Program Description I

Program Title THE GAME OF 3-D TIC-TAC-TOE

Contributor's Name ALAN STEIN
Address 372 SHOTWELL ST
City SAN FRANCISCO **State** CA **Zip Code** 94110

Program Description, Equations, Variables This program pits the HP 67 against a human opponent in the game of 3-D TicTacToe. The rules of this game are simple:

1) The board is a cube consisting of 4 levels, each of which is 4 rows deep and 4 columns across, making a total of 64 squares on a 3-dimensional board.

2) Two players move alternately by placing a black or white marker on a square(or making an X or a 0 on a paper layout of the board). Once a move is made, the piece is never moved or removed. In this game, the human always goes first.

3) A player wins by placing four of her markers in a straight line. The line can lie in more than one level; e.g. diagonals are perfectly legitimate wins.

In short, the game is played just like regular TicTacToe, except that the board has one additional dimension, and is one square bigger in all dimensions. Unlike regular TicTacToe, there is no known winning strategy for the 3-D version. It is a much more complex game which can require considerable skill in a player, allowing for very complicated strategies.

The 67 plays and remembers the game by dividing the board into its 16 component rows and storing an entire row in one register. The registers R_0 through R_{15}(R_{S5}) are reserved for the game board. (see illustrations on next page).

Each square on the board can be characterized by its level=z, its row=y and its column=x. X,y,& z can have values from 1 through 4.

Operating Limits and Warnings When entering moves, make sure they are 3 digit decimal numbers. All three digits must be between 1 and 4. Entering a move outside this range may cause the program to make erroneous entries in the board, and blow up the game.

02925D · Program Description I

Program Title 3-d tictactoe

Contributor's Name

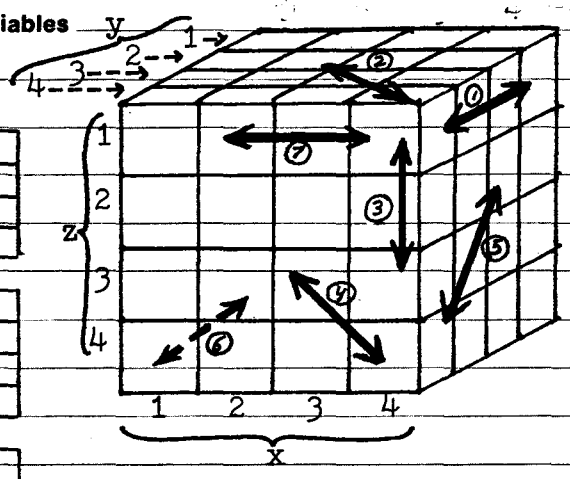
Address

City **State** **Zip Code**

Program Description, Equations, Variables

LEVEL (z)	ROW (y)	REG. #	BOARD
1	1	0	
1	2	1	
1	3	2	
1	4	3	
2	1	4	
2	2	5	
2	3	6	
2	4	7	
3	1	8	
3	2	9	
3	3	10	
3	4	11	
4	1	12	
4	2	13	
4	3	14	
4	4	15	

Column(x) = 1 2 3 4



Note: The arrows with circled numbers indicate the seven kinds of win lines, and will be referred to later.

The table to the left shows how various combinations of level and row numbers correspond to various registers. Note that the value of x does not affect the register number.

The player enters a move by providing the machine with the z,y & x positions for the move. The format is a:

3 digit decimal number : .ZYX.

DO NOT FORGET THE LEADING DECIMAL POINT when entering a move. The machine will respond in the same form, with one exception-- if the machine wins, it will indicate this by outputting a move in the form 1.ZYX -- the leading "1" indicates this is a winning move.

If the player makes a winning move, the machine will recognize this and halt during the execution of Card 1 with 4.000 in the display.

Program Description I

Program Title 3-d tictactoe

Contributor's Name

Address

City State Zip Code

Program Description, Equations, Variables The 67 stores the moves in a row(register) of the board in the following form:

N.0a0b0c0d. : N=total number of moves made so far in that register(0 through 4). a,b,c,& d indicate the conditions of spaces (x-value) 4,3,2 and 1 in that row.

If a--d = 0, the space is empty. If a--d = 1, the space contains a machine's piece. If a--d = 5, the space contains a human player's piece.

For example: If R_3 = 3.01000501, this means that there are three pieces in the topmost front row(see illustration on preceding page). The first and last pieces are machine pieces, and the second space(x=2) has a player's piece in it. The third square is empty

THE FOLLOWING SECTION DESCRIBES HOW THE COMPUTER DECIDES WHAT MOVE TO MAKE. Skip the rest of Program Description I, if you wish to play against the machine without knowing its strategy ahead of time.

The machine plays both a strategic game-- based on obtaining a strong overall position--and a tactical game, which either counters an immediate specific threat or takes immediate advantage of an unblocked line.

While both games are determined by the player's moves, the 67 does not actually look more than one move ahead.

There are two periods in a game. During the opening game, the machine will respond to a threatening situation produced by the human player, but will not attempt to to advantage of a strong tactical move that would advance its position. Rather, it makes strategic moves whenever possible, to lay out a good opening pattern of pieces.

During the main game, strategic moves are made only when there is no tactical move to be made.

The machine makes a tactical move whenever it finds that the last move (of either player) has produced an unblocked line of two or three pieces for a player. The 67 moves to either block an opponent's line or

Program Description I

Program Title: 3-d tictactoe

Contributor's Name:

Address:

City: State: Zip Code:

Program Description, Equations, Variables

extend its own unblocked line. The 67 cannot remember from one move to the next. Hence, if it does not follow up on a tactical situation within one move, it will 'forget' about it, until another move causes a tactical search in that region. To keep running time down, the machine does not examine the entire board after each move. Instead it looks only at the win lines which are potentially changed by that move, (with minor exceptions which will be discussed later).

SEQUENCE OF OPERATIONS -- MAIN GAME

ENTRY OF PLAYER'S MOVE: The human player enters a 3-digit decimal fraction for her move, in the form .zyx: where z is the level of the move, y is the row, and x is the column. This number is stored in R_9 for future reference. The move is decomposed into its component values, which are stored as $R_B = y-1$, $R_C = z-1$, & $R_D = 4(z-1)+(y-1)$. This is the number of the register containing that row of the board. $R_A = 10^{10-2x}$. This number corresponds to the position in the register of the desired move. R_A*reg shifts the register to put the column corresponding to x in the first two integer digits to the left of the decimal. By taking INT, dividing by 100, and taking FRAC, we can then isolate those two digits. This procedure will be used frequently in the program.

Once the square is isolated, it is compared to 0. If it is $\neq 0$, then that move is illegal; i.e. the square is already filled, and the program attempts to jump to a nonexistant label, producing an error message. The player must then enter a correct move.

If the move is good, a 5 is entered in the appropriate location by adding $5/R_A$ to the reg. 1 is also added to the reg, to maintain the correct move count.

ANALYSIS OF HUMAN'S MOVE

There are seven types of possible win lines, as illustrated on pp2. They are:

1) A horizontal column, where x and z are constant

2) A horizontal diagonal, where z is constant. Note that there are two such diagonals, perpendicular to each other.

Program Description I

Program Title 3-d tictactoe

Contributor's Name

Address

City **State** **Zip Code**

Program Description, Equations, Variables 3) A vertical column; x and y are constant.

4) and 5) are two different kinds of vertical diagonals. To distinguish them, we will refer to (4) as a 'constant-y' diagonal and (5) and a 'constant-x' diagonal. Note that , like (2), there are two of each type, perpendicular to each other.

6) A full diagonal. There are four of these, running through the cube to opposite corners. Neither x,y, nor z is constant.

7) A row; y and z are constant.

For any given square on the board, (1),(3),and (7) exist. Also, at least one diagonal will exist. If the square lies on a full diagonal (6), then all diagonals are possible. Otherwise, only one diagonal, (2),(4), or (5) is possible.

It turns out that deciding which diagonals apply, while not difficult, takes longer than testing them all! The checking routines are designed to look at all types of diagonals. For those which have a constant term, it is set equal to the coordinate of the move; e.g. a move of .144 (in the illustration, the upper right near corner) would check (2) withz set =1 (illustrated), (4) with y=4 (illustrated), and (5) with x=4(illustrated). Note that the diagonals perpendicular to these would also be checked, even though they don't actually include the given move. Also, the four full-diagonals are checked every move.

The test of a line is performed by adding together the four appropriate squares. Since a player's move has a value of 5 and a 67 move, 1, there is a unique sum for each possible condition of the line. For example, a human's unblocked line must always have a sum divisible by 5. A machine's unblocked line must have a sum less than 5, etc.etc. In practice, the 67 is uninterested in all cases except unblocked lines of two or more pieces.

Only one subroutine is required to perform the summings and to test both the human and 67 win-lines. This subroutine is controlled by the following inputs:

R_I holds the starting register in the line.

R_E holds the amount by which R_I must be incremented to

Program Title 3-d tictactoe

Contributor's Name

Address

City State Zip Code

Program Description, Equations, Variables get to the next register in the line (e.g. for (3), R_E = 4; for (7), R_E=0; for (5), R_E = 3 or 5)

F_0 indicates that a player move is being analyzed, and is used to control the comparison tests run on the final sum.

F_1 controls the summing process itself. If F_1 is off, then the registers are simply added together, meaning all the squares having the same x-value are summed. If F_1 is on, after each term is added, the sum is multiplied by 100, giving a final sum of the form $10^6*R(i)+10^4*R(i+e)+100*R(i+2e)+R(i+3e)$. The diagonal will be summed in the first two digits to the right of the decimal.

It turns out that lines (1) thru (6), comprising 12 distinct cases, can be defined by four sets of R_I and R_E. Three kinds of summing are done on each set: F_1=off; F_1= on, with the registers summed in ascending order; and F_1=on, with the registers summed in descending order (which gives the two diagonals perpendicular to each other). Case (7) is treated as a seperate case--note that the summing is done with F_1 on; i.e. it is treated as a diagonal.

Once the sum is completed, and the desired two-digit sum isolated, it is divided by 5 and a remainder looked at. If it's 0, then the row is unblocked. The quotient gives the number of pieces in the row. This number is compared with R_{s7} which holds the highest such number found in the round (and is initialized to 2, at the start of the round). If the quotient is $\geq R_{s7}$, it replaces the old R_{s7}. In addition, F_2 is set, to indicate a tactical move was found, and the # of this particular test is recalled from R_{s8} and stored in R_{s6}.

The 67 repeats this sum and check procedure until all win-lines have been examined.

ANALYSIS OF MACHINE'S MOVE

The testing procedure is identical with that of the previous section. F_0 is cleared to indicate different test criteria for eliminating blocked lines (and to skip the /-by-5 section). F2 is cleared: since it will be reset if a machine tactical status replaces a player's status it provides an indicator for later that we are dealing with a machine offense rather than a defense. F_3 is cleared to prepare for later use.

02925D Program Description I

Program Title 3-d tictactoe

Contributor's Name

Address

City State Zip Code

Program Description, Equations, Variables

The previous 67 move is recalled and parsed, so that the correct x,y, and z values are available.

The test subroutine LBL 0 is run.

PREPARE FOR CARD 2

F_2 is checked--if it is off, then F_3 is set and the player's move is parsed to load the x,y,and z values into the registers.

The test # in R_{56} is recalled and analyzed to tell the 67 which line was the critical one. R_I is loaded, and F_0 & F_1 are set appropriately for the routines on Card 2.

Execution then jumps to LBL C, which is a loop containing a MERGE/PAUSE sequence. The MERGE retains the flag statuses set by part 1. LBL C loops until Card 2 is read in.

DECIDE ON MACHINE'S NEW MOVE

Card 2 begins with a GTO(i) which jumps to any of 6 routines. 1 thru 5 simply initialize the 67 to check for open spaces in a win-line defined by the test #. It is similar to the procedure used to sum a specific win-line. This will be discussed in more detail, later.

A"0"(GTO LBL 0) indicates that no tactical position passed the test criteria. In this case, a strategic move is made. The 67 selects the move in the following fashion:

Each of the levels of the board is summed, by adding the registers together, starting with level 2. The level having the least # of moves on it is selected. If two levels have the same number, the first one found is the chosen one. This favors the inner levels.

The same sort of selection process is performed on the files(the column of registers having y constant), starting with file 2. The intersection of the chosen level and file defines one register in which the machine will attempt to move. Depending on the reg.#, one of two sequences of square-searching is selected; this provides a better layout of pieces, then would be gotten from using the same pattern all the time.

Each square in the register is checked to see if it is empty, in a manner similar to that used to check the validity of the player's move.

Program Description I

Program Title 3-d tictactoe

Contributor's Name

Address

City State Zip Code

Program Description, Equations, Variables

For a tactical move, registers and squares within a register are examined along the chosen win-line. The selection procedure for the registers is identical with that in the original testing routine. Instead of summing the registers, though, R_A is used to isolate the appropriate square, and it is tested to see if it is empty, in the fashion previously described. If a diagonal is being examined (F_1=on), no staggered sum is performed, rather, R_A is multiplied or divided by 100, as is appropriate, between registers.

RECORDING THE MACHINE'S MOVE

Once a satisfactory move is found, $1/R_A$ is added to the appropriate register, loading a '1' into the appropriate square. 1 is added to the register to update the move counter. Then, x is reconstructed from R_A and the coordinates combined to produce .zyx. This is stored in R_{s8}. The tactical status in R_7 is recalled, divided by 3 and the INT taken. If R_7=3, this produces a value of 1, otherwise, it =0. F_3 is checked. If it is off, it means that this was an offensive move and maybe the machine has moved in a row which had 3-in-a-row. That is, the 67 has won. If this is the case, then the value added to .zyx will be 1 and the output will be of the form 1.zyx. If this is not the case 0 will be added, and the output will be the standard one.

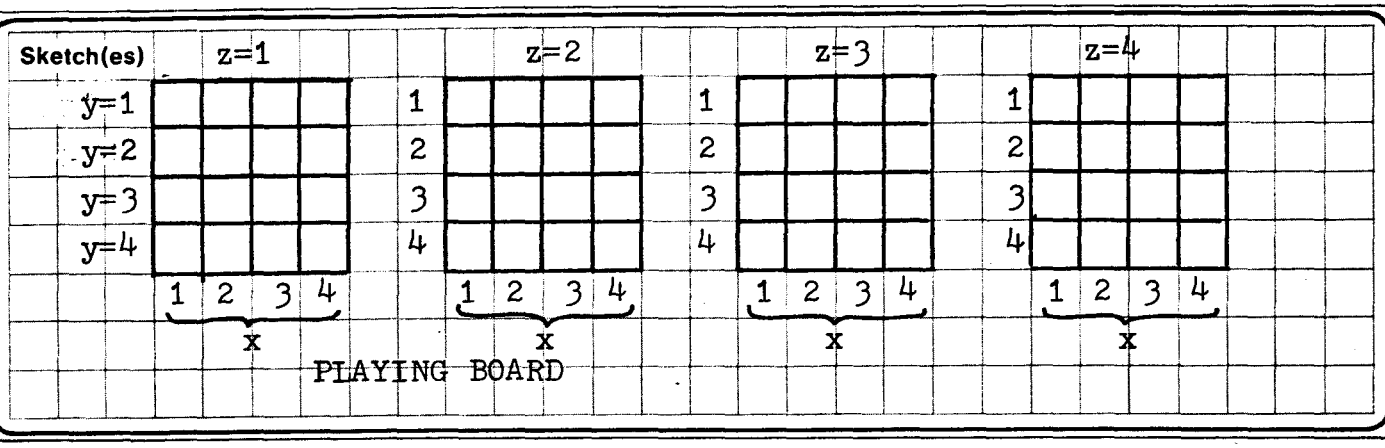
OPENING GAME

The opening game differs in the emphasis given tactics, and specifically in the use of R_{s8}. If R_{s8}=0, it is the first round, and the 67 skips the tactical analysis of Card 1. R_{s8} is used as a roundcounter= .01*# of rounds. So long as the count is less than .06, F_2 is set before Card 2 is loaded. F_2 inhibits the writing of the machine move in R_{s8}. During the opening game, the machine also skips the tactical analysis of its own moves, to emphasize strategy, but it does examine the player's move, starting in round 2, to look for threats. So long as R_{s8}<.1, the 67 plays opening game. Once a machine move is written into the register, the machine will skip the opening game variant.

This completes the description of the machine's playing pattern.

Program Description II

Sketch(es)



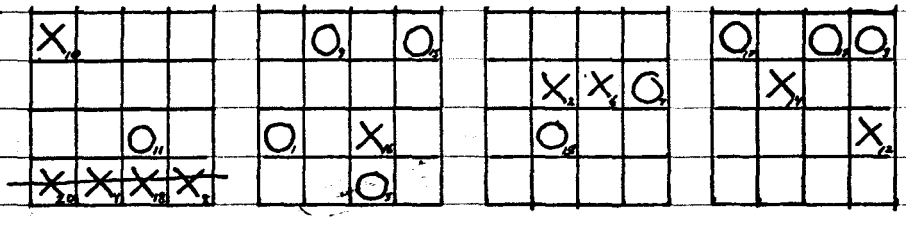
PLAYING BOARD

Sample Problem(s) A SAMPLE GAME IS PLAYED OUT HERE. This is a real game; since the program is deterministic, if you make the same moves, the machine will make the same responses as these.

move	ALAN	67	COMMENTS
1	.231	.322	Unusual human opening.
2	.414	.142	both sides are trying to set up position
3	.243	.323	
4	.324	.144	machine blocks two-in-row
5	.212	.111	machine blocks two-in-row
6	.133	.434	human blocks two-in-row
7	.332	.422	
8	.214	.233	human tries to set up trap, machine does so 1st.
9	.411	.143	aaarrrggghhh
10	.413	1.141	MACHINE WINS!

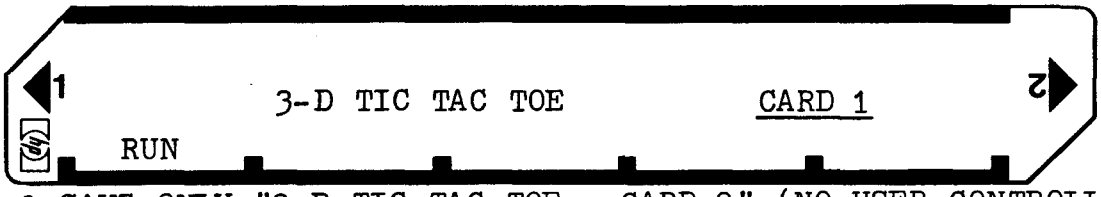
In general, the machine is more likely to win a short game, the human, a long one.

Solution(s)



Reference(s)

User Instructions



CARD 2 SAYS ONLY "3-D TIC TAC TOE : CARD 2".(NO USER CONTROLLED FUNCTIONS)

STEP	INSTRUCTIONS	INPUT DATA/UNITS	KEYS	OUTPUT DATA/UNITS
1	Load both sides of card 1			
2	Enter player's move(level,row & column)	.zyx	A	
3	Load both sides of Card 2 (Merge)	-----	-- ---	mach.move .zyx
4	Go to Step 1, and repeat sequence until end of game.			

NOTE 1: The machine takes from ¼ to 2 min to make a move.

NOTE 2: The loading of card 2 is controlled by a Merge/Pause loop. When the machine wants card 2 it will flash .500 until card 2 is entered. Alternately, Card 2 may be inserted in the reader as soon as "A" is started, and the machine will automatically read the first side when it comes to the loop.

NOTE 3: If the machine makes a winning move the output will display 1.zyx, instead of .zyx. If the machine detects a winning player move, it will halt during Card 1 and display 4.000.

NOTE 4: If a card-read error occurs at step 3, press R/S to clear the error. DO NOT reinsert Card 2; instead press C , and immediately insert Card 2, at the Pause. The machine will then ask for "Crd". Load Side 2 of Card 2 into the reader without pressing any keys.

This procedure is necessary because Card 2 makes use of the flags set by Card 1-- without the MERGE instruction in LBL C, the machine would erase the flags when reading Card 2.

STEP	KEY ENTRY	KEY CODE	COMMENTS
001	* LBL C	31 25 13	MERGE 2nd CARD
	MERGE	32 41	
	PAUSE	35 72	auto-read or wait
	GTO C	22 13	for card in loop
	* LBL A	31 25 11	DETERMINE BOARD STATUS
	P ↔ S	31 42	
	STO 9	33 09	R_{s9}= player move
	0	00	
	STO 6	33 06	zero test #
010	2	02	set minimum test
	STO 7	33 07	for two-in-row
	R ↓	35 53	return player
	R ↓	35 53	move to X-reg.
	P ↔ S	31 42	
	GSB B	31 22 12	parse move &store
	ST I	35 33	reg# in I
	RCL A	34 11	column entry
	RCL(i)	34 24	shift reg so that
	x	71	proposed move's
020	INT	31 83	position is in
	EEX	43	1st 2 integer
	2	02	places, then chop
	/	81	off fraction, di-
	FRAC	32 83	vide and frac to
	x ≠ 0?	31 61	isolate digits.
	GTO E	22 15	If filled, ERROR.
	5	05	otherwise, store
	RCL A	34 11	05 in that posi-
	/	81	tion of reg #
030	STO+(i)	33 61 24	to record move
	ISZ(i)	32 34	add one to total
	P ↔ S	31 42	move tally
	RCL 8	34 08	If R_8=0, then it
	x = 0?	31 51	is first round.
	GTO d	22 31 14	F_0 means it's test
	SF 0	35 51 00	of player's move.
	GSB 0	31 22 00	TEST MOVE
	CF 2	35 61 02	reset flags
	CF 3	35 61 03	for machine's
040	CF 0	35 61 00	move tests.
	.	83	Initially, R_8
	0	00	contains # of
	6	06	rounds/100. For
	RCL 8	34 08	7 rounds, play
	FRAC	32 83	only opening game.
	STO 8	33 08	if not, continue
	x > y?	32 81	tests at e.
	GTO e	22 31 15	if R_8=.06, prepare
	x = y?	32 51	to store mach.
050	GTO c	22 31 13	move, next play.
	* LBL d	32 25 14	INCREMENT ROUND COUNT
	EEX	43	
	CHS	42	F_2 prevents store
	2	02	of mach.move in
	STO + 8	33 61 08	opening plays
	SF 2	35 51 02	

STEP	KEY ENTRY	KEY CODE	COMMENTS
	GTO e	22 31 13	skip tactics
	* LBL E	32 25 15	TEST MACH.MOVE
	GSB B	31 22 12	parse mach move
060	GSB 0	31 22 00	test for tact.
	F ? 2	35 71 02	status. If F_2,
	GTO c	22 31 13	then mach.move
	SF 3	35 51 03	has priority. If
	RCL 9	34 09	not, SF 3 and
	GSB B	31 22 12	reset memory for
	* LBL c	32 25 13	player's posit'n.
	RCL 6	34 06	recall test #
	P ↔ S	31 42	
	INT	31 83	determine proper
070	1	01	test call for
	-	51	card 2, and set
	3	03	flags to match
	/	81	test condition
	1	01	on card 1; i.e.
	+	61	initialize so
	ST I	35 33	that proper line
	FRAC	32 83	or diagonal is
	x = 0?	31 51	tested. Note
	CF 1	35 61 01	that #=0 means
080	.	83	strategy, rather
	5	05	than tactic will
	x > y?	32 81	determine mach
	SF 0	35 51 00	response.
	GTO C	22 13	CALL CARD 2
	* LBL B	31 25 12	PARSE MOVE
	EEX	43	.zyx
	1	01	10
	x	71	z.yx
	INT	31 83	z.00
090	LST x	35 82	
	FRAC	32 83	.yx
	EEX	43	10
	1	01	
	x	71	y.x
	INT	31 83	y
	LST x	35 82	
	FRAC	32 83	.x
	.	83	
	5	05	.x-.5
100	-	51	
	CHS	42	.5-.x
	2	02	
	0	00	20
	x	71	2(5-x)
	10**x	32 53	10**2(5-x) = position in regist
	STO A	33 11	y
	R ↓	35 53	
	1	01	R_B = y-1
	-	51	
110	STO B	33 12	z
	x ↔ y	35 52	
	1	01	

REGISTERS

0	1	2	3	4	5	6	7	8	9
****PLAYING	BOARD*****	*PLAYING	BOARD*****	*PLAYING	BOARD*****	current test #	tactic status	test # mach move	player move
S0	S1	S2	S3	S4	S5	S6	S7	S8	S9

A	B	C	D	E	I
$10^{2(5-x)}$	Y-1	4(Z-1)	Register #	position increment	indirect

STEP	KEY ENTRY	KEY CODE	COMMENTS
	-	51	z-1
	4	04	
	x	71	$R_C = 4(z-1)$
	STO C	33 13	Reg # =
	+	61	$4(z-1)+(y-1)$
	STO D	33 14	R_D = Reg #
	RTN	35 22	
120	* LBL 0	31 25 00	INITIALIZE TEST CONTROLS
	1	01	There are 5 classes of registers corresponding to possible win-lines. All but the last are associated with 3 possible orientations of such lines. This routine sets up the classes for LBL 7 to look at.
	RCL C	34 13	
	GSB 7	31 22 07	
	4	04	
	RCL B	34 12	
	GSB 7	31 22 07	
	5	05	
	ENTER	41	
	0	00	
130	GSB 7	31 22 07	
	3	03	
	ENTER	41	
	GSB 7	31 22 07	
	0	00	
	STO E	33 15	
	RCL D	34 14	
	ST I	35 33	
	GTO 9	22 09	
	*LBL 7	31 25 07	
140	CF 1	35 61 01	This test looks only for win in a single row.
	ST I	35 33	SET TEST POINTS
	R↓	35 53	non-diagonal test
	STO E	33 15	1st register in row.
	GSB 9	31 22 09	Interval between TEST!
	SF 1	35 51 01	Set diagonals reverse interval to count back to start TEST!
	RCL E	34 15	
	CHS	42	
	STO E	33 15	
	GSB 9	31 22 09	
150	RCL E	34 15	Count up; check diag running other way.
	CHS	42	
	STO E	33 15	TESTING CYCLE
	*LBL 9	31 25 09	Increment test#.
	1	01	set primary reg's.
	STO+8	33 61 08	
	P↔S	31 42	fill stack with 100's
	EEX	43	
	2	02	
	ENTER	41	
160	ENTER	41	
	RCL(i)	34 24	pull first reg.
	x↔I	35 24	
	RCL E	34 15	increment I
	+	61	
	x↔I	35 24	
	F ? 1	35 71 01	
	x	71	if looking at diagonals, x 100.
	RCL(i)	34 24	next register.

STEP	KEY ENTRY	KEY CODE	COMMENTS
	+	61	add registers.
170	x↔I	35 24	
	RCL E	34 15	increment I
	+	61	
	x↔I	35 24	continue staggered summing for diagonals
	F ? 1	35 71 01	
	x	71	
	RCL (i)	34 24	end sum is $reg_1+reg_2+reg_3+reg_4$ or $10^6 reg_1 + 10^4 reg_2 + 10^2 reg_3 + reg_4$. (i.e. staggered sum).
	+	61	
	x↔I	35 24	
	RCL E	34 15	Note that "single row" test "staggers".
180	+	61	
	x↔I	35 24	Normalize non-diagonal sum = S
	F ? 1	35 71 01	$100/A = 10^{x-8}$
	x	71	
	RCL (i)	34 24	$S_{new} = S_{old} * 10^{8-x}$
	+	61	
	F ? 1	35 71 01	ANALYZE SUM
	GTO D	22 14	Isolate 1st two decimal digits- which are sum of moves in tested line.
	R↑	35 54	
	RCL A	34 11	set 2nd reg's. y=4, x=Sum
190	/	81	
	/	81	If looking at player, then skip to LBL 2.
	* LBL D	31 25 14	
	FRAC	32 83	If sum>4, then line is blocked by player.
	R↑	35 54	
	x	71	ANALYZE PLAYER SUM
	INT	31 83	y=4, x=Sum/5
	P↔S	31 42	
	4	04	If true, then player has WON.
	x↔y	35 52	
200	F ? 0	35 71 00	If not integer, then line is blocked by mach.
	GTO 2	22 02	
	x>y?	32 81	recall Sum/5
	RTN	35 22	TEST TAC STATUS
	GTO D	22 14	If old status is higher than Sum, end test.
	* LBL 2	31 25 02	
	5	05	If not, SF 2, to indicate tact.
	/	81	
	x = y?	32 51	Store Sum as new tactical status, store test # in R_6.
	R/S	84	
210	FRAC	32 83	
	x ≠ 0?	31 61	
	RTN	35 22	
	LST x	35 82	
	*LBL D	31 25 14	
	RCL 7	34 07	
	x>y?	32 81	
	RTN	35 22	
	x↔y	35 52	
	SF 2	35 51 02	
220	STO 7	33 07	
	RCL 8	34 08	
	STO 6	33 06	
	RTN	35 22	
	----	84	

LABELS

A RUN	B parse move	C Merge#2	D used	E -----	
a -----	b ------	c initial. Cd 2	d -----	e ------	
0 set tests	1 -----	2 used	3 ------	4 ------	
5 -----	6 ------	7 set tests	8 -----	9 run test	

FLAGS

0 player move	
1 diagonals	
2 used	
3 mach. move	

SET STATUS

FLAGS			TRIG	DISP	
	ON	OFF			
0	☐	☒	DEG ☒	FIX ☒	
1	☐	☒	GRAD ☐	SCI ☐	
2	☐	☒	RAD ☐	ENG ☐	
3	☐	☒		n 3	

Program Listing I

OLD Tactics--Card 2

STEP	KEY ENTRY	KEY CODE	COMMENTS
001	GTO(i)	22 24	Jump to proper test call. Note that these LBL's 1-5 correspond to the test initializations in LBL 0 on card 1. The functions handled by LBL 7 on card 1 are handled by the status of F_0 & F_1 which select the proper register set. LBL 0 on this card is strategic play--used in openings and when there is no tactical move indicated.
	*LBL 1	31 25 01	
	1	01	
	RCL C	34 13	
	GTO D	22 14	
	*LBL 2	31 25 02	
	4	04	
	RCL B	34 12	
	GTO D	22 14	
010	*LBL 3	31 25 03	
	5	05	
	ENTER	41	
	0	00	
	GTO D	22 14	
	*LBL 4	31 25 04	
	3	03	
	ENTER	41	
	GTO D	22 14	
	*LBL 5	31 25 05	
020	SF 1	35 51 01	
	0	00	
	RCL D	34 14	
	*LBL D	31 25 14	
	ST I	35 33	
	R↓	35 53	
	STO E	33 15	
	EEX	43	
	2	02	
	EEX	43	
030	8	08	
	F ? 0	35 71 00	
	x ↔ y	35 52	
	F ? 1	35 71 01	
	STO A	33 11	
	RCL A	34 11	
	*LBL a	32 25 11	
	GSB c	32 22 13	
	x ↔ I	35 24	
	RCL E	34 15	
040	+	61	
	x ↔ I	35 24	
	EEX	43	
	CHS	42	
	2	02	
	F ? 0	35 71 00	
	1/x	35 62	
	RCL A	34 11	
	F ? 1	35 71 01	
	x	71	
050	GTO a	22 31 11	
	*LBL 0	31 25 00	
	1	01	
	6	06	
	STO A	33 11	
	4	04	
	ST I	35 33	

TACTICAL PLAY

Initialize reg counter and increment value. For diagonal case, A must be set for appropriate diagonal, by F_0 (i.e. 100 or 10^8) If non-diagonal (F_1=off) use A set by parser.

FIND MOVE

test potential move if no good, then increment I to look at next position in line.

if looking at diagonal case, then change A by 10^2 or 10^{-2} depending on which diagonal.

go test new move

STRATEGIC PLAY

A is used as comparison reg. R_A = 16, initially. Set I,E for second level.

STEP	KEY ENTRY	KEY CODE	COMMENTS
	1	01	test 2nd level
	STO E	33 15	
	GSB e	32 22 15	
060	2	02	test 3rd level
	GSB e	32 22 15	
	3	03	test 4th level
	GSB e	32 22 15	
	0	00	test 1st level. recall z-1 of emptiest level.
	GSB B	31 22 12	
	RCL C	34 13	
	4	04	
	STO E	33 15	change increment
	x	71	
070	STO D	33 14	R_D= 4(z-1)
	1	01	
	6	06	reset comparison
	STO A	33 11	
	1	01	test 2nd file
	GSB B	31 22 12	
	2	02	test 3rd file
	GSB B	31 22 12	
	3	03	test 4th file
	GSB B	31 22 12	
080	0	00	test 1st file
	GSB B	31 22 12	recall y-1 of emptiest file
	RCL C	34 13	
	RCL D	34 14	I=reg #=4(z-1)
	+	61	+y-1
	ST I	35 33	
	* LBL E	31 25 15	FIND MOVE
	RCL(i)	34 24	x=4, y=register
	RCL E	34 15	if register is
	x ≤ y?	32 71	full, then try
090	GTO 9	22 09	another. Otherwise, go to LBL 6 or LBL 7 to find empty space.
	RC I	35 34	
	2	02	
	/	81	
	FRAC	32 83	
	x = 0?	31 51	
	GTO 6	22 06	
	GTO 7	22 07	
	* LBL 9	31 25 09	RESET REG #
	RC I	35 34	
100	+	61	Increment reg # by 4. If this gives # greater than 15, normalize modulo 16. Then return to "FIND MOVE".
	1	01	
	6	06	
	x > y?	32 81	
	CLX	44	
	-	51	
	ST I	35 33	
	GTO E	22 15	
	* LBL B	31 25 12	FIND TOTAL MOVES
	ST I	22 15	store initial #
110	* LBL e	32 25 15	call 1st register
	RCL(i)	34 24	
	x ↔ I	35 24	

REGISTERS

0	1	2	3	4	5	6	7	8	9
**********	PLAYING	BOARD	**********	**********	PLAYING	BOARD	**********	**********	**********
S0	S1	S2	S3	S4	S5	S6	S7	S8	S9
**********	PLAYING	BOARD	**********	**********	**********	Test #	tactic status	mach. move	player move

A	B	C	D	E	I
10^{10-2x} /scratch	Y - 1	(Z-1)	Reg #	increment	indirect

Program Listing II

STEP	KEY ENTRY	KEY CODE	COMMENTS
	RCL E	34 15	Increment index
	+	61	
	x ↔ I	35 24	
	RCL(i)	34 24	add next register
	+	61	
	x ↔ I	35 24	When done, all
	RCL E	34 15	four registers
120	+	61	represnting a given
	x ↔ I	35 24	level or file
	RCL(i)	34 24	of the board will
	+	61	be summed. The
	x ↔ I	35 24	integer part of
	RCL E	34 15	the sum is the
	+	61	total # of moves
	x ↔ I	35 24	made so far, on
	RCL(i)	34 24	that level or file
	+	61	
130	x ↔ I	35 24	
	RCL E	34 15	
	+	61	
	x ↔ I	35 24	
	INT	31 83	# of moves.
	RCL A	34 11	previous low #.
	x ≤ y ?	32 71	if new # is no
	RTN	35 22	lower, then return.
	R ↓	35 53	otherwise, store
	STO A	33 11	new low #
140	R ↓	35 53	and store appro-
	STO C	33 13	priate y-1 or z-1.
	RTN	35 22	
	* LBL 6	31 25 06	FIND EMPTY SPACE
	EEX	43	
	2	02	Look for empty
	GSB c	32 22 13	spaces in the
	EEX	43	order: 4th, 3rd,
	4	04	1st, and finally
	GSB c	32 22 13	2nd space in
150	EEX	43	row chosen by
	8	08	LBL 0 and E.
	GSB c	32 22 13	
	* LBL 7	31 25 07	FIND EMPTY SPACE
	EEX	43	Look in the
	6	06	order: 2nd, 3rd,
	GSB c	32 22 13	4th, and 1st.
	EEX	43	
	4	04	
	GSB c	32 22 13	
160	EEX	43	
	2	02	
	GSB c	32 22 13	
	EEX	43	
	8	08	
	* LBL c	32 25 13	----------
	STO A	33 11	CHECK/STORE MOVE
	RCL(i)	34 24	recall chosen reg
	x	71	and shift so that

STEP	KEY ENTRY	KEY CODE	COMMENTS
	INT	31 83	chosen space is
170	EEX	43	in integer range
	2	02	Take int. part,
	/	81	divide by 100
	FRAC	32 83	take fraction,
	x ≠ 0?	31 61	to isolate. If
	RTN	35 22	≠0, then move is
	RCL A	34 11	no good. Other-
	1/x	35 62	wise, store 1 in
	STO+(i)	33 61 24	that position
	ISZ(i)	32 34	for machine's
180	log	31 53	move & add 1 to
	2	02	integer part, to
	/	81	record total
	5	05	moves.
	+	61	x=log(A)/2 +5
	RC I	35 34	I=4(z-1)+y-1
	4	04	
	/	81	integer part=
	INT	31 83	z-1; frac part =
	LST x	35 82	y-1/4
190	FRAC	32 83	
	4	04	
	x	71	y-1
	1	01	
	+	61	y
	x ↔ y	35 52	z-1
	1	01	
	+	61	z
	EEX	43	
	1	01	10
200	x	71	10z
	+	61	10z+y
	EEX	43	
	1	01	10
	x	71	100z+10y
	+	61	100z+10y+x
	EEX	43	
	3	03	1000
	/	81	output = .zyx
	F ? 2	35 71 02	if F_2 is on, it
210	R/S	84	is still open-
	P ↔ S	31 42	ing game; R_8 is
	STO 8	33 08	being used to
	RCL 7	34 07	store # of round.
	P ↔ S	31 42	Otherwise, store
	3	03	mach. move in R8
	/	81	
	INT	31 83	recall old tact
	F ? 3	35 71 03	status. If = 3,
	CLX	44	then if F_3=on,
220	+	61	indicating
	R/S	84	machine tact.
			status, this is
			a machine win.
			Add 1 to output

LABELS					FLAGS	SET STATUS		
A -------	B sum rows	C -----	D set tests	E -----	0 choose diagonal	FLAGS	TRIG	DISP
a find move	b ------	c test move	d -----	e sum rows	1 set diagonal	ON OFF		
c strategy	1 tactic	2 tactic	3 tactic	4 tactic	2 openings	0 ☐ ☒	DEG ☒	FIX ☒
5 tactic	6 try move	7 try move	8 -----	Keep i<16	3 mach. move	1 ☐ ☒	GRAD ☐	SCI ☐
						2 ☐ ☒	RAD ☐	ENG ☐
						3 ☐ ☒		n 3