1 27 h3 ♘e4 28 ♗e1 ♘ec3 29 ♗d3 ♖d1 30 ♗c4 ♘f6 31 ♔h2 ♘ce4 32 ♗e2 ♖c1 33 ♗d3 ♖d1 34 ♗b1 ♖c8 35 g3 ♖cc1 36 ♗×e4 ♘×e4 37 ♖e7 ♘f6 38 ♖f7 ♘e4 39 ♖f8+ ♔g7 40 ♖f4 ♘f6 41 ♗b4? ♘d5 42 ♗f8+ ♔g8 43 ♗h6 ♘×f4 44 ♗×f4 ♖c2 45 ♔g2 ♖a2 46 e4 ♖d3 47 e5 ♖aa3 48 ♗e3 ♖a2 49 ♗c1 ♖a1 50 ♗g5 ♖aa3 51 ♘e1 ♖d1 52 ♘f3 ♖dd3 53 ♗e3 ½-½

*Round 4*

**Paul Neuer (1983) – Chess 4.9** 1 e4 ♘c6 2 d4 d5 3 e5 f6 4 f4 ♗f5 5 c3 e6 6 ♘e2 ♘ge7 7 c×f6 g×f6 8 ♘g3 ♗g6 9 ♗d3 ♗×d3 10 ♕×d3 h5 11 ♕e2 h4 12 ♘h5 ♔f7 13 g4 h×g3 14 h×g3 ♘f5 15 ♘d2 ♗d6 16 ♘f1 ♘g7 17 ♔f3 ♘×h5 18 ♖×h5 ♘e7 19 ♗d2 c5 20 0-0-0 c×d4 21 c×d4 ♕b6 22 ♗c3 ♖ac8 23 ♖×h8 ♖×h8 24 ♘e3 ♕b5 25 g4 ♕a4 26 ♔b1 ♖h2 27 ♖d2? ♖×d2 28 ♗×d2 ♕×d4 29 ♘f1 ♗b4 30 ♗×b4 ♕×b4 31 g5 ♕e1+ 32 ♔c2 ♘c6 33 ♕h5+ ♔e7 34 g×f6+ ♔d6 35 f7 ♕e4+ 36 ♔d1 ♕b1+ 37 ♔e2 ♕×b2+ 38 ♔f3 ♔e7 39 ♘e3 ♕d2 40 f8(♕)+ ♔×f8 41 ♕h8+ ♔e7 42 ♕g7+ ♔d6 43 ♕f8+ ♔d7 44 ♕f7+ ♘e7 45 f5 e×f5 46 ♘×f5 ♕d3+ 47 ♘e3 d4 0-1

**Duchess – Fred Olson (1968)** 1 e4 c5 2 d4 c×d4 3 c3 d×c3 4 ♘×c3 ♘c6 5 ♘f3 d6 6 ♗c4 e6 7 0-0 ♘f6 8 ♕e2 ♗e7 9 ♗g5 0-0 10 ♖fd1 e5 11 ♕c2 a6 12 ♖ac1 ♗e6 13 ♗×f6 ♗×f6 14 ♘d5 ♗g4 15 ♗e2 ♗×f3 16 ♘×f6+ ♕×f6 17 ♗×f3 ♘d4 18 ♕d3 ♖ac8 19 ♖×c8 ♖×c8 20 ♗g4 ♖c2 21 ♖d2 ♖×d2 22 ♕×d2 ♕e7 23 ♕c3 g6 24 ♕c4 ♔g7 25 ♕c8 a5 26 a3 b5 27 b4 a×b4 28 a×b4 ♕a7 29 ♕c1 ♕b7 30 ♕b1 ♕c6 31 h3 ♕c2 32 ♕×c2 ♘×c2 33 ♗e2 ♘d4 34 ♗d3 f5 35 f3 ♔f6 36 h4 ♔e6 37 ♔f2 d5 38 e×d5+ ♔×d5 39 ♔e3 h6 40 g3 g5 41 h×g5 h×g5 42 ♗f1 f4+ 43 ♔f2 e4 44f×e4+ ♔×e4 45 ♗h3 ♔d3 46 ♗f1+ ♔c3 47 ♗×b5! f3 48 ♗a6 g4 49 ♗c8 ½-½

**John Shoosmith (1705) – Belle** 1 e4 e5 2 f4 e×f4 3 ♘f3 d5 4 e5? g5 5 h4 g4 6 ♘g5 h6 7 ♘×f7 ♔×f7 8 d4 ♗e7 9 ♗e2 ♗×h4+ 10 ♔f1 ♕g5 11 c4 ♘e7 12 c×d5 ♘×d5 13 ♕b3 ♖d8 14 ♗c4 c6 15 ♘c3 ♔g6 16 ♘e4 ♕e7!? 17 ♘d6? ♖×d6! 18 ♗d3+ ♔h5 19 ♕c2 ♖e6 20 ♗f5 ♕f8 21 ♕f2 ♔g5 22 ♕×h4+ ♔×f5 23 ♕h5+ ♔e4 24 ♔f2 g3+ 25 ♔e2 ♔×d4 26 ♖d1+ ♔e4 27 ♖e1 ♘d7 28 ♔f1+ ♘e3+ 29 ♔g1 ♘×e5 30 b3 ♖g6 31 ♗b2 ♗f5 32 ♖ad1 ♖d6 33 ♗×e5 ♖×d1 34 ♕×d1 ♔×e5 35 ♕a1+ ♔d5 36 ♖c1 ♗e4 37 ♕c3 ♗×g2 38 ♕a5+ ♔d6 39 ♕c5+ ♔c7 40 ♕e5+ ♕d6 41 ♕g7+ ♔b6 42 ♕c3 ♖e8 43 b4 f3 0-1

**Chaos – Charles Aronowitz (1505)** 1 d4 ♘f6 2 c4 c5 3 d5 b5 4 c×b5 a6 5 e3 d6 6 ♘c3 g6 7 ♗c4 ♘bd7 8 ♘f3 ♗g7 9 0-0 ♘b6 10 ♗e2 0-0 11 ♘g5 h6 12 ♘f3 ♘g4 13 h3 ♘e5 14 ♘×e5 ♗×e5 15 b×a6 ♗×a6 16 ♗×a6 ♖×a6 17 e4 ♗g7 18 ♕e2 ♕a8 19 ♖d1 ♘d7 20 ♕b5 ♘e5 21 f4 ♖a5 22 ♕e2 ♘d7 23 ♖d2 ♖c8 24 ♖c2 ♘b6 25 ♗d2 ♖a7 26 ♖b1 ♗×c3 27 b×c3 ♘d7 28 ♗e1 ♖a4 29 ♖cb2 ♘f6 30 c4 ♕a6 31 e5 ♘h5 32 e×d6 ♘×f4?? 33 ♕×e7 ♕×c4 34 d7 ♖ca8 35 ♖b8+ ♖×b8 36 ♖×b8+ ♔g7 37 ♕e5+ 1-0

*Round 5*

**Chess 4.9 – George Shrier (2152)** 1 e4 e6 2 d4 d5 3 ♘c3 ♗b4 4 e5 b6 5 ♕g4 ♗f8 6 ♘f3 ♕d7 7 ♗g5 ♘e7 8 ♗d3 ♗a6 9 0-0 ♘f5 10 ♖ac1 ♗b4 11 a3 ♗×c3 12 b×c3 ♗×d3 13 c×d3 ♘c6 14 ♕h5 h6 15 g4 ♘fe7 16 ♗d2 0-0-0 17 ♕h4 ♘a5 18 ♖ce1 ♘g6 19 ♕g3 ♕a4 20 h4 h5 21 ♗g5 ♖dg8 22 g×h5 ♖×h5 23 ♘d2 ♖gh8 24 ♖a1 ♘×h4 25 ♗×h4 ♖×h4 26 ♘f3 ♖h1+ 27 ♔g2 ♖×f1 28 ♖×f1 g6 29 ♘g5 ♖f8 30 ♕f4 ♕×a3 31 ♘h7 ♖d8 32 ♕×f7 ♕×c3 33 ♘g5!? ♕×d4 34 ♕×c7+ ♔×c7 35 ♘×c6+ ♔d7 36 ♘×d4 ♘c6 37 e6+ ♔d6 38 ♘×c6 ♔×c6 39 ♖a1 a5 40 ♔g3 ♔d6 41 ♖b1 ♖b8 42 ♖e1 ♔e7 White lost on time. 0-1

**Zoltan Bartok (2109) – Duchess** 1 c4 ♘f6 2 ♘c3 e6 3 g3 d5 4 c×d5 e×d5 5 ♗g2 ♗e7 6 b3 0-0 7 h4 ♖e8 8 a4 a5 9 ♗b2 ♘c6 10 ♖c1 ♘b4 11 ♔f1 c6 12

♘h3 ♗d6 13 ♗a1 d4 14 ♘b1 ♗g4 15 f3 ♘h5 16 e4 d×e3 17 d×e3 ♘×g3+ 18 ♔g1 ♘a2 19 ♖c4 ♘×h1 20 ♖d4 ♖×e3 21 ♘g5 ♖a7?? 22 ♗×h1 ♕e7 23 ♔f1 h6 24 ♖×d6 h×g5 25 ♖d8+ ♔h7 26 f×g4 ♘b4 27 ♘a3 ♖×b3 28 ♕d4 ♕f6+ 29 ♕×f6 g×f6 30 ♗×f6 ♔g6 31 h×g5 ♖e3 32 ♗d4? ♖×a3 33 ♖g8+ ♔h7 34 ♖g7+ ♔h8 35 ♖g6+ f6 36 ♖×f6? ♔g8 37 ♗×a7 ♖×a4 38 ♔f2 ♔g7 39 ♔g3 ♖a2 40 ♗d4 ♖d2 41 ♗a1 ♔g8 42 ♗e4 ♖e2 43 ♗g6 ♖e3+ 44 ♔f2 ♖a3 45 ♗d4 ♖a2+ 46 ♔g3 ♖a3+ 47 ♔h4 c5 48 ♗×c5 ♖c3 49 ♗d4 ♖c6 50 ♗f7+ ♔f8 51 ♖f5 ♔e7 52 g6 ♖c8 53 g7 a4 54 g8(♕) ♖×g8 55 ♗×g8 ♔d7 1-0

**Belle – Douglas Bellizzi (2066)** 1 e4 c5 2 c3 ♘f6 3 e5 ♘d5 4 d4 c×d4 5 c×d4 e6 6 a3 b6 7 ♘f3 ♗b7 8 ♗d3 ♘a6 9 ♗d2 ♗e7 10 0-0 0-0 11 ♕e2 ♕c8 12 ♘c3 ♘ac7 13 ♖fc1 ♘×c3 14 ♖×c3 ♗c6 15 ♘g5 ♗×g5 16 ♗×g5 ♕e8 17 ♗e4? ♗×e4 18 ♕×e4 ♘d5 19 ♖h3 f5 20 e×f6 ♕g6 21 ♕×g6 h×g6 22 f×g7 ♖fc8 23 ♗h6 ♔h7 24 ♖e1 ♘f6 25 d5? ♘×d5 26 f4 ♘f6 27 ♖d1 d5 28 ♖e1 ♖c6 29 b4 b5 30 a4 a6 31 a×b5 a×b5 32 ♔h1 ♘g4 33 ♗g5+ ♔×g7 34 ♔g1 ♖a4 35 ♗e7 ♘f6 36 ♗c5 ♔f7 37 ♖h8 ♖a2 38 ♖b8 ♘e4 39 ♖×b5 ♘×c5 40 b×c5 ♖c2 41 ♖a1 ♖2×c5 42 ♖a7+ ♖c7 43 ♖×c5 ♖×a7 44 ♔f2 ♔f6 45 ♖c2 ♔f5 46 g3 g5 47 f×g5 ♔×g5 48 ♖e2 ♔f5 49 ♖d2 ♖h7 50 h4 ♖f7 51 ♔g2 ♖d7 52 ♖f2+ ½-½

**David Burris (2056) – Chaos** 1 d4 ♘f6 2 c4 e6 3 ♘c3 ♗b4 4 e3 c5 5 ♘e2 d5 6 a3 ♗×c3+ 7 ♘×c3 c×d4 8 e×d4 d×c4 9 ♗×c4 ♘c6 10 ♗e3 0-0 11 0-0 ♗d7 12 ♕d3 ♕c7 13 d5?? ♘e5 14 ♕d4 ♘×c4 15 ♗f4 e5 16 d6 e×d4 17 d×c7 d×c3 18 b×c3 ♘d5 19 ♗g3 ♖ac8 20 ♖fd1 ♗e6 21 h3 ♘×c7 22 ♖ab1 b6 23 a4 ♘d5 24 ♖a1 ♘×c3 25 ♖e1 ♘a5 26 ♔h2 ♖c4 27 ♗e5 ♘×a4 28 ♖ed1 ♘c6 29 ♗g3 ♘c3 30 ♖e1 ♖c8 31 h4 ♘e4 32 ♖e3 a5 33 h5 ♘×g3 34 ♖×g3 ♖h4+ 35 ♔g1 ♘d4 36 ♖e1 ♘e2+ 37 ♔f1 ♖c2 38 ♖h3 ♖×h3 39 g4 ♗d5 0-1

*Round 6*

**Jose Espinosa (1896) – Chess 4.9** 1 d4 d5 2 c4 d×c4 3 e3 e5 4 ♗×c4 e×d4 5 ♕×d4 ♕×d4 6 e×d4 ♘c6 7 ♗b5 ♗d7 8 ♘f3 a6 9 ♗a4 ♘f6 10 0-0 ♗d6 11 ♘c3 0-0 12 ♗g5 b5 13 ♗×f6 b×a4 14 ♘e5 ♗e8 15 ♘×c6 ♗×c6 16 ♗e5 ♖ab8 17 ♖ab1 ♖fe8 18 f4 ♗d7 19 ♖f2 f6 20 ♗×d6 c×d6 21 ♖d2 a3 22 b4 ♖bc8 23 ♖b3 ♖e3 24 ♘d1? ♖×b3 25 a×b3 ♗f5 26 ♘e3 ♗b1 27 ♘c4 a2 28 ♖×a2 ♗×a2 29 ♘×d6 ♖d8 30 ♘f5 ♔f8 31 ♔f2 g6 32 ♔e3 g×f5 33 ♔d3 ♗×b3 34 ♔c3 ♗d5 35 g3 ♖c8+ 0-1

**Duchess – Bruce Jurin (1756)** 1 e4 e5 2 ♘f3 ♘c6 3 ♗b5 a6 4 ♗a4 ♘f6 5 0-0 ♗e7 6 ♕e2 b5 7 ♗b3 d6 8 a4 ♗g4 9 c3 ♘a5 10 ♗c2 c6 11 b4 ♘c4 12 a5 d5 13 e×d5 ♕×d5 14 d3 ♗×f3 15 g×f3 ♘d6 16 ♖e1 0-0 17 ♕×e5 ♕×f3 18 ♕f4 ♕×f4 19 ♗×f4 ♘d5 20 ♗e5 ♖fe8 21 ♘d2 f6 22 ♗g3 ♘f5 23 ♗b3 ♘×g3 24 h×g3 ♔f8 25 ♖ac1 ♗d6 26 ♖×e8+ ♖×e8 27 ♘e4 ♗e7 28 ♗×d5 c×d5 29 ♘c5 ♖c8 30 ♘×a6 ♖c6 31 ♘c5 ♗×c5 32 b×c5 ♖×c5 33 d4 ♖c7 34 ♖b1 ♖b7 35 a6 ♖a7 36 ♖×b5 ♖×a6 37 ♖×d5 ♖c6 38 ♖c5 ♖×c5 39 d×c5 ♔e7 40 ♔f1 ♔d7 41 ♔e2 ♔c6 42 ♔d3 ♔×c5 43 f4 f5 44 c4 h6 45 ♔c3 g5 46 f×g5 h×g5 47 ♔d3 ½-½

**Darin Treadwell (1629) – Belle** 1 ♘f3 d5 2 c4 e6 3 b3 ♘f6 4 ♗b2 ♗e7 5 g3 0-0 6 ♗g2 ♘c6 7 d4 d×c4 8 b×c4 ♗d7 9 0-0 ♘a5 10 ♕d3 c5 11 ♘bd2 c×d4 12 ♘×d4 ♖c8 13 ♖ac1 ♘c6 14 ♘2f3 ♕a5 15 a3 ♘×d4 16 ♘×d4 ♕a6 17 ♘f3 ♗c6 18 ♘g5 ♖fd8 19 ♕c2 ♗×g2 20 ♗×f6? ♗×f6 21 ♕×h7+ ♔f8 22 ♔×g2 ♗×g5 23 e3 ♗f6 24 ♕e4 ♖d2 25 c5 ♗b2 26 ♖b1 ♕c6 27 ♕×c6 ♖×c6 28 ♖fd1 ♖×d1 29 ♖×d1 ♖×c5 30 ♖d7 ♖b5 31 a4 ♖b4 32 a5 ♔e8 33 ♖c7 ♗e5 34 ♖c5 ♗d6 35 ♖g5 g6 36 h4? ♗e7! 37 ♖e5 f6 38 ♖×e6 ♔d7 0-1

**Chaos – Jason Luchan (1625)** 1 d4 e6 2 c4 ♘f6 3 ♘c3 ♗b4 4 ♕c2 d5 5 a3 ♗×c3+ 6 ♕×c3 0-0 7 e3 ♘bd7 8 ♘f3 ♖e8 9 ♗d3 ♕e7 10 c×d5! e×d5 11 ♕×c7 h6 12 ♗d2 b6 13 0-0 ♕e6 14 ♖ac1 ♖e7 15 ♕c6 ♕×c6 16 ♖×c6 ♖e6 17 ♖×e6 f×e6 18 ♖c1 ♘e8 19 ♗b5 ♘d6 20 ♖×c8+ ♘×c8 21 ♗×d7 ♔f7 22 ♘e5+ ♔e7 23 ♗c6 ♔d6 24 ♗×a8 a5 25 b4 ♘a7 26 b×a5 b5 27 a6 ♔c7 28 ♗b7 g5 29 ♗a5+ ♔d6 30 ♗b6 b4 31 a×b4 ♘b5 32 ♗c6 ♘c3 33 ♗d8 ♘e2+ 34 ♔f1 ♘g3+ 35 h×g3 h5 36 b5 1-0

*Synopsis*

The computer team finished with a match score of 4 out of 6 and a game score of 15 out of 24. Chess 4.9 got 4 out of 6 points for a performance rating of 2168; Duchess 3/6 pr=1762; Belle 3/6 pr=1644; and Chaos got 5/6 for a pr of 1946.

**Mississippi State Championship, 1981**

The following game was played against a U.S. Master rated 2262.

**Human (2262) v Cray Blitz**

*Times in minutes and seconds are in parentheses. Computer's evaluation is shown in pawn equivalents.*

| *White* | | *Black* | | *Predicted Variation* | *Nodes* | *Evaluation* |
|---|---|---|---|---|---|---|
| 1 c4 | (0.15) | ♘f6 | (0.00) | Book | 0 | 0.066 |
| 2 ♘c3 | (0.21) | c6 | (0.00) | Book | 0 | 0.241 |
| 3 ♘f3 | (0.16) | d5 | (2.17) | 4 e3 e6 5 d4 ♘bd7 6 ♕a4 ♗d6 | 408,258 | – 0.048 |
| 4 cd | (0.28) | cd | (2.17) | 5 d4 ♘c6 6 e3 e6 7 ♗b5 ♗b4 | 558,670 | 0.000 |
| 5 d4 | (0.50) | ♘c6 | (0.50) | Back in opening book | 0 | 0.241 |
| 6 g3 | (0.14) | ♘e4 | (2.19) | 7 ♗g2 ♘×c3 8 bc ♕a5 9 ♕d3 e6 | 384,099 | 0.183 |

| White | | Black | | Predicted Variation | Nodes | Evaluation |
|---|---|---|---|---|---|---|
| 7 ♗g2 | (1.14) | ♘×c3 | (3.23) | 8 bc e6 9 0-0 ♗e7 10 e3 0-0 | 877,648 | 0.085 |
| 8 bc | (0.11) | e6 | (3.06) | 9 0-0 ♗d6 10 ♕d3 0-0 11 e4 f5 12 ed ed | 565,507 | 0.129 |
| 9 0-0 | (0.28) | ♗d6 | (2.36) | 10 ♖b1 0-0 11 ♕a4 f5 12 ♗g5 ♕d7 | 509,561 | 0.127 |
| 10 ♕c2 | (2.15) | 0-0 | (2.55) | 11 ♖b1 f5 12 ♗d2 ♖e8 13 e3 e5 | 508,413 | 0.117 |
| 11 ♘g5 | (5.10) | f5 | (2.43) | 12 f4 ♘a5 13 ♖b1 ♘c4 14 e3 | 433,129 | 0.167 |
| 12 f4 | (2.58) | ♘a5 | (0.00) | 13 e3 ♘c4 14 ♘f3 ♕f6 15 ♘e5 ♗×e5 16 fe | 493,756 | 0.148 |
| 13 ♕d3 | (3.55) | ♕d7 | (2.27) | 14 ♘f3 ♘c4 15 ♘d2 ♘×d2 16 ♗×d2 | 381,838 | 0.138 |
| 14 ♗d2 | (6.13) | ♘c4 | (2.18) | 15 e3 ♕c6 16 ♖fb1 b5 17 ♖b3 | 348,683 | 0.147 |
| 15 ♗c1 | (2.22) | ♕a4 | (2.18) | 16 e3 ♗d7 17 ♖b1 b5 | 355,534 | 0.193 |
| 16 g4 | (30.55) | h6 | (2.18) | 17 gf hg 18 fg ♗a3 19 ♗d2 ♘×d2 20 ♕×d2 | 378,020 | 1.243 |
| 17 gf | (1.08) | hg | (1.09) | 18 fg ♕e8 19 f6 ♕h5 20 ♗h3 | 326,700 | 1.193 |
| 18 fg | (0.27) | ♗a3 | (1.52) | 19 ♗×a3 ♕×a3 20 e4 ef 21 ef | 320,602 | 1.068 |
| 19 g6 | (1.50) | ♗×c1 | (2.21) | 20 ♖f×c1 ♖×f5 21 ♕h3 ♖g5 22 ♕h7+ ♔f8 23 ♖f1+ ♔e7 24 ♕×g7+ ♔d8 | 403,760 | 1.212 |
| 20 ♖a×c1 | (0.26) | ♘d6 | (2.21) | 21 ♕h3 ♖×f5 22 ♕h7+ ♔f8 23 ♕h8+ ♔e7 24 ♕×g7+ ♔d8 25 ♖×f5 ♘×f5 | 362,642 | 0.891 |
| 21 ♕h3 | (0.36) | ♖×f5 | (1.44) | 22 ♕h7+ ♔f8 23 ♗h3 ♖×f1+ 24 ♖×f1+ ♔e7 25 ♕×g7+ ♔d8 26 ♖f8+ ♘e8 | 408,459 | 0.557 |
| 22 ♕h7+ | (2.59) | ♔f8 | (0.00) | 23 ♗h3 ♖×f1+ 24 ♖×f1+ ♔e7 25 ♕×g7+ ♔d8 26 ♖f8+ ♘e8 | 426,135 | 0.557 |
| 23 ♕h8+ | (1.52) | ♔e7 | (0.00) | Forced move | 349 | 0.288 |
| 24 ♕×g7+ | (2.28) | ♔d8 | (2.29) | 25 ♕h8+ ♔c7 26 g7 ♘f7 27 ♖×f5 ef 28 ♗×d5 | 358,566 | − 0.431 |
| 25 ♖×f5 | (6.33) | ♘×f5 | (2.29) | 26 ♕f8+ ♕e8 27 g7 ♘e7 28 c4 ♗d7 29 ♕×e8+ ♔×e8 30 cd ed | 419,712 | 0.413 |
| 26 ♕f6+ | (0.32) | ♘e7 | (2.29) | 27 g7 ♕e8 28 c4 ♔d7 29 cd ed | 417,621 | 0.455 |
| 27 g7 | (2.27) | ♕e8 | (2.09) | 28 e4 ♔d7 29 ed ♘×d5 30 ♕f8 ♘e7 31 ♕×e8+ ♔×e8 | 383,644 | 0.266 |
| 28 ♗f3 | (3.01) | ♔d7 | (2.35) | 29 e4 ♖b8 30 ♔h1 de 31 ♕e5 ♘c6 32 ♕×e4 | 385,273 | 0.451 |
| 29 ♖f1 | (4.47) | ♘g8 | (2.35) | 30 ♕g5 ♕e7 31 ♕×e7+ ♔×e7 32 e4 ♘f6 33 ed ed | 382,522 | 0.306 |
| 30 ♗g2 | (X.XX) | ♕e7 | (0.00) | 31 ♕g6 ♔d6 32 e4 de 33 ♗×e4 ♔d7 34 ♖f7 | 515,188 | 0.209 |
| 31 ♕g6 | (X.XX) | ♔d6 | (1.49) | 32 ♕g3+ ♔c6 33 c4 ♕d8 34 ♕g6 ♔c7 35 cd ed | 351,300 | 0.278 |
| 32 e4 | (10.08) | de | (2.45) | 33 ♕g3+ e5 34 ♗×e4 ♕e6 35 ♗f5 ♕d5 36 de+ ♔c7 | 385,266 | − 0.219 |
| 33 ♗×e4 | (1.18) | ♕h4 | (2.46) | 34 ♗g2 ♗d7 35 ♗×b7 ♖b8 36 ♕g2 ♗b5 | 391,722 | − 0.713 |
| 34 ♕g3+ | (6.23) | ♕×g3+ | (2.46) | 35 hg ♗d7 36 ♖f8 ♖c8 37 ♖×c8 ♗×c8 38 ♗h7 ♘e7 39 g8♕ ♘×g8 40 ♗×g8 | 610,609 | − 1.459 |
| 35 hg | (0.37) | ♗d7 | (2.08) | 36 ♖f8 ♖c8 37 ♖×c8 ♗×c8 38 ♗h7 ♘e7 39 g8♕ ♘×g8 | 470,710 | − 1.359 |
| 36 ♖f8 | (7.31) | ♖c8 | (0.00) | 37 ♖×c8 ♗×c8 38 ♗h7 ♘e7 39 g8♕ ♘×g8 | 1,395,040 | − 1.311 |
| 37 ♗h7 | (1.56) | ♖×c3 | (3.01) | 38 ♔h2 ♖c1 39 ♖f2 ♖c8 40 ♖f8 ♖c1 | 604,912 | 0.000 |
| 38 ♖×g8 | (4.14) | ♖×g3+ | (0.00) | 39 ♔f2 ♖g4 40 ♗e4 b5 41 ♖a8 ♖×g7 | 610,779 | − 0.090 |
| 39 ♔f2 | (0.07) | ♖g5 | (2.52) | 40 ♗e4 e5 41 de+ ♔×e5 42 ♗×b7 ♗e6 ♗e6 43 ♖e8 ♖×g7 | 615,741 | − 0.010 |
| 40 ♗e4 | (0.46) | b6 | (2.14) | 41 ♔e3 e5 42 a3 ♗e6 43 ♖d8+ ♔e7 44 ♖a8 | 568,593 | 0.009 |
| 41 ♔e3 | (0.35) | e5 | (2.30) | 42 ♖a8 ♖×g7 43 ♖×a7 ♖g3+ 44 ♔f2 ♖c3 45 de+ ♔×e5 46 ♖×d7 ♔×e4 | 616,227 | 0.004 |
| 42 ♖a8 | (1.37) | ♖g3+ | (1.32) | 43 ♗f3 ed+ 44 ♔×d4 ♗e6 45 ♗b7 ♖g4+ 46 ♔e3 ♗×a2 47 ♖×a7 ♖×g7 | 702,613 | 0.636 |
| 43 ♔f2 | (0.08) | ♖×g7 | (3.22) | 44 ♖×a7 ed 45 ♖b7 b5 46 ♗d3 ♖g5 47 a3 ♖d5 | 787,876 | 0.751 |
| 44 de+ | (0.07) | ♔×e5 | (3.22) | 45 ♗f3 ♗e6 46 a3 ♖c7 47 ♖e8 ♖c2+ 48 ♔g3 ♔d6 | 708,164 | 0.785 |
| 45 ♗f3 | (0.58) | ♗e6 | (3.07) | 46 ♖e8 . . . | 675,932 | 0.761 |
| 46 a4 | (0.07) | ♖f7 | (3.25) | 47 ♖e8 ♖f4 48 a5 ba 49 ♔e3 ♖f7 50 ♗e4 | 732,838 | 1.665 |
| 47 ♔e3?? | (0.11) | ♖×f3+ | (3.26) | 48 ♔d2 ♖f7 49 ♔e3 ♖h7 50 ♔d2 ♖h2+ 51 ♔e3 ♖h3+ 52 ♔d2 | 760,544 | 4.093 |

| *White* | | *Black* | | *Predicted Variation* | *Nodes* | *Evaluation* |
|---|---|---|---|---|---|---|
| 48 ♔×f3 | (0.06) | ♗d5+ | (2.00) | 49 ♔e2 ♗×a8 50 ♔d3 ♗c6 51 a5 ba | | |
| | | | | 52 ♔c4 a4 53 ♔c3 ♗e4 | 703,217 | 5.071 |
| 49 ♔e3 | (0.05) | ♗×a8 | (0.40) | 49 a5 ba 50 ♔d3 | 266,098 | 5.079 |
| 50 a5 | (0.06) | ♗e4 | (1.53) | – | 722,277 | 7.694 |
| 51 ♔d2 | (1.04) | ♔d4 | (2.01) | 52 ♔d1 ♔c4 53 ♔c1 ♔c3 54 ♔d1 a6 | | |
| | | | | 55 ♔c1 ♗d3 56 ♔d1 ♔b2 | 774,460 | 10.866 |
| 52 ♔c1 | (0.25) | b5 | (2.00) | 53 ♔b2 ♔c4 54 ♔c1 ♔c3 55 ♔d1 a6 | | |
| | | | | 56 ♔c1 ♗d3 57 ♔d1 ♔b2 | 752,079 | 11.151 |
| 53 ♔b2 | (1.57) | ♔c4 | (0.02) | 54 ♔c1 b4 55 ♔b2 ♗d3 | | |
| | | | | 56 ♔c1 b3 57 ♔b2 ♗e4 | 834,892 | 11.319 |
| 54 a6 | (0.06) | b4 | (2.00) | 55 ♔c1 ♔c3 56 ♔d1 ♗d3 57 ♔e1 b3 58 ♔f2 | 912,576 | 20.412 |
| 55 ♔a2 | (1.28) | ♔c3 | (0.02) | Cray Blitz announced mate in six. | | |
| | | | | 56 ♔a1 ♗c2 57 ♔a2 b3+ 58 ♔a3 b2 | | |
| | | | | 59 ♔a2 b1♕+ 60 ♔a3 ♕b2 mate. | | |

**1983 United States Open Championship**

Four computers played in the 1983 U.S. Open Chess tournament August 7-19 in Pasadena, California. This is a 12-round tournament played one round each evening with a time limit of 50 moves in 150 minutes. There were 100 players rated 2200 or better in a field of 820 players. The winners were Korchnoi and Christiansen with 10½ points. Belle finished with 8½ points. Its performance rating was 2363. Super Constellation became the first micro to beat a master over the board.

**Super Constellation – J. Simon (2203)** 1 c4 e5 2 ♘c3 ♘c6 3 ♘f3 ♘f6 4 g3 d6 5 ♗g2 ♗e6 6 d3 ♗e7 7 0-0 ♕c8 8 ♕b3 0-0 9 ♘d5 ♗d8 10 ♗g5 ♖b8 11 ♘×f6+ g×f6 12 ♗h6 ♖e8 13 ♕c3 ♗h3 14 ♖ac1 ♗×g2 15 ♔×g2 ♕g4 16 ♗d2 f5 17 h3 ♕h5 18 e3 ♖e6 19 d4 e4 20 ♘g1 ♖g6 21 f3 ♗h4 22 ♘e2 ♗g5 23 ♘f4 ♗×f4 24 e×f4 ♖e8 25 f×e4 ♖×e4 26 ♖ce1 ♖e2+ 27 ♖f2 ♖×f2+ 28 ♔×f2 ♕×h3 29 ♖e8+ ♔g7 30 d5+ ♔h6 31 d×c6 ♕h2+ 32 ♔f1 ♖×g3 33 ♗e3 ♕h1+ 34 ♔e2 ♕h5+ 35 ♔d3 b×c6 36 ♕f6+ ♕g6 37 ♕h4+ ♔g7 38 ♕d8 ♔h6 39 ♕×c7 d5 40 ♕×a7 d×c4+ 41 ♔×c4 ♖g2 42 ♕d4 ♖c2+ 43 ♔b3 c5 44 ♕d5 ♖h2 45 ♕×c5 ♖h3 46 ♔c2 ♖h2+ 47 ♗d2 ♕g2 48 ♕f8+ ♔h5 49 ♕×f7+ ♔h4 50 ♕e7+ ♔h5 51 ♕×h7+ ♔g4 52 ♕g7+ ♔f3 53 ♕c3+ ♔f2 54 ♕e3+ ♔f1 55 ♕e1# 1-0

**Belle – Fred Munson (1600)** 1 e4 e5 2 ♗c4 ♘f6 3 d4 e×d4 4 ♘f3 ♗b4+ 5 c3 d×c3 6 b×c3 ♗c5 7 e5 ♘g4 8 ♗×f7+ ♔f8 9 ♗h5 ♘×f2 10 ♕d5 g6 11 ♕×c5+ ♔g7 12 ♕×f2 ♖f8 13 ♕g3 d6 14 0-0 d×e5 15 ♕×e5+ ♕f6 16 ♕×c7+ ♔h8 17 ♘fd2 ♕b6+ 18 ♕×b6 ♖×f1+ 19 ♔×f1 a×b6 20 ♗f3 ♘c6 21 ♘c4 ♗e6 22 ♘ba3 b5 23 ♘d6 b4 24 ♗×c6 b×c6 25 c×b4 ♖a4 26 b5 c×b5 27 ♘d×b5 ♖e4 28 ♗b2+ ♔g8 29 ♘d4 ♗c4+ 30 ♘×c4 1-0

**Bruce Boer (1553) – Belle** 1 c4 e6 2 b3 ♘f6 3 ♘f3 d5 4 ♗b2 ♗e7 5 g3 0-0 6 ♗g2 a5 7 a3 ♘c6 8 d4 ♗d7 9 0-0 ♘e4 10 ♘bd2 f5 11 c×d5 e×d5 12 ♘e5 ♘×e5 13 d×e5 ♗e6 14 f4 ♗c5+ 15 ♔h1 ♘f2+ 16 ♖×f2 ♗×f2 17 ♘f3 c6 18 ♘g5 ♕d7 19 ♕d3 d4 20 ♕f3 ♗e3 21 ♗c1 ♗d5 22 ♕h5 ♗×g2+ 23 ♔×g2 ♕d5+ 24 ♔h3 h6 25 ♗×e3 h×g5 26 ♖d1 g4+ 27 ♔h4 ♕e6 28 ♖×d4 g6 29 ♖d6 ♕e7+ 30 ♖f6 ♖×f6 31 e×f6 ♕×f6+ 32 ♕g5 ♕h8+ 0-1

**Belle – Peter Negel (1856)** 1 e4 d5 2 e×d5 ♘f6 3 d4 ♘×d5 4 ♘f3 ♗g4 5 ♗e2 e6 6 c4 ♘b6 7 0-0 ♗e7 8 ♘c3 ♘c6 9 d5 e×d5 10 c×d5 ♗×f3 11 ♗×f3 ♘e5 12 ♗e4 ♕d7 13 ♖e1 f5 14 ♗c2 ♘g6 15 ♗b3 0-0-0 16 ♗e3 ♗d6 17 ♗×b6 a×b6 18 ♗c2 ♖hf8 19 ♕h5 ♖f6 20 ♗a4 ♕f7 21 ♖ad1 ♗b4 22 ♗b3 ♗d6 23 ♘b5 ♘f8 24 ♕h4 ♖h6 25 ♘a7+ ♔d7 26 ♕a4+ b5 27 ♕×b5+ c6 28 ♕×b7+ ♗c7 29 d×c6# 1-0

**George Church (1891) – Belle** 1 e4 e5 2 ♘c3 ♘f6 3 ♗c4 ♘c6 4 f4 ♘×e4 5 ♘f3 ♘×c3 6 d×c3 ♕e7 7 ♕e2 e×f4 8 0-0 ♕×e2 9 ♗×e2 ♗c5+ 10 ♔h1 ♗e3 11 ♗c4 0-0 12 ♗×e3 f×e3 13 ♖ae1 ♘a5 14 ♗d3 ♖e8 15 ♘g5 h6 16 ♘×f7 d6 17 ♗g6 ♗d7 18 ♘×d6 c×d6 19 ♗f7+ ♔h7 20 ♗×e8 ♖×e8 21 b3 ♗c6 22 ♖f5 b5 23 c4 ♔g6 24 ♖f4 b×c4 25 b4 ♘b7 26 ♖×c4 ♘d8 27 ♖g4+ ♔h7 28 ♖g3 e2 29 ♔g1 ♗b5 30 ♖g4 ♘c6 31 a3 ♖e3 32 ♔f2 ♖×a3 33 ♖×e2 ♗×e2 34 ♔×e2 ♖a4 35 c3 a5 36 ♖c4 ♖a2+ 37 ♔e3 ♘e5 38 ♖d4 ♖×g2 39 ♔e4 a×b4 40 c×b4 ♘c6 41 ♖×d6 ♘×b4 42 ♔f3 ♖×h2 43 ♔g3 ♖c2 44 ♔h3 ♖c4 45 ♔g3 g5 46 ♖e6 h5 47 ♔h3 ♖c3+ 48 ♔h2 g4 49 ♔g2 ♘d5 50 ♖e4 h4 51 ♔h2 ♘e3 52 ♔g1 ♖c1+ 53 ♔f2 g3+ 54 ♔×e3 ♖e1+ 0-1

**Mitchel Saltzberg (2310) – Belle** 1 d4 d5 2 c4 e6 3 ♘c3 c6 4 e4 d×e4 5 ♘×e4 ♗b4+ 6 ♘c3 ♘f6 7 ♘f3 c5 8 a3 ♗×c3+ 9 b×c3 ♕a5 10 ♗d2 ♗d7 11 ♗d3 ♗c6 12 ♕e2 0-0 13 0-0 ♘bd7 14 ♖fe1 ♗×f3 15 ♕×f3 c×d4 16 ♕g3 ♔h8 17 ♖eb1 d×c3 18 ♖b5 ♕a6 19 ♗×c3 ♕c6 20 ♖e1 ♖g8 21 ♕h4 a6 22 ♖g5 ♕a4 23 ♖g3 b6 24 ♖f3 e5 25 ♗f5 ♖gc8 26 ♗×d7 ♘×d7 27 ♖h3 h6 28 ♕e7 ♖×c4 29 ♗d2 ♔g8 30 ♗×h6 g×h6 31 ♖×h6 ♖g4 32 h3 ♖g7 33 g4 ♖g6 34 ♖×g6+ f×g6 35 ♖c1 ♖f8 36 ♕g5 ♕e4 37 ♖d1 ♖f3 38 ♕h4 ♖×a3 39 ♕e7 ♖d3 40 ♖c1 ♘f8 0-1

**Belle – Leslie Au (2228)** 1 e4 c5 2 c3 ♕c7 3 ♘f3 d6 4 d4 a6 5 h3 ♘d7 6 ♘bd2 ♘gf6 7 ♗e2 g6 8 0-0 ♗g7 9 b4 0-0 10 b×c5 d×c5 11 e5 ♘d5 12 ♘e4 c×d4 13 c×d4 ♘b8 14 ♕b3 ♗e6 15 ♗c4 ♖d8 16 ♘c5 b5 17 ♘×e6 f×e6 18 ♗e2 ♘c6 19 a4 ♘a5 20 ♕a2 b4 21 ♘g5 ♕b6 22 ♗g4 b3 23 ♕a3 ♘c4 24 a5 ♕b5 25 ♗×e6+ ♔f8 26 ♘×h7+ ♔e8 27 ♕a4 b2 28 ♕c2 ♖db8 29 ♕×g6+ ♔d8 30 ♗×b2 ♘×b2 31 ♕×g7

♘d3 32 ♕g8+ 1-0

**Belle – Harry Radke (2313)** 1 e4 d6 2 d4 ♘f6 3 ♘c3 g6 4 f4 ♗g7 5 ♘f3 c5 6 e5 ♘g4 7 d×c5 d×e5 8 ♕×d8+ ♔×d8 9 h3 ♘h6 10 f×e5 ♘d7 11 ♘e4 f5 12 e×f6 e×f6 13 ♘d6 ♔e7 14 ♗e3 b6 15 ♗c4 g5 16 0-0-0 b×c5 17 ♗b5 ♘e5 18 ♘×e5 f×e5 19 ♖hf1 1-0

**Belle – William Wharton (2236)** 1 e4 c5 2 c3 e6 3 d4 d5 4 e×d5 e×d5 5 ♘f3 ♘c6 6 ♗e3 c×d4 7 ♗×d4 ♗e6 8 ♘bd2 ♘f6 9 ♕b3 ♗d6 10 ♗×f6 g×f6 11 ♕×b7 ♘e5 12 ♗b5+ ♔f8 13 ♕a6 ♖b8 14 ♘d4 ♕b6 15 ♕×b6 ♖×b6 16 f4 ♘g4 17 g3 h5 18 ♘2b3 h4 19 ♔e2 h3 20 ♖ae1 ♔g7 21 ♘×e6+ f×e6 22 ♗d7 ♘h6 23 ♗×e6 ♗a3 24 ♗×d5 ♗×b2 25 ♔d3 ♖d8 26 c4 ♘g4 27 ♖e7+ ♔g6 28 ♔c2 ♗a3 29 ♖×a7 ♗b4 30 c5 ♗×c5 31 ♗f7+ ♔h6 32 ♘×c5 ♖c6 33 ♖a5 ♖dc8 34 a4 ♖×c5+ 35 ♖×c5 ♖×c5+ 36 ♔b3 f5 37 ♗c4 ♘e3 38 ♖c1 ♘d5 39 ♖e1 ♘f6 40 ♗f1 ♘g4 41 ♗×h3 ♘×h2 42 ♖h1 1-0

**Craig Mar (2403) – Belle** 1 b3 e5 2 ♗b2 ♘c6 3 c4 ♘f6 4 e3 d6 5 ♘e2 ♗e7 6 ♘bc3 0-0 7 d3 ♖e8 8 ♘g3 d5 9 c×d5 ♘×d5 10 ♗e2 ♗a3 11 ♕d2 ♗b4 12 0-0 ♘d4 13 ♕c1 ♘×e2+ 14 ♘c×e2 c6 15 e4 ♘f6 16 ♕e3 ♘g4 17 ♕f3 ♕h4 18 h3 ♘f6 19 ♘f5 ♕g5 20 ♗c1 ♕g6 21 ♘eg3 h6 22 ♘h4 ♕h7 23 ♗b2 b6 24 a3 ♗f8 25 ♕e2 ♗a6 26 ♘gf5 ♖ad8 27 ♖ad1 ♘d7 28 b4 c5 29 ♕c2 c×b4 30 a×b4 ♗×b4 31 ♕a4 ♗a5 32 ♗a3 g5 33 ♘e7+ ♔h8 34 ♘hf5 ♗b7 35 ♘c6 ♗×c6 36 ♕×c6 f6 37 ♘d6 ♘b8 38 ♕c4 ♖e7 39 d4 e×d4 40 ♕×d4 ♖f8 41 ♘f5 ♖d7 42 ♗×f8 ♖×d4 43 ♖×d4 ♕b7 44 ♖fd1 ♔h7 45 ♖d8 ♕×e4 46 ♘×h6 ♕b7 47 ♘g4 ♗c3 48 ♖e8 ♘c6 49 ♖e6 ♔g8 50 ♗a3 ♔f7 51 ♖ed6 ♕a6 52 ♖d7+ ♔e8 53 ♗c1 ♕a4 54 ♖7d3 ♔f7 55 ♘e3 ♗e5 56 ♘f5 ♔g6 57 g4 ♔h7 58 ♗a3 ♕f4 59 ♖d7+ ♔h8 60 ♖1d3 ♕h2+ 61 ♔f1 ♕h1+ 62 ♔e2 a5 63 ♗f8 ♕e4+ ½-½

# 9.6 Miscellaneous Games Played By Computer Programs

The three games that follow were played by Bobby Fischer against the program written by Richard Greenblatt at M.I.T. The games were published in issue 2 of the Computer Chess Newsletter, 1977

**Greenblatt – Fischer** 1 e4 c5 2 ♘f3 d6 3 d4 cd 4 ♘×d4 ♘f6 5 ♘c3 a6 6 ♗e2 e5 7 ♘b3 ♗e7 8 ♗e3 0-0 9 ♕d3 ♗e6 10 0-0 ♘bd7 11 ♘d5 ♖c8 12 ♘×e7+ ♕×e7 13 f3 d5 14 ♘d2 ♕b4 15 ♘b3 de 16 ♕d1 ♘d5 17 ♗a7 b6 18 c3 ♕e7 19 fe ♘e3 20 ♕d3 ♘×f1 21 ♕×a6 ♘e3 22 ♗×b6 ♕g5 23 g3 ♖a8 24 ♗a7 h5 25 ♕b7 h4 26 ♔f2 hg+ 27 hg f5 28 ef ♖×f5+ 29 ♔e1 ♖af8 30 ♔d2 ♘c4+ 31 ♔c2 ♕g6 32 ♕e4 ♘d6 33 ♕c6 ♖f2 34 ♔d1 ♗g4 35 ♗×f2 ♕d3+ 36 ♔c1 ♗×e2 37 ♘d2 ♖×f2 38 ♕×d7 ♖f1+ 39 ♘×f1 ♕d1 mate

**Fischer – Greenblatt** 1 e4 e5 2 f4 ef 3 ♗c4 d5 4 ♘c3 ♗b4 5 ♘c3 ♗b4 6 ♘f3 0-0 7 0-0 ♘×d5 8 ♘×d5 ♗d6 9 d4 g5 10 ♘×g5 ♕×g5 11 e5 ♗h3 12 ♖f2 ♗×e5 13 de c6 14 ♗×f4 ♕g7 15 ♘f6+ ♔h8 16 ♕h5 ♖d8 17 ♕×h3 ♘a6 18 ♖f3 ♕g6 19 ♖c1 ♔g7 20 ♖g3 ♖h8 21 ♕h6 mate

**Greenblatt – Fischer** 1 e4 c5 2 ♘f3 g6 3 d4 ♗g7 4 ♗×d5 ♘f6 5 ♘×d4 ♘c6 6 ♗e3 ♘f6 7 ♘×c6 bc 8 e5 ♘g8 9 f4 f6 10 ef ♘×f6 11 ♗c4 d5 12 ♗e2 ♖b8 13 b3 ♘g4 14 ♗d4 e5 15 fe 0-0 16 ♗×g4 ♕h4+ 17 g3 ♕×g4 18 ♕×g4 ♗×g4 19 ♖f1 ♖×f1+ 20 ♔×f1 c5 21 ♗f2 ♗×e5 22 ♗e1 ♖f8+ 23 ♔g2 ♖f3 24 h3 ♖×c3 25 ♗×c3 ♗×c3 26 ♖f1 ♗f5 27 ♖f2 h5 28 ♖e2 ♔f7 29 ♖e3 ♗d4 30 ♖f3 ♔e6 31 c3 ♗e5 32 ♖e3 d4 33 cd cd 34 ♖e1 d3 35 h4 d2 36 ♖d1 ♗c3 37 ♔f2 ♗g4 38 ♖h1 ♗d4+ 39 ♔g2 ♔d5 40 a3 ♔e4 41 ♖f1 ♔d3 42 ♔h2 ♔e2 43 ♔g2 ♗h3+ 44 ♔×h3 ♔×f1 45 b4 d1=♕ 46 ♔h2 ♕e2+ 47 ♔h3 ♕g2 mate

# 9.7 The World Microcomputer Championships

Pressure of space prevents us from giving the games of the World Microcomputer Championships, which began in 1980. The tables of results are given below.

**First World Microcomputer Championship, London, England, September 4th - 6th 1980**

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 Chess Challenger (USA) | x | 1 | . | . | 1 | . | 1 | . | . | 1 | . | 1 | . | . | 5 |
| 2 Boris Experimental (USA) | 0 | x | 1 | . | . | . | 1 | 1 | . | . | . | . | . | 1 | 4 |
| 3 Mike 3.0 (England) | . | 0 | x | ½ | . | . | ½ | . | . | 1 | 1 | . | . | . | 3 |
| 4 Rook 4.0 (Sweden) | . | . | ½ | x | . | 1 | . | . | 0 | 1 | . | ½ | . | . | 3 |
| 5 Sargon 2.0 (USA) | 0 | . | . | . | x | . | . | . | 1 | . | 0 | 1 | 1 | . | 3 |
| 6 Gambiet (Netherlands) | . | . | . | 0 | . | x | 0 | 1 | . | . | 1 | 1 | . | . | 3 |
| 7 Modular Game System (USA) | 0 | 0 | ½ | . | . | 1 | x | . | 1 | . | . | . | . | . | 2½ |
| 8 Auto Response Board (USA) | . | 0 | . | . | . | 0 | . | x | ½ | . | 1 | . | 1 | . | 2½ |
| 9 Vega 1.7 (England) | . | . | . | 1 | 0 | . | 0 | ½ | x | . | . | . | 1 | . | 2½ |
| 10 Viktor (Switzerland) | 0 | . | 0 | 0 | . | . | . | . | . | x | . | . | 1 | 1 | 2 |
| 11 Albatross (England) | . | . | 0 | . | 1 | 0 | . | 0 | . | . | x | . | . | 1 | 2 |
| 12 Fafner 2 (England) | 0 | . | . | ½ | 0 | 0 | . | . | . | . | . | x | . | 1 | 1½ |
| 13 Princhess 1.0 (Sweden) | . | . | . | . | 0 | . | . | 0 | 0 | 0 | . | . | x | 1 | 1 |
| 14 K. Chess IV (England) | . | 0 | . | . | . | . | . | . | . | 0 | 0 | 0 | 0 | x | 0 |

**Second World Microcomputer Championship**
**Travemünde, West Germany, September 21st - 29th 1981**

| *Open Section* | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | total |
|---|---|---|---|---|---|---|---|---|---|
| 1 Fidelity X | x | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 6 |
| 2 Princhess 2.9 | 0 | x | 0 | 1 | 1 | 1 | 1 | 1 | 5 |
| 3 Novag X | 0 | 1 | x | 1 | ½ | 0 | 1 | 1 | 4½ |
| 4 Philidor X | 1 | 0 | 0 | x | 1 | ½ | 1 | 1 | 4½ |
| 5 LogiChess 2.1 | 0 | 0 | ½ | 0 | x | ½ | ½ | 1 | 2½ |
| 6 Conic X | 0 | 0 | 1 | ½ | ½ | x | 0 | 0 | 2 |
| 7 SciSys X | 0 | 0 | 0 | 0 | ½ | 1 | x | ½ | 2 |
| 8 Applied Concepts X | 0 | 0 | 0 | 0 | 0 | 1 | ½ | x | 1½ |

| *Commercial Section* | 1 | 2 | 3 | 4 | total |
|---|---|---|---|---|---|
| 1 Chess Champion Mark V | | 1½ | 3½ | 3½ | 8½ |
| 2 Champion Sensory Challenger | 2½ | | 1½ | 2½ | 6½ |
| 3 Savant | ½ | 2½ | | 3 | 6 |
| 4 Gruenfeld/Morphy/Capablanca | ½ | 1½ | 1 | | 3 |

| *Playoff, September 27-29* | 1 | 2 | 3 | total |
|---|---|---|---|---|
| Fidelity X | 1 | 1 | 1 | 3 |
| Chess Champion Mark V | 0 | 0 | 0 | 0 |

**Third World Microcomputer Championship, Budapest, Hungary, 1983**

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | total |
|---|---|---|---|---|---|---|---|---|
| 1 Elite A/S | 10+ | 15+ | 2+ | 3= | 8+ | 4= | 13+ | 6 |
| 2 Mephisto X | 11+ | 6+ | 1– | 8= | 5= | 12+ | 3+ | 5 |
| 3 Novag X | 16+ | 9= | 5+ | 1= | 10+ | 13+ | 2– | 5 |
| 4 Super Constellation | 12= | 5– | 15+ | 9+ | 6+ | 1= | 10+ | 5 |
| 5 Prestige | 8+ | 4+ | 3– | 11+ | 2= | 10+ | 6– | 4½ |
| 6 Chess 2001 | 14+ | 2– | 10– | 7+ | 4– | 15+ | 5+ | 4 |
| 7 Gedeon X | – | – | 18+ | 6– | 16+ | 8+ | 11+ | 4 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 8 Chess 2001 X | 5– | 16+ | 9+ | 2= | 1– | 7– | 15+ | 3½ |
| 9 Mephisto Y | 13+ | 3= | 8– | 4– | 14+ | 11– | 16+ | 3½ |
| 10 Mephisto Excalibur | 1– | 17+ | 6+ | 12+ | 3– | 5– | 4– | 3 |
| 11 Constellation | 2– | 12= | 17+ | 5– | 15= | 9+ | 7– | 3 |
| 12 Sensory 9 | 4= | 11= | 14+ | 10– | 13– | 2– | 17+ | 3 |
| 13 Superstar X | 9– | 14– | 16+ | 18+ | 12+ | 3– | 1– | 3 |
| 14 Micromurks | 6– | 13+ | 12– | 15= | 9– | 17– | 18+ | 2½ |
| 15 Logichess 2.2 | 17+ | 1– | 4– | 14= | 11= | 6– | 8– | 2 |
| 16 Chessmaster | 3– | 8– | 13– | 17+ | 7– | 18+ | 9– | 2 |
| 17 65 Cyrus X | 15– | 10– | 11– | 16– | 18+ | 14+ | 12– | 2 |
| 18 Labirint 64 | – | – | 7– | 13– | 17– | 16– | 14– | 0 |

**Fourth World Microcomputer Championship, Glasgow, Scotland, 1984**

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | total |
|---|---|---|---|---|---|---|---|---|
| 1 Elite X | 2= | 18+ | 8+ | 4+ | 5= | 3– | 11+ | 5 |
| 2 Mephisto A | 1= | 10+ | 16+ | 5= | 12+ | 4= | 3= | 5 |
| 3 Princhess X | 6+ | 15+ | 4– | 12= | 8+ | 1+ | 2= | 5 |
| 4 Psion | 9+ | 17+ | 3+ | 1– | 6+ | 2= | 5= | 5 |
| 5 Conchess X | 11+ | 16= | 15+ | 2= | 1= | 6= | 4= | 4½ |
| 6 Elegance | 3– | 19+ | 17+ | 11+ | 4– | 5= | 16+ | 4½ |
| 7 Elite Y | 16– | 11= | 19+ | 13= | 17+ | 8= | 14+ | 4½ |
| 8 Mephisto C | 10+ | 13= | 1– | 16+ | 3– | 7= | 12+ | 4½ |
| 9 Private Line | 4– | bye= | 11– | 19+ | 18= | 13+ | 17+ | 4 |
| 10 Sargon/Apple | 8– | 2– | 18+ | 14+ | 13+ | 11– | 19+ | 4 |
| 11 Mephisto B | 5– | 7= | 9+ | 6– | 15+ | 10+ | 1– | 3½ |
| 12 Sargon/MacIntosh | 17– | 14+ | 13+ | 3= | 2– | 16+ | 8– | 3½ |
| 13 Conchess Y | 18+ | 8= | 12– | 7= | 10– | 9– | bye= | 2½ |
| 14 Intelligent Chess | 15– | 12– | bye= | 10– | 19+ | 18+ | 7– | 2½ |
| 15 Sargon/Compaq | 14+ | 3– | 5– | 17= | 11– | 19+ | 18– | 2½ |
| 16 Colossus 2.1 | 7+ | 5= | 2– | 8– | bye= | 12– | 6– | 2 |
| 17 L'Empereur | 12+ | 4– | 06– | 15= | 7– | bye= | 9– | 2 |
| 18 Orwell | 13– | 1– | 10– | bye= | 9= | 14– | 15+ | 2 |
| 19 Chessnut 3 | bye= | 6– | 7– | 9– | 14– | 15– | 10– | ½ |

**Fifth World Microcomputer Championship, Amsterdam, September 7th - 15th 1985**

8 rounds, Swiss System: 1 Mephisto Amsterdam, 8; 2-3 Mephisto Amsterdam 2, Mephisto Amsterdam 3, 7; 4 Princhess 6, 4½; 5-6 Blitz Monster Y, Plymate, 4; 7-10 Orwell X, Orwell Y, Plymate Z, Turbostar K, 3½; 11-14 Blitz Monster C, Orwell Z, Plymate X, Turbostar 440, 3; 15 Turbostar G, 2½; 16 Blitz Monster X, 1.

**Sixth World Microcomputer Championship, Dallas, USA, 1986**

| | rate | perf | 1 | 2 | 3 | 4 | 5 | 6 | 7 | total |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 Mephisto 3 | 1000 | 1255 | 11+ | 7+ | 4+ | 2+ | 8+ | 9+ | 6– | 6 |
| 2 Fidelity A | 1000 | 1190 | 14+ | 3+ | 5+ | 1– | 6= | 8+ | 12+ | 5½ |
| 3 Mephisto 2 | 1000 | 1137 | 10+ | 2– | 6= | 11+ | 7= | 13+ | 4+ | 5 |
| 4 Fidelity C | 1000 | 1088 | 9+ | 6+ | 1– | 8= | 5+ | 12+ | 3– | 4½ |
| 5 Mephisto 1 | 1000 | 1088 | 8+ | 10+ | 2– | 7+ | 4– | 11+ | 9= | 4½ |
| 6 Recom A | 1000 | 1044 | 12+ | 4– | 3= | 10+ | 2= | 7– | 1+ | 4 |
| 7 Fidelity B | 1000 | 999 | 13+ | 1– | 12+ | 5– | 3= | 6+ | 10– | 3½ |
| 8 Recom C | 1000 | 999 | 5– | 14+ | 11+ | 4= | 1– | 2– | 13+ | 3½ |
| 9 Recom B | 1000 | 999 | 4– | 11– | 14+ | 12+ | 10+ | 1– | 5= | 3½ |
| 10 Cyrus B | 1000 | 955 | 3– | 5– | 13+ | 6– | 9– | 14+ | 7+ | 3 |
| 11 Cyrus A | 1000 | 910 | 1– | 9+ | 8– | 3– | 13= | 5– | 14+ | 2½ |
| 12 Cyrus C | 1000 | 862 | 6– | 13+ | 7– | 9– | 14+ | 4– | 2– | 2 |
| 13 Chess Monster | 1000 | 808 | 7– | 12– | 10– | 14+ | 11= | 3– | 8– | 1½ |
| 14 Kempelen | 1000 | 527 | 2– | 8– | 9– | 13– | 12– | 10– | 11– | 0 |

# Bibliography

This bibliography contains every reference found in each of the papers in this collection. I have added a number of items contained in the bibliography published by Tony Marsland as a Technical Report from the University of Alberta, and I have also added other references which are more recent than Marsland's latest edition.

Altogether there are almost 400 references in this bibliography, yet it is certainly not fully exhaustive. There must almost certainly be some items which have escaped the net by accident, and I would be very pleased for additions and corrections to be sent to me, via the publisher, so that future editions of this volume may be even more complete and accurate.

The following abbreviations are employed in this bibliography:

| | |
|---|---|
| ACM | Association for Computing Machinery |
| FJCC | Fall Joint Computer Conference |
| IJCAI | International Joint Conference on Artificial Intelligence |
| MIRU | Machine Intelligence Research Unit, Edinburgh University |
| SIGART | ACM Special Interest Group on Artificial Intelligence |
| SJCC | Spring Joint Computer Conference |
| WJCC | Western Joint Computer Conference |
| *** | Indicates a reference which is published in this compendium. |

Readers are also referred to the ICCA Journal, which is the best source of regular papers on computer chess. The Journal is sent free to all members of the ICCA (International Computer Chess Association).

Adelson-Velskiy, G. M., Arlazarov, V. L., Bitman, A. R., Zhivotovskii, A. A. and Uskov, A. V. (1969): Programming a Computer to Play Chess. *Proc. 1st summer school Math. Prog. vol 2,* 1969, pp216-252. Also Russian Math. Surveys, vol 25, no 2, Mar-Apr 1970, pp 221-262.

Adelson-Velskiy, G. M., Arlazarov, V. L. and Donskoy, M. V. (1974): More Commentary on the Cichelli heuristics. *SIGART* 45, Apr. 74, p 12

Adelson-Velskiy, G. M., Arlazarov, V. L. and Donskoy, M. V. (1975): On the Structure of an Important Class of Exhaustive Problems and On Ways of Search Reduction for Them. *Proc. 4th IJCAI*, Tbilisi, Sept. 75, pp 304-308. Also Advances in Computer Chess 1: Clarke M. R. B. (ed.), pp 1-6

Adelson-Velskiy, G. M., Arlazarov, V. L. and Donskoy, M. V. (1975): Some Methods of Controlling the Tree Search in Chess Programs. *Artificial Intelligence*, vol 6, 1975, pp 361-371 ***

Adelson-Velskiy, G. M., Arlazarov, V. L. and Donskoy, M. V. (1979): Algorithms of adaptive search. Elcock, E. W. and Michie, D. (eds.), *Machine Intelligence* 9, pp 373-384. Wiley, New York.

Adelson-Velskiy, G. M., Arlazarov, V. L. and Uskov, A. G. (1966): Programme playing chess. *Report on Symposium on Theory and Computing Methods in the Upper Mantle Problem*

Akl, S. G. and Newborn, M. M. (1977): The principal continuation and the killer heuristic. *Proc. ACM77*, Seattle, pp 466-473

Alden B. E. and Bramer M. A. (1982): Development of a Program for Solving Retrograde Analysis Chess Problems. *Advances in Computer Chess 3*, Clarke M. R. B. (ed.), pp 121-138

Alekseev, V. E. (1967): Compilation of chess problems on a computer. Technical translation FSTC-HT-23-124-69, US Army, NTIS AD 689470 (*Problemy Kibernetiki*, no 19, 1967)

Alexander, C. H. O'D. (1972): *Fischer v. Spassky, Reykjavik 1972.* Penguin, Harmondsworth

Anderson, R. H. (1967): Electronic chess won by Soviet. *New York Times*, 26 Nov., 1967, p 146

Arlazarov, V. L. and Futer, A. L. (1979): *Computer Analysis of a Rook End-game.* Elcock, E. W. and Michie, D. (eds.), *Machine Intelligence 9*, pp 361-371. Horwood-Wiley, Chichester.

Arnold, G. and Newborn, M. M. (1972): A chess playing program: the OSTRICH. *ACM72 Computer chess notes*, Boston 1972, pp 10-14

Ashby, W. R. (1952): Can a Mechanical Chess Player Outplay its Designer? *British Journal of Philosophy of Science*, vol 3, 1952, pp 44-57

Atkin, L. R. (1975): *Chess 3.6: A Chess Playing Computer Program.* Master's Thesis, Northwestern University, June 1975

Atkin, L. R., Gorlen, K. and Slate, D. J. (1971): *Chess 3.0 — An Experiment in Heuristic Programming.* Privately circulated, Northwestern University.

Atkin, L. R. and Slate, D. J. (1977): CHESS 4.5 - The Northwestern University Chess Program. Frey P. W. (ed.), *Chess Skill in Man and Machine*, pp 82-118 ***

Atkin, R. H. (1972): Multi-dimensional Structure in the Game of Chess. *Int. Journ. of Man-Machine Studies*, vol 4, pp 341-362

Atkin, R. H. (1977): Positional Play in Chess by Computer. *Advances in Computer Chess 1*, Clarke M. R. B. (ed.), pp 60-73

Atkin, R. H. and Witten, I. H. (1975): A Multidimensional Approach to Positional Chess. *Int. Journ. of Man-machine Studies*, vol 7, 1975, pp 727-750. Also Bell A. G. (ed.) Computer Chess, pp 37-79

Atkin, R. H., Harston, W. R. and Whitten, I. H. (1976): Fred CHAMP, Positional-Chess Analyst. *Int. Journ. of Man-Machine Studies*, vol 8, June 1976, pp 517-529

Averbakh, Y. and Chekhover, V. (1977): *Knight Endings*. Batsford, London.

Averbakh, Y. and Maizelis, I. (1974): *Pawn Endings*. Chess Digest, Dallas.

Awerbach, J. (1979): *Lehrbuch der Schachendspiele, Band 1 und Band 2*, autorisierte Uebersetzung aus dem Russischen: Hermann Mohaupt. Sportverlag Berlin

Babaoglu, O. (1977): *Hardware Implementation of the Legal Move Generation and Relative Ordering for the Game of Chess*. Master's Thesis, University of California, Berkeley

Banerji, R. B. (1969): *An Overview of Game Playing Programs*. Tech. Report, Case Western Reserve University, Cleveland

Banerji, R. B. (1971): Similarities in Games and Their Use in Strategy Construction. *Computers and Automata Proceedings of 21st Brooklyn Polytechnic Symposium*, pp337-357

Banerji, R. B. and Ernst, G. W. (1971): Changes in Representation which Preserve Strategies in Games. Proc.*2nd IJCAI*, pp 651-658 (longer version published as technical report of same name, Case Western Reserve University)

Barnes, C. (1976): Computer Chess Techniques. *Report for CS H198, Comp. Sci. and Elec. Eng. Dept.*, University of California at Berkeley.

Baudet, G. M. (1978): On the branching factor of the alpha-beta pruning algorithm. *Artificial Intelligence*, vol 10, pp 173-199

Baylor, G. W. (1965): *Report on a Mating Combination Program*. System Development Corporation, Santa Monica, CA, Paper SP-2150.

Baylor, G. W. (1966): A Computer Model of Checkmating Behaviour in Chess. *Heuristic Processes in Thinking*, De Groot, A. D. and Reitman, W. R., (eds.). Nauka, Moscow.

Baylor, G. W. and Simon, H. A. (1966): A Chess Mating Combinations Program. *Proc. SJCC*, pp 431-447 ***

Beal, D. F. (1977): *Discriminating wins from draws in King + Pawn versus King chess endgames*. Privately circulated, Queen Mary College London

Beal, D. F. (1980): An Analysis of Minimax. *Advances in Computer Chess 2*, Clarke M. R. B. (ed.), pp 103-109

Beal, D. F. (1982): Benefits of Minimax Search. *Advances in Computer Chess 3*, Clarke M. R. B. (ed.), pp 17-24

Beal, D. F. and Clarke, M. R. B. (1980): The Construction of Economical and Correct Algorithms for King and Pawn against King. *Advances in Computer Chess 2*, Clarke M. R. B. (ed.), pp 1-30

Bell, A. G. (1970): Algorithm 50: How to program a Computer to Play Legal Chess. *Computer Journal*, vol 13, no 2, 1970, pp 208-219

Bell, A. G. (1972): *Game Playing with Computers*. Allen and Unwin, London.

Bell, A. G. (1973): Computer Chess experiments. *Computer Chess*, Bell, A. G. (ed.), pp 1-14

Bell, A. G. (ed.) (1973): Computer Chess. *Proc. May 1973 Meeting on chess playing by computer*. Science Research Council, Atlas Computer Labs., Chilton.

Bell, A. G., Hallowell, P. J. and Long, D. H. (1973): A Universal Benchmark. *Software Practice and Experience, vol 3*, pp 355-357

Bell, A. G. and Jacobi, N. (1979): How to read, make and store chess moves. *Computer Journal 22*, 71-75

Bellman, R. (1968): Stratification and control of large systems with applications to chess and checkers. *Information Sciences 1*, pp 7-21

Bellman, R. (1965): On the Application of Dynamic Programming to the Determination of Optimal Play in Chess and Checkers. *Proc. Natl. Academy of Sciences*, vol 53, pp 244-247

Berliner, H. J. (1969): Chess Playing Programs. *SIGART 17*, pp 19-20

Berliner, H. J. (1970): Experiences Gained in Constructing and Testing a Chess Program. *Proc. IEEE Symposium on Systems Sciences and Cybernetics*, Pittsburg, pp 216-223

Berliner, H. J. (1973) Some Necessary Conditions for a Master Chess Program. *Proc. 3rd IJCAI*, Stanford, pp 77-85 ***

Berliner, H. J. (1974): *Chess as Problem Solving: The Development of a Tactics Analyzer.* Ph.D. Thesis, Carnegie-Mellon University.

Berliner, H. J. (1974): A comment on improvement of chess playing programs. *SIGART* 48, Oct. 1974, p. 16

Berliner, H. J. (1975): A new sub-field of computer chess. *SIGART* 53, Aug. 75, pp 20-21

Berliner, H. J. (1975): Computer chess. *SIGART* 55, Dec 75, pp 14-15

Berliner, H. J. (1976): Outstanding performance by CHESS 4.5 against human competition. *SIGART* 60, Nov. 76, pp 12-13

Berliner, H. J. (1976): Man against machine 1974. *Firbush News* 6, MIRU, Edinburgh University, July 1976, pp 63-70

Berliner, H. J. (1977): A Representation and Some Mechanisms for a Problem Solving Chess program. *Advances in Computer Chess 1*, Clarke M. R. B. (ed.), pp 7-29

Berliner, H. J. (1977): On the use of domain-independent descriptions in tree searching. *Perspectives in Computer Sci., 10th Aniv. of C.S.C.M.U.*, Jones (ed.). Academic Press.

Berliner, H. J. (1977): Search and Knowledge. *Proc. 5th IJCAI*, pp 975-979

Berliner, H. (1978): A chronology of computer chess and its literature. *Artificial Intelligence*, vol 10, 1978, pp 201-204

Berliner, H. J. (1978): Computer Chess. *Nature*, vol 274, 24 Aug. 1978, pp 745-748

Berliner, H. J. (1979): The B* tree search algorithm: A best-first proof procedure. *Artificial Intelligence*, vol 12, pp 23-40

Bernstein, A. and Roberts, M. de V. (1958). Computer vs Chess Player. *Scientific American*, 198, pp 96-105 ***

Bernstein, A., Roberts, M. de V., Arbuckle, T. and Belsky, M. A. (1958): A Chess Playing Program for the IBM 704. *Proc. WJCC*, pp 157-159

Bijl, C. M. (1980): *Het Eindspel, Koning + 2 Paarden tegen Koning + Pion, Theorie en Praktijk.* Leidschendam

Binet, A. (1893): Mnemonic Virtuosity: A Study of Chess Players. *Genetic Psych. Monog.*, vol 74, 1966, pp 127-162 (Translation of 1893 paper in "Revue des Deux Mondes", vol 117)

Birmingham, J. A. and Kent, P. (1977): Tree-searching and Tree-Pruning Techniques. *Advances in Computer Chess 1*, Clarke, M. R. B. (ed.), pp 89-107 ***

Birmingham, J. A. and Kent, P. (1980): Mate at a Glance. *Advances in Computer Chess 2*, Clarke M. R. B. (ed.), pp 122-130 ***

Bond, A. H. (1973): Psychology and Computer Chess.*Computer Chess*, Bell A. G. (ed.), pp 29-36

Bond, A. H. (1973): Descriptor Index. *Computer Chess*, Bell (ed.), Oct. 73, pp 95-112

Boos, G., Cooper, D. W., Gillogly, J. J., Levy, D. N. L., Raymond, H., Slate, D. J., Smith, R. C. and Mittman, B. (1971): Computer Chess Programs (Panel). *Proc. 1971 Annual ACM Conference*, pp 97-102

Boos, G. (1972): The VIKING. *ACM72 Computer chess notes*, Boston 1972, pp 41-42

Botvinnik, M. M. (1960): *One Hundred Selected Games* (translated by S. Garry). Dover, New York.

Botvinnik, M. M. (1970): *Computers, Chess and Long-Range Planning.* Springer-Verlag, New York.

Botvinnik, M. M. (1975): Will computers get self-respect. *Sovietsky Sport*, June 15.

Botvinnik, M. M. (1982): Decision Making and Computers. *Advances in Computer Chess 3*, Clarke M. R. B. (ed.), pp 169-180 ***

Botvinnik, M. M. (1984): *Computers in Chess*. Springer-Verlag, New York.

Bramer, M. A. (1975): *Representation of Knowledge for Chess Endgames*. Report from the Open University, Milton Keynes, U.K.

Bramer, M. A. (1976): Computer Chess: The knowledge approach. *Chess*, vol 41, pp 347-349

Bramer, M. A. (1977): *King and pawn against King: Some Quantitative Data*. Report from the Open University, Milton Keynes, U.K.

Bramer, M. A. (1977): *King and Pawn against King: using effective distance*. Faculty of Math., Open University, Milton Keynes, U.K.

Bramer, M. A. (1977): *Representation of knowledge for chess endgames: Towards a self-improving system*. Ph.D. Thesis, Open University, Milton Keynes, U.K.

Bramer, M. A. (1978): *Computer-generated databases for the endgame in chess*. Tech. Rept., Mathematics Faculty, Open University, Milton Keynes, U.K.

Bramer, M. A. (1980): An Optimal Algorithm for King and Pawn Against King using Pattern Knowledge. *Advances in Computer Chess 2*, Clarke M. R. B. (ed.), pp 82-96

Bramer, M. A. (1980): Pattern-Based Representations of Knowledge in the Game of Chess. *Proceedings of the Conference on Artificial Intelligence*, Amsterdam.

Bramer M. A. (1982): Machine Aided Refinement of Correct Strategies for the Endgame in Chess. *Advances in Computer Chess 3*, Clarke M. R. B. (ed.), pp 93-112

Bramer, M. A. and Clarke, M. R. B. (1979): A model for the representation of pattern-knowledge for the endgame in chess. *Int. Journ. of Man-Machine Studies, vol 11*, pp 635-649

Bratko, I. (1978): Proving correctness of strategies in the AL1 assertional language. *Information Processing Letters, vol 7*, no 5, Aug 1978, pp 223-230

Bratko, I. and Gams, M. (1982): Error Analysis of the Minimax Principle. *Advances in Computer Chess 3*, Clarke M. R. B. (ed.), pp 1-16

Bratko, I., Kopec, D. and Michie, D. (1978): Pattern-based representation of chess end-game knowledge. *Computer Journal, vol 21*, no 2, May 78, pp 149-153

Bratko, I. and Michie, D. (1978): Advice table representations of chess endgame knowledge. *Proc. 3rd Artificial intelligence and Simulation of Behaviour / GI Conf.*, Sleeman D. (ed.), pp 194-200

Bratko, I. and Michie, D. (1980): A Representation for Pattern-Knowledge in Chess Endgames. *Advances in Computer Chess 2*, Clarke M. R. B. (ed.), pp 31-54

Bratko, I. and Niblett, T. (1979): Conjectures and refutations in a framework for chess end-game knowledge. Michie, D. (ed.), *Expert Systems in the Micro-electronic Age*, pp 83-101. Edinburgh University Press.

Bratko, I., Tancig, P. and Tancig, S. (1976): Some new aspects of chess board reconstruction experiments. *3rd European meeting on Cybernetic and Systems Research*, Vienna.

Bratko, I. and Tancig P. (1975): On the role of strategy in computer chess. *3rd Int. Conf. on Cybernetics and General Systems*, Bucharest 1975. Abacus Press.

Brudno, A. L. (1968): Bounds and Valuations for Shortening the Scanning of Variations. *Probl. Kibernetiki, vol 10*, pp 141-150.

Buckholtz, T. J. and Wetherell, C. S. (1975): A Program to Referee Kriegspiel and Chess. *Computer Journal, vol 18*, no 2, May 1975, pp 177-183

Burger, J. F. (1967): UMPIRE: An Automatic Kriegspiel Referee for a Time-Shared Computer. *Proc. 22nd Nat. Conf., ACM, 1967*, pp 187-193

Byard, S. (19??): Robots which play Games. *Science News*, no 16.

Byrne, R. (1978): Chess: Ma Bell wins with upset in an inhuman competition. *New York Times*, 11 Dec. 1978

Cahlander, D. (1977): *Tournament games of a chess playing computer.* CDC Corp., Minneapolis.

Callon, R. and Willoner, R. (1975): *A Chess-problem Solver.* For Course CS502, University of British Columbia.

Carnap, R. (1937): *Logical syntax of language.* Harcourt, Brace and Co., New York.

Cerf, V. and Kilne, C. (1969): *The Greenblatt Chess Program.* Unpublished term paper at UCLA.

Charness, N. (1974): *Memory for chess positions: The effects of inference and input modality.* Ph.D. dissertation, Carnegie-Mellon University.

Charness, N. (1977): Human chess skill. *Chess Skill in Man and Machine*, Frey P. W. (ed.), pp 34-53

Chase, W. G. and Simon, H. A. (1973): Perception in Chess.*Cog. Psych.*, vol 4, no 1, January 1973, pp 55-81

Chase, W. G. and Simon, H. A. (1973): The mind's eye in chess. Visual Information Processing, Chase W. G. (ed.),*Proc. of Eighth Annual Carnegie Psychology Syposium.* Academic Press, New York.

Chernev (1937): *Curious Chess Facts.* The Black Knight Press.

Cheron, A. (1970-1973): *Lehr- und Handbuch der Endspiele, Band I, II, III und IV.* Berlin-Frohnau: Siegfried Engelhardt Verlag.

Church, K. W. (1978): *Co-Ordinate Squares: A solution to Many Chess Pawn Endingames.* B.Sc. thesis, Dept. of Electrical Engineering, MIT. ***

Church, R. M. and Church, K. W. (1977): Plans, Goals, and Search Strategies for the Selection of a Move in Chess. *Chess Skill in Man and Machine*, Frey P. W. (ed.), pp 131-156

Cichelli, R. J. (1973): Research progress report in computer chess. *SIGART 41*, June 1973, pp 32-36

Cichelli, R. J. (1973): Preliminary testing of the effectiveness of the Cichelli depth-2 and refutation heuristics. *SIGART 42*, Oct. 1973, pp 49-52

Cichelli, R. J. (1973): Reader Commentary on the Cichelli Heuristics. *SIGART*, Dec. 1973, pp 27-28

Citrenbaum, R. L. (1970): *Efficient Representation of Optimal Solutions for a Class of Games.* Thesis, Case Western Reserve University, Cleveland, Tech. Report SRC-69-5

Clarke, M. R. B. (1973): Some Ideas for a Chess Compiler. *Artificial and Human Thinking*, Elithorn A. and Jones (eds.), pp 189-198. Elsevier. ***

Clarke, M. R. B. (1977): A Quantitative Study of King and Pawn Against King. *Advances in Computer Chess 1*, Clarke M. R. B. (ed.), pp 108-118

Clarke, M. R. B. (editor): *Advances in Computer Chess 1.* Edinburgh University Press, 1977

Clarke, M. R. B. (editor): *Advances in computer Chess 2.* Edinburgh University Press, 1980

Clarke, M. R. B. (1980): Kings and Pawns: The Theory of Coordinate Squares. *Advances in Computer Chess 2*, Clarke M. R. B. (ed.), pp 97-102

Clarke, M. R. B. (editor): *Advances in Computer Chess 3.* Pergamon Press, 1982.

Condon, J. H. and Thompson K. (1982): Belle Chess Hardware.*Advances in Computer Chess 3*, Clarke M. R. B. (ed.), pp 45-54 ***

Condon, J. H. and Thompson K. (1983): Belle.*Chess Skill in Man and Machine*, Frey P. W. (ed.), 2nd edition, pp 201-210

Cooper, D. W. (1972): *Heuristic tree searching in the game of chess.* Privately circulated.

Cooper, D. W. and Kozdrowicki, E. W. (1973): *COKO user manual.* Privately circulated.

Cooper, R. and Elithorn, A. (1973): The organization of search procedures. Elithorn A. and Jones (eds.), *Artificial and Human Thinking*, pp 199-213. Elsevier.

Coplan, K. (1982): A Special-Purpose Machine for an Improved Search Algorithm for Deep Chess Combinations. *Advances in Computer Chess 3*, Clarke M. R. B. (ed.), pp 25-44

Coriat, I. H. (1941): The Unconscious Motives of Interest in Chess. *Psychoanalytic Review*, vol 28, pp 30-36

Courtois, G. (1975): *Etaoin Shrdlu Documentation.* Privately circulated from University of Boulder, Colorado.

Davies, D. W. (19??): A Theory of Chess and Noughts and Crosses. *Science News*, no 16.

Davis, R. (1976): Applications of meta level knowledge to the construction, maintenance and use of large knowledge bases. *AIM-283*, Computer Science Department, Stanford University.

De Groot, A. D. (1946): *Het Denken van den Schaker.*

De Groot, A. D. (1964): Chess Playing Programs. *Proc. Kononkl Nederlands Akod. Wentensch,*, Amsterdam, ser A-67, pp 385-398

De Groot, A. D. (1965): *Thought and Choice in Chess.* Mouton, the Hague, 1965. English translation, with additions, of the Dutch edition (1946). *** (excerpt)

De Groot, A. D. (1966): Perception and Memory versus Thought: Some old ideas and recent findings. Kleinmuntz, B. (ed.), *Problem Solving.* John Wiley, New York.

Devlin, J. C. (1970): Chess Computer Loses in King-size blunder. *New York Times*, 2 Sep., p 31

Djakow, I. N., Petrowski, N. W. and Rudik, P. A. (1927): *Psychologie des Schachspiels.* Walter de Gruyter, Berlin.

Donskoy, M. V. (1974): O programme, igrayushchey v shahmaty. *Problemy Kybernet., vol 29,* pp 169-200

Duncker, K. (1935): On Problem Solving (translated by L. S. Lees from the 1935 original). *Psychological Monographs*, vol 58, no 270.

Dunnings, C. A., Ko, H. M. and Banerji, R. B. (1969): Some results on Graph Interpretable Games. *Tech. Report, Case Western Reserve University*, Cleveland.

Edwards, D. J. and Hart, T. P. (1963): The Alpha-Beta Heuristic. *MIT AI Memo, 30.*

Eisenstadt, M. and Kareev, Y. (1973): Towards a Model of Human *Game Playing. Proc. 3rd IJCAI*, pp 458-463

Elithorn, A. and Telford, A. (1969): Computer Analysis of Intellectual Skills. *Int. Journ. Man-Machine Studies*,vol 1, pp 189-209

Elithorn, A. and Telford, A. (1970): Game and Problem Structure in Relation to the Study of Human and Artificial Intelligence. *Nature*, vol 227, pp 1205-1210

Elithorn, A. and Jagoe, R. (1969): The computer analysis of human problem solving behaviour: The choice of problem. *Proc. NATO Symposium on the Computer Simulation of Human Behaviour*, Paris

Ellis, S. H. and Chase, W. G. (1971): Parallel processing in item recognition. *Perception and Psychophysics*, vol 10, pp 379-84

Euwe, M. (1970): Computers and chess. *The Encyclopaedia of Chess*, Sunnucks P. A. (ed.), pp 78-88. St Martins Press.

Euwe, M. (1940-1941): *Het Eindspel, deel 1 t/m 12. 's-Gravenhage*-Batavia, G. B. van Goor zonen's Uitgeversmaatschappij

Euwe, M. and Donner, J. H. (1977): *Het Eindspel 1, Het mat zetten van de koning alleen, Stukken tegen stukken, Het pionneneindspel.* Uitgeverij Het Spectrum, Utrecht/Antwerpen.

Ewart, B. (1980): *Chess: Man vs. Machine.* A. S. Barnes.

Fahlman, S. E. (1974): A planning system for robot construction tasks. *Artificial Intelligence,* vol 5, pp 1-49

Feigenbaum, E. A. (1961): The simulation of verbal learning behaviour. *Proc. WJCC*, pp 121-132 (reprinted in Feigenbaum E. A. and Feldman J. (eds.), Computers and Thought. McGraw-Hill, New York.

Fenner, C. J. (1979): Computer Chess, News about the North American Computer Chess Championship. *The British Chess Magazine*, vol 99, no 5, pp 193-200

Findler, N. V. (1973): Computer Experiments on the Formation and Optimization of Heuristic Rules. Elithorn, A. and Jones (ed.), *Artificial and Human Thinking*, pp 177-188. Elsevier.

Fine, R. (1941): *Basic Chess Endings.*

Fine, R. (1942): *Chess the Easy Way.* David McKay, New York.

Fine, R. (1952): *The Middle Game in Chess.* David McKay, New York.

Frey, P. W. (editor) (1977): *Chess Skill in Man and Machine.* [2nd edition, with additions, 1983]. Springer-Verlag, New York.

Frey, P. W. (1977): An Introduction to Computer Chess.*Chess Skill in man and Machine*, Frey P. W. (ed.), pp 54-81

Frey P. W. (1977): Chess 4.5: Competition in 1976. *Chess Skill in man and Machine*, Frey P. W. (ed.), pp 243-247

Frey, P. W. (1983): Chess 4.5 and Chess 4.6: Competition in 1977 and 1978. *Chess Skill in Man and Machine*, 2nd edition, Frey P. W. (ed.), pp 248-256

Frey, P. W. and Atkin, L. R. (1978): Creating a Chess Player. *Byte*, Oct. 1978, pp 182-191, Nov. 1978, pp 162-181, Dec. 1978, pp 140-157, Jan 1979, pp 126-145, (Reprinted in Levy, D. N. L., *Computer Games I*, Springer-Verlag, New York).

Fuller, S. H., Gaschnig, J. G. and Gillogly, J. J. (1973): *Analysis of the Alpha-Beta Pruning Algorithm.* Computer Science Dept., Carnegie-Mellon University.

Garst, B. (1976): *A report on the validation of AQVAL's King-can-take-Pawn expressions.* Project for CS397, University of Illinois, Urbana.

Gillogly, J. J. (1970): *Max: A Fortran Chess Player.* P-4428, Rand Corporation.

Gillogly, J. J. (1972): The Technology Chess Program. *Artificial Intelligence*, vol 3, pp 145-163 ***

Gillogly, J. J. (1973): Reader commentary on the Cichelli heuristics. *SIGART* 43, Dec 1973, pp 27-28

Gillogly, J. J. (1977): Games. *Encyclopaedia of Computer Science and Technology*, vol 8, pp 392-408

Gillogly, J. J. (1978): Performance Analysis of the Technology Chess program. *Tech. Rept. CMU-CS-78-189*, Computer Science Dept., Carnegie-Mellon University.

Good (1950a): *Probability and Weighting of Evidence.* Griffin, London.

Good (1950b): *Contribution to discussion on Slater (1950).* ***

Good (1962): How Rational Should a Manager Be? *Management Science*, vol. 8, pp 383-393. Reprinted in *Executive Readings in Management Science*, Starr, M. K. (ed.), Macmillan, New York 1965.

Good (1964): Measurement of Decisions. Cooper, W. W., Leavitt, J. and Shelly, M. W. (eds.) *New Perspectives in Organisation Research*, pp 391-404. Wiley, New York.

Good, I. J. (1968): A Five Year Plan for Automatic Chess. Dale, E. and Michie, D. (eds.), *Machine Intelligence* 2, pp 89-118. Oliver and Boyd. *** (excerpt)

Good, I. J. (1969): Analysis of the Machine Chess Game J. Scott (White), ICL-1900 versus R. D. Greenblatt, PDP-10. Meltzer, B. and Michie, D. (eds.), *Machine Intelligence 4*, pp 267-269. Edinburgh University Press.

Good, I. J. (1976): Dynamic probability, computer chess, and the measurement of knowledge. *Firbush News* 6, MIRU, Edinburgh University, pp 43-62

Greenblatt, R. D., Eastlake, D. E. III and Crocker, S. D. (1967): The Greenblatt Chess Program. *Proc. FJCC*, vol 31, pp 801-810 ***

Griffith, A. K. (1976): Empirical Exploration of the Performance of the Alpha-Beta Tree Search Heuristic. *IEEE Trans. on Computers*, Jan. 1976, pp 6-10

Hadiev, R. E. (1975): Experimental studies of human decision-making and ... (on the basis of a chess endgame). *Proc. 4th IJCAI*, Tbilisi, pp 912-916

Haldane, J. B. S. (1952): The Mechanical Chess Player. *British Journal of Philosophy of Science*, vol 3, pp 189-191

Hansen, R. (1977): *TREEFROG: A chess program. Master of Math. essay*, Computer Science Dept., University of Waterloo.

Harding, T. D. (1982): *The Chess Computer Book*. Pergamon, Oxford.

Hardy and Wright (1938): *The Theory of Numbers*. OUP, Oxford.

Harkness, K. and Battell, J. S.(1947): This made chess history. *Chess Review*, Feb-Nov 1947

Harris, L. R. (1972): A Model for Adaptive Problem Solving Applied to Natural Language Acquisition.*Technical Report TR72-133*, Cornell University.

Harris, L. R. (1973): The Bandwidth Heuristic Search. *Proc. 3rd IJCAI*, Stanford, pp 23-29

Harris, L. R. (1974): The Heuristic Search Under Conditions of Error. *Artificial Intelligence*, vol 5, no 3, pp 217-239

Harris, L. R. (1975): The Heuristic Search and the Game of Chess: A study of Quiescence, Sacrifices, and Plan Oriented Play. *Proc. 4th IJCAI*, Tbilisi, pp 334-339 ***

Harris, L. R. (1977): The Heuristic Search: An Alternative to the Alpha-Beta Minimax Procedure. Frey P. W. (ed.), *Chess Skill in Man and Machine*, pp 157-166

Hayes, J. E. and Levy, D. N. L. (1976): *The World Computer Chess Championship*. Edinburgh University Press.

Hearst, E. (1977): Man and Machine: Chess Achievements and Chess Thinking. Frey P. W. (ed.), *Chess Skill in Man and Machine*, pp 167-200

Horning, J. J. (1972): *Outline of a Strategy For a Chess Playing Program*. Unpublished, University of Toronto, Dept. of Computer Science.

Horowitz, I. A. and Mott-Smith, G. (1973): *Point count chess*. Allen & Unwin.

Huberman, B. J. (1968): *A Program to Play Chess End Games*. Ph.D. thesis, published as Stanford Technical Memo CS 106, Computer Science Department.

Hyatt, R. M. (1977): *BLITZ V, A computer chess program*. Privately circulated.

Hyatt, R. M. (1985): M.Sc. thesis.

Hyatt, R. M., Gower, A. and Nelson, H. (1985): CRAY BLITZ. Beal, D. F. (ed.), *Advances in Computer Chess 4*, Pergamon, Oxford. ***

Jackson, P. C. (1974): *Introduction to Artificial Intelligence.* Petrocelli/Charter.

Jennings, P. R. (1978): The second world computer chess championship. *BYTE*, Jan. 1978, pp 108-118

Jongman, R. W. (1968): *Het Oog van de Meester.* Doctoral dissertation, University of Amsterdam. Assen: Van Gorcum and Company.

Jones, M. (1978): *Parallel search of chess game trees.* M.Sc. Thesis, School of Computer Science, McGill University.

Jones, R. D. (1971): *The design and implementation of a chess playing program for the IBM 360.* M.Sc. Thesis, University of Toronto

Kaindl H. (1982): Positional Long-Range Planning in Computer Chess. *Advances in Computer Chess 3*, Clarke M. R. B. (ed.), pp 145-168

Kalme, C. I. (1974): Teaching Chess at the Human and Machine Level. *Indiana Math. Psychology Progress Report.*

Kalme, C. I. (1974): The basic search routine for the move in chess. *Indiana Mathematical Psychology Program report.*

Kent, P. and Birmingham, J. A. (1977): Review of Chess and Computers by Levy. *Computer Bulletin*, no 11, Mar. 77, p 32

Kent, P. (1973): A Simple Working Model. Bell, A. G. (ed.), *Computer Chess*, pp 15-27

Kent P. and Birmingham J. A. (1980): The MASTER Chess Program. *Advances in Computer Chess 2*, Clarke M. R. B. (ed.), pp 131-142

Kernighan, B. W. and Richie, D. M. (1978): *The C Programming Language.* Bell Laboratories, Murray Hill, New Jersey. Prentice Hall.

King, P. F. (1971): A Computer Program for Positional Games. *Report 1107*, Jennings Computer Center, Case Western Reserve University.

Kister, J., Stein, P., Ulam, S., Walden, W. and Wells, M. (1957): Experiments in Chess. *Journal of the ACM*, vol 4, pp 174-177

Kitov, A. I. and Krinitsky, N. A. (1962): The solution of chess problems: Programme controlled computers playing chess. *Electronic Computers*, pp 106-108. Pergamon Press, Oxford.

Kmoch, H. (1969): *Pawn Power in Chess.* David McKay, New York.

Knuth, D. E. and Moore, R. (1975): An analysis of alpha-beta pruning. *Artificial Intelligence*, vol 6, Sept 1975, pp 293-326

Koffman, E. B. (1967): Learning Through Pattern Recognition Applied to a Class of Games. *System Research Center Report SRC 107-1-67-45*, Case IT.

Komissarchik, E. and Futer, A. (1974): Ob analize ferzevogo endshpilia pri pomoshchi EVM. *Problemy Kybernet*, vol 29, pp 211-220

Kopec, D. (1977): Recent developments in computer chess. *Firbush News* 7, MIRU, May 1977, Edinburgh University, pp 28-36

Kopec, D. and Bratko, I. (1982): The Bratko-Kopec Experiment: A Comparison of Human and computer Performance in Chess. Clarke M. R. B. (ed.), *Advances in Computer Chess 3*, pp 57-72

Kopec, D. and Niblett, T. (1980): How hard is the Play of the King-Rook-King-Knight ending? Clarke M. R. B. (ed.), *Advances in Computer Chess 2*, pp 57-81

Kotok, A. (1962): *A Chess Playing Program for the IBM 7090.* B.S. Thesis, MIT. Also published as *AI Project memo 41*, MIT Computation Center, 1962 ***

Kotov, A. (1971): *Think Like a Grandmaster.* Batsford, London.

Kozdrowicki, E. W. (1967): *An Adaptive Tree Pruning System: A Language for Programming Heuristic Tree searches.* Defense Documentation Center, No AD663750.

Kozdrowicki, E. W. (1968): A Practical Application of Machine Learning: Use of Learning in an Interpreter for a Tree Searching Language. *Proc. IEEE Systems Science and Cybernetics Conf.*, San Francisco.

Kozdrowicki, E. W. (1968): An Adaptive Tree Pruning System: A Langauge for Programming Heuristic Tree Searches. *Proc. ACM68*, pp 725-735

Kozdrowicki, E. W. (1969): The complexity of chess algorithms: The limitations of machines when compared with human thought. *Proc. 3rd Princeton Conf. on Information Science and Systems*, pp 92-96

Kozdrowicki, E. W. and Cooper, D. W. (1972): COKO III: The Cooper-Koz Chess program. *ACM72 Computer chess notes*, Boston 1972, pp 25-32

Kozdrowicki, E. W. and Cooper, D. W. (1973): COKO III: The Cooper-Koz Chess program. *Communications of the ACM*, vol 16, no 7, July 1973, pp 411-427

Kozdrowicki, E. W. and Cooper, D. W. (1973): COKO III and the future of inter-snap judgement communication. *Proc. ACM73*, Atlanta, pp 213-218

Kozdrowicki , E. W. and Cooper, D. W. (1974): COKO III: The Cooper-Kozdrowicki Chess Program. *Int. Journ. of Man-machine studies*, no 6, pp 627-699

Kozdrowicki, E. W. and Cooper, D. W. (1974): When will a Computer be World Chess Champion? *Computer Decisions*, August 1974, pp 28-32

Kozdrowicki, E. W., Licwinko, J. S. and Cooper, D. W. (1971): Algorithms for a minimal Chess player: A Blitz Player. *Int. Journ. Man-machine Studies*, vol 3, pp 141-165

Levy, D. N. L. (1969): Computerised Chess: Prospects. *Chess*, April 22, pp 242-251

Levy, D. N. L. (1969): *A measure of the performance of evaluation functions.* Privately circulated.

Levy, D. N. L. (1971) Computer Chess – A Case Study on the CDC6600. Meltzer, B. and Michie, D. (eds.), *Machine Intelligence* 6, pp 151-163. Edinburgh University Press.

Levy, D. N. L. (1973): Computer Chess – Past, Present and Future. *Chess Life and Review*, vol 28, Dec 1973, pp 723-726

Levy, D. N. L. (1976): *1975 U.S. Computer Chess Championship.* Computer Science Press.

Levy, D. N. L. (1976): The Robots are Coming. Or Are They? *Chess Life and Review*, May 1976, pp 250-260

Levy, D. N. L. (1976): *Chess and Computers.* Batsford, London.

Levy, D. N. L. (1977): *1976 U.S. Computer Chess Championship.* Computer Science Press

Levy, D. N. L. (1978): Can Computer Programs Play Real Chess? *Proc. 3rd Jerusalem Conf. on information Technology.*

Levy, D. N. L. (1978): Man Beats Machine. *Chess Life and Review*, vol 33, no 11, November 1978, pp 600-603

Levy, D. N. L. (1983): *The Computer Chess Handbook.* Batsford, London. Also published under the title *The Joy of Computer Chess*, Prentice-Hall.

Levy, D. N. L. (1984): Chess Master v Computer. *ICCA Journ.*, vol 7, no 2, pp 106-117. Also in Beal, D. F. (editor) *Advances in Computer Chess 4*, Pergamon, Oxford.

Levy, D. N. L. and Newborn, M. M. (1980): *More About Chess and Computers.* Computer Science Press.

Levy, D. N. L. and Newborn, M. M. (1982): *All About Chess and Computers.* Computer Science Press.

London, R. L. and Salako, A. (19??): *Game playing by computer: a selected bibliography.* Computer Science Dept., University of Wisconsin.

Luce, D. and Raiffa, H. (1957): *Games and Decisions.* John Wiley and Sons, New York

Maier, N. R. F. (1960): Screening Solutions to Upgrade Quality: A new Approach to Problem Solving under Conditions of Uncertainty. *Journal of Psychology*, vol 49, pp 217-231

Malik, R. (1973): Observations. Bell A. G. (ed.), *Computer Chess*, pp 89-94

Manning, J. R. (1971): White to move and mate in N moves. *Computer Journal*, vol 14, Algorithm 68, pp 209-213

Marino, L. R. (1966): Winning and Non-Losing Strategies in Games and Control. *Tech. Report SRC 91-A-66-36*, Case Western Reserve University, Cleveland.

Marsland, T. A. (1971): Review of "Computer chess – a case study on the CDC 6600" by Levy. *Computing Reviews*, No. 21859, Sept. 71, p 394

Marsland, T. A. (1972): *WITA, a selective search chess program in AlgolW.* Privately Circulated.

Marsland, T. A. (1974): Users guide to WITA, a chess playing program. *Technical report TR74-2*, Dept. of Computing Science, University of Alberta, Edmonton.

Marsland, T. A. (1977): 1976 Canadian computer chess workshop. *Firbush News* 7, MIRU, Edinburgh University, May 1977, pp 37-40

Marsland, T. A., Campbell, M. S. and Rivera, A. L. (1980): Parallel search of game trees. *Technical report TR80-7*, Department of Computing Science, University of Alberta, 1980

Marsland, T. A. and Popowich, F. (1983): A Multiprocessor Tree-Searching System Design. *Technical Report TR83-6*, Computing Science Dept., University of Alberta, Edmonton.

Marsland, T. A. and Rushton, P. G. (1973): Mechanisms for Comparing Chess programs. *Proc. ACM73*, pp 202-205

Marsland, T. A. and Rushton, P. G. (1973): A study of techniques for game-playing programs. *Journ. of Institute for Computer Science*, vol 4, no 2, pp 26-30. Also in *Advances in Cybernetics & Systems*, vol 1, 1974, pp 363-371

Maurer, W. D. (1979): Alpha-Beta Pruning. *Byte*, no 11, pp 84-96

Maynard-Smith, J. and Michie, D. (1961): Machines that play games. *New Scientist*, Nov. 1961, pp 367-369

McCarthy, J. (1956) The Inversion of Functions Defined by Turing Machines. Shannon, C. E. and McCarthy, J. (eds.), *Automata Studies*, Princeton University Press, pp 177-181

McLean, R. S. and Gregg, L. W. (1967): Effects of induced chunking on temporal aspects of serial recitation. *Journal of Experimental Psychology* vol 74, no 4, pp 455-59

Messerschmidt, H. J. (1980): Parallel programming for a chess endgame database. *Software – Practice and Experience*, no 10, pp 475-487

Michalski, R. and Negri, P. (1977): An experiment on inductive learning in chess endgames: The King-Pawn case. Elcock, E. W. and Michie, D. (eds.). *Machine Intelligence 8, pp 175-192. Edinburgh University Press.*

Michie, D. (1966): Game-Playing and Game-Learning Automata. Fox (ed.),*Advances in Programmimg and Non-Numerical Computation*, pp 183-200. Pergamon, Oxford. [Includes Appendix: Rules of SOMAC by Maynard-Smith, J.]

Michie, D. (1972): Programmer's Gambit. *New Scientist*, vol 56, Aug 1972, pp 329-332.

Michie, D. (1973): Knowledge Engineering. Kybernetes, *International Journal of Cybernetics and General Systems*, vol 2, no 4, pp 197-200

Michie, D. (1974): On Machine Intelligence, pp 31-49, pp 135-142, pp 186-192. Edinburgh University Press.

Michie, D. (1976): An advice taking system for computer chess. *Computer Bulletin*, Dec. 1976, pp 12-14

Michie, D. (1976): AL1: A package for generating strategies from tables. *SIGART 59*, Aug 76, pp 12-14

Michie, D. (1976): A theory of evaluative comments in chess. *Report MIPR-105*, Machine Intelligence Lab., Edinburgh University. ***

Michie, D. (1976): Measuring the knowledge content of programs. Computer Science Dept., University of Illinois, Urbana, *Rept. UIUCDCS-R-76-786.*

Michie, D. (1977): King and Rook against King: Historical background and a Problem on the Infinite Board. *Advances in Computer Chess 1*, Clarke M. R. B. (ed.), pp 30-59

Michie, D. (1977): A Theory of Advice. Elcock, E. W. and Michie, D. (eds.), *Machine Intelligence* 8, pp 151-168. Edinburgh University Press.

Michie, D. (1977): End-game secrets. *Computer Weekly*, Chesslab, Nov. 3.

Michie, D. (1978): A prototype knowledge refinery. *Advances in Computer Chess Conference*, Edinburgh.

Michie, D. (1979): Computer Chess. *Interdisciplinary Science Reviews.*

Michie, D. (1982): Information and Comlexity in Chess.*Advances in Computer Chess 3*, Clarke, M. R. B. (ed.), pp 139-144

Miller, G. A. (1956): The magical number seven, plus or minus two. *Psychological Review*, vol 63, pp 81-97

Minker, J. (1975): A review of COKO III. *ACM Computing Reviews, no 28820*, vol 16, Sept 1975, pp 369-370

Mittman, B. (1973): Can a Computer Beat Bobby Fischer? *Datamation*, vol 19, June 1973, pp 84-87

Mittman, B. (1977): A Brief History of Computer Chess Tournaments: 1970-1975. Frey, P. W. (ed.), *Chess Skill in Man and Machine*, pp 1-33

Mittman, B. (1974): First World Computer Chess Championship at IFIP Congress, Stockholm, August 1974. *Communications of the ACM*, vol 17, Oct 1974, p 604

Moullen, F. L. (1968): Chess and the Computer. *Datamation*, vol 14, no 4, April, 1968, pp 65-68

Moussouris, J., Holloway, J. and Greenblatt, R. (1979): CHEOPS: A chess-oriented processing system. Hayes, J. E., Michie, D. and Mikulich, L. I. (eds.), *Machine Intelligence* 9, pp 351-360. Horwood-Wiley, Chichester. ***

Myron, M. S. and May, W. H. (1963): A Note on Serendipity, Aesthetics and Problem Solving. *Behavioural Science*, vol 8, pp 242-243

Negri, P. (1977): Inductive Learning in a Hierarchical Model for Representing Knowledge in Chess End Games. Elcock, E. W. and Michie, D. (eds.). *Machine Intelligence* 8, pp 193-204. Edinburgh University Press.

Neisser, U. (1963): The imitation of man by machine. *Science*, vol 139, pp 193-97

Nemes, T. N. (1949): Mechanical Solution of Diophantic Problems - Part I. *Publications of the University of Technical Sciences, Hungary*, vol 9, pp 29-50.

Nemes, T. N. (1951): Mechanical Solution of Diophantic Problems - Part II (The Chess Playing Machine). *Acta technica*, Hungarian Academy of Sciences, pp 215-239

Nemes, T. N. (1969): *Cybernetic Machines.* Iliffe Books, (translated by I. Foldes from 1962 Hungarian edition).

Newborn, M. M.(1972): A Review of the 1st and 2nd US Computer Championships and an analysis of Recent Developments in Computer Chess. *ACM72 Computer chess notes*, Boston 1972, pp 1-9

Newborn, M. M. (1975): *Computer Chess.* Academic Press, New York.

Newborn, M. M. (1977): PEASANT: An endgame program for Kings and Pawns. Frey, P. W. (ed.), *Chess Skill in Man and Machine*, pp 119-130

Newborn, M. M. (1977): The efficiency of the alpha-beta search on trees with branch-dependent terminal node scores. *Artificial Intelligence*, vol 8, pp 137-153

Newborn, M. M. (1978): Recent progress in computer chess. Yovits, M. (ed.), *Advances in Computers*, vol 18, pp 59-117. Academic Press. New York. (Reprinted in Levy, D. N. L. (editor), *Computer Games I*, Springer-Verlag, New York.)

Newborn, M. M. (1978): Computer chess: Recent progress and future expectations. *Proc. 3rd Jerusalem Conf. on Information Technology.*

Newborn, M. M. (1982): OSTRICH/P - A Parallel Search Chess program. School of Computer Science report 82.3, McGill University, Montreal.

Newborn, M. M., Ziv, A. and Neumann, A. (1974): *An Analysis of the Alpha-Beta Algorithm for Trees of Depth Two.* Privately circulated.

Newell, A. (1955): The Chess Machine: An example of Dealing with a Complex Task by Adaptation. *Proc. WJCC*, pp 101-108 ***

Newell, A. (1962): Some Problems of Basic Organization in Problem-Solving Programs. Yovitts, M., Jacobi, G. T. and Goldstein, G. D. (eds.), *Self-Organizing Systems*, pp 393-423. Spartan Books, New York.

Newell, A. (1966): *On the Representations of Problems.* CSRR Carnegie-Mellon University.

Newell, A. and Ernst, G. (1965): The Search for Generality. *Proceedings of IFIPS Congress 65*, vol 1, pp 17-24. Spartan Books, Washington, D.C.

Newell, A. and Prasad, N. S. (1963): IPL-V Chess Position Program. *Internal Memo no 63*, Computer Science Dept., Carnegie-Mellon University.

Newell, A. and Shaw, J. C. (1957): Programming the Logic Theory Machine. *Proc. of the 1957 WJCC.*

Newell, A., Shaw, J. C. and Simon, H. A. (1957): Empirical Explorations of the Logic Theory Machine. *Proc. of the 1957 WJCC.* Also in Fiegenbaum, E. A. and Feldman, J. (eds.), Computers and Thought, pp 109-133. McGraw Hill, New York.

Newell, A., Shaw, J. C. and Simon, H. A. (1958): Chess Playing Programs and the Problem of Complexity. *IBM Journal of Research and Development*, Oct. 1958, pp 320-355. Also in Fiegenbaum, E. A. and Feldman, J. (eds.), Computers and Thought, 1963, pp 39-70 ***

Newell, A., Shaw, J. C. and Simon, H. A. (1958): Elements of a Theory of Human Problem Solving. *Psych. Rev.*, vol 65, May 1958.

Newell, A., Shaw, J. C. and Simon, H. A. (1962): The process of Creative Thinking. Gruber, H. E., Terrell, G. and Wertheimer, M.(eds.), *Contemporary Approaches to Creative Thinking*, pp 63-119. Atherton Press, New York.

Newell, A. and Simon, H. A. (1956): The Logic Theory Machine. *Transactions on Information Theory, IT-2*, no 3, September 1956.

Newell, A. and Simon, H. A. (1961): Computer Simulation of Human Thinking. *Science*, vol 134, pp 2011-2017

Newell, A. and Simon, H. A. (1965): An Example of Human Chess Play in the Light of Chess Playing Programs. Weiner, N. and Schade, J. P. (eds.), *Progress in Biocybernetics*, vol 2, pp 19-75. Elsevier.

Newell, A. and Simon, H. A. (1972): *Human Problem Solving.* Prentice-Hall.

Newman, C. and Uhr, L. (1965): Bogart: A Discovery and Induction Program for Games. *Proc. ACM65*, pp 176-186.

Nievergelt, J. (1977): Information Content of Chess Positions – Information for Chess-Specific Knowledge of Chess Players. *SIGART* 62, April 1977, pp 13-15

Nievergelt, J. and Farrar, J. C. (1972): What Machines Can and Cannot Do. *Computing Surveys*, vol 4, June 1972, pp 81-96

Nilsson, N. J. (1971): Problem Solving Methods in Artificial Intelligence. McGraw-Hill, New York.

Nilsson, N. J. (1968): Searching problem solving and game-playing trees for minimal cost solutions. *Proc. IFIP Conference*, Edinburgh, pp 1556-1562. North-Holland Publishing Company.

Nitsche T. (1982): A Learning Chess Program. *Advances in Computer Chess 3*, Clarke, M. R. B. (ed.), pp 113-120

Noordzij, P. (1967): *Registratie van oogbewegingen bij schakers*. Unpublished working paper, Psychology Laboratory of the University of Amsterdam.

Noton, D. and Stark, L. (1971): Scan-paths in eye movements during pattern perception. *Science*, vol 171, pp 308-311

Orbaan, C. (1980): *Kombineren. Schaakbulletin*, no 154, pp 597-598

Parry, J. (1974): Computer Chess. *Chess Federation of Canada Bulletin*, Sep-Oct 1974, pp 11-13

Parry, J. (1974): *The Ribbit Papers*. Privately circulated from University of Waterloo.

Penrod, D. (1978-): *Computer Chess Newsletter*. [Penrod was the founding editor, Shershow, H. took over and for a while published computer chess information in Personal Computing.]

Penrose, J. (1965): The Psychology of Chess. *New Society*, vol 29, pp 967-968

Perdue, C. and Berliner, H. J. (1977): EG – A Case Study in Problem Solving with King and Pawn Endings. *Proc, 5th IJCAI*, pp 421-427

Piasetski, L. (1976): *An evaluation function for simple King and Pawn endings*. M.Sc. Thesis, McGill University, Montreal.

Pitrat, J. (1968): Realization of a General Game-Playing Program. *Proc. IFIP 68*, pp 1570-1574

Pitrat, J. (1971): A General Game-Playing program. Findler, N. V. and Meltzer, B. (eds.), *Artificial Intelligence and Heuristic Programming*. Edinburgh Universlty Press.

Pitrat, J. (1976): A program for learning to play chess. Chen, C. H. (ed.), *Pattern Recognition and Artificial Intelligence*, pp 399-419. Academic Press.

Pitrat, J. (1974): Realization of a program learning to find combinations in chess. *Computer orientated learning processes*, NATO Advanced Study Institute.

Pitrat, J. (1977): A Chess Combination program which uses plans. *Artificial Intelligence*, vol 8, no 3, June 1977, pp 275-321

Pitrat, J. (1977): Using Plans in a Chess Playing Program.*5th IJCAI*, pp 979-982

Pitrat, J. (1977): The Behaviour of a Chess Combination Program using Plans. *Advances in Computer Chess 1*, Clarke, M. R. B. (ed.), pp 110-121

Pohl, I. (1973): Avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving. *Proc. 3rd IJCAI*, Stanford, pp 12-17

Popowich, F. and Marsland, T. A. (1983): Parabelle: Experience With a Parallel Chess Program. *Technical Report TR83-7*, Computing Science Dept., University of Alberta, Edmonton.

Prinz, D. G. (1952): Robot Chess. *Research*, vol 5, 1952, pp 261-266 ***

Proskurowski, W. (1974): Ordering method to accelerate the solution of mate-in-two chess problems by computer. *Computer Journal*, vol 17, no 1, pp 80-81

Reinfield, F. (1958): *Win at Chess*, Dover Books

Reksten, H. (1978): *An annotated bibliography on computer chess.* Computer Science Dept., University of Delaware.

Reshevsky, S. (1972): Analysis Puts Fischer Ahead of IBM. *New York Times*, August 18.

Richards, P. I. (1952): On Game Learning Machines. *Scientific Monthly*, vol 74, April 1952, pp 201-205

Richter, H. (1976): *A chess program based on principles of human thought processes on playing chess.* Tech. Rept., Institut fuer Informatik, Hamburg.

Rollason, J. (1977): *MERLIN a chessplaying program.* Privately circulated, Westfield College, London University.

Ruben, I., Swartz, F., Winograd, J., Berman, V. and Toikka, W. (1973): *Chaos.* Privately circulated from Sperry Univac, Cinaminson, NJ.

Rushton, P. G. and Marsland, T. A. (1973): Current Chess Programs: A summary of their potential and limitations. *INFOR*, vol 11, no 1, February 1973, pp 13-20

Rushton, P. (1972): *A Critique of Programming Techniques for Playing Chess.* M.Sc. Thesis, University of Alberta, Edmonton.

Sacerdoti, E. D. (1974): Planning in a hierarchy of abstraction spaces. *Artificial Intelligence* vol 5, pp 115-135

Sacerdoti, E. D. (1975): *The nonlinear nature of plans.* Stanford Research Institute Technical Note 101.

Saliba, M. A. (1976): *Some Aspects of the Automatic Simulation of Human Behaviour in the Choice of a Strategy: The Case of a Chess Algorithm.* Doctor of Mathematics thesis, Universita degli Studi di Pisa.

Samson, P. (1965): MIDAS. *Artificial Intelligence Project Memo 90*, MIT.

Samuel, A. L. (1959): Some Studies in Machine Learning Using the Game of Checkers. *IBM Journ. of Research and Development*, vol 3, July 1959, pp 210-229. Also in Feigenbaum, E. and Feldman, J. (eds.), *Computers and Thought*, pp 71-105. McGraw-Hill.

Samuel, A. L. (1960): Programming Computers to Play Games. *Advances in Computers 1*, pp 165-192.

Samuel, A. L. (1967): Some Studies in Machine Learning using the Game of Checkers II – Recent Progress. *IBM Journ. of Research and Development*, vol 11, pp 601-617

Schaeffer, J. (1983): The History Heuristic. *ICCA Journ.*, vol 6, no 3, pp 16-19.

Schaeffer, J., Olafsson, M. and Marsland, T. A. (1984): Experiments in Distributed Tree-Search. *Technical Report TR 84-4*, Computing Science Dept., University of Alberta, Edmonton.

Scott, J. J. (1969): A Chess-Playing Program. Meltzer B. and Michie D. (eds.), *Machine Intelligence* 4, pp 255-265. Edinburgh University Press.

Scott, J. J. (1969): Lancaster vs. MAC HACK. *SIGART*, vol 16, pp 9-11

Scurrah, M. J. and Wagner, D. A. (1970): Cognitive Model of Problem-Solving in Chess. *Science*, vol 169, July 1970, pp 209-211

Selfridge, O. (1965): Reasoning in Game Playing by Machine. Sass, M. A. and Wilkinson, W. D. (eds.), *Symposium on Computer Augmentation of Human Reasoning.* Spartan Books, Washington D.C.

Shannon, C. E. (1950a): A Chess-Playing Machine. *Scientific American*, vol 182, Feb 1950, pp 48-51

Shannon, C. E. (1950b): Programmimg a Computer for Playing Chess. *Philosophical Mag.*, vol 41, pp 256-275 ***

Shapiro, A. and Niblett, T. (1982): Automatic Induction of Classification Rules for a Chess Endgame. *Advances in Computer Chess 3*, Clarke, M. R. B. (ed.), pp 73-92

Shaw, J. C., et al. (1958): A Command Structure for Complex Information Processing. *Proc. WJCC.*

Simon, H. A. (1969): *The Sciences of the Artificial*, pp 35-38. MIT Press, Cambridge, Massachusets.

Simon, H. A. (1974): How big is a chunk?. *Science*, vol 183, pp 482-488

Simon, H. A. and Barenfield, M. (1969): Information-Processing Analysis of Perceptual Processes in Problem Solving. *Psych. Review*, vol 76, pp 473-483

Simon, H. A. and Chase, W. G. (1972): The Mind's Eye in Chess. Visual Information Processing, ed. Chase, W. G. *Proc. 8th Annual Carnegie Psychology Symposium*, (1972) Academic Press, N.Y., pp 215-281

Simon, H. A. and Chase, W. G. (1973): Skill in Chess. *American Scientist*, vol 61, no 4, pp 394-403 ***

Simon, H. A. and Chase, W. G. (1973): Perception in Chess. *Cognitive Psychology*, vol 4, pp 55-81

Simon, H. A. and Feigenbaum, E. A. (1964): On information and meaningfulness in verbal learning. *Journal of Verbal Learning and Verbal Behavior*, vol 3, pp 385-396

Simon, H. A. and Gilmartin, K. (1974): A Simulation of Memory for Chess Positions. *Cognitive Psychology*, vol 5, pp 29-46

Simon, H. A. and Newell, A. (1956): Models: Their Uses and Limitations. White, L. D. (ed.), *The State of the Social Sciences*, pp 66-83. University of Chicago Press.

Simon, H. A. and Newell, A. (1964): Information processing in computer and man. *American Scientist*, vol 52, pp 281-300

Simon, H. A. and Simon, P. A. (1962): Trial and Error Search in Solving Difficult Problems: Evidence From the Game of Chess. *Behavioural Science*, vol 7, no 4, pp 425-429

Slagle, J. R. (1963): Game Trees, M & N Minimaxing, and the M & N alpha-beta procedure. *Artificial Intelligence Group Report 3*, University of California.

Slagle, J. R. (1971): *Artificial Intelligence: The Heuristic Programming Approach.* McGraw-Hill, New York.

Slagle, J. R. and Bursky, P. (1968): Experiments with a Multi-purpose Theorem-proving Heuristic Program. *Journal of the ACM*, vol 15, no 1, Jan 1968, pp 85-99.

Slagle, J. R. and Dixon, J. K. (1969): Experiments with Some Programs which Search Game Trees. *Journ. of the ACM*, vol 16, no 2 April 1969, pp 189-207

Slagle, J. R. and Dixon, J. K. (1970): Experiments with the M & N Tree Searching Program. *Communications of the ACM*, vol 13, no 3, pp 147-154 and 159

Slate, D. J. and Atkin, L. R. (1969): CDC *Program File Printout from University of London Computer Centre*

Slate, D. J. and Mittman, B. (1978): CHESS 4.6 – where do we go from here. *Proc. 3rd Jerusalem Conference on Information Technology.*

Slate, D. J., Atkin, L. and Gorlen, K. (1972): Chess 3.5. *ACM72 Computer chess notes*, Boston 1972, pp 15-21

Salte, D. J. (1977): Chess 4.5 *SIGART 63*, June 1977, pp 84-85

Slater, E. (1950): Statistics for the Chess Computer and the Factor of Mobility. *Symposium on Information Theory*, pp 150-152. Ministry of Supply, London. ***

Smith, R. C. and Ceruti, F. D. (1972): SCHACH. *ACM72 Computer chess notes*, Boston 1972, pp 37-40

Smith, R. C. (1969): The Schach Chess Program. *SIGART*, vol 15, pp 8-12

Soule, S. and Marsland, T. A. (1975): Canadian computer-chess tournament. *SIGART 54*, Oct 75, pp 12-13

Spracklen, D. and Spracklen, K. (1978): *SARGON: A Computer chess program*. Hayden Book Co., Rochelle Park, NJ.

Spracklen, D. and Spracklen, K. (1978): An exchange evaluator for computer chess. *Byte*, Nov. 1978, pp 16-28

Spracklen, K. and Spracklen, D. (1978): First steps in computer chess programming. *Byte*, Oct. 1978, pp 86-98

Sternberg, S. (1969): Memory-scanning: Mental processes revealed by reaction-time experiments. *American Scientist*, vol 57, pp 421-57

Strachey, C. S. (1952): Logical or Non-Mathematical Programmes. *Proc. ACM Conference in Toronto*, pp 46-49

Strohlein, T. (1970): *Untersuchungen ueber kombinatorische Spiele*. Dissertation, Fakultaet fuer allgemeine Wissenschaften der Technischen Hochschule Muenchen.

Sugar, L. (1975): An experimental evaluation of chess playing heuristics. *Technical Report CSRG-63*, University of Toronto.

Swets, B. (1978): *Second World Computer Chess Championship*. Privately circulated translation of *Wereldkampionschap Computerschacken* de ingenier, no 8, 23 Feb 78, pp 167-173

Swets, B. and Koppelaar, H. (1978): De Struktuur van Schaakprogramma's. *Informatie*, vol 20, no 6, pp 334-343

Tan, S. T. (1972): *Representation of Knowledge for Very Simple Pawn Endings in Chess*. Ph.D. thesis, Department of Machine Intelligence, Edinburgh University Published as MIP-R-98.

Tan, S. T. (1973): A Knowledge Based Program to play Chess Endgames. Bell, A. G. (ed.), *Computer Chess*, pp 81-88

Tan, S. T. (1974): Kings, Pawn and Bishop. *Technical Report MIP-R-108*, Department of Machine Intelligence, Edinburgh University.

Tan, S. T. (1976): The winning program. *Firbush News* 5 (chess supplement), MIRU, pp 38-45, Edinburgh University.

Tan, S. T. (1977): Describing Pawn structures. *Advances in Conputer Chess 1*, Clarke, M. R. B. (ed.), pp 74-88

Tarrasch, Siegbert (1976): *The Game of Chess*. David McKay, New York

Taube, M. (1961): *Computers and Common Sense, the Myth of Thinking Machines*. Columbia University Press.

Thompson, K. (1973): Tinker Belle, a 'C' language chess program under UNIX. Privately circulated.

Thompson, K. (1982): Computer Chess Strength. *Advances in Computer Chess 3*, Clarke, M. R. B. (ed.), pp 55-56

Turing, A. M. (1950): Computing Machinery and Intelligence. *Mind*, vol 59, pp 433-460. Reprinted in Newman, J. R. (ed.), *The World of Mathematics*, vol 4, no 19, pp 2099-2123; and in Fiegenbaum, E. A. and Feldman, J. (eds.), *Computers and Thought*, pp 11-35; and in Anderson, A. R. (ed.), *Minds and Machines*.

Turing, A. M. (1953): Chess. Digital Computers Applied to Games. Bowden (ed.), *Faster than Thought*, pp 286-310 ***

Valenti, F. M. and Vranesic, Z. G. (1977): Experiences with CHUTE. *Proc. ACM77*, Seattle, pp 474-478

Valenti, F. M. (1974): *CHUTEI, An Easily Modifiable Chess Playing Program.* M.Sc. Thesis, University of Toronto.

Van den Herik, H. J. (1980a): *Computerschaak, Direkte evaluatie door middel van patroonherkenning in het KNPK-eindspel.* Technische Hogeschool Delft.

Van den Herik, H. J. (1980b): Goal-directed Search in Chess End games. *Delft Progress Report* no 5, pp 253-279. Delft University of Technology. ***

Vigneron, H. (1914): Robots. Levy, D. N. L., *Chess and Computers*, pp 14-23 (translation of "Les Automates", La Nature, 1914, pp 56-61) ***

Von Neuman, J. and Morgenstern, O. (1947): Application to Chess. *Theory of Games and Economic Behaviour*, Princeton University Press.

Wagner, D. A. and Scurrah, M. J. (1971): Some characteristics of human problem-solving in chess. *Cognitive Psychology*, vol 2, pp 454-478

Wason, P. (1972): The Psychology of Chess. *New Scientist*, July 20 1972, pp 134-137

Waterman, D. (1970): Generalization Learning Techniques for Automating the Learning of Heuristics. *Artificial Intelligence*, vol 1, pp 121-170

Weinberg, M. (1951): Mechanism in neurosis. *American Scientist*, vol 39, January 1951, pp 74-98

Wetherell, C. S., Buckholtz, T. J. and Booth, K. S. (1972): A Director for Kriegspiel: A Variant of Chess. *Computer Journal*, vol 15, no 1, Feb 1972, pp 66-70

Whaland, N. D. (1978): A Computer Chess Tutorial. *Byte*, October 1978, pp 168-183. ***

Wiener, N. (1948): *Cybernetics.* John Wiley, New York.

Wilkins, D. E. (1979): Using patterns and plans to solve problems and control search. *AIM-329*, Computer Science Dept., Stanford University.

Wilkins, D. E. (1980): Casuality analysis in chess. *Proceedings of 3rd Conference of Canadian Society for the Computational Study of Intelligence*, Victoria.

Wilkins, D. E. (1980): Using patterns and plans in chess. *Artificial Intelligence*, vol 14, pp 165-203 ***

Wilkins, D. E. (1982): Using knowledge to control tree searching. *Artificial Intelligence*, vol 18, pp 1-51

Wilkins, D. E. (1983): Using Chess Knowledge to reduce Search. Frey, P. W. (ed.), *Chess Skill in Man and Machine*, 2nd edition, pp 211-242

Williams, L. G. (1966): The effect of target specification on objects fixated during visual search. *Perception and Psychophysics* vol 1, pp 315-18

Williams, T. G. (1965): *Some Studies in Game Playing with a Digital Computer.* Thesis, Carnegie-Mellon University.

Wolf, G. (1973): Implementation of a dynamic tree searching algorithm in a chess program. *Proc. ACM73*, Atlanta, pp 206-208

Zermelo, E. (1912): An application of set theory to the theory of chess-playing. English translation of (German) paper at 5th Int. Congress of Mathematics, Cambridge, England. In *Firbush News* 6, July 1976, pp 37-42

Zielinski, G. (1975): *Heuristics for computer chess.* Applied Mathematics and Physics Dept., Warsaw Tech. University.

Zielinski, G. (1976): *Arrays for Programming Chess.* Kybernetes, vol 5, no 2, pp 91-96

Zobrist, A. L. and Carlson, F. R. (1973): An Advice-Taking Chess Computer. *Scientific American*, vol 228, no 6, June 1973, pp 92-105

Zobrist, A. L. (1969): A Model of Visual Organisation for the Game of Go. *Proc. SJCC 69*, pp 103-112

Zobrist, A. L. (1969): A Hashing Method with Applications for Game Playing. *Technical Report 88*, Computer Sciences Dept., University of Wisconsin.

Zobrist, A. L. and Carlson, F. R. (1973): The USC chess program. *Proc. ACM73*, Atlanta, pp 209-212

Zuidema, C. (1974): Chess, How to Program the Exceptions? *Afdeling Informatica, IW21/74*, Math. Centrum, Amsterdam, 1974

Zuse, K. (1976): Chess Programs. *Rept. no 106, Gesellschaft fuer Mathematik und Datenverarbeitung*, Bonn, 1976, pp 201-204 (translation of "Plankalkuel", 1945)