Qubic: 4 × 4 × 4 Tic-Tac-Toe
Author(s): Oren Patashnik
Source: *Mathematics Magazine*, Vol. 53, No. 4 (Sep., 1980), pp. 202-216
Published by: Taylor & Francis, Ltd. on behalf of the Mathematical Association of America
Stable URL: https://www.jstor.org/stable/2689613
Accessed: 15-11-2019 19:43 UTC

Qubic: 4×4×4 Tic-Tac-Toe

Pruning the game tree of three-dimensional tic-tac-toe makes possible a computer-aided proof that the first player can always win.

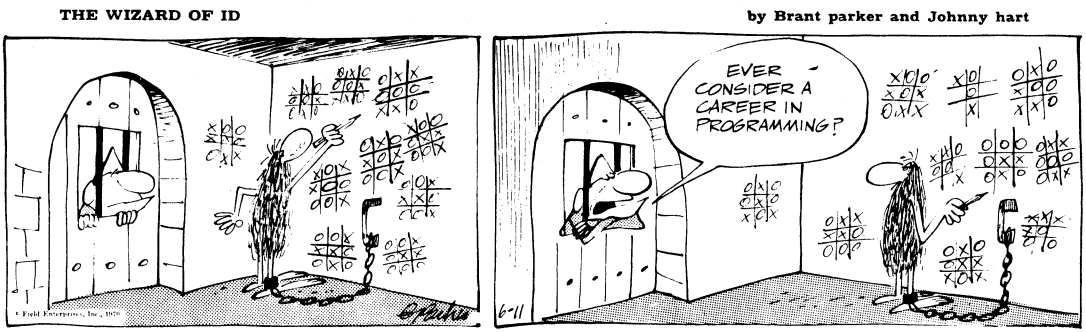
Oren Patashnik
Bell Laboratories
Murray Hill, NJ 07974

The computer is a powerful new tool for the mathematician. Its sheer size and speed make accessible many problems previously inaccessible (or at least unaccessed). The four-color theorem is a well-known example. In solving this problem Appel, Haken, and Koch [1], [2], [3] combined one hundred years of previous mathematical research with large high-speed computers to obtain a solution involving an intricate man-machine interaction. In this paper, we examine the problem of determining the outcome of 4×4×4 tic-tac-toe under optimal play. Not only is the problem itself interesting, but—as with the four-color theorem—so is the method of solution, which combines mathematics, human game-playing skill, and 1500 hours of computer time. We look first at the problem's mathematical background, then at the computer-aided solution, and finally at the solution's implications.

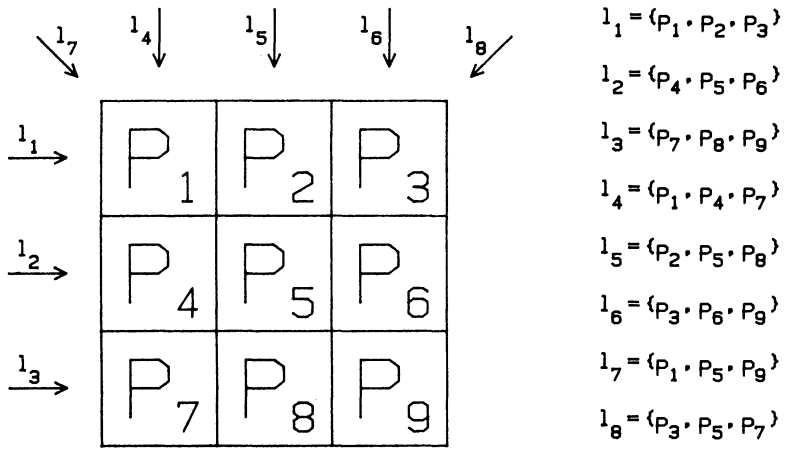
Mathematical background

The familiar two-dimensional 3×3 tic-tac-toe has been played for a long time; in fact, we don't know its origins. One needn't play very long, though, to see that each player can force a draw if he or she plays properly. A more interesting version of tic-tac-toe is played in three dimensions on a 4×4×4 cube. (This game is marketed by Parker Brothers as Qubic; henceforth we refer to it as such.) Before examining Qubic, we discuss q-player positional games in general, 2-player k^n tic-tac-toe games in particular.

A **q-player positional** game consists of a set P of points (the board); a set L of lines (the winning combinations), each a subset of P; and q players who take turns choosing unchosen points from P (i.e., moving) until either: (a) a player has chosen all the points in any line (that player wins), or (b) all the points in P are chosen (the players draw). For example, regular tic-tac-toe for two players has nine points and eight lines, as shown in FIGURE 1. For a given



THE WIZARD OF ID, by permission of Johnny Hart and Field Enterprises, Inc.



The nine points and eight lines of 3^2 tic-tac-toe.

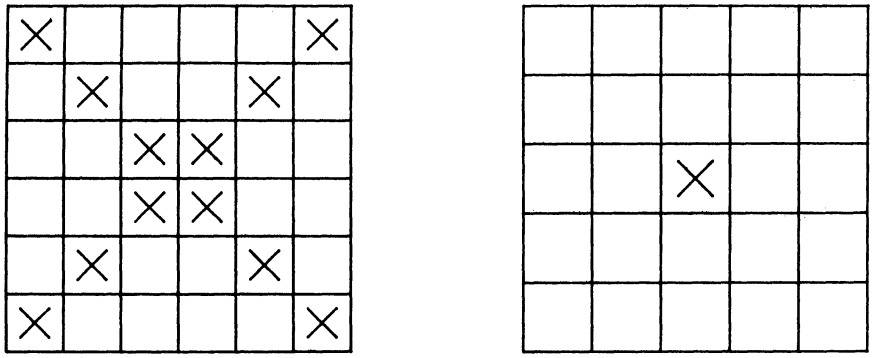
FIGURE 1

q-player positional game, our main problem of interest, known as the **strategy problem**, is: under optimal play, is the game a draw, and if not, which player wins? We restrict the strategy problem, however. Positional games with one player are trivial (the one player can always win if L is nonempty, except in certain ethnic jokes), and positional games with three or more players are too complicated for analysis (it's not clear, for example, how the rules should handle coalitions). Accordingly, for the rest of this paper, we consider only 2-player positional games.

From game theory [5] we know that for any finite 2-player perfect information game, either the first player can force a win, the second player can force a win, or each player can force a draw. Furthermore, for positional games the second player can't force a win, as the following argument shows. Suppose the second player *can* force a win; that is, suppose the second player has a strategy S that allows him to win no matter what the first player does. (Think of S as an oracle that tells the second player which point to choose, given the first player's choice.) The first player can then "steal" strategy S by choosing his first point at random and thereafter (a) following S, or (b) choosing another random point if S dictates he choose the already chosen random point. In a positional game, having an extra point can't possibly harm the first player, since it neither hinders the first player nor helps the second player in forming a line. (There is one apparent exception: having an extra point may leave no unchosen point to choose at the game's end—that is, it may prevent the first player from choosing the last point when the board is full. In this case, however, this last point must be the first player's random point; the first player must *already* have won, then.) Thus by following S the first player can always win, contradicting the assumption that the second player can always win. Therefore, the second player can't force a win in any positional game. Hence, for a given positional game, the strategy problem boils down to: can the first player force a win or can the second player force a draw?

The k^n tic-tac-toe games (henceforth referred to as the **k^n-games**) form a subclass of the positional games (see Gardner [10] and Berlekamp, Conway, and Guy [4] for discussions of tic-tac-toe and related games). We can view a game in this subclass as an n-dimensional hypercube with k cells on an edge, whose points P are the centers of these cells and whose lines L are the sets of k collinear points. Regular tic-tac-toe (FIGURE 1) is the 3^2-game and Qubic is the 4^3-game.

For any k^n-game, the number $|P|$ of points is k^n, and the number $|L|$ of lines is $[(k+2)^n - k^n]/2$. The expression for the number of points is obvious; the expression for the number of lines is elegantly proved by Moser [13] as follows. Consider a k^n hypercube H whose lines we



The points marked X are the strongest points in their respective games. In the 5^2-game there are $(3^n-1)/2=4$ lines containing the strongest point. In the 6^2-game there are $2^n-1=3$ lines containing each strongest point.

FIGURE 2

want to count. Embed H in a $(k+2)^n$ hypercube H' so that H' has a layer of cells on each side of H and let S be the shell of points in H' surrounding H. The points in S are those in H' that are not in H, so there are $(k+2)^n-k^n$ points in S. For any line l in H, its unique extension l' to H' (the only l' in H' containing all the points of l) contains exactly two points of S, one at either end of l'. Furthermore, for any point in S, exactly one line of H extends to it (this requires a little thought). Hence, the lines of H must number half the points of S, or $[(k+2)^n-k^n]/2$, as claimed. (Technically this expression for the number of lines fails for the 1^n-games if $n>1$. We can, however, revise our definition of a line to include a notion of direction, making the expression valid; we avoid the details of this revision since they aren't crucial.)

Our last property, given without proof, concerns a **strongest-point**—one contained in the most lines (see FIGURE 2). How many lines, denoted by SPL (strongest-point lines), contain such a point? If k is odd, there is a single strongest point; it is in the center of the hypercube. In this case

$$\text{SPL}=(3^n-1)/2 \quad \text{if } k \text{ is odd.}$$

If k is even, each point in a main diagonal is a strongest point (a main diagonal is a line containing opposite corner points of the hypercube). There are 2^{n-1} main diagonals; they are pairwise disjoint, so there are $k2^{n-1}$ strongest points. For each

$$\text{SPL}=2^n-1 \quad \text{if } k \text{ is even.}$$

We can now consider the strategy problem for the k^n-games, summarized in TABLE 1. We divide the k^n-games into three classes. Class 1 consists of those games for which draws are impossible even under nonoptimal play; that is, no draw position exists. This means that the first player can always force a win in these games, since, as we saw earlier, the second player can't. Class 2 consists of those games for which a draw position exists but for which the first player can nevertheless force a win. Class 3 consists of those games for which the second player can force a draw. Thus, under optimal play, games in classes 1 and 2 are first-player wins, and games in class 3 are draws.

In a 1963 paper, Hales and Jewett [**11**] show, using Hall's well-known marriage theorem [**12**], that the second player can force a draw if $k \geqslant 2\times\text{SPL}$; that is, if

$$k \geqslant 3^n-1 \quad \text{if } k \text{ is odd,}$$

or

$$k \geqslant 2^{n+1}-2 \quad \text{if } k \text{ is even.}$$

Thus, these k^n-games belong to class 3 (draws). (Their argument uses the following pairing strategy: there exists a set of mutually exclusive pairs of points such that a player can't win

Number of points in a line $k=$	$2^k=$	Dimension n $n=1$	$n=2$	$n=3$	$n=4$	$n=5$
1	2	1 1 1	1 4 4	1 13 13	1 40 40	1 121 121
2	4	2 1 1	4 6 3	8 28 7	16 120 15	32 496 31
3	8	3 1 1	tic 9 tac 8 toe 4	27 49 13	81 272 40	243 1441 121
4	16	4 1 1	16 10 3	64 Qubic 76 7	256 520 15	1024 3376 31
5	32	5 1 1	25 12 4	125 109 13	625 888 40	3125 6841 121
6	64	6 1 1	36 14 3	216 148 7	1296 1400 15	7776 12496 31
7	128	7 1 1	49 16 4	343 193 13	2401 2080 40	16807 21121 121
8	256	8 1 1	64 18 3	512 244 7	4096 2952 15	32768 33616 31
9	512	9 1 1	81 20 4	729 301 13	6561 4040 40	59049 51001 121
10	1024	10 1 1	100 22 3	1000 364 7	10000 5368 15	100000 74416 31
11	2048	11 1 1	121 24 4	1331 433 13	14641 6960 40	161051 105121 121
12	4096	12 1 1	144 26 3	1728 508 7	20736 8840 15	248832 144496 31
13	8192	13 1 1	169 28 4	2197 589 13	28561 11032 40	371293 194041 121
14	16384	14 1 1	196 30 3	2744 676 7	38416 13560 15	537824 255376 31
15	32768	15 1 1	225 32 4	3375 769 13	50625 16448 40	759375 330241 121

A summary of the strategy problem for the k^n-games, for small k and n. For each game, the table gives the classes of which the game is possibly a member (the games in the outlined area are unsolved—they are possibly members of both classes 2 and 3). The table also gives for each game, top to bottom, the number of points, the number of lines, and the number of strongest-point lines.

TABLE 1

unless he chooses both points of some pair; the second player prevents the first player from winning by choosing one point of a pair any time the first player chooses the other.)

Erdős and Selfridge [8] improve the Hales and Jewett result by showing (but not with a pairing strategy) that the second player can force a draw if $\mathrm{SPL}+|L|<2^k$; that is, if

$$(3^n-1)/2+\left[(k+2)^n-k^n\right]/2<2^k \quad \text{if } k \text{ is odd,}$$

or

$$2^n-1+\left[(k+2)^n-k^n\right]/2<2^k \quad \text{if } k \text{ is even.}$$

Hales and Jewett also show that, given a k, there exists an n_k such that whenever $n \geqslant n_k$ the k^n-game has no draw position; these k^n-games are thus in class 1 (wins). (They don't specify values for n_k; however, we know that the minimum $n_k=k$ for $k=1,2,3$, but the minimum $n_4 \neq 4$.)

Paul [18] adds to another Hales and Jewett result to show that whenever $k \geqslant n+1$, a draw position exists; these k^n-games, then, belong either to class 2 (wins) or class 3 (draws). Paul improves this result for $n \geqslant 4$, showing that a draw position exists if $k \geqslant n$. (For a complete discussion of draw position existence and related issues, see [15], [16], [17], [18].)

These results support several conjectures:

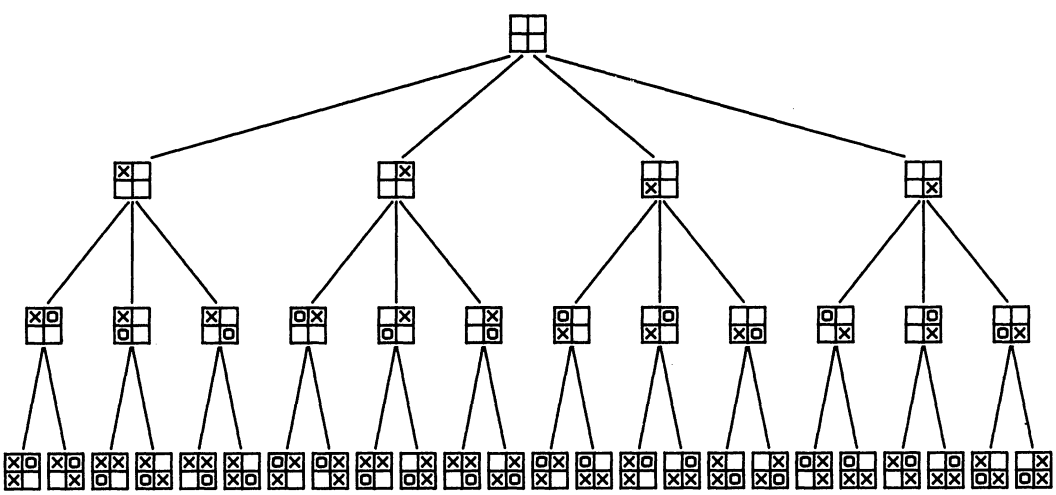
1. Hales and Jewett [11] conjecture that the second player can force a draw, because a pairing strategy exists, whenever there are at least twice as many points as lines (i.e., $k \geqslant 2/(2^{1/n}-1)$). (Proving this reduces to showing that a certain ratio, namely the number of lines in a subset of L divided by the number of points aggregately contained in this subset, is a maximum when this subset is L itself. Upon close inspection this ratio conjecture is very compelling, not only when a pairing strategy exists, but for any k^n-game.)
2. A modification of Gammill's conjecture [9] predicts that the second player can force a draw if and only if there are more points than lines (i.e., $k > 2/(3^{1/n}-1)$).
3. Citrenbaum [6] conjectures that the second player can force a draw if and only if there are more points in a line than there are dimensions (i.e., $k > n$).
4. A modification of conjecture 3 predicts that the second player can force a draw if and only if a draw position exists (i.e., it predicts that class 2 is empty).

These conjectures are consistent with all the information in TABLE 1, but differ on the strategy problem for the smallest unsolved member of the k^n-games, the 4^3-game, Qubic: conjecture 2 predicts the first player can force a win; conjectures 3 and 4 predict the second player can force a draw. (Conjecture 1 makes no prediction for Qubic.) The next section describes my solution to the strategy problem for Qubic.

Computer-aided solution to Qubic

The program I eventually used to solve the Qubic strategy problem evolved from an attempt to analytically prove Qubic a first-player win, which I "knew" was true from years of playing the game. Unable to find a purely analytic proof, I had to resort to a brute-force search, requiring a computer.

A brute-force search is based on a game tree, discussion of which demands some terminology. An ***r*-position** (sometimes just **position**) is an enumeration of the points of P each player has



The complete game tree for the 2^3-game.

FIGURE 3

chosen after r total moves. An r-position is at **level r** of the game tree. A **terminal position** is one for which the game has ended. A **game tree** is a representation of all positions reachable, subject to a given set of restrictions, from the 0-position (no points chosen). Thus, a path from the 0-position to a terminal position in the game tree represents a single game. A **brute-force search** (sometimes just **search**) of a game tree, then, is an examination of every position of the game tree.

This terminology applies to a general game tree. Next, we discuss three specific game trees: a complete game tree and two successive refinements of it.

A **complete game tree** is a representation of all positions reachable from the 0-position, subject to no restrictions (except of course that points be chosen legally). FIGURE 3 shows the complete game tree for the 2^2-game. A **naive brute-force search** is specifically a brute-force search of a complete game tree. (Abstractly, this entails examining all legally possible games.)

Using a naive brute-force search to determine the identity and location of the terminal positions in the complete game tree solves the strategy problem; that is, it tells us whether the first player can force a win or the second player can force a draw. A naive brute-force search for Qubic, however, becomes exponentially unreasonable. For example, after three total moves there are $64\times63\times62=249,984$ 3-positions in the complete game tree. Thus, I needed to refine the complete game tree, making the resulting search reasonable.

Although it's not clear that I made the brute-force search of Qubic reasonable (ten and a half months of brute forcing hardly seems reasonable), I *did* substantially reduce the size of the game tree. For example, instead of a quarter million 3-positions, my new game tree had only seven. I employed two processes to accomplish this reduction, corresponding to our two refinements of the complete game tree:

1. I restricted the set of first-player moves that the search examined (for a given position) to a single "winning" move (henceforth referred to as a **first-player move**).
2. I eliminated any position equivalent to one previously examined.

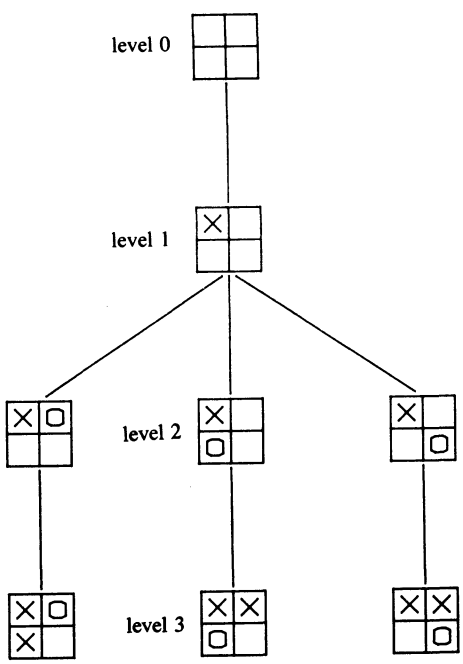


FIGURE 4

FIGURE 4 shows a first player game tree for the 2^2-game. There are only three terminal positions instead of the 24 for the complete game tree in FIGURE 3. Switching any two points leaves the set of lines unchanged in the 2^2-game; thus, any such switch can't change the outcome of any game. FIGURE 5 shows the result of switching p_2 and p_4. FIGURE 6 shows a distinct-position tree for the 2^2-game. Compare it to the trees in FIGURES 3 and 4.

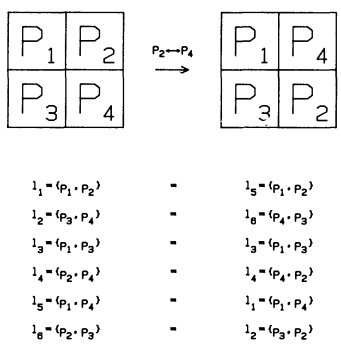


FIGURE 5

FIGURE 6

Process 1 followed from my belief that Qubic was a first-player win. I didn't need to search the complete game tree— I wasn't interested in how poorly the first player could play; I was interested only in his optimal play. Hence, to prove Qubic a first-player win, I needed only find a game tree, all of whose terminal positions were first-player wins, that placed no restrictions on the second-player moves. Thus, no matter what the second player did, the first player would win. This game tree, henceforth referred to as a **first-player game tree**, is our first refinement of the complete game tree. FIGURE 4 shows a first-player game tree for the 2^2-game. Process 1 reduced the number of 3-positions from 24 in the complete game tree to three in the first-player game tree. For Qubic this reduction was from 249,984 3-positions in the complete game tree to $1 \times 63 \times 1 = 63$ in the first-player game tree.

Process 2 involved recognizing equivalent positions. Intuitively, the first two positions on level 2 in FIGURE 4 are equivalent—they are mere reflections of each other. Further reflection reveals that these positions are equivalent to the third position on level 2. To see this, consider the following. Since *any* two points constitute a line in the 2^2-game, switching two points (or in fact any permutation of the points) leaves the set of lines unchanged (FIGURE 5 illustrates this, switching points p_2 and p_4). Thus, such a switch can't change the outcome of any game. Since the third position on level 2 is just a p_2—p_4 switch of the first position, it must be equivalent to the first (and therefore to the second). In fact, any one-to-one mapping of the set of four points onto

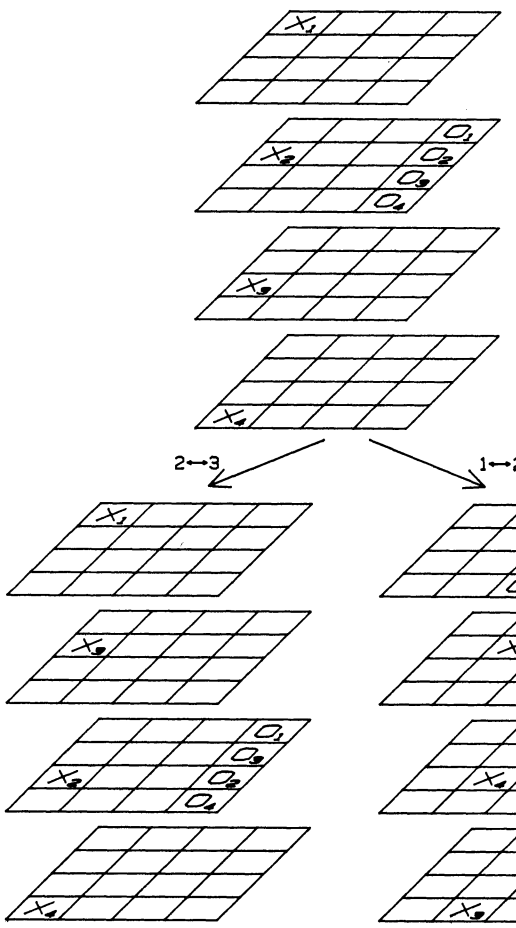


FIGURE 7

FIGURE 7 shows two unintuitive automorphisms for Qubic. Identify the points by ordered triples: X_1 in the top position is (1,1,1); O_3 is (2,3,4). The left automorphism switches coordinate-value 2 with coordinate-value 3. The right automorphism switches 1 with 2, and 3 with 4. Thus the automorphisms map X_1 from (1,1,1) to (1,1,1) and (2,2,2) respectively. They map O_3 from (2,3,4) to (3,2,4) and (1,4,3) respectively.

FIGURE 8 shows the two distinct 1-positions for Qubic. On the first player's first move, the 16 points marked X are equivalent to each other—these are the strongest points for Qubic (compare this to FIGURE 2). The remaining 48 points are equivalent to each other. Thus, the first player has a choice of just two distinct moves, yielding only two distinct 1-positions for Qubic. (Of course, Qubic's distinct-position tree contains only the strong one.)

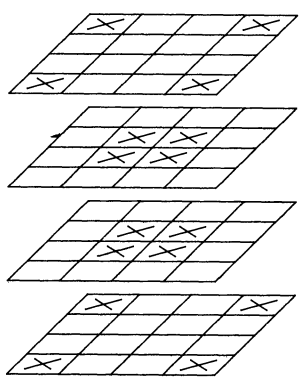


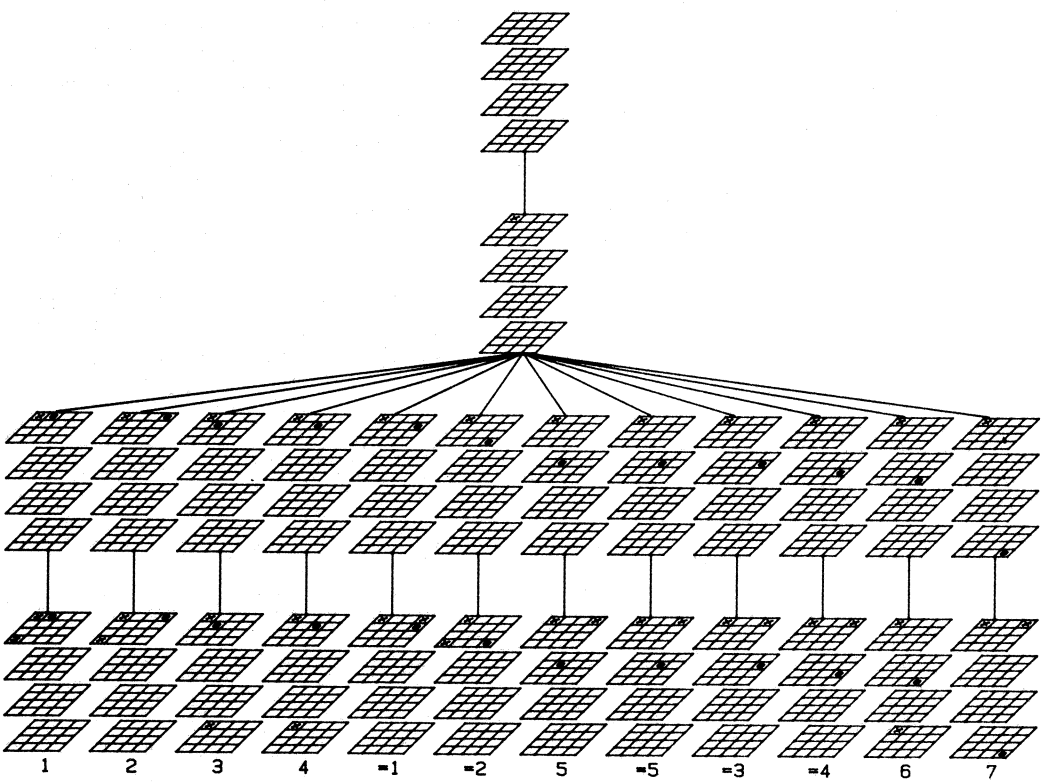
FIGURE 8

itself preserves the set of six lines, so there are $4! = 24$ transformations, or automorphisms, for the 2^2-game.

An **automorphism**, then, is a one-to-one mapping of the set of points onto itself that preserves the set of lines. Two positions are **equivalent** if they are automorphic images of each other; one of these positions is **redundant**. We call a first-player game tree with redundant positions removed a first-player distinct-position game tree, **distinct-position tree** for short. This game tree is our second refinement of the complete game tree. FIGURE 6 shows a distinct-position tree for the 2^2-game.

For Qubic, it turns out, there are 192 automorphisms, proved by Silver **[20]**. Just as some of the 24 automorphisms for the 2^2-game are unintuitive, so are most of Qubic's 192 automorphisms. In fact, only 48 are ordinary rotations or reflections of the cube; the remaining 144 are combinations of these 48 with at least one of the two automorphisms shown in FIGURE 7. These 192 automorphisms yield only two distinct 1-positions, as indicated in FIGURE 8.

These automorphisms also yield only $1 \times 12 = 12$ 2-positions in the distinct-position tree for Qubic. Thus, I reduced the number of 2-positions in Qubic's three specific game trees from $64 \times 63 = 4032$ in the complete game tree, to $1 \times 63 = 63$ in the first-player game tree, to $1 \times 12 = 12$ in the distinct-position tree. It seems, then, there should be $1 \times 12 \times 1 = 12$ 3-positions in Qubic's distinct-position tree. However, by judiciously choosing the first-player moves for the twelve 2-positions, combining processes 1 and 2, I rendered five of these twelve redundant; this completed the reduction of the 249,984 3-positions in the complete game tree to the seven 3-positions in the distinct-position tree. FIGURE 9 shows the top four levels of my distinct-position tree for Qubic.



Levels 0, 1, 2, and 3 of Qubic's distinct-position tree. There are only seven distinct 3-positions, reduced from twelve 2-positions by judiciously choosing first-player moves for these twelve positions.

FIGURE 9

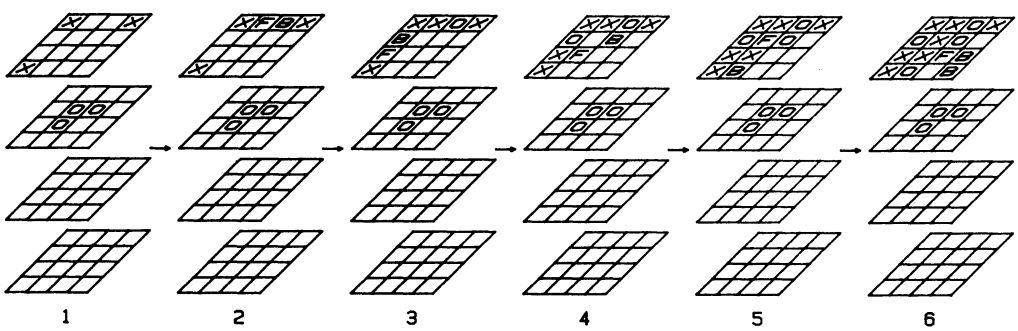
My problem, then, was reduced to finding a distinct-position tree for Qubic—a game tree, all of whose terminal positions were first-player wins, that made no restrictions on second-player moves, that restricted the first player to a single move for each position, and that contained no redundant positions. The only nonalgorithmic step in constructing such a tree was in finding the first-player moves. The rest—generating all possible second-player moves, removing redundant positions, and bookkeeping—was purely mechanical.

Initially I had planned to use a set of programmed heuristics to generate these first-player moves. ("Heuristics" comes from the Greek, *εὑρίσκειν*, to find [14].) For several weeks I experimented with heuristics based on, for example, the number of lines and planes each player controlled. Unfortunately, the best set of heuristics I could find generated some moves worse than those I, as an experienced human player, could generate. Although finding a good set of heuristics would have been an interesting Artificial Intelligence problem, it didn't seem fruitful for proving Qubic a first-player win. I therefore had to abandon having my program make all the first-player moves—I had to make some of the first-player moves myself. We call these **strategic** moves.

My program was very good (perfect), however, at making certain first-player moves: at blocking a second-player three-in-a-row, and at finding a forced sequence (explained shortly) if one existed. These two types of moves, called **tactical** moves, comprised all but 2929 (the strategic moves) of the more than one million first-player moves made in the search. Thus, the most promising approach to proving Qubic a first-player win had my program making the tactical first-player moves, at which programs tend to be good, and had me making the strategic first-player moves, at which experienced humans tend to be good. (Incidentally, chess too shows these differing strengths of humans and computer programs. The human advantage in strategy is currently great enough to overcome the computer advantage in tactics: experts and masters, strategists, still beat the best programs, tacticians. Though the day of computer supremacy (in chess) is approaching, it seems this supremacy will arise not through strategic ("smart") programs, but through "smart" programmers using brute-force tactics.)

The two processes (choosing a single first-player move and eliminating redundant positions) kept the distinct-position tree small at the top levels (through level 5); they did not, however, change Qubic's exponential nature. In fact, the 192 automorphisms were virtually useless in reducing the tree size below level 5. Hence, to prevent this approach, too, from blowing up, I needed a third process—one to limit the search at lower levels of the tree.

Employing the forced-sequence search, alluded to above, was that process. A **forced sequence** is a sequence of moves from a given position, in which Player O must continually block Player X's three-in-a-row until at some move he or she must simultaneously block two such threes-in-a-



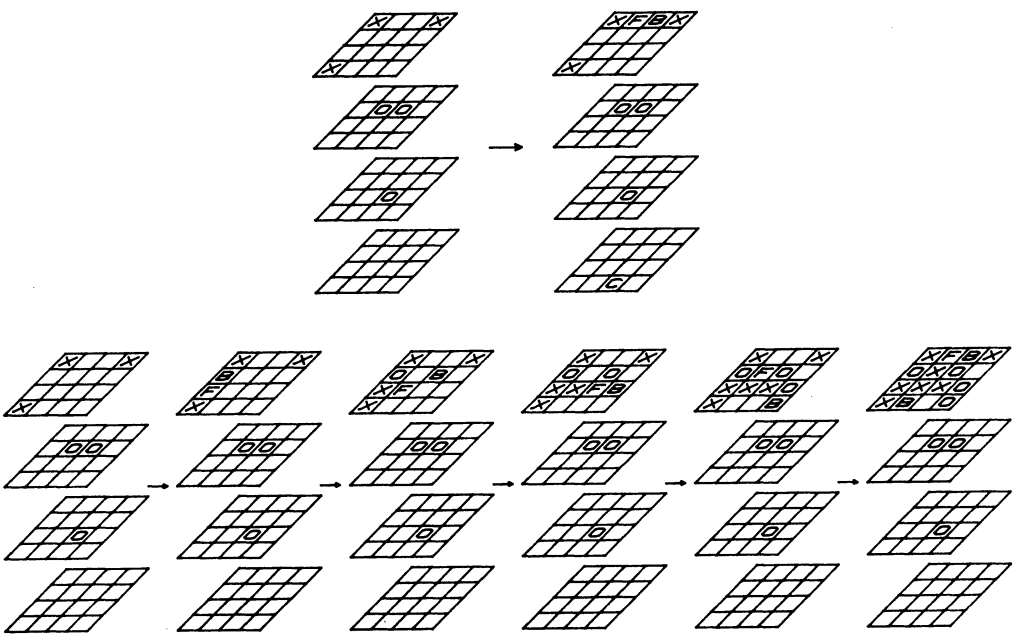
A typical Qubic forced sequence. Player X's moves at *F* from each position continually force Player O to block at *B*, until Player O must block two threes-in-a-row in position 6; since this is impossible, Player O loses.

FIGURE 10

row; this is impossible, so Player X wins. FIGURE 10 gives a typical example. (Note that a forced-sequence search is part of the overall search of the distinct-position tree.) A complication in finding a forced sequence may arise, however: Player O's block (of Player X's three-in-a-row) may give Player O a three-in-a-row, forcing Player X to block, typically ending the potential forced sequence. FIGURE 11 shows one such complication and its solution by reordering moves.

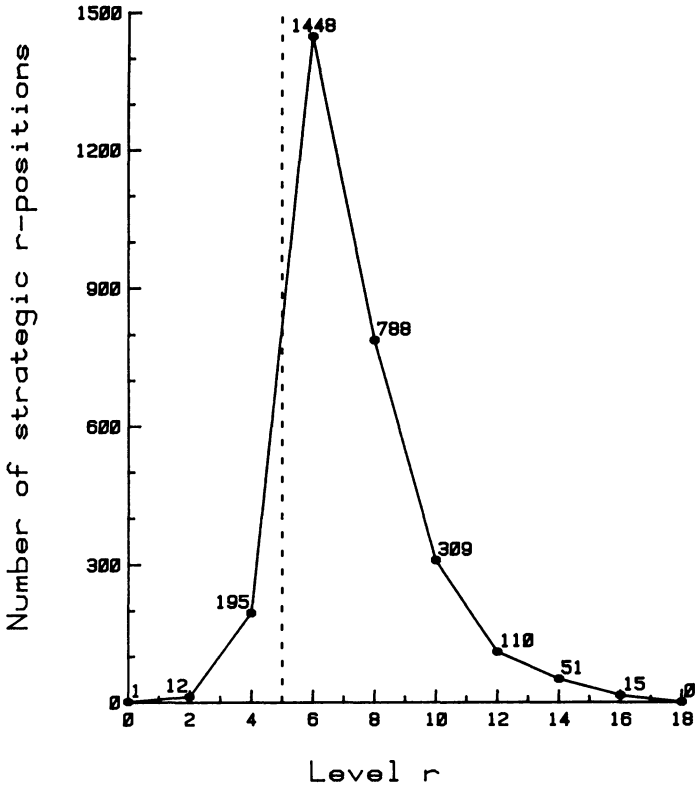
A forced-sequence search such as one shown in FIGURE 10 or 11 was very quick—such a search took a second or two of computer time. Some forced-sequence searches, however, took much longer, especially those for positions from which a forced sequence didn't exist. In fact, since the forced-sequence search had to find a forced sequence from a given position if one existed, to show that one did not exist from a given position, it had to examine *all* possible forced sequences. We saw that a naive brute-force search for Qubic (which examined the complete game tree) blew up because it had to examine *all* possible positions. Why, then, didn't the forced-sequence search blow up for some positions?

The answer has two parts. (Keep in mind that the forced-sequence search *was* at times precariously close to blowing up.) First: we saw that the naive brute-force search became a reasonable search when we refined the complete game tree to a first-player game tree; that is, we restricted one of the players (the first player in that case) to a single move for each position. For the forced-sequence search, the corresponding restriction was *implicit*; Player O being forced was implicitly restricted to a single move—to block the three-in-a-row or lose immediately. Thus the forced-sequence search, too, was reasonable. Second: the forced-sequence search was reasonable only if it was given a reasonable position from which to search; the positions from which it searched were reasonable because my strategic moves were good. The forced-sequence search, we saw, typically took only a second or two; however, for a few positions, it searched as long as half an hour. Hence, had my strategic moves been even slightly worse, the forced-sequence search would have blown up for a few positions.



Complications arise in some forced sequences. In the top sequence, if Player X tries to win as in FIGURE 10, Player O's first block at *B* produces three-in-a-row, forcing Player X to block at *C*, ending the potential forced sequence. Player X can avoid this by reordering moves as in the bottom sequence.

FIGURE 11



The distribution of the 2929 strategic *r*-positions in Qubic's distinct-position tree. (A strategic *r*-position is one from which I made a strategic move.) Level 6 contained almost half; there were no strategic positions below level 16, eight moves by each player. The forced-sequence search took effect after the dotted line.

FIGURE 12

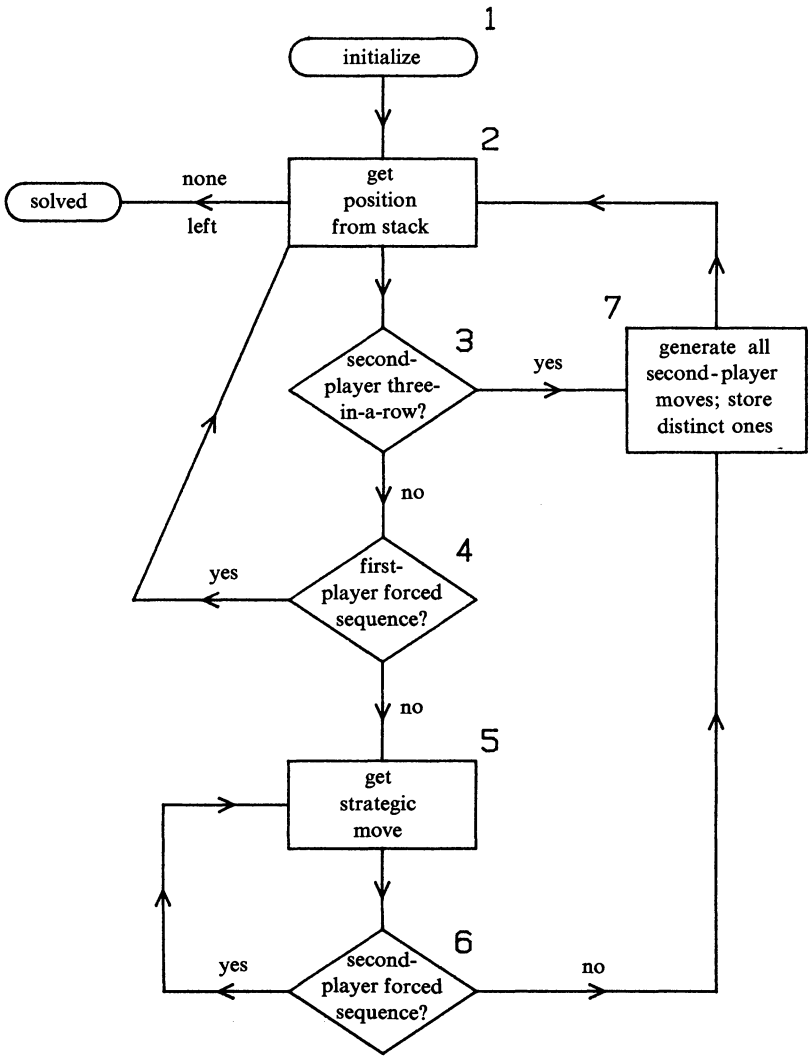
This explains why the forced-sequence search itself didn't blow up. It does not explain why the forced-sequence search kept the overall search of the distinct-position tree from blowing up below level 5. Very simply, the overall search didn't blow up below level 5 because most positions (roughly 98% of them) from which it was the first player's turn to move yielded a forced sequence (rendering further search from those positions unnecessary). To see this, consider an arbitrary *r*-position from which it was the first player's turn to move (that is, *r* was even). It consisted of $r/2$ first-player points, which were chosen to be good, and $r/2$ second-player points, which were arbitrary and therefore likely to be lousy. Such an arbitrary *r*-position was therefore likely to yield a first-player forced sequence. Thus, positions terminated rapidly at lower levels of the distinct-position tree, keeping its search from blowing up.

FIGURE 12 shows Qubic's exponential nature. Since a forced sequence could exist after three moves by each player (as was the case in FIGURE 10), the forced-sequence search took effect after the dotted line (level 5). The jump between level 4 and level 6 would have been astronomical (instead of just enormous) had I not used the forced-sequence search.

My program, then, used the following algorithm (outlined in FIGURE 13) to construct and search the distinct-position tree for Qubic:

Step 1. Initialize: put the 0-position into the tree and onto the stack of unexamined positions. Go to step 2.

Step 2. Get the next unexamined position from the stack. If none, terminate the algorithm, the game is solved; otherwise go to step 3 (start examining).



A flowchart for the Qubic algorithm.

FIGURE 13

Step 3. Check for a second-player three-in-a-row (step 6 insures at most one). If one exists, go to step 7; otherwise go to step 4.

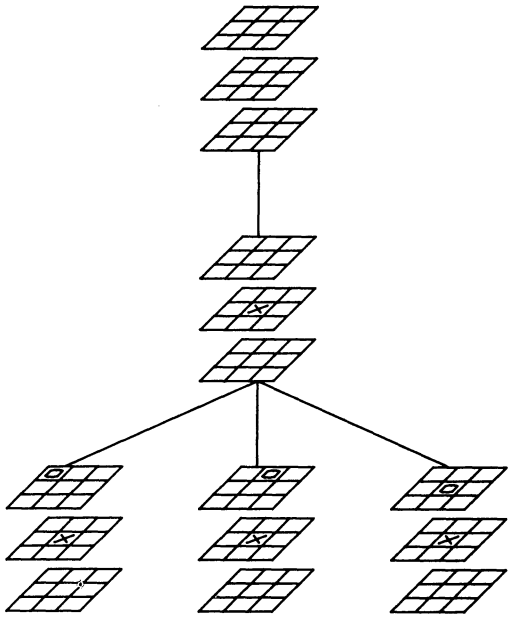
Step 4. Check for a first-player forced sequence. If one exists, add its positions to the tree and go back to step 2; otherwise go to step 5.

Step 5. Get a strategic move and go to step 6.

Step 6. Check for a second-player forced sequence. If one exists, complain and go back to step 5; otherwise go to step 7.

Step 7. Block a second-player three-in-a-row if one exists. Generate all possible second player moves and resulting positions. Remove redundant positions. Put the distinct positions into the tree and onto the unexamined-position stack. Go back to step 2.

To show the algorithm's simplicity, we use it to prove the trivial 3^3-game a first-player win. (Of course, we must change "three-in-a-row" to "two-in-a-row" for the 3^3-game.) FIGURE 14 gives the first three levels of the distinct-position tree we construct for this game.

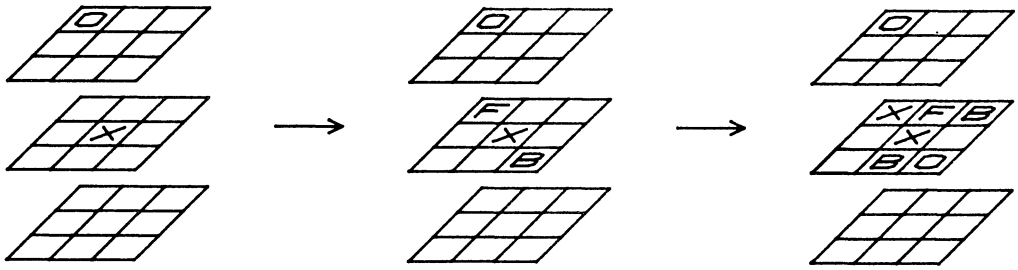


Levels 0, 1, and 2 of the distinct-position tree for the 3^3-game. Each position at level 2 yields a first-player forced sequence, so the 3^3-game is a first-player win.

FIGURE 14

1. Step 1—Put the 0-position into the tree and onto the unexamined-position stack.
2. Step 2—Get it from the stack.
3. Step 3—No second-player two-in-a-row.
4. Step 4—No first-player forced sequence.
5. Step 5—Get a strategic move. We know from the mathematical background section of the paper that the only strongest point for this game is the center point, so we choose it as our first (and only) strategic move.
6. Step 6—No second-player forced sequence.
7. Step 7—No second-player two-in-a-row. Generate all possible second-player moves and resulting positions. Remove redundant positions. Only three are distinct; put them into the tree and onto the stack.
8. Step 2—Get the first of these three from the stack.
9. Step 3—No second-player two-in-a-row exists.
10. Step 4—A first-player forced sequence exists (see FIGURE 15). Put its positions into the tree.
11. Step 2—Get the second of three from the stack.
12. Step 3—No second-player two-in-a-row exists.
13. Step 4—A first-player forced sequence exists, as before. Put its positions into the tree.
14. Step 2—Get the last of three from the stack.
15. Step 3—No second-player two-in-a-row exists.
16. Step 4—The same first-player forced sequence exists. Put its positions into the tree.
17. Step 2—The stack is empty; we have proved the 3^3-game a first-player win.

Proving the 3^3-game a first-player win, as above, would have taken a few seconds of computer time. Proving Qubic took a bit longer, about 1500 computer hours on the Yale



A first-player forced sequence for the first 2-position in the distinct-position tree for the 3^3-game; the other two 2-positions yield the same forced sequence.

FIGURE 15

Computer Science Department PDP10, about half of which were wasted—occasionally I chose a bad strategic move and had to backtrack; more frequent mishaps ranged from memory parity errors (hardware problems) to tape-drive malfunctions (hardware problems). Even 750 unwasted computer hours, however, overestimates the time required to solve the problem by more than an order of magnitude. Had I known in advance it would have taken so long, I would have made my program faster by, for example, writing it in a lower level language instead of in Algol. Furthermore, the set of 2929 strategic moves I produced was not minimal. Given the hours I was generally allowed to use the machine (midnight to 8 A.M.—but far from complaining, I thoroughly appreciate the access to the machine at all), and my usual physical state (tired—I had to write an alarm into my program to wake me up for the strategic moves), I know that a better choice of strategic moves would slightly reduce 2929. Of my solution:

> 'twas much deeper than a well, and wider than a church door;
> 'twas more than enough, but it served. [**19**]

Implications

The Qubic solution completes one more cell of TABLE 1: Qubic is the first known member of class 2, a k^n-game for which a draw position exists but for which the first player can nevertheless force a win; this disproves conjecture 4. This result also disproves conjecture 3—it shows that there exist first-player win k^n-games with $k>n$. The Qubic solution doesn't settle conjectures 1 or 2, although it supports conjecture 2. The strategy problem for the general k^n-game is still unsolved.

These are the solved problem's implications. The implications of the method of solution also merit discussion.

Why believe such a computer-aided result? After all, no one can possibly hand-check ten magnetic tapes of positions (body-check, maybe); no one can possibly prove that the 23 pages of Algol were correct, that the entire operating system was correct, and that the hardware functioned properly.

In an interesting, very readable, and highly recommended paper, DeMillo, Lipton, and Perlis [7] argue that a mathematical proof withstands a social process. The proof is believed, in successive stages, by the mathematician who discovered it, by colleagues, by journal reviewers, and finally by the general mathematical community. At each stage, the proof passes a test, gradually gaining acceptance.

The corresponding test for a computer proof such as mine (or Appel, Haken, and Koch's four-color theorem proof) is independent verification. For my proof, I gave Ken Thompson of Bell Laboratories a file of my 2929 strategic positions and moves. His *C* language program took "only" 50 hours on an Interdata 8/32 to verify the first-player win. In theory, his program did

exactly what mine did, except that his program took the strategic moves from a file rather than from a human. In practice, his program was very different; it did, however, prove the same result.

Such verifications, in fact, may have advantages over verifications of some mathematical proofs. If a mathematician makes a subtle error in constructing a proof, a second mathematician is likely to miss the subtlety in inspecting the proof—the second mathematician's line of thinking is partly constrained by the first mathematician's line of thinking. However, two programmers working independently will be less likely to make identical errors in writing their programs. That is, one proof construction together with its dependent inspection is more error prone than two independent proof constructions.

Furthermore, certain mathematical proofs are inherently more error prone than their computer counterparts, regardless of verification. A 100-page existence proof for a certain group will much likelier contain an error than will an instance of the group, which the program finds after hours of searching. Thus, there are problems for which a computer proof seems preferred.

Finally, combining the respective strengths of humans and computers, fully exploiting the resources of both, may be the most (perhaps the only) feasible approach for solving certain problems; respective human and computer limitations may render approaches relying on either alone insufficient. The four-color theorem and the strategy problem for Qubic were two such problems. There will likely be others. The computer will remain an important mathematical tool.

Acknowledgements

I am deeply indebted to S. C. Eisenstat and the Yale Computer Science Department for allowing me to complete this project. At commercial rates, ten cents per kilo-core-second, the project would have cost me about fifty million dollars. I am also very grateful to K. Thompson for verifying my proof, and to E. R. Berlekamp, M. Gardner, R. L. Graham, J. B. Kruskal, and A. M. Odlyzko for their assistance and general encouragement in making the result known. Finally, I thank all those whose comments on earlier drafts greatly improved this paper.

References

[1] K. Appel and W. Haken, The solution of the four-color-map problem, Sci. Amer., 237, 4(Oct. 1977)108–121.
[2] _____, Every planar map is four colorable, Part I: Discharging, Illinois J. Math., 21(1977)429–490.
[3] K. Appel, W. Haken, and J. Koch, Every planar map is four colorable, Part II: Reducibility, Illinois J. Math., 21(1977)491–567.
[4] E. R. Berlekamp, J. H. Conway, and R. K. Guy, Winning Ways, Academic Press, Chapter 22 (to appear).
[5] D. Blackwell and M. A. Girshick, Theory of Games and Statistical Decisions, Wiley, New York, 1954, p. 21.
[6] R. L. Citrenbaum, Efficient Representations of Optimal Solutions for a Class of Games, Doctoral Dissertation, Case Western Reserve University, Univ. Microfilms International, 1970, pp. 116–7.
[7] R. A. DeMillo, R. J. Lipton, and A. J. Perlis, Social processes and proofs of theorems and programs, Comm. ACM., 22(1979)271–280.
[8] P. Erdős and J. L. Selfridge, On a combinatorial game, J. Combin. Theory Ser. A, 14(1973)298–301.
[9] R. C. Gammill, An examination of tic-tac-toe like games, AFIPS Confer. Proc., 43(1974)349–355.
[10] M. Gardner, The Scientific American Book of Mathematical Puzzles and Diversions, Simon & Schuster, New York, 1959, pp. 37–46.
[11] A. W. Hales and R. I. Jewett, On regularity and positional games, Trans. Amer. Math. Soc., 106(1963)222–229.
[12] P. Hall, On representatives of subsets, J. London Math. Soc., 10(1935)26–30.
[13] L. Moser, Solution to problem E 773 [1947, 281], Amer. Math. Monthly, 55(1948)99.
[14] Oxford English Dictionary, Oxford Univ. Press, New York, 1971.
[15] J. L. Paul, The q-regularity of lattice points in R^n, Bull. Amer. Math. Soc., 81(1975)492–494.
[16] _____, Addendum, The q-regularity of lattice points in R^n, Bull. Amer. Math. Soc., 81(1975)1136.
[17] _____, Tic-tac-toe in n-dimensions, this MAGAZINE, 51(1978)45–49.
[18] _____, Partitioning the lattice points in R^n, J. Combin. Th. Ser. A, 26(1979)238–248.
[19] W. Shakespeare, Romeo and Juliet, III. i. 97–98.
[20] R. Silver, The group of automorphisms of the game of 3-dimensional ticktacktoe, Amer. Math. Monthly, 74(1967)247–254.