JOHDATUS TEKOÄLYYN

TEEMU ROOS
PELIPUU
PELIPUU
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 $l$. 

MAX
MIN
MAX
MIN
MAX
MIN
MAX
MIN
MAX
MIN
MAX
MIN
MAX
MIN
MAX
MIN
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.

In the maximum nodes the maximum value is generated from the maximum value of the successor nodes, and in the minimum nodes the minimum likewise. 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.
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 \( l \).
PELIPUU

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 $l$. 

MAX

MIN

MAX

MIN

0 7 9
PELIPUU

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.

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. 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.
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.

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 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.

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 \textit{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 \textit{b}. Since the first child of \textit{b} has the value 2, the minimum to be generated for \textit{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 \textit{b} can be pruned.

The same reasoning applies for the node \textit{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 \textit{k} the current value $\beta \leq \alpha$, then the search under \textit{k} can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to \textit{k}.
- If at a maximum node \textit{l} the current value $\alpha \geq \beta$, then the search under \textit{l} can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to \textit{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.

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 \( l \).
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 \( \alpha \), 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 \( \beta \). Since the first child of \( \beta \) has the value 2, the minimum to be generated for \( \beta \) 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 \( \beta \) can be pruned.

The same reasoning applies for the node \( \gamma \). 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 $l$. 
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

MAX-ARVO(Solmu)

    if LOPPUTILA(Solmu) return(ARVO(Solmu))
    v = -\infty
    for each Lapsi in Solmun lapset
        v = MAX(v, MIN-ARVO(Lapsi))
    return(v)

MIN-ARVO(Solmu)

    if LOPPUTILA(Solmu) return(ARVO(Solmu))
    v = +\infty
    for each Lapsi in Solmun lapset
        v = MIN(v, MAX-ARVO(Lapsi))
    return(v)
SHAKKI
SHAKKI

Deep Blue beat G. Kasparov in 1997
SHAKKI
Wolfgang von Kempelen rakentaa "Turkin"
L. Torres y Quevedo rakentaa koneen kuningas&torni
vs kuningas -loppupeleihin
Norbert Wiener esittää syvyysrajoitetun minimax-
algoritmin heuristisella arviontifunktiolla
Claude Shannon julkaisee artikkelin “Programming a
Computer for Playing Chess”
Alan Turing kehittää ensimmäisen algoritmin, joka pystyy
pelaamaan kokonaisen shakkiottelun
Los Alamos chess: ensimmäinen tietokoneohjelma, joka
pelaa (yksinkertaistettua) shakkia
John McCarthy keksii alpha-beta-karsinnan
Ensimmäiset oikeaa shakkia pelaavat ohjelmat
Ensimmäiset tietokoneohjelmien väliset ottelut
(Moskova voittaa.)
<table>
<thead>
<tr>
<th>vuosi</th>
<th>tapahtuma</th>
</tr>
</thead>
<tbody>
<tr>
<td>1967</td>
<td>Ensimmäinen tietokoneohjelman voitto turnauksessa.</td>
</tr>
<tr>
<td>1981</td>
<td>Cray Blitz voittaa Mississippin osavaltion mestaruuden ja saa ensimmäisenä tietokoneena mestarin statuksen.</td>
</tr>
<tr>
<td>1989</td>
<td>Garry Kasparov voittaa kaksi näytösottelua Deep Thoughtia vastaan.</td>
</tr>
<tr>
<td>1997</td>
<td>Deep Blue voittaa Garry Kasparovin kuuden pelin ottelussa.</td>
</tr>
<tr>
<td>2006</td>
<td>Deep Fritz voittaa maailmanmestari Vladimir Kramnikin.</td>
</tr>
</tbody>
</table>
SHAKKI

TILA: (LAUDAN TILANNE)

SIIRTYMÄT: (SALLITUT SIIRROT)

MENETELMÄ: SYVYYYSRAJOITETTU ALPHA-BETA-KARSINTA
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 \( l \).

ARVIOITA TILANTEEN HYVYYDESTÄ
SHAKKI

TILA: (LAUDAN TILANNE)

SIIRTYMÄT: (SALLITUT SIIRROT)

MENETELMÄ: SYVYYYSRAJOITETTU ALPHA-BETA-KARSINTA

TEHTÄVÄ: SUUNNITTELE HEURISTINEN ARVIOINTIFUNKTIO
HEURISTIIKKOJEN VALINNASTA

- Heuristikan hyvyyys vaikuttaa pelin tulokseen: 
  **Hyvä heuristiikka -> Hyvä tulos**

- Vastaavasti heuristiikan hyvyyttä voi mitata tarkkailemalla pelien tuloksia: 
  **Hyvä tulos -> Hyvä heuristiikka**

- Joskus hyväkin pelaaja voi silti hävittää huonommalleen ja toisinpäin, joten arviointi ei ole helppoa

- Yleinen menetelmä hyvyyden arviointiin: 
  **ELO-RATING**
ALPHA-BETA-KARSINTA

MAX

MIN

MAX

MIN

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 $l$. 

MIN-ARVO $\leq 1$
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 $l$.

$\Rightarrow \text{MAX-ARVO} = 3$
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 $l$.

$\min$-arvo $\leq 1$ $\Rightarrow \max$-arvo $= 3$
MAX-ARVO(Solmu, $\alpha$, $\beta$)

if LOPPUTILA(Solmu) return(ARVO(Solmu))

$v = -\infty$

for each Lapsi in LAPSET(Solmu)

$v = \text{MAX}(v, \text{MIN-ARVO}(Lapsi, \alpha, \beta))$

if $v \geq \beta$ return($v$)

$\alpha = \text{MAX}(\alpha, v)$

return($v$)
MAX-ARVO(Solmu, $\alpha$, $\beta$)

if LOPPUTILA(Solmu) return(ARVO(Solmu))

$v = -\infty$

for each Lapsi in LAPSET(Solmu)

$v = \text{MAX}(v, \text{MIN-ARVO}(\text{Lapsi}, \alpha, \beta))$

if $v \geq \beta$ return($v$)

$\alpha = \text{MAX}(\alpha, v)$

return($v$)

MIN-ARVO(Solmu, $\alpha$, $\beta$)

if LOPPUTILA(Solmu) return(ARVO(Solmu))

$v = +\infty$

for each Lapsi in LAPSET(Solmu)

$v = \text{MIN}(v, \text{MAX-ARVO}(\text{Lapsi}, \alpha, \beta))$

if $v \leq \alpha$ return($v$)

$\beta = \text{MIN}(\beta, v)$

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.

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 \( l \).

\[
\text{ALPHA-BETA-KARSINTA}
\]
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. 6.19 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 $l$. 

**Algorithm: Alpha-Beta Search**

1. Initialize $\alpha = -\infty$ and $\beta = +\infty$.
2. Recursively search the game tree, updating $\alpha$ and $\beta$ as you go.
3. When reaching a leaf node, return the value of the leaf node.
4. If $\alpha \geq \beta$, prune the rest of the subtree.
5. If $\alpha \leq \beta$, prune the rest of the subtree.

**Example:**

In the example tree, $\alpha = 1$ is shown. All nodes with values $\leq 1$ can be pruned because they will not affect the maximum value at the root.
MIN

MAX

MIN

MAX

MIN

α = 6

0 7 9 1 6 7

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 ≤ 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.
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 \( l \).
ALPHA-BETA-KARSINTA

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 \( l \).
\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}

\begin{minipage}{\textwidth}
\begin{verbatim}
MIN-ARVO(Solmu, $\alpha$, $\beta$)
  if LOPPUTILA(Solmu) return(ARVO(Solmu))
  $v = +\infty$
  for each Lapsi in LAPSET(Solmu)
    $v = \text{MIN}(v, \text{MAX-ARVO}(Lapsi, \alpha, \beta))$
    if $v \leq \alpha$ return($v$)
    $\beta = \text{MIN}(\beta, v)$
  return($v$)
\end{verbatim}
\end{minipage}
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.

\[
\begin{align*}
\text{MAX} & \quad \text{MIN} \\
\text{MAX} & \quad \text{MIN} \\
\text{MAX} & \quad \text{MIN}
\end{align*}
\]

- 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 \( l \).

\[
\begin{align*}
\text{MIN-ARVO}(\text{Solmu}, \alpha, \beta) \\
\text{if LOPPUTILA}(\text{Solmu}) \text{ return } (\text{ARVO}(\text{Solmu})) \\
v = +\infty \\
\text{for each } \text{Lapsi in LAPSET}(\text{Solmu}) \\
v = \text{MIN}(v, \text{MAX-ARVO}(\text{Lapsi}, \alpha, \beta)) \\
\text{if } v \leq \alpha \text{ return } (v) \\
\beta = \text{MIN}(\beta, v) \\
\text{return } (v)
\end{align*}
\]
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 Algorithm

**Algorithm:**

```
MIN-ARVO(Solmu, \( \alpha \), \( \beta \))
```

```
if LOPPUTILA(Solmu) return(ARVO(Solmu))

v = +\( \infty \)
for each Lapsi in LAPSET(Solmu)
    v = MIN(v, MAX-ARVO(Lapsi, \( \alpha \), \( \beta \))
    if v \leq \( \alpha \) return(v)

\( \beta = \text{MIN}(\beta, v) \)
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 an alpha-beta search, a minimum node generates the minimum from the minimum value of the successor nodes and a maximum node generates the maximum from 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 alpha-beta search requires traversing only half of the tree as compared to a pure minimax search.
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 $l$. 

\[ \text{MIN} \quad 3 \quad \text{MAX} \quad \alpha = 3 \quad \text{MIN} \quad \alpha = 3 \]
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 $l$.

MIN-ARVO(Solmu, $\alpha$, $\beta$)

if LOPPUTILA(Solmu) return(ARVO(Solmu))

$v = +\infty$

for each Lapsi in LAPSET(Solmu)

$v = 2$

if $v \leq \alpha$ return($v$)

$\beta = \text{MIN}(\beta, v)$

return($v$)
MIN-ARVO(Solmu, α, β)

if LOPPUTILA(Solmu) return(ARVO(Solmu))

v = +∞

for each Lapsi in LAPSET(Solmu)

v = 2

if 2 ≤ 3 return(v)

β = MIN(β, v)

return(v)
MIN-ARVO(Solmu, α, β)

if LOPPUTILA(Solmu) return(ARVO(Solmu))

v = +∞

for each Lapsi in LAPSET(Solmu)

v = 2

if 2 ≤ 3 return(v)

β = MIN(β, v)

return(v)
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 $l$.
ENSI VIIKOLLA

* LOGIIKASTA (TEKOÄLYN HISTORIAA)
* TODENNÄKÖISYYS