THE GAME OF PENTOMINOES

by

MICHAEL KUTTNER

B.Sc., University College London, 1966

A thesis submitted in partial fulfilment of

the requirements for the degree of

MASTER OF SCIENCE

in the Department

of

COMPUTER SCIENCE

We accept this thesis as conforming to

the required standard.

THE UNIVERSITY OF BRITISH COLUMBIA

December, 1972

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the Head of my Department or by his representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of COMPUTER SCIENCE

The University of British Columbia
Vancouver 8, Canada

Date 29/12/72

ABSTRACT

A study in game-playing programming is made using the game of pentominoes which has a very large branching factor and where there exists almost no precise, factual information to guide the conduct of the play.

The difficulties encountered imply that some apparent advantages of heuristic techniques are more heavily problem-dependent than is usually conceded.

A guiding device capable of learning is incorporated which significantly improves the program's play in competition with versions lacking it and shows subjective improvement with human competition.

TABLE OF CONTENTS

ACKNOWLEDGMENT

I wish to express my thanks to Dr. Richard Rosenberg for his patient supervision during the preparation of this thesis, to the National Research Council of Canada part of this work being carried out under grant A-5552 and to Sheila Taylor for enduring typing from my handwriting.

INTRODUCTION

The traditional reason, employed by people working in heuristic problem solving, for choosing game playing as a vehicle for their experimental research is that situations of great difficulty arise but there are very few formal parameters involved. It is known that there are no psychological or social factors influencing their models, merely a few simply defined rules.

However, it turns out with most games much played by humans that there is a body of knowledge that provides guidance for players far beyond the given rules and the complete search space. The knowledge, specific to some games, that guides players also guides programmers and may make general heuristic techniques less likely to evolve.

A game has been chosen - pentominoes - where very little is known, indeed. Furthermore, since it involves spatial judgment and combinatorial considerations, it is by no means completely straightforward to write a program for it.

Initially, thought centred on finding an internal representation which might be useful for manipulation in a manner analogous to a human approach to this game. As will be seen there had been one program described before but the representation had to be much modified; like most early polyomino programs it was concerned only with counting or dissection of rectangles into sets of pentominoes.

The chosen representation had effects on how the program would "visualise" the state of the board. It seemed pointless to redirect it, place it under the same constraints as a human, since its model of board

state produced its own advantages. We might waste time dealing with an area of the board which seemed to be open and yet accommodated no pieces. The program however would dismiss this area immediately. Fortunately or unfortunately, these differences would lead the program development towards a situation where different strategies should be deployed against a human or a machine opponent. This highlights the curious situation (foretold by Good [11] for a different reason) in which machines play chess only against other machines; viz. the annual ACM tournament.

The first step in building and testing this game-playing program was to develop the basic code for manipulating the board and pieces and to include a random move generator to enable moves to be made, games to be played and statistics and other information to be gathered. These routines revealed for the first time the total size of the move tree and the number of choices at each ply.

Since a position with 100 choices for a move is quite close to the end of the game one should hope to determine the consequences of all moves fairly thoroughly, but having to calculate them anew for every square on the board made this computationally too time consuming. So it became apparent that some extra basic facilities are needed. These are described in Section 2.2. Firstly, a large table was constructed to provide a list of possible moves for any given neighbourhood. A program was written to construct lists to restrict them to manageable proportions, an interesting problem in itself. Then to sustain, minimally, the analogy with a human player, an exhaustive search is required for the very late stages of the game when it is clear that a complete analysis is necessary.

Heuristics are futile with only two pieces remaining to be played. Because it takes place at the end of the game a simplified alpha-beta search can be used. It will be clear that even here the environment constrains the search to be depth first and nothing else.

With these routines present the program can play at a reasonable rate and becomes foolproof near the end. There remains to consider how to make the program play better in the early stages and more importantly in the crucial part of the game before an exhaustive search is feasible. Chapter 3 describes what has been done to convert the detailed numerical information gathered by the updating and move-making routines of the program into something like the 'gestalt' concepts of a human player. In one view this simply condenses to a mapping of position into a smallish set of numerical values. In another, it represents an evaluation function that can be applied to positions in the latter parts of a game, once the idea is established of the importance of the moves preceding the search. Supporting routines were written that create and maintain a table of values that give a judgment on whether particular kinds of positions are good or bad. The goodness is measured by the only means that have turned up; pragmatically - did one win when that kind of position occurred in previous games?

Clearly there are too many positions to give each a distinct value so they are grouped by a numerical computation, a form of mapping that is described in Section 3.1. Also it is seen that, since the evaluation is based on past performance, therefore the ability is inherent to improve future results. The results in this chapter show the significant

improvement in playing strength over the earlier version.

Results with human opponents were more difficult to evaluate since the machine that moved randomly until near the end obtained a remarkably good record, better than 25%, we estimate, against all kinds of opponents from programmers all the way up to beginners. It is clear that pentominoes is an odd game.

CHAPTER 1 THE GAME-PLAYING MILIEU

1.1 General Considerations

Interest in mechanisms behind the playing of games is probably almost as old as games themselves. Work on mathematical solutions to puzzles of all kinds is always taking place. The fraud of Maelzl's chess automaton last century illustrated, if nothing else, a demand for and fascination with machines of that kind. In 1914 the Spanish inventor L. Torres y Quevedo produced a machine that wins the chess endgame of king and rook against king. He commented, "the limits within which thought is really necessary need to be better defined...the automaton can do many things that are popularly classed as thought."

The advent of computers revolutionised the possibilities. The original paper of Shannon [22] sparked much work in chess although there were already some hand-simulations around at the time. In fact nothing in his general suggestions need be confined to chess; there is equal applicability to checkers, tic-tac-toe, the eight-puzzle and many other popular pastimes.

However there have been many more reasons for computerised game-playing than the entertainment of games. Samuel's work with checkers [20] and [21] has stood as one of the most successful efforts in the field. His intention was to have the program play just as well as possible and to this end he incorporated as much information about valuable checker board positions as he could provide himself and from expert players. In addition, he used polynomial evaluation functions, efficiently coded representations and was one of the first to use the

alpha-beta search algorithm, before it had even been christened.

Samuel's interest was not in simulation of human thought processes but in demonstrating intelligent results using some techniques peculiar to digital computing. On the other hand, many people have been employing games for just such simulations. Newell, Shaw and Simon's chess player [16] is one of the earliest examples. They were not so much concerned with results from the most efficient programming but with the underlying problem solving processes. Games have always been useful in this area because of their formal simplicity that yet leads to very complex behaviour.

Chess has always had a special place in the programming of games and in artificial intelligence because it has been considered the game of thought, par excellence, the mastery of which would be considered to require intelligence by all but the definitional recidivists who hold that thinking is what machines don't do.

Many groups of people have built chess playing programs without any specific research goal but to make a machine play better than other machines or other people. A landmark here was Greenblatt's program [12] which included very efficient code, extensive routines for the generation of plausible moves and time saving tree pruning methods. This program beat Dreyfus who had written that such things were impossible [5]. It has been enrolled in the U.S. Chess Federation, has played in tournaments and it seems that further efforts along the same lines have little to contribute to the state of knowledge in the field.

Heuristic searching in problem solving has often used games as the environment for experiments. The Graph Traverser (Doran and Michie [6]) worked with the eight and fifteen-puzzles, Slagle [23] developed an almost unbeatable program that plays Kalah while producing refinements in tree searching and pruning techniques.

Most work in computing with games is aimed at uncovering some things unrelated to the games themselves (with the exception of chess mentioned above); either methods of human thought in problem solving or general heuristic methods. However in almost all cases there is a considerable body of knowledge beyond the rules that relates to the playing of the game. Each game has its own special approach that may make generalisable results harder to come by. There is one further class of games that have been used as research tools; those where little is known or what is known is supposed to be ignored. Zobrist's work with Go, [24], using numerical values to represent visual organisation of the stones on a Go board, is a good example and it is into this area that this work falls.

1.2 Polyominoes

Golomb [10] describes the n-ominoes as generalisations of the domino. See, for example, Figure 1. They are groups of simply connected unit squares in two dimensions. We shall be considering only pentominoes during the course of this study. From Figure 1, there are 12 basic 5-celled animals and they have been ordered and nicknamed F, I, L, N, P, T, U, V, W, X, Y, Z for convenience and due to the mild resemblance in their shapes.

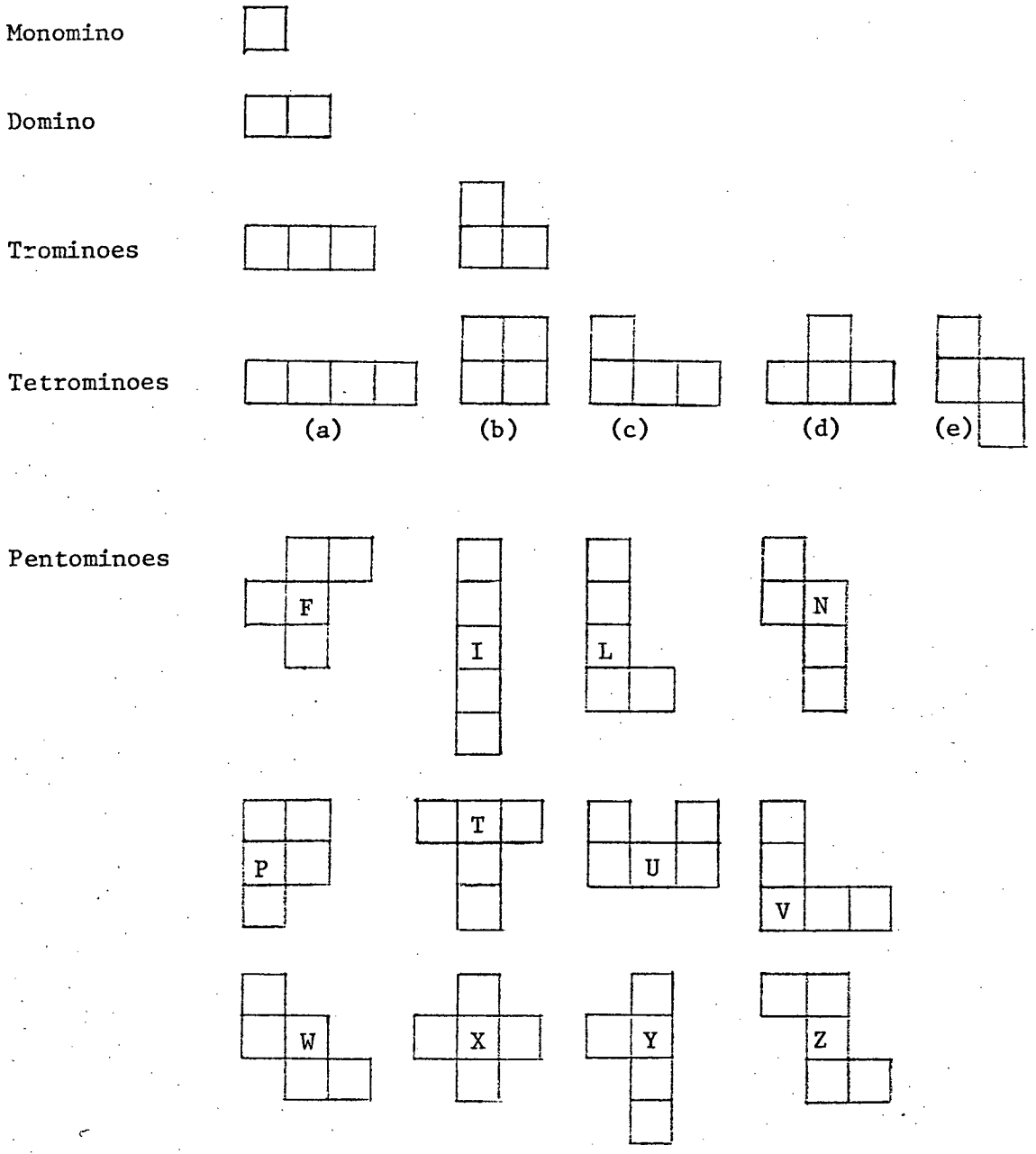


Figure 1

THE N-OMINOES FOR N ≤ 5

When one allows rotations and reflections there are then 63 different orientations to be accommodated; consisting of one for the 'X' piece to as many as eight for such as the 'F' piece (Figure 2).

Pentominoes have been mentioned in several places in the puzzle literature; perhaps one of the earliest was in the 74th problem of Dudeney [7] in 1908 who described how a chess board was broken over the head of the Dauphin by Prince Henry. While the Prince was busy escaping it was discovered that the board had been broken into 13 pieces - 12 pentominoes and the square tetromino, the problem incidentally was merely to reconstruct the board.

1.3 The Game of Pentominoes

Martin Gardner has written several articles on pentominoes in the Scientific American [9], based on communications with Golomb and he has included descriptions of possible games and some examples.

The rules adhered to while programming are quite arbitrary and merely represent some choices among many variants. A few things have been borne in mind of course; for example, it must not be possible to employ a simple algorithm in order to win and it should preferably be a reasonable game to play.

The rules that we shall use throughout are the following:-

(i) A rectangular checkerboard is available for play which need not necessarily have all its squares free.

(ii) A subset of the 12 pieces are available (in practice usually all of them). A move is made by selecting one of these pieces and placing it uniformly in some vacant space on the board.

1	F	1
2		2
3		3
4		4
5		5
6		6
7		7
8		8
9	I	1
10		2
11	L	1
12		2
13		3
14		4
15		5
16		6
17		7
18		8
19	N	1
20		2
21		3
22		4
23		5
24		6
25		7
26		8
27	P	1
28		2
29		3
30		4
31		5
32		6
33		7
34		8
35	T	1
36		2
37		3
38		4
39	U	1
40		2
41		3
42		4
43	V	1
44		2
45		3
46		4
47	W	1
48		2
49		3
50		4
51	X	1
52	Y	1
53		2
54		3
55		4
56		5
57		6
58		7
59		8
60	Z	1
61		2
62		3
63		4

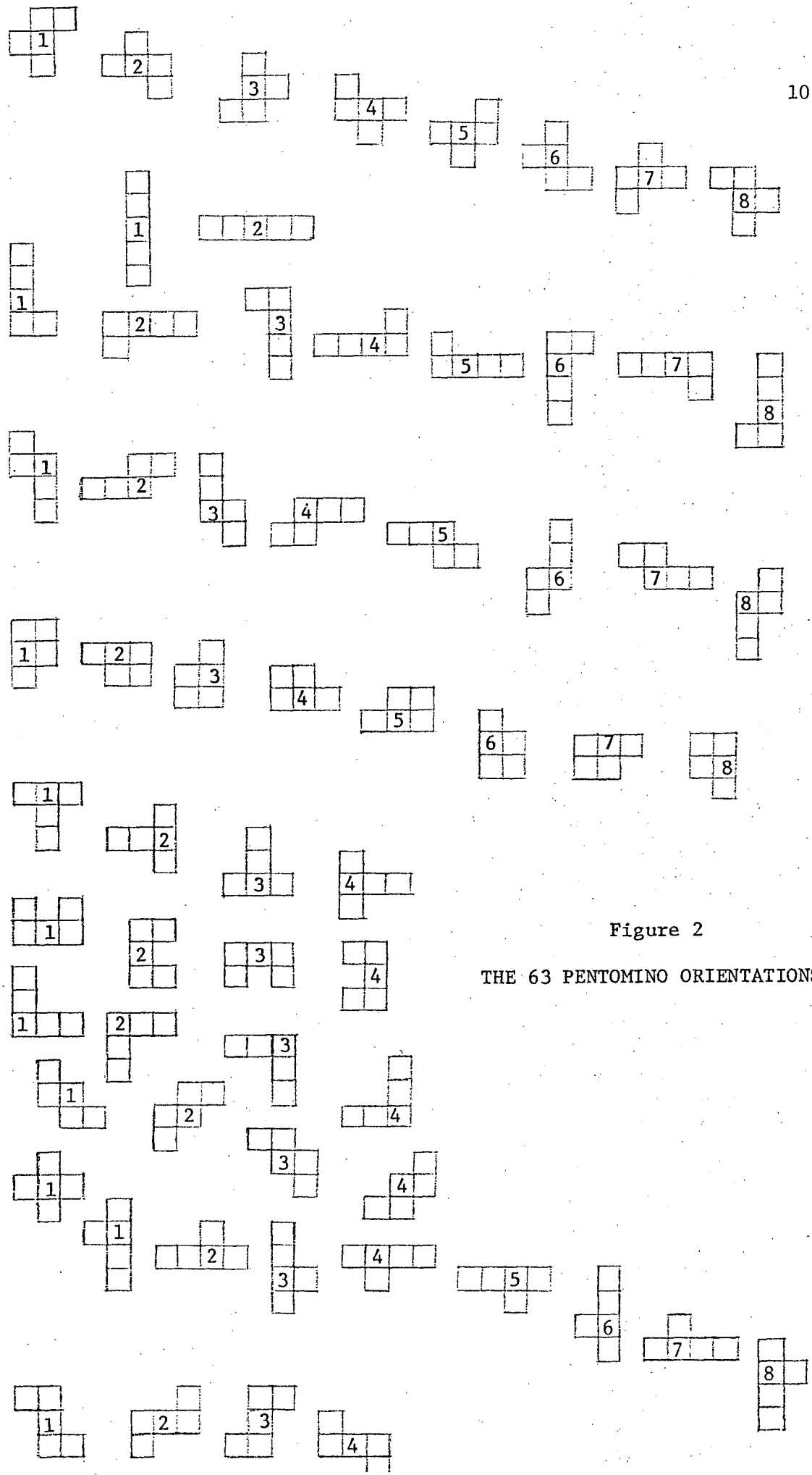


Figure 2

THE 63 PENTOMINO ORIENTATIONS

This piece is then no longer available to be chosen again and may not be moved.

(iii) A game is between two players who must move alternately. The player who is unable to move loses.

Certain rectangles recommend themselves for use. The 8 x 8 is natural, the 6 x 10 is the 'squarest' rectangle into which the 12 pieces can be made to fit exactly, the 6 x 6 board for the special reason that it is the smallest rectangle whose game outcome is unknown.

Most of our work is confined to the 6 x 10 board, concerning which it has been said that there are, "at least 5 moves, at most 12, no draw, more openings than chess." Indeed the program calculated that there are 2056 possible choices for an opening move (not all of them essentially different).

In Polyominoes [10], Golomb writes concerning strategy,

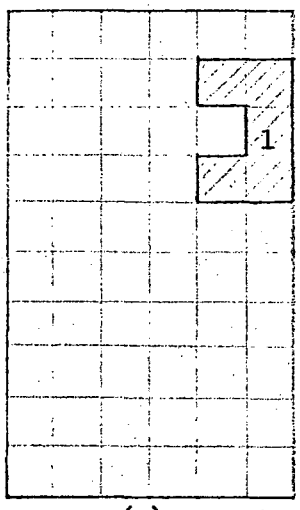
"1. Try to move in such a way that there will be room for an <u>even</u> number of pieces.

2. If a player cannot analyse the situation, he should do something to complicate the placement so that the next player will have even more difficulty analysing it than he did."

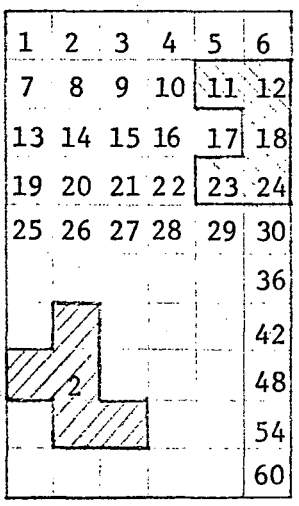
Also in the Scientific American [9], October 1965, "The most useful strategy is to try to split the board into separate and equal areas. Then there will be an excellent chance that the opponent's move can be matched by your next move and so on."

Thus, it is clear how well this game fulfils our requirement that nothing be well known in advance.

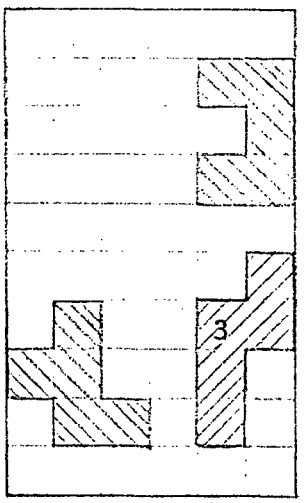
SAMPLE GAME SHOWING SQUARE NUMBERING SCHEME



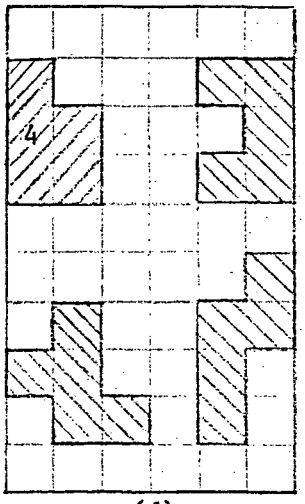
(a)



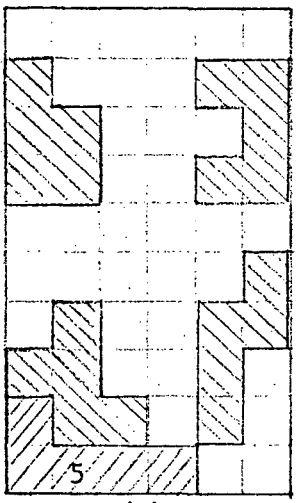
(b)



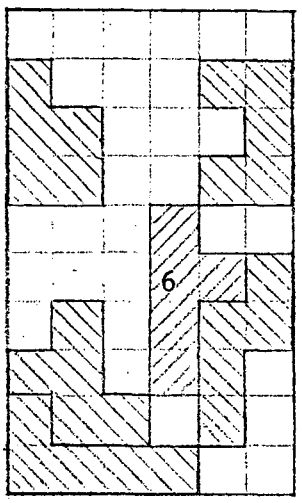
(c)



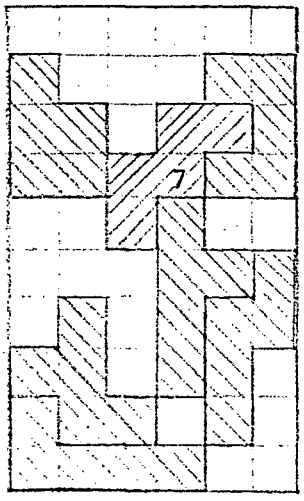
(d)



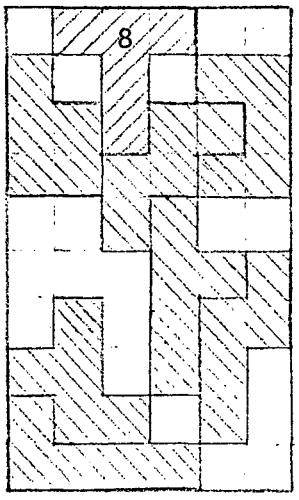
(e)



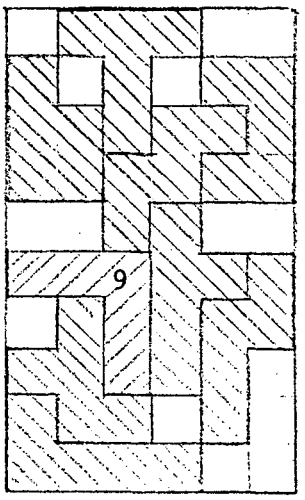
(f)



(g)



(h)



(i)

Figure 3

A sample game is illustrated in Figure 3. The second player's moves in (b) and (d) are attempts to maintain symmetry, but one can see that the concept lacks precision since no two pieces are alike. In (f) he abandons this plan and loses. In this case it is quite likely that he would have lost also by playing 'symmetrically'. In (g) the first player's fourth move ensures a win by placing the 'W' so as to divide the remaining area neatly enough into two areas into both of which one piece will fit.

The second player cannot circumvent the loss by playing the piece that fits in the second area into the first since both the 'N' and the 'T' fit into either area. Note also that there are ten free squares in the upper area and one could fit the 'W' and 'Y' pieces in there together but not here since both have already been played.

The eighth and ninth moves need not even be played since the outcome is fully determined by now. As an aside, a not too sophisticated version of the program discovered the winning move when confronted with the position after move 6, though it was different from the move shown.

Referring to the 6 x 10 game, Golomb is quoted in [9] October 1965, "...complete analysis is just at the limit of what might be performed by the best high speed electronic computer given a generous allotment of computer time and a painstakingly sophisticated program."

We have noted that there are 2056 opening moves, one quarter of which are essentially different. Selecting a typical 9 move game we find in this instance over 1100 choices after 2 moves have been made (after one move the average is over 1400). After 4 moves, 504 choices remained; and after 6, 104; and so on.

A survey of many sample games reveals at least two things clearly. Firstly, the total number of nodes in the move tree seems to be of the order of 10^{21}. A generous allowance for duplication of positions would leave ~3×10^{15} positions to be examined. An increase in efficiency of this program of 10^5 or 10^6 would reduce the time required to search the tree exhaustively to 30 years. In other words, combinatorial considerations eliminate the above mentioned possibility of a brute force approach.

Secondly, what must be noticed is the inexorable narrowing of the tree width from its overwhelmingly large start to the inevitable conclusion of the game at most 12 moves later (an average of just under 9, in fact). Just before the conclusion the game is trivial, just before that the position may be analysed, (meaning here that the outcome will be determined by inspection at length) but before that the situation is probably impossible. Considerable advantage must be taken of this structure, as will be seen later. It is possible to construe this as a major part of our knowledge of the game.

CHAPTER 2 THE PENTOMINO PROGRAM

2.1 Basic Representations

The first attempts to manipulate polyominoes computationally stemmed from attempts simply to count them, an open problem. Read [18] was one of the earliest and at that time it was not fully agreed how many 10-ominoes there are. This led to programs designed to do the same thing, the greatest attempt so far being by Lunnon [13] who has enumerated n-ominoes for $n \leq 20$ using about 150 hours of background computer time on an Atlas.

Counting like this and dissection of rectangles requires a machine representation for the pentominoes. In determining that there are 2339 ways of filling the 6 x 10 rectangle with all 12 pieces, Fletcher [8] used a representation based on the leading square of some pentomino as centre, the orientation being determined by the displacements of the remaining cells from the first within a 512 square grid which also contained the board.

Using this leading square approach he was obliged to execute a couple of special tests to reject impossible board situations. Because of this, the lack of symmetry properties, and the fact that the influence of a single piece from its leading square could extend over most of the board, that representation was rejected for this project.

The program employs the 'centre' of each pentomino, always a unique cell; the one that minimises the sum of the rook-wise distances to all the other cells. In Figure 1 the centre is the square that contains the letter of the alphabet. Clearly the 'distance' to any other cell is

never more than 2 units. The 12-neighbourhood shown in Figure 4, is used extensively, as well as its associated cell-ordering scheme. Any of the 63 orientations will fit in the 12-neighbourhood when the centres coincide.

This device is used to represent a piece and also parts of the board, internally, by translating to strings of bits.

Firstly a piece. For example orientation 50, which is W 4 is shown in Figure 5. This is uniquely represented by cells 1, 2, 5 and 7 in the 12-neighbourhood; alternatively by the bit-string (from 1 to 12) 0011 0101 1111 with zeros indicating the cells of the piece.

In complementary fashion the state of the board with respect to any particular square that we choose can be represented. In Figures 6 and 7 the neighbourhood of square 26 has cells 2, 4, 6 and 8 free so we can represent it as 0101 0101 0000 at this time.

Now this square 26 can be isolated, and in order to see which pieces can be moved here (a move meaning that the centre of the piece will be placed on this square) one needs merely to 'OR' its neighbourhood bit string with the bit strings for the orientations, and if the result is all ones then there is a 'hit'. For example, square 24 would have 0101 0001 1010 and orientation 42 which is U 4 has 1010 1100 1111.

Now (0101 0001 1010) 'OR' (1010 1100 1111) is (1111 1101 1111) thus U 4 will not fit in square 24. However for square 26 and orientation 60 or Z 1

(0101 0101 0000) 'OR' (1010 1010 1111) = (1111 1111 1111)

and it clearly fits.

In the program the board is kept as a simple array, but each square

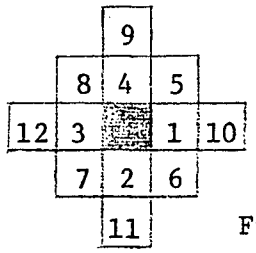


Figure 4

THE 12-NEIGHBOURHOOD

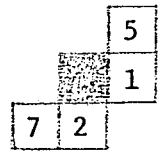


Figure 5

SQUARES OCCUPIED BY ORIENTATION W 4

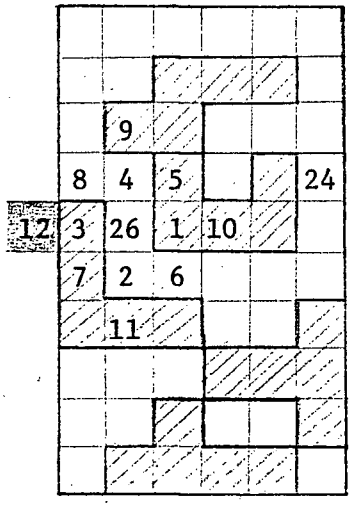
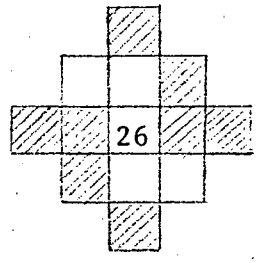


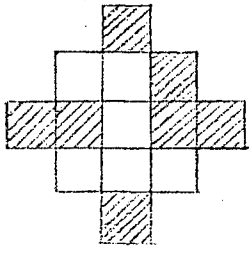
Figure 6



0101 0101 000

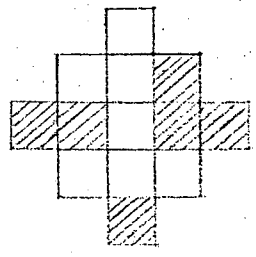
Figure 7

APPLICATION OF 12-NEIGHBOURHOOD TO MOVE DETERMINATION



0101 0111 0000

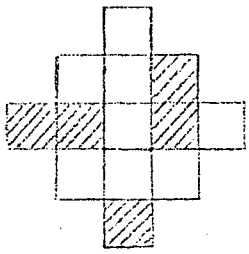
Figure 8



0101 0111 1000

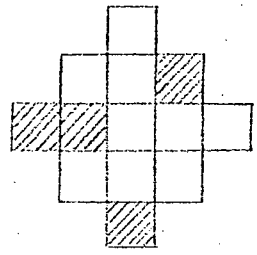
Figure 9

BIT STRINGS FOR CERTAIN NEIGHBOURHOODS



0101 0111 1100

Figure 9a



1101 0111 1100

Figure 10

is kept informed of the status of its 12-neighbourhood by having its bit string updated when moves are made nearby.

When executing, the program is informed of the board dimension and the piece availability, and then calls the routine that initialises the neighbourhoods correctly. If a move is made another routine is called that updates the situation, eliminates the piece and so on.

2.2 Some Necessary Improvements

The bit-matching tests just described are quite efficient but there are usually 60 squares to consider and 63 orientations to watch and to do every test every time would consume too much time even before we start thinking of doing some analysis of positions.

There are 2^{12} or 4096 different possible neighbourhoods. It would be useful if we could decide which neighbourhood was applicable, dial it up somewhere and receive a short list of candidate orientations, all of which are known to fit. However this would entail, instead of calculation, maintenance of 4096 tables each of up to 63 items, an unlikely trade-off unless we were desperate.

Still, there are some things to notice. The list for Figure 7 would be 60 only, and would be a subset of the list for Figure 8 which is 42, 60; which in turn is a subset of that for Figure 9; i.e., 11, 18, 42, 60. Figure 9â and Figure 9 are equivalent for this purpose. We can also see from the bit strings involved that 0000 0000 0000 requires a null list, while 1111 1111 1111 returns the full 63 orientations, and to go from the one to the other changing one bit at a time is equivalent to finding paths on a 12-cube.

Perhaps one could produce the smallest possible list of orientations which would have the property that it could be indexed for any neighbourhood and return the correct subset. A method of constructing this shortest list could not be discovered during the course of this work.

So, what has been written instead is a program that builds as few lists as possible to satisfy the requirement and concatenate them into a super-list. This turned out to comprise 4974 items, a considerable savings over the threatened $2^{12}(2^6 - 1)$ or about 250,000. With this super-list must be associated merely the 4096 indexes and lengths of sub-lists.

Thus what is done is this: for any neighbourhood, the big list is indexed to return the list of orientations that fit and these need only be checked for piece availability.

Two examples will illustrate this. Firstly, referring to Figure 9a, the binary number 010101111100 is converted to decimal 1404. A couple is sought which will give the location and size of the move list. To this end one looks into the big list at location (2 x 1404) + a constant offset of 4975 = 7783.

Consulting the chart at this location, (these parts of the lists are exhibited in Appendix 2B) one finds the couple (2185 4). So, the four possible moves are at 2185 in the big list. These are seen to be orientation 11, 18, 42 and 60. The moves are L 1, L 8, U 4, and Z 1 as was verified by inspection.

The second example, Figure 10, provides binary 110101111100 or decimal 3452. 4975 + 3452 + 3452 = 11879. At this location one finds

(3274 14). The 14 orientations at 3274 are 3, 8, 11, 18, 21, 22, 25, 32, 38, 42, 43, 49, 54, and 60. These moves are F 3, F 8, L 1, L 8, N 3, N 4, N 7, P 6, T 4, V 1, W 3, Y 3, and Z 1.

The second problem to be faced says that even if we had some heuristics to guide move selection on some more or less reliable basis, when we are sufficiently close to the end of the game, and the search tree is narrowing fast, then it is essential to stop and analyse the position thoroughly, or risk overlooking the right move. That is, to any player of this game, brute force analysis takes over eventually. Of course, deciding when this is possible during the course of a game is as challenging as the analysis itself. Because of the sudden termination feature, there has to be a complete search implemented somewhere.

This is trivial to do at the very end, and increasingly arduous earlier on until the time required to do it is clearly not available. Much judgment in playing the game is concerned with choosing the point when one must be exhaustive instead of making estimations. It is possible that one part of a successful strategy will be to arrange that it is you and not your opponent that will be left with a position of that sort.

Carrying the analogy to the machine it is equally clear that regardless of what else it is doing, when the game is detectably close enough to its finish, the program also must make an exhaustive search.

A reasonable estimate of proximity to the end is a count of the number of orientations remaining for a player to choose from in the current position, perhaps modified by the degree to which they are con-

centrated on a few squares on the board. The more widely they are scattered, the harder it is to decide the result, as a rule.

Theoretically, the 6 x 10 pentomino board and its state can be represented by a string of 60 bits, on or off for occupancy. Unfortunately the game is unlike checkers in that there is much more to a move than changing a couple of bits. In fact, given only this bit string it is difficult to calculate what is possible, let alone do it. If one wants to carry along 12-neighbourhoods and other useful information then the storage requirements in core (hundreds of bytes per position) mean that only very few nodes could be expanded at once in core, and even then much time would be used maintaining these neighbourhood statuses.

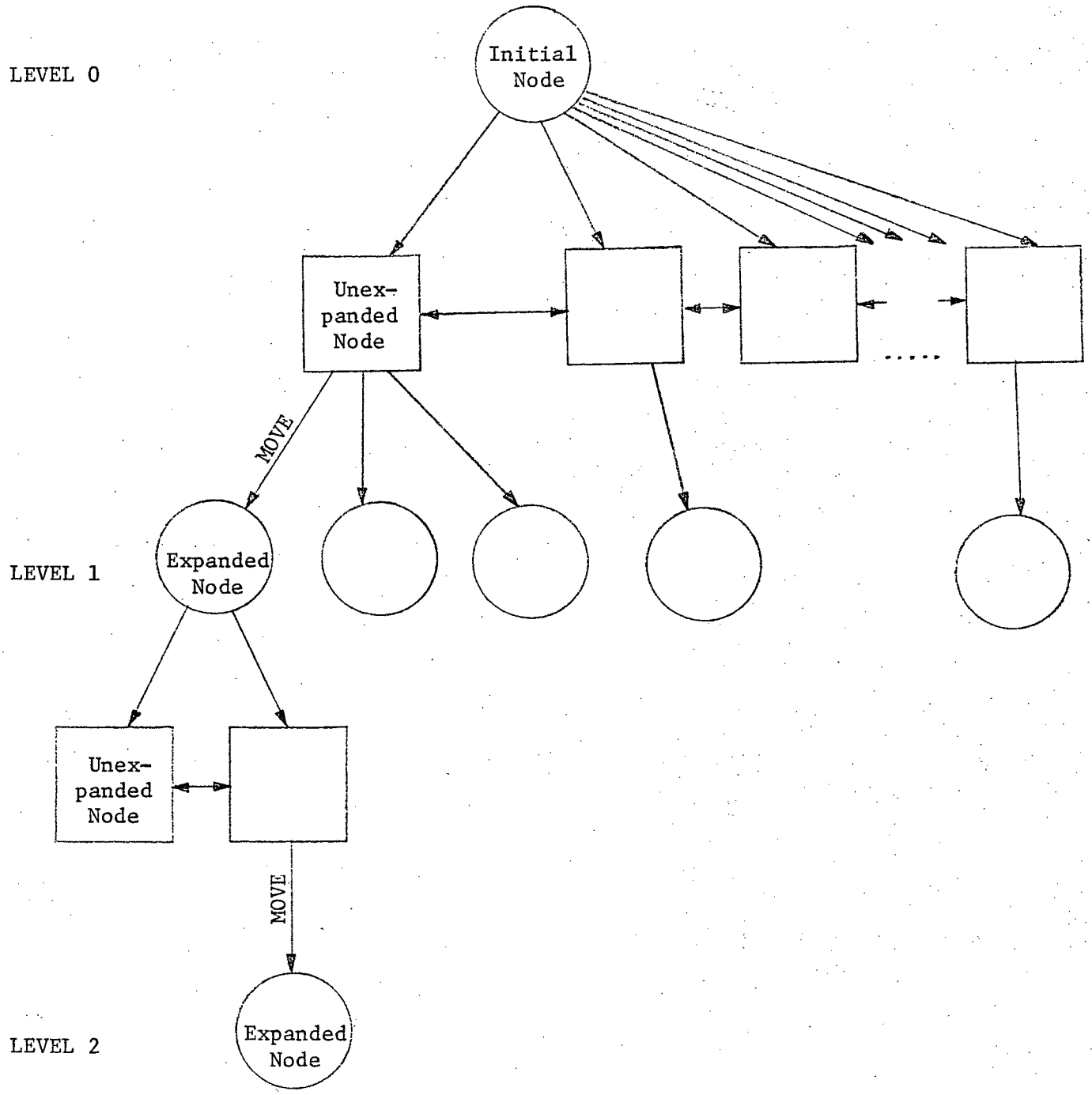
The method was selected whereby the board is not stored at all at each node but only the move made to reach the node. An abbreviated board (without details of move counts etc.) is maintained and movement up and down the game tree is made by playing and unplaying moves on this 'fast' board.

By this means, large numbers of variations can be saved and all of the remaining moves in the game can be played merely by moving down the tree. In fact, there is less value than might appear in saving the tree. The search can only be made quite close to the end of the game. Two moves later, when next the opportunity arises, the search will by now have become trivial.

Since the positions are not stored at nodes, the search is depth first by necessity. If this were not so the tree width, even at this

Figure 11

FORMAT FOR THE TREE SEARCH



late stage, is too wide for a breadth first search. The lowest depth reached is a terminal position which 'evaluates' simply to win or lose, which makes minimaxing and application of the alpha-beta procedure very straightforward.

There are two kinds of nodes in the tree, 'expanded' and 'unexpanded'. Nodes representing moves on the board are the expanded nodes. From one of these is developed an unexpanded node, in effect half a level lower down, which indicates upon which square of the board attention is being focused. In general several orientations may be placed with this square as centre, and these become the expanded nodes.

From the initial node, the full set of unexpanded nodes are developed one for each square available as a move centre. From one of these at a time the moves are played by developing the expanded nodes, then the full set of unexpanded from the current expanded node, and so on (see Figure 11).

Most nodes contain pointers to father, brothers, son etc., locations for ply level, whom to move, backed up values, etc. The unexpanded nodes record the board square and the status of the 12-neighbourhood, the expanded nodes the piece and its orientation as the move played.

The search routine is written in PL/I without using pointer variables. The search space is an array of fixed size whose garbage collection is simply controlled in a manner just as easy to implement in Fortran, say. Each node points to the next available location; the final item points to 0 (see Figure 12). When a node is freed, the node that was pointing to 0 now points to it and it now points to 0. On the IBM 360/67, if the

No. of Item		Location of Next Item
1		2
2		3
3		4
4		5
5		6
.		.
.		.
.		.
.		.
.		.
3000		0

Initially

No. of Item		Location of Next Item
1		2
2		3
3		4
4		0
5		6
.		.
.		.
.		.
.		.
.		.
3000		4

After Freeing Item 4

Figure 12

GARBAGE COLLECTION FOR THE TREE SEARCHING ARRAY

chainings were restricted to within discrete pages then there would be relatively few occasions for pointing across page boundaries. However in PL/I one doesn't control page alignments. Consequently there is some deterioration if the search continues expanding nodes long beyond the array size limit.

In practice, the search develops in excess of 200 nodes per CPU second (somewhat less when the paging demands rise excessively). A dozen integers are needed for a node plus a few bits. Each integer uses 2 bytes at least in this system. Thus each node uses 26 bytes and the array of 3000 items occupies 78,000 bytes, not too heavy a price. Using this array the search has been able to develop over 9000 nodes.

A limit of 60 move choices was set as the point when the program must make the full search; and 100 when playing human opponents. I have encountered no case where it failed with this search space. There is a very noticeable contrast in node development between those cases where a winning move is found (invariably just a few CPU seconds at most) and those where a loss must be established (frequently more than 4000 nodes or 20 seconds) i.e. if a win will be found, it will be found quickly. My tests revealed that wins could be established in positions offering well in excess of 100 choices, but that if the win was absent, the search would overflow any reasonable search space. (In other words, it is very useful to know whether one can win before trying to find out). As it is, there is evidence for the utility of cutting off the search early, when the probability of a successful outcome is much reduced.

CHAPTER 3 THE "INTELLIGENT" ADDITIONS AND THEIR EFFECTS

3.1 Aids to Move Selection

One of the final problems was how to replace the random move generator by a new one and be able to notice the improvement.

The program collects numbers as its routine goes around updating neighbourhoods after a move has been made. The routine counts the number of squares where moves are possible, and for each one of those it counts the number of orientations that are available and do fit in that square. An example from a sample game after 2 moves on a 6 x 6 board is shown in Figure 13, a total of 287 choices for a possible move with 19 possible choices for a square as centre for that move. In Figure 14, after another move there remain 137 moves and 13 centres.

These two numbers and their ratio do offer some idea of space remaining, roughly how many moves and prevalence of open areas in the position. It is certainly not known, a priori, how to convert these concepts into the correct next move. However, once several games have been played, there may exist some reason to believe that 137:13 might indicate a position where one was more likely to win than to lose. It might be found that 200:13 was unfavourable, also 137:18 and 137:8.

The program that uses such information to modify its behaviour can at least be 'punished' for its mistakes and be hoped to learn to improve its play.

The numerical measures are a form of feature extraction. If it had happened that too many heterogeneous characteristics of the position had mapped into the same small set of numbers then some alternative mapping

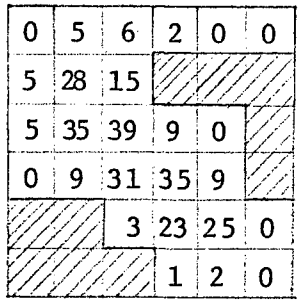


Figure 13

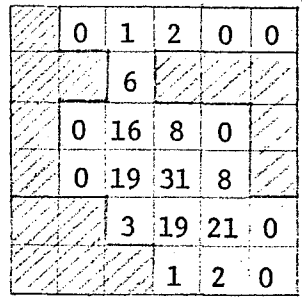


Figure 14

COUNTS OF POSSIBLE ORIENTATIONS IN THEIR CENTRE SQUARES

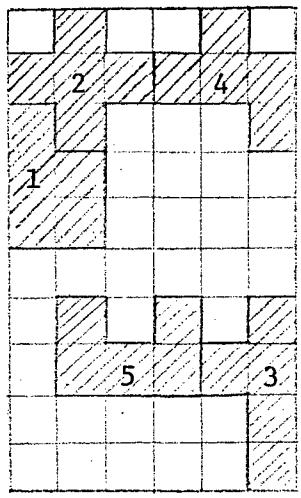


Figure 15

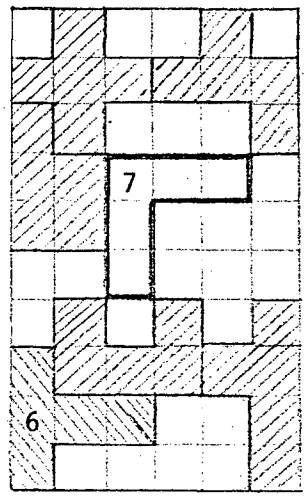


Figure 16

GAME POSITION TO ILLUSTRATE INFORMATION GATHERING PROCESS

would have to be discovered. What have been used are figures that represent concentrations of moves all over the board and from which the influences of the pieces can be deduced. This approach is somewhat akin to that of Zobrist in GO [24], since the influence of a bishop in chess is clearly enough defined but that of an anticipated move of a GO stone is not and of a Pentomino piece even less so.

In chess, and other games, one can have a clearly 'winning' position even though a forced conclusion cannot be demonstrated; for example, being a queen ahead, other things being equal. This can be measured by a count of relative material strengths, which will be part (or all) of a static evaluation of a chess position, and is only the fact that these features exist that makes such an evaluation possible.

In Pentominoes, either a win (or loss) can be demonstrated (by exhausting cases) or the position is not clear. Strategical concepts, such as symmetry and the size of the discrete, remaining areas have been mentioned, but their effect is never such as to cause a player to give up as most would when a queen down in chess.

One knows by experience that it may be wise to avoid certain kinds of positions, areas, symmetries, etc. The intention is to have the program develop similar kinds of information, pragmatically, gaining better ideas about good and bad positions by the experience of games it has played, using information that it has computed itself.

To this end one hundred games of random moves were generated within the program and the details of move counts and concentrations stored for each move. All the games were analysed from their terminating move back-

wards until the positions were beyond reasonable study. A person would not expect to spend more than 20 minutes on one position and the computer, which had some routines developed to do the same thing, rapidly became too expensive to use for this. At the position nearest to the start of the game, where the outcome was still clear, the details were recorded and built into a file later to be accessed and augmented by the program.

A 'random' game here means the following: using an available random number generator, a free square was selected, then one of the possible orientations for that square was chosen and played. This process was repeated until the game ended. The method is equivalent to choosing a location, then selecting something to place there and is distinct from at least one other random scheme that was not used, of taking a piece and finding somewhere to put it.

An example position (Figure 15) will help reveal what kind of information is to be stored. We guess that it is too difficult to examine exhaustively (it may not be) so one more move is generated, move 6, the T piece at square 49 in its fourth orientation, abbreviated T 4 49, see Figure 16.

Now the result is easy to determine. There are 114 choices of moves in 15 available squares. The move suggested by the program, V 2 21 will win since exactly two moves remain.

Returning to Figure 15, assume that the program is turning over moves. It computes that if it plays T 4 49, there will remain 114 moves in 15 squares. We have seen that it would then lose, so we want some way to tell it that this is a position to be avoided so that no opponent

should be lucky enough to be left with it.

So we make an entry opposite 114 under number of squares = 15. We subtract 8 from the stored value. This value is arbitrary but is arranged in order to obtain a ripple effect. If 114:15 is a poor position to leave, then so may well be 113:15 or 115:15, only less conclusively so; thus the subtractions go as follows:

111:15	-1
112:15	-2
113:15	-4
114:15	-8
115:15	-4
116:15	-2
117:15	-1

The position before the T was placed had 161 choices in 27 squares. Even though it cannot be analysed (supposedly) we guess that it may have opposite favourability from its successor (perhaps we should leave this position to the opponent so he can leave us the good one). So we file away:

158:27	+1
159:27	+2
160:27	+3
161:27	+4
162:27	+3
163:27	+2
164:27	+1

The numbers are smaller since we are less convinced. The position before move 5 had 483 choices in 34 squares, as a last gesture we store:

482:34 -1

483:34 -2

484:34 -1

A set of routines have been written that receive a series of move counts and the amount to be added or subtracted and rebuild the list structure that will reflect updated values.

Currently, all the pointers and position evaluations are stored in an array of just over 3400 numbers. The complete list is provided in Appendix 2D. Its operation can be made explicit by giving an example of how the program uses it to compare potential moves.

Suppose that the program is trying to find the most favourable move in the position drawn in Figure 15. It will play the moves hypothetically and then examine the resulting positions to count the number of moves remaining open and their respective centres. If this number of moves lies between 49 and 520, there may be information to look up.

Imagine that one move leaves 89 choices in 15 squares. The pointer for 89 is found in the list offset by 47, i.e., location 42 (Appendix 2D). Thus the values are to be found beginning at location 946. There are pairs of scores, a value for move centre counts 13, 15, 16, 23 and 25. The value given for 89:15 is -10, a poor choice.

As a second case, suppose that another move leaves the position 75:20. Now 75 - 47 = 28. Pointer at 28 = 798. Scanning through the pairs starting at 798, the value for centre count = 20 is +1.

Clearly the program will prefer the move that leaves this position. In cases such as this one, all moves will be tested and the one with the highest score will be chosen.

3.2 Experimental Results

On examining the 100 random games in some detail and incorporating the assumption that the foolproof minimax search took over when there were 60 or fewer choices for a move, we found that the first player wins 55 games and the second player wins 45.

36 games were played with the modified version of the program competing against a version of itself which had no access to the guidance tables, but did employ the search equally well at the end. An average of 60 seconds CPU time per game and limited access to the computer prevented really large scale tests.

When the modified program moved first (15 cases) it won 12 and lost 3. When it moved second, it won 11 and lost 10. These results are perhaps more remarkable than was expected. The newer version obtaining a plus score in all cases, considering the unusual random effects in this game (such as seeming to have a 'good position' but losing anyway.)

During the internecine phase of one program version playing against another, the minimax search was initiated only when 60 or fewer move choices remained (the limit was 100 when people participated). As its sole strategy, some version could be programmed to make every attempt to avoid giving its opponent the chance to invoke the search routine. By being the one to search first it could be suggested that a crucial advantage would be gained since most other strategies could be claimed

to be only marginally effective.

This was not so in practice. Several times during games, a program was forced to move without searching. Its opponent was then allowed the futile gesture of exhaustively searching to discover that it was just about to lose. In fact, no version used the search cut-off point in decision making; mostly because it had to perform against people, and they take no notice of such arbitrary, and to them undetectable, boundaries.

What of games between people and versions of the program? It would have been very expensive to undertake the series of controlled experiments required to be able to distinguish the 'smart' version from the random program with search, because of the high random success rate and the large amounts of CPU time needed for each game.

How should one measure the improvement in an under or overmatched opponent? Not by counting victories. It must be noted that in the final series of 6 controlled games with human subjects as opponents the machine lost but once. That loss was with the improved version, the random version won all its 3 games.

What one tries to search for instead (viz. Samuel [20]) is an improvement in the class of individual moves. These moves would be compared with those suggested by acknowledged good players (sadly, hard to come by in Pentominoes).

Here is the most noticeable improvement, the newest version makes individual moves that make sense intuitively to its opponent; so much so that one subject ascribed to the rationality of its play his ability to beat it when he did. The random player, on the other hand, was confusing;

no one could tell what it was up to, as, of course, it was up to nothing at all until the search phase.

There are four sample games in Appendix 2E for illustration. Game one shows the random machine moving second against its human opponent. In Figure 17a the program has generated moves 2, 4 and 6, the last being particularly indifferent. This game serves just as well as an example of random versus random, they looked much like this.

The first player plans to leave just 2 moves but makes the mistake of using one of the two pieces that fit around squares 6 and 12. The program in its search uses the other one to finish the game. The line of figures above the winning move shows that the search took less than one CPU second to expand only about 46 nodes. The human could have won by playing the N piece as shown in Figure 17c.

In game 2, the subject plays first and plays his first moves around the edge of the board. The program which now has access to its tables plays its first move with the intention of reducing the open space and plays in the middle of the board. Its position in Figure 18a is very difficult and the move it chose seems to be a good try. Unfortunately, its opponent can find the move of the T piece, in Figure 18c, leaving moves of the N and Y pieces in the areas shown. The program verifies this in a 24-node search; its play now exhibits some purpose, whether this is relevant to winning is another matter.

In game 3, the program moves first, and after 4 moves the position it considers is Figure 19a. The move of the N piece that it plays (Figure 19b) is remarkably strong, quite possibly winning against all

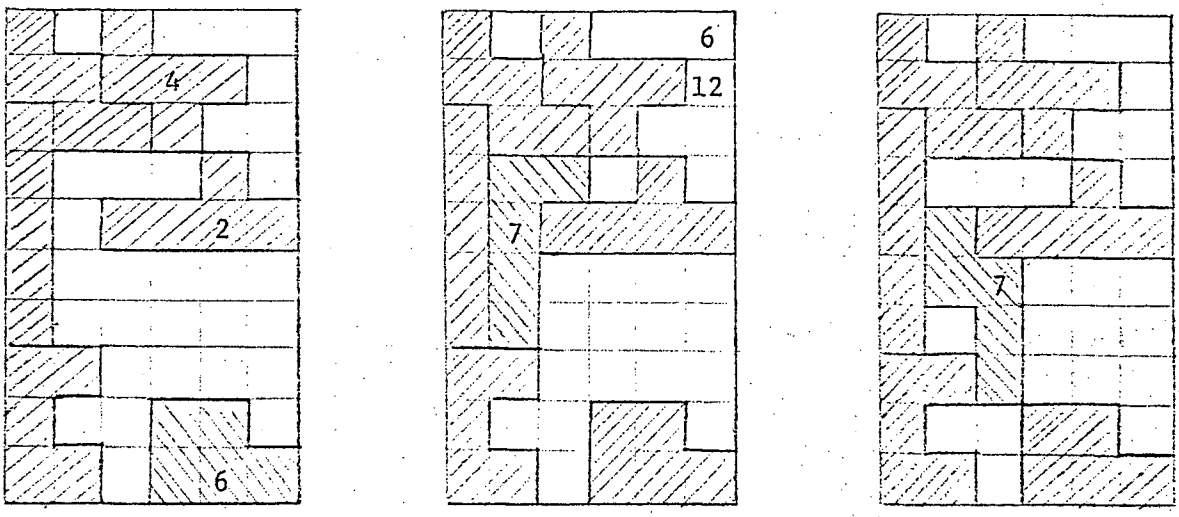


Figure 17

POSITIONS FROM EXAMPLE GAME ONE

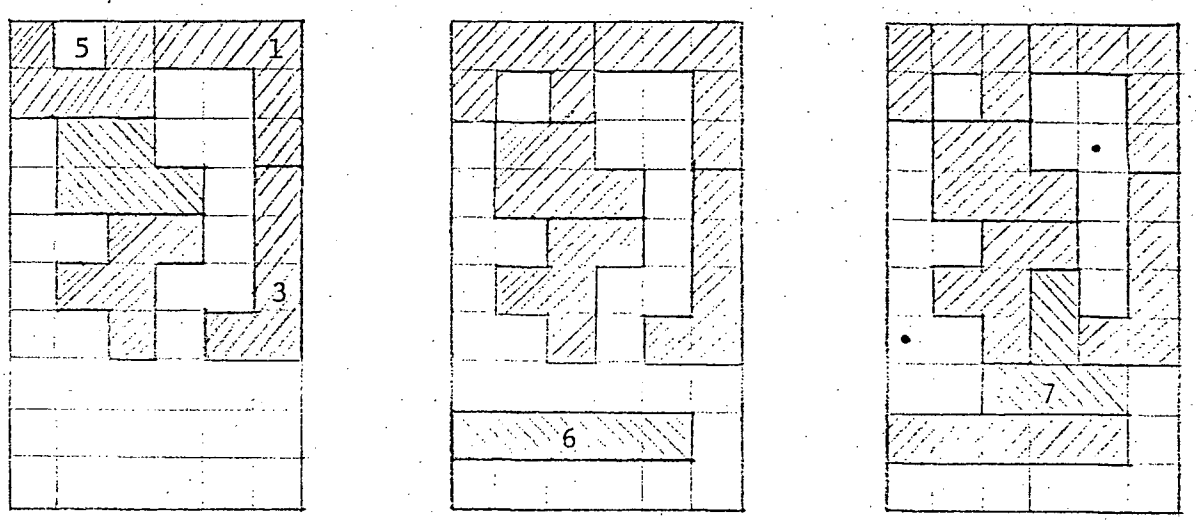


Figure 18

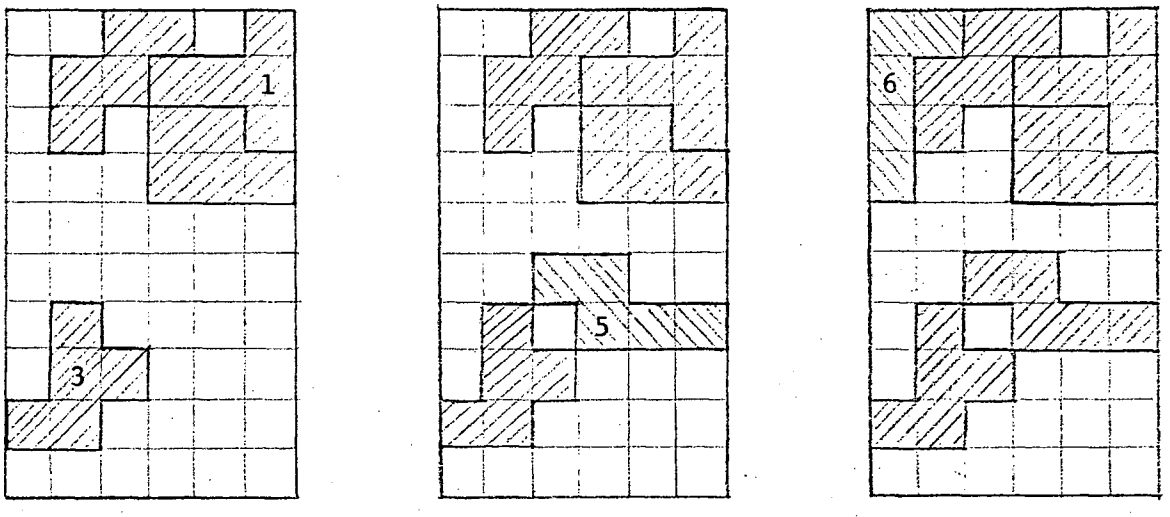
POSITIONS FROM EXAMPLE GAME TWO

replies. The opponent's choice of the U piece allows the game to finish in one move, after a 121 node search. Placing the L piece in Figure 19c looks like a better attempt to parry. Why it fails might entertain the reader to discover.

In game 4 the program moves first again, (its first moves are very likely to be different from each other.) Once again, after examining the position after 4 moves (in Figure 20a) the program plays a fine move (Figure 20b). Finding it annoyingly hard to analyse, its opponent played the Y piece to close the area at the top of the board (Figure 20c). After 5 seconds CPU time and over 1000 nodes the program's search yields a forced win as shown. In all its games, the longest successful search took 42 seconds CPU time expanding over 8200 nodes from a position with 100 move choices.

The author is forced to say that this program plays a respectable game of Pentominoes since it once beat him when he was trying to win. It may be pointed out that there is no 'algorithm' (or set of moves) which would enable anyone to win against this program. This is not the case with many game-playing programs that use an evaluation function. In their cases, once a winning line has been found it can be repeated until the programmer changes his program.

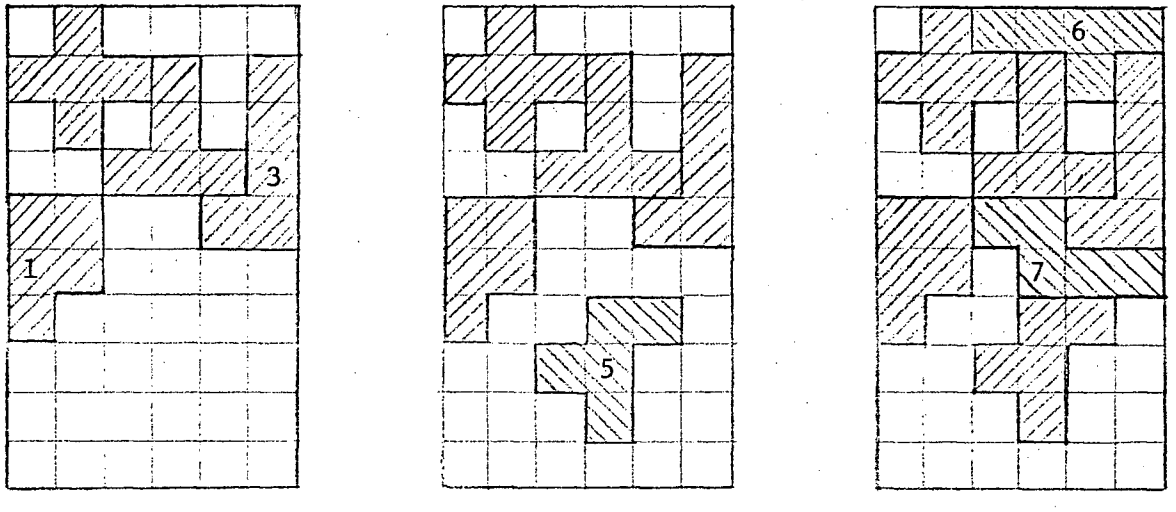
For one thing here, the first move is chosen randomly from 2056 (the random choice among equals is usually the most that an evaluation function achieves in varying from game to game). More crucially, the results of all games are open to analysis, the results of which will be reflected in a new edition of the move evaluation lists. Positions that



a b c

Figure 19

POSITIONS FROM EXAMPLE GAME 3



a b c

Figure 20

POSITIONS FROM EXAMPLE GAME 4

led to a loss will be penalised heavily enough to dissuade the program from choosing moves that lead there. Appendix 2D contains the second edition of the lists. One generally bad position had had a few favourable results during the random games, but not later against real opposition. These positions of later games were the information used in the update.

CHAPTER 4 CONCLUSIONS

The problems encountered in this research give rise to conclusions that can be extended beyond pentominoes to embrace much of the work in game playing and heuristic searching.

As Minsky [15] writes, planning is now a more crucial aspect than before (compared with searching) in the human problem solving areas of artificial intelligence. He says this is because there is reason to believe that human chess or checker players consider only dozens, or at most, a few hundred different positions when deciding on a move. Some programs have commonly evaluated hundreds of thousands. He claims that these were early works and that later on there were reductions thanks to development of some theory.

This theory, however, is heavily dependent on specialised knowledge of a problem area of which the programmer is aware - the ability to 'evaluate' a position or node in a graph. Without this we suggest that the theory might as well not exist for all the help it can offer.

It is clear from the closing remarks in the last chapter that, in some sense, this Pentomino playing program can 'learn'. However, we deliberately omitted the routines that update the evaluation lists from the main body of the program in order not to give an exaggerated impression. Since there is no alteration to the conceptual model into which the new information is received, this 'learning' had much better be called adaptation.

It does have the advantage of being selective. After the start given to the program by the analysis of the 100 games, not each trial has to be

recorded in these tables. Thus the problem that was encountered in BOXES [14] is avoided, which was that the effect of an erroneous move given early reinforcement was still apparent thousands of trials later.

It is suggested that the 'evaluation' of positions that is carried out by the program is analogous in some ways to that of human players of games who learn to abstract certain favourable features from their positions. In spite of its very naive representation (in the form of straightforward lists) in its own way it is more natural than the artificial calculations employed by many game playing programs, and perhaps some of its effects could be incorporated into more orthodox forms.

'Evaluation' is crucial not only to game playing but also to heuristic search methods in general. How crucial it is has not been made explicit. As we have seen, strategies result from knowledge of the game or search space that is apart from the formal rules. When there is almost no such knowledge, nodes in the search tree come to be indistinguishable. Slagle [23] finds a very useful evaluation function for Kalah, if there had not been one the heuristic search would have stopped before it began. Nilsson [17] is trying to improve searching by cost estimation methods, the 'estimate' which must be available at any node of the graph is an evaluation of course. In its absence, there is nothing left.

In the GO-playing program of Ryder [19] the issue of position evaluation has been shelved for the sake of calculating numerous local effects, which enable him also to discover plausible moves and to avoid the search in a space that explodes combinatorially like few others do; hence his

mere passing reference to the work of Zobrist [24].

Pentominoes has a paucity of local considerations indeed, there is so little strategical information that it is doubly remarkable that such an effective playing program has been produced. The Game is a counter-example to the argument that there will always exist strategical information to use when one wants to make an heuristic-type search. In fact, humans make 'plausible moves' when they interact socially, not based on rules of strategy or an evaluation function so much as using the very large amount of relevant information that they have access to internally. The development, storage and use of complex information relevant to playing games seems to be quite an important direction if computer game playing research is not going to descend to a competition just to beat the latest digital chess champion. Minsky [15] suggested that what is needed for summarising search trees is not a numerical utility-like value but a description-like expression that can be used for analysis.

In chess, Botvinnik [1] has produced a way of mathematically representing the board, using horizons, which will make it easier to consider plans and actions. He pointed out that the chess playing programs of the last 20 years had got bogged down because they first coded a scheme for a chess board and men failing to know firstly what they intended to do. He expects to produce a very good player this way but his view seems inadequate.

Clarke [2] goes so far as to propose a special language for dealing with chess (Algol 64). This seems quite interesting, but like Botvinnik

must face the problem of the large body of information accumulated by experience.

De Groot [3] took protocols from many levels of chess players with the intention of distinguishing a Grandmaster from a Master and so on. This was a relative failure. There were clearly things happening below the level of the most introspective protocol that the players were unaware of. Another series of tests that he did ([3] and [4]) were more revealing. A position was exposed to various players for just a few seconds, and they were then asked to reconstruct the position. Only the Grandmaster was able to produce the full correct position and also the winning move! Lesser players, typically, recalled less. This only applied to 'meaningful' chess positions.

The problem, of course, in computing, is how to build up a 'vocabulary' that increases comprehension of chess positions in parallel. This problem of large amounts of new knowledge will not only occupy game playing research but also the computer linguists working with semantic models.

The view, exemplified by Minsky [15], that 1,000,000 'facts' will be enough for a great intelligence, is clearly inadequate even if someone knew what a 'fact' was. It is shown to be so by the current work of Papert et al. on comprehending very young children's stories. The problem is one central to all of Artificial Intelligence, not only to game playing.

REFERENCES

[1] Botvinnik, M.M. (1970), Computers, Chess and Long Range Planning, Springer-Verlag.

[2] Clarke, M.R.B. (1971), "Some Ideas for a Chess Compiler", NATO Symposium on Human Thinking - Computer Techniques for its Evaluation, St. Maximin, France.

[3] De Groot, A.D. (1965), Thought and Choice in Chess, G.W. Baylor (ed.), The Hague and Paris: Mouton.

[4] _____ (1966), "Perception and Memory Versus Thought: Some Old Ideas and Recent Findings," in B. Kleinmuntz (ed.), Problem Solving: Research, Method, and Theory, New York, Wiley, pp.19-50.

[5] Dreyfus, H.L. (1965), Alchemy and Artificial Intelligence, Santa Monica, California, Rand Corp.

[6] Doran, J. and D. Michie (1966), "Experiments with the Graph Traverser Program," Proc. R. Soc. (A), 294, pp.235-259.

[7] Dudeney, H.E. (1908), The Canterbury Puzzles, Dover (1958).

[8] Fletcher, J.G. (1965), "A Program to Solve the Pentominoes Problem by the Recursive Use of Macros," Comm. ACM 8(10), pp.621-623.

[9] Gardner, M. "Mathematical Games Department" in Scientific American Magazine.

[10] Golomb, S. (1965), Polyominoes, Scribner's, New York.

[11] Good, I.J. (1968), "A Five-Year Plan for Automatic Chess," in E. Dale and D. Michie (eds.), Machine Intelligence 2, American Elsevier Publishing Company, Inc. pp.89-118.

[12] Greenblatt, R., D. Eastlake and S. Crocker (1967) "The Greenblatt Chess Program," Proc. FJCC, pp.801-810.

[13] Lunnon, W.F. (1971), "Counting Polyominoes," in Computers in Number Theory, Atkin and Birch (eds.), Academic Press, New York, pp.347-372.

[14] Michie, D. and R.A. Chambers (1968), "Boxes: An Experiment in Adaptive Control," in E. Dale and D. Michie (eds.), Machine Intelligence 2, American Elsevier Publishing Company, Inc. pp.137-152.

[15] Minsky, M. (1968), Semantic Information Processing, Introduction, M.I.T. Press, pp.1-31.

[16] Newell, A., J.C. Shaw and H.A. Simon (1963), "Chess Playing Programs and the Problems of Complexity," IBM J. Res. Dev., 2, pp.39-70. Reprinted in E. Feigenbaum and J. Feldman (eds.), Computers and Thought, McGraw-Hill Book Co., 1963.

[17] Nilsson, N.J. (1969), "Searching Problem Solving and Game Playing Trees for Minimum Lost Solutions," in A.J.H. Morrell (ed.), Information Processing 68, Vol.2, pp.1556-1562, North-Holland, Amsterdam.

[18] Read, R.C. (1962), "Contributions to the Cell Growth Problem," Can. J. Math. XIV, pp.1-20.

[19] Ryder, J.L. (1971), Heuristic Analysis of Large Trees as Generated in the Game of GO, Stanford University AIM-155.

[20] Samuel, A.L. (1959), "Some Studies in Machine Learning Using the Game of Checkers," IBM J. Res. Dev.3(3), pp.210-229. Reprinted in E. Feigenbaum and J. Feldman (eds.), Computers and Thought, McGraw-Hill Book Co., 1963.

[21] _____ (1967), "Some Studies in Machine Learning Using the Game of Checkers II - Recent Progress," IBM J. Res. Dev. 11(6), pp.601-617.

[22] Shannon, C.E. (1950), "Programming a Computer for Playing Chess," Phil. Mag. 41, pp.256-275.

[23] Slagle, J.R. and J.K. Dixon (1969), "Experiments with Some Programs that Search Game Trees," JACM 16(2), pp.189-207.

[24] Zobrist, A.L. (1969), "A Model of Visual Organisation for the Game of GO," Proc. SJCC, Vol. 34, pp.103-112.

APPENDICES

1. Operation of the Programs

The main sections of this program were written in PL/I. Most of the external supporting routines were written in Fortran, due to being exposed to some of the fine points of PL/I implementation and support.

The game-player PENTO expects its input to be in PL/I LIST format, i.e., items must be followed by spaces.

One can begin by typing something like:

```
$R  RSR1:MHKØB + NEW:PL1LIB      PAR=FFF=RSR1:GGG
MHKPNF = RSR1:MHKPNF
```

The program spends a while reading in its rather cumbersome files and then types 'INPUT NEW GAME SPECS'.

We must then type in some numbers:

A B C D E F G H

A is the area of the board's rectangle (eg. 60)

B is the number of columns (6)

C is the number of rows (10)

D represents the number of squares of the rectangle that are unavailable

E is the number of pieces unavailable

F is the debugging mode (usually 0)

G is the minimaxing search limit (defaults to 60)

H is the mode of play

Note: If D > 0, we must next input the list of the numbers of the squares concerned, similarly if E > 0 the list of pieces concerned must follow.

If F is not 0 then a plethora of helpful (to the programmer) values are likely to be printed out. The values that can be taken by H are as follows (some are now redundant):

1 You move first, program plays randomly with search.

2 Program moves first, otherwise as 1.

3 You play all the moves.

4 The program plays all the moves.

5 This was once used to gather statistics about first moves.

6 A device used to analyse some games while playing backwards.

7 Used to generate and file a series of random games (an amount controlled by G).

8-10 Not used now.

11 You move first, program has access to search and all tables.

12 Program moves first, as 11.

To play a move one must type something of this form:

'T' 3 45 remembering the space at the end or 'N' 7 34

To restart a game, type 999 instead of the 3. One can always terminate with $END.

One might also wish to update the file of move evaluations. Say, for example, the choice to centre ratio 87:21 gave a bad result. And by extrapolation 196:28 a good one in the same game, and 402:38 was the state the move before. Then one would input the following lines:

```
$SOURCE MHKLD
 87 21 1   (in format 3I4)
196 28 2
402 38 3
 $END
```

The following is the content of MHKLD:

```
$R MHKLOBJ 4=MHKPNF 5=*MSOURCE* 6=-WMK 7=*MSINK*

$R *PERMIT PAR=MHKPNF NONE

$E MHKPNF

$C -WMK MHKPNF

$R *PERMIT PAR=MHKPNF RO
```

2. Listings of Routines, Table and Games

A. The Pentomino Playing Program

B. The Move Lists (Excerpts)

C. The Evaluation List Maintenance Program

D. The Move Evaluation Tables

E. Four Sample Games Against Human Opponents

1 The Random Version with Search

2, 3 and 4 The Improved Version

(A) LISTING OF THE MAIN PROGRAM

```
   PENTO: PROC OPTIONS(MAIN);
DCL
 CPUTIME ENTRY RETURNS(FLOAT BIN),
 (WTIM,NTIM) STATIC FLOAT BIN,
  MVW INIT(50),
 MVCNRW,MCNCRW,
 MAVL(12) BIT(1),
 1 JIN, 2 JX BIT(4) INIT('0000'B), 2 JL(12) BIT(1), MJ DEFINED JIN,
 MFSTBD(256) INIT ((256)2),
 MKA(4008) STATIC,
 MSLST(13166) STATIC,
 MUD BIT(32) INIT((32)'0'B),
 LINE DEC FIXED(9,3),
 ASIZE, MO BIN(15,4), MRCH(6),
 1 STATS(100), 2 MOVES, 2 GSIZE, WESG CHAR(8),
 TIME ENTRY(BIN(31,0),BIN(31,0),BIN(31,0)),
 (EMAT,EMUT,MITA,MITB,MITC,MITD,MITE) BIN(31,0),
 MGCNT INIT(1),
 1 KVAR(7), 2 KPNO, 2 KONO, 2 KOSQ,
 CARA CHAR(1),
 MMM(12) INIT((12)0) STATIC,
 XXN FIXED BIN,
 PANTRY(12) BIT(12) INIT('100000000000'B,'010000000000'B,
   '001000000000'B,'000100000000'B,'000010000000'B,'000001000000'B,
   '000000100000'B,'000000010000'B,'000000001000'B,'000000000100'B,
   '000000000010'B,'000000000001'B),
 XPANTRY(12) BIT(12),
 TFOPS(12) STATIC FIXED BIN INIT(1,9,11,19,27,35,39,43,47,51,52,60),
 SPONO(64) STATIC FIXED BIN INIT((8)1,2,2,(8)3,(8)4,(8)5,(4)6,(4)7,
  (4)8,(4)9,10,(8)11,(4)12),
 SST(64) STATIC
    BIT(12) INIT('100001111111'B,'010010111111'B,'001011011111'B,
                 '000111101111'B,'000101111111'B,'100010111111'B,
                 '010011011111'B,'001011101111'B,
                 '101011110101'B,'010111111010'B,
                 '101010110111'B,'010111011011'B,'101011101101'B,
                 '010101111110'B,'010111101011'B,'101001111101'B,
                 '010110111110'B,'101011010111'B,
                 '100111101101'B,'110001111110'B,'011010110111'B,
                 '001111011011'B,'100110111110'B,'110011010111'B,
                 '011011101011'B,'001101111101'B,
                 '001001111111'B,'000110111111'B,'100011011111'B,
                 '010011101111'B,'010001111111'B,'001010111111'B,
                 '000111011111'B,'100011101111'B,
      '000111111101'B,'100011111110'B,'010011110111'B,'001011111011'B,
      '010101101111'B,'101000111111'B,'010110011111'B,'101011001111'B,
      '011011110011'B,'001111111001'B,'100111111100'B,'110011110110'B,
      '100110101111'B,'110001011111'B,'011010101111'B,'001101011111'B,
      '000011111111'B,
      '100011111101'B,'010011111110'B,'001011110111'B,'000111111011'B,
      '000111111110'B,'100011110111'B,'010011111011'B,'001011111101'B,
   '101010101111'B,'010101011111'B,'101001011111'B,'010110101111'B),
1 SFT(64) STATIC,
 2 MSFF(4) INIT(-16,-15,-1,16,-16,-1,1,17,-16,1,15,16,-17,-1,1,16,
   -15,-1,1,16,-16,-1,16,17,-16,-1,1,15,-17,-16,1,16,
  -32,-16,16,32,-2,-1,1,2,
  -32,-16,16,17,-1,1,2,15,-17,-16,16,32,-15,-2,-1,1,
  -17,-1,1,2,-16,-15,16,32,-2,-1,1,17,-32,-16,15,16,
  -17,-1,16,32,-16,-15,-2,-1,-32,-16,1,17,1,2,15,16,
```

```
   -2,-1,16,17,-32,-16,-1,15,-17,-16,1,2,-15,1,16,32,
   -16,-15,1,16,-1,1,16,17,-16,-1,15,16,-17,-16,-1,1,
   -16,-15,-1,1,-16,1,16,17,-1,1,15,16,-17,-16,-1,16,
   -1,1,16,32,-16,-2,-1,16,-32,-16,-1,1,-16,1,2,16,
   -17,-15,-1,1,-16,-15,16,17,-1,1,15,17,-17,-16,15,16,
   -32,-16,1,2,1,2,16,32,-2,-1,16,32,-32,-16,-2,-1,
   -17,-1,16,17,-16,-15,-1,15,-17,-16,1,17,-15,1,15,16,
   -16,-1,1,16,
   -16,-1,16,32,-16,-2,-1,1,-32,-16,1,16,-1,1,2,16,
   -2,-1,1,16,-32,-16,-1,16,-16,-1,1,2,-16,1,16,32,
   -17,-16,16,17,-15,-1,1,15,-16,-15,15,16,-17,-1,1,17),
  IX,KTEST,IY,IZ,IZA(5),
  MRAT BIN(15,5), MESG CHAR(8), DY FIXED BIN,
  BTEST BIT(12), BTS(12) BIT(1) DEFINED BTEST,
      PAVL(12) FIXED BIN,
    PONO(12) FIXED BIN INIT(8,2,8,8,8,4,4,4,4,1,8,4),
  MDNA(12) INIT(1,0,-1,0,1,1,-1,-1,0,2,0,-2),
  MDNB(12) INIT(0,1,0,-1,-1,1,1,-1,-2,0,2,0),
  UPD(12) BIT(12) INIT('011111111011'B,'101111111101'B,'110111111110'B,
                       '111011110111'B,'111101111111'B,'111110111111'B,
                       '111111011111'B,'111111101111'B,'111111110111'B,
                      '111111111011'B,'111111111101'B,'111111111110'B),
  ALLSET BIT(12) INIT((12)'1'B),
  ALLOFF BIT(12) INIT((12)'0'B),
LNKA BIT(12),
  LK(12) BIT(1) DEFINED LNKA,
  LNKB BIT(12), LJ(12) BIT(1) DEFINED LNKB,
  1 LNKE, 2 LNKF BIT(4) INIT('0000'B), 2 LNKD BIT(12),
  LNKG DEF LNKE,
  XCHKA(4) FIXED(1) INIT(2,1,2,1),
  XCHKB(4) FIXED(1) INIT(4,3,4,3),
  YCHKA(4) FIXED(1) INIT(6,6,7,5),
  YCHKB(4) FIXED(1) INIT(5,7,8,8),
  INT1 BIT(12) INIT('111001100111'B),
  INT2 BIT(12) INIT('101110011101'B),
  INT3 BIT(12) INIT('110111001110'B),
  INT4 BIT(12) INIT('011100111011'B),
  ALPH(12) CHAR(1) INIT('F','I','L','N','P','T','U','V','W','X','Y','Z'
   ),
  TMRLNK(12) STATIC FIXED BIN INIT(3,4,1,2,7,8,5,6,11,12,9,10);
  ON ENDFILE (SCARDS) GOTO REPENT;
  GOTO SEE; SAW: JONETM=0;
  IF MSLST(13166)¬=63 THEN DO;
   PUT LIST((MSLST(I) DO I=13142 TO 13166)); STOP; END;
 GSTART:
  PUT LIST ('INPUT NEW GAME SPECS'); PUT SKIP(2);
  GET LIST (L,M,N,NA,NP,KTEST,MVW,MDNDX);
  IF JONETM=0 THEN DO; JONETM=1;
  IF MDNDX=6 THEN GOTO TOO; FRO:
  IF KTEST=-2 THEN PUT LIST(MKA);
  END;
  JINITS,JINIT=0;
  IF MDNDX¬=7 THEN GOTO GSTARTA;
  MOVES(*)=0; GSIZE(*)=1;
  WESG=DATE; MITA=SUBSTR(WESG,7);
  MITB=SUBSTR(WESG,4,2); MITC=SUBSTR(WESG,1,2);
  MITE=MITA+100*MITB+10000*MITC; EMAT=EMAT*MITE/10000;
  PUT LIST(DATE); PUT SKIP; PUT FILE(PSTAT) LIST(DATE);
  GSTARTA: IF L ¬= M*N THEN DO;
   MXXN = 1; GO TO XX; END;
```

```
 IF NA >= L | NP >= 12 THEN DO;
 MXXN = 2;
XX: PUT LIST ('NO GAME',MXXN); GOTO REPENT; END;
 IF MVW=0 THEN MVW=60;
 XPANTRY(1) = '011111111111'B;
 XPANTRY(2) = '101111111111'B;
 XPANTRY(3) = '110111111111'B;
 XPANTRY(4) = '111011111111'B;
 DO K = 5 TO 12;
  XPANTRY(K) = UPD(K); END;
 IF MDNDX=8 THEN DO;
 KLOOQ: IF MGCNT>MVW THEN GOTO REPENT; KH=0;
  KLOOP: KH=KH+1; GET SKIP FILE(PSTAT) LIST(CARA);
  IF CARA='*' THEN DO; KH=KH-2; GOTO KLOOR; END;
  GET FILE(PSTAT) LIST(KONO(KH),KOSQ(KH)); KI=1;
  KLOP: IF CARA=ALPH(KI) THEN GOTO KLOQ; KI=KI+1; GOTO KLOP;
 KLOQ: KPNO(KH)=KI; KONO(KH)=KONO(KH)+TFOPS(KI)-1; GOTO KLOOP;
 KLOOR: IF LOOTT=1 THEN DO; LOOTT=0; MGCNT=MGCNT+1;
  GOTO KLOOQ; END;
 KH=KH-1; IF KH<1 THEN GOTO KLOOQ; END;
BOARD: BEGIN; DCL
 ARCH(M) CHAR(1),
 MVALLH,MCNCW,MVALLL,MCNH,MCNL,MVCNTA,MMAXFG INIT(0),
 LNKC BIT(12),
 1 MCS(L), 2 MCNS, 2 MCNSQ, 2 MCNSD(8) BIT(1),
 1 LSQ,
  2 NBOARD(L),
   4 LNK BIT(12),
   4 MVCNT,
   4 SWS,
    6 CNTRE BIT(1),
    6 INFSW BIT(1),
    6 DEAD FIXED BINARY,
  2 MVCNTALL INIT(0),
  2 MCNCN INIT(0);
 DEAD,MVCNT=0;
 PAVL=0; MAVL='1'B;
 MCNSD(*,1)='1'B;
 IF MDNDX=8 THEN DO; NA,NP=1; DO I=1 TO KH;
 PAVL(KPNO(I))=1; BTEST=SST(KONO(I)); IZB=KOSQ(I);
 DEAD(IZB)=KPNO(I); JJ=2; KK=1;
 DO WHILE(JJ<6); IF BTS(KK)='0'B THEN DO;
 IZC=IZB+MDNA(KK)+M*MDNB(KK);
  DEAD(IZC)=KPNO(I); JJ=JJ+1; END;
  KK=KK+1; END; END; GOTO INTL; END;
 IF NA ¬= 0 THEN     BEGIN;
 DECLARE A(NA) FIXED BIN;
  GET LIST ((A(II) DO II = 1 TO NA));
  DO II = 1 TO NA;
  DEAD(A(II)) = 13; END; END;
 IF NP ¬= 0 THEN     BEGIN;
 DECLARE B(NP) FIXED BIN;
  GET LIST ((B(II) DO II = 1 TO NP));
 DO II=1 TO NP;
 MAVL(B(II))='0'B;
  PAVL(B(II)) = 1; END;  END;
INTL: BEGIN;
  DO I = 1 TO L;
 LNK(I) = ALLSET;
   IF I <= M THEN LNK(I) = LNK(I) & INT1; ELSE
```

```
  IF I<=M+M THEN LNK(I)=LNK(I) & UPD(9);
  IF I > M * (N - 1) THEN LNK(I) = LNK(I) & INT2; ELSE
   IF I > M * (N -2) THEN LNK(I) = LNK(I) & UPD(11);
  IF MOD(I,M) = 1 THEN LNK(I) = LNK(I) & INT3; ELSE
  IF MOD(I,M)=2 THEN LNK(I)=LNK(I)&UPD(12);
   IF MOD(I,M) = 0 THEN LNK(I) = LNK(I) & INT4; ELSE
   IF MOD(I,M) = M - 1 THEN LNK(I) = LNK(I) & UPD(10);
 END;
 IF NA=0 THEN GOTO INC;
INB:
 DO I = 1 TO L;
 IF DEAD(I) ¬= 0 THEN DO;
  LNKB = LNK(I);
 DO J = 1 TO 12;
  IF LJ(J) ¬= '0'B THEN DO;
  K=I+MDNA(J)+M*MDNB(J);
 IF DEAD(K) = 0 THEN DO;
 LNK(K) = LNK(K) & UPD(TMRLNK(J));
 END INB;
INC: DO I = 1 TO L;
 IF DEAD(I)¬=0 THEN GOTO NDINC;
  LNKD = LNK(I);
 LNKH=4975+LNKG+LNKG;
 LNKI=MSLST(LNKH); IF LNKI=0 THEN GOTO NDIND;
 LNKJ=MSLST(LNKH+1); IF NP=0 THEN DO;
  MVCNT(I)=LNKJ; GOTO NDIND; END;
 DO II=0 TO LNKJ-1;
 IF PAVL(SPONO(MSLST(LNKI+II))) =0 THEN MVCNT(I)=MVCNT(I)+1;
 END;
   NDIND:
 IF MVCNT(I)>0 THEN DO CNTRE(I)='1'B;
   MCNCN = MCNCN + 1;
 MVCNTALL=MVCNTALL+MVCNT(I); END; ELSE CNTRE(I)='0'B;
 NDINC: END INC;
  MCNCW=MCNCN;
  END INTL;
CHIEF: BEGIN;
VAGA: MXF=0;
 IF (MDNDX=1)|(MDNDX=3)|(MDNDX=11) THEN GOTO THEFOE;
 IF (MDNDX=6)|(MDNDX=12) THEN GOTO APMV;
 VAGB:
 IF MVCNTALL ¬= 0 THEN GOTO PMV;
 IWIN = 0;
GEND:
 IF MDNDX>2 THEN  DO;
 MITA=1; MITB=0; CALL TIME(MITA,MITB,MITC);
 MITC=MITC-MITD;
 IF MDNDX=6 THEN
 IF JINIT=1 THEN MESG=' SMT WON';
 ELSE MESG='SMT LOST'; END;
 ELSE IF MDNDX=99 THEN MESG='PARTIAL ';
 ELSE MESG = ' '; ELSE IF IWIN = 1 THEN
  MESG = 'I WIN'; ELSE MESG = ' YOU WIN';
 PUT LIST (MESG,MGCNT,JINITS,MITC);
 IF MDNDX=7 THEN DO; PUT LIST(MOVES(MGCNT),GSIZE(MGCNT));
  PUT SKIP FILE(PSTAT) LIST('*',MOVES(MGCNT),GSIZE(MGCNT));
 END; PUT SKIP(2);
  DO KK = 1 TO N;
  DO LL = 1 TO M;
   DY = DEAD(M * (KK - 1) + LL);
```

```
  IF DY = 0 THEN ARCH(LL) = '0'; ELSE
   IF DY < 13 THEN ARCH(LL) = ALPH(DY);
   ELSE ARCH(LL) = '*'; END;
  PUT SKIP(2) EDIT (ARCH) ((M)(X(1), A(1)));  END;
 PUT SKIP(3);
 IF MGCNT=MVW THEN GOTO WINDUP;
 MGCNT=MGCNT+1;
 GOTO NDBD;
WINDUP: PUT SKIP(2) LIST('NO. OF MOVES','NO. OF GAMES');
 ASIZE,MO=0;
 DO II=1 TO MGCNT;
 ASIZE=ASIZE+GSIZE(II); MO=MO+MOVES(II);
 MMM(MOVES(II))=MMM(MOVES(II))+1; END;
 MO=MO/MGCNT; ASIZE=ASIZE/MGCNT;
 DO II=1 TO 12; PUT SKIP LIST(II,MMM(II)); END;
 PUT SKIP FILE(PSTAT) EDIT(MMM) (12(X(2),F(2)));
 PUT SKIP(2) LIST('TOTAL GAMES','AVGE MOVES','AV PROD OF CHOICES');
 PUT SKIP LIST(MGCNT,MO,ASIZE);
 PUT SKIP FILE(PSTAT) LIST(MGCNT,MO,ASIZE);
 GOTO REPENT;
PMV:
 IF MDNDX<3 THEN
 IF MVCNTALL<=MVW THEN DO; CALL MMAX;
 IF MXF=1 THEN GOTO EPMV; GOTO APMV; END;
 IF MDNDX=8 THEN DO; CALL MMAX; GOTO KLOOR; END;
APMV: MITA=MCNCN; MITB=7;
  CALL KUTRAND(MITB,MITA,EMAT); MQDX=EMAT;
 KLU = 0; KLUX = 1; KKK:
  IF DEAD(KLUX) = 0 THEN
   IF CNTRE(KLUX) = '1'B THEN
 KLU = KLU + 1;
 IF MQDX <= KLU THEN GOTO KKKA;
  KLUX = KLUX + 1; GOTO KKK; KKKA:
 MITC=8; MITE=MVCNT(KLUX);
 CALL KUTRAND(MITC,MITE,EMAT); KLAN=EMAT;
 KWIZ = 0; J = 0; LNKD = LNK(KLUX);
 LNKI=MSLST(4975+LNKG+LNKG); II=0;
GEN: DO WHILE (KLAN > KWIZ);
 IF PAVL(SPONO(MSLST(LNKI+II)))=0 THEN KWIZ=KWIZ+1;
 II=II+1; END;
  J=MSLST(LNKI+II-1);
 IZ = KLUX; IX = SPONO(J); IY = J - TFOPS(IX) + 1;
EPMV:
 MRAT=MVCNTALL/MCNCN;
 PUT SKIP EDIT (ALPH(IX),IY,IZ,MVCNTALL,MCNCN)
 (X(2),A(1),X(2),F(2),X(2),F(2),X(2),F(4),X(2),F(2));
 IF MDNDX=7 THEN DO; PUT EDIT(MRAT) (X(2),F(5,2));
  PUT SKIP FILE(PSTAT) EDIT(ALPH(IX),IY,IZ,MVCNTALL,MCNCN,MRAT)
  (X(2),A(1),X(2),F(2),X(2),F(2),X(2),F(4),X(2),F(2),X(2),F(5,2));
  MOVES(MGCNT)=MOVES(MGCNT)+1;
 GSIZE(MGCNT)=GSIZE(MGCNT)*MVCNTALL; END;
 PUT SKIP(2);
 CALL UPDT;
 IF MVNDX = 0 THEN OUCH: DO;
  PUT LIST ('INV MV GEN'); STOP; END;
 IF MDNDX=7 THEN IF MVCNTALL¬=0 THEN DO; LL,KK=0;
 XYZ: LL=LL+1; KK=KK+1;
  MRCH(LL)=0;
  IF DEAD(KK)=0 THEN IF CNTRE(KK)='1'B THEN MRCH(LL)=MVCNT(KK);
  IF LL¬=M THEN GOTO XYZ;
```

```
  PUT SKIP(2) EDIT(MRCH) (6(X(1),F(2))); LL=0;
  IF KK<L THEN GOTO XYZ; PUT SKIP(3); END;
 IF MDNDX>10 THEN GOTO THEFOE;
 IF MDNDX=6 THEN GOTO BSMTR;
 IF MDNDX > 2 THEN GOTO VAGA;
THEFOE:
 IF MVCNTALL = 0 THEN DO; IWIN = 1;
  GOTO GEND; END;
THEFA: PUT LIST('YOUR MOVE'); PUT SKIP(1); GET LIST (CARA,J,IZ);
 II=1;
CLOP: IF II>12 THEN DO; J=J+100; GOTO CLOQ; END;
 IF CARA=ALPH(II) THEN DO; J=J-1+TFOPS(II); GOTO CLOQ; END;
 II=II+1; GOTO CLOP; CLOQ:
 IF J >9998 THEN STOP;
 IF J >998 THEN DO;
  MVCNTALL,MCNCN = 0; MDNDX=99; GOTO GEND; END;
 CALL UPDT;
 IF MVNDX = 0 THEN DO;
  PUT LIST ('             IT IS STILL'); GOTO THEFA; END;
 IF MDNDX<3 THEN GOTO VAGB;
 IF MDNDX<11 THEN GOTO THEFOE;
 JINIT=2;
BSMTR:
 IF MVCNTALL=0 THEN GOTO GEND;
 IF JINIT=2 THEN JINIT=1; ELSE IF JINIT=1 THEN JINIT=2;
 ELSE DO; MITA=8; MITB=100; CALL KUTRAND(MITA,MITB,MITC);
 IF MITC>50 THEN JINIT=2; ELSE DO;
 JINIT,JINITS=1; END; JINITS=JINITS+1;
 MITE=1; MITA=0; CALL TIME(MITE,MITA,MITD); END;
 IF MVCNTALL<=MVW THEN DO;
 IF MXF=1 THEN GOTO APMV; CALL MMAX;
 IF MXF¬=1 THEN GOTO APMV; GOTO EPMV; END;
 ELSE IF JINIT=2 THEN GOTO APMV;
 ELSE  MVSMPLR: BEGIN;
DCL 1 LOOKING(MCNCN), 2 LKLOC, 2 LKORN, 2 MVCNR, 2 MCNCR,
 2 MSCOR, 2 MVLDX, 2 MVFLP, 2 MVLSDP;
 MCNI=1;
 JINDX=M+19; DO II=1 TO N; JINDX=JINDX+16-M;
 DO JJ=1 TO M; KK=M*(II-1)+JJ;
 IF DEAD(KK)=0 THEN DO; MFSTBD(JINDX)=0;
 IF CNTRE(KK)='1'B THEN DO; MCNS(MCNI)=KK; MCNSQ(MCNI)=JINDX;
 MCNI=MCNI+1; END; END;
 ELSE MFSTBD(JINDX)=1; JINDX=JINDX+1; END; END;
 LKMCN=0; MVLDX=0;
MVMKR: DO II=1 TO L;
 IF DEAD(II)=0 THEN IF CNTRE(II)='1'B THEN DO;
 LKMCN=LKMCN+1; LNKD=LNK(II);
 MLOOK=MSLST(4975+LNKG+LNKG);
 IF MLOOK =0 THEN DO;
  PUT LIST('TRAP 10'); GOTO NDMVMKR; END;
 MLKD=0; LKLOC(LKMCN)=II; MVFLP(LKMCN)=MLOOK;
 MVLSDP(LKMCN)=MSLST(4976+LNKG+LNKG);
LKLP: LKORNW=MSLST(MLOOK+MLKD);
 IF MAVL(SPONO(LKORNW))='1'B THEN DO;
 MVLDX(LKMCN)=MLKD; LKORN(LKMCN)=LKORNW; GOTO MVMKRS; END;
 MLKD=MLKD+1; MVLDX(LKMCN)=MVLDX(LKMCN)+1;
 IF MLKD<MVLSDP(LKMCN) THEN GOTO LKLP;
 PUT LIST('TRAP 11'); GOTO NDMVMKR;
MVMKRS:
 CALL FPLAY(LKORNW,MCNSQ(LKMCN),1); CALL KOUNT;
```

```
  MVCNR(LKMCN)=MVCNRW; MCNCR(LKMCN)=MCNCRW;
  CALL FUNPLAY(LKORNW,MCNSQ(LKMCN));
   IF KTEST=-3 THEN PUT LIST(LOOKING(LKMCN)); NDMVMKR: END MVMKR;
 MVTSTR: DO II=1 TO MCNCN;
  CALL SCORER(MVCNR(II),MCNCR(II),MSCOR(II)); END;
  IF KTEST=-3 THEN PUT LIST(MSCOR);
  LKSCW=-2000;
  DO II=1 TO MCNCN;
  IF MSCOR(II)>LKSCW THEN DO;
  LKPTR=II; LKSCW=MSCOR(II); END;
  IF MSCOR(II)=LKSCW THEN
  IF MVCNR(II)<MVCNR(LKPTR) THEN LKPTR=II; END;
  IF LKSCW¬=-900 THEN GOTO CNTSCH;
 FNDMV:  IZ=LKLOC(LKPTR);
  J=LKORN(LKPTR); IX=SPONO(J);
  IY=LKORN(LKPTR)-TFOPS(IX)+1; GOTO EPMV;
 CNTSCH: IF MVCNTALL>700 THEN GOTO APMV;
  DO II=1 TO MCNCN;
  MVLDX(II)=MVLDX(II)+1;
  DO WHILE(MVLDX(II)<MVLSDP(II));
  LKORNW=MSLST(MVFLP(II)+MVLDX(II));
  IF MAVL(SPONO(LKORNW))='1'B THEN DO;
  CALL FPLAY(LKORNW,MCNSQ(II),1);
  CALL KOUNT; CALL SCORER(MVCNRW,MCNCRW,LKSCOR);
  IF LKSCOR>LKSCW THEN DO;
  LKORN(II)=LKORNW; MVCNR(II)=MVCNRW;
  MCNCR(II)=MCNCRW; LKSCW,MSCOR(II)=LKSCOR;
  LKPTR=II; END;
  CALL FUNPLAY(LKORNW,MCNSQ(II)); END;
  MVLDX(II)=MVLDX(II)+1; END; END;
  GOTO FNDMV; END MVSMPLR;
 NCHIEF: END CHIEF;  GOTO GSTART;
UPDT: PROCEDURE;
  IF IZ > L | IZ < 1 THEN DO;
   PUT LIST ('NO SQUARE'); GOTO MV; END;
  IF DEAD(IZ) > 0 THEN DO ;
   PUT LIST ('DEAD SQUARE'); GO TO MV; END;
  IX = SPONO(J);
  IF(J<1 | J>63) THEN DO;
   PUT LIST ('NO SUCH ORIENTATION'); GO  TO MV; END;
  IF PAVL (IX) = 1 THEN DO;
   PUT LIST ('PIECE USED UP'); GOTO MV; END;
  BTEST = SST(J);
  IF ALLSET ¬= (LNK(IZ) | BTEST) THEN DO;
   PUT LIST ('NO FIT'); GO TO MV; END;
  PAVL(IX) = 1; MAVL(IX)='0'B;
  INFSW (*) = '0'B;
  IZA(1) = IZ; DEAD(IZ) = IX; JJ = 2;
  DO I = 1 TO 12 WHILE (JJ<6);
   IF BTS(I) = '0'B THEN DO;
    IZA(JJ)=IZ+MDNA(I)+M*MDNB(I);
    DEAD(IZA(JJ))=IX; JJ=JJ+1; END; END;
 DOUP: DO LL = 1 TO 5;
  IF CNTRE(IZA(LL)) = '1'B THEN DO;
   MCNCN = MCNCN - 1;
   MVCNTALL = MVCNTALL - MVCNT(IZA(LL)); END;
  LNKB = LNK(IZA(LL));
  DO MM = 1 TO 12;
  IF LJ(MM) = '1'B THEN DO;
   K=IZA(LL)+MDNA(MM)+M*MDNB(MM);
```

```
 IF DEAD(K) = 0 THEN DO;
  INFSW(K) = '1'B;
 LNK(K) = LNK(K) & UPD(TMRLNK(MM));
  END DOUP;
CHK:  DO NN = 1 TO L;
 IF DEAD(NN) ¬= 0 THEN GOTO NCHK;
 IF CNTRE(NN) = '0'B THEN GOTO NCHK;
 LNKD = LNK(NN);
 LNKH=4975+LNKG+LNKG;
 LNKI=MSLST(LNKH); IF LNKI=0 THEN GOTO CHB;
 MVCNTA=0; LNKJ=MSLST(LNKH+1);
 IF INFSW(NN)='0'B THEN DO; DO II=0 TO LNKJ-1;
  IF SPONO(MSLST(LNKI+II))=IX THEN MVCNTA=MVCNTA+1; END;
  IF MVCNTA=MVCNT(NN) THEN GOTO CHB;
  MVCNTALL=MVCNTALL-MVCNTA; MVCNT(NN)=MVCNT(NN)-MVCNTA;
  GOTO NCHK; END; ELSE DO; DO II=0 TO LNKJ-1;
 IF PAVL(SPONO(MSLST(LNKI+II)))=0 THEN MVCNTA=MVCNTA+1; END;
  IF MVCNTA=0 THEN GOTO CHB;
  MVCNTALL=MVCNTALL-MVCNT(NN)+MVCNTA; MVCNT(NN)=MVCNTA;
   GOTO NCHK; END;
 CHB: MVCNTALL=MVCNTALL-MVCNT(NN);
   CNTRE(NN) = '0'B; MCNCN = MCNCN - 1;
NCHK: END CHK;
  MVNDX = 1;
 IF MMAXFG=0 THEN MCNCW=MCNCN;
 RETURN;
MV: PUT SKIP(1); MVNDX = 0; RETURN;
    END UPDT;
KOUNT: PROC;   MVCNRW,MCNCRW=0;
 JINDX=M+19; DO MM=1 TO N; JINDX=JINDX+16-M;
 DO JJ=1 TO M; KK=M*(MM-1)+JJ;
 IF DEAD(KK)=0 THEN IF CNTRE(KK)='1'B THEN
 IF MFSTBD(JINDX)=0 THEN DO;
 CALL NEIGH(JINDX); MWIND=4975+MJ+MJ;
 IF MSLST(MWIND)>0 THEN DO;;
 LKLN=MSLST(MWIND+1); LKLC=MSLST(MWIND);
 LKDP,LKCT=0;
 DO WHILE(LKDP<LKLN);
 IF MAVL(SPONO(MSLST(LKLC+LKDP)))='1'B THEN DO;
 LKCT=LKCT+1; MVCNRW=MVCNRW+1; END;
 LKDP=LKDP+1; END;
 IF LKCT>0 THEN MCNCRW=MCNCRW+1;
 END; END; JINDX=JINDX+1;
 NDKOUNT: END KOUNT;
SCORER: PROC(MSCMV,MSCMC,MSCORW);
 MSCORW=0;
 IF MSCMV>520 THEN DO; MSCORW=-900; RETURN; END;
 IF MSCMV<49 THEN DO; MSCORW=-1000; RETURN; END;
 IF MSCMC>46 THEN DO; MSCORW=-950; RETURN; END;
 IF MSCMC<10 THEN DO; MSCORW=-980; RETURN; END;
 MTSTNDX=MKA(MSCMV-47);
 IF MTSTNDX=0 THEN RETURN;
NRLP: MKAW=MKA(MTSTNDX);
 IF MKAW=0 THEN DO;
 MSCORW=MKA(MTSTNDX-1)*.5; RETURN; END;
 IF MKAW>MSCMC THEN DO;
 MSCORW=(MKA(MTSTNDX-1)+MKA(MTSTNDX+1))*.5; RETURN ; END;
 IF MKAW<MSCMC THEN DO;
 MTSTNDX=MTSTNDX+2; GOTO NRLP; END;
 MSCORW=MKA(MTSTNDX+1);
```

```
  END SCORER;
MMAX: PROCEDURE;
 DCL  WRES BIT(2),
  1 SPLIST(3000),
  2 MDEP, 2 MLEN, 2 MNEI,
   2 MLBR, 2 MRBR, 2 MSON, 2 MORN, 2 MLOC, 2 MLEV, 2 MNEX, 2 MFAT,
   2 MINX, 2 MMOV BIT(2), 2 MRES BIT(2);
  MWRK=0; KODEC,NODEC=1;
  MCNSFZ,MCNI,MMAXFG,MTRAV,MARK=1;
  MNEXZ=2;
  DO II=1 TO 12;
   IF PAVL(II)=0 THEN MAVL(II)='1'B; ELSE MAVL(II)='0'B; END;
  DO II=1 TO 2999; MNEX(II)=II+1; END;
   MNEX(3000),MLBR(1),MRBR(1),MORN(1),MLOC(1)=0; MMOV(1)='01'B;
   MNEI(1),NORN,NLOC=0;
  MRES(1)='10'B;
  MSON(1)=-999; MLBR(1),MRBR(1)=0;
  MWND=3000; MLEV(1)=1; MCNSD(*,1)='1'B;
  JINDX=M+19; DO II=1 TO N; JINDX=JINDX+16-M;
  DO JJ=1 TO M;  KK=M*(II-1)+JJ;
  IF DEAD(KK)=0 THEN DO; MFSTBD(JINDX)=0;
   IF CNTRE(KK)='1'B THEN DO; MCNS(MCNI)=KK; MCNSQ(MCNI)=JINDX;
   MCNI=MCNI+1; END; END;
    ELSE MFSTBD(JINDX)=1;
  JINDX=JINDX+1;  END; END;
 CONROT:
  IF KTEST>3 THEN
   PUT LIST('CNRT',SPLIST(MTRAV));
   IF MTRAV=MARK THEN IF MRES(MTRAV)<'10'B THEN DO;
    NTIM=CPUTIME; MBTM=NTIM-WTIM; GOTO MMA; END;
     ELSE WTIM=CPUTIME;
  IF MRES(MTRAV)='11'B THEN GOTO ACREN;
  IF MRES(MTRAV)='10'B THEN GOTO CRUN;
  IF MSON(MTRAV)=0 THEN DO;
   IF MNEI(MTRAV)¬=0 THEN DO;
    IF MRBR(MTRAV)=0 THEN DO;
  MRES(MFAT(MTRAV))=MRES(MTRAV);
   CALL DLT(MTRAV);
   IF MLBR(MTRAV)¬=0 THEN DO;
    MTRAV=MLBR(MTRAV); MRBR(MTRAV)=0;
    MSON(MFAT(MTRAV))=MTRAV; END;
   ELSE MSON(MFAT(MTRAV))=0;
   MTRAV=MFAT(MTRAV); GOTO CONROT; END;
  IF MLBR(MTRAV)¬=0 THEN MRBR(MLBR(MTRAV))=MRBR(MTRAV);
    CALL DLT(MTRAV); MTRAV=MRBR(MTRAV);
    MLBR(MTRAV)=MLBR(MLBR(MTRAV)); MSON(MFAT(MTRAV))=MTRAV;
    GOTO CONROT; END;
  MRES(MFAT(MTRAV))=MRES(MTRAV);
  CALL FUNPLAY(MORN(MTRAV),MCNSQ(MINX(MTRAV)));
    MTRAV=MFAT(MTRAV); GOTO CONROT; END;
  IF MNEI(MTRAV)=0 THEN DO;
   IF MRES(MTRAV)='00'B THEN DO;
    CALL FUNPLAY(MORN(MTRAV),MCNSQ(MINX(MTRAV)));
   MTRAV=MFAT(MTRAV); MRES(MTRAV)='00'B; GOTO CONROT; END;
   IF MMOV(MTRAV)='00'B THEN IF MLBR(MTRAV)¬=0 THEN DO;
     IE=MLBR(MTRAV); MLBR(MTRAV)=0; IEAD: CALL DLT(IE);
    IF MLBR(IE)¬=0 THEN DO; IE=MLBR(IE); GOTO IEAD; END; END;
  CALL FUNPLAY(MORN(MTRAV),MCNSQ(MINX(MTRAV)));
   MTRAV=MFAT(MTRAV); MRES(MTRAV)='01'B; GOTO CONROT; END;
  IF MMOV(MTRAV)='01'B THEN IF MRES(MTRAV)='00'B THEN
```

```
  GOTO CONEXP; ELSE DO;
  IF MLBR(MTRAV)=0 THEN GOTO MIDEL;
    IE=MLBR(MTRAV); MLBR(MTRAV)=0; IEBD: CALL DLT(IE);
   IF MLBR(IE)¬=0 THEN DO; IE=MLBR(IE); GOTO IEBD; END;
 MIDEL: IF MRBR(MTRAV)<=0 THEN GOTO ENDEL;
   IE=MRBR(MTRAV); MRBR(MTRAV)=0; IECD: CALL DLT(IE);
   IF MRBR(IE)¬=0 THEN DO; IE=MRBR(IE); GOTO IECD; END;
  ENDEL: MTRAV=MFAT(MTRAV);
  MRES(MTRAV)='01'B; GOTO CONROT; END;
  IF MRES(MTRAV)='00'B THEN DO;
  MTRAV=MFAT(MTRAV); MRES(MTRAV)='00'B; GOTO CONROT; END;
 CONEXP: IF MDEP(MTRAV)>=MLEN(MTRAV) THEN DO;
   IF MRBR(MTRAV)=0 THEN DO;
  MRES(MFAT(MTRAV))=MRES(MTRAV);
  MTRAV=MFAT(MTRAV); GOTO CONROT; END;
   ELSE DO; MTRAV=MRBR(MTRAV); MSON(MFAT(MTRAV))=MTRAV;
    GOTO CONROT; END; END;
  MWOR=MSLST(MORN(MTRAV)+MDEP(MTRAV));
  IF MAVL(SPONO(MWOR))='0'B THEN DO;
  MDEP(MTRAV)=MDEP(MTRAV)+1; GOTO CONEXP; END;
  MWT=MTRAV; GOTO BCREN;
CRUN:
 IF KTEST>4 THEN
 PUT LIST('CRUN');
  IA=MCNCW; MWT=MTRAV; MWL=MLEV(MTRAV);
LCRUN: IF IA=0 THEN GOTO NDCRUN;
 IF MCNSD(IA,MWL)¬='1'B THEN DO; IA=IA-1; GOTO LCRUN; END;
 CALL NEIGH(MCNSQ(IA));
 MWIND=4975+MJ+MJ;
 IF MSLST(MWIND)=0 THEN DO;
  MCNSD(IA,MWL)='0'B; IA=IA-1; GOTO LCRUN; END;
  IF MNEXZ=0 THEN GOTO NDISHT;
  NODEC=NODEC+1;
  KODEC=KODEC+1;
 MFAT(MNEXZ)=MTRAV;
 MMOV(MNEXZ)=MMOV(MTRAV);
 MLEV(MNEXZ)=MLEV(MTRAV);
 MSON(MTRAV),MTRAV=MNEXZ;
 MNEI(MTRAV)=MJ;
 MORN(MTRAV)=MSLST(MWIND);
 MNEXZ=MNEX(MTRAV);
 MLEN(MTRAV)=MSLST(MWIND+1);
 MRBR(MTRAV),MDEP(MTRAV)=0;
 MINX(MTRAV)=IA;
 MLOC(MTRAV)=MCNS(IA);
 MRES(MTRAV)='11'B;
 MLBR(MTRAV),MSON(MTRAV)=-999;
MCRUN: IA=IA-1; IF IA=0 THEN GOTO NDCRUN;
 IF MCNSD(IA,MWL)¬='1'B THEN GOTO MCRUN;
 CALL NEIGH(MCNSQ(IA));
 MWIND=4975+MJ+MJ;
 IF MSLST(MWIND)=0 THEN DO;
  MCNSD(IA,MWL)='0'B; GOTO MCRUN; END;
 IF MNEXZ=0 THEN GOTO NDISHT;
  NODEC=NODEC+1;
 KODEC=KODEC+1;
 MLBR(MTRAV)=MNEXZ;
 MFAT(MNEXZ)=MFAT(MTRAV);
 MSON(MFAT(MTRAV))=MNEXZ;
 MLEV(MNEXZ)=MLEV(MTRAV);
```

```
   MMOV(MNEXZ)=MMOV(MTRAV);
   MRBR(MNEXZ)=MTRAV;
   MTRAV=MNEXZ;
   MNEXZ=MNEX(MTRAV);
   MNEI(MTRAV)=MJ;
   MORN(MTRAV)=MSLST(MWIND);
   MINX(MTRAV)=IA;
   MLEN(MTRAV)=MSLST(MWIND+1);
   MDEP(MTRAV)=0;
   MLOC(MTRAV)=MCNS(IA);
   MRES(MTRAV)='11'B;
   MLBR(MTRAV),MSON(MTRAV)=-999;
    GOTO MCRUN;
  NDCRUN: IF MWT=MTRAV THEN DO; MSON(MTRAV)=0;
    IF MMOV(MTRAV)='00'B THEN MRES(MTRAV)='01'B; ELSE
    MRES(MTRAV)='00'B;
   END;
   ELSE MLBR(MTRAV)=0;
   GOTO CONROT;
  ACREN:
   IF KTEST>4 THEN
   PUT LIST('CREN');
  CREN: IF MDEP(MTRAV)>=MLEN(MTRAV) THEN DO;
    IF MMOV(MTRAV)='00'B THEN MRES(MTRAV)='01'B;
     ELSE MRES(MTRAV)='00'B;
    MSON(MTRAV)=0; GOTO NDCREN; END;
   MWOR=MSLST(MORN(MTRAV)+MDEP(MTRAV));
   IF MAVL(SPONO(MWOR))='0'B THEN DO;
    MDEP(MTRAV)=MDEP(MTRAV)+1; GOTO CREN; END;
  BCREN:
   IF MNEXZ=0 THEN GOTO NDISHT;
    NODEC=NODEC+1;
    KODEC=KODEC+1;
   MDEP(MTRAV)=MDEP(MTRAV)+1;
   IF MSON(MTRAV)=-999 THEN MLBR(MNEXZ)=0;
    ELSE DO; MLBR(MNEXZ)=MSON(MTRAV); MRBR(MSON(MTRAV))=MNEXZ; END;
   MSON(MTRAV)=MNEXZ;
   MRES(MNEXZ)='10'B;
   IF MMOV(MTRAV)='01'B THEN MMOV(MNEXZ)='00'B;
    ELSE MMOV(MNEXZ)='01'B;
   MNEI(MNEXZ)=0;
   MORN(MNEXZ)=MWOR;
   MLOC(MNEXZ)=MLOC(MTRAV);
   MINX(MNEXZ)=MINX(MTRAV);
   MFAT(MNEXZ)=MTRAV;
    MRBR(MNEXZ),MSON(MNEXZ)=-999;
   MLEV(MNEXZ)=MLEV(MTRAV)+1;
   IF MLEV(MNEXZ)>8 THEN DOFL: DO;
   PUT LIST('DPTH OFL',NODEC); MXF=-1; RETURN; END;
   MTRAV=MNEXZ;
    MNEXZ=MNEX(MTRAV);
   CALL FPLAY(MWOR,MCNSQ(MINX(MTRAV)),MLEV(MTRAV));
  NDCREN:
   GOTO CONROT;
  MMA:
   WRES=MRES(MTRAV);
/*   IF NORN¬=0 THEN NODO: DO;
    MTRAV=MSON(MTRAV);
    ABLQ: IF MRBR(MTRAV)<=0 THEN GOTO ABGR;
    MTRAV=MRBR(MTRAV); GOTO ABLQ;
```

```
   ABGR: IF MLOC(MTRAV)=NLOC THEN GOTO ABFD;
   CALL DLT(MTRAV); MTRAV=MLBR(MTRAV); GOTO ABGR;
   ABFD: IF MLBR(MTRAV)=0 THEN GOTO ABAD;
   II=MTRAV; ABRD: II=MLBR(II); CALL DLT(II);
   IF MLBR(II)¬=0 THEN GOTO ABRD;
   ABAD: IF WRES='01'B THEN GOTO ABMW;
   IF MSON(MTRAV)<0 THEN GOTO ABCR;
   MTRAV=MSON(MTRAV);
   ABLR: IF MLBR(MTRAV)=0 THEN GOTO ABLS;
   MTRAV=MLBR(MTRAV); GOTO ABLR;
   ABLS: IF MORN(MTRAV)=NORN THEN GOTO ABFN;
   CALL DLT(MTRAV); IF MRBR(MTRAV)<0 THEN GOTO ABCR;
   MTRAV=MRBR(MTRAV); GOTO ABLS;
   ABCR: CALL DLT(MTRAV);
   IF MNEXZ=0 THEN GOTO NDISHT;
   MNEI(MNEXZ),MLBR(MNEXZ),MRBR(MNEXZ)=0;
   MRES(MNEXZ)='10'B; MMOV(MNEXZ)='01'B;
   IF MNEI(MTRAV)=0 THEN MLEV(MNEXZ)=MLEV(MTRAV); ELSE
   MLEV(MNEXZ)=MLEV(MTRAV)+1;
    IF MLEV(MNEXZ)>8 THEN GOTO DOFL;
   MCNI=1; ABLP: IF MCNS(MCNI)¬=NLOC THEN DO;
    MCNI=MCNI+1; GOTO ABLP; END;
   MARK,MTRAV=MNEXZ;
   CALL FPLAY(NORN,MCNSQ(MCNI),MLEV(MTRAV));
  NORN,NLOC=0;
  MNEXZ=MNEX(MTRAV); GOTO CONROT;
   ABMW: MTRAV=MSON(MTRAV);
   ABMP: IF MORN(MTRAV)¬= NORN THEN DO;
    MTRAV=MLBR(MTRAV); GOTO ABMP; END;
   ABFN: CALL FPLAY(NORN,MCNSQ(MINX(MTRAV)),MLEV(MTRAV));
   NORN,NLOC=0; END NODO; */
   ABEX: MTRAV=MSON(MTRAV);
  IF MTRAV<=0 THEN DO;
   PUT FILE(EEE) LIST(SPLIST); MXF=-1; GOTO NDMMAX; END;
   IF MRES(MTRAV)='00'B THEN DO;
  MXF=2; GOTO PUKE; END;
/*      DO WHILE(MRBR(MTRAV)¬=0);
    PUT LIST('TRAP 2');
   MTRAV=MRBR(MTRAV); END;
   NLEN=MLEN(MTRAV); NTRAV=MTRAV;
   DO WHILE(MLBR(MTRAV)¬=0);
   MTRAV=MLBR(MTRAV);
   IF MLEN(MTRAV)<NLEN THEN DO;
   NLEN=MLEN(MTRAV); NTRAV=MTRAV; END;
   END; MTRAV=NTRAV; END;
  ABEY: IF MSON(MTRAV)<=0 THEN DO;
   CALL DLT(MTRAV); MTRAV=MLBR(MTRAV); GOTO ABEY; END;
  IF MRES(MTRAV)='00'B THEN DO;
   IF MLBR(MTRAV)>0 THEN DO;
    IK=MLBR(MTRAV); MRBR(IK)=0; CALL DLT(MFAT(IK)); END; END; */
   MTRAV=MSON(MTRAV);
   J=MORN(MTRAV); IX=SPONO(J);
   IY=J-TFOPS(IX)+1; IZ=MLOC(MTRAV);
  CALL FPLAY(J,MCNSQ(MINX(MTRAV)),MLEV(MTRAV));
  MXF=1; PUKE:
  PUT LIST(KODEC,MBTM,MTRAV,NODEC,MXF);
  RETURN;
/*   PUT SKIP(2); CALL UPDT;
  IF MVNDX=0 THEN DO;
   PUT LIST('INV MACH MOVE'); GOTO NDISHT; END;
```

```
  ENEMA: IF MVCNTALL=0 THEN DO; IWIN=1; GOTO NDMMAX; END;
  ENEMO: PUT LIST('YOU TO MOVE'); PUT SKIP(2);
  GET LIST(CARA,NORN,NLOC);
 II=1;
  FLOP: IF II>12 THEN DO; NORN=NORN+100; GOTO FLOQ; END;
   IF CARA=ALPH(II) THEN DO; NORN=NORN-1+TFOPS(II);
   GOTO FLOQ; END; II=II+1; GOTO FLOP; FLOQ:
  IF NORN>9998 THEN STOP; IF NORN>998 THEN DO;
   MVCNTALL,MCNCN=0; MDNDX=6; GOTO NDMMAX; END;
  J=NORN; IZ=NLOC;
  CALL UPDT; IF MVNDX=0 THEN DO;
   PUT LIST('          IT IS STILL'); GOTO ENEMO; END;
  IF MVCNTALL¬=0 THEN
  GOTO MMA; IWIN=0; */
  GOTO NDMMAX;
DLT: PROC(ID);
  IF KTEST>1 THEN
   PUT LIST('DLT',ID);
  IDW=ID;
  MNEX(ID)=0;
  MNEX(MWND),MWND=ID;
  KODEC=KODEC-1;
  IF MSON(ID)<=0 THEN GOTO NDDLT;
 INLP: ID=MSON(ID);
 LP: IF MLBR(ID)¬=0 THEN DO;
   ID=MLBR(ID); GOTO LP; END;
 ILLP:
  IF MSON(ID)<=0 THEN GOTO IMLP;
  GOTO INLP;
 IMLP: IF ID = IDW THEN GOTO NDDLT;
  MNEX(ID)=0;
  MNEX(MWND),MWND=ID;
  KODEC=KODEC-1;
   IF MRBR(ID)<=0 THEN GOTO IOLP;
    ID=MRBR(ID); GOTO ILLP;
  IOLP: IF (MFAT(ID)<=0)|(MFAT(ID)>3000) THEN DO;
   IF KTEST>1 THEN DO;
   PUT LIST((SPLIST(II) DO II=MAX(1,ID-100) TO MIN(3000,ID+100)));
    END; GOTO NDDLT; END;
    ID=MFAT(ID);
   MSON(ID)=0; GOTO IMLP;
 NDDLT: ID=IDW; END DLT;
 NDISHT: MBTM=CPUTIME-WTIM; LOOTT=1;
  PUT LIST ('OVERFLOW',MBTM,NODEC); MXF=-1;
  IF MDNDX=8 THEN DO; PUT SKIP(3); GOTO NDMMAX; END;
/*   MTRAV=MARK; GOTO MMA; */
NDMMAX: END MMAX;
NEIGH: PROCEDURE(JJ);
  MJ=0;
  IF MFSTBD(JJ+1)=0 THEN DO; JL(1)='1'B; IF MFSTBD(JJ+2)=0
                             THEN JL(10)='1'B;
                              IF MFSTBD(JJ-15)=0 THEN JL(5)='1'B;
                              IF MFSTBD(JJ+17)=0 THEN JL(6)='1'B; END;
  IF MFSTBD(JJ-1)=0 THEN DO; JL(3)='1'B; IF MFSTBD(JJ-2)=0
                            THEN JL(12)='1'B;
                              IF MFSTBD(JJ-17)=0 THEN JL(8)='1'B;
                              IF MFSTBD(JJ+15)=0 THEN JL(7)='1'B; END;
  IF MFSTBD(JJ-16)=0 THEN DO; JL(4)='1'B; IF MFSTBD(JJ-32)=0
                         THEN JL(9)='1'B;
                              IF MFSTBD(JJ-15)=0 THEN JL(5)='1'B;
```

```
                                IF MFSTBD(JJ-17)=0 THEN JL(8)='1'B; END;
  IF MFSTBD(JJ+16)=0 THEN DO; JL(2)='1'B; IF MFSTBD(JJ+32)=0
                       THEN JL(11)='1'B;
                                IF MFSTBD(JJ+15)=0 THEN JL(7)='1'B;
                              IF MFSTBD(JJ+17)=0 THEN JL(6)='1'B; END;
 END NEIGH;
KUTRAND: PROC(EMYT,KRNG,KRES);
  DCL
  (EMYT,KRNG,KRES) FIXED BIN(31),
   TRUNC FIXED DEC(2,0);
  IF EMYT>7 THEN EMYT=8; ELSE EMYT=7;
  CALL TIME(EMYT,0,EMAT);
  TRUNC=EMAT;
  KRES=((TRUNC+1)*KRNG)/1000;
  IF KRES<KRNG THEN KRES=KRES+1;
 END KUTRAND;
FPLAY: PROCEDURE(IWO,IWL,MWLV);
  IF KTEST>4 THEN
  PUT LIST('FPLY');
  MAVL(SPONO(IWO))='0'B;
  MFSTBD(IWL)=1;
  DO IH = 1 TO 4; MFSTBD(IWL+MSFF(IWO,IH))=1; END;
  IF MMAXFG=1 THEN DO;
  DO IH=1 TO MCNCW; MCNSD(IH,MWLV)='0'B;
   IF MCNSD(IH,MWLV-1)='1'B THEN
   IF MFSTBD(MCNSQ(IH))=0 THEN MCNSD(IH,MWLV)='1'B; END;
 END FPLAY;
FUNPLAY: PROCEDURE(IWO,IWL);
  IF KTEST>4 THEN
  PUT LIST('FUNP');
  MAVL(SPONO(IWO))='1'B;
  MFSTBD(IWL)=0;
  DO IH=1 TO 4; MFSTBD(IWL+MSFF(IWO,IH))=0; END;
 END FUNPLAY;
 NDBD: END BOARD; IF MDNDX¬=7 THEN GOTO GSTART;
   ELSE GOTO GSTARTA;
 TOO: GET FILE(MHKPNF) LIST((MKA(II) DO II=1 TO 4008)); GOTO FRO;
 SEE: GET FILE(FFF) LIST((MSLST(II) DO II=1 TO 13166)); GOTO SAW;
 REPENT: PUT SKIP LIST('THANK YOU AND BYE-BYE');
  END PENTO;
```

(B) EXCERPTS FROM THE COMPLETE MOVE LISTS

0 :	1	57	34	20	36	46	9	13	16	19
10 :	45	52	18	24	29	42	48	62	6	11
20 :	23	40	47	60	2	3	4	5	7	8
30 :	14	17	21	26	27	28	30	31	32	33
40 :	35	37	39	41	49	50	51	53	54	56
50 :	59	61	63	10	12	15	22	25	38	43
60 :	44	55	58	2	58	31	21	37	43	10
70 :	14	17	20	46	53	15	25	30	39	49
80 :	63	7	12	24	41	48	61	1	3	4
90 :	5	6	8	11	18	22	23	27	28	29
100 :	32	33	34	36	38	40	42	47	50	51
110 :	54	55	56	57	60	62	3	59	32	22
120 :	38	44	9	11	18	21	43	54	8	13
130 :	25	42	49	60	16	26	27	40	50	62
140 :	1	2	4	5	6	7	12	15	19	24
150 :	28	29	30	31	33	34	35	37	39	41
160 :	47	48	51	52	55	57	58	61	63	4
170 :	56	33	19	35	45	10	12	15	22	44
180 :	55	17	23	28	41	47	63	5	14	26
190 :	39	50	61	1	2	3	6	7	8	13
200 :	16	20	25	27	29	30	31	32	34	36
210 :	38	40	42	48	49	51	52	53	58	59
220 :	60	62	5	55	28	26	35	44	10	14
230 :	17	23	45	56	4	15	19	39	47	63
240 :	1	2	6	8	13	16	20	25	27	30
250 :	31	32	34	36	38	40	49	51	52	53
260 :	58	59	60	9	11	21	37	43	46	54
270 :	57	6	52	29	23	36	45	9	11	18
280 :	24	46	57	13	19	34	42	47	60	2
290 :	3	4	7	8	17	21	28	30	32	33
300 :	35	37	41	49	51	53	54	56	59	63
310 :	10	12	15	22	25	38	43	44	55	58
320 :	7	53	30	24	37	46	10	12	15	25
330 :	43	58	2	17	21	41	49	63	3	4
340 :	6	8	11	18	22	23	28	29	32	33
350 :	34	36	38	42	47	51	54	55	56	57
360 :	60	8	54	27	25	38	43	9	13	16
370 :	26	44	59	3	18	22	42	50	62	1
380 :	4	5	7	12	15	19	24	29	30	31
390 :	33	34	35	37	39	48	51	52	55	57
400 :	58	61	10	14	20	36	45	46	53	56
410 :	9	13	16	18	42	62	11	40	60	1
420 :	6	19	24	29	34	47	48	52	57	2
430 :	3	4	5	7	8	21	26	27	28	30
440 :	31	32	33	35	37	39	41	49	50	51
450 :	54	59	61	63	10	15	12	17	41	63
460 :	14	39	61	2	7	20	25	30	31	48
470 :	49	53	58	1	3	4	5	6	8	22
480 :	23	27	28	29	32	33	34	36	38	40
490 :	42	47	50	51	55	56	60	62	11	9
500 :	18	13	42	60	3	8	21	32	49	54
510 :	59	16	26	27	40	50	62	12	10	17
520 :	41	14	61	2	7	20	31	48	53	58
530 :	21	24	37	43	46	1	3	5	6	11
540 :	18	22	23	27	28	29	32	33	36	38
550 :	40	50	51	54	55	56	57	62	9	16
560 :	26	35	44	45	52	59	13	42	60	16
570 :	40	62	1	6	19	29	34	47	48	52
580 :	20	23	36	45	2	3	4	5	7	8

590 :	14	17	26	27	28	30	31	32	33	35
600 :	39	41	49	50	51	53	56	59	61	63
610 :	14	10	17	15	39	63	4	5	23	28
620 :	47	55	56	12	22	33	41	50	61	15
630 :	12	41	63	39	61	2	7	25	30	31
640 :	48	49	58	21	24	37	43	1	3	4
650 :	5	6	8	11	18	22	27	28	29	32
660 :	33	34	38	40	42	47	50	51	54	55
670 :	57	60	62	16	9	18	62	11	40	1
680 :	6	24	29	48	52	57	20	23	36	45
690 :	46	2	3	5	7	14	17	21	26	27
700 :	28	31	32	33	35	37	41	50	51	53
710 :	54	56	59	61	17	10	15	63	2	25
720 :	30	49	53	58	7	12	41	3	4	6
730 :	8	22	23	28	29	32	33	34	36	38
740 :	42	47	51	55	56	60	13	19	35	44
750 :	45	52	59	18	9	13	42	3	8	54
760 :	59	22	25	38	43	44	4	7	12	15
770 :	19	24	29	30	33	34	35	37	51	52
780 :	55	57	58	10	36	45	46	53	56	19
790 :	45	23	47	6	13	34	36	52	60	29
800 :	42	2	3	4	7	8	17	28	30	32
810 :	33	35	41	49	51	53	56	59	63	20
820 :	46	24	48	1	18	29	36	57	62	34
830 :	42	6	11	23	40	47	60	2	3	4
840 :	5	7	8	14	17	21	27	28	30	31
850 :	32	33	37	39	41	49	50	51	53	54
860 :	56	61	63	21	43	25	49	2	15	30
870 :	37	58	63	31	39	1	4	5	6	8
880 :	11	27	28	32	34	38	40	47	51	54
890 :	55	57	60	10	14	17	20	23	36	46
900 :	53	56	22	44	26	50	3	16	27	38
910 :	59	62	32	40	9	11	18	21	43	54
920 :	1	2	5	6	7	12	24	28	29	31
930 :	33	35	37	41	48	51	52	55	57	58
940 :	61	23	45	6	36	52	9	11	46	57
950 :	1	16	20	40	13	19	34	47	60	2
960 :	4	5	8	14	17	21	26	27	28	30
970 :	31	32	35	37	39	49	51	53	54	56
980 :	59	63	24	46	7	37	53	10	12	43
990 :	58	2	17	21	41	3	6	11	18	22
1000 :	23	28	29	32	33	36	38	51	54	55
1010 :	56	57	9	35	44	45	52	59	25	43
1020 :	8	38	54	9	13	44	59	11	21	32
1030 :	49	60	16	26	27	40	1	2	4	5
1040 :	6	15	19	28	30	31	34	35	37	39
1050 :	47	51	52	55	57	58	63	26	44	5
1060 :	35	55	10	14	45	56	4	15	19	39
1070 :	12	22	33	50	61	1	3	7	8	13
1080 :	16	20	25	27	29	30	31	34	36	38
1090 :	42	48	51	52	53	58	59	62	27	38
1100 :	43	54	11	21	32	40	8	25	49	60
1110 :	3	18	22	42	50	62	28	35	44	55
1120 :	12	22	33	41	5	26	50	61	10	14
1130 :	17	23	45	56	1	2	3	6	7	16
1140 :	20	27	29	31	32	36	38	40	48	51
1150 :	52	53	58	59	62	29	36	45	52	13
1160 :	19	34	42	9	18	24	46	57	3	4
1170 :	7	8	30	33	35	37	51	53	54	56
1180 :	59	1	5	14	16	20	26	27	31	39

1190 :	48	50	61	62	30	53	37	46	14	20
1200 :	31	39	7	24	48	61	10	12	15	25
1210 :	43	58	1	3	4	5	8	18	22	27
1220 :	29	33	34	36	38	42	50	51	54	55
1230 :	56	57	62	31	58	37	43	15	25	30
1240 :	39	10	14	20	46	53	1	4	5	8
1250 :	27	34	36	38	51	54	55	56	57	9
1260 :	13	16	19	26	35	44	45	52	59	32
1270 :	59	38	44	16	26	27	40	9	11	21
1280 :	43	54	1	2	5	6	28	31	35	37
1290 :	51	52	55	57	58	10	14	17	20	23
1300 :	36	45	46	53	56	33	56	35	45	17
1310 :	23	28	41	4	19	47	63	5	14	26
1320 :	39	50	61	34	36	46	57	18	24	29
1330 :	42	6	11	23	47	60	2	3	4	7
1340 :	8	17	21	28	30	32	33	37	41	49
1350 :	51	53	54	56	63	35	33	4	19	28
1360 :	41	47	63	12	15	22	44	55	5	26
1370 :	39	50	61	1	2	3	6	7	8	13
1380 :	16	25	27	29	30	31	32	34	38	40
1390 :	42	48	49	51	52	58	59	60	62	36
1400 :	45	52	9	46	57	13	19	34	6	11
1410 :	23	47	60	2	4	8	17	21	28	30
1420 :	32	35	37	49	51	53	54	56	59	63
1430 :	10	15	25	38	43	44	55	58	37	30
1440 :	7	24	31	39	48	61	12	15	25	43
1450 :	58	1	3	4	5	8	18	22	27	29
1460 :	33	34	38	42	50	51	54	55	57	62
1470 :	38	32	3	22	27	40	50	62	11	18
1480 :	21	43	54	1	2	5	6	7	12	24
1490 :	28	29	31	33	37	41	48	51	55	57
1500 :	58	61	39	14	61	10	12	15	4	5
1510 :	22	33	50	55	56	1	3	7	8	20
1520 :	25	27	29	30	31	34	36	38	42	48
1530 :	51	53	58	62	40	16	62	1	6	29
1540 :	48	52	20	23	36	45	2	3	5	7
1550 :	14	17	26	27	28	31	32	33	35	41
1560 :	50	51	53	56	59	61	41	17	63	14
1570 :	39	61	2	7	20	30	31	48	49	53
1580 :	21	24	37	46	42	18	62	11	40	60
1590 :	1	6	24	29	34	47	48	57	2	3
1600 :	4	5	7	8	21	27	28	30	31	32
1610 :	33	37	39	41	49	50	51	54	61	63
1620 :	43	37	58	2	21	10	17	46	53	15
1630 :	25	30	49	63	4	6	8	11	23	28
1640 :	32	34	36	38	47	51	54	55	56	57
1650 :	60	44	38	59	3	22	9	18	43	54
1660 :	16	26	27	50	62	1	5	7	12	24
1670 :	29	31	33	35	37	48	51	52	55	57
1680 :	58	61	10	14	20	36	45	46	53	56
1690 :	45	35	56	4	19	10	15	44	55	17
1700 :	23	28	47	63	2	6	8	13	25	30
1710 :	32	34	36	38	49	51	52	53	58	59
1720 :	60	46	36	57	1	20	9	16	45	52
1730 :	18	24	29	48	62	3	5	7	14	26
1740 :	27	31	33	35	37	50	51	53	54	56
1750 :	59	61	47	23	6	34	36	60	29	42
1760 :	1	20	40	48	62	2	3	4	5	7
1770 :	8	14	17	27	28	30	31	32	33	39
1780 :	41	49	50	51	53	56	61	63	48	20

1790 :	1	29	36	62	34	42	13	16	19	45
1800 :	52	3	4	5	7	8	14	26	27	30
1810 :	31	33	35	39	50	51	53	56	59	61
1820 :	49	25	8	32	38	60	27	40	13	16
1830 :	26	44	59	3	22	42	50	62	50	26
1840 :	5	33	35	61	28	41	14	17	23	45
1850 :	56	51	36	53	56	35	45	52	59	3
1860 :	7	29	33	10	12	22	38	44	55	58
1870 :	2	6	17	23	28	32	41	52	29	9
1880 :	18	24	57	6	11	13	19	34	42	47
1890 :	60	2	3	4	7	8	21	28	30	32
1900 :	33	35	37	41	49	51	54	59	63	12
1910 :	15	22	25	38	43	44	55	58	53	10
1920 :	58	7	12	15	25	30	14	20	31	39
1930 :	48	61	54	27	9	16	26	59	8	13
1940 :	3	18	42	50	62	1	4	5	7	19
1950 :	24	29	30	31	33	34	35	37	39	48
1960 :	51	52	57	61	55	28	10	17	23	56
1970 :	5	14	12	22	33	41	50	61	1	2
1980 :	3	6	7	20	27	29	31	32	36	38
1990 :	40	48	51	53	58	62	56	10	55	4
2000 :	15	12	22	33	17	23	28	41	47	63
2010 :	57	34	9	13	19	52	1	16	6	11
2020 :	40	47	60	2	4	5	8	21	26	27
2030 :	28	30	31	32	35	37	39	49	51	54
2040 :	59	63	58	7	12	2	41	10	17	53
2050 :	3	6	22	23	28	29	32	33	36	38
2060 :	51	55	56	59	9	54	8	13	11	21
2070 :	32	49	60	16	26	27	40	60	13	9
2080 :	11	16	40	61	14	10	12	7	20	31
2090 :	48	53	58	24	37	43	46	1	3	5
2100 :	18	22	27	29	33	36	38	50	51	54
2110 :	55	56	57	62	62	16	13	42	1	19
2120 :	29	34	48	52	9	18	24	57	63	17
2130 :	14	39	2	20	30	31	49	53	21	37
2140 :	46	1	4	5	6	8	11	23	27	28
2150 :	32	34	36	40	47	51	54	56	57	60
2160 :	24	48	7	31	37	61	12	43	58	2
2170 :	21	41	11	60	40	1	6	34	47	57
2180 :	20	23	36	46	11	18	42	60	6	24
2190 :	29	34	47	57	2	3	4	7	8	21
2200 :	28	30	32	33	37	41	49	51	54	63
2210 :	12	15	22	25	38	43	55	58	42	60
2220 :	40	62	1	6	29	34	47	48	2	3
2230 :	4	5	7	8	27	28	30	31	32	33
2240 :	39	41	49	50	51	61	63	13	16	19
2250 :	26	35	52	59	13	16	40	60	1	6
2260 :	19	34	47	52	20	23	36	45	2	4
2270 :	5	8	14	17	26	27	28	30	31	32
2280 :	35	39	49	51	53	56	59	63	18	62
2290 :	11	40	1	6	24	29	48	57	20	23
2300 :	36	46	2	3	5	7	14	17	21	27
2310 :	28	31	32	33	37	41	50	51	53	54
2320 :	56	61	42	62	1	29	34	48	18	24
2330 :	57	3	4	5	7	8	27	30	31	33
2340 :	37	39	50	51	54	61	14	20	36	46
2350 :	53	56	11	40	9	16	1	6	52	57
2360 :	2	5	21	26	27	28	31	32	35	37
2370 :	51	54	59	14	17	20	23	36	45	46
2380 :	53	56	40	60	1	6	34	47	20	23

2390 :	36	2	4	5	8	14	17	27	28	30
2400 :	31	32	39	49	51	53	56	63	10	15
2410 :	25	38	55	58	40	62	1	6	29	48
2420 :	20	23	36	2	3	5	7	14	17	27
2430 :	28	31	32	33	41	50	51	53	56	61
2440 :	19	47	4	28	35	63	15	44	55	5
2450 :	26	39	1	2	6	8	13	16	25	27
2460 :	30	31	32	34	38	40	49	51	52	58
2470 :	59	60	6	29	23	36	11	18	24	46
2480 :	57	2	3	7	17	21	28	32	33	37
2490 :	41	51	53	54	56	9	35	45	52	59
2500 :	1	34	20	36	13	16	19	45	52	4
2510 :	5	8	14	26	27	30	31	35	39	51
2520 :	53	56	59	9	37	46	54	57	21	49
2530 :	2	30	37	63	31	39	7	24	41	48
2540 :	61	15	63	39	4	5	28	47	55	12
2550 :	22	33	41	50	61	1	2	3	6	7
2560 :	8	25	27	29	30	31	32	34	38	40
2570 :	42	48	49	51	58	60	62	12	41	61
2580 :	2	7	31	48	58	1	3	5	6	22
2590 :	27	28	29	32	33	38	40	50	51	55
2600 :	62	16	26	35	44	52	59	41	63	39
2610 :	61	2	7	30	31	48	49	15	39	10
2620 :	14	4	5	55	56	1	8	20	25	27
2630 :	30	31	34	36	38	51	53	58	13	16
2640 :	19	26	35	44	45	52	59	12	61	15
2650 :	39	7	25	30	31	48	58	1	3	4
2660 :	5	8	22	27	29	33	34	38	42	50
2670 :	51	55	62	13	16	19	26	35	44	52
2680 :	59	39	61	7	30	31	48	14	20	53
2690 :	1	3	4	5	8	27	29	33	34	36
2700 :	42	50	51	56	62	14	17	41	61	2
2710 :	7	20	31	48	53	21	24	37	46	39
2720 :	63	2	30	31	49	15	25	58	10	14
2730 :	17	20	53	41	61	2	7	31	48	21
2740 :	24	37	1	3	5	6	11	18	27	28
2750 :	29	32	33	40	50	51	54	57	62	9
2760 :	16	26	35	52	59	7	30	12	15	25
2770 :	58	24	37	43	2	21	41	49	63	31
2780 :	37	30	39	1	4	5	8	27	34	51
2790 :	54	57	14	20	36	46	53	56	2	31
2800 :	21	37	14	17	20	46	53	1	5	6
2810 :	11	23	27	28	32	36	40	51	54	56
2820 :	57	10	38	43	55	58	22	50	3	27
2830 :	38	62	18	43	54	8	25	42	3	32
2840 :	11	18	21	54	9	59	16	26	27	40
2850 :	50	62	8	27	25	38	13	16	26	44
2860 :	59	3	22	42	50	62	4	33	12	15
2870 :	22	55	19	35	44	5	26	39	50	61
2880 :	5	28	26	35	14	17	23	45	56	4
2890 :	19	39	47	63	9	52	57	6	11	13
2900 :	19	34	47	60	2	4	8	21	28	30
2910 :	32	35	37	49	51	54	59	63	15	25
2920 :	38	43	44	55	58	29	34	42	36	3
2930 :	4	7	8	30	33	51	53	56	13	19
2940 :	35	45	52	59	10	12	15	22	25	38
2950 :	44	55	58	6	23	36	11	46	57	34
2960 :	47	60	2	4	8	17	21	28	30	32
2970 :	37	49	51	53	54	56	63	6	11	57
2980 :	1	40	20	23	36	46	1	20	36	16

2990 :	45	52	29	48	62	3	5	7	14	26
3000 :	27	31	33	35	50	51	53	56	59	61
3010 :	10	12	22	38	44	55	58	1	16	52
3020 :	9	57	18	24	29	48	62	3	5	7
3030 :	26	27	31	33	35	37	50	51	54	59
3040 :	61	1	6	40	16	52	20	23	36	45
3050 :	2	5	14	17	26	27	28	31	32	35
3060 :	51	53	56	59	10	38	44	55	58	37
3070 :	46	53	10	43	58	15	25	30	4	8
3080 :	34	36	38	51	54	55	56	57	9	13
3090 :	19	35	44	45	52	59	7	24	37	12
3100 :	43	58	2	21	41	3	6	11	18	22
3110 :	28	29	32	33	38	51	54	55	57	9
3120 :	35	44	52	59	2	17	53	10	58	14
3130 :	20	31	1	5	6	23	27	28	32	36
3140 :	38	40	51	55	56	2	21	37	17	46
3150 :	53	30	49	63	7	24	41	2	7	41
3160 :	17	53	30	49	63	3	4	6	8	23
3170 :	28	29	32	33	34	36	42	47	51	56
3180 :	60	30	31	39	14	20	53	10	15	25
3190 :	58	38	43	54	9	44	59	11	21	32
3200 :	2	6	28	35	37	51	52	55	57	58
3210 :	10	17	23	36	45	46	53	56	8	13
3220 :	59	3	42	32	49	60	22	25	38	44
3230 :	2	4	6	7	12	15	19	28	29	30
3240 :	33	34	35	41	47	51	52	55	58	63
3250 :	8	25	38	13	44	59	3	22	42	4
3260 :	7	12	15	19	29	30	33	34	35	51
3270 :	52	55	58	3	22	38	18	43	54	8
3280 :	25	42	11	21	32	49	60	3	18	54
3290 :	9	59	16	26	27	50	62	3	8	42
3300 :	18	54	27	50	62	11	21	32	40	49
3310 :	60	27	32	40	38	3	8	22	25	42
3320 :	49	50	60	62	35	44	55	10	45	56
3330 :	12	22	33	17	23	28	41	4	19	35
3340 :	15	44	55	5	26	39	1	8	13	16
3350 :	25	27	30	31	34	38	51	52	58	59
3360 :	9	37	43	54	57	4	15	55	5	39
3370 :	12	22	33	50	61	28	33	41	35	4
3380 :	5	19	26	39	47	50	61	63	5	14
3390 :	56	10	55	12	22	33	50	61	26	35
3400 :	44	45	5	26	35	14	45	56	4	19
3410 :	39	33	50	61	4	5	39	14	56	33
3420 :	50	61	17	23	28	41	47	63	13	19
3430 :	34	52	36	45	4	8	30	35	51	53
3440 :	56	59	9	37	46	54	57	18	24	29
3450 :	57	36	46	9	45	52	3	7	33	35
3460 :	37	51	53	54	56	59	10	12	22	38
3470 :	43	44	55	58	6	34	47	60	11	57
3480 :	2	4	8	21	28	30	32	37	49	51
3490 :	54	63	15	25	38	43	55	58	1	29
3500 :	48	62	16	52	3	5	7	26	27	31
3510 :	33	35	50	51	59	61	12	22	38	44
3520 :	55	58	15	25	30	58	10	53	4	8
3530 :	34	36	38	51	55	56	13	19	35	44
3540 :	45	52	59	2	30	49	63	17	53	4
3550 :	6	8	23	28	32	34	36	47	51	56
3560 :	60	13	19	35	45	52	59	14	20	31
3570 :	53	10	58	37	43	46	1	5	27	36
3580 :	38	51	54	55	56	57	9	16	26	35

4790 :	58	1	3	5	7	27	29	31	33	48
4800 :	50	51	61	62	2	6	28	32	40	41
4810 :	3	7	9	18	24	29	33	35	37	51
4820 :	52	54	57	59	2	6	10	17	23	28
4830 :	32	36	38	51	53	55	56	58	2	6
4840 :	11	21	28	32	37	38	43	51	54	55
4850 :	57	58	1	5	27	31	40	1	5	9
4860 :	16	26	27	31	35	37	51	52	54	57
4870 :	59	1	2	5	6	11	21	27	28	31
4880 :	32	37	40	51	54	57	3	7	12	18
4890 :	22	24	29	33	37	38	43	51	54	55
4900 :	57	58	2	6	11	17	21	23	28	32
4910 :	36	37	46	51	53	54	56	57	1	5
4920 :	14	16	20	26	27	31	35	36	45	51
4930 :	52	53	56	59	3	4	7	8	12	15
4940 :	22	25	29	30	33	34	38	42	51	55
4950 :	58	10	36	53	56	1	2	4	5	6
4960 :	8	27	28	30	31	32	34	39	40	47
4970 :	49	51	60	63	0	0	0	0	0	0
4980 :	0	0	0	0	0	0	0	0	0	0
4990 :	0	0	0	0	0	0	0	0	0	0
5000 :	0	0	0	0	0	0	0	0	0	0
5010 :	0	0	0	0	0	0	0	0	0	0
5020 :	0	0	0	0	0	0	0	0	0	0
5030 :	0	0	0	0	0	0	0	0	0	0
5040 :	0	0	0	0	0	0	0	0	0	0
5050 :	0	0	0	0	0	0	0	0	0	0
5060 :	0	0	0	0	0	0	0	0	0	0
5070 :	0	0	0	0	0	0	0	0	0	0
5080 :	0	0	0	0	0	0	0	0	0	0
5090 :	0	0	0	0	0	0	0	0	0	0
5100 :	0	0	0	0	0	0	0	0	0	0
5110 :	0	0	0	0	0	0	0	0	0	0
5120 :	0	0	0	0	0	0	0	0	0	0
5130 :	0	0	0	0	0	0	0	0	0	0
5140 :	0	0	0	0	0	0	0	0	0	0
5150 :	0	0	0	0	0	0	0	0	0	0
5160 :	0	0	0	0	0	0	0	0	0	0
5170 :	0	0	0	0	0	0	0	0	0	0
5180 :	0	0	0	0	0	0	0	0	0	0
5190 :	0	0	0	0	0	0	0	0	0	0
5200 :	0	0	0	0	0	0	0	0	0	0
5210 :	0	0	0	0	0	0	0	0	0	0
5220 :	0	0	0	0	0	0	0	0	0	0
5230 :	0	0	0	0	0	0	0	0	0	0
5240 :	0	0	0	0	0	0	0	0	0	0
5250 :	0	0	0	0	0	0	0	0	0	0
5260 :	0	0	0	0	0	0	0	0	0	0
5270 :	0	0	0	0	0	0	0	0	0	0
5280 :	0	0	0	0	0	0	0	0	0	0
5290 :	0	0	0	0	0	0	0	0	0	0
5300 :	0	0	0	0	0	0	0	0	0	0
5310 :	0	0	0	0	0	0	0	0	0	0
5320 :	0	0	0	0	0	0	0	0	0	0
5330 :	0	0	0	0	0	0	0	0	0	0
5340 :	0	0	0	0	0	0	0	0	0	0
5350 :	0	0	0	0	0	0	0	0	0	0
5360 :	0	0	0	0	0	0	0	0	0	0
5370 :	0	0	0	0	0	0	0	0	0	0
5380 :	0	0	0	0	0	0	0	0	0	0

```
6590 :   983    1   983    2   983    1   983    2   983    1
6600 :   983    2   983    1   983    2     0    0     0    0
6610 :     0    0     0    0     0    0     0    0     0    0
6620 :     0    0   983    1   983    2   983    1   983    2
6630 :   983    1   983    2   983    1   983    2     0    0
6640 :     0    0     0    0     0    0     0    0     0    0
6650 :     0    0     0    0     0    0  1722    1     0    0
6660 :  1722    1     0    0  1722    1     0    0  1722    1
6670 :     0    0     0    0     0    0     0    0     0    0
6680 :     0    0     0    0     0    0     0    0  1722    1
6690 :     0    0  1722    1     0    0  1722    1     0    0
6700 :  1722    1     0    0     0    0     0    0     0    0
6710 :     0    0     0    0     0    0     0    0   983    1
6720 :   983    2   983    1   983    2   983    1   983    2
6730 :   983    1   983    2     0    0     0    0     0    0
6740 :     0    0     0    0     0    0     0    0     0    0
6750 :   983    1   983    2   983    1   983    2   983    1
6760 :   983    2   983    1   983    2     0    0   820    1
6770 :     0    0   820    1     0    0   820    1     0    0
6780 :   820    1     0    0   820    2     0    0   820    2
6790 :     0    0   820    2     0    0   820    2     0    0
6800 :   820    1     0    0   820    1     0    0   820    1
6810 :     0    0   820    1     0    0   820    2     0    0
6820 :   820    2     0    0   820    2     0    0   820    2
6830 :  1789    1  1789    2  1789    1  1789    2  1789    1
6840 :  1789    2  1789    1  1789    2  2161    2   820    4
6850 :  2161    2   820    4  2161    2   820    4  2161    2
6860 :   820    4  1789    1  1789    2  1789    1  1789    2
6870 :  1789    1  1789    2  1789    1  1789    2  2161    2
6880 :   820    4  2161    2   820    4  2161    2   820    4
6890 :  2161    2   820    4     0    0   820    1     0    0
6900 :   820    1     0    0   820    1     0    0   820    1
6910 :     0    0   820    2     0    0   820    2     0    0
6920 :   820    2     0    0   820    2     0    0   820    1
6930 :     0    0   820    1     0    0   820    1     0    0
6940 :   820    1     0    0   820    2     0    0   820    2
6950 :     0    0   820    2     0    0   820    2  1789    1
6960 :  1789    2  1789    1  1789    2  1789    1  1789    2
6970 :  1789    1  1789    2  2161    2   820    4  2161    2
6980 :   820    4  2161    2   820    4  2161    2   820    4
6990 :  1789    1  1789    2  1789    1  1789    2  1789    1
7000 :  1789    2  1789    1  1789    2  2161    2   820    4
7010 :  2161    2   820    4  2161    2   820    4  2161    2
7020 :   820    4     0    0     0    0     0    0     0    0
7030 :     0    0     0    0     0    0     0    0     0    0
7040 :     0    0     0    0     0    0     0    0     0    0
7050 :     0    0     0    0     0    0     0    0     0    0
7060 :     0    0     0    0     0    0     0    0     0    0
7070 :     0    0     0    0     0    0     0    0     0    0
7080 :     0    0     0    0     0    0     0    0     0    0
7090 :     0    0     0    0     0    0     0    0     0    0
7100 :     0    0     0    0     0    0     0    0     0    0
7110 :     0    0     0    0     0    0     0    0     0    0
7120 :     0    0     0    0     0    0     0    0     0    0
7130 :     0    0     0    0     0    0     0    0     0    0
7140 :     0    0     0    0     0    0     0    0     0    0
7150 :     0    0     0    0     0    0     0    0     0    0
7160 :     0    0     0    0     0    0     0    0     0    0
7170 :     0    0     0    0     0    0     0    0     0    0
7180 :     0    0     0    0     0    0     0    0     0    0
```

7190 :	0	0	0	0	0	0	0	0	0	0
7200 :	0	0	0	0	0	0	0	0	0	0
7210 :	0	0	0	0	0	0	0	0	0	0
7220 :	0	0	0	0	0	0	0	0	0	0
7230 :	0	0	0	0	0	0	0	0	0	0
7240 :	0	0	0	0	0	0	0	0	0	0
7250 :	0	0	0	0	0	0	0	0	0	0
7260 :	0	0	0	0	0	0	0	0	0	0
7270 :	0	0	0	0	0	0	0	0	0	0
7280 :	0	0	0	0	0	0	0	0	0	0
7290 :	0	0	0	0	0	0	0	0	0	0
7300 :	0	0	0	0	0	0	0	0	0	0
7310 :	0	0	0	0	0	0	0	0	0	0
7320 :	0	0	0	0	0	0	0	0	0	0
7330 :	0	0	0	0	0	0	0	0	0	0
7340 :	0	0	0	0	0	0	0	0	0	0
7350 :	0	0	0	0	0	0	0	0	0	0
7360 :	0	0	0	0	0	0	0	0	0	0
7370 :	0	0	0	0	0	0	0	0	0	0
7380 :	0	0	0	0	0	0	0	0	0	0
7390 :	0	0	0	0	0	0	0	0	0	0
7400 :	0	0	0	0	0	0	0	0	0	0
7410 :	0	0	0	0	0	0	0	0	0	0
7420 :	0	0	0	0	0	0	0	0	0	0
7430 :	0	0	0	0	0	0	0	0	0	0
7440 :	0	0	0	0	0	0	0	0	0	0
7450 :	0	0	0	0	0	0	0	0	0	0
7460 :	0	0	0	0	0	0	0	0	0	0
7470 :	0	0	0	0	0	0	0	0	0	0
7480 :	0	0	0	0	0	0	0	0	0	0
7490 :	0	0	0	0	0	0	0	0	0	0
7500 :	0	0	0	0	0	0	0	0	0	0
7510 :	0	0	0	0	0	0	0	0	0	0
7520 :	0	0	0	0	0	0	0	0	0	0
7530 :	0	0	0	0	0	0	0	0	0	0
7540 :	0	0	0	0	0	0	0	0	0	0
7550 :	0	0	0	0	411	1	411	1	0	0
7560 :	0	0	411	1	411	1	0	0	0	0
7570 :	567	1	567	1	0	0	0	0	567	1
7580 :	567	1	0	0	0	0	411	2	411	2
7590 :	0	0	0	0	411	2	411	2	0	0
7600 :	0	0	0	0	0	0	0	0	0	0
7610 :	0	0	0	0	754	1	754	1	754	2
7620 :	754	2	754	1	754	1	754	2	754	2
7630 :	1585	1	1585	1	567	2	567	2	1585	1
7640 :	1585	1	567	2	567	2	1585	2	1585	2
7650 :	754	4	754	4	1585	2	1585	2	754	4
7660 :	754	4	0	0	0	0	0	0	0	0
7670 :	0	0	0	0	0	0	0	0	499	1
7680 :	499	1	499	2	499	2	499	1	499	1
7690 :	499	2	499	2	2078	1	2078	1	2078	2
7700 :	2078	2	2078	1	2078	1	2078	2	2078	2
7710 :	2173	2	2173	2	2078	4	2078	4	2173	2
7720 :	2173	2	2078	4	2078	4	0	0	0	0
7730 :	0	0	0	0	0	0	0	0	0	0
7740 :	0	0	2185	2	2185	2	499	3	499	3
7750 :	2185	2	2185	2	499	3	499	3	2219	2
7760 :	2219	2	567	3	567	3	2219	2	2219	2
7770 :	567	3	567	3	2185	4	2185	4	499	6
7780 :	499	6	2185	4	2185	4	499	6	499	6

11390 :	0	0	0	0	1058	1	1058	1	0	0
11400 :	0	0	1058	2	1058	2	0	0	0	0
11410 :	1058	1	1058	1	0	0	0	0	1058	2
11420 :	1058	2	0	0	0	0	1058	1	1058	1
11430 :	0	0	0	0	1058	2	1058	2	1839	1
11440 :	1839	1	1839	2	1839	2	2827	2	2827	2
11450 :	903	4	903	4	1839	1	1839	1	1839	2
11460 :	1839	2	2827	2	2827	2	903	4	903	4
11470 :	1839	1	1839	1	1839	2	1839	2	2827	2
11480 :	2827	2	903	4	903	4	1839	1	1839	1
11490 :	1839	2	1839	2	2827	2	2827	2	903	4
11500 :	903	4	0	0	0	0	1058	1	1058	1
11510 :	0	0	0	0	1058	2	1058	2	0	0
11520 :	0	0	1058	1	1058	1	0	0	0	0
11530 :	1058	2	1058	2	0	0	0	0	1058	1
11540 :	1058	1	0	0	0	0	1058	2	1058	2
11550 :	0	0	0	0	1058	1	1058	1	0	0
11560 :	0	0	1058	2	1058	2	1839	1	1839	1
11570 :	1839	2	1839	2	2827	2	2827	2	903	4
11580 :	903	4	1839	1	1839	1	1839	2	1839	2
11590 :	2827	2	2827	2	903	4	903	4	1839	1
11600 :	1839	1	1839	2	1839	2	2827	2	2827	2
11610 :	903	4	903	4	1839	1	1839	1	1839	2
11620 :	1839	2	2827	2	2827	2	903	4	903	4
11630 :	0	0	0	0	2064	1	2064	1	1471	1
11640 :	1471	1	1652	3	1652	3	1933	1	1933	1
11650 :	2064	3	2064	3	3192	3	3192	3	3192	6
11660 :	3192	6	362	1	362	1	3219	3	3219	3
11670 :	3251	3	3251	3	3251	6	3251	6	362	2
11680 :	362	2	2064	5	2064	5	1019	5	1019	5
11690 :	1019	9	1019	9	117	1	117	1	117	2
11700 :	117	2	3274	3	3274	3	1652	5	1652	5
11710 :	3288	3	3288	3	3288	5	3288	5	3274	6
11720 :	3274	6	1652	9	1652	9	3298	3	3298	3
11730 :	3219	5	3219	5	4107	6	4107	6	3251	9
11740 :	3251	9	3298	5	3298	5	754	8	754	8
11750 :	3274	9	3274	9	754	13	754	13	1270	1
11760 :	1270	1	1270	2	1270	2	1471	2	1471	2
11770 :	1270	4	1270	4	3619	4	3619	4	3619	6
11780 :	3619	6	4116	6	4116	6	3192	9	3192	9
11790 :	3629	4	3629	4	3629	6	3629	6	1821	6
11800 :	1821	6	3629	9	3629	9	4415	7	4415	7
11810 :	2064	10	2064	10	4415	10	4415	10	1019	14
11820 :	1019	14	2839	2	2839	2	117	3	117	3
11830 :	1471	4	1471	4	117	6	117	6	2839	6
11840 :	2839	6	2839	8	2839	8	4116	9	4116	9
11850 :	117	12	117	12	4125	6	4125	6	3219	8
11860 :	3219	8	4107	9	4107	9	3219	12	3219	12
11870 :	4125	10	4125	10	499	13	499	13	3274	14
11880 :	3274	14	117	18	117	18	1099	1	1099	1
11890 :	3653	4	3653	4	1099	2	1099	2	3653	6
11900 :	3653	6	1933	2	1933	2	1933	6	1933	6
11910 :	1099	4	1099	4	3653	9	3653	9	2853	2
11920 :	2853	2	4135	6	4135	6	2853	4	2853	4
11930 :	2853	9	2853	9	362	3	362	3	1933	8
11940 :	1933	8	362	6	362	6	362	12	362	12
11950 :	3672	4	3672	4	4425	7	4425	7	2827	6
11960 :	2827	6	903	10	903	10	3672	6	3672	6
11970 :	3288	10	3288	10	2827	9	2827	9	1652	14
11980 :	1652	14	4149	6	4149	6	4135	10	4135	10

11990 :	4149	9	4149	9	2853	14	2853	14	3298	8
12000 :	3298	8	1933	13	1933	13	2827	12	2827	12
12010 :	362	18	362	18	3312	3	3312	3	4158	6
12020 :	4158	6	3312	4	3312	4	1270	8	1270	8
12030 :	4182	6	4182	6	3619	10	3619	10	1099	8
12040 :	1099	8	1270	13	1270	13	4206	6	4206	6
12050 :	4206	10	4206	10	1821	8	1821	8	1821	13
12060 :	1821	13	4182	9	4182	9	2064	14	2064	14
12070 :	1099	12	1099	12	1019	18	1019	18	4232	6
12080 :	4232	6	4158	9	4158	9	1471	8	1471	8
12090 :	903	12	903	12	3672	10	3672	10	2839	14
12100 :	2839	14	1471	13	1471	13	903	18	903	18
12110 :	4232	10	4232	10	4135	14	4135	14	3312	13
12120 :	3312	13	1821	18	1821	18	3298	14	3298	14
12130 :	499	19	499	19	1099	18	1099	18	117	24
12140 :	117	24	0	0	1997	1	1356	1	1691	3
12150 :	1965	1	1997	3	3325	3	3325	6	0	0
12160 :	1997	1	1356	1	1691	3	1965	1	1997	3
12170 :	3325	3	3325	6	170	1	170	2	3338	3
12180 :	1691	5	3366	3	1997	5	3338	6	1691	9
12190 :	170	1	170	2	3338	3	1691	5	3366	3
12200 :	1997	5	3338	6	1691	9	1306	1	1306	2
12210 :	1356	2	1306	4	3682	4	3682	6	4242	6
12220 :	3325	9	1306	1	1306	2	1356	2	1306	4
12230 :	3682	4	3682	6	4242	6	3325	9	2867	2
12240 :	170	3	1356	4	170	6	2867	6	1997	8
12250 :	2867	9	170	12	2867	2	170	3	1356	4
12260 :	170	6	2867	6	1997	8	2867	9	170	12
12270 :	1117	1	3692	4	1117	2	3692	6	1965	2
12280 :	1965	6	1117	4	3692	9	1117	1	3692	4
12290 :	1117	2	3692	6	1965	2	1965	6	1117	4
12300 :	3692	9	3711	4	4432	7	2441	6	4432	10
12310 :	3711	6	3711	10	2441	9	1691	14	3711	4
12320 :	4432	7	2441	6	4432	10	3711	6	3711	10
12330 :	2441	9	1691	14	3376	3	4252	6	3376	4
12340 :	1306	8	4261	6	3682	10	1117	8	3325	13
12350 :	3376	3	4252	6	3376	4	1306	8	4261	6
12360 :	3682	10	1117	8	3325	13	4270	6	4252	9
12370 :	1356	8	1306	12	4270	10	1997	14	1356	13
12380 :	170	18	4270	6	4252	9	1356	8	1306	12
12390 :	4270	10	1997	14	1356	13	170	18	223	1
12400 :	3389	3	3403	3	3403	6	223	2	3389	5
12410 :	1058	5	1058	9	223	1	3389	3	3403	3
12420 :	3403	6	223	2	3389	5	1058	5	1058	9
12430 :	3415	3	3415	5	4280	6	3403	9	3366	5
12440 :	2618	8	3338	9	1058	13	3415	3	3415	5
12450 :	4280	6	3403	9	3366	5	2618	8	3338	9
12460 :	1058	13	3735	4	3735	6	1839	6	3735	9
12470 :	4442	7	3389	10	4242	10	3389	14	3735	4
12480 :	3735	6	1839	6	3735	9	4442	7	3389	10
12490 :	4242	10	3389	14	4306	6	3415	8	4280	9
12500 :	3403	12	3366	10	1503	13	2867	14	1058	18
12510 :	4306	6	3415	8	4280	9	3403	12	3366	10
12520 :	1503	13	2867	14	1058	18	2881	2	4316	6
12530 :	2881	4	2881	9	223	3	1965	8	223	6
12540 :	223	12	2881	2	4316	6	2881	4	2881	9
12550 :	223	3	1965	8	223	6	223	12	4326	6
12560 :	4316	10	4326	9	2881	14	2542	8	611	13
12570 :	2441	12	223	18	4326	6	4316	10	4326	9
12580 :	2881	14	2542	8	611	13	2441	12	223	18

12590 :	4335	6	4335	10	1839	8	1839	13	4261	9
12600 :	1965	14	1117	12	1117	18	4335	6	4335	10
12610 :	1839	8	1839	13	4261	9	1965	14	1117	12
12620 :	1117	18	4306	10	3415	14	3376	13	1306	18
12630 :	2542	14	611	19	1356	18	170	24	4306	10
12640 :	3415	14	3376	13	1306	18	2542	14	611	19
12650 :	1356	18	170	24	1852	1	1852	4	3744	4
12660 :	1852	8	3763	4	3763	8	3744	8	3763	13
12670 :	3782	4	3782	8	4449	8	4449	13	4470	8
12680 :	4470	13	4761	13	3763	19	3810	5	3810	8
12690 :	4566	10	3429	14	4581	10	3523	14	4581	16
12700 :	3523	21	4491	8	3782	12	4491	14	3429	19
12710 :	4014	14	3070	19	4014	21	3070	27	3825	5
12720 :	3825	8	4505	8	1852	12	4597	10	3825	14
12730 :	4505	14	1852	19	4612	10	4612	14	4811	14
12740 :	3448	19	4887	16	3825	21	4505	21	3448	27
12750 :	4626	10	2926	13	4566	15	2926	19	4935	17
12760 :	4935	21	3251	23	2926	28	4626	15	3782	19
12770 :	3908	21	1156	26	4626	23	3782	28	754	30
12780 :	754	36	3846	5	4649	10	3846	8	4649	14
12790 :	4526	8	4825	14	3744	12	3692	19	4663	10
12800 :	4903	16	4663	14	4449	21	4839	14	4470	21
12810 :	3192	19	3192	27	4774	13	3544	18	4345	18
12820 :	3544	24	4774	18	3711	24	3629	24	1691	31
12830 :	3475	18	2954	24	2895	24	1400	31	3475	24
12840 :	1621	31	2895	31	1400	39	4677	10	4677	15
12850 :	3846	13	3846	19	4597	15	2043	21	3744	19
12860 :	1852	26	3929	17	2473	23	3929	21	2473	28
12870 :	3097	23	983	30	3097	28	983	36	3950	19
12880 :	3158	24	3884	24	790	30	3950	26	715	32
12890 :	3219	32	715	39	2185	26	1324	32	1878	32
12900 :	272	39	2185	34	321	41	1878	41	272	49
12910 :	3865	5	4692	10	4713	10	4919	16	3865	8
12920 :	4692	14	4713	14	4692	21	4545	8	4545	14
12930 :	4858	14	4545	21	3865	12	3568	19	3653	19
12940 :	3568	27	4727	10	3810	15	3976	17	2501	23
12950 :	4077	15	2618	21	3338	23	2618	30	2780	13
12960 :	2780	19	3976	21	2501	28	3865	19	1234	26
12970 :	3338	28	1234	36	4792	13	4397	18	3499	18
12980 :	2987	24	3595	18	3595	24	3499	24	2987	31
12990 :	4363	18	4053	24	3018	24	1722	31	4363	24
13000 :	2084	31	1652	31	1652	39	4727	19	2682	24
13010 :	4280	26	1789	32	2648	26	1503	32	2648	34
13020 :	1058	41	2323	24	2323	30	1933	32	1156	39
13030 :	1439	32	1195	39	362	41	362	49	4746	10
13040 :	3997	17	4746	15	3042	23	4526	13	3125	21
13050 :	4526	19	3042	28	4872	15	2799	23	2353	21
13060 :	2353	30	4839	19	2799	28	1270	26	1270	36
13070 :	4956	19	2383	26	4206	26	2255	34	4077	24
13080 :	2383	32	2441	32	223	41	4182	24	2129	32
13090 :	2011	32	942	41	864	30	864	39	1019	39
13100 :	223	49	4792	19	2415	26	4158	24	1535	32
13110 :	2578	24	1965	32	2578	30	1117	39	2734	26
13120 :	2289	34	2734	32	674	41	1471	32	518	41
13130 :	903	39	518	49	2219	29	1753	36	2219	36
13140 :	567	44	2542	36	455	44	1356	44	170	53
13150 :	1585	36	820	44	411	44	1	53	630	44
13160 :	64	53	117	53	1	63	0	0	0	0

(C) ROUTINE THAT BUILDS AND UPDATES THE MOVE EVALUATION TABLES

```
      IMPLICIT INTEGER*2 (A-Z)
      DIMENSION A(4020),E(36,478)
      DO 20 I=1,36
      DO 20 J=1,478
   20 E(I,J)=0
      DO 21 I=1,4008,12
      K=I+11
   21 READ (4,1212,END=90) (A(J), J=I,K)
   90 I=1
   50 I=I+1
      IF (I-473) 70,70,10
   70 J=A(I)
      IF (J.LE.0) GO TO 50
   80 Q=A(J)-9
      R=I+2
      E(Q,R)=A(J+1)
      J=J+2
      IF (A(J)) 50,50,80
   10 READ (5,14,END=1000) B,C,D
   14 FORMAT (3I4)
      IF (B.LT.49) GO TO 1000
      IF (B.GT.520) GO TO 1000
      IF (C.LT.10) GO TO 1000
      IF (C.GT.46) GO TO 1000
      G=C-9
      B=B-45
      IF (D-2) 100,200,300
  300 E(G,B)=E(G,B)-2
      E(G,B-1)=E(G,B-1)-1
      E(G,B+1)=E(G,B+1)-1
      GO TO 10
  200 E(G,B-3)=E(G,B-3)+1
      E(G,B-2)=E(G,B-2)+2
      E(G,B-1)=E(G,B-1)+3
      E(G,B+1)=E(G,B+1)+3
      E(G,B+2)=E(G,B+2)+2
      E(G,B+3)=E(G,B+3)+1
      E(G,B)=E(G,B)+4
      GO TO 10
  100 E(G,B-3)=E(G,B-3)-1
      E(G,B-2)=E(G,B-2)-2
      E(G,B-1)=E(G,B-1)-4
      E(G,B+1)=E(G,B+1)-4
      E(G,B+2)=E(G,B+2)-2
      E(G,B+3)=E(G,B+3)-1
      E(G,B)=E(G,B)-8
      GO TO 10
 1000 I=4
      J=2
      K=474
      A(2)=0
      L=0
 1010 L=L+1
      IF (L.GT.36) GO TO 1100
      IF (E(L,I)) 1015,1010,1015
 1015 IF (A(J)) 1030,1020,1030
 1020 A(J)=K
 1030 A(K)=L+9
      A(K+1)=E(L,I)
```

```
      K=K+2
      GO TO 1010
1100  IF (A(K-1)) 1110,1120,1110
1110  A(K)=0
      K=K+1
1120  IF (J.GT.472) GO TO 1200
      I=I+1
      J=J+1
      L=0
      A(J)=0
      GO TO 1010
1200  A(1)=K
1210  WRITE (6,1212)  A
1212  FORMAT (12I6)
      WRITE (7,1215) A(1)
1215  FORMAT (16H ARRAY SIZE NOW ,I4)
      STOP
      END
```

(D) THE MOVE EVALUATION LISTS

1	3436	474	485	496	511	526	539	552	569	586
2	603	616	627	638	649	664	683	700	713	724
3	735	744	753	764	773	780	787	798	811	826
4	839	852	865	880	891	902	909	916	923	930
5	937	946	957	968	981	994	1007	1020	1035	1050
6	1065	1084	1105	1124	1141	1154	1167	1176	1185	1192
7	1201	1212	1225	1236	1247	1258	1269	1280	1289	1296
8	1303	1312	1321	1330	1337	1344	1353	1362	1371	1378
9	1385	1394	1405	1416	1423	1428	1437	1444	1451	1456
10	1461	1470	1479	1484	1487	1490	1495	1504	1515	1526
11	1537	1548	1561	1572	1585	1594	1605	1616	1625	1630
12	1637	1644	1649	1654	1663	1672	1681	1694	1707	1718
13	1727	1736	1747	1760	1773	1786	1797	1808	1819	1828
14	1835	1842	1849	1860	1871	1882	1893	1904	1913	1922
15	1931	1942	1957	1974	1989	2004	2015	2026	2037	2046
16	2053	2060	2065	2068	2073	2076	2079	2082	2085	2088
17	2093	2100	2107	2114	2119	2124	2129	2138	2147	2156
18	2165	2174	2183	2192	2197	2200	2203	0	2206	2209
19	2216	2223	2230	2239	2248	2259	2270	2279	2286	2293
20	2302	2311	2320	2327	2334	2341	2348	2353	2358	2363
21	2368	2375	2380	2383	2386	2391	2396	2401	2404	2409
22	2414	2419	2428	2437	2448	2461	2472	2483	2492	2497
23	0	2500	2505	2510	2515	2520	2525	2530	0	2535
24	2538	2545	2552	2559	2566	2577	2590	2601	2610	2619
25	2630	2639	2644	2649	2654	2659	2664	0	2667	2672
26	2677	2682	2687	2692	2699	2706	2715	2726	2735	2746
27	2753	2762	2771	2776	2781	0	0	2786	2789	2792
28	0	2795	2798	2801	2806	2811	2816	2819	0	0
29	2822	2825	2828	2833	2838	2845	2854	2863	2870	2879
30	2886	2893	2896	2901	2908	2915	2918	2921	2924	2929
31	2936	2943	2950	2955	2958	0	0	0	0	0
32	0	2961	2964	2969	2974	2983	2990	2997	3002	3005
33	3008	0	0	0	0	0	3011	3014	3019	3024
34	3029	3032	0	0	0	0	3035	3038	3043	3048
35	3053	3056	3059	3062	3065	3068	3071	3074	3077	3080
36	3083	3086	3089	0	0	0	3092	3095	3102	3109
37	0	3114	3117	3122	3127	0	0	3130	3133	3136
38	0	0	0	0	0	0	3139	3142	3145	0
39	3148	3151	3156	3161	3166	3171	0	0	0	0
40	0	0	0	0	0	0	0	0	3176	3179
41	3182	3185	3190	3197	3204	0	0	0	0	3209
42	3214	3219	3226	3229	3232	3237	3242	3249	3256	3263
43	3268	0	0	3273	3276	3279	3282	3285	3288	0
44	0	0	0	0	3291	3294	3297	0	0	0
45	3300	3303	3306	0	0	3309	3314	3319	3324	3329
46	3336	3343	3352	3355	3358	3361	3366	3373	3380	3385
47	3390	3395	0	0	0	3400	3403	3406	3409	3414
48	3419	3424	3431	12	-1	14	-12	15	2	16
49	-4	18	-8	0	12	-2	14	-12	15	1
50	16	-8	18	-4	0	10	-1	12	-5	14
51	-6	15	-3	16	-5	17	-2	18	-2	0
52	10	-2	12	-10	14	-4	15	-10	16	-4
53	17	-4	18	-1	0	10	-4	12	-8	14
54	-3	15	-11	16	-5	17	-8	0	10	-8
55	12	-10	14	-4	15	-6	16	-8	17	-16
56	0	10	-5	12	-5	13	-1	14	-8	15
57	-3	16	-4	17	-8	19	-1	0	10	-4
58	12	-2	13	-2	14	-4	15	-1	16	-2
59	17	-4	19	-2	0	10	-5	12	-1	13

60	-4	14	-2	16	-1	17	-2	19	-4	20
61	-1	0	10	-8	13	-8	14	-1	18	-1
62	19	-9	20	-2	0	10	-4	13	-4	18
63	-2	19	-6	20	-4	0	10	-3	13	-2
64	18	-4	19	-6	20	-8	0	10	-3	13
65	-2	18	-8	19	-9	20	-4	0	10	-4
66	13	-2	14	1	16	2	18	-4	19	-4
67	20	-2	0	10	-8	13	-4	14	2	16
68	4	17	1	18	-2	19	-2	20	-1	22
69	1	0	10	-4	13	-8	14	3	16	6
70	17	1	18	-1	19	-1	22	2	0	10
71	-2	13	-4	14	4	16	8	17	1	22
72	4	0	10	-1	13	-2	14	3	16	6
73	22	6	0	13	-1	14	2	16	4	17
74	-5	22	6	0	14	1	16	2	17	-2
75	22	6	0	16	-1	17	-1	20	1	22
76	4	0	10	-1	16	-2	17	-1	20	2
77	22	2	0	10	-2	16	-5	20	3	22
78	1	0	10	-4	16	-10	20	4	0	10
79	-8	16	-8	20	3	0	10	-4	15	-1
80	16	-10	20	2	21	1	0	10	-2	15
81	-2	16	-5	17	-1	20	1	21	2	0
82	10	-1	13	-1	14	-1	15	-4	16	-2
83	17	-2	21	3	0	13	-2	14	-2	15
84	-8	16	-1	17	-4	21	4	0	13	-4
85	14	-4	15	-4	16	-1	17	-8	21	3
86	0	13	-8	14	-8	15	-2	16	-2	17
87	-4	21	2	0	10	-1	13	-4	14	-4
88	15	-1	16	-4	17	-2	21	1	0	10
89	-2	13	-2	14	-2	16	-8	17	-1	0
90	10	-4	13	-1	14	-1	15	-2	16	-4
91	0	10	-8	15	-4	16	-3	0	10	-4
92	15	-8	16	-3	0	10	-2	15	-16	16
93	-4	0	10	-1	15	-9	16	-8	0	15
94	-6	16	-4	23	-2	0	13	-1	15	-7
95	16	-3	23	-4	0	13	-2	15	-10	16
96	-3	23	-8	25	1	0	13	-4	15	-8
97	16	-4	23	-16	25	2	0	12	-1	13
98	-8	15	-10	16	-8	23	-8	25	3	0
99	12	-2	13	-4	15	-5	16	-4	23	-4
100	25	4	0	12	-4	13	-2	15	-3	16
101	-2	23	-2	25	3	0	12	-8	13	-1
102	15	-3	16	-1	25	2	31	-1	0	10
103	-1	12	-4	15	-4	23	1	24	1	25
104	1	31	-2	0	10	-2	12	-2	15	-8
105	22	1	23	2	24	2	31	-4	0	10
106	-4	12	-1	15	-4	22	1	23	3	24
107	4	31	-8	0	10	-8	15	-2	18	-1
108	20	-1	21	-1	22	-1	23	4	24	6
109	31	-4	0	10	-4	13	-1	15	-1	18
110	-2	20	-2	21	-2	22	4	23	3	24
111	6	31	-2	0	10	-2	13	-2	18	-4
112	20	-3	21	-4	22	5	23	2	24	6
113	31	-1	0	10	-1	13	-4	18	-8	20
114	-6	21	-8	22	7	23	1	24	4	0
115	13	-8	18	-4	20	-1	21	-4	22	9
116	24	2	0	13	-4	18	-2	20	1	21
117	-2	22	6	24	1	0	13	-2	18	-1
118	21	-1	22	4	0	13	-1	20	-2	21
119	1	22	2	0	20	-7	21	2	22	1

120	0	20	-4	21	3	22	2	23	-1	0
121	20	-2	21	4	22	3	23	-2	26	-1
122	0	12	1	20	-1	21	3	22	4	23
123	-4	26	-2	0	12	2	21	1	22	3
124	23	-8	26	-1	0	12	3	15	-1	21
125	-1	22	2	23	-3	0	12	4	15	-2
126	20	-1	21	-4	22	1	0	12	3	15
127	-4	20	-4	21	-8	23	2	0	12	2
128	15	-8	20	-1	21	-4	23	4	0	12
129	1	15	-5	21	-2	23	3	0	15	-4
130	21	-1	23	2	0	15	-5	17	-1	23
131	1	0	14	-2	15	-8	17	-2	22	-1
132	0	14	-4	15	-4	17	-4	22	-4	0
133	14	-8	15	-3	17	-8	22	-1	0	14
134	-16	15	-3	17	-4	0	14	-8	15	-4
135	17	-2	0	14	-4	15	-8	17	-1	21
136	-1	0	14	-2	15	-4	21	-2	23	1
137	0	15	-2	16	-1	21	-1	23	2	0
138	15	-1	16	-2	23	3	0	16	-4	19
139	-1	23	4	0	16	-8	19	-2	23	3
140	32	-1	0	16	-4	17	-1	19	-4	23
141	2	32	-2	0	16	-2	17	-2	19	-8
142	23	1	32	-1	0	16	-1	17	-4	19
143	-4	0	17	-8	19	-2	0	17	-4	19
144	-1	20	1	27	1	0	17	-2	20	2
145	27	2	0	17	-1	20	3	27	3	0
146	20	4	27	4	0	20	3	27	3	0
147	18	-1	20	2	27	2	28	-1	0	18
148	-2	20	1	27	1	28	-2	0	18	-4
149	28	-1	0	18	-8	0	18	-4	0	13
150	-1	18	-2	0	13	-2	18	-1	21	1
151	26	-1	0	13	-4	21	2	24	1	26
152	-2	32	1	0	13	-8	21	3	24	2
153	26	-1	32	2	0	13	-4	19	-1	21
154	4	24	3	32	3	0	13	-2	19	-2
155	21	3	24	3	32	4	0	13	-1	19
156	-4	21	1	24	1	28	1	32	3	0
157	19	-8	21	-1	24	-2	28	2	32	2
158	0	19	-3	20	1	21	-4	24	-7	28
159	3	32	1	0	20	2	21	-8	24	-4
160	28	4	0	19	2	20	3	21	-4	24
161	-2	28	3	0	19	4	20	4	21	-2
162	24	-1	28	2	0	19	3	20	2	21
163	-1	28	1	0	19	2	25	-1	0	19
164	1	20	-3	25	-2	0	20	-8	25	-1
165	27	1	0	20	-3	27	2	0	27	3
166	30	2	0	20	2	23	-1	27	4	30
167	4	0	20	4	23	-2	27	3	30	6
168	0	20	3	23	-4	27	2	30	8	0
169	19	-1	20	2	22	-1	23	-8	27	1
170	30	6	0	19	-2	20	1	22	-2	23
171	-5	30	4	31	1	0	19	-4	22	-4
172	23	-5	30	2	31	2	0	19	-8	22
173	-8	23	-7	31	3	0	19	-4	22	-4
174	23	-12	31	4	0	19	-2	22	-2	23
175	-12	29	1	31	3	0	19	-1	22	-1
176	23	-6	25	1	29	2	31	3	0	23
177	-3	24	1	25	2	27	1	29	3	31
178	3	0	23	-1	24	2	25	4	27	2
179	29	4	31	3	0	24	3	25	6	27

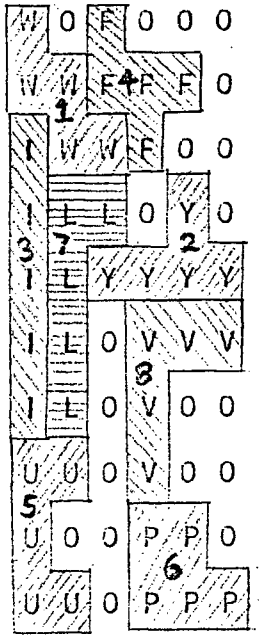
180	3	29	3	31	4	0	24	4	25	6
181	27	4	29	2	31	3	0	24	3	25
182	6	27	3	29	1	31	2	0	24	2
183	25	4	27	2	31	1	0	24	1	25
184	2	27	1	0	15	-1	18	-1	25	1
185	0	15	-2	18	-2	19	1	0	15	-4
186	18	-4	19	2	20	-1	27	1	0	15
187	-8	18	-8	19	3	20	-1	27	2	0
188	15	-4	18	-4	19	4	20	-2	27	3
189	0	15	-2	18	-2	19	3	20	-5	27
190	4	0	15	-1	18	-1	19	2	21	1
191	27	3	0	19	1	20	1	21	2	27
192	3	0	16	-1	20	1	21	3	27	3
193	0	16	-2	20	1	21	4	27	3	0
194	16	-4	21	3	23	1	27	4	31	1
195	0	16	-8	21	2	22	-1	23	2	24
196	-1	27	3	31	2	0	16	-4	21	1
197	22	-2	23	3	24	-2	26	1	27	1
198	31	2	0	16	-2	22	-4	23	4	24
199	-4	26	2	27	-1	31	2	0	16	-1
200	22	-8	23	3	24	-8	26	3	27	-1
201	31	-1	0	22	-4	23	2	24	-4	26
202	5	31	-6	0	22	-2	23	1	24	-2
203	26	5	31	-3	0	17	-1	22	-1	24
204	-1	26	5	31	-2	0	17	-2	26	5
205	30	-1	31	-1	0	17	-4	26	3	30
206	-2	0	17	-8	26	2	30	-1	0	17
207	-4	26	1	0	17	-2	0	17	-1	31
208	-1	0	31	-2	0	31	-4	0	31	-8
209	0	31	-4	0	31	-2	0	21	1	31
210	-1	0	21	2	25	1	28	-1	0	21
211	3	25	2	28	-2	0	21	3	25	3
212	28	-1	0	21	1	25	4	0	21	-2
213	25	3	0	21	-7	25	2	0	21	-4
214	25	1	26	-1	30	1	0	21	-2	24
215	-1	26	-2	30	2	0	21	-1	24	-2
216	26	-4	30	3	0	23	1	24	-4	26
217	-8	30	4	0	23	2	24	-8	26	-4
218	30	3	0	23	3	24	-4	26	-2	30
219	2	0	23	4	24	-2	26	-1	30	1
220	0	23	3	24	-1	0	23	2	0	23
221	1	0	22	1	0	22	1	0	20	1
222	27	1	28	1	0	20	2	27	2	28
223	2	0	20	3	27	3	28	3	0	20
224	4	27	4	28	4	31	-1	0	20	3
225	27	3	28	2	31	-2	0	20	2	27
226	2	28	-1	31	-1	32	-1	0	20	1
227	27	1	28	-5	32	-2	40	1	0	25
228	-1	28	-9	32	-1	40	2	0	25	-2
229	28	-5	40	3	0	25	-1	28	-4	40
230	4	0	22	1	23	-1	28	-5	40	3
231	0	22	2	23	-2	28	-8	40	2	0
232	22	3	23	-4	28	-4	40	1	0	22
233	4	23	-7	28	-2	0	22	3	23	-2
234	28	-1	0	22	2	23	1	30	1	0
235	22	1	23	3	30	2	0	23	3	30
236	3	0	23	2	30	4	0	23	1	30
237	3	0	25	-1	30	2	0	21	1	25
238	-2	30	1	0	21	2	25	-1	0	21
239	3	0	21	4	0	21	3	29	-1	0

240	21	2	29	-2	0	21	1	29	-1	0
241	31	-1	0	26	-1	31	-2	0	26	-2
242	31	-1	0	26	-4	32	-1	0	26	-8
243	27	1	30	1	32	-2	0	26	-4	27
244	2	30	2	32	-3	0	22	1	26	-2
245	27	3	30	3	32	-6	0	22	2	26
246	-1	27	4	30	4	32	-1	40	-1	0
247	22	3	27	3	30	3	32	2	40	-2
248	0	22	4	27	2	30	2	32	2	40
249	-1	0	22	3	27	1	30	1	32	2
250	0	22	2	32	1	0	22	1	0	24
251	-1	27	-1	0	24	-2	27	-2	0	24
252	-4	27	-4	0	24	-8	27	-8	0	24
253	-4	27	-4	0	24	-2	27	-2	0	24
254	-1	27	-1	0	25	1	0	24	1	25
255	2	27	1	0	24	2	25	3	27	2
256	0	24	3	25	4	27	3	0	24	4
257	25	3	27	4	0	22	1	24	3	25
258	2	27	3	30	1	0	22	2	24	2
259	25	1	27	2	29	-1	30	2	0	22
260	3	24	1	27	1	29	-2	30	3	0
261	22	4	28	-1	29	-1	30	4	0	22
262	3	27	-1	28	-2	30	3	0	22	2
263	24	-1	27	-2	28	-1	30	2	0	22
264	1	24	-3	27	-1	30	1	0	24	-8
265	43	-1	0	24	-9	43	-2	0	24	-4
266	43	-1	0	24	-2	31	-1	0	24	-1
267	31	-2	0	31	-1	0	27	1	28	-1
268	0	27	2	28	-2	0	27	3	28	-1
269	0	27	4	29	-1	0	27	3	29	-2
270	0	27	2	28	-1	29	-1	0	26	-1
271	27	1	28	-2	0	23	1	26	-2	28
272	-1	31	1	0	23	2	26	-1	27	-1
273	29	-2	31	2	0	23	3	27	-2	29
274	-4	31	3	0	23	4	27	-1	29	-2
275	30	1	31	4	0	23	3	30	2	31
276	3	0	23	2	28	-1	30	3	31	2
277	0	23	1	28	-2	30	4	31	1	0
278	28	-2	30	3	0	28	-2	30	2	0
279	28	-1	30	1	0	32	-1	0	32	-2
280	0	32	-1	0	31	1	0	31	2	0
281	21	-1	31	3	0	21	-2	31	4	0
282	21	-1	31	3	0	31	2	0	31	1
283	0	29	1	0	29	2	0	29	3	30
284	1	0	29	4	30	2	0	26	1	29
285	3	30	3	0	26	2	29	2	30	4
286	32	-1	0	26	3	29	1	30	3	32
287	-2	0	26	4	30	2	32	-1	0	25
288	-1	26	3	28	-1	30	1	0	25	-2
289	26	2	28	-2	0	25	-4	26	1	28
290	-1	0	25	-8	0	25	-4	30	-1	0
291	25	-2	30	-2	35	-1	0	25	-1	30
292	-1	35	-2	0	35	-1	0	26	1	0
293	26	2	0	26	3	31	1	0	26	4
294	31	2	39	-1	0	26	3	31	3	39
295	-2	0	26	2	31	4	39	-1	0	26
296	1	31	3	0	31	2	0	31	1	0
297	39	1	0	26	-1	39	2	0	26	-2
298	39	3	0	25	1	26	-1	39	4	41
299	-1	0	25	2	39	3	41	-2	0	25

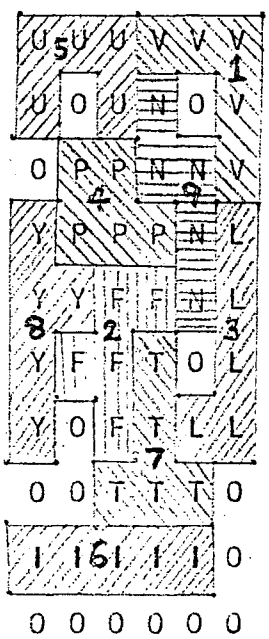
300	3	39	2	41	-1	0	25	4	39	1
301	0	25	3	0	25	2	0	25	1	0
302	39	-1	0	31	-1	39	-2	0	31	-2
303	39	-1	0	31	-1	36	-1	0	36	-2
304	0	36	-1	0	36	1	0	29	-1	36
305	2	0	29	-2	36	3	0	29	-1	36
306	4	0	36	3	0	36	2	0	36	1
307	0	29	1	0	29	2	0	29	3	0
308	29	5	0	29	5	0	29	5	0	29
309	5	0	29	3	0	29	2	0	29	1
310	0	34	-1	0	30	-1	34	-2	38	-1
311	0	30	-2	34	-1	38	-2	0	30	-1
312	38	-1	0	34	-1	0	34	-2	42	-1
313	0	34	-1	42	-2	0	42	-1	0	38
314	-1	0	38	-2	0	38	-1	0	41	-1
315	0	41	-2	0	41	-1	0	29	-1	0
316	29	-2	32	-1	0	29	-1	32	-3	0
317	32	-4	41	-1	0	32	-3	41	-2	0
318	32	-1	41	-1	0	33	1	0	33	2
319	0	33	3	0	33	4	37	-1	0	33
320	3	37	-2	41	-1	0	33	2	37	-1
321	41	-2	0	33	1	41	-1	0	33	-1
322	37	-1	0	33	-2	37	-2	0	31	-1
323	33	-1	37	-1	0	31	-2	0	31	-1
324	0	30	1	33	1	0	30	2	33	2
325	0	26	-1	30	3	33	3	0	26	-2
326	30	4	33	4	0	26	-1	30	3	33
327	3	0	30	2	33	2	0	30	1	33
328	1	0	38	-1	0	38	-2	0	38	-1
329	0	36	-1	0	36	-2	0	36	-1	0
330	34	-1	0	34	-2	0	34	-1	0	29
331	-1	0	29	-2	0	29	-1	0	34	1
332	41	1	0	34	2	41	2	0	34	3
333	41	3	0	34	4	41	4	0	30	-1
334	34	3	41	3	0	30	-2	34	2	41
335	2	0	30	-1	32	1	34	1	41	1
336	0	32	2	0	32	3	0	32	4	0
337	31	1	32	3	0	31	2	32	2	36
338	-1	0	31	3	32	1	36	-2	0	31
339	4	36	-1	0	31	3	36	-1	0	31
340	2	36	-2	0	31	1	36	-1	0	37
341	1	0	37	2	0	37	3	0	30	1
342	37	4	0	30	2	37	3	0	30	3
343	37	2	0	30	4	37	1	38	-1	0
344	30	3	38	-2	0	-2	0	31	1	36
345	-1	0	0	0	0	37	1	0	37	2
346	0	37	3	0	30	1	37	4	0	30
347	2	37	3	0	30	3	37	2	0	30
348	4	37	1	38	-1	0	30	3	38	-2
349	0	0	0	0	0	0	0	0	0	0
350	0	0	0	0	0	0	0	0	0	0
351	0	0	0	0	0	0	0	0	0	0
352	0	0	0	0	0	0	0	0	0	0
353	0	0	0	0	0	0	0	0	0	0
354	0	0	0	0	0	0	0	0	0	0
355	0	0	0	0	0	0	0	0	0	0
356	0	0	0	0	0	0	0	0	0	0
357	0	0	0	0	0	0	0	0	0	0
358	0	0	0	0	0	0	0	0	0	0
359	0	0	0	0	0	0	0	0	0	0

(E) SAMPLE GAMES

```
60 6 10 0 0 0 100 1                                    (1)

YOUR MOVE
'w' 1 8

  Y   2  29  1507  53

YOUR MOVE
'i' 2 21
NO FIT
            IT IS STILL  YOUR MOVE


'i' 1 25

  F   4  10   668  43

YOUR MOVE
'u' 2 49

  P   4  59   283  26

YOUR MOVE
'l' 6 26
        8                    0                    46
  V   2  34    49  13

I WIN                        1                    0
```

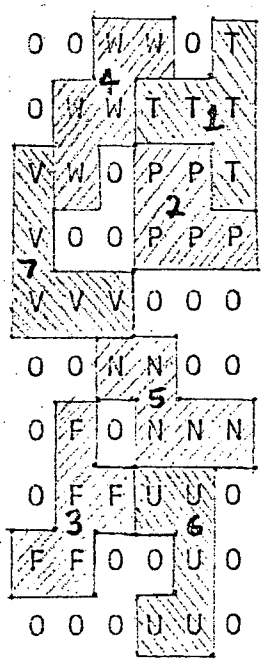


```
INPUT NEW GAME SPECS
60 6 10 0 0 0 100 11

YOUR MOVE
'v' 3 6

  F   1  33  1698  51

YOUR MOVE
'l' 8 36

  P   4  21   518  40

YOUR MOVE
'u' 3 2

  I   2  51   173  24

YOUR MOVE
't' 3 46
      23                    0                  24
  Y   8  25      6   3

YOUR MOVE
'n' 1 17
YOU WIN                     2                  0
```



```
INPUT NEW GAME SPECS
```

```
INPUT NEW GAME SPECS
60 6 10 0 0 0 100 12

  T   2  12  2056  60

YOUR MOVE
'p' 4 23

  F   3  44  1070  48

YOUR MOVE
'w' 2 9

  N   7  40   370  32

YOUR MOVE
'u' 4 53
      30                    0                    121

  V   1  25    42  10

I WIN                       2                    0
```



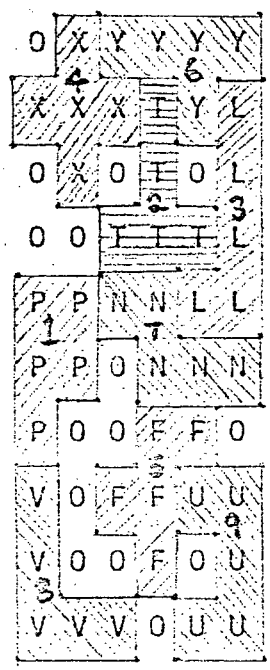

```
INPUT NEW GAME SPECS
```

```
INPUT NEW GAME SPECS
60 6 10 0 0 0 100 12

  P   1  31  2056  60

YOUR MOVE
't' 3 22

  L   8  24   716  48

YOUR MOVE
'x' 1 8

  F   1  46   450  28

YOUR MOVE
'y' 5 5
       94
  N   7  34    45  22

YOUR MOVE
'v' 1 55

  U   4  54     5   3

I WIN
```

④

5 743 1065

1 0




```
INPUT NEW GAME SPECS
$end

 THANK YOU AND BYE-BYE
#EXECUTION TERMINATED
```