Artificial Intelligence

Adversarial Search

Vibhav Gogate

The University of Texas at Dallas

Some material courtesy of Rina Dechter, Alex Ihler and Stuart Russell, Luke Zettlemoyer, Dan Weld
Today

- Adversarial Search
  - Minimax search
  - $\alpha$-$\beta$ search
  - Evaluation functions
  - Expectimax
Game Playing State-of-the-Art
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. Checkers is now solved!
Game Playing State-of-the-Art.
Game Playing State-of-the-Art

- **Chess:** IBM’s Deep Blue defeated human world champion Gary Kasparov in a six-game match in **1997**.
Game Playing State-of-the-Art

- Chess: (Deep Blue vs Kasparov 1996)
Game Playing State-of-the-Art

- Chess: (Deep Blue vs Kasparov 1996)
  - Game 1: Deep Blue wins
Game Playing State-of-the-Art

- Chess: (Deep Blue vs Kasparov 1996)
  - Game 1: Deep Blue wins
  - Game 2: Kasparov adjusts and wins!
Game Playing State-of-the-Art

- Chess: (Deep Blue vs Kasparov 1996)
  - Game 1: Deep Blue wins
  - Game 2: Kasparov adjusts and wins!
  - Game 3 and 4
Game Playing State-of-the-Art

- Chess: (Deep Blue vs Kasparov 1996)
  - Game 1: Deep Blue wins
  - Game 2: Kasparov adjusts and wins!
  - Game 3 and 4
  - Game 5 and 6: Kasparov wins easily!
Game Playing State-of-the-Art

- Chess: (Deep Blue vs Kasparov 1996)
  - Game 1: Deep Blue wins
  - Game 2: Kasparov adjusts and wins!
  - Game 3 and 4
  - Game 5 and 6: Kasparov wins easily!

4 million geeks watched the game online!
Game Playing State-of-the-Art

- Chess: (Deep Blue vs Kasparov 1997)
Game Playing State-of-the-Art

- Chess: (Deep Blue vs Kasparov 1997)
  - Game 1: Kasparov wins, Deep Blue makes a random move!!!
Game Playing State-of-the-Art

- Chess: (Deep Blue vs Kasparov 1997)
  - Game 1: Kasparov wins, Deep Blue makes a random move!!!
  - Game 2: Deep Blue wins. Kasparov misses an opportunity
Game Playing State-of-the-Art

- Chess: (Deep Blue vs Kasparov 1997)
  - Game 1: Kasparov wins, Deep Blue makes a random move!!!
  - Game 2: Deep Blue wins. Kasparov misses an opportunity
  - Game 3, 4 and 5: End in a draw
Game Playing State-of-the-Art

- Chess: (Deep Blue vs Kasparov 1997)
  - Game 1: Kasparov wins, Deep Blue makes a random move!!!
  - Game 2: Deep Blue wins. Kasparov misses an opportunity
  - Game 3, 4 and 5: End in a draw
  - Game 6: Kasparov plays risky. Has a chance to draw but quits!
Game Playing State-of-the-Art

- Chess: (Deep Blue vs Kasparov 1997)
  - Game 1: Kasparov wins, Deep Blue makes a random move!!!
  - Game 2: Deep Blue wins. Kasparov misses an opportunity
  - Game 3, 4 and 5: End in a draw
  - Game 6: Kasparov plays risky. Has a chance to draw but quits!

4 million geeks watched the game online!
Game Playing State-of-the-Art
Game Playing State-of-the-Art

- **Othello**: Human champions refuse to compete against computers, which are too good.
Game Playing State-of-the-Art

- **Othello**: Human champions refuse to compete against computers, which are too good.

- **Go**: Human champions are beginning to be challenged by machines, though the best humans still beat the best machines on the full board. In go, \( b > 300 \), so need pattern knowledge bases and monte carlo search (UCT).
Game Playing State-of-the-Art

- **Othello**: Human champions refuse to compete against computers, which are too good.

- **Go**: Human champions are beginning to be challenged by machines, though the best humans still beat the best machines on the full board. In go, $b > 300$, so need pattern knowledge bases and monte carlo search (UCT).

- **Pacman**: unknown
Types of Games

- **Perfect Information**
  - Deterministic: chess, checkers, go, othello
  - Imperfect Information: stratego

- **Imperfect Information**
  - Deterministic: backgammon, monopoly
  - Chance: bridge, poker, scrabble, nuclear war

Number of Players? 1, 2, …?
Deterministic Games

- Many possible formalizations, one is:
  - States: $S$ (start at $s_0$)
  - Players: $P=\{1...N\}$ (usually take turns)
  - Actions: $A$ (may depend on player / state)
  - Transition Function: $S \times A \rightarrow S$
  - Terminal Test: $S \rightarrow \{t,f\}$
  - Terminal Utilities: $S \times P \rightarrow R$

- Solution for a player is a policy: $S \rightarrow A$
Deterministic Single-Player

- Deterministic, single player, perfect information:
  - Know the rules, action effects, winning states
  - E.g. Freecell, 8-Puzzle, Rubik’s cube
- ... it’s just search!
Deterministic Single-Player

- Deterministic, single player, perfect information:
  - Know the rules, action effects, winning states
  - E.g. Freecell, 8-Puzzle, Rubik’s cube
- ... it’s just search!

  - Slight reinterpretation:
    - Each node stores a value: the best outcome it can reach
    - This is the maximal outcome of its children (the max value)
    - Note that we don’t have path sums as before (utilities at end)
  - After search, can pick move that leads to best node
Deterministic Single-Player

- Deterministic, single player, perfect information:
  - Know the rules, action effects, winning states
  - E.g. Freecell, 8-Puzzle, Rubik’s cube
- ... it’s just search!

- Slight reinterpretation:
  - Each node stores a value: the best outcome it can reach
  - This is the maximal outcome of its children (the max value)
  - Note that we don’t have path sums as before (utilities at end)
- After search, can pick move that leads to best node
Deterministic Single-Player

- Deterministic, single player, perfect information:
  - Know the rules, action effects, winning states
  - E.g. Freecell, 8-Puzzle, Rubik’s cube
- ... it’s just search!

- Slight reinterpretation:
  - Each node stores a value: the best outcome it can reach
  - This is the maximal outcome of its children (the max value)
  - Note that we don’t have path sums as before (utilities at end)
- After search, can pick move that leads to best node
Deterministic Two-Player

- E.g. tic-tac-toe, chess, checkers
- Zero-sum games
  - One player maximizes result
  - The other minimizes result
Deterministic Two-Player

- E.g. tic-tac-toe, chess, checkers
- Zero-sum games
  - One player maximizes result
  - The other minimizes result

- Minimax search
  - A state-space search tree
  - Players alternate
  - Choose move to position with highest minimax value = best achievable utility against best play
Tic-tac-toe Game Tree
Tic-tac-toe Game Tree
Tic-tac-toe Game Tree

\[
\begin{array}{cccccccc}
\text{MAX (X)} & \text{MIN (O)} \\
X & X & X & X & X & X & X & X \\
\end{array}
\]
Tic-tac-toe Game Tree
Tic-tac-toe Game Tree

MAX (X)

MIN (O)

MAX (X)

MIN (O)
Tic-tac-toe Game Tree

MAX (X)

MIN (O)

MAX (X)

MIN (O)

TERMINAL

Utility

-1 0 +1
Minimax Example

max

min
Minimax Example

max

min

A_{11} A_{12} A_{13} \quad 3 \quad 12 \quad 8 \quad A_{21} A_{22} A_{23} \quad 4 \quad 2 \quad 6 \quad A_{31} A_{32} A_{33} \quad 14 \quad 5 \quad 2
Minimax Example

max

min
Minimax Example
Minimax Example

max

min
Minimax Example

max

min
Minimax Search

function $\text{Max-Value}(\text{state})$ returns a utility value

if $\text{Terminal-Test}(\text{state})$ then return $\text{Utility}(\text{state})$

$v \leftarrow -\infty$

for $a, s$ in $\text{Successors}(\text{state})$ do $v \leftarrow \text{Max}(v, \text{Min-Value}(s))$

return $v$

function $\text{Min-Value}(\text{state})$ returns a utility value

if $\text{Terminal-Test}(\text{state})$ then return $\text{Utility}(\text{state})$

$v \leftarrow \infty$

for $a, s$ in $\text{Successors}(\text{state})$ do $v \leftarrow \text{Min}(v, \text{Max-Value}(s))$

return $v$
Minimax Properties

- Optimal?
- Time complexity?
- Space complexity?
Minimax Properties

- Optimal?
  - Yes, against perfect player. Otherwise, can do even better! Why?

- Time complexity?

- Space complexity?
Minimax Properties

- **Optimal?**
  - Yes, against perfect player. Otherwise, can do even better! Why?

- **Time complexity?**
  - $O(b^m)$

- **Space complexity?**
Minimax Properties

- Optimal?
  - Yes, against perfect player. Otherwise, can do even better! Why?

- Time complexity?
  - $O(b^m)$

- Space complexity?
  - $O(bm)$
Minimax Properties

- **Optimal?**
  - Yes, against perfect player. Otherwise, can do even better! Why?

- **Time complexity?**
  - $O(b^m)$

- **Space complexity?**
  - $O(bm)$

- **For chess, $b \sim 35$, $m \sim 100$**
  - Exact solution is completely infeasible
  - But, do we need to explore the whole tree?
Do We Need to Evaluate Every Node?
$\alpha$-$\beta$ Pruning Example

Progress of search...
**α-β Pruning**

- General configuration
  - $\alpha$ is the best value that MAX can get at any choice point along the current path.
  - If $n$ becomes worse than $\alpha$, MAX will avoid it, so can stop considering $n$’s other children.
  - Define $\beta$ similarly for MIN.
Alpha-Beta Pseudocode

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

returns: a utility value

\textbf{function} \textsc{Max-Value}(\textit{state}, \alpha, \beta) \quad \textbf{function} \textsc{Min-Value}(\textit{state}, \alpha, \beta)

\textbf{if} TERMINAL-TEST(\textit{state}) \textbf{then} \quad \textbf{if} TERMINAL-TEST(\textit{state}) \textbf{then}
\quad \text{return UTILITY(\textit{state})} \quad \text{return UTILITY(\textit{state})}
\quad \nu \leftarrow -\infty \quad \nu \leftarrow +\infty
\textbf{for} a, s \text{ in SUCCESSORS}(\textit{state}) \textbf{do} \quad \textbf{for} a, s \text{ in SUCCESSORS}(\textit{state}) \textbf{do}
\quad \nu \leftarrow \textsc{Max}(\nu, \textsc{Min-Value}(s, \alpha, \beta)) \quad \nu \leftarrow \textsc{Min}(\nu, \textsc{Max-Value}(s, \alpha, \beta))
\quad \textbf{if} \nu \geq \beta \textbf{ then return } \nu \quad \textbf{if} \nu \leq \alpha \textbf{ then return } \nu
\quad \alpha \leftarrow \textsc{Max}(\alpha, \nu) \quad \beta \leftarrow \textsc{Min}(\beta, \nu)
\text{return } \nu \quad \text{return } \nu
Alpha-Beta Pseudocode

inputs: state, current game state
    α, value of best alternative for MAX on path to state
    β, value of best alternative for MIN on path to state
returns: a utility value

function MAX-VALUE(state, α, β)
    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

function MIN-VALUE(state, α, β)
    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

At max node:
    Prune if v ≥ β;
    Update α

At min node:
    Prune if v ≤ α;
    Update β
Alpha-Beta Pruning Example

At max node:
Prune if $v \geq \beta$;
Update $\alpha$ 

At min node:
Prune if $v \leq \alpha$;
Update $\beta$
Alpha-Beta Pruning Example

At max node:
- Prune if $v \geq \beta$
- Update $\alpha$

At min node:
- Prune if $v \leq \alpha$
- Update $\beta$

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

$\alpha$ is MAX’s best alternative here or above
$\beta$ is MIN’s best alternative here or above
Alpha-Beta Pruning Example

At max node:
Prune if $v \geq \beta$;
Update $\alpha$

At min node:
Prune if $v \leq \alpha$;
Update $\beta$

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

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

$\alpha$ is MAX’s best alternative here or above
$\beta$ is MIN’s best alternative here or above
**Alpha-Beta Pruning Example**

At max node:
- Prune if $v \geq \beta$;
- Update $\alpha$

At min node:
- Prune if $v \leq \alpha$;
- Update $\beta$

$\alpha = -\infty$

$\beta = +\infty$

$\alpha$ is MAX’s best alternative here or above

$\beta$ is MIN’s best alternative here or above
At max node:
Prune if $v \geq \beta$;
Update $\alpha$

At min node:
Prune if $v \leq \alpha$;
Update $\beta$

$\alpha$ is MAX’ s best alternative here or above
$\beta$ is MIN’ s best alternative here or above
At max node:
- Prune if \( v \geq \beta \);
- Update \( \alpha \)

At min node:
- Prune if \( v \leq \alpha \);
- Update \( \beta \)

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

\( \alpha = -\infty \)
\( \beta = 3 \)

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

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

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

\( \alpha = -\infty \)
\( \beta = 3 \)

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

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

\( \alpha = -\infty \)
\( \beta = 3 \)

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

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

\( \alpha = -\infty \)
\( \beta = 3 \)

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

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

\( \alpha = -\infty \)
\( \beta = 3 \)

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

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

\( \alpha = -\infty \)
\( \beta = 3 \)

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

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

\( \alpha = -\infty \)
\( \beta = 3 \)

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

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

\( \alpha = -\infty \)
\( \beta = 3 \)

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

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

\( \alpha = -\infty \)
\( \beta = 3 \)

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

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

\( \alpha = -\infty \)
\( \beta = 3 \)

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

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

\( \alpha = -\infty \)
\( \beta = 3 \)

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

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

\( \alpha = -\infty \)
\( \beta = 3 \)

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

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

\( \alpha = -\infty \)
\( \beta = 3 \)

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

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

\( \alpha = -\infty \)
\( \beta = 3 \)

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

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

\( \alpha = -\infty \)
\( \beta = 3 \)

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

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

\( \alpha = -\infty \)
\( \beta = 3 \)

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

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

\( \alpha = -\infty \)
\( \beta = 3 \)

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

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

\( \alpha = -\infty \)
\( \beta = 3 \)

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

\( \alpha = -\infty \)
At max node:
Prune if $v \geq \beta$;
Update $\alpha$

At min node:
Prune if $v \leq \alpha$;
Update $\beta$

$\alpha$ is MAX’s best alternative here or above
$\beta$ is MIN’s best alternative here or above
At max node:
Prune if $v \geq \beta$;
Update $\alpha$

At min node:
Prune if $v \leq \alpha$;
Update $\beta$

$\alpha$ is MAX’s best alternative here or above
$\beta$ is MIN’s best alternative here or above
At max node:
Prune if \( v \geq \beta \);
Update \( \alpha \)

At min node:
Prune if \( v \leq \alpha \);
Update \( \beta \)

\( \alpha \) is MAX’ s best alternative here or above
\( \beta \) is MIN’ s best alternative here or above
Alpha-Beta Pruning Example

At max node:
Prune if \( v \geq \beta \);
Update \( \alpha \)

At min node:
Prune if \( v \leq \alpha \);
Update \( \beta \)

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

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

\[ \alpha = -\infty \quad \alpha = -\infty \]
\[ \beta = 3 \quad \beta = 3 \]

\[ 3 \]
\[ 12 \]

\[ 8 \]

\( \alpha \) is MAX’s best alternative here or above
\( \beta \) is MIN’s best alternative here or above
At max node:
Prune if $v \geq \beta$;
Update $\alpha$

At min node:
Prune if $v \leq \alpha$;
Update $\beta$

$\alpha$ is MAX’s best alternative here or above
$\beta$ is MIN’s best alternative here or above
At max node:
Prune if $v \geq \beta$;
Update $\alpha$

At min node:
Prune if $v \leq \alpha$;
Update $\beta$

$\alpha$ is MAX’ s best alternative here or above
$\beta$ is MIN’ s best alternative here or above
At max node:
Prune if $v \geq \beta$;
Update $\alpha$

At min node:
Prune if $v \leq \alpha$;
Update $\beta$

$\alpha$ is MAX’s best alternative here or above
$\beta$ is MIN’s best alternative here or above
At max node:
Prune if $v \geq \beta$;
Update $\alpha$

At min node:
Prune if $v \leq \alpha$;
Update $\beta$

$\alpha$ is MAX’s best alternative here or above
$\beta$ is MIN’s best alternative here or above
At max node:
Prune if $v \geq \beta$;
Update $\alpha$

At min node:
Prune if $v \leq \alpha$;
Update $\beta$

$\alpha$ is MAX’s best alternative here or above
$\beta$ is MIN’s best alternative here or above
At max node:
Prune if $v \geq \beta$;
Update $\alpha$

At min node:
Prune if $v \leq \alpha$;
Update $\beta$

$\alpha = -\infty$  $\beta = +\infty$
$\alpha = 3$   $\beta = +\infty$

$\alpha = -\infty$  $\beta = +\infty$
$\alpha = -\infty$  $\beta = 3$
$\alpha = 3$   $\beta = 3$

$\alpha = -\infty$  $\beta = 3$
$\alpha = 8$   $\beta = 3$
$\alpha = 8$   $\beta = 3$

$\alpha$ is MAX’s best alternative here or above
$\beta$ is MIN’s best alternative here or above
Alpha-Beta Pruning Example

At max node:
Prune if $v \geq \beta$;
Update $\alpha$

At min node:
Prune if $v \leq \alpha$;
Update $\beta$

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

$\alpha = 3$
$\beta = +\infty$

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

$\alpha = -\infty$
$\beta = 3$

$\alpha = -\infty$
$\beta = 3$

$\alpha = 8$
$\beta = 3$

$\geq 8$

$\alpha = -\infty$
$\beta = 3$

$\alpha = 8$
$\beta = 3$

$\alpha$ is MAX’ s best alternative here or above
$\beta$ is MIN’ s best alternative here or above
**Alpha-Beta Pruning Example**

At max node:
- Prune if \( v \geq \beta \);
- Update \( \alpha \)

At min node:
- Prune if \( v \leq \alpha \);
- Update \( \beta \)

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

\[ \alpha = 3 \]
\[ \beta = +\infty \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 8 \]
\[ \beta = 3 \]

\[ \alpha = -\infty \]
\[ \beta = 3 \]

\[ \alpha = 8 \]
\[ \beta = 3 \]

\[ \alpha = -\infty \]
\[ \beta = 3 \]

\[ \alpha = 8 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 8 \]
\[ \beta = 3 \]

\[ \alpha = 8 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]

\[ \alpha = 3 \]
\[ \beta = 3 \]
At max node:
Prune if $v \geq \beta$;
Update $\alpha$

At min node:
Prune if $v \leq \alpha$;
Update $\beta$

$\alpha$ is MAX’s best alternative here or above
$\beta$ is MIN’s best alternative here or above
Alpha-Beta Pruning Example

At max node:
- Prune if $v \geq \beta$;
- Update $\alpha$

At min node:
- Prune if $v \leq \alpha$;
- Update $\beta$

$\alpha$ is MAX’ s best alternative here or above
$\beta$ is MIN’ s best alternative here or above
At max node:
Prune if \( v \geq \beta \);
Update \( \alpha \)

At min node:
Prune if \( v \leq \alpha \);
Update \( \beta \)

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

\( \alpha = 3 \)
\( \beta = +\infty \)

\( \alpha = 3 \)
\( \beta = 2 \)

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

\( \alpha = -\infty \)
\( \beta = 3 \)

\( \alpha = 3 \)
\( \beta = +\infty \)

\( \alpha = 8 \)
\( \beta = 3 \)

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

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

\( \alpha = 8 \)
\( \beta = 3 \)

\( \alpha \) is MAX’ s best alternative here or above
\( \beta \) is MIN’ s best alternative here or above
Alpha-Beta Pruning Example

At max node:
Prune if $v \geq \beta$;
Update $\alpha$

At min node:
Prune if $v \leq \alpha$;
Update $\beta$

$\alpha$ is MAX’ s best alternative here or above
$\beta$ is MIN’ s best alternative here or above
At max node:
Prune if $v \geq \beta$;
Update $\alpha$

At min node:
Prune if $v \leq \alpha$;
Update $\beta$

$\alpha$ is MAX’s best alternative here or above
$\beta$ is MIN’s best alternative here or above
At max node:
Prune if $v \geq \beta$;
Update $\alpha$

At min node:
Prune if $v \leq \alpha$;
Update $\beta$

$\alpha$ is MAX’s best alternative here or above
$\beta$ is MIN’s best alternative here or above
At max node:
Prune if \( v \geq \beta \);
Update \( \alpha \)

At min node:
Prune if \( v \leq \alpha \);
Update \( \beta \)

\( \alpha \) is MAX’ s best alternative here or above
\( \beta \) is MIN’ s best alternative here or above
At max node:
Prune if $v \geq \beta$;
Update $\alpha$

At min node:
Prune if $v \leq \alpha$;
Update $\beta$

$\alpha$ is MAX’s best alternative here or above
$\beta$ is MIN’s best alternative here or above
At max node:
Prune if $v \geq \beta$;
Update $\alpha$

At min node:
Prune if $v \leq \alpha$;
Update $\beta$

$\alpha$ is MAX’ s best alternative here or above
$\beta$ is MIN’ s best alternative here or above
At max node:
Prune if $v \geq \beta$;
Update $\alpha$

At min node:
Prune if $v \leq \alpha$;
Update $\beta$

$\alpha$ is MAX’s best alternative here or above
$\beta$ is MIN’s best alternative here or above
At max node:
Prune if $v \geq \beta$;
Update $\alpha$

At min node:
Prune if $v \leq \alpha$;
Update $\beta$

$\alpha$ is MAX’s best alternative here or above
$\beta$ is MIN’s best alternative here or above
At max node:
Prune if $v \geq \beta$;
Update $\alpha$

At min node:
Prune if $v \leq \alpha$;
Update $\beta$

$\alpha$ is MAX’s best alternative here or above
$\beta$ is MIN’s best alternative here or above
Alpha-Beta Pruning Example

At max node:
Prune if \( v \geq \beta \);
Update \( \alpha \)

At min node:
Prune if \( v \leq \alpha \);
Update \( \beta \)

\( \alpha \) is MAX’s best alternative here or above
\( \beta \) is MIN’s best alternative here or above
α is MAX’s best alternative here or above
β is MIN’s best alternative here or above
α is MAX’s best alternative here or above
β is MIN’s best alternative here or above
Alpha-Beta Pruning Properties

• This pruning has no effect on final result at the root

• Values of intermediate nodes might be wrong!
  – but, they are bounds

• Good child ordering improves effectiveness of pruning

• With “perfect ordering”:
  – Time complexity drops to $O(b^{m/2})$
  – Doubles solvable depth!
  – Full search of, e.g. chess, is still hopeless...
Resource Limits

- Cannot search to leaves
- Depth-limited search
  - Instead, search a limited depth of tree
  - Replace terminal utilities with heuristic eval function for non-terminal positions
- Guarantee of optimal play is gone
- Example:
  - Suppose we have 100 seconds, can explore 10K nodes / sec
  - So can check 1M nodes per move
  - $\alpha-\beta$ reaches about depth 8 decent chess program
Resource Limits

- Cannot search to leaves
- Depth-limited search
  - Instead, search a limited depth of tree
  - Replace terminal utilities with heuristic eval function for non-terminal positions
- Guarantee of optimal play is gone
- Example:
  - Suppose we have 100 seconds, can explore 10K nodes / sec
  - So can check 1M nodes per move
  - $\alpha-\beta$ reaches about depth 8 decent chess program
Resource Limits

- Cannot search to leaves
- Depth-limited search
  - Instead, search a limited depth of tree
  - Replace terminal utilities with heuristic eval function for non-terminal positions
- Guarantee of optimal play is gone
- Example:
  - Suppose we have 100 seconds, can explore 10K nodes / sec
  - So can check 1M nodes per move
  - $\alpha-\beta$ reaches about depth 8 decent chess program
Heuristic Evaluation Function

- Function which scores non-terminals

![Chess Diagram]

Black to move
White slightly better

![Chess Diagram]

White to move
Black winning
Heuristic Evaluation Function

- Function which scores non-terminals

- Ideal function: returns the utility of the position
Heuristic Evaluation Function

- Function which scores non-terminals

\[ \text{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) \]

- Ideal function: returns the utility of the position
- In practice: typically weighted linear sum of features:
  - e.g. \( f_1(s) = (\text{num white queens} - \text{num black queens}) \), etc.
What features would be good for Pacman?

$$Eval(s) = w_1f_1(s) + w_2f_2(s) + \ldots + w_nf_n(s)$$
Why Pacman Starves

- He knows his score will go up by eating the dot now.
- He knows his score will go up just as much by eating the dot later on.
- There are no point-scoring opportunities after eating the dot.
- Therefore, waiting seems just as good as eating.
Iterative Deepening

Iterative deepening uses DFS as a subroutine:

1. Do a DFS which only searches for paths of length 1 or less. (DFS gives up on any path of length 2)
2. If “1” failed, do a DFS which only searches paths of length 2 or less.
3. If “2” failed, do a DFS which only searches paths of length 3 or less.

….and so on.

Why do we want to do this for multiplayer games?
Stochastic Single-Player

• What if we don’t know what the result of an action will be? E.g.,
  – In solitaire, shuffle is unknown
  – In minesweeper, mine locations
Stochastic Single-Player

- What if we don’t know what the result of an action will be? E.g.,
  - In solitaire, shuffle is unknown
  - In minesweeper, mine locations

- Can do **expectimax search**
  - Chance nodes, like actions except the environment controls the action chosen
  - Max nodes as before
  - Chance nodes take average (expectation) of value of children
• Why should we average utilities? Why not minimax?

• Principle of maximum expected utility: an agent should choose the action which maximizes its expected utility, given its knowledge
  – General principle for decision making
  – Often taken as the definition of rationality
  – We’ll see this idea over and over in this course!

• Let’s decompress this definition...
A random variable represents an event whose outcome is unknown. A probability distribution is an assignment of weights to outcomes.

Example: traffic on freeway?
- Random variable: T = whether there’s traffic
- Outcomes: T in {none, light, heavy}
- Distribution: P(T=none) = 0.25, P(T=light) = 0.55, P(T=heavy) = 0.20

Some laws of probability (more later):
- Probabilities are always non-negative
- Probabilities over all possible outcomes sum to one

As we get more evidence, probabilities may change:
- P(T=heavy) = 0.20, P(T=heavy | Hour=8am) = 0.60
- We’ll talk about methods for reasoning and updating probabilities later
What are Probabilities?

- Objectivist / frequentist answer:

- Subjectivist / Bayesian answer:
What are Probabilities?

- **Objectivist / frequentist answer:**
  - Averages over repeated *experiments*
  - E.g. empirically estimating $P(\text{rain})$ from historical observation
  - E.g. pacman’s estimate of what the ghost will do, given what it has done in the past
  - Assertion about how future experiments will go (in the limit)
  - Makes one think of *inherently random* events, like rolling dice

- **Subjectivist / Bayesian answer:**
What are Probabilities?

- **Objectivist / frequentist answer:**
  - Averages over repeated *experiments*
  - E.g. empirically estimating $P(\text{rain})$ from historical observation
  - E.g. pacman’s estimate of what the ghost will do, given what it has done in the past
  - Assertion about how future experiments will go (in the limit)
  - Makes one think of *inherently random* events, like rolling dice

- **Subjectivist / Bayesian answer:**
  - Degrees of belief about unobserved variables
  - E.g. an agent’s belief that it’s raining, given the temperature
  - E.g. pacman’s belief that the ghost will turn left, given the state
  - Often *learn* probabilities from past experiences (more later)
  - New evidence *updates beliefs* (more later)
Uncertainty Everywhere
Uncertainty Everywhere

• Not just for games of chance!
  – I’m sick: will I sneeze this minute?
  – Email contains “FREE!”: is it spam?
  – Tooth hurts: have cavity?
  – 60 min enough to get to the airport?
  – Robot rotated wheel three times, how far did it advance?
  – Safe to cross street? (Look both ways!)
Uncertainty Everywhere

• Not just for games of chance!
  – I’m sick: will I sneeze this minute?
  – Email contains “FREE!”: is it spam?
  – Tooth hurts: have cavity?
  – 60 min enough to get to the airport?
  – Robot rotated wheel three times, how far did it advance?
  – Safe to cross street? (Look both ways!)

• Sources of uncertainty in random variables:
  – Inherently random process (dice, etc)
  – Insufficient or weak evidence
  – Ignorance of underlying processes
  – Unmodeled variables
  – The world’s just noisy – it doesn’t behave according to plan!
Reminder: Expectations

- We can define function \( f(X) \) of a random variable \( X \)

- The expected value of a function is its average value, weighted by the probability distribution over inputs

- Example: How long to get to the airport?
  - Length of driving time as a function of traffic:
    \( L(\text{none}) = 20, \ L(\text{light}) = 30, \ L(\text{heavy}) = 60 \)
  - What is my expected driving time?
    - Notation: \( E_{P(T)}[\ L(T) \] 
    - Remember, \( P(T) = \{\text{none: 0.25, light: 0.5, heavy: 0.25}\} \)

\[
E[\ L(T) \] = L(\text{none}) \ast P(\text{none}) + L(\text{light}) \ast P(\text{light}) + L(\text{heavy}) \ast P(\text{heavy})
\]

\[
E[\ L(T) \] = (20 \ast 0.25) + (30 \ast 0.5) + (60 \ast 0.25) = 35
\]
Review: Expectations

- Real valued functions of random variables:
  \[ f : X \rightarrow R \]

- Expectation of a function of a random variable
  \[ E_{P(X)}[f(X)] = \sum_x f(x)P(x) \]

- Example: Expected value of a fair die roll

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P )</th>
<th>( f )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1/6</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1/6</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1/6</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1/6</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>1/6</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>1/6</td>
<td>6</td>
</tr>
</tbody>
</table>

\[
1 \times \frac{1}{6} + 2 \times \frac{1}{6} + 3 \times \frac{1}{6} + 4 \times \frac{1}{6} + 5 \times \frac{1}{6} + 6 \times \frac{1}{6} = 3.5
\]
Utilities

- Utilities are functions from outcomes (states of the world) to real numbers that describe an agent’s preferences.

- Where do utilities come from?
  - In a game, may be simple (+1/-1)
  - Utilities summarize the agent’s goals
  - Theorem: any set of preferences between outcomes can be summarized as a utility function (provided the preferences meet certain conditions)

- In general, we hard-wire utilities and let actions emerge (why don’t we let agents decide their own utilities?)

- More on utilities soon...
Stochastic Two-Player

- E.g. backgammon
- Expectiminimax (!)
  - Environment is an extra player that moves after each agent
  - Chance nodes take expectations, otherwise like minimax

if \textit{state} is a \texttt{Max} node then

\textbf{return} the highest ExpectiMinimax-Value of Successors(\textit{state})

if \textit{state} is a \texttt{Min} node then

\textbf{return} the lowest ExpectiMinimax-Value of Successors(\textit{state})

if \textit{state} is a chance node then

\textbf{return} average of ExpectiMinimax-Value of Successors(\textit{state})
Stochastic Two-Player

- Dice rolls increase $b$: 21 possible rolls with 2 dice
  - Backgammon $\approx$ 20 legal moves
  - Depth 4 = $20 \times (21 \times 20)^3 = 1.2 \times 10^9$
- As depth increases, probability of reaching a given node shrinks
  - So value of lookahead is diminished
  - So limiting depth is less damaging
  - But pruning is less possible...
- TDGammon uses depth-2 search + very good eval function + reinforcement learning: world-champion level play
Expectimax Search Trees

• What if we don’t know what the result of an action will be? E.g.,
  – In solitaire, next card is unknown
  – In minesweeper, mine locations
  – In pacman, the ghosts act randomly
Expectimax Search Trees

- What if we don’t know what the result of an action will be? E.g.,
  - In solitaire, next card is unknown
  - In minesweeper, mine locations
  - In pacman, the ghosts act randomly

- Can do **expectimax search**
  - Chance nodes, like min nodes, except the outcome is uncertain
  - Calculate **expected utilities**
  - Max nodes as in minimax search
  - Chance nodes take average (expectation) of value of children
Expectimax Search Trees

- What if we don’t know what the result of an action will be? E.g.,
  - In solitaire, next card is unknown
  - In minesweeper, mine locations
  - In pacman, the ghosts act randomly

- Can do **expectimax search**
  - Chance nodes, like min nodes, except the outcome is uncertain
  - Calculate **expected utilities**
  - Max nodes as in minimax search
  - Chance nodes take average (expectation) of value of children

- Later, we’ll learn how to formalize the underlying problem as a **Markov Decision Process**
Expectimax Search
In expectimax search, we have a probabilistic model of how the opponent (or environment) will behave in any state

- Model could be a simple uniform distribution (roll a die)
- Model could be sophisticated and require a great deal of computation
- We have a node for every outcome out of our control: opponent or environment
- The model might say that adversarial actions are likely!
In expectimax search, we have a probabilistic model of how the opponent (or environment) will behave in any state:
- Model could be a simple uniform distribution (roll a die)
- Model could be sophisticated and require a great deal of computation
- We have a node for every outcome out of our control: opponent or environment
- The model might say that adversarial actions are likely!

For now, assume for any state we magically have a distribution to assign probabilities to opponent actions / environment outcomes
Expectimax Pseudocode

def value(s):
    if s is a max node return maxValue(s)
    if s is an exp node return expValue(s)
    if s is a terminal node return evaluation(s)

def maxValue(s):
    values = [value(s') for s' in successors(s)]
    return max(values)

def expValue(s):
    values = [value(s') for s' in successors(s)]
    weights = [probability(s, s') for s' in successors(s)]
    return expectation(values, weights)
Expectimax for Pacman

• Notice that we’ve gotten away from thinking that the ghosts are trying to minimize pacman’s score
• Instead, they are now a part of the environment
• Pacman has a belief (distribution) over how they will act
• Quiz: Can we see minimax as a special case of expectimax?
• Quiz: what would pacman’s computation look like if we assumed that the ghosts were doing 1- ply minimax and taking the result 80% of the time, otherwise moving randomly?
# Expectimax for Pacman

## Results from playing 5 games

<table>
<thead>
<tr>
<th></th>
<th>Minimizing Ghost</th>
<th>Random Ghost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimax Pacman</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Expectimax Pacman</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Pacman does depth 4 search with an eval function that avoids trouble
Minimizing ghost does depth 2 search with an eval function that seeks Pacman
# Expectimax for Pacman

Results from playing 5 games

<table>
<thead>
<tr>
<th></th>
<th>Minimizing Ghost</th>
<th>Random Ghost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimax Pacman</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Expectimax Pacman</td>
<td></td>
<td>Won 5/5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Avg. Score: 503</td>
</tr>
</tbody>
</table>

Pacman does depth 4 search with an eval function that avoids trouble
Minimizing ghost does depth 2 search with an eval function that seeks Pacman
## Expectimax for Pacman

Results from playing 5 games

<table>
<thead>
<tr>
<th></th>
<th>Minimizing Ghost</th>
<th>Random Ghost</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Minimax Pacman</strong></td>
<td>Won 5/5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avg. Score: 493</td>
<td></td>
</tr>
<tr>
<td><strong>Expectimax Pacman</strong></td>
<td></td>
<td>Won 5/5</td>
</tr>
<tr>
<td></td>
<td>Avg. Score: 503</td>
<td></td>
</tr>
</tbody>
</table>

Pacman does depth 4 search with an eval function that avoids trouble
Minimizing ghost does depth 2 search with an eval function that seeks Pacman
## Expectimax for Pacman

### Results from playing 5 games

<table>
<thead>
<tr>
<th></th>
<th>Minimizing Ghost</th>
<th>Random Ghost</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Minimax Pacman</strong></td>
<td>Won 5/5</td>
<td>Won 5/5</td>
</tr>
<tr>
<td></td>
<td>Avg. Score: 493</td>
<td>Avg. Score: 483</td>
</tr>
<tr>
<td><strong>Expectimax Pacman</strong></td>
<td>Won 5/5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avg. Score: 503</td>
<td></td>
</tr>
</tbody>
</table>

Pacman does depth 4 search with an eval function that avoids trouble
Minimizing ghost does depth 2 search with an eval function that seeks Pacman
Expectimax for Pacman

Results from playing 5 games

<table>
<thead>
<tr>
<th></th>
<th>Minimizing Ghost</th>
<th>Random Ghost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimax Pacman</td>
<td>Won 5/5</td>
<td>Won 5/5</td>
</tr>
<tr>
<td></td>
<td>Avg. Score: 493</td>
<td>Avg. Score: 483</td>
</tr>
<tr>
<td>Expectimax Pacman</td>
<td>Won 1/5</td>
<td>Won 5/5</td>
</tr>
<tr>
<td></td>
<td>Avg. Score: -303</td>
<td>Avg. Score: 503</td>
</tr>
</tbody>
</table>

Pacman does depth 4 search with an eval function that avoids trouble
Minimizing ghost does depth 2 search with an eval function that seeks Pacman
Expectimax Pruning?
Expectimax Pruning?

- Not easy
  - exact: need bounds on possible values
  - approximate: sample high-probability branches
Expectimax Evaluation

• Evaluation functions quickly return an estimate for a node’s true value (which value, expectimax or minimax?)

• For minimax, evaluation function scale doesn’t matter
  – We just want better states to have higher evaluations (get the ordering right)
  – We call this insensitivity to monotonic transformations

• For expectimax, we need *magnitudes* to be meaningful
Expectimax Evaluation

- Evaluation functions quickly return an estimate for a node’s true value (which value, expectimax or minimax?)
- For minimax, evaluation function scale doesn’t matter
  - We just want better states to have higher evaluations (get the ordering right)
  - We call this insensitivity to monotonic transformations
- For expectimax, we need *magnitudes* to be meaningful
Expectimax Evaluation

- Evaluation functions quickly return an estimate for a node’s true value (which value, expectimax or minimax?)
- For minimax, evaluation function scale doesn’t matter
  - We just want better states to have higher evaluations (get the ordering right)
  - We call this insensitivity to monotonic transformations
- For expectimax, we need *magnitudes* to be meaningful
Mixed Layer Types

- E.g. Backgammon
- Expectiminimax
  - Environment is an extra player that moves after each agent
  - Chance nodes take expectations, otherwise like minimax

```
if state is a Max node then
    return the highest ExpectiMinimax-Value of Successors(state)
if state is a Min node then
    return the lowest ExpectiMinimax-Value of Successors(state)
if state is a chance node then
    return average of ExpectiMinimax-Value of Successors(state)
```
Stochastic Two-Player

- Dice rolls increase $b$: 21 possible rolls with 2 dice
  - Backgammon $\approx$ 20 legal moves
  - Depth 4 = $20 \times (21 \times 20)^3$ = $1.2 \times 10^9$
- As depth increases, probability of reaching a given node shrinks
  - So value of lookahead is diminished
  - So limiting depth is less damaging
  - But pruning is less possible...
- TDGammon uses depth-2 search + very good eval function + reinforcement learning: world-champion level play
Multi-player Non-Zero-Sum Games

- Similar to minimax:
  - Utilities are now tuples
  - Each player maximizes their own entry at each node
  - Propagate (or back up) nodes from children
  - Can give rise to cooperation and competition dynamically...

```
1,2,6  4,3,2  6,1,2  7,4,1  5,1,1  1,5,2  7,7,1  5,4,5
```