CS 188: Artificial Intelligence
Search with other Agents II

[These slides adapted from Dan Klein and Pieter Abbeel]
Minimax Implementation (Dispatch)

**def value(state):**
- if the state is a terminal state: return the state’s utility
- if the next agent is MAX: return max-value(state)
- if the next agent is MIN: return min-value(state)

**def max-value(state):**
- initialize \( v = -\infty \)
- for each successor of state:
  - \( v = \max(v, \text{value(successor)}) \)
- return \( v \)

**def min-value(state):**
- initialize \( v = +\infty \)
- for each successor of state:
  - \( v = \min(v, \text{value(successor)}) \)
- return \( v \)
Minimax Example
Resource Limits
Minimax Properties

Optimal against a perfect player. Otherwise?

[Demo: min vs exp (L6D2, L6D3)]
Video of Demo Min vs. Exp (Min)
Video of Demo Min vs. Exp (Exp)
Minimax Efficiency

- How efficient is minimax?
  - Just like (exhaustive) DFS
    - Time: $O(b^m)$
    - Space: $O(bm)$

- Example: For chess, $b \approx 35$, $m \approx 100$
  - Exact solution is completely infeasible
  - But, do we need to explore the whole tree?
Game Tree Pruning
Minimax Example
Alpha-Beta Pruning

- General configuration (MIN version)
  - We’re computing the MIN-VALUE at some node $n$
  - We’re looping over $n$’s children
  - $n$’s estimate of the childrens’ min is dropping
  - Who cares about $n$’s value? MAX
  - Let $a$ be the best value that MAX can get at any choice point along the current path from the root
  - If $n$ becomes worse than $a$, MAX will avoid it, so we can stop considering $n$’s other children (it’s already bad enough that it won’t be played)

- MAX version is symmetric
**Alpha-Beta Implementation**

\[ \alpha: \text{MAX's best option on path to root} \]
\[ \beta: \text{MIN's best option on path to root} \]

---

**def max-value(state, \(\alpha\), \(\beta\)):**

- Initialize \(v = -\infty\)
- For each successor of state:
  - \(v = \max(v, \text{value}(\text{successor}, \alpha, \beta))\)
  - If \(v \geq \beta\) return \(v\)
  - \(\alpha = \max(\alpha, v)\)
- Return \(v\)

**def min-value(state, \(\alpha\), \(\beta\)):**

- Initialize \(v = +\infty\)
- For each successor of state:
  - \(v = \min(v, \text{value}(\text{successor}, \alpha, \beta))\)
  - If \(v \leq \alpha\) return \(v\)
  - \(\beta = \min(\beta, v)\)
- Return \(v\)
Alpha-Beta Pruning Properties

- This pruning has no effect on minimax value computed for the root!

- Values of intermediate nodes might be wrong
  - Important: children of the root may have the wrong value
  - So the most naïve version won’t let you do action selection

- 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...

- This is a simple example of metareasoning (computing about what to compute)
Alpha-Beta Quiz 2

Diagram:

- Node a connects to nodes b, e, and h.
- Node b connects to nodes c and d.
- Node e connects to nodes f and g.
- Node h connects to nodes i and l.
- Node i connects to nodes j and k.
- Node l connects to nodes m and n.

Leaf nodes:
- c, d, f, g, j, k, m, n
- 10, 6, 100, 8, 1, 2, 20, 4
Resource Limits

- Problem: In realistic games, cannot search to leaves!

- Solution: Depth-limited search
  - Instead, search only to a limited depth in the tree
  - Replace terminal utilities with an evaluation function for non-terminal positions

- 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

- Guarantee of optimal play is gone

- More plies makes a BIG difference

- Use iterative deepening for an anytime algorithm
Video of Demo Limited Depth (2)
Video of Demo Limited Depth (10)
Evaluation Functions
Evaluation Functions

- Evaluation functions score non-terminals in depth-limited search.

- Ideal function: returns the actual minimax value of the position.
- In practice: typically weighted linear sum of features:
  \[ \text{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) \]
- e.g. \( f_1(s) = (\text{num white queens} - \text{num black queens}) \), etc.
Evaluation for Pacman

[Demo: thrashing d=2, thrashing d=2 (fixed evaluation function), smart ghosts coordinate (L6D6, 7, 8, 10)]
Video of Demo Thrashing (d=2)
Why Pacman Starves

- A danger of replanning agents!
  - He knows his score will go up by eating the dot now (west, east)
  - He knows his score will go up just as much by eating the dot later (east, west)
  - There are no point-scoring opportunities after eating the dot (within the horizon, two here)
  - Therefore, waiting seems just as good as eating: he may go east, then back west in the next round of replanning!
Video of Demo Thrashing -- Fixed (d=2)
Video of Demo Smart Ghosts (Coordination)
Video of Demo Smart Ghosts (Coordination) – Zoomed In
Evaluation Functions
Depth Matters

- Evaluation functions are always imperfect.
- The deeper in the tree the evaluation function is buried, the less the quality of the evaluation function matters.
- An important example of the tradeoff between complexity of features and complexity of computation.

[Demo: depth limited (L6D4, L6D5)]
Other Game Types
Multi-Agent Utilities

- What if the game is not zero-sum, or has multiple players?

- Generalization of minimax:
  - Terminals have utility tuples
  - Node values are also utility tuples
  - Each player maximizes its own component
  - Can give rise to cooperation and competition dynamically…

```
1,6,6
       / \          / \          / \          / \          / \
1,6,6 | 7,1,2 | 6,1,2 | 7,2,1 | 5,1,7 | 1,5,2 | 7,7,1 | 5,2,5
```

1,6,6
Uncertain Outcomes
Worst-Case vs. Average Case

Idea: Uncertain outcomes controlled by chance, not an adversary!
Why wouldn’t we know what the result of an action will be?
- Explicit randomness: rolling dice
- Unpredictable opponents: the ghosts respond randomly
- Unpredictable humans: humans are not perfect
- Actions can fail: when moving a robot, wheels might slip

Values should now reflect average-case (expectimax) outcomes, not worst-case (minimax) outcomes

**Expectimax search:** compute the average score under optimal play
- Max nodes as in minimax search
- Chance nodes are like min nodes but the outcome is uncertain
- Calculate their expected utilities
- I.e. take weighted average (expectation) of children

Later, we’ll learn how to formalize the underlying uncertain-result problems as **Markov Decision Processes**
Video of Demo Minimax vs Expectimax (Min)
Video of Demo Minimax vs Expectimax (Exp)
**Expectimax Pseudocode**

```python
def value(state):
    if the state is a terminal state: return the state’s utility
    if the next agent is MAX: return max-value(state)
    if the next agent is EXP: return exp-value(state)

def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v

def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```
def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v

v = (1/2) (8) + (1/3) (24) + (1/6) (-12) = 10
Expectimax Example
Expectimax Pruning?
Depth-Limited Expectimax

Estimate of true expectimax value (which would require a lot of work to compute)
What Probabilities to Use?

- 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 chance node for any outcome out of our control: opponent or environment
  - The model might say that adversarial actions are likely!

- For now, assume each chance node magically comes along with probabilities that specify the distribution over its outcomes

*Having a probabilistic belief about another agent’s action does not mean that the agent is flipping any coins!*
Quiz: Informed Probabilities

- Let’s say you know that your opponent is actually running a depth 2 minimax, using the result 80% of the time, and moving randomly otherwise
- Question: What tree search should you use?

Answer: Expectimax!

- To figure out EACH chance node’s probabilities, you have to run a simulation of your opponent
- This kind of thing gets very slow very quickly
- Even worse if you have to simulate your opponent simulating you...
- ... except for minimax and maximax, which have the nice property that it all collapses into one game tree

This is basically how you would model a human, except for their utility: their utility might be the same as yours (i.e. you try to help them, but they are depth 2 and noisy), or they might have a slightly different utility (like another person navigating in the office)
Modeling Assumptions
The Dangers of Optimism and Pessimism

Dangerous Optimism
Assuming chance when the world is adversarial

Dangerous Pessimism
Assuming the worst case when it’s not likely
Assumptions vs. Reality

Pacman used depth 4 search with an eval function that avoids trouble
Ghost used depth 2 search with an eval function that seeks Pacman

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

[Results from playing 5 games]

[Demos: world assumptions (L7D3,4,5,6)]
# Assumptions vs. Reality

![Pacman and Ghost game](image)

<table>
<thead>
<tr>
<th></th>
<th>Adversarial 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: 483</td>
<td>Avg. Score: 493</td>
</tr>
<tr>
<td><strong>Expectimax Pacman</strong></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>

Results from playing 5 games

Pacman used depth 4 search with an eval function that avoids trouble
Ghost used depth 2 search with an eval function that seeks Pacman

[Demos: world assumptions (L7D3,4,5,6)]
Video of Demo World Assumptions
Random Ghost – Expectimax Pacman
Video of Demo World Assumptions
Adversarial Ghost – Minimax Pacman
Video of Demo World Assumptions
Adversarial Ghost – Expectimax Pacman
Video of Demo World Assumptions
Random Ghost – Minimax Pacman
Why not minimax?

- Worst case reasoning is too conservative
- Need average case reasoning
Mixed Layer Types

- E.g. Backgammon
- Expectiminimax
  - Environment is an extra “random agent” player that moves after each min/max agent
  - Each node computes the appropriate combination of its children
Example: Backgammon

- Dice rolls increase by 21 possible rolls with 2 dice
  - Backgammon ≈ 20 legal moves
  - Depth 2 = 20 × (21 × 20)^3 = 1.2 × 10^9

- As depth increases, probability of reaching a given search node shrinks
  - So usefulness of search is diminished
  - So limiting depth is less damaging
  - But pruning is trickier…

- Historic AI: TDGammon uses depth-2 search + very good evaluation function + reinforcement learning:
  - world-champion level play

- 1st AI world champion in any game!

Utilities
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 “rational” preferences can be summarized as a utility function

- We hard-wire utilities and let behaviors emerge
  - Why don’t we let agents pick utilities?
  - Why don’t we prescribe behaviors?