Recap: Minimax
Resource Limits
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:

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

- 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...
Uncertain Outcomes
Worst-Case vs. Average Case

Idea: Uncertain outcomes controlled by chance, not an adversary!
Expectimax Search

- 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:
        \( p = \text{probability}(\text{successor}) \)
        \( v += p * \text{value}(\text{successor}) \)
    return \( v \)

\[
v = \frac{1}{2} (8) + \frac{1}{3} (24) + \frac{1}{6} (-12) = 10
\]
Expectimax Example
Expectimax Pruning?
Depth-Limited Expectimax

Estimate of true expectimax value (which would require a lot of work to compute)
Probabilities
Reminder: Probabilities

- 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: \( P(T=\text{none}) = 0.25, P(T=\text{light}) = 0.50, P(T=\text{heavy}) = 0.25 \)

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:
- \( P(T=\text{heavy}) = 0.25, P(T=\text{heavy} \mid \text{Hour=8am}) = 0.60 \)
- We’ll talk about methods for reasoning and updating probabilities later
Reminder: Expectations

- The expected value of a function of a random variable is the average, weighted by the probability distribution over outcomes.

- Example: How long to get to the airport?

<table>
<thead>
<tr>
<th>Time</th>
<th>Probability</th>
<th>Calculation</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 min</td>
<td>0.25</td>
<td>$20 \times 0.25$</td>
</tr>
<tr>
<td>30 min</td>
<td>0.50</td>
<td>$30 \times 0.50$</td>
</tr>
<tr>
<td>60 min</td>
<td>0.25</td>
<td>$60 \times 0.25$</td>
</tr>
</tbody>
</table>

**Total:**

$$\text{Expected Time} = 20 \times 0.25 + 30 \times 0.50 + 60 \times 0.25 = 35 \text{ min}$$
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

<table>
<thead>
<tr>
<th></th>
<th>Adversarial 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>

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)]
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>Algorithm</th>
<th>Adversarial 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: 483</td>
<td>Avg. Score: 493</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>

Results from playing 5 games

[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 $b$: 21 possible rolls with 2 dice
  - Backgammon $\approx 20$ legal moves
  - Depth 2 = $20 \times (21 \times 20)^3 = 1.2 \times 10^9$

- 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 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?