TERMINAL TEST

- A **terminal test**, which determines when the game is over. States where the game has ended are called **terminal states**.

PAYOFF FUNCTION

- A **utility function** (also called a **payoff function**), which gives a numeric value for the outcome of a game. In chess, the outcome is a win, loss, or draw, which we can represent by the values +1, -1, or 0. Some games have a wider variety of possible outcomes; for example, the payoffs in backgammon range from +192 to -192.

STRATEGY

If this were a normal search problem, then all MAX would have to do is search for a sequence of moves that leads to a terminal state that is a winner (according to the utility function), and then go ahead and make the first move in the sequence. Unfortunately, MIN has something to say about it. MAX therefore must find a **strategy** that will lead to a winning terminal state regardless of what MIN does, where the strategy includes the correct move for MAX for each possible move by MIN. We will begin by showing how to find the optimal (or rational) strategy, even though normally we will not have enough time to compute it.

Figure 5.1 shows part of the search tree for the game of Tic-Tac-Toe. From the initial state, MAX has a choice of nine possible moves. Play alternates between MAX placing X's and MIN placing O's until we reach leaf nodes corresponding to terminal states: states where one player has three in a row or all the squares are filled. The number on each leaf node indicates the utility value of the terminal state from the point of view of MAX; high values are assumed to be good for MAX and bad for MIN (which is how the players get their names). It is MAX's job to use the search tree (particularly the utility of terminal states) to determine the best move.

PLY

Even a simple game like Tic-Tac-Toe is too complex to show the whole search tree, so we will switch to the absolutely trivial game in Figure 5.2. The possible moves for MAX are labelled A_1, A_2, and A_3. The possible replies to A_1 for MIN are A_{11}, A_{12}, A_{13}, and so on. This particular game ends after one move each by MAX and MIN. (In game parlance, we say this tree is one move deep, consisting of two half-moves or two **ply**.) The utilities of the terminal states in this game range from 2 to 14.

MINIMAX

The **minimax** algorithm is designed to determine the optimal strategy for MAX, and thus to decide what the best first move is. The algorithm consists of five steps:

- Generate the whole game tree, all the way down to the terminal states.
- Apply the utility function to each terminal state to get its value.
- Use the utility of the terminal states to determine the utility of the nodes one level higher up in the search tree. Consider the leftmost three leaf nodes in Figure 5.2. In the $\bigtriangledown$ node above it, MIN has the option to move, and the best MIN can do is choose A_{11}, which leads to the minimal outcome, 3. Thus, even though the utility function is not immediately applicable to this $\bigtriangledown$ node, we can assign it the utility value 3, under the assumption that MIN will do the right thing. By similar reasoning, the other two $\bigtriangledown$ nodes are assigned the utility value 2.
- Continue backing up the values from the leaf nodes toward the root, one layer at a time.
- Eventually, the backed-up values reach the top of the tree; at that point, MAX chooses the move that leads to the highest value. In the topmost $\bigtriangleup$ node of Figure 5.2, MAX has a choice of three moves that will lead to states with utility 3, 2, and 2, respectively. Thus, MAX's best opening move is A_1. This is called the **minimax decision**, because it maximizes the utility under the assumption that the opponent will play perfectly to minimize it.

MINIMAX DECISION