STANFORD ARTIFICIAL INTELLIGENCE PROJECT
Memo No. 22

September 3, 1964

KALAH - THE GAME AND THE PROGRAM

by Richard Russell

Abstract: A description of Kalah and the Kalah program, including subroutine descriptions and operating instructions.

The research reported here was supported in part by the Advanced Research Project Agency of the Office of the Secretary of Defense (SD-183)

KALAH - THE GAME AND THE PROGRAM

by Richard Russell

Description of Kalah:

For beginners the game is best for two players with the board resting crosswise between them. Each player controls a row of round PITS on his side and the capsule-shaped bowl at his right called his KALAH. The object of the game is to get the largest number of counters (playing pieces) in each KALAH.

The number of counters used depends upon the time available to play and the age of the players. For a short game and for youthful players three counters are placed in each PIT. Adults use four, five or six counters, six making the most interesting game.

The first player is selected by lot or agreement and alternates in succeeding games. Each player empties any of his PITS deemed advantageous and leaving it empty, distributes counters one by one around to the right as far as they go. If there are enough to reach beyond his own KALAH,they are distributed one by one into the PITS on the opposite side and then belong to the other player. The only place ever skipped is the opponent's KALAH. Once in a KALAH, the counters remain until the end of the game.

The method of play is distributing one by one around to the right subject to two simple rules: 1. If the last counter lands in your own KALAH, you have another turn. By planning to have the right number of counters in two or more PITS it is possible to have several turns in succession. 2. If the last counter lands in an empty PIT on your own side you capture all of the counters in the PIT opposite and place them together with the one making the capture, in your own KALAH. A capture ends your turn.

In reaching an empty PIT on your own side, it makes no difference whether you move a single counter one space, or distribute all around the board and back to your own side.

Colors have no meaning except to make counting easier. Counters may be counted in any PIT on either side at any time.

The game ends when all PITS one one side are empty. Counters remaining on the other side go into the KALAH on that side. The score is the number captured from the opponent. The score is quickly counted by restoring the original number of counters in each PIT. For instance, if one side has four left over his score is four to nothing. A series of games ends with a score of forty on one side. In tournaments, the eliminating score is also forty.

Description of the Kalah Program:

The program is designed to search the move tree from any given board position to a set depth, and minimaxing back through the tree to determine the principle variation (best play by both sides -- one side seeking to minimize, the other to maximize the final evaluation). The recursive equations for this simple minimax procedure are:

sup(l,f) = if null l then $-\infty$ else max [f(car(l)), sup(cdr(l),f)]

inf(l,f) = if null l then ∞ else min[f(car(l)), inf(cdr(l),f)]

where l is a list (the move tree)

f is a function (the evaluation function)
car(l) is the first item on the list
cdr(l) is the remaining items on the list

sup(l,f) gives the maximum value of a position evaluated by function f

inf(l,f) gives the minimum value.

The true value of a board position p(the value resulting from best play by both sides) is found by the following equations for function f.

$v^+(p)$ = if ter (p) then val (p) else sup [successors of p, v^-]

$v^-(p)$ = if ter (p) then val (p) else inf [successors of p, v^+]

where ter(p) is true if p is terminal (end of game or end of search depth) val (p) is the immediate evaluation of a position.

$v^+(p)$ is then the true value of a position when it is the maximizing player's turn.

$v^-(p)$ is the value of a position when it is the minimizing player's turn.

A program which solved these equations for a given position would search the entire move tree. That this is not necessary can be illustrated by the example below. Consider a move tree with only 3 branchings at each point (fig. 1). The program will make the moves for both the maximizing and minimizing players (call them a and b, respectively) until the set depth, in this case 4, is reached at E.

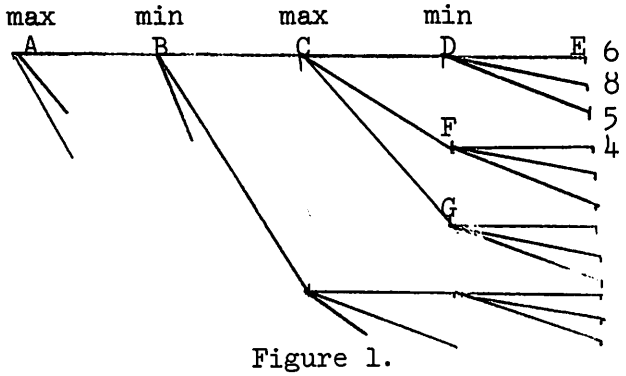


Figure 1.

This position is then evaluated. The last move made is retracted and the second and third moves from position D made and evaluated. Since it is b's turn, D would be assigned the minimum value, 5. There are now no further moves at D, and the program returns to position C to make a second move (to F). At F, the first move evaluated has a value of 4; but the maximizing player (a) is already guaranteed a value of at least 5 by the move to D. Therefore the remaining moves at F need not be evaluated: the move from C to F is poor. If the program can take advantage of such situations, the necessary area of search can be decreased immensely: for such situations occur at every level of the move tree. This is accomplished by the α - β heuristic; and the modified minimax equations of that heuristic are given below:

$$\sup(l;f;\alpha;\beta) = \underline{\text{if}}\ \text{null}\ (l)\ \underline{\text{then}}\ \alpha\ \underline{\text{else if}}\ f[\text{car}(l)] \geq \beta\ \underline{\text{then}}\ \beta\ \underline{\text{else}}\ \sup[\text{cdr}(l);f;\ \max[\alpha,f(\text{car}(l))];\beta]$$

$$\inf(l;f;\alpha;\beta) = \underline{\text{if}}\ \text{null}\ (l)\ \underline{\text{then}}\ \beta\ \underline{\text{else if}}\ f(\text{car}(l)) \leq \alpha\ \underline{\text{then}}\ \alpha\ \underline{\text{else}}\ \inf[\text{cdr}(l);f;\alpha;\ \min[\beta,f(\text{car}(l))]]$$

$$v^{+}(p,\alpha,\beta) = \underline{\text{if}}\ \text{ter}\ (p)\ \underline{\text{then}}\ \max\ [\alpha,\ \min(\beta,\ \text{val}(p))]\ \underline{\text{else}}\ \sup\ [\text{successors of p};\ \lambda[\pi,\ v^{-}(\pi,\ \alpha,\ \beta)];\ \alpha;\beta;]$$

$$v^{-}(p,\alpha,\beta) = \underline{\text{if}}\ \text{ter}\ (p)\ \underline{\text{then}}\ \max\ [\alpha,\ \min\ (\beta,\text{val}(p))]\ \underline{\text{else}}\ \inf\ [\text{successors of p};\ \lambda[\pi;v^{+}(\pi,\alpha,\beta)];\ \alpha;\beta]$$

where: π is a position and $\lambda[\pi;v^{+}(\pi,\alpha,\beta)]$ is

notation for: " λ , where λ is a function such that $\lambda\ (\pi) = v^{+}(\pi,\alpha,\beta)$"

If α and β are initially set to $-\infty$ and $+\infty$, a simple minimax procedure will be followed, taking advantage of all situations similar to the example above. If α and β are set closer together, any values generated not between α and β will be set to the appropriate (closest) limit. Some evaluator discrimination will thus be lost, and search time decreased accordingly.

In practice, the α - β heuristic not only saves a great deal of time, but makes unnecessary the storage of more than three numbers at each level of the tree: α,β, and the move ordering; a position is never expressly assigned a value. When the program has reached the full permissable depth or is at the end of a game, the board position is evaluated and stored in a variable val. If the last move was made by the minimizing player (b), val is compared with α: if val $> \alpha$, val becomes the new value of β (if val $< \beta$; otherwise β remains unchanged) the move is then retracted, and a new move made, evaluated, and compared. But if val $\leq \alpha$, two moves are retracted, and the push down list pointer is moved back one level. It is now the maximizing player's turn.

Similarly, if when the program has reached full depth the last move was made by the maximizing player (a), val is compared with β. If val $< \beta$, val replaces the old value of α (but only if val $> \alpha$; otherwise α is unchanged). If val $\geq \beta$, the search is abandoned, and the last two moves are retracted. Thus α and β approach each other as the search continues, continually decreasing the number of moves that must be examined. This process takes place at each level of the move tree, as can be illustrated by returning to our examples.

Assume α and β at all levels to be set initially to $-\infty$ and $+\infty$; when the program examines the possible moves from D, level 4 β is set to 5, the minimum value. Val is also 5 when the program returns to C. At C it is the maximizing player's turn, and the val returned from position D is compared with level 3β and replaces the level ($\alpha<5<\beta$) ($5 < +\infty$). The program moves to F, and evaluates the first move from that position. Since at F it is the minimizing player's turn, val (=4) is compared with level 4 α(=5). Val $< \alpha$, so the search at F is abandoned and the program returns to C. Val (=4) is less than level 3β, but also less than level 3α, so α and β remain unchanged and the program moves to G. This process continues until the program has worked back through the entire move tree.

At this point it should be clear that the order in which moves are examined by this minimax program (the long look ahead) is very important: for if the best moves are examined first, α and β at each level are immediately set to their final values, and many bad moves are not investigated. All evaluations will be outside (or equal to) the bounds set by α and β, and the program will proceed with maximum speed. In fact, only one of the possible moves at each point of the tree where it is the winning player's turn will be examined. Since this is in each case the best move, it will defeat all possible replies. The saving in effort can be easily demonstrated. Suppose a search 10 deep is being carried out and at each point in the tree 6 moves are possible (a reasonable average). If at each point the best move is examined last, the total number of positions examined will be 6^{10}. But if the best move is always examined first, at half the points on the tree, only one move will be made. The number of positions examined will be only $2x6^5$. It is of course most important

to correctly order the first moves, since every error results in the examination of the entire move tree following that move.

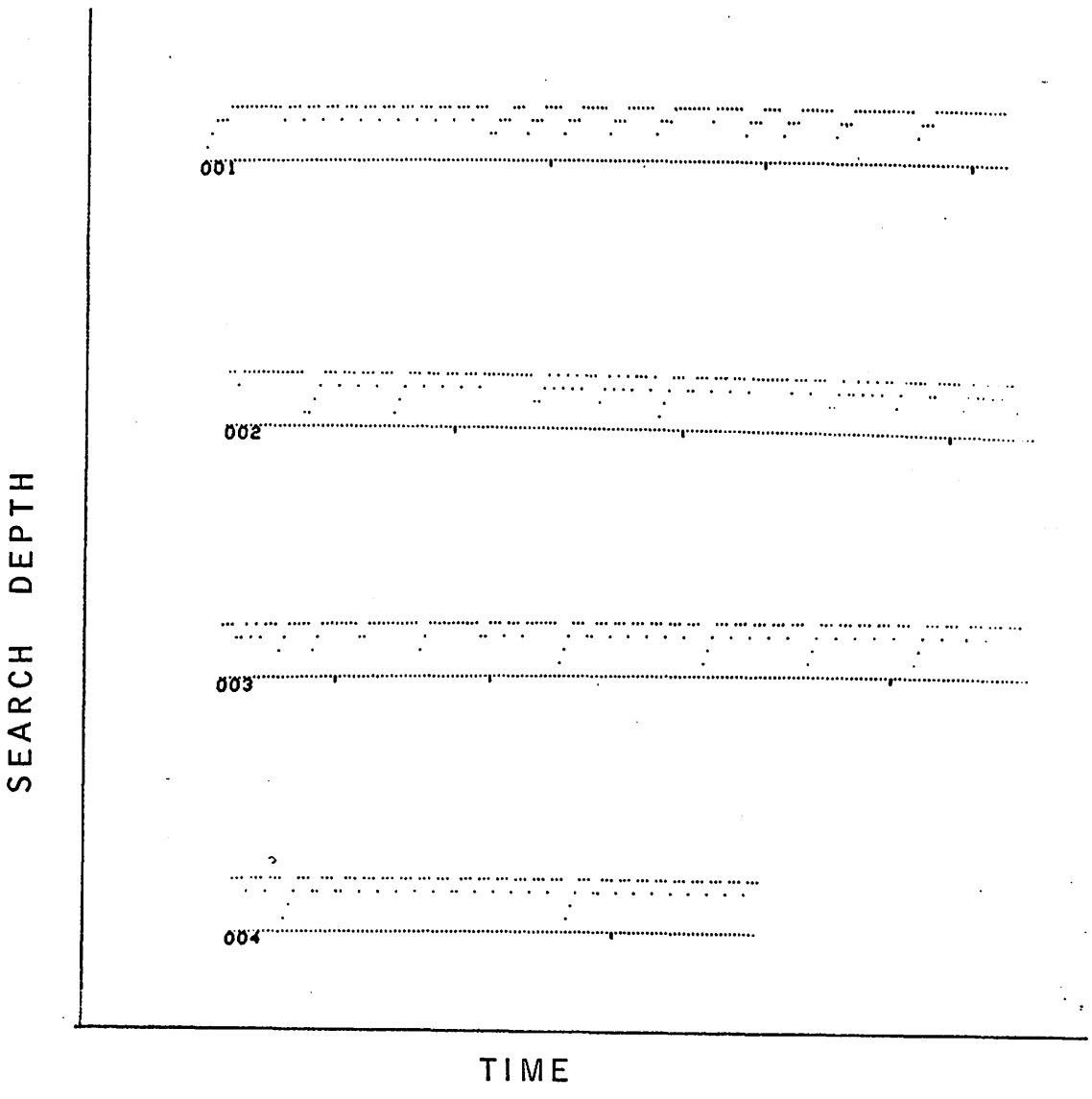
This ordering is accomplished in the Kalah program by the short look ahead order generator. This is a copy of the main minimax program,but which searches the move tree to a lesser depth, and orders the possible moves from a position in by their estimated values. On a deep search by the long look ahead, the order generator commonly looks to a depth of 5 for the first few moves; this depth is governed by a short look depth table. During a long analysis, the most efficient setting of the order generator results in spending 90 percent of the machine time in the short look ahead; this usually guarantees a near perfect ordering of the moves: and the long look ahead becomes simply a proof routine for the moves so generated.

There remain a few tricks that further improve program speed. One of these is the "gamma mode" option. Under the option, if a move is evaluated to be gamma (gm) greater than β (for a) or gamma less than α (for b) further moves are not investigated: this move is considered adequate. Gamma is now initialized at 8 (10 octal), a number too large to result in much saving of effort, since the evaluation routine is rarely likely to return a value 8 more or less than the limits. (See Subroutine write-up for a description of the evaluator). This routine makes the assumption that the evaluator is reliable. It cannot be used in a proof, and only operates if $\alpha < 0$ and $\beta > 0$.

Another timesaving routine is the postponable move checker: any move that can be made later with exactly the same effect is put off. As a result, it may not have to be investigated at all.

Finally, two other shortcuts may be employed if the user has a good guess as to the probable outcome of the analysis. They are especially effective in an analysis which looks to the end of the game. The first of these is to set the initial α and β (for each level) close on each side of the suspected value of the game. If the guess is right, a close α and β will result in a more rapid analysis; but if wrong, no information at all will be gained. Thus in an analysis to the end of the game, α and β should be set to 145 and 146 (octal) respectively if a win by the first player is expected (that is, 101 and 102 decimal: val ± 100 signifies end of game; therefore a victory of at least 2 is demanded - the smallest win possible). If the position is indeed a win, 102 will be the final value; if it is not, or if the program has not been set to a great enough depth to prove it, 101 will be the value. Unless depth has been set very deep, this tells you nothing. (α and β are set by ia and ib - see Operating Instructions). Finally, if a win by one side or the other seems fairly certain, the short look ahead for the losing side may be turned off (another option) since all possible replies will have to be checked anyway. These shortcuts are really only helpful in an analysis for a proof of the efficiency of a move: they depend on correct guessing.

Lastly, some concrete results from the Kalah program should be mentioned. Set to a depth of 15 octal, with $\alpha = 145$ and $\beta = 146$, two in a pit Kalah can be proved a win by the first player in about five minutes (short look ahead depth = 3). A prolonged analysis of 3 in a pit Kalah has also demonstrated a win by the first player (α is the winning move - see Operating Instructions for move notations). The Kalah program also provides(2 is as on option) a CRT display of search depth against time. A photograph of the same appears below (fig. 2): horizontal lines of dots signify repeated moves (a transfer of stones ending in the kalah) or the final evaluations at search depth. Height is depth of search; the analysis itself and the variations which are being plotted are appended. This is an analysis of 2 in a pit Kalah to depth 5. There is no short look ahead; pit ordering is by nearness to kalah: Moves 1, 2, 3, 4, 5, 6 are considered in that order.



```
STG

HALT.     AT LOC 041631     AC- 377000        IO- 442000

T
\U#

\O\D\I\N
T,1632
M1-       R1,213-213 E60 P60 V0
          R21,213-3 E8 P8 V0                          Photo1
          R23,213-1 E8 P8 V0
          R24,213-1 E10 P10 V0
          R25,213-1 E10 P10 V0
          R26,214-1 E19 P19 V0
          R3,213-1 E8 P8 V0
          R4,213-1 E11 P11 V0
          R5,213-1 E17 P17 V0

HALT.     AT LOC 041631     AC- 375001        IO- 442000
T
          R6,214-1 E35 P35 V0
M21-      R1,3-213 E19 P19 V0
M23-      R1,6-213 E30 P30 V1
          R21,6-3 E14 P14 V1                          Photo2
          R23,1-1 E13 P13 V0
M24-      R1,36-213 E36 P36 V4
          R21,36-3 E14 P14 V4
          R23,31-1 E14 P14 V4
          R24,36-1 E11 P11 V4

HALT.     AT LOC 041631     AC- 376002        IO- 442000
T
          R26,35-1 E16 P16 V4
          R3,1-1 E7 P7 V4
          R4,31-1 E23 P23 V4                          Photo3
          R6,35-1 E30 P30 V4
M25-      R1,1-213 E31 P31 V4
M26-      R1,1-213 E22 P22 V4
M3-       R1,1-213 E22 P22 V4
M4-
HALT.     AT LOC 041631     AC- 377003        IO- 442000
T
          R1,1-213 E35 P35 V4
M5-       R1,1-213 E61 P61 V4
M6-       R1,1-213 E43 P43 V4                         Photo4
 TE627 TP627
PV24-1,36-213
2.4,
```

```
T,3742

HALT.    AT LOC 041631    AC- 377000       IO- 442000
T
1-1,213-213 V0
1-1,213-214 V2
1-1,213-215 V6
1-1,213-23, V8
1-1,213-24, V2
1-1,213-25, V6
1-1,213-3, V2
1-1,213-4, V8
1-1,213-5, V6
1-1,214-213 V-2
1-1,215-213 V-6
1-1,216-213 V0
1-1,23-213, V-8
1-1,24-213, V-2
1-1,25-213, V-6
1-1,26-213, V-1
1-1,3-213, V-2
1-1,4-213, V-8
1-1,5-213, V-6
1-1,6-213, V-2
1-21,213-3, V2
1-21,213-4, V2
1-21,213-5, V8
1-23,213-1, V10
P1M1,1-23,213-2, V12
1-23,213-4, V10
1-23,213-5, V6
1-24,213-1, V4
1-24,213-31 V6
1-24,213-32 V8
1-24,213-35 V2
1-24,213-5, V6
1-25,213-1, V8
1-25,213-31 V10
1-25,213-32 V12
1-25,213-34 V12
1-25,213-4, V12
1-26,213-1, V1
1-26,213-31 V0
1-26,213-32 V1
1-26,213-34 V1
1-26,213-35 V1
1-26,213-4, V2
1-26,213-5, V-1
1-26,214-1, V8
1-26,214-31 V10
1-26,214-32 V12
1-26,214-34 V12
1-26,214-4, V12
P1M2,1-3,213-1, V10
1-3,213-2, V12
1-3,213-4, V7
1-3,213-5, V6
1-4,213-1, V8
1-4,213-2, V10
1-4,213-31, V12
1-4,213-32, V14
1-4,213-35, V8
1-4,213-5, V7
1-5,213-1, V6
1-5,213-21, V8
1-5,213-231 V10
1-5,213-232 V12
1-5,213-234 V12
1-5,213-24, V12
1-5,213-31, V10
1-5,213-32, V12
1-5,213-34, V14
1-5,213-4, V7
1-6,213-1, V2
1-6,213-21, V1
1-6,213-231 V0
1-6,213-232 V1
1-6,213-234 V1
P1M3,1-6,213-235 V1
1-6,213-24, V2
1-6,213-25, V-1
1-6,213-31, V1
1-6,213-32, V1
1-6,213-34, V2
1-6,213-35, V2
```

```
HALT.    AT LOC 041631    AC- 375001       IO- 442000
T
1-6,213-4, V3
1-6,213-5, V3
1-6,214-1, V6
1-6,214-21, V8
1-6,214-231 V10
1-6,214-232 V12
1-6,214-234 V12
1-6,214-24, V12
1-6,214-31, V10
1-6,214-32, V12
1-6,214-34, V14
1-6,214-4, V7
21-1,3-213, V-2
21-1,4-213, V-2
21-1,5-213, V-8
21-1,6-213, V-1
23-1,1-213, V-10
23-1,2-213, V-12
P2M1,23-1,4-213, V-10
23-1,5-213, V-6
23-1,6-213, V1
23-1,6-215, V1
23-1,6-216, V1
23-1,6-23, V2
23-1,6-25, V2
23-1,6-26, V2
23-1,6-3, V3
23-1,6-5, V3
23-1,6-6, V3
23-21,1-3, V-8
23-21,2-3, V-5
23-21,4-3, V-5
23-21,5-3, V-5
23-21,6-3, V2
23-21,6-5, V2
23-21,6-6, V2
23-23,1-1, V0
23-23,2-1, V-1
23-23,4-1, V-1
23-23,5-1, V2
23-23,5-2, V3
23-23,5-5, V-2
23-23,6-1, V-1
P2M2,24-1,1-213, V-4
24-1,31-213 V-6
24-1,32-213 V-8
24-1,35-213 V-2
24-1,36-213 V4
24-1,36-216 V4
24-1,36-23, V6
24-1,36-26, V4
24-1,36-3, V6
24-1,36-6, V4
24-1,5-213, V-6
24-1,6-213, V1
24-21,1-3, V-2
24-21,31-3, V-4
24-21,32-3, V-6
24-21,35-3, V-4
24-21,36-3, V6
24-21,36-6, V6
24-23,1-1, V3
24-23,1-2, V4
24-23,1-4, V4
24-23,1-5, V4
24-23,1-6, V4
24-23,31-1, V4
24-23,31-2, V5
P2M3,24-23,31-4, V5
24-23,31-5, V5
24-23,31-6, V5
24-24,1-1, V0
24-24,31-1, V-2
24-24,32-1, V-4
24-24,36-1, V4
24-24,36-6, V4
24-26,1-1, V3
24-26,1-3, V4
```

```
HALT.    AT LOC 041631    AC- 376002       IO- 442000
T
24-26,1-4, V4
24-26,1-5, V4
24-26,1-6, V4
24-26,31-1, V2
24-26,32-1, V3
24-26,35-1, V8
24-26,35-3, V10
24-26,35-5, V8
24-3,1-1, V4
24-3,1-2, V4
24-3,1-4, V5
24-3,1-5, V5
24-3,1-6, V5
24-4,1-1, V4
24-4,1-2, V4
P3M1,24-4,1-31, V3
24-4,1-32, V3
24-4,1-35, V2
24-4,1-36, V4
24-4,1-5, V-2
24-4,31-1, V5
24-4,31-2, V5
24-4,31-31, V6
24-4,31-32, V8
24-4,31-34, V5
24-4,31-35, V5
24-4,31-36, V5
24-4,31-4, V6
24-4,31-5, V6
24-4,31-6, V6
24-6,1-1, V4
24-6,1-21, V3
24-6,1-23, V4
24-6,1-24, V4
24-6,1-25, V4
24-6,1-26, V4
24-6,1-3, V5
24-6,1-4, V5
24-6,1-5, V5
24-6,1-6, V5
P3M2,24-6,31-1, V5
24-6,31-21, V2
24-6,32-1, V4
24-6,32-21, V3
24-6,35-1, V6
24-6,35-21, V8
24-6,35-23, V10
24-6,35-25, V8
24-6,35-3, V10
24-6,35-5, V6
25-1,1-213, V-8
25-1,31-213 V-10
25-1,32-213 V-12
25-1,34-213 V-12
25-1,36-213 V0
25-1,4-213, V-12
25-1,6-213, V-2
26-1,1-213, V-3
26-1,3-213, V-4
26-1,4-213, V-4
26-1,5-213, V0
26-1,6-213, V-2
3-1,1-213, V-10
3-1,2-213, V-12
3-1,4-213, V-7
P3M3,3-1,5-213, V-6
3-1,6-213, V0
4-1,1-213, V-8
4-1,2-213, V-10
4-1,31-213, V-12
4-1,32-213, V-14
4-1,35-213, V-8
```

```
HALT.     AT LOC 041631     AC- 377003        IO- 442000
T
4-1,36-213, V1
4-1,5-213, V-7
4-1,6-213, V-3
5-1,1-213, V-6
5-1,21-213, V-8
5-1,231-213 V-10
5-1,232-213 V-12
5-1,234-213 V-12
5-1,236-213 V0
5-1,24-213, V-12
5-1,26-213, V-2
5-1,31-213, V-10
5-1,32-213, V-12
5-1,34-213, V-14
5-1,36-213, V1
5-1,4-213, V-7
5-1,6-213, V-3
6-1,1-213, V-4
P4M1,6-1,21-213, V-3
6-1,23-213, V-4
6-1,24-213, V-4
6-1,25-213, V0
6-1,26-213, V-2
6-1,3-213, V-5
6-1,4-213, V-5
6-1,5-213, V-5
6-1,6-213, V-3
2.4,
                    0  2  2  2  2  0  2
                       2  2  0  3  0  3  4
         #

\O\D\I\N
E,F

IP/       LAW 2
OP/       MV 135    =          221
IA/       LAW I MV+313        =          710377
IB/       LAW MV+313          =          700377
ID/       LAW 5
TD/       RV 176    =          777
```

(some for 200,225 in this instance)

Kalah Subroutines

MV - Move subroutine

This subroutine is entered with a jda, the accumulator containing the number of the pit to be moved from. The subroutine makes the move, if possible, it returns to the call location + 1. The move made, and stones captured, are stored in the push down list. (PDL) EGC is called before return to the main program.

EGC - End Game Check

This routine checks for the end of the game. If one side has move than half the stones, or if neither side can prevent the other from moving the rest of his stones, the game is considered over and flag 5 is set. The final score is placed in the accumulator.

RV - Revert

Entered with a jda, this takes back the last move, moves back the punch pointer, and undoes everything the move routine has accomplished.

DI - Double Length Interpreter

This is entered by jsp DI. It causes all commands following the jsp to be treated as double precision commands, until a jmp command is reached. Single precision mode is then resumed. This makes possible accurate computation of effort and other variables otherwise too large.

MM - Minimax Routine

The heart of the program. Applies the α - β minimax procedure to a board position, returning to ST with the best move it has found when done. It examines moves in the order determined by the short look ahead and contains the display routine (MMS). Move orderings, current α and β, and principle variations are stored in the push down list.

XMX - Marker Display

Enters display markers every y searches to depth or end.

PMV - Prints Move

In analysis printout, prints "m", followed by the current move under consideration.

PRP - Prints Reply

In analysis, prints reply to each move, followed by the principle variation.

storage of principle variation or call of short look ahead. Moves ordered by ORD. Returns value of given board position in AC.

ORD - Ordering Subroutine

Order moves for SM, returning the ordering (best at right) in AC. Ordering is as follows: possible moves are divided into two groups: those that make captures or land in the kalah (a), those that do not (b). Each group is internally ordered by pit number: smallest (nearest to kalah) first. Finally, group a precedes group b in the last ordering. Thus, if we have (a) 2,4,5 and b(1,6) the returned order will be 061542.

ST - Start

This routine listens to the typewriter for program commands, and executes them, returning always (except at end of game) to the listen loop. End of game causes a halt. Entrance at ST causes initialization of board and values. Entrance at ST + L causes only value initialization to take place.

PM - Postponable Moves

This routine checks a given move (in the AC when called) to see if it can be made later with exactly the same effect. (It is then postponable). If this is the case, control returns to point of call + 1; otherwise, to point of call + 2.

The routine is called by option 3; postponable moves are then skipped. this saves a considerable amount of machine time.

CNS - Constants

Adds one word to PDL.

ERS - Erase

Returns one word of PDL to free storage.

CER - Error Printout

Types "out of free storage".

MFS - Make Free Storage

Generates free storage for PDL.

QES - Tests Sense Switches

QIA - Prints Primary Variation when SS3 down.

DPT - Decimal Printout

Prints in decimal the number in the AC when called.

PT - Octal Printer

Prints in octal the number in AC when called.

XPO - Marker Printout

Prints current variation at marker: requires SS5.

EV - Board Evaluator

Returns evaluation in AC. If more than 16 stones are on the board (kalahs excluded). KA (kalah a) - KB (Kalah b) is the returned value. Otherwise all moves are made simultaneously (no captures are made) and the number of stones on side a less the number on side b is the returned value.

OG - Short Look Order Generator

Looks into the game a depth specified by table (TD) using α - β heuristic, and returns in the AC a listing of possible moves, ordered according to value such that the best move is the furthest right. In a long analysis, most of the time is spent in the short look ahead.

IN - Initialization Routine

All values except the board and push down list (PDL) are initialized.

SU - Board Set Up

The board position and PDL are initialized.

PB - Print Board

Requires SS4. Prints board.

PP - Principle Variation Printer

Unravels and prints out current principle variation; called by XPO and QIA. (SS5 and SS3 calls for current variation, respectively)

SM - Straight Minimax

Minimax routine for short look ahead. Identical to MM except no

Important Variables:

al - alpha
bt - beta
dp - depth
ef - effort count = number of positions examined
sap - alpha for short look ahead
sbt - beta for short look ahead
pd - pointer for push down list
pdl - first location of free storage
bpv - end of free storage, last word of program
fsl - free storage pointer
mpv - end of minimax principle variation list = current move
pvt - pointer for principle variation list
sor - st - 1 - option to call ord(rather then order 654321)

ia - sets alpha
ib - sets beta
ip - sets number stones per pit
id - sets depth
td - depth table for short look ahead
op - option register

kb b1 b2 b3 b4 b5 b6

a6 a5 a4 a3 a2 a1 ka

Actual board. a6 - kb are consecutive locations.

Program Flags:

1 - not used
2 - local use
3 - local use (decimal printout subroutine)
4 - on at end of turn
5 - on at end of game
6 - on if second player (b) to move

Operating Instructions for Kalah Program:

The current version of the Kalah program includes microflit, a limited DDT which permits no tape reading or punching. Microflit starts at 6740; the lowest location used by the program is location 40. The microflit symbols table includes only 2 and 1 letter symbols of the Kalah program.

1. The program proper begins at ST. This initializes all values, including the initial board position. Entrance at ST + 1 initializes everything but the board position.

2. The board itself is stored in the consecutive locations A6 ... Kb. The board is represented as follows:

	b1	b2	b3	b4	b5	b6
Kb						Ka
	a6	a5	a4	a3	a2	a1

Important constants must be set before operation by microflit. They are:

ia = law α ib = law β	These set α - β in the α - β heuristic used by the minimax routine. Values returned by the board evaluator outside of these limits will be replaced the appropriate limit (α - β).
id = law dp,	where dp is the depth of search for the minimax routine
ip = law p,	where p is the initial number of stones per pit.
td = x00yyy	This is the first location of a table governing depth of search for the short look ahead. x is the depth searched until the program is y moves into the game (where one emptying of a pit is considered a move). Then depth is set by the second member of the table, and so on. The last entry in the table is customarily set to 777 (search depth = 0 until 777 moves deep).

y - marker frequency: a marker is placed every y searches to full depth or end of game.

op - option register: each of the first 9 bits in this register provides an option. They are:

1- short look ahead taken from the table; otherwise depth set to 3.
2- gamma mode: if a move is gm more than the value required by $\alpha - \beta$, it is considered adequate. No other moves are investigated. This option only operates if $\alpha < 0$ and $\beta > 0$.
3 - postponable moves are postponed.
4 - the evaluator is set to strictly "bird in hand": a position is evaluated as Ka - Kb.
5 - short look ahead table entries are compared against the number of moves into the game. If off, the table (y port) is compared with the number of stones in the combined kalahs = Ka + Kb.
6- no short look ahead on a's turn.
7 - no short look ahead on b's turn.
8 - photo halt. at the end of a display line, computer halts. Pressing continue causes continuation until end of next line: facilitates photography.
9 - display cutoff: no display if on.

Don't forget: all values are in octal.

4. Sense switches:

1+2 - do nothing
3 - prints current variations
4 - prints board
5 - prints current variations at display markers
6 - no analysis printout if on

5. In the printout:

m = move
r = reply
v = value of position (± 100 if at end of game).
c = effort = number of board positions examined.
p = effort in long look ahead

The current principle variation is the list of moves currently evaluated. Pits are listed by number, 1 being the closest to the Kalah, 6 the farthest away. Each ply is separated from the next by dashes; each move (one ply for each side) by commas.

Thus the printout

m5 r1,2-312,416e77p38v100

would mean: move five, reply 1,principle variation from there ply of 2 followed by 3, 1, 2, ply 4, 1, 6. e,p,v printout is decimal: 77 positions examined, 38 by the long look ahead, end of game seen, a tie.

At the end of an analysis,

te - gives total effort
tp - gives total long look ahead effort
pv - gives final principle variation board printout is decimal

In SS5 printout,

p - is the number of the display line (appearing on the scope)
m - is the marker number
both of these are octal numbers.

6. Typewriter control:

r - takes back the last move; hit at the beginning, r makes side b start first.

space - tells the machine to move; other output then the move depends on sense switches.

digit from 1 - 6 - makes move indicated. If this move ends in the Kalah, a period is typed. Otherwise, a comma is printed, and the machine awaits further instruction.

Any other characters, and illegal moves are not accepted. Backspace x is typed.

7. CRT Display

Depth of search is displayed against time.

8. Initial condition:

When read in, the above constants are:

ia - law i 377
ib - law 377
id - law 5
ip - law 3
td - 50010
td+1 - 30040
td+2 - 20070
td+3 - 777
y - 31
op - 6