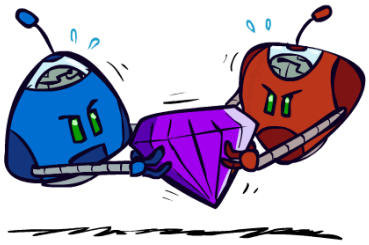
CSE 473: Artificial Intelligence

Expecti-Max Search for Stochastic Games

Dan Weld



Based on slides from

Mausam, Andrey Kolobov, Dan Klein, Stuart Russell, Pieter Abbeel,

(best illustrations from ai.berkeley.edu)

Outline

- Adversarial Games
  - Minimax search
  - α-β search
  - Evaluation functions
  - Multi-player, non-0-sum
- Stochastic Games
  - Expectimax
  - Markov Decision Processes
  - Reinforcement Learning

Utilities

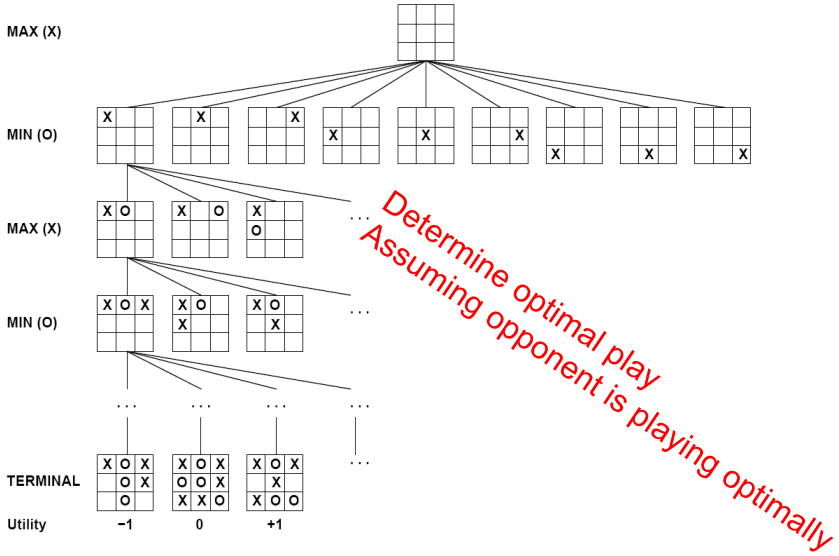
- Utility function describes an agent's preferences
- Function from outcomes (states of the world) to $\mathbb{R}$
  (so we can take an expectation)
- In a game, may be simple (+1/-1)
- In general, we hard-wire an agent's utilities
  - let its actions emerge

Types of "Games"

	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon, monopoly
imperfect information	stratego	bridge, poker, scrabble, nuclear war

Number of Players? 1, 2, …?

Tic-tac-toe Game Tree



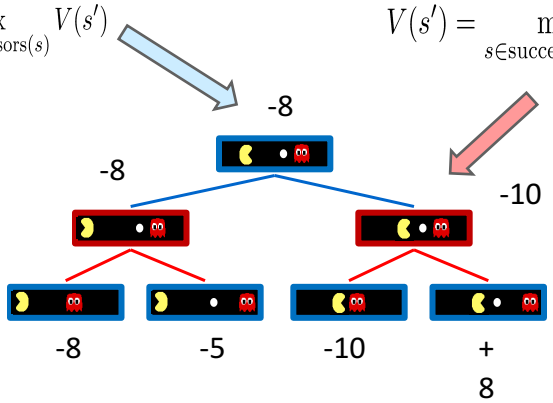
Minimax Values

States Under Agent's Control:

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

States Under Opponent's Control:

$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$



Terminal States:

$$V(s) = \text{known}$$

Slide from Dan Klein & Pieter Abbeel - ai.berkeley.edu

Minimax Implementation

```
def max-value(state):
    if leaf?(state), return U(state)
    initialize v = -∞
    for each c in children(state)
        v = max(v, min-value(c))
    return v
```

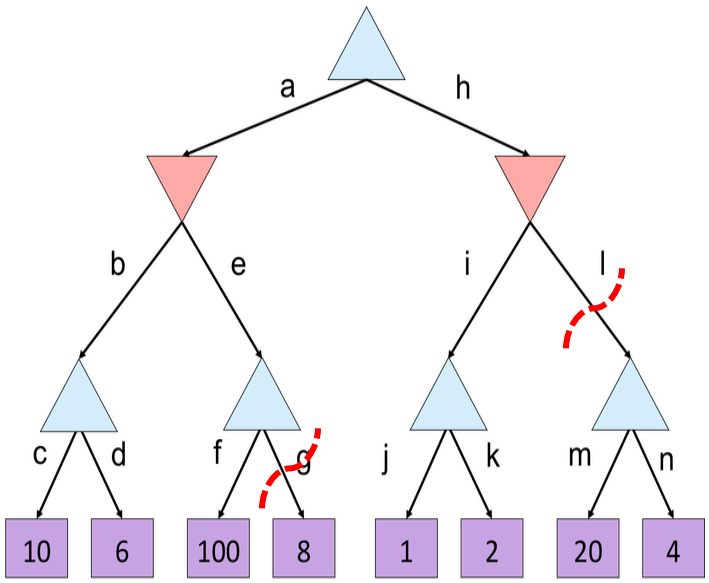
```
def min-value(state):
    if leaf?(state), return U(state)
    initialize v = +∞
    for each c in children(state)
        v = min(v, max-value(c))
    return v
```

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$

Slide from Dan Klein & Pieter Abbeel - ai.berkeley.edu

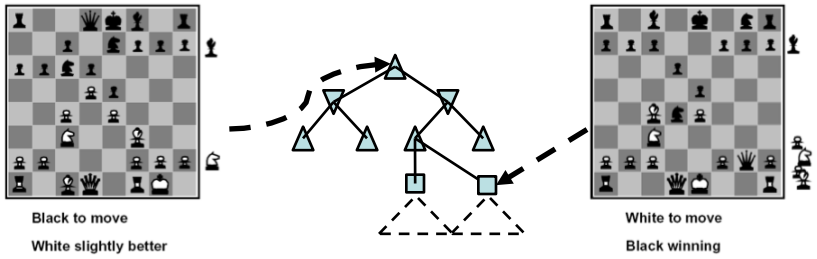
Alpha-Beta Example



Slide from Dan Klein & Pieter Abbeel - ai.berkeley.edu

When can't search to terminal states, use Heuristic Evaluation Function

- Function which scores non-terminals

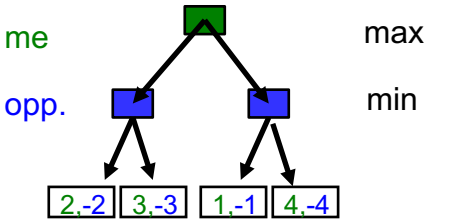


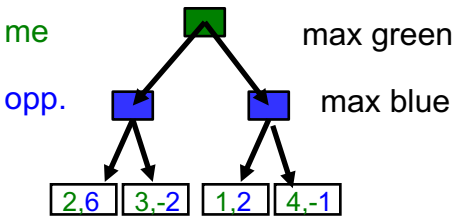
- Ideal function: returns the true utility of the position
- In practice: typically weighted linear sum of features:
  - e.g. $f_1(s)$ = (num white queens – num black queens), etc.

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

Non- Zero-Sum Games

- Zero-Sum games
  - My gain is your loss
  - Utility(me) - Utility(you) = 0
- General Case
  - "Non-zero sum"
  - We each maximize our own utilities

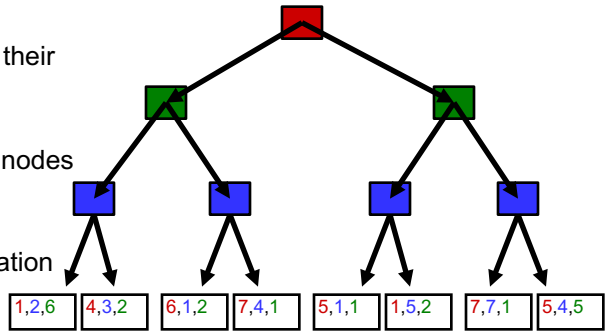




Multi-player Non-Zero-Sum Games

Again, 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…

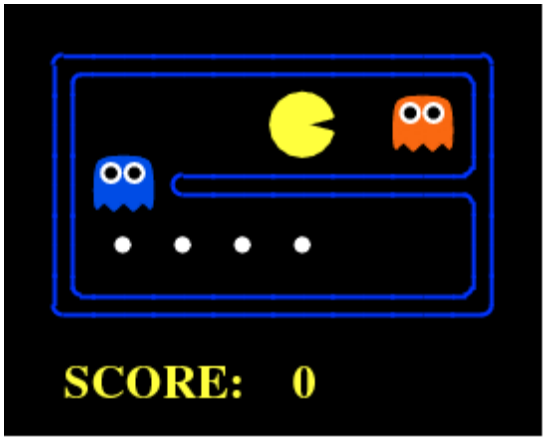


Stochastic Games

Much more useful model than "games"

	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon, monopoly
imperfect information		bridge, poker, scrabble, nuclear war

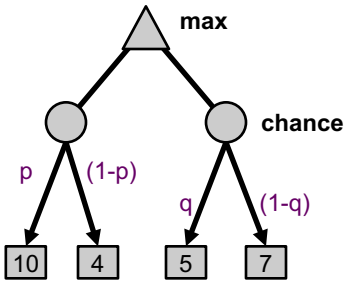
Alpha-Beta



3 ply look ahead, ghosts move 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



- Soon, we'll learn how to formalize this underlying problem as a **Markov Decision Process**

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

X	P	f
1	1/6	1
2	1/6	2
3	1/6	3
4	1/6	4
5	1/6	5
6	1/6	6

$$1 \times \frac{1}{6} + 2 \times \frac{1}{6} + 3 \times \frac{1}{6} + 4 \times \frac{1}{6} + 5 \times \frac{1}{6} + 6 \times \frac{1}{6}$$

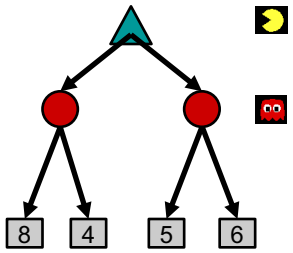
$$= 3.5$$

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)
```

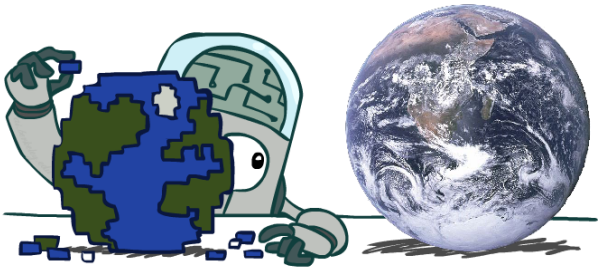


Expectimax: wins some of the time

QuickTime™ and a
GIF decompressor
are needed to see this picture.

3 ply look ahead, ghosts move randomly

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

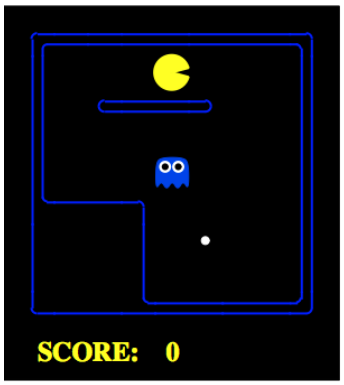
Expectimax for Pacman

- 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: Is minimax 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

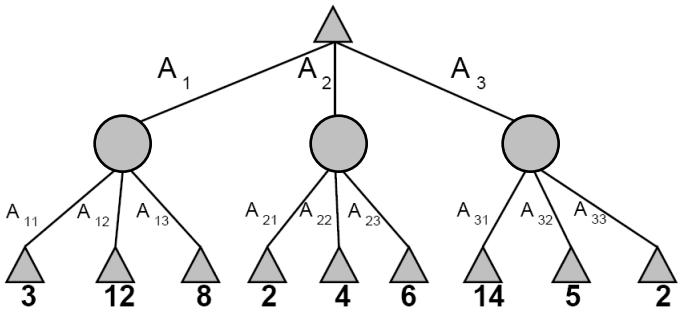
Results from playing 5 games

	Minimizing Ghost	Random Ghost
Minimax Pacman	Won 5/5 Avg. Score: 493	Won 5/5 Avg. Score: 483
Expectimax Pacman	Won 1/5 Avg. Score: -303	Won 5/5 Avg. Score: 503



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

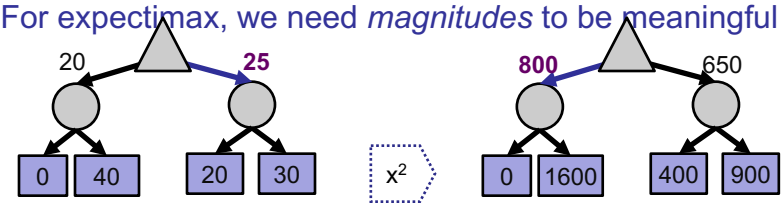
Expectimax Pruning?



- 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

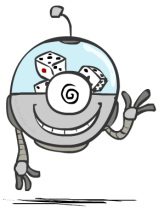


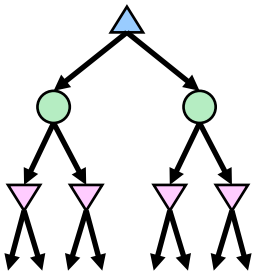
Example: Backgammon

Image: Wikipedia

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



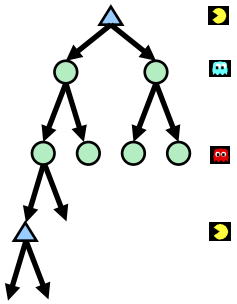


Example: Backgammon

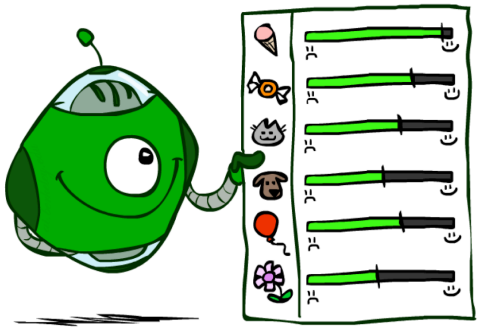
- Dice rolls increase *b*: 21 possible rolls with 2 dice
  - Backgammon ≈ 20 legal moves
  - Depth 2 = 20 x $(21 \times 20)^3 = 1.2 \times 10^9$
- As depth increases, probability of reaching a given search node shrinks
  - So usefulness of search is diminished
  - So limiting depth is less damaging
  - But pruning is trickier…
- Historic AI (1992): TDGammon uses depth-2 search + very good evaluation function + reinforcement learning: world-champion level play
- 1st AI world champion in any game!

Image: Wikipedia

Different Types of Ghosts



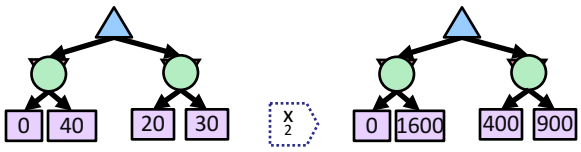
Utilities



Maximum Expected Utility

- Why should we average utilities?
- Principle of maximum expected utility:
  - A rational agent should chose the action that maximizes its expected utility, given its knowledge
- Questions:
  - Where do utilities come from?
  - How do we know such utilities even exist?
  - How do we know that averaging even makes sense?
  - What if our behavior (preferences) can't be described by utilities?

What Utilities to Use?

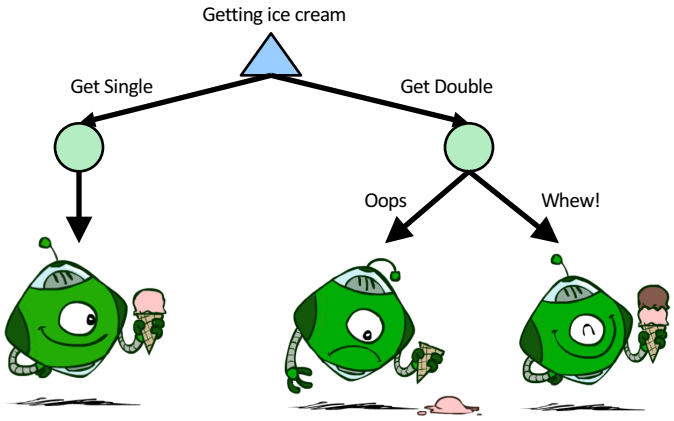


- For worst-case minimax reasoning, terminal function scale doesn't matter
  - We just want better states to have higher evaluations (get the ordering right)
  - We call this insensitivity to monotonic transformations
- For average-case expectimax reasoning, we need *magnitudes* to be meaningful

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?

Utilities: Uncertain Outcomes

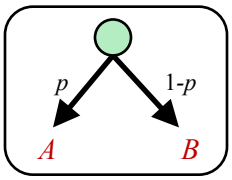


Preferences

- An agent must have preferences among:
  - Prizes: A, B, etc.
  - Lotteries: $L = [p, A;\ (1-p), B]$ certain prizes

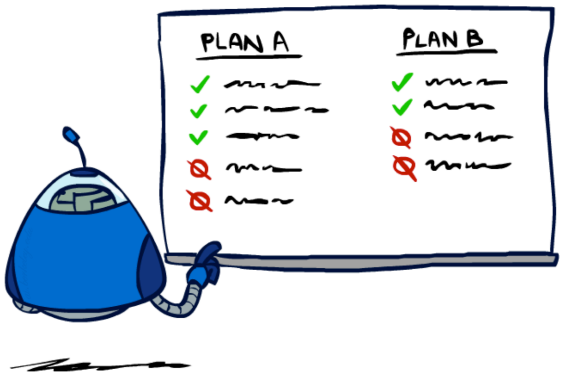
A Prize

A Lottery



- Notation:
  - Preference: $A \succ B$
  - Indifference: $A \sim B$

Rationality

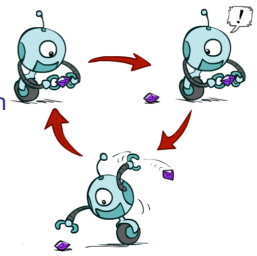


Rational Preferences

- We want some constraints on preferences before we call them rational, such as:

  Axiom of Transitivity: $(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$

- For example: an agent with intransitive preferences can be induced to give away all of its money
  - If B > C, then an agent with C would pay (say) 1 cent to get B
  - If A > B, then an agent with B would pay (say) 1 cent to get A
  - If C > A, then an agent with A would pay (say) 1 cent to get C



Rational Preferences

The Axioms of Rationality

Orderability
$(A \succ B) \vee (B \succ A) \vee (A \sim B)$
Transitivity
$(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$
Continuity
$A \succ B \succ C \Rightarrow \exists p\ [p, A;\ 1-p, C] \sim B$
Substitutability
$A \sim B \Rightarrow [p, A;\ 1-p, C] \sim [p, B;\ 1-p, C]$
Monotonicity
$A \succ B \Rightarrow$
$(p \geq q \Leftrightarrow [p, A;\ 1-p, B] \succeq [q, A;\ 1-q, B])$

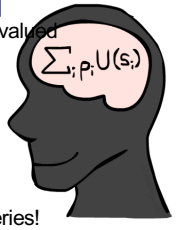
Theorem: Rational preferences imply behavior describable as maximization of expected utility

MEU Principle

- Theorem [Ramsey, 1931; von Neumann & Morgenstern, 1944]
  - Given any preferences satisfying these constraints, there exists a real-valued function U such that:

$$U(A) \geq U(B) \;\Leftrightarrow\; A \succeq B$$

$$U([p_1, S_1;\; \ldots\; ;\; p_n, S_n]) = \sum_i p_i U(S_i)$$



  - I.e. values assigned by U preserve preferences of both prizes and lotteries!

- Maximum expected utility (MEU) principle:
  - Choose the action that maximizes expected utility
  - Note: an agent can be entirely rational (consistent with MEU) without ever representing or manipulating utilities and probabilities
  - E.g., a lookup table for perfect tic-tac-toe, a reflex vacuum cleaner