Kasparov Vs. Deep Junior
August 2, 2003
Match ends in a 3 / 3 tie!
Games in AI

- In AI, “games” usually refers to deterministic, turn-taking, two-player, zero-sum games of perfect information
  - Deterministic: next state of environment is completely determined by current state and action executed by the agent (not probabilistic)
  - Turn-taking: 2 agents whose actions must alternate
  - Zero-sum games: if one agent wins, the other loses
  - Perfect information: fully observable
### Other Games

<table>
<thead>
<tr>
<th>Deterministic</th>
<th>Chance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chess, Checkers, Go, Othello</td>
<td>Backgammon, Monopoly</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Perfect Information</th>
<th>Imperfect Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bridge, Poker, Scrabble, Nuclear War</td>
<td></td>
</tr>
</tbody>
</table>
Games as Search

• Components:
  – States:
  – Initial state:
  – Successor function:
  –
  – Terminal test:
  – Utility function:
Games as Search

- **Components:**
  - **States:** board configurations
  - **Initial state:** the board position and which player will move
  - **Successor function:** returns list of (move, state) pairs, each indicating a legal move and the resulting state
  - **Terminal test:** determines when the game is over
  - **Utility function:** gives a numeric value in terminal states (eg, -1, 0, +1 in chess for loss, tie, win)
Games as Search

- **Components:**
  - States: board configurations
  - Initial state: the board position and which player will move
  - Successor function: returns list of (move, state) pairs, each indicating a legal move and the resulting state
  - Terminal test: determines when the game is over
  - Utility function: gives a numeric value in terminal states (e.g., -1, 0, +1 in chess for loss, tie, win)

- **Convention:** first player is MAX, 2nd player is MIN
- State utility values from MAX’s perspective
- Initial state and legal moves define the game tree
Intuition
Mini-Max
Mini-Max Properties

- Complete?
- Optimal?
  - Against an optimal opponent?
  - Otherwise?
- Time complexity?
- Space complexity?
Mini-Max Properties

- Complete? Yes, if tree is finite
- Optimal?
  - Against an optimal opponent? Yes
  - Otherwise? Then MAX does even better
- Time complexity? $O(b^m)$
- Space complexity? $O(bm)$
Good Enough?

Chess:
- branching factor $b \approx 35$
- game length $m \approx 100$
- search space $b^m \approx 35^{100} \approx 10^{154}$

the Universe:
- number of atoms $\approx 10^{78}$
- age $\approx 10^{21}$ milliseconds
Alpha-Beta Pruning
Do we need to check this node?
No - this branch is guaranteed to be worse than what max already has.
MinVal(state, alpha, beta) {
    if (terminal(state)) return utility(state);
    for (s in children(state)) {
        child = MaxVal(s, alpha, beta);
        beta = min(beta, child);
        if (alpha >= beta) return child;
    }
    return beta;
}

alpha = the highest value for MAX along the path
beta = the lowest value for MIN along the path
\textbf{Alpha-Beta}

\begin{verbatim}
MaxVal(state, alpha, beta) {
    if (terminal(state)) return utility(state);
    for (s in children(state)) {
        child = MinVal(s, alpha, beta);
        alpha = max(alpha, child);
        if (alpha >= beta) return child;
    }
    return alpha;
}
\end{verbatim}

\textit{alpha} = the highest value for \textbf{MAX} along the path
\textit{beta} = the lowest value for \textbf{MIN} along the path
\(\alpha\) – the best value for \textbf{max} along the path

\(\beta\) – the best value for \textbf{min} along the path
\( \alpha \) – the best value for max along the path

\( \beta \) – the best value for min along the path
\( \alpha \) – the best value for **max** along the path

\( \beta \) – the best value for **min** along the path
α – the best value for \textbf{max} along the path
β – the best value for \textbf{min} along the path

α = -29
β = -37

β < α prune!
\( \alpha \) – the best value for \textbf{max} along the path

\( \beta \) – the best value for \textbf{min} along the path
\( \alpha \) – the best value for \textbf{max} along the path
\( \beta \) – the best value for \textbf{min} along the path
\( \alpha \) – the best value for \textbf{max} along the path
\( \beta \) – the best value for \textbf{min} along the path
\( \alpha \) – the best value for \textbf{max} along the path

\( \beta \) – the best value for \textbf{min} along the path

\( \beta < \alpha \) prune!
\( \alpha \) – the best value for \textbf{max} along the path
\( \beta \) – the best value for \textbf{min} along the path
$\alpha$ – the best value for $\textbf{max}$ along the path
$\beta$ – the best value for $\textbf{min}$ along the path
\( \alpha \) – the best value for \textbf{max} along the path
\( \beta \) – the best value for \textbf{min} along the path

\( \alpha = -43 \)
\( \beta = \infty \)

\( \beta < \alpha \) prune!
Still guaranteed to find the best move
Best case time complexity: $O(b^{m/2})$
Can double the depth of search!
Best case when best moves are tried first
Good static evaluation function helps!
But still too slow for chess...
Good Enough?

- **Chess:**
  - branching factor $b \approx 35$
  - game length $m \approx 100$
  - search space $b^{m/2} \approx 35^{50} \approx 10^{77}$

- **The Universe:**
  - number of atoms $\approx 10^{78}$
  - age $\approx 10^{21}$ milliseconds

The universe can play chess -- can we?
Partial Space Search

- Strategies:
  - search to a fixed depth
  - iterative deepening (most common)
  - ignore ‘quiescent’ nodes
- Static Evaluation Function assigns a score to a non-terminal state
The diagram represents a decision tree with the following structure:

- The root node is labeled as 'max 0'.
- From the root, two branches extend downward:
  - One branch labeled 'min 0' leads to another 'max 0' node.
  - The other branch labeled 'max 0' leads to the 'Cutoff' label.
- From the 'Cutoff' level, eight branches extend downward:
  - Seven of these branches are labeled as 'min 0', leading to leaf nodes with values:
    - 84, -29, -37, -25, 1, -43, -75, 49, -21, -51, 58, -46, -3, -13, 26, 79.
  - One branch is labeled as 'min 0' and leads to a leaf node with no further values shown.

The tree structure illustrates a binary decision-making process with values at the leaf nodes.
Evaluation Functions

Othello: multiply pieces by their positions

\[(9\ 1\ 3\ 3\ 3\ 3\ 1\ 9)\]
\[(1\ 1\ 1\ 1\ 1\ 1\ 1\ 1)\]
\[(3\ 1\ 4\ 3\ 3\ 4\ 1\ 3)\]
\[(3\ 1\ 3\ 4\ 4\ 3\ 1\ 3)\]
\[(3\ 1\ 3\ 4\ 4\ 3\ 1\ 3)\]
\[(3\ 1\ 4\ 3\ 3\ 4\ 1\ 3)\]
\[(1\ 1\ 1\ 1\ 1\ 1\ 1\ 1)\]
\[(9\ 1\ 3\ 3\ 3\ 3\ 1\ 9)\]
Chess:

eval(s) = 
  \( w_1 \) \(*\) material(s) + 
  \( w_2 \) \(*\) mobility(s) + 
  \( w_3 \) \(*\) king safety(s) + 
  \( w_4 \) \(*\) center control(s) + ... 

- In practice MiniMax improves accuracy of heuristic eval function
- But one can construct pathological games where more search hurts performance! (Nau 1981)
End-Game Databases

- Ken Thompson - all 5 piece end-games
- Lewis Stiller - all 6 piece end-games
  - Refuted common chess wisdom: many positions thought to be ties were really forced winds – 90% for white
  - Is perfect chess a win for white?
White wins in 255 moves
- the longest longest shortest forced win

(the shortest path to mate is longer than all other shortest paths with the same material - and longer than all known longest shortest paths with any other material)

(Stiller, 1991)
Deterministic Games in Practice

• **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 (!)

• **Chess**: Deep Blue defeated human world champion Gary Kasparov in a 6 game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply

• **Othello**: human champions refuse to play against computers because software is **too good**
Deterministic Games in Practice

- **Go**: human champions refuse to compete against computers, because software is too bad.

<table>
<thead>
<tr>
<th></th>
<th>Chess</th>
<th>Go</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size of board</td>
<td>8 x 8</td>
<td>19 x 19</td>
</tr>
<tr>
<td>Average no. of moves per game</td>
<td>100</td>
<td>300</td>
</tr>
<tr>
<td>Avg branching factor per turn</td>
<td>35</td>
<td>235</td>
</tr>
<tr>
<td>Additional complexity</td>
<td></td>
<td>Players can pass</td>
</tr>
</tbody>
</table>
Deterministic Games Summary

- Basic idea: minimax – too slow for most games
- Alpha-Beta pruning can reduce the branching factor by up to 2
- Limited depth search may be necessary
- Static evaluation functions necessary for limited depth search and help alpha-beta
- Opening game and End game databases can help
- Computers can beat humans in some games (checkers, chess, othello) but not in others (Go)
# Other Games

<table>
<thead>
<tr>
<th>Perfect Information</th>
<th>Deterministic</th>
<th>Chess, Checkers, Go, Othello</th>
</tr>
</thead>
<tbody>
<tr>
<td>Imperfect Information</td>
<td>Chance</td>
<td>Backgammon, Monopoly</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bridge, Poker, Scrabble, Nuclear War</td>
</tr>
</tbody>
</table>
Nondeterministic Games

- Involve chance: dice, shuffling, etc.
- Chance nodes: calculate the expected value (e.g., weighted average over all possible dice rolls)
white has 4 possible moves--but doesn’t know what Black will roll, and so doesn’t know what Black’s legal moves will be
In Practice...

- Chance adds dramatically to size of search space
  - Backgammon: number of distinct possible rolls of dice is 21
  - Branching factor $b$ is usually around 20, but can be as high as 4000 (dice rolls that are doubles)
- Alpha-beta pruning is generally less effective
- Best Backgammon programs use other methods
Imperfect Information

- E.g. card games, where opponents’ initial cards are unknown
- Idea: For all deals consistent with what you can see
  - compute the minimax value of available actions for each of possible deals
  - compute the expected value over all deals