Game Playing
$\alpha$-$\beta$ Pruning

Introduction to Artificial Intelligence
Hadi Moradi
2. $\alpha$-$\beta$ pruning: search cutoff

- **Pruning**: eliminating a branch of the search tree from consideration without exhaustive examination of each node.

- **Does it work?**
  - Yes, it roughly cuts the branching factor from $b$ to $\sqrt{b}$, resulting in double as far look-ahead than pure minimax.
$\alpha-\beta$ pruning: example
$\alpha$-$\beta$ pruning: example

\[ \text{MAX} \]

\[ \text{MIN} \]

\[ \geq 6 \]

\[ \leq 2 \]

\[ \times \]

\[ \times \]
\( \alpha - \beta \) pruning: example

\[
\begin{array}{c}
\text{MAX} \\
\geq 6 \\
\text{MIN} \\
\leq 2 \\
\leq 5
\end{array}
\]
\( \alpha - \beta \) pruning: example

MAX

MIN

Selected move

\[ \geq 6 \]

\[ \leq 2 \]

\[ \leq 5 \]
\( \alpha - \beta \) pruning: general principle

If \( \alpha > \nu \) then MAX will choose \( m \) so prune tree under \( n \)

Similar for \( \beta \) for MIN

\( \alpha = 6 \)

\( \nu = 2 \)
Properties of $\alpha$-$\beta$

Pruning does not affect final result

Good move ordering improves effectiveness of pruning

With "perfect ordering," time complexity $= O(b^{m/2})$

$\Rightarrow$ doubles depth of search

$\Rightarrow$ can easily reach depth 8 and play good chess

A simple example of the value of reasoning about which computations are relevant (a form of metareasoning)
The $\alpha$-$\beta$ algorithm

```
function Alpha-Beta-Search(state) returns an action
    inputs: state, current state in game
    v ← Max-Value(state, −∞, +∞)
    return the action in Successors(state) with value v

function Max-Value(state, α, β) returns a utility value
    inputs: state, current state in game
    α, the value of the best alternative for MAX along the path to state
    β, the value of the best alternative for MIN along the path to state
    if Terminal-Test(state) then return Utility(state)
    v ← −∞
    for a, s in Successors(state) do
        v ← Max(v, Min-Value(s, α, β))
    if v ≥ β then return v
    α ← Max(α, v)
    return v
```

//the leaf node (terminal state)
The $\alpha$-$\beta$ algorithm (cont.)

```plaintext
function Min-Value(state, $\alpha$, $\beta$) returns a utility value
    inputs: state, current state in game
            $\alpha$, the value of the best alternative for MAX along the path to state
            $\beta$, the value of the best alternative for MIN along the path to state
    if Terminal-Test(state) then return Utility(state)
    $v \leftarrow +\infty$
    for $a$, $s$ in Successors(state) do
        $v \leftarrow \min(v, \text{Max-Value}(s, \alpha, \beta))$
        if $v \leq \alpha$ then return $v$
        $\beta \leftarrow \min(\beta, v)$
    return $v$
```

//the leaf node (terminal state)
Remember Minimax: Recursive implementation

function `MINIMAX-DECISION(state)` returns an action

\[ v \leftarrow \text{MAX-VALUE}(state) \]
return the action in `SUCCESSORS(state)` with value \( v \)

function `MAX-VALUE(state)` returns a utility value

if `TERMINAL-TEST(state)` then return `UTILITY(state)`
\[ v \leftarrow -\infty \]
for \( a, s \) in `SUCCESSORS(state)` do
\[ v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s)) \]
return \( v \)

function `MIN-VALUE(state)` returns a utility value

if `TERMINAL-TEST(state)` then return `UTILITY(state)`
\[ v \leftarrow \infty \]
for \( a, s \) in `SUCCESSORS(state)` do
\[ v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s)) \]
return \( v \)

**Complete:** Yes, for finite state-space  
**Optimal:** Yes

**Time complexity:** \( O(b^m) \)  
**Space complexity:** \( O(bm) \) (= DFS)  
Does not keep all nodes in memory.)
More on the $\alpha$-$\beta$ algorithm: start from Minimax

```python
function ALPHA-BETA-SEARCH(state) returns an action
    inputs: state, current state in game
    $v \leftarrow$ MAX-VALUE(state, $-\infty$, $+\infty$)
    return the action in SUCCESSORS(state) with value $v$

function MAX-VALUE(state, $\alpha$, $\beta$) returns a utility value
    inputs: state, current state in game
    $\alpha$, the value of the best alternative for MAX along the path to state
    $\beta$, the value of the best alternative for MIN along the path to state
    if TERMINAL-TEST(state) then return UTILITY(state)
    $v \leftarrow -\infty$
    for $a$, $s$ in SUCCESSORS(state) do
        $v \leftarrow$ MAX($v$, MIN-VALUE(s, $\alpha$, $\beta$))
        if $v \geq \beta$ then return $v$
        $\alpha \leftarrow$ MAX($\alpha$, $v$)
    return $v$
```
The $\alpha$-$\beta$ algorithm

Note: These are both Local variables. At the Start of the algorithm, We initialize them to $\alpha = -\infty$ and $\beta = +\infty$
A Walk Through Example

$\alpha$: Best choice so far for MAX
$\beta$: Best choice so far for MIN

MAX

MIN

function $\text{ALPHA-BETA-SEARCH}(\text{state})$ returns an action
inputs: state, current state in game
$v \leftarrow \text{MAX-VALUE}(\text{state}, -\infty, +\infty)$
return the action in $\text{SUCCESSORS}(\text{state})$ with value $v$

function $\text{MAX-VALUE}(\text{state}, \alpha, \beta)$ returns a utility value
inputs: state, current state in game
$\alpha$, the value of the best alternative for MAX along the path to state
$\beta$, the value of the best alternative for MIN along the path to state
In Max-Value:

**α**: Best choice so far for MAX

**β**: Best choice so far for MIN

MAX

\[ \alpha = -\infty \]
\[ \beta = +\infty \]
\[ V = -\infty \]

MIN

Max-Value loops over these

MAX

5 10 4 2 8 7

---

function `MAX-VALUE(state, \alpha, \beta)` returns a utility value

inputs: `state`, current state in game

\[ \alpha \], the value of the best alternative for MAX along the path to `state`

\[ \beta \], the value of the best alternative for MIN along the path to `state`

if `TERMINAL-TEST(state)` then return `UTILITY(state)`

\[ v \leftarrow -\infty \]

for \( a, s \) in `SUCCESSORS(state)` do

\[ v \leftarrow \max(v, \text{MIN-VALUE}(s, \alpha, \beta)) \]

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

\[ \alpha \leftarrow \max(\alpha, v) \]

return \( v \)
In Max-Value:

\[ \alpha : \text{Best choice so far for MAX} \]
\[ \beta : \text{Best choice so far for MIN} \]

\[ \alpha = -\infty \]
\[ \beta = +\infty \]
\[ v = -\infty \]

\[
\text{function} \quad \text{MAX-VALUE}(state, \alpha, \beta) \quad \text{returns a utility value}
\]
\[
\text{inputs: state, current state in game} \quad \text{MAX along the path to state} \quad \text{MIN along the path to state}
\]
\[
\alpha, \beta \quad \text{the value of the best alternative for} \quad \text{for} \quad \text{if} \quad \text{for} \quad \text{return}
\]
\[
\text{if \ TERMINAL-TEST(state) then return \ UTILITY(state)}
\]
\[
v \leftarrow -\infty \quad \text{MIN-VALUE}(s, \alpha, \beta)
\]
\[
\text{for} a, s \text{ in \ SUCCESSORS} \text{state do} \quad v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s, \alpha, \beta)) \quad \text{if} \quad \text{if} \quad \text{if}
\]
\[
\text{if} \quad \text{return} \quad \alpha \leftarrow \text{MAX}(\alpha, v) \quad v \leftarrow \text{MIN}(\alpha, v) \quad \text{return} \quad \text{return}
\]
\[
\text{return} \quad v \quad v \quad v \quad v
\]

\[
\text{MAX} \quad 5 \quad 10 \quad 4 \quad 2 \quad 8 \quad 7
\]

\[
\text{MIN} \quad \alpha = -\infty \quad \beta = +\infty
\]
In Min-Value:

- $\alpha$: Best choice so far for MAX
- $\beta$: Best choice so far for MIN

**MAX**

- $\alpha = -\infty$
- $\beta = +\infty$
- $v = -\infty$

**MIN**

- $\alpha = -\infty$
- $\beta = +\infty$
- $v = +\infty$

**MAX**

- 5
- 10
- 4

**MAX**

- 2
- 8
- 7

---

**Algorithm:**

```plaintext
function MIN-VALUE(state, \( \alpha, \beta \)) returns a utility value
    inputs: state, current state in game
    \( \alpha \), the value of the best alternative for MAX along the path to state
    \( \beta \), the value of the best alternative for MIN along the path to state

    if TERMINAL-TEST(state) then return UTILITY(state)
    \( v \leftarrow +\infty \)
    for \( a, s \) in SUCCESSORS(state) do
        \( v \leftarrow \min(v, \text{MAX-VALUE}(s, \alpha, \beta)) \)
        if \( v \leq \alpha \) then return \( v \)
        \( \beta \leftarrow \min(\beta, v) \)
    return \( v \)
```
In Min-Value:

\[ \alpha : \text{Best choice so far for MAX} \]
\[ \beta : \text{Best choice so far for MIN} \]

\[ \alpha = -\infty \]
\[ \beta = +\infty \]
\[ v = -\infty \]

\[ \alpha = -\infty \]
\[ \beta = +\infty \]
\[ v = +\infty \]

Min-Value loops over these

function `MIN-VALUE(state, \alpha, \beta)` returns a utility value
inputs: `state`, current state in game
\[ \alpha, \beta, \text{the value of the best alternative for MAX along the path to state } \]
\[ \beta, \text{the value of the best alternative for MIN along the path to state } \]
\[ \text{if Terminal-Test(state) then return Utilty(state)} \]
\[ v \leftarrow +\infty \]
\[ \text{for a, s in Successors(state) do } \]
\[ v \leftarrow \text{Min}(v, \text{MAX-VALUE}(s, \alpha, \beta)) \]
\[ \text{if } v \leq \alpha \text{ then return } v \]
\[ \beta \leftarrow \text{MIN}(\beta, v) \]
\[ \text{return } v \]
\( \alpha \): Best choice so far for MAX
\( \beta \): Best choice so far for MIN

**Utility(state)** = 5
In Min-Value:

\( \alpha \): Best choice so far for MAX
\( \beta \): Best choice so far for MIN

```
function MIN-VALUE(state, \( \alpha \), \( \beta \)) returns a utility value
inputs: state, current state in game
        \( \alpha \), the value of the best alternative for MAX along the path to state
        \( \beta \), the value of the best alternative for MIN along the path to state

if TERMINAL-TEST(state) then return UTILITY(state)
\( v \leftarrow +\infty \)
for \( a, s \) in SUCCESSORS(state) do
    \( v \leftarrow \min(v, \text{MAX-VALUE}(s, \alpha, \beta)) \)
    if \( v \leq \alpha \) then return \( v \)
    \( \beta \leftarrow \min(\beta, v) \)
return \( v \)
```
In Min-Value:

α: Best choice so far for MAX
β: Best choice so far for MIN

MAX

MIN

function MIN-VALUE(state, α, β) returns a utility value
inputs: state, current state in game
        α, the value of the best alternative for MAX along the path to state
        β, the value of the best alternative for MIN along the path to state
if TERMINAL-TEST(state) then return UTILITY(state)
v ← +∞
for a, s in SUCCESSORS(state) do
    v ← MIN(v, MAX-VALUE(s, α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
return v
In Min-Value:

\[ \alpha : \text{Best choice so far for MAX} \]

\[ \beta : \text{Best choice so far for MIN} \]

\[ \alpha = -\infty \]
\[ \beta = +\infty \]

\[ \alpha = -\infty \]
\[ \beta = 5 \]
\[ v = 5 \]

\[ \alpha = -\infty \]
\[ \beta = 5 \]
\[ v = 5 \]

\[ \alpha = -\infty \]
\[ \beta = 5 \]
\[ v = 5 \]

function \text{MIN-VALUE}(state, \alpha, \beta) \text{ returns a utility value}
inputs: \text{state, current state in game}
\alpha, \text{ the value of the best alternative for MAX along the path to state}
\beta, \text{ the value of the best alternative for MIN along the path to state}
if \text{TERMINAL-TEST}(state) \text{ then return } \text{UTILITY}(state)
\quad v \leftarrow +\infty
for a, s in \text{SUCCESSORS}(state) \text{ do}
\quad v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s, \alpha, \beta))
\quad \text{if } v \leq \alpha \text{ then return } v
\quad \beta \leftarrow \text{MIN}(\beta, v)
return v
**α**: Best choice so far for MAX  
**β**: Best choice so far for MIN

```plaintext
function MAX-VALUE(state, α, β) returns a utility value
inputs: state, current state in game
        α, the value of the best alternative for MAX along the path to state
        β, the value of the best alternative for MIN along the path to state
if TERMINAL-TEST(state) then return UTILITY(state)
v ← −∞
for a, s in SUCCESSORS(state) do
  v ← MAX(v, MIN-VALUE(s, α, β))
  if v ≥ β then return v
  α ← MAX(α, v)
return v
```
In Min-Value:

- \( \alpha \): Best choice so far for \( \text{MAX} \)
- \( \beta \): Best choice so far for \( \text{MIN} \)

### Code Snippet

```python
function MIN-VALUE(state, \( \alpha \), \( \beta \)) returns a utility value
    inputs: state, current state in game
            \( \alpha \), the value of the best alternative for \( \text{MAX} \) along the path to state
            \( \beta \), the value of the best alternative for \( \text{MIN} \) along the path to state
    if TERMINAL-TEST(state) then return UTILITY(state)
    \( v \leftarrow +\infty \)
    for \( a, s \) in SUCCESSORS(state) do
        \( v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s, \alpha, \beta)) \)
        if \( v \leq \alpha \) then return \( v \)
        \( \beta \leftarrow \text{MIN}(\beta, v) \)
    return \( v \)
```

### Diagram

- **MAX**
  - \( \alpha = -\infty \)
  - \( \beta = +\infty \)

- **MIN**
  - \( \alpha = -\infty \)
  - \( \beta = 5 \)
  - \( v = 5 <10 \)

- **MAX**
  - 5
  - 10
  - 4
  - 2
  - 8
  - 7
\( \alpha \): Best choice so far for MAX
\( \beta \): Best choice so far for MIN

function \textsc{max-value}(state, \alpha, \beta) \textbf{returns} a utility value

inputs: state, current state in game
\( \alpha \), the value of the best alternative for MAX along the path to state
\( \beta \), the value of the best alternative for MIN along the path to state

if \textsc{terminal-test}(state) then return \textsc{utility}(state)

\( v \leftarrow -\infty \)

for \( a, s \) in \textsc{successors}(state) do

\( v \leftarrow \text{max}(v, \textsc{min-value}(s, \alpha, \beta)) \)

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

\( \alpha \leftarrow \text{max}(\alpha, v) \)

return \( v \)

Utility(state) = 4
In Min-Value:

\[ \alpha : \text{Best choice so far for MAX} \]
\[ \beta : \text{Best choice so far for MIN} \]

\[ \alpha = -\infty \]
\[ \beta = +\infty \]
\[ v = -\infty \]

\[ \alpha = -\infty \]
\[ \beta = 5 \]
\[ v = 5 > 4 \]

\text{function } \text{MIN-VALUE}(\text{state, } \alpha, \beta) \text{ returns a utility value}

\text{inputs: } \text{state, current state in game}
\alpha, \text{ the value of the best alternative for MAX along the path to state}
\beta, \text{ the value of the best alternative for MIN along the path to state}

\text{if TERMINAL-TEST(state) then return } \text{UTILITY(state)}
\quad v \leftarrow +\infty
\text{for } s, a \text{ in SUCCESSORS(state) do}
\quad v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s, \alpha, \beta))
\quad \text{if } v \leq \alpha \text{ then return } v
\quad \beta \leftarrow \text{MIN}(\beta, v)
\quad \text{return } v
In Min-Value:

\( \alpha \): Best choice so far for MAX
\( \beta \): Best choice so far for MIN

```
function MIN-VALUE(state, \alpha, \beta) returns a utility value
    inputs: state, current state in game
            \alpha, the value of the best alternative for MAX along the path to state
            \beta, the value of the best alternative for MIN along the path to state
    if TERMINAL-TEST(state) then return UTILITY(state)
    \( v \leftarrow +\infty \)
    for \( a, s \) in SUCCESSORS(state) do
        \( v \leftarrow \min(v, \text{MAX-VALUE}(s, \alpha, \beta)) \)
        if \( v \leq \alpha \) then return \( v \)
        \( \beta \leftarrow \min(\beta, v) \)
    return \( v \)
```
In Min-Value:

\[ \alpha : \text{Best choice so far for MAX} \]
\[ \beta : \text{Best choice so far for MIN} \]

\[ \alpha = -\infty \]
\[ \beta = +\infty \]
\[ v = -\infty \]

function \text{MIN-VALUE}(state, \alpha, \beta) \text{ returns a utility value}

inputs: state, current state in game
\[
\alpha, \beta, \text{the value of the best alternative for MAX along the path to state}
\]
\[
\beta, \text{the value of the best alternative for MIN along the path to state}
\]

if \text{TERMINAL-TEST}(state) then return \text{UTILITY}(state)
\[ v \leftarrow +\infty \]
for \( a, s \) in \text{SUCCESSORS(state)} do
\[ v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s, \alpha, \beta)) \]
if \( v \leq \alpha \) then return \( v \)
\[ \beta \leftarrow \text{MIN}(\beta, v) \]
return \( v \)
\[ \alpha : \text{Best choice so far for MAX} \]

\[ \beta : \text{Best choice so far for MIN} \]

\[ \alpha = -\infty \]

\[ \beta = +\infty \]

\[ v = 4 \]

---

function \text{MAX-VALUE}(state, \alpha, \beta) \text{ returns a utility value}

\[ \alpha, \text{the value of the best alternative for MAX along the path to state} \]

\[ \beta, \text{the value of the best alternative for MIN along the path to state} \]

\text{if} \text{ TERMINAL-TEST}(state) \text{ then return UTILITY(state)}

\[ v \leftarrow -\infty \]

\text{for} \ a, s \ in \text{SUCCESSORS(state)} \ \text{do}

\[ v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s, \alpha, \beta)) \]

\text{if} \ v \geq \beta \ \text{then return} \ v

\[ \alpha \leftarrow \text{MAX}(\alpha, v) \]

\text{return} \ v
\( \alpha \): Best choice so far for MAX
\( \beta \): Best choice so far for MIN

\[
\begin{align*}
\alpha &= 4 \\
\beta &= +\infty \\
v &= 4
\end{align*}
\]

function \( \text{MAX-VALUE}(state, \alpha, \beta) \) returns a utility value
inputs: \( state \), current state in game
\( \alpha \), the value of the best alternative for \( \text{MAX} \) along the path to \( state \)
\( \beta \), the value of the best alternative for \( \text{MIN} \) along the path to \( state \)

if \( \text{TERMINAL-TEST}(state) \) then return \( \text{UTILITY}(state) \)
\( v \leftarrow -\infty \)
for \( a, s \) in \( \text{SUCCESSORS}(state) \) do
\( v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s, \alpha, \beta)) \)
if \( v \geq \beta \) then return \( v \)
\( \alpha \leftarrow \text{MAX}(\alpha, v) \)
return \( v \)
In Max-Value:

\( \alpha \): Best choice so far for MAX

\( \beta \): Best choice so far for MIN

\[ \alpha = -\infty \quad \beta = +\infty \quad v = -\infty \]

\[ \alpha = 4 \quad \beta = +\infty \quad v = 4 \]

---

function MAX-VALUE(state, \( \alpha \), \( \beta \)) returns a utility value

inputs: state, current state in game

- \( \alpha \), the value of the best alternative for MAX along the path to state
- \( \beta \), the value of the best alternative for MIN along the path to state

if TERMINAL-TEST(state) then return UTILITY(state)

\[ v \leftarrow -\infty \]

for \( a, s \) in SUCCESSORS(state) do

\[ v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s, \alpha, \beta)) \]

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

\[ \alpha \leftarrow \text{MAX}(\alpha, v) \]

return \( v \)
\( \alpha \): Best choice so far for MAX

\( \beta \): Best choice so far for MIN

\begin{align*}
\alpha &= -\infty \\
\beta &= +\infty \\
v &= -\infty
\end{align*}

\begin{align*}
\alpha &= 4 \\
\beta &= +\infty \\
v &= +\infty
\end{align*}

\text{function } \text{MIN-VALUE}(\text{state}, \alpha, \beta) \text{ returns a utility value}

\text{inputs: } \text{state}, \text{current state in game}

\quad \alpha, \text{ the value of the best alternative for MAX along the path to state}

\quad \beta, \text{ the value of the best alternative for MIN along the path to state}

\text{if } \text{TERMINAL-TEST}(\text{state}) \text{ then return } \text{UTILITY}(\text{state})

\quad v \leftarrow +\infty

\text{for } a, s \text{ in } \text{SUCCESSORS}(\text{state}) \text{ do}

\quad v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s, \alpha, \beta))

\quad \text{if } v \leq \alpha \text{ then return } v

\quad \beta \leftarrow \text{MIN}(\beta, v)

\text{return } v
\(\alpha\): Best choice so far for MAX

\(\beta\): Best choice so far for MIN

function \(\text{MIN-VALUE}(state, \alpha, \beta)\) returns a utility value

inputs: \(state\), current state in game

\(\alpha\), the value of the best alternative for MAX along the path to \(state\)
\(\beta\), the value of the best alternative for MIN along the path to \(state\)

if \(\text{TERMINAL-TEST}(state)\) then return \(\text{UTILITY}(state)\)

\(v \leftarrow +\infty\)

for \(a, s\) in \(\text{SUCCESSORS}(state)\) do

\(v \leftarrow \min(v, \text{MAX-VALUE}(s, \alpha, \beta))\)

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

\(\beta \leftarrow \min(\beta, v)\)

return \(v\)
\( \alpha \): Best choice so far for MAX

\( \beta \): Best choice so far for MIN

---

```
function MAX-VALUE(state, \( \alpha \), \( \beta \)) returns a utility value
inputs: state, current state in game
\( \alpha \), the value of the best alternative for MAX along the path to state
\( \beta \), the value of the best alternative for MIN along the path to state
if TERMINAL-TEST(state) then return UTILITY(state)
\( v \leftarrow -\infty \)
for \( a, s \) in SUCCESSORS(state) do
\( v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s, \alpha, \beta)) \)
if \( v \geq \beta \) then return \( v \)
\( \alpha \leftarrow \text{MAX}(\alpha, v) \)
return \( v \)
```
\( \alpha \): Best choice so far for MAX

\( \beta \): Best choice so far for MIN

```
function MIN-VALUE(state, \alpha, \beta) returns a utility value
    inputs: state, current state in game
    \alpha, the value of the best alternative for MAX along the path to state
    \beta, the value of the best alternative for MIN along the path to state

    if TERMINAL-TEST(state) then return UTILITY(state)
    \( v \leftarrow +\infty \)
    for \( a, s \) in SUCCESSORS(state) do
        \( v \leftarrow \min(v, \text{MAX-VALUE}(s, \alpha, \beta)) \)
        if \( v \leq \alpha \) then return \( v \)
        \( \beta \leftarrow \min(\beta, v) \)
    return \( v \)
```
α: Best choice so far for MAX
β: Best choice so far for MIN

function MIN-VALUE(state, α, β) returns a utility value
inputs: state, current state in game
α, the value of the best alternative for MAX along the path to state
β, the value of the best alternative for MIN along the path to state
if TERMINAL-TEST(state) then return UTILITY(state)
v ← +∞
for a, s in SUCCESSORS(state) do
v ← MIN(v, MAX-VALUE(s, α, β))
if v ≤ α then return v
β ← MIN(β, v)
return v
$\alpha$: Best choice so far for MAX
$\beta$: Best choice so far for MIN

MAX

MIN

MAX

5 10 4 2 8 7

function MAX-VALUE(state, $\alpha$, $\beta$) returns a utility value
inputs: state, current state in game
$\alpha$, the value of the best alternative for MAX along the path to state
$\beta$, the value of the best alternative for MIN along the path to state
if TERMINAL-TEST(state) then return UTILITY(state)
$v \leftarrow -\infty$
for $a, s$ in SUCCESSORS(state) do
\begin{align*}
v & \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s, \alpha, \beta)) \\
\text{if } v \geq \beta \text{ then return } v \\
\alpha & \leftarrow \text{MAX}(\alpha, v)
\end{align*}
return $v$
Another way to understand the algorithm

From:

For a given node N,

\( \alpha \) is the value of N to MAX
\( \beta \) is the value of N to MIN
Alpha/Beta pruning - example

A move (max)

B move (min)

A move (max)
Alpha/Beta pruning - example

A move (max)

B move (min)

A move (max)

8 7 3 9 1 6 2 4 1 1 3 5 3 9 2 6 5 2 1 2 3 9 7 2
State-of-the-art for deterministic games

Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions.


Othello: human champions refuse to compete against computers, who are too good.

Go: human champions refuse to compete against computers, who are too bad. In go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.
Chess As An Example

- Chess basics
  - 8x8 board, 16 pieces per side, average branching factor of 38
  - Rating system based on competition
    - 500 --- beginner/legal
    - 1200 --- good weekend warrior
    - 2000 --- world championship level
    - 2500+ --- grand master
  - time limited moves
  - important aspects: position, material
Sketch of Chess History

- First discussed by Shannon, Sci. American, 1950

- Initially, two approaches
  - human-like
  - brute force search

- 1966 MacHack ---1100 --- average tournament player

- 1970’s
  - discovery that 1 ply = 200 rating points
  - hash tables
  - quiescence search
Sketch of Chess History

- Chess 4.x reaches 2000 (expert level), 1979
- Belle 2200, 1983
  - special purpose hardware
- 1986 --- Cray Blitz and Hitech 100,000 to 120,000 position/sec using special purpose hardware
Deep Blue History

- 1985, Hsu build BLSI move generator (using DARPA funding!)
- Anantharaman combined with chess program leading to 50K moves/second algorithm
- 1986 CCC ---- no luck, but notice “blind forced move” was a problem
  - singular extension: if you see a position where only a single move determines the line, follow it
- 1987 --- new algorithm won (Chiptest) 400-500K position/sec
IBM checks in

- Deep thought:
  - 250 chips (2M pos/sec /// 6-7M pos/soc)
  - Evaluation hardware
    - piece placement
    - pawn placement
    - passed pawn eval
    - file configurations
    - 120 parameters to tune
  - Tuning done to master’s games
    - hill climbing and linear fits
  - 1989 --- rating of 2480 === Kasparov beats
IBM Checks In

- Deep Blue is the next generation
  - parallel version of Deep Thought
  - 200 M pos/sec → 60B positions in the 3 minutes allotted for move
  - DB 1 = 32 Rs/6000’s with 6 chess proc/node
  - DB 2 = faster 32 nodes w 8 chess proc/node (256 proc)
  - message passing architecture
  - search as much as 20-30 levels deep using singular extension

- In 1997, Kasparov beaten
  - Kasparov changed strategy in earlier games
  - As much a psychological as mental victory

Nondeterministic games

E.g., in backgammon, the dice rolls determine the legal moves.

Simplified example with coin-flipping instead of dice-rolling:

```
MAX

CHANCE

MIN
```

```
2 4 7 4 0 6 0 5 -2
```
Algorithm for nondeterministic games

*Expectiminimax* gives perfect play

Just like *Minimax*, except we must also handle chance nodes:

\[ \text{if } state \text{ is a chance node then} \]
\[ \quad \text{return average of } \text{Expectiminimax-Value of Successors}(state) \]

\[ \text{...} \]

A version of $\alpha$-$\beta$ pruning is possible but only if the leaf values are bounded. Why??
function Minimax-Decision(game) returns an operator

   for each op in Operators[game] do
       Value[op] ← Minimax-Value(Apply(op, game), game)
   end

   return the op with the highest Value[op]

function Minimax-Value(state, game) returns a utility value

   if Terminal-Test[game](state) then
       return Utility[game](state)
   else if MAX is to move in state then
       return the highest Minimax-Value of Successors(state)
   else
       return the lowest Minimax-Value of Successors(state)
Nondeterministic games: the element of chance

Expectimax and Expectimin, expected values over all possible outcomes.
Evaluation functions: Exact values DO matter

Order-preserving transformation do not necessarily behave the same!

NOTE: It is not the same as MINIMAX algorithm
State-of-the-art for nondeterministic games

Dice rolls increase $b$: 21 possible rolls with 2 dice
Backgammon $\approx 20$ legal moves (can be 6,000 with 1-1 roll)

\[ \text{depth} 4 = 20 \times (21 \times 20)^3 \approx 1.2 \times 10^9 \]

As depth increases, probability of reaching a given node shrinks
$\Rightarrow$ value of lookahead is diminished

$\alpha-\beta$ pruning is much less effective
Summary

Games are fun to work on! (and dangerous)

They illustrate several important points about AI

◊ perfection is unattainable ⇒ must approximate

◊ good idea to think about what to think about

◊ uncertainty constrains the assignment of values to states

Games are to AI as grand prix racing is to automobile design
Exercise: Game Playing

Consider the following game tree in which the evaluation function values are shown below each leaf node. Assume that the root node corresponds to the maximizing player. Assume the search always visits children left-to-right.

(a) Compute the backed-up values computed by the minimax algorithm. Show your answer by writing values at the appropriate nodes in the above tree.

(b) Compute the backed-up values computed by the alpha-beta algorithm. What nodes will not be examined by the alpha-beta pruning algorithm?

(c) What move should Max choose once the values have been backed-up all the way?
Exercise:
Game Playing

Max-Value loops over these

\[ \alpha = -\infty \]
\[ \beta = +\infty \]
Exercise: Game Playing

Min-Value loops over these

\[ \alpha = -\infty \]
\[ \beta = +\infty \]
Exercise: Game Playing

\[ \alpha = -\infty \]
\[ \beta = +\infty \]

Max-Value loops over these
Exercise:

Game Playing

\[ \alpha = -\infty \]
\[ \beta = +\infty \]
Exercise:
Game Playing

\[ \alpha = -\infty \]
\[ \beta = +\infty \]
Exercise:
Game Playing

\[ \alpha = -\infty \]
\[ \beta = +\infty \]
Exercise:
Game Playing

\[ \alpha = -\infty \]
\[ \beta = +\infty \]
Exercise:
Game Playing

α = -∞
β = +∞
Exercise: Game Playing

\[ \alpha = -\infty \]
\[ \beta = +\infty \]
Exercise:
Game Playing

\[ \alpha = -\infty \]
\[ \beta = +\infty \]
**BACKGAMMON**
- 2 players
- 15 pieces each
- Goal: Move all pieces off the board
- Rules:
  - Dice roll determines number of moves
  - Players move in opposite directions
  - Piece cannot land on a point occupied by 2 or more of opponent's pieces
  - Single piece can be "hit" if landed on by opponent; hit piece must start anew
- Program: TD-Gammon*
- Advantage: Too close to call

**BRIDGE**
- 4 players in 2 teams
- 13 cards dealt to each player
- Goal: Make 2 "game contracts," or a "rubber"
- Rules:
  - The bid: Each player predicts how many times his or her card will be the highest (a trick)
  - The play: Put down 1 card at a time and compare it with others; this occurs 13 times
  - The scoring: Points scored if bid is made or exceeded; otherwise points go to the opposing team
- Program: GIB*
- Web site: www.githubware.com
- Advantage: Human

**CHECKERS**
- 2 players
- 12 pieces each
- Goal: Avoid being the player who can no longer move (usually when a player has no pieces left)
- Rules:
  - Move forward on dark diagonal, 1 square at a time
  - Opponent's piece captured when jumped to empty square diagonally behind opponent's piece
  - Creation of a "king," a piece that can move backward and forward, occurs when piece is moved to opponent's last row
- Program: Chinook
- Web site: www.cs.ualberta.ca/~chinook
- Advantage: Machine

**CHESS**
- 2 players
- 16 pieces each (1 king, 1 queen, 2 rooks, 2 bishops, 2 knights, 8 pawns)
- Goal: Capture opponent's king (checkmate)
- Rules:
  - Pieces are captured when landed on by opponent's piece
  - Type of piece dictates movement options
- Program: Deep Blue
- Advantage: Too close to call
GO
- 2 players
- Black-and-white stones
- Grid size of board can vary: typical game is on 19-by-19 grid points
- Goal: Conquer a larger part of the board (conquered part encompasses stones placed on board plus stones that could be added safely—that is, within the player's walls)
- Rules:
  Both sides alternate in placing stones on the board
  Stones surrounded by an opponent's stones are captured and removed from the board
- Program: Handtalk*
- Web site: www.webwind.com/go
- Advantage: Human, by a huge margin

OTHELLO
- 2 players
- Black-and-white disks
- Goal: Have most disks on the board at the end of the game
- Rules:
  Players alternate placing disks on unoccupied board spaces
  If opponent's disks are trapped between other player's disks, opponent's disks are flipped to the other player's color
- Program: Logistello
- Web site: www.nei.nj.nec.com/homepages/mic/log.html
- Advantage: Machine

POKER (Texas Hold 'Em)
- 3 to 20 players
- 2 cards dealt to each player; 5 cards placed in center of table
- Goal: Obtain the best hand and win the "pot"
- Rules:
  5 center (community) cards start facedown
  First round of betting ensues; 3 community cards are turned over
  Subsequent rounds of betting ensue; 4th and 5th community cards turned over
  Players select best 5 from the community cards and their hands to obtain identical kinds of cards (pairs, 3- and 4-of-a-kind, flushes (all same suit), straights (sequential) or their combinations
  Final round of betting ensues
- Program: LOKI
- Web site: www.cs.ualberta.ca/~games/poker
- Advantage: Human, by a huge margin

SCRABBLE
- 2 to 4 players
- 100 tiled letters
- Goal: Accumulate most points by creating high-scoring words
- Rules:
  Each player draws 7 letters
  Each letter has a value
  Squares on the board have values
  Words created must join an array
- Program: Maven* (used in Scrabble CD-ROM)
- Advantage: Machine, by a slight margin

*Indicates commercial software that runs on personal computers