Games of No Chance
MSRI Publications
Volume **29**, 1996

Pentominoes: A First Player Win

HILARIE K. ORMAN

Abstract. This article reports on a search-based computer proof that the game of pentominoes is a first-player win. Three search strategies were used in this proof, with dramatically different effects on the running time of the search. The two most effective strategies are compared and discussed.

The Short History of Pentominoes

Pentominoes is a two-player game involving twelve pieces—the regular 5-ominoes shown in Figure 1—and an 8×8 board. Players alternate placing pieces on the board, covering whole squares and without overlap. The player who cannot make a move loses. The game was first proposed by Solomon W. Golomb in the mid-fifties. Martin Gardner [1959] popularized it, also citing Golomb's work [1954] about polyominoes and checkerboards. See also [Golomb 1962; 1965].

In 1971, it was suggested that the game could be solved by computer search [Beeler et al. 1972], but no attempts to implement such a solution are known to me. In 1975 I wrote a computer program that played pentominoes, and used a PDP-11/45 computer to investigate the feasibility of a complete solution. Although the program was an excellent player against human opponents, at that time a complete solution was unattainable.

Today, high-speed workstations have changed the picture. A new search program exhaustively examined the game subtrees arising from certain two first moves. One of the moves was proved to be a win, and this assertion was verified by an independent program. The winning move was determined in about two weeks of execution time on a 64-bit 175 MHz DEC Alpha processor. The verification was done on a Sun IPC Sparcstation in about five days.

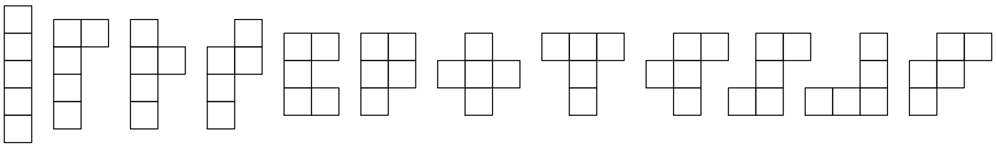

Figure 1. The twelve regular pentominoes.

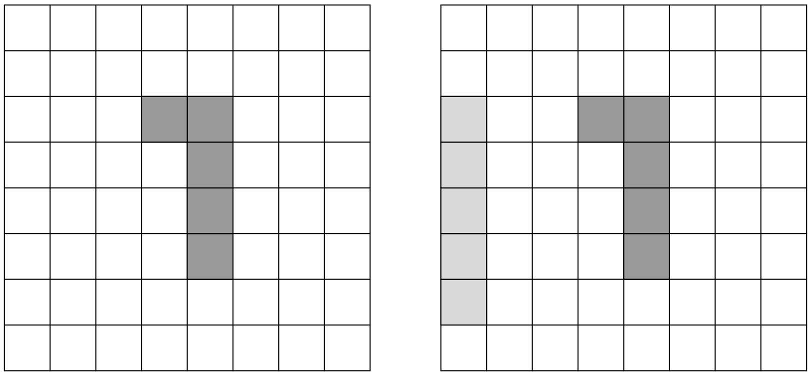

Figure 2. Left: One of the first-player moves that allows the minimum number of responses. Right: That move turns out to be losing, if the second player counters as shown.

The Search Begins

The computer program used to search the game tree is written in C and uses a simple backtracking search method. The moves and board state are represented as bit vectors. At the beginning of the game all possible moves are in the legal move list. At each ply, the legal move list is reduced, eliminating moves that are no longer possible. If all moves at depth n are losses, the value Win is returned to depth $n-1$. If some move at depth n evaluates to Win, the value Loss is returned to depth $n-1$.

A Losing Move. At the start of the game, there are 2308 possible moves, or 296 when symmetries are discounted. After the first move there are between 1181 and about 2000 replies. The search was originally conducted for one of the optimally restrictive moves, using the long "L" piece (Figure 2, left). There are 1181 replies to this move.

All replies to this first move were analyzed, the work being divided among several computers: an Intel Paragon, six Hewlett-Packard 720's, and four DEC RS3000 model 600 (Alpha) machines. The program ran for about two months. Of the 1181 replies, exactly two of them refute the opening move. In addition to the reply shown in Figure 2 (right), the straight piece can be moved down one square.

It is interesting to note the importance of the straight piece in this. A possibly related fact is that all solutions for packing all twelve pentominoes onto the board involve placing the this piece along an edge.

While this program was running, a few inefficiencies in its implementation were discovered. The program always sorted the move list by the number of replies available; in effect, one move lookahead was used. This is useful if there are a large number of moves available, but it is wasteful if there are few. Also,

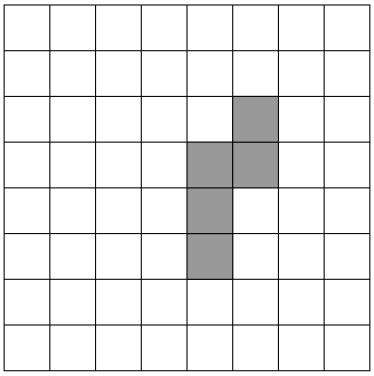

Figure 3. A winning move.

the program did not take advantage of the 64-bit wordsize of the DEC Alpha machines. With the 64-bit words, testing a possible move takes only one instruction. The program was rewritten to correct both deficiencies before proceeding. The sorting was restricted to the first few plies.

A Winning Move. Using the new program, the search was restarted with the opening move shown in Figure 3, one of the second most restrictive (1197 replies). Using two DEC Alphas, almost all replies were examined in two weeks of running time. The remaining replies were examined using other machines. This provided strong evidence that the move illustrated is a win for the first player.

To validate the move, Richard Schroeppel wrote an independent program to check the claimed win. The original program recorded the winning third-ply move that it found for each second-ply move, and the checking program took as input three plies and then solved the game with the usual backtracking search. The checking program ran on a Sun IPC Sparcstation for about five days.

Statistics on Strategies

The solving program and the checking program kept statistics about the amount of effort expended for each validation, which revealed interesting facts about the game tree. The most important contributor to the speed of the program is the ability to choose a winning move for the first player quickly at each first-player ply. Based on examining two opening moves, we might conjecture that about half of the possible opening moves are wins for the first player. But the fact that the first move is only very narrowly defeated by the second player indicates that it is unreasonable to expect the game tree to be well balanced.

The statistics recorded were the number of third-ply moves that were examined before finding a winning move for the first player, and the total number of board positions examined at the third and fourth plies.

lower limit of bucket	1	2	2^2	2^3	2^4	2^5	2^6	2^7	2^8
frequency	121	235	252	181	198	134	80	17	1

Table 1. Distribution of third-ply moves examined by the solving program.

Overall, the solving program examined 22 billion board positions. Empirical evidence indicates that a typical game involves ten moves; it is known that the smallest possible number of moves is five, and the largest is twelve [Gardner 1959]. On the average, for each second-ply move, the program examined eighteen third-ply moves for the first player before finding a winning move. A good strategy for selecting the third-ply move is the key to minimizing the running time of the program.

Table 1 shows the distribution of third-ply moves examined by the solving program. Each column represents a bucket (range of trials); the bottom row gives the number of times that a winning move was found using a number of trials in the range. Although the program often found a win within seven tries (the first three buckets), it sometimes had to examine nearly half the available moves. The sum of the entries in the bottom row is slightly greater than the number of replies (1197) because there was some overlap in the ranges examined by the computers running the solving program.

Table 2 shows the distribution of number of board positions examined after each third-ply move. Each column represents a range of board positions, and the two bottom rows give the number of instances of counts in this range for the solving and checking programs. For example, the checking program needed to examine fewer than 1,048,576 positions in four cases; the solving program never examined this few. (Again, the counts reflect some overlap in the positions examined.)

It is interesting to note that the checking program did less work than the solving program as measured by the number of board positions examined. This means that it is more effective at finding a winning move for the first player. This was surprising, because the checking program does not have the moves sorted by number of replies; it tries the first piece at all possible board locations, then the next piece, etc., in a fixed pattern (pruned at each ply by eliminating illegal moves). This latter algorithm avoids some of the very long searches done by the first method. A third simple search strategy is to try each piece at board position $(0, 0)$, then each piece at position $(0, 1)$, etc. This turns out to be at

lower limit of bucket	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}	2^{25}	2^{26}	2^{27}
solver frequency		5	41	256	481	282	127	24	3
checker frequency	4	9	58	289	527	279	49	3	

Table 2. Distribution of total evaluations for the solving and checking programs, which used different strategies.

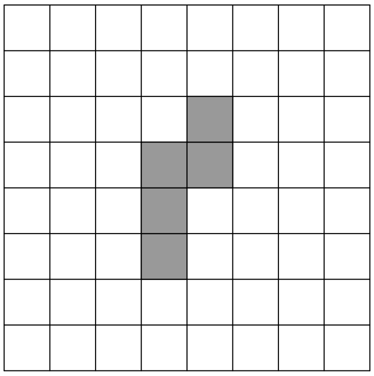

Figure 4. Another winning move, suggested by Golomb.

least ten times slower than the first two methods, and it is definitely not suitable for solving the game.

There are two plausible explanations for the checking program's search strategy being faster than that of the solving program. One reason might be that, in general, occupying a critical region of the board is more important than which piece is used for the occupation. The checking program does this by attempting to place pieces quickly without regard for which piece it is placing. A second explanation is based on noting that the program tries the thin pieces (having a linear block of four squares) first. The key to the game might be the placement of these pieces; the importance of the straight piece in refuting the most restrictive opening move is corroborating evidence.

Another Winning Move

Golomb (personal communication) suggested another winning move, one that comes closest to dividing the board symmetrically (Figure 4). This move indeed survives examination by the checking program. Schroeppel's validating program has not been used, because experience in double-checking the two moves described above has given us confidence in the first program.

The strategy of dividing the board symmetrically may be a sound one, and it would be interesting to see if it applies through the game tree.

Other Problems

It should be possible to determine all opening moves that are first-player wins, thus determining how much of an advantage the first player has. This task becomes much more difficult as less restrictive opening moves are examined. However, the checking program's strategy for move selection might offset this effect enough to make the overall running time reasonable.

A computationally challenging problem is to solve the similar game involving 35 hexominoes and a 15×15 board.

References

[Beeler et al. 1972] M. Beeler, R. Gosper, and R. Schroeppel, "HAKMEM", Technical Report Memo 239, MIT Artificial Intelligence Laboratory, 1972.

[Gardner 1959] M. Gardner, *Mathematical Puzzles and Diversions*, Simon and Schuster (sixth edition), 1959.

[Golomb 1954] S. W. Golomb, "Checker boards and polyominoes", *Amer. Math. Monthly* **61** (Dec. 1954), 672–682.

[Golomb 1962] S. W. Golomb, "General theory of polyominoes", *Recreational Math. Mag.* **4** (Aug. 1961), 3–12; **5** (Oct. 1961), 3–12 (see also Notes 12–14); **6** (Dec. 1961), 3–20; **8** (Apr. 1962), 7–16.

[Golomb 1965] S. W. Golomb, *Polyominoes*, Scribner, New York, 1965; second edition by Princeton Univ. Press, Princeton, 1994.

Hilarie K. Orman
1942 W. Camino Bajio
Tucson AZ, 85737
ho@cs.arizona.edu