with exactly \( n \) \( X \)'s and no \( O \)'s. Similarly, \( O \) is the number of rows, columns, or diagonals with just \( n \) \( O \)'s. The utility function assigns +1 to any position with \( X = 1 \) and -1 to any position with \( O = 1 \). All other terminal positions have utility 0. For nonterminal positions, we use a linear evaluation function defined as

\[
Eval(s) = 3X2(s) + X1(s) - 3O2(s) - O1(s)
\]

a. Approximately how many possible games of tic-tac-toe are there?

b. Show the whole game tree starting from an empty board down to depth 2 (i.e., one \( X \) and one \( O \) on the board), taking symmetry into account.

c. Mark on your tree the evaluations of all the positions at depth 2.

d. Using the minimax algorithm, mark on your tree the backed-up values for the positions at depths 1 and 0, and use those values to choose the best starting move.

e. Circle the nodes at depth 2 that would not be evaluated if alpha—beta pruning were applied, assuming the nodes are generated in the optimal order for alpha—beta pruning.

5.10 Consider the family of generalized tic-tac-toe games, defined as follows. Each particular game is specified by a set \( S \) of squares and a collection \( W \) of winning positions. Each winning position is a subset of \( S \). For example, in standard tic-tac-toe, \( S \) is a set of 9 squares and \( W \) is a collection of 8 subsets of \( W \): the three rows, the three columns, and the two diagonals. In other respects, the game is identical to standard tic-tac-toe. Starting from an empty board, players alternate placing their marks on an empty square. A player who marks every square in a winning position wins the game. It is a tie if all squares are marked and neither player has won.

a. Let \( N = |S| \), the number of squares. Give an upper bound on the number of nodes in the complete game tree for generalized tic-tac-toe as a function of \( N \).

b. Give a lower bound on the size of the game tree for the worst case, where \( W = \{ \} \).

c. Propose a plausible evaluation function that can be used for any instance of generalized tic-tac-toe. The function may depend on \( S \) and \( W \).

d. Assume that it is possible to generate a new board and check whether it is a winning position in 100N machine instructions and assume a 2 gigahertz processor. Ignore memory limitations. Using your estimate in (a), roughly how large a game tree can be completely solved by alpha—beta in a second of CPU time? a minute? an hour?

5.11 Develop a general game-playing program, capable of playing a variety of games.

a. Implement move generators and evaluation functions for one or more of the following games: Kalah, Othello, checkers, and chess.

b. Construct a general alpha—beta game-playing agent

c. Compare the effect of increasing search depth, improving move ordering, and improving the evaluation function. How close does your effective branching factor come to the ideal case of perfect move ordering?

d. Implement a selective search algorithm, such as B* (Berliner, 1979), conspiracy number search (McAllester, 1988), or MGIS* (Russell and Wefald, 1989) and compare its performance to A*.
Describe how the minimax and alpha-beta algorithms change for two-player, non-zero-sum games in which each player has a distinct utility function and both utility functions are known to both players. If there are no constraints on the two terminal utilities, is it possible for any node to be pruned by alpha-beta? What if the player’s utility functions on any state differ by at most a constant $k$, making the game almost cooperative?

Develop a formal proof of correctness for alpha-beta pruning. To do this, consider the situation shown in Figure 5.18. The question is whether to prune node $n_j$, which is a max-node and a descendant of node $n_1$. The basic idea is to prune it if and only if the minimax value of $n_j$ can be shown to be independent of the value of $n_j$.

- Mode $n_1$ takes on the minimum value among its children: $p_1 = \min(p_2, p_3, \ldots, p_n)$. Find a similar expression for $p_2$ and hence an expression for $n_j$ in terms of $n_j$.
- Let $l_i$ be the minimum (or maximum) value of the nodes to the left of node $n_j$ at depth $i$. Similarly, let $r_i$ be the minimum (or maximum) value of the unexplored nodes to the right of $n_j$ at depth $i$. Rewrite your expression for $p_i$ in terms of the $l_i$ and $r_i$ values.
- Now reformulate the expression to show that in order to affect $n_j$, $p_j$ must not exceed a certain bound derived from the $l_i$ values.
- Repeat this process for the case when $n_j$ is a min-node.

Prove that alpha-beta pruning takes time $O(2m^2)$ with optimal move ordering, where $m$ is the maximum depth of the game tree.

Suppose you have a chess program that can evaluate 10 million nodes per second. Decide on a compact representation of a game state for storage in a transposition table. About how many entries can you fit in a 2-gigabyte in-memory table? Will that be enough for the