6.1
Figure S6.1 shows the game tree, with the evaluation function values below the terminal nodes and the backed-up values to the right of the non-terminal nodes. The values imply that the best starting move for X is to take the center. The terminal nodes with a bold outline are the ones that do not need to be evaluated, assuming the optimal ordering.

![Game Tree Diagram]

Figure S6.1 Part of the game tree for tic-tac-toe, for Exercise 6.1.

6.5
The derivation below leads directly to a definition of $\alpha$ and $\beta$ values. The notation $n_i$ refers to (the value of) the node of depth $i$ on the path from the root to the leaf node $n_j$. Nodes $n_{i1}...n_{ih}$ are the siblings of node $i$.

a) We can write $n_2 = \max(n_1, n_{31}, ..., n_{3h})$, giving
$$n_i = \min(\max(n_{31}, n_{33}, ..., n_{3h}), n_{21}, ..., n_{2h}).$$
Then $n_2$ can be similarly replaced, until we have an expression containing $n_j$ itself.

b) In terms of the $l$ and $r$ values, we have
$$n_i = \min(l_2, \max(l_3, n_3, r_3), r_2).$$
Again, $n_2$ can be expanded out down to $n_j$. The most deeply nested term will be
$$\min(l_2, n_3, r_3).$$

c) If $n_j$ is a max node, then the lower bound on its value only increases as its successors are evaluated. Clearly, if it exceeds $l_j$ it will have no further effect on $n_i$. By extension, if it exceeds $\min(l_2, l_4, ..., l_j)$ it will have no effect. Thus, by keeping track of this value we can decide when to prune $n_j$. This is exactly what $\alpha$-$\beta$ does.

d) The corresponding bound for min nodes $n_k$ is $\max(l_3, l_5, ..., l_k)$.

6.8
This procedure will give incorrect results. Mathematically, the procedure amounts to assuming that averaging commutes with min and max, which it does not. Intuitively, the choices made by each payer in the deterministic trees are based on full knowledge of
future dice rolls, and bear no necessary relationship to the moves made without such knowledge (notice the connection to the discussion of card games on page 179 and to the general problem of fully and partially observable Markov decision problems in Chapter 17). In practice, the method works reasonably well.

6.15
The minimax algorithm for non-zero-sum games works exactly as for multiplayer games, described on p.165-6; that is, the evaluation function is a vector of values, one for each player, and the backup step selects whichever vector has the highest value for the player whose turn it is to move. The example at the end of Section 6.2 (p.167) shows that alpha-beta pruning is not possible in general non-zero-sum games, because an unexamined leaf node might be optimal for both players.
The game tree for the four-square game. Terminal states are in single boxes, loop states in double boxes. Each state is annotated with its minimax value in a circle.

The “?” values are handled by assuming that an agent with a choice between winning the game and entering a “?” state will always choose the win. That is, \( \min(-1,?) \) is \(-1\) and \( \max(+1,?) \) is \(+1\). If all successors are “?” the backed-up value is “?”.