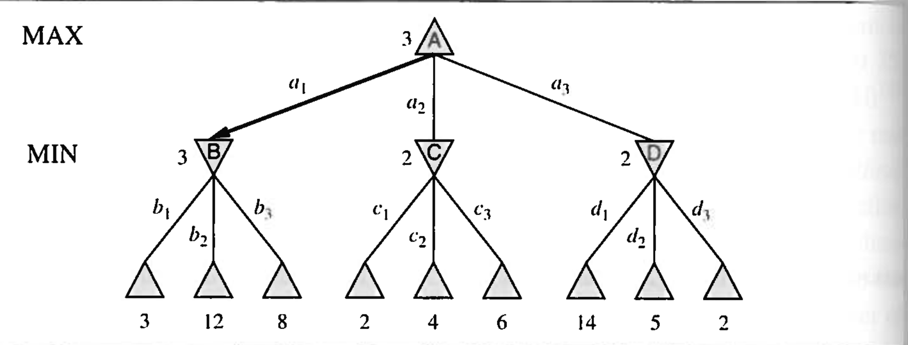

Figure 5.2 A two-ply game tree. The △ nodes are "MAX nodes," in which it is MAX's turn to move, and the ▽ nodes are "MIN nodes." The terminal nodes show the utility values for MAX; the other nodes are labeled with their minimax values. MAX's best move at the root is a_1, because it leads to the state with the highest minimax value, and MIN's best reply is b_1, because it leads to the state with the lowest minimax value.

MIN, then MAX's moves in the states resulting from every possible response by MIN to *those* moves, and so on. This is exactly analogous to the AND–OR search algorithm (Figure 4.11) with MAX playing the role of OR and MIN equivalent to AND. Roughly speaking, an optimal strategy leads to outcomes at least as good as any other strategy when one is playing an infallible opponent. We begin by showing how to find this optimal strategy.

Even a simple game like tic-tac-toe is too complex for us to draw the entire game tree on one page, so we will switch to the trivial game in Figure 5.2. The possible moves for MAX at the root node are labeled a_1, a_2, and a_3. The possible replies to a_1 for MIN are b_1, b_2, b_3, and so on. This particular game ends after one move each by MAX and MIN. (In game parlance, we say that this tree is one move deep, consisting of two half-moves, each of which is called a **ply**.) The utilities of the terminal states in this game range from 2 to 14.

PLY

MINIMAX VALUE

Given a game tree, the optimal strategy can be determined from the **minimax value** of each node, which we write as MINIMAX(n). The minimax value of a node is the utility (for MAX) of being in the corresponding state, *assuming that both players play optimally* from there to the end of the game. Obviously, the minimax value of a terminal state is just its utility. Furthermore, given a choice, MAX prefers to move to a state of maximum value, whereas MIN prefers a state of minimum value. So we have the following:

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$

Let us apply these definitions to the game tree in Figure 5.2. The terminal nodes on the bottom level get their utility values from the game's UTILITY function. The first MIN node, labeled B, has three successor states with values 3, 12, and 8, so its minimax value is 3. Similarly, the other two MIN nodes have minimax value 2. The root node is a MAX node; its successor states have minimax values 3, 2, and 2; so it has a minimax value of 3. We can also identify

the **minimax decision** at
the state with the highes
This definition o
maximizes the *worst-c*
easy to show (Exerci
opponents may do b
against optimal op

5.2.1 The mi

The **minimax**
It uses a simp
implementir
of the tree.
unwinds.
left node
8, respe
up value o
Finally, we take the
The minimax algorit
If the maximum depth of the tree i
time complexity of the minimax algorithm is
algorithm that generates all actions at once, or $O(m$
one at a time (see page 87). For real games, of course, the
but this algorithm serves as the basis for the mathematical analysis
practical algorithms.

5.2.2 Optimal decisions in multiplayer games

Many popular games allow more than two players. Let us examine how to extend the mi idea to multiplayer games. This is straightforward from the technical viewpoint, but some interesting new conceptual issues.

First, we need to replace the single value for each node with a *vector* of values example, in a three-player game with players A, B, and C, a vector $\langle v_A, v_B, v_C \rangle$ is asso with each node. For terminal states, this vector gives the utility of the state from each pl viewpoint. (In two-player, zero-sum games, the two-element vector can be reduced to a value because the values are always opposite.) The simplest way to implement this is to the UTILITY function return a vector of utilities.

Now we have to consider nonterminal states. Consider the node marked X in the tree shown in Figure 5.4. In that state, player C chooses what to do. The two choice to terminal states with utility vectors $\langle v_A = 1, v_B = 2, v_C = 6 \rangle$ and $\langle v_A = 4, v_B = 2, v_C$ Since 6 is bigger than 3, C should choose the first move. This means that if state X is rea subsequent play will lead to a terminal state with utilities $\langle v_A = 1, v_B = 2, v_C = 6 \rangle$. H the backed-up value of X is this vector. The backed-up value of a node n is always the