INTRODUCTION TO ARTIFICIAL INTELLIGENCE

EPISODE 3
TODAY’S MENU

1. GAMES
2. MINIMAX
3. ALPHA-BETA PRUNING
GAME TREE
GAME TREE
Fig. 6.18: A minimax game tree with look-ahead of four half-moves.

Fig. 6.19: An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked $a$, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node $b$. Since the first child of $b$ has the value 2, the minimum to be generated for $b$ can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of $b$ can be pruned.

The same reasoning applies for the node $c$. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in $\alpha$.
- For every minimum node the current smallest child value is saved in $\beta$.
- If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$. 

Fig. 6.18 A minimax game tree with look-ahead of four half-moves.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked \( a \), all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node \( b \). Since the first child of \( b \) has the value 2, the minimum to be generated for \( b \) can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of \( b \) can be pruned.

The same reasoning applies for the node \( c \). However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
Fig. 6.18
A minimax game tree with look-ahead of four half-moves

Fig. 6.19
An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked $a$, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node $b$. Since the first child of $b$ has the value 2, the minimum to be generated for $b$ can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of $b$ can be pruned.

The same reasoning applies for the node $c$. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in $\alpha$.
- For every minimum node the current smallest child value is saved in $\beta$.
- If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$. 

GAME TREE
A minimax game tree with look-ahead of four half-moves.

Fig. 6.19: An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

In the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
Search, Games and Problem Solving

Fig. 6.18 A minimax game tree with look-ahead of four half-moves

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

• At every leaf node the evaluation is calculated.
• For every maximum node the current largest child value is saved in $\alpha$.
• For every minimum node the current smallest child value is saved in $\beta$.
• If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
• If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$. 

GAME TREE

MAX

MIN

MAX

MIN

MAX

MIN

0 7
Search, Games and Problem Solving

Fig. 6.18 A minimax game tree with look-ahead of four half-moves.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked $a$, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node $b$. Since the first child of $b$ has the value 2, the minimum to be generated for $b$ can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of $b$ can be pruned.

The same reasoning applies for the node $c$. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

• At every leaf node the evaluation is calculated.
• For every maximum node the current largest child value is saved in $\alpha$.
• For every minimum node the current smallest child value is saved in $\beta$.
• If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
• If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$. 

GAME TREE

MAX

MIN

MAX

MIN

0 7 9
Fig. 6.18
A minimax game tree with look-ahead of four half-moves.

Fig. 6.19
An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

• At every leaf node the evaluation is calculated.
• For every maximum node the current largest child value is saved in $\alpha$.
• For every minimum node the current smallest child value is saved in $\beta$.
• If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
• If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$. 

A minimax game tree with look-ahead of four half-moves.

An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18.

At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in $\alpha$.
- For every minimum node the current smallest child value is saved in $\beta$.
- If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$. 
Fig. 6.18 A minimax game tree with look-ahead of four half-moves.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked $a$, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node $b$. Since the first child of $b$ has the value 2, the minimum to be generated for $b$ can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of $b$ can be pruned.

The same reasoning applies for the node $c$. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in $\alpha$.
- For every minimum node the current smallest child value is saved in $\beta$.
- If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$. 
Fig. 6.18
A minimax game tree with look-ahead of four half-moves.

Fig. 6.19
An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
Search, Games and Problem Solving

Fig. 6.18 A minimax game tree with look-ahead of four half-moves.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

• At every leaf node the evaluation is calculated.
• For every maximum node the current largest child value is saved in α.
• For every minimum node the current smallest child value is saved in β.
• If at a minimum node k the current value β ≤ α, then the search under k can end. Here α is the largest value of a maximum node in the path from the root to k.
• If at a maximum node l the current value α ≥ β, then the search under l can end. Here β is the smallest value of a minimum node in the path from the root to l.
Fig. 6.18 A minimax game tree with look-ahead of four half-moves

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

To the right. Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked $a$, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node $b$. Since the first child of $b$ has the value 2, the minimum to be generated for $b$ can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of $b$ can be pruned.

The same reasoning applies for the node $c$. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in $\alpha$.
- For every minimum node the current smallest child value is saved in $\beta$.
- If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$. 
Fig. 6.18 A minimax game tree with look-ahead of four half-moves.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in $\alpha$.
- For every minimum node the current smallest child value is saved in $\beta$.
- If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$. 
Search, Games and Problem Solving

Fig. 6.18 A minimax game tree with look-ahead of four half-moves

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in $\alpha$.
- For every minimum node the current smallest child value is saved in $\beta$.
- If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$. 

GAME TREE

\[ \text{MAX} \quad \text{MIN} \quad \text{MAX} \quad \text{MIN} \]

\[ \begin{array}{cccccc}
0 & 7 & 9 & 1 & 6 & 3 & 4 \\
\end{array} \]
Fig. 6.18 A minimax game tree with look-ahead of four half-moves.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked $a$, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node $b$. Since the first child of $b$ has the value 2, the minimum to be generated for $b$ can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of $b$ can be pruned.

The same reasoning applies for the node $c$. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in $\alpha$.
- For every minimum node the current smallest child value is saved in $\beta$.
- If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$. 

GAME TREE
Search, Games and Problem Solving

Fig. 6.18 A minimax game tree with look-ahead of four half-moves.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked \( a \), all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node \( b \). Since the first child of \( b \) has the value 2, the minimum to be generated for \( b \) can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of \( b \) can be pruned.

The same reasoning applies for the node \( c \). However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
Search, Games and Problem Solving

Fig. 6.18

A minimax game tree with look-ahead of four half-moves.

Fig. 6.19

An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

104

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

• At every leaf node the evaluation is calculated.
• For every maximum node the current largest child value is saved in $\alpha$.
• For every minimum node the current smallest child value is saved in $\beta$.
• If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
• If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$. 
Search, Games and Problem Solving

Fig. 6.18 A minimax game tree with look-ahead of four half-moves.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
A minimax game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result. Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of b can be pruned. The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in $\alpha$.
- For every minimum node the current smallest child value is saved in $\beta$.
- If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$. 

GAME TREE

- MAX
- MIN
- MAX
- MIN

0 7 9 1 6 7 3 4 1 5 8
Search, Games and Problem Solving

Fig. 6.18 A minimax game tree with look-ahead of four half-moves

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in $\alpha$.
- For every minimum node the current smallest child value is saved in $\beta$.
- If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$. 
Fig. 6.18 A minimax game tree with look-ahead of four half-moves

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18.

At the node marked $a$, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node $b$. Since the first child of $b$ has the value 2, the minimum to be generated for $b$ can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of $b$ can be pruned.

The same reasoning applies for the node $c$. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in $\alpha$.
- For every minimum node the current smallest child value is saved in $\beta$.
- If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$. 
The minimax algorithm works by recursively exploring the game tree, evaluating the utility of game states at the leaf nodes, and backpropagating these evaluations up the tree. At each node, the algorithm alternates between maximizing (MAX) and minimizing (MIN) the utility. The MAX node chooses the child with the highest utility, while the MIN node chooses the child with the lowest utility. When a leaf node is reached, its utility is directly assigned.

At each MAX node, the algorithm records the maximum utility value encountered so far, which is denoted as $\alpha$. Similarly, at each MIN node, the algorithm records the minimum utility value encountered so far, which is denoted as $\beta$. These values are updated as the algorithm progresses up the tree. When $\beta \leq \alpha$, the search can be terminated at that node and its subtree can be pruned, as the utility at the node will not exceed $\alpha$. Conversely, if $\alpha \geq \beta$, the search can be terminated at that node and its subtree can be pruned, as the utility at the node will not be less than $\beta$.

The alpha-beta pruning algorithm further improves efficiency by pruning branches of the game tree that cannot possibly lead to a better outcome than the best outcomes already found.

In the given game tree, the alpha-beta pruning algorithm can be applied as follows:

- At the leaf nodes, the evaluations are calculated:
  - $0 \rightarrow 0$
  - $1 \rightarrow 1$
  - $6 \rightarrow 6$
  - $3 \rightarrow 3$
  - $1 \rightarrow 1$
  - $7 \rightarrow 7$
  - $4 \rightarrow 4$
  - $1 \rightarrow 1$
  - $5 \rightarrow 5$
  - $8 \rightarrow 8$
  - $9 \rightarrow 9$

- At the MAX node, the current $\alpha$ is updated to the maximum of the child values.
- At the MIN node, the current $\beta$ is updated to the minimum of the child values.
- Pruning occurs when $\beta \leq \alpha$ for a MIN node, or $\alpha \geq \beta$ for a MAX node.

In the figure, the alpha-beta pruning algorithm is applied to trim the game tree by ignoring branches that cannot lead to better outcomes than the ones already found.
Fig. 6.18 A minimax game tree with look-ahead of four half-moves.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18.

At the node marked \( a \), all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node \( b \). Since the first child of \( b \) has the value 2, the minimum to be generated for \( b \) can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of \( b \) can be pruned.

The same reasoning applies for the node \( c \). However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
Fig. 6.18
A minimax game tree with look-ahead of four half-moves

Fig. 6.19
An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in $\alpha$.
- For every minimum node the current smallest child value is saved in $\beta$.
- If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$.
Fig. 6.18 A minimax game tree with look-ahead of four half-moves

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in $\alpha$.
- For every minimum node the current smallest child value is saved in $\beta$.
- If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$. 
Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked $a$, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node $b$. Since the first child of $b$ has the value 2, the minimum to be generated for $b$ can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of $b$ can be pruned.

The same reasoning applies for the node $c$. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in $\alpha$.
- For every minimum node the current smallest child value is saved in $\beta$.
- If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$. 
Fig. 6.18 A minimax game tree with look-ahead of four half-moves.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked \( a \), all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node \( b \). Since the first child of \( b \) has the value 2, the minimum to be generated for \( b \) can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of \( b \) can be pruned.

The same reasoning applies for the node \( c \). However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
Fig. 6.18: A minimax game tree with look-ahead of four half-moves.

Fig. 6.19: An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19, this process is depicted for the tree from Fig. 6.18. At the node marked \( a \), all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously, the tree will be trimmed at node \( b \). Since the first child of \( b \) has the value 2, the minimum to be generated for \( b \) can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of \( b \) can be pruned.

The same reasoning applies for the node \( c \). However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node, the evaluation is calculated.
- For every maximum node, the current largest child value is saved in \( \alpha \).
- For every minimum node, the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
Search, Games and Problem Solving

Fig. 6.18 A minimax game tree with look-ahead of four half-moves.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
Fig. 6.18
A minimax game tree with look-ahead of four half-moves.

Fig. 6.19
An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked \( a \), all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node \( b \). Since the first child of \( b \) has the value 2, the minimum to be generated for \( b \) can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of \( b \) can be pruned.

The same reasoning applies for the node \( c \). However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
104 6 Search, Games and Problem Solving

Fig. 6.18 A minimax game tree with look-ahead of four half-moves

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked \( a \), all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node \( b \). Since the first child of \( b \) has the value 2, the minimum to be generated for \( b \) can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of \( b \) can be pruned.

The same reasoning applies for the node \( c \). However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
Fig. 6.18 A minimax game tree with look-ahead of four half-moves.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
Fig. 6.18 A minimax game tree with look-ahead of four half-moves

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked $a$, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node $b$. Since the first child of $b$ has the value 2, the minimum to be generated for $b$ can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of $b$ can be pruned.

The same reasoning applies for the node $c$. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in $\alpha$.
- For every minimum node the current smallest child value is saved in $\beta$.
- If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$. 

GAME TREE
Fig. 6.18 A minimax game tree with look-ahead of four half-moves. 

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

To the right. Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked 𝑎, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be ≤ 1. It could even become smaller still, but that is irrelevant since the maximum is already ≥ 3 one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node 𝑏. Since the first child of 𝑏 has the value 2, the minimum to be generated for 𝑏 can only be less than or equal to 2. But the maximum at the root node is already sure to be ≥ 3. This cannot be changed by values ≤ 2. Thus the remaining subtrees of 𝑏 can be pruned.

The same reasoning applies for the node 𝑐. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in 𝛼.
- For every minimum node the current smallest child value is saved in 𝛽.
- If at a minimum node 𝑘 the current value 𝛽 ≤ 𝛼, then the search under 𝑘 can end. Here 𝛼 is the largest value of a maximum node in the path from the root to 𝑘.
- If at a maximum node 𝑏 the current value 𝛼 ≥ 𝛽, then the search under 𝑏 can end. Here 𝛽 is the smallest value of a minimum node in the path from the root to 𝑏.
Search, Games and Problem Solving

Fig. 6.18 A minimax game tree with look-ahead of four half-moves.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of b can be pruned. The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
Search, Games and Problem Solving

Fig. 6.18
A minimax game tree with look-ahead of four half-moves.

Fig. 6.19
An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be ≤ 1. It could even become smaller still, but that is irrelevant since the maximum is already ≥ 3 one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be ≥ 3. This cannot be changed by values ≤ 2. Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
In Fig. 6.19, this process is depicted for the tree from Fig. 6.18. At the node marked \( a \), all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node \( b \). Since the first child of \( b \) has the value 2, the minimum to be generated for \( b \) can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of \( b \) can be pruned.

The same reasoning applies for the node \( c \). However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
Search, Games and Problem Solving

Fig. 6.18 A minimax game tree with look-ahead of four half-moves.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

• At every leaf node the evaluation is calculated.
• For every maximum node the current largest child value is saved in $\alpha$.
• For every minimum node the current smallest child value is saved in $\beta$.
• If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
• If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$. 

GAME TREE

MAX

MIN

MAX

MIN

0 7 9 1 6 7 3 4 1 5 8 9 2 2 3 4 5 1 2 7 6
Search, Games and Problem Solving

Fig. 6.18 A minimax game tree with look-ahead of four half-moves

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
Search, Games and Problem Solving

Fig. 6.18
A minimax game tree with look-ahead of four half-moves.

Fig. 6.19
An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked \( a \), all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node \( b \). Since the first child of \( b \) has the value 2, the minimum to be generated for \( b \) can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of \( b \) can be pruned.

The same reasoning applies for the node \( c \). However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
Fig. 6.18
A minimax game tree with look-ahead of four half-moves.

Fig. 6.19
An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked \( a \), all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node \( b \). Since the first child of \( b \) has the value 2, the minimum to be generated for \( b \) can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of \( b \) can be pruned.

The same reasoning applies for the node \( c \). However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
At every leaf node the evaluation is calculated.

For every maximum node the current largest child value is saved in $\alpha$.

For every minimum node the current smallest child value is saved in $\beta$.

If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.

If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$. 
Search, Games and Problem Solving

Fig. 6.18 A minimax game tree with look-ahead of four half-moves

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked $a$, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node $b$. Since the first child of $b$ has the value 2, the minimum to be generated for $b$ can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of $b$ can be pruned.

The same reasoning applies for the node $c$. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in $\alpha$.
- For every minimum node the current smallest child value is saved in $\beta$.
- If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$.
Fig. 6.18 A minimax game tree with look-ahead of four half-moves.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in $\alpha$.
- For every minimum node the current smallest child value is saved in $\beta$.
- If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$.
Fig. 6.18
A minimax game tree with look-ahead of four half-moves.

Fig. 6.19
An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
MINIMAX ALGORITHM

1: max_value(node):
2:   if end_state(node): return value(node)
3:   v = –Inf
4:   for each child in node.children():
5:     v = max(v, min_value(child))
6:   return v

1: min_value(node):
2:   if end_state(node): return value(node)
3:   v = +Inf
4:   for each child in node.children():
5:     v = min(v, max_value(child))
6:   return v
HEURISTIC EVALUATION FUNCTIONS

ESTIMATES OF THE VALUE OF THE POSITION
HEURISTIC EVALUATION FUNCTIONS

• The quality (accuracy) of the heuristic will affect the outcome:
  – the better the heuristic => the better the outcome

• Consequently, you can measure the quality of the heuristic by looking at the outcome in a number of games:
  – the better the outcomes => the better the heuristic

• Sometimes even a good player loses to a bad player, so comparing heuristics is not easy

• A common technique for player ranking: Elo rating
ALPHA-BETA PRUNING

Fig. 6.18
A minimax game tree with look-ahead of four half-moves.

Fig. 6.19
An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in $\alpha$.
- For every minimum node the current smallest child value is saved in $\beta$.
- If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$. 

MAX

MIN

MAX

MIN

0 7 9 1 6 7 3 4 1
**ALPHA-BETA PRUNING**

---

Fig. 6.18
A minimax game tree with look-ahead of four half-moves

Fig. 6.19
An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

---

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in $\alpha$.
- For every minimum node the current smallest child value is saved in $\beta$.
- If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$. 

---

The diagram illustrates the process of alpha-beta pruning. At each node, the minimum value (MIN) and the maximum value (MAX) are calculated. If the minimum value of a minimum node is less than or equal to the maximum value of a maximum node, the subtrees below that node can be pruned. Similarly, if the maximum value of a maximum node is greater than or equal to the minimum value of a minimum node, the subtrees above that node can be pruned.

---

MIN-VALUE $\leq 1$
ALPHA-BETA PRUNING

Fig. 6.18 A minimax game tree with look-ahead of four half-moves

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

To the right. Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum.

In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in $\alpha$.
- For every minimum node the current smallest child value is saved in $\beta$.
- If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$.

MIN-VALUE $\leq 1$ $\Rightarrow$ MAX-VALUE = 3
ALPHA-BETA PRUNING

Fig. 6.18 A minimax game tree with look-ahead of four half-moves.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).

MIN-VALUE \( \leq 1 \) \( \Rightarrow \) MAX-VALUE = 3
ALPHA-BETA PRUNING

1: max_value(node, alpha, beta):
2:     if end_state(node): return value(node)
3:     v = –Inf
4:     for each child in node.children():
5:         v = max(v, min_value(child, alpha, beta))
6:         alpha = max(alpha, v)
7:         if alpha >= beta: return v
8:     return v

1: min_value(node, alpha, beta):
2:     if end_state(node): return value(node)
3:     v = +Inf
4:     for each child in node.children():
5:         v = min(v, max_value(child, alpha, beta))
6:         beta = min(beta, v)
7:         if alpha >= beta: return v
8:     return v
Fig. 6.18: A minimax game tree with look-ahead of four half-moves.

Fig. 6.19: An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

In Fig. 6.19, the process is depicted for the tree from Fig. 6.18. At the node marked $a$, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously, the tree will be trimmed at node $b$. Since the first child of $b$ has the value 2, the minimum to be generated for $b$ can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of $b$ can be pruned.

The same reasoning applies for the node $c$. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node, the evaluation is calculated.
- For every maximum node, the current largest child value is saved in $\alpha$.
- For every minimum node, the current smallest child value is saved in $\beta$.
- If at a minimum node $k$, the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$, the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$. 
ALPHA-BETA PRUNING

Fig. 6.18
A minimax game tree with look-ahead of four half-moves

Fig. 6.19
An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked \(a\), all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \(\leq 1\). It could even become smaller still, but that is irrelevant since the maximum is already \(\geq 3\) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node \(b\). Since the first child of \(b\) has the value 2, the minimum to be generated for \(b\) can only be less than or equal to 2. But the maximum at the root node is already sure to be \(\geq 3\). This cannot be changed by values \(\leq 2\). Thus the remaining subtrees of \(b\) can be pruned.

The same reasoning applies for the node \(c\). However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \(\alpha\).
- For every minimum node the current smallest child value is saved in \(\beta\).
- If at a minimum node \(k\) the current value \(\beta \leq \alpha\), then the search under \(k\) can end. Here \(\alpha\) is the largest value of a maximum node in the path from the root to \(k\).
- If at a maximum node \(l\) the current value \(\alpha \geq \beta\), then the search under \(l\) can end. Here \(\beta\) is the smallest value of a minimum node in the path from the root to \(l\).
ALPHA-BETA PRUNING

Fig. 6.18
A minimax game tree with look-ahead of four half-moves.

Fig. 6.19
An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked $a$, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node $b$. Since the first child of $b$ has the value 2, the minimum to be generated for $b$ can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of $b$ can be pruned.

The same reasoning applies for the node $c$. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in $\alpha$.
- For every minimum node the current smallest child value is saved in $\beta$.
- If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$.
ALPHA-BETA PRUNING

Fig. 6.18
A minimax game tree with look-ahead of four half-moves

Fig. 6.19
An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

To the right. Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked \( a \), all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node \( b \). Since the first child of \( b \) has the value 2, the minimum to be generated for \( b \) can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of \( b \) can be pruned.

The same reasoning applies for the node \( c \). However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

\begin{itemize}
\item At every leaf node the evaluation is calculated.
\item For every maximum node the current largest child value is saved in \( \alpha \).
\item For every minimum node the current smallest child value is saved in \( \beta \).
\item If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
\item If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
\end{itemize}
ALPHA-BETA PRUNING

Fig. 6.18 A minimax game tree with look-ahead of four half-moves.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

• At every leaf node the evaluation is calculated.
• For every maximum node the current largest child value is saved in $\alpha$.
• For every minimum node the current smallest child value is saved in $\beta$.
• If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
• If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$. 

$\alpha = 0$
ALPHA-BETA PRUNING

At every leaf node the evaluation is calculated.

For every maximum node the current largest child value is saved in $\alpha$.

For every minimum node the current smallest child value is saved in $\beta$.

If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.

If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$.

The same reasoning applies for the node $c$. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

Fig. 6.18

A minimax game tree with look-ahead of four half-moves

Fig. 6.19

An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

The dotted portions of the tree are not traversed because they have no effect on the end result.

In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked $a$, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node $b$. Since the first child of $b$ has the value 2, the minimum to be generated for $b$ can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of $b$ can be pruned.

The same reasoning applies for the node $c$. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

• At every leaf node the evaluation is calculated.

• For every maximum node the current largest child value is saved in $\alpha$.

• For every minimum node the current smallest child value is saved in $\beta$.

• If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.

• If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$. 

The same reasoning applies for the node $c$. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

Fig. 6.18

A minimax game tree with look-ahead of four half-moves

Fig. 6.19

An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked $a$, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node $b$. Since the first child of $b$ has the value 2, the minimum to be generated for $b$ can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of $b$ can be pruned.

The same reasoning applies for the node $c$. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

• At every leaf node the evaluation is calculated.

• For every maximum node the current largest child value is saved in $\alpha$.

• For every minimum node the current smallest child value is saved in $\beta$.

• If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.

• If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$. 

The same reasoning applies for the node $c$. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.
ALPHA-BETA PRUNING

Fig. 6.18 A minimax game tree with look-ahead of four half-moves

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

To the right. Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum.

In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

• At every leaf node the evaluation is calculated.
• For every maximum node the current largest child value is saved in $\alpha$.
• For every minimum node the current smallest child value is saved in $\beta$.
• If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
• If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$.
ALPHA-BETA PRUNING

Fig. 6.18 A minimax game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. At the node marked \( a \), all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node \( b \). Since the first child of \( b \) has the value 2, the minimum to be generated for \( b \) can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of \( b \) can be pruned.

The same reasoning applies for the node \( c \). However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

• At every leaf node the evaluation is calculated.
• For every maximum node the current largest child value is saved in \( \alpha \).
• For every minimum node the current smallest child value is saved in \( \beta \).
• If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
• If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
ALPHA-BETA PRUNING

Fig. 6.18 A minimax game tree with look-ahead of four half-moves

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked \( a \), all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node \( b \). Since the first child of \( b \) has the value 2, the minimum to be generated for \( b \) can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of \( b \) can be pruned.

The same reasoning applies for the node \( c \). However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
ALPHA-BETA PRUNING

Fig. 6.18 A minimax game tree with look-ahead of four half-moves.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be ≤ 1. It could even become smaller still, but that is irrelevant since the maximum is already ≥ 3 one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be ≥ 3. This cannot be changed by values ≤ 2. Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in α.
- For every minimum node the current smallest child value is saved in β.
- If at a minimum node k the current value β ≤ α, then the search under k can end. Here α is the largest value of a maximum node in the path from the root to k.
- If at a maximum node l the current value α ≥ β, then the search under l can end. Here β is the smallest value of a minimum node in the path from the root to l.
Alpha-Beta pruning

In alphabeta pruning, we maintain two values: 

- \( \alpha \) for the maximum value seen so far in the current branch.
- \( \beta \) for the minimum value seen so far in the current branch.

At every leaf node, the evaluation is calculated.

For every maximum node, the current largest child value is saved in \( \alpha \).

For every minimum node, the current smallest child value is saved in \( \beta \).

- If at a minimum node \( k \), the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \), the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
ALPHA-BETA PRUNING

Fig. 6.18 A minimax game tree with look-ahead of four half-moves

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

To the right. Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in $\alpha$.
- For every minimum node the current smallest child value is saved in $\beta$.
- If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$.

$\alpha = 6$

$\begin{array}{cccccc}
\text{MAX} & & & & & \\
\text{MIN} & & & & & \\
\text{MAX} & & & & & \\
\text{MIN} & & & & & \\
0 & 7 & 9 & 1 & 6 & 7 \\
\end{array}$
ALPHA-BETA PRUNING

Fig. 6.18 A minimax game tree with look-ahead of four half-moves.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

To right. Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
ALPHA-BETA PRUNING

Fig. 6.18 A minimax game tree with look-ahead of four half-moves.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked $a$, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node $b$. Since the first child of $b$ has the value 2, the minimum to be generated for $b$ can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of $b$ can be pruned.

The same reasoning applies for the node $c$. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in $\alpha$.
- For every minimum node the current smallest child value is saved in $\beta$.
- If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$.
ALPHA-BETA PRUNING

Fig. 6.18

Fig. 6.19

Search, Games and Problem Solving

At every leaf node the evaluation is calculated.
• For every maximum node the current largest child value is saved in $\alpha$.
• For every minimum node the current smallest child value is saved in $\beta$.
• If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
• If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$.
ALPHA-BETA PRUNING

Fig. 6.18
A minimax game tree with look-ahead of four half-moves

Fig. 6.19
An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result to right. Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked \( a \), all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node \( b \). Since the first child of \( b \) has the value 2, the minimum to be generated for \( b \) can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of \( b \) can be pruned.

The same reasoning applies for the node \( c \). However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
ALPHA-BETA PRUNING

Fig. 6.18 A minimax game tree with look-ahead of four half-moves.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

The same reasoning applies for the node $c$. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in $\alpha$.
- For every minimum node the current smallest child value is saved in $\beta$.
- If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$.
**ALPHA-BETA PRUNING**

Fig. 6.18 A minimax game tree with look-ahead of four half-moves

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

To right. Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked \( a \), all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node \( b \). Since the first child of \( b \) has the value 2, the minimum to be generated for \( b \) can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of \( b \) can be pruned.

The same reasoning applies for the node \( c \). However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
**ALPHA-BETA PRUNING**

Fig. 6.18 A minimax game tree with look-ahead of four half-moves.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

To the right. Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked $a$, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node $b$. Since the first child of $b$ has the value 2, the minimum to be generated for $b$ can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of $b$ can be pruned.

The same reasoning applies for the node $c$. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in $\alpha$.
- For every minimum node the current smallest child value is saved in $\beta$.
- If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$.
ALPHA-BETA PRUNING

Fig. 6.18 A minimax game tree with look-ahead of four half-moves

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked \(a\), all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \(\leq 1\). It could even become smaller still, but that is irrelevant since the maximum is already \(\geq 3\) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node \(b\). Since the first child of \(b\) has the value 2, the minimum to be generated for \(b\) can only be less than or equal to 2. But the maximum at the root node is already sure to be \(\geq 3\). This cannot be changed by values \(\leq 2\). Thus the remaining subtrees of \(b\) can be pruned.

The same reasoning applies for the node \(c\). However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \(\alpha\).
- For every minimum node the current smallest child value is saved in \(\beta\).
- If at a minimum node \(k\) the current value \(\beta \leq \alpha\), then the search under \(k\) can end. Here \(\alpha\) is the largest value of a maximum node in the path from the root to \(k\).
- If at a maximum node \(l\) the current value \(\alpha \geq \beta\), then the search under \(l\) can end. Here \(\beta\) is the smallest value of a minimum node in the path from the root to \(l\).
**ALPHA-BETA PRUNING**

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in $\alpha$.
- For every minimum node the current smallest child value is saved in $\beta$.
- If at a minimum node $k$, the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$, the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$.

```
1: min_value(node, alpha, beta):
2:     if end_state(node): return value(node)
3:     v = +Inf
4:     for each child in node.children():
5:        v = min(v, max_value(child, alpha, beta))
6:        beta = min(beta, v)
7:        if alpha >= beta: return v
8:     return v
```
**ALPHA-BETA PRUNING**

1: min_value(node, alpha, beta):
2:   if end_state(node): return value(node)
3:   v = +Inf
4:   for each child in node.children():
5:     v = min(v, max_value(child, alpha, beta))
6:     beta = min(beta, v)
7:     if alpha >= beta: return v
8:   return v

**Figure 6.18** A minimax game tree with look-ahead of four half-moves.

**Figure 6.19** An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

In alpha-beta pruning, at every leaf node the evaluation is calculated. For every maximum node the current largest child value is saved in $\alpha$. For every minimum node the current smallest child value is saved in $\beta$. If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$. If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$. 

```python
1: def min_value(node, alpha, beta):
2:     if end_state(node): return value(node)
3:     v = +Inf
4:     for each child in node.children():
5:         v = min(v, max_value(child, alpha, beta))
6:         beta = min(beta, v)
7:         if alpha >= beta: return v
8:     return v
```
ALPHA-BETA PRUNING

In a minimax game tree, alpha-beta pruning works by maintaining two bounds, \( \alpha \) (lower bound) and \( \beta \) (upper bound), as we traverse the tree. The bounds are updated as we explore the tree, and any subtree that cannot possibly improve upon the current bounds can be pruned.

At every leaf node, the evaluation is calculated. For every maximum node, the current largest child value is saved in \( \alpha \). For every minimum node, the current smallest child value is saved in \( \beta \).

If at a minimum node \( k \), the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).

If at a maximum node \( l \), the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).

Here is a Python function that implements the alpha-beta pruning algorithm:

```python
1: def min_value(node, alpha, beta):
2:     if end_state(node):
3:         return value(node)
4:     v = +Inf
5:     for each child in node.children():
6:         v = min(v, max_value(child, alpha, beta))
7:         beta = min(beta, v)
8:         if alpha >= beta:
9:             return v
10:     return v
```

```
1:  min_value(node, alpha, beta):
2:     if end_state(node):
3:         return value(node)
4:     v = +Inf
5:     for each child in node.children():
6:         v = min(v, max_value(child, alpha, beta))
7:         beta = min(beta, v)
8:         if alpha >= beta:
9:             return v
10:     return v
```
ALPHA-BETA PRUNING

The same reasoning applies for the node \( c \). However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).

Here is the pseudocode for the \( \alpha \beta \) pruning function:

```python
1: min_value(node, alpha, beta):
2:   if end_state(node): return value(node)
3:   v = +Inf
4:   for each child in node.children():
5:       v = min(v, max_value(child, alpha, beta))
6:       beta = min(beta, v)
7:       if alpha >= beta: return v
8:   return v
```

Figure 6.18: A minimax game tree with look-ahead of four half-moves.

Figure 6.19: An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result to the right. Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked \( a \), all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node \( b \). Since the first child of \( b \) has the value 2, the minimum to be generated for \( b \) can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of \( b \) can be pruned.
### Alpha-Beta Pruning

#### Algorithm

1. **min_value(node, alpha, beta):**
2.   if end_state(node): return value(node)
3.   v = +\(\text{Inf}\)
4.   for each child in node.children():
5.     v = min(v, max_value(child, alpha, beta))
6.     beta = min(beta, v)
7.     if alpha >= beta: return v
8.   return v

#### Example

**Fig. 6.18** A minimax game tree with look-ahead of four half-moves

**Fig. 6.19** An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

- At the node marked \(a\), all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \(\leq 1\). It could even become smaller still, but that is irrelevant since the maximum is already \(\geq 3\) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node \(b\). Since the first child of \(b\) has the value 2, the minimum to be generated for \(b\) can only be less than or equal to 2. But the maximum at the root node is already sure to be \(\geq 3\). This cannot be changed by values \(\leq 2\). Thus the remaining subtrees of \(b\) can be pruned.

- The same reasoning applies for the node \(c\). However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

**Key Points**

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \(\alpha\).
- For every minimum node the current smallest child value is saved in \(\beta\).
- If at a minimum node \(k\) the current value \(\beta \leq \alpha\), then the search under \(k\) can end. Here \(\alpha\) is the largest value of a maximum node in the path from the root to \(k\).
- If at a maximum node \(l\) the current value \(\alpha \geq \beta\), then the search under \(l\) can end. Here \(\beta\) is the smallest value of a minimum node in the path from the root to \(l\).

**Python Code**

```
1: def min_value(node, alpha, beta):
2:     if end_state(node): return value(node)
3:     v = +\(\text{Inf}\)
4:     for each child in node.children():
5:        v = min(v, max_value(child, alpha, beta))
6:        beta = min(beta, v)
7:        if alpha >= beta: return v
8:     return v
```
ALPHA-BETA PRUNING

Fig. 6.18 A minimax game tree with look-ahead of four half-moves.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked \( a \), all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node \( b \). Since the first child of \( b \) has the value 2, the minimum to be generated for \( b \) can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of \( b \) can be pruned.

The same reasoning applies for the node \( c \). However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
**ALPHA-BETA PRUNING**

**Fig. 6.18** A minimax game tree with look-ahead of four half-moves.

**Fig. 6.19** An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

In Fig. 6.19, this process is depicted for the tree from Fig. 6.18. At the node marked $a$, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node $b$. Since the first child of $b$ has the value 2, the minimum to be generated for $b$ can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of $b$ can be pruned.

The same reasoning applies for the node $c$. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in $\alpha$.
- For every minimum node the current smallest child value is saved in $\beta$.
- If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$. 

\[ \beta = 6 \quad \alpha = 3 \quad \alpha = 3 \]
**ALPHA-BETA PRUNING**

---

**Fig. 6.18** A minimax game tree with look-ahead of four half-moves.

**Fig. 6.19** An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
ALPHA-BETA PRUNING

At every leaf node the evaluation is calculated.

For every maximum node the current largest child value is saved in $\alpha$.

For every minimum node the current smallest child value is saved in $\beta$.

If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.

If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$.
**ALPHA-BETA PRUNING**

Fig. 6.18 A minimax game tree with look-ahead of four half-moves.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
ALPHA-BETA PRUNING

Fig. 6.18 A minimax game tree with look-ahead of four half-moves

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked \(a\), all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \(\leq 1\). It could even become smaller still, but that is irrelevant since the maximum is already \(\geq 3\) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node \(b\). Since the first child of \(b\) has the value 2, the minimum to be generated for \(b\) can only be less than or equal to 2. But the maximum at the root node is already sure to be \(\geq 3\). This cannot be changed by values \(\leq 2\). Thus the remaining subtrees of \(b\) can be pruned.

The same reasoning applies for the node \(c\). However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \(\alpha\).
- For every minimum node the current smallest child value is saved in \(\beta\).
- If at a minimum node \(k\) the current value \(\beta \leq \alpha\), then the search under \(k\) can end. Here \(\alpha\) is the largest value of a maximum node in the path from the root to \(k\).
- If at a maximum node \(l\) the current value \(\alpha \geq \beta\), then the search under \(l\) can end. Here \(\beta\) is the smallest value of a minimum node in the path from the root to \(l\).
ALPHA-BETA PRUNING

Fig. 6.18 A minimax game tree with look-ahead of four half-moves

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked \( a \), all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node \( b \). Since the first child of \( b \) has the value 2, the minimum to be generated for \( b \) can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of \( b \) can be pruned.

The same reasoning applies for the node \( c \). However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).

\( \alpha = 3 \)

\( \alpha = 3 \)
**ALPHA-BETA PRUNING**

**Fig. 6.18** A minimax game tree with look-ahead of four half-moves.

**Fig. 6.19** An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

To the right. Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked \( a \), all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node \( b \). Since the first child of \( b \) has the value 2, the minimum to be generated for \( b \) can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of \( b \) can be pruned.

The same reasoning applies for the node \( c \). However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

1. At every leaf node the evaluation is calculated.
2. For every maximum node the current largest child value is saved in \( \alpha \).
3. For every minimum node the current smallest child value is saved in \( \beta \).
4. If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
5. If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
**ALPHA-BETA PRUNING**

In minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked \( a \), all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node \( b \). Since the first child of \( b \) has the value 2, the minimum to be generated for \( b \) can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of \( b \) can be pruned.

The same reasoning applies for the node \( c \). However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
ALPHA-BETA PRUNING

Fig. 6.18 A minimax game tree with look-ahead of four half-moves

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

To the right. Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in $\alpha$.
- For every minimum node the current smallest child value is saved in $\beta$.
- If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$. 

$\alpha = 3$

$\alpha = 3$
ALPHA-BETA PRUNING

Fig. 6.18 A minimax game tree with look-ahead of four half-moves

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

To the right. Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

• At every leaf node the evaluation is calculated.
• For every maximum node the current largest child value is saved in $\alpha$.
• For every minimum node the current smallest child value is saved in $\beta$.
• If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
• If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$. 

\[ \alpha = 3 \]
\[ \alpha = 3 \]
ALPHA-BETA PRUNING

Fig. 6.18 A minimax game tree with look-ahead of four half-moves.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

To the right. Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked $a$, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node $b$. Since the first child of $b$ has the value 2, the minimum to be generated for $b$ can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of $b$ can be pruned.

The same reasoning applies for the node $c$. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in $\alpha$.
- For every minimum node the current smallest child value is saved in $\beta$.
- If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$.
**ALPHA-BETA PRUNING**

Max

Min

Max

1: min_value(node, alpha, beta):
2: if end_state(node): return value(node)
3:  v = +Inf
4: for each child in node.children():
5:  v = min(v, max_value(child, alpha, beta))
6:  beta = min(beta, v)
7:  if alpha >= beta: return v
8: return v
ALPHA-BETA PRUNING

MAX
MIN
MAX

1:  min_value(node, alpha, beta):
2:    if end_state(node): return value(node)
3:      v = +Inf
4:    for each child in node.children():
5:        v = min(v, max_value(child, alpha, beta))
6:        beta = min(beta, v)
7:    if alpha >= beta: return v
8:  return v

α = 3

α = 3

α = 3
ALPHA-BETA PRUNING

MAX
MIN
MAX

1: min_value(node, alpha, beta):
2: if end_state(node): return value(node)
3: v = +Inf
4: for each child in node.children():
5: v = 2
6: beta = min(beta, v)
7: if alpha >= beta: return v
8: return v

Fig. 6.18 A minimax game tree with look-ahead of four half-moves

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

To the right. Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

• At every leaf node the evaluation is calculated.
• For every maximum node the current largest child value is saved in \( \alpha \).
• For every minimum node the current smallest child value is saved in \( \beta \).
• If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
• If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
ALPHA-BETA PRUNING

Fig. 6.18 A minimax game tree with look-ahead of four half-moves

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked \( a \), all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node \( b \). Since the first child of \( b \) has the value 2, the minimum to be generated for \( b \) can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of \( b \) can be pruned.

The same reasoning applies for the node \( c \). However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end.
- Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end.
- Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).

```python
1: def min_value(node, alpha, beta):
2:     if end_state(node): return value(node)
3:     v = +Inf
4:     for each child in node.children():
5:         v = min(v, max_value(child, alpha, beta))
6:         beta = min(beta, v)
7:         if alpha >= beta: return v
8:     return v
```

\( \alpha = 3 \)
ALPHA-BETA PRUNING

1: min_value(node, alpha, beta):
2:     if end_state(node): return value(node)
3:     v = +Inf
4:     for each child in node.children():
5:         v = min(v, max_value(child, alpha, beta))
6:         beta = min(beta, v)
7:         if alpha >= beta: return v
8:     return v

MAX

MIN

MAX

α = 3

2

α = 3

2

2

2

2

2

2
**ALPHA-BETA PRUNING**

1: min_value(node, alpha, beta):
2:   if end_state(node): return value(node)
3:   v = +Inf
4:   for each child in node.children():
5:     v = min(v, max_value(child, alpha, beta))
6:     beta = min(beta, v)
7:     if alpha >= beta: return v
8:   return v

\[ \alpha = 3 \]

\begin{align*}
\text{MAX} & \quad \text{MIN} \\
3 & \quad 2 \\
6 & \quad 3 \\
\end{align*}
ALPHA-BETA PRUNING

Fig. 6.18 A minimax game tree with look-ahead of four half-moves.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be ≤ 1. It could even become smaller still, but that is irrelevant since the maximum is already ≥ 3 one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be ≥ 3. This cannot be changed by values ≤ 2. Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in α.
- For every minimum node the current smallest child value is saved in β.
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).