Games, unlike most of the toy problems studied in Chapter 3, are interesting because they are too hard to solve. For example, chess has an average branching factor of about 35, and games often go to 50 moves by each player, so the search tree has about $35^{50}$ or $10^{151}$ nodes (although the search graph has "only" about $10^{40}$ distinct nodes). Games, like the real world, therefore require the ability to make some decision even when calculating the optimal decision is infeasible. Gaines also penalize inefficiency severely. Whereas an implementation of $A'$ search that is half as efficient will simply take twice as long to run to completion, a chess program that is half as efficient in using its available time probably will be beaten into the ground, other things being equal. Game-playing research has therefore spawned a number of interesting ideas on how to make the best possible use of time.

We begin with a definition of the optimal move and an algorithm for finding it. We then look at techniques for choosing a good move when time is limited. Pruning allows us to ignore portions of the search tree that make no difference to the final choice, and heuristic evaluation functions allow us to approximate the true utility of a state without doing a complete search. Section 5.5 discusses games such as backgammon that include an element of chance; we also discuss bridge, which includes elements of imperfect information because not all cards are visible to each player. Finally, we look at how state-of-the-art game-playing programs fare against human opposition and at directions for future developments.

We first consider games with two players, whom we call MAX and MIN for reasons that will soon become obvious. MAX moves first, and then they take turns moving until the game is over. At the end of the game, points are awarded to the winning player and penalties are given to the loser. A game can be formally defined as a kind of search problem with the following elements:

- $S_0$: The initial state, which specifies how the game is set up at the start.
- $\text{PLAYER}(s)$: Defines which player has the move in a state.
- $\text{ACTIONS}(s)$: Returns the set of legal moves in a state.
- $\text{RESULT}(s, a)$: The transition model, which defines the result of a move.
- $\text{TERMINAL-TEST}(s)$: A terminal test, which is true when the game is over and false otherwise. States where the game has ended are called terminal states.
- $\text{UTILITY}(s, p)$: A utility function (also called an objective function or payoff function), defines the final numeric value for a game that ends in terminal state $s$ for a player $p$. In chess, the outcome is a win, loss, or draw, with values $+1, 0, \text{ or } -1$. Some games have a wider variety of possible outcomes; the payoffs in backgammon range from 0 to +192. A zero-sum game is (confusingly) defined as one where the total payoff to all players is the same for every instance of the game. Chess is zero-sum because every game has payoff of either $0 + 1, 1 + 0 \text{ or } x + 0$.

The initial state, ACTIONS function, and RESULT function define the game tree for the game—a tree where the nodes are game states and the edges are moves. Figure 5.1 shows part of the game tree for tic-tac-toe (noughts and crosses). From the initial state, MAX has nine possible moves. Play alternates between MAX placing an X and MIN placing an 0.
until we reach leaf nodes corresponding to terminal states such that 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).

For tic-tac-toe the game tree is relatively small—fewer than $9! = 362,880$ terminal nodes. But for chess there are over $10^{40}$ nodes, so the game tree is best thought of as a theoretical construct that we cannot realize in the physical world. But regardless of the size of the game tree, it is MAX’s job to search for a good move. We use the term search tree for a tree that is superimposed on the full game tree, and examines enough nodes to allow a player to determine what move to make.

![Figure 5.1 A (partial) game tree for the game of tic-tac-toe. The top node is the initial state, and MAX moves first, placing an x in an empty square. We show part of the tree, giving alternating moves by MIN (0) and MAX (X), until we eventually reach terminal states, which can be assigned utilities according to the rules of the game.](image)

### 5.2 OPTIMAL DECISIONS IN GAMES

In a normal search problem, the optimal solution would be a sequence of actions leading to a goal state—a terminal state that is a win. In adversarial search, MIN has something to say about it. MAX therefore must find a contingent strategy, which specifies MAX’s move in the initial state, then MAX’s moves in the states resulting from every possible response by