DATA15001
INTRODUCTION TO ARTIFICIAL INTELLIGENCE
EPISODE 3
TODAY’S MENU

1. GAMES

2. MINIMAX

3. ALPHA-BETA PRUNING
GAME TREE

[Diagram of a game tree with tic-tac-toe positions and numerical values at the leaves.]
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 \( l \).
104 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
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 $β \leq α$, 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 $α \geq β$, 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.

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.

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.
A minimax game tree with look-ahead of four half-moves is depicted in Fig. 6.18. The tree shows the decision process for a game where the maximum player aims to maximize the value and the minimum player aims to minimize it.

In the minimax search, the minimum node computes the minimum of the successor nodes, and the maximum node computes the maximum of the successor nodes. This process is repeated until a leaf node is reached, at which point the evaluation is calculated.

In the alpha-beta pruning technique, the search is further optimized by pruning branches that cannot affect the final decision. The process is as follows:

- 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 pruning allows the search to be more efficient by eliminating branches that will not influence the final decision.
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 \( \text{a} \), 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 \( \text{b} \). Since the first child of \( \text{b} \) has the value 2, the minimum to be generated for \( \text{b} \) 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 \( \text{b} \) can be pruned.

The same reasoning applies for the node \( \text{c} \). 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

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

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

MAX
MIN
MAX
MIN

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.

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 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 $\leq 1$
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.

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 \( \leq 1 \)
\[ \Rightarrow \text{MAX-VALUE} = 3 \]
**ALPHA-BETA PRUNING**

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

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

\[\alpha = 0\]
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.

- 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 $l$. 

$\alpha = 6$
ALPHA-BETA PRUNING

MAX
MIN
MAX
MIN

\( \beta = 6 \)
\( \alpha = 3 \)
\( \alpha = 3 \)
\( \alpha = 3 \)

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

![Diagram of 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.]

Fig. 6.18

Fig. 6.19

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 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 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.

β = 6
α = 3

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

\[
\text{min}\_\text{value}(\text{node, alpha, beta}):
\]
1: if end_state(node): return value(node)
2: \( v = +\text{Inf} \)
3: for each child in node.children():
4: \( v = \min(v, \max\_\text{value}(\text{child, alpha, beta})) \)
5: \( \beta = \min(\beta, v) \)
6: if \( \beta \leq \alpha \): return v
7: return v
ALPHA-BETA PRUNING

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.

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 $l$.

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

---

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