CSE 573: Artificial Intelligence

Autumn 2010

Lecture 5: Expectimax Search
10/14/2008

Luke Zettlemoyer

Most slides over the course adapted from either Dan Klein, Stuart Russell or Andrew Moore

Announcements

- PS1 due tomorrow, 5pm
  - DropBox instructions are on assignment page
  - No late assignments
    - Email Luke for extension (requires good reason)
- PS2 will go out soon
- MDP/RL Readings
  - will assign chapters from RL Book by Sutton & Barto, freely available online:
    - http://webdocs.cs.ualberta.ca/~sutton/book/the-book.html

Outline

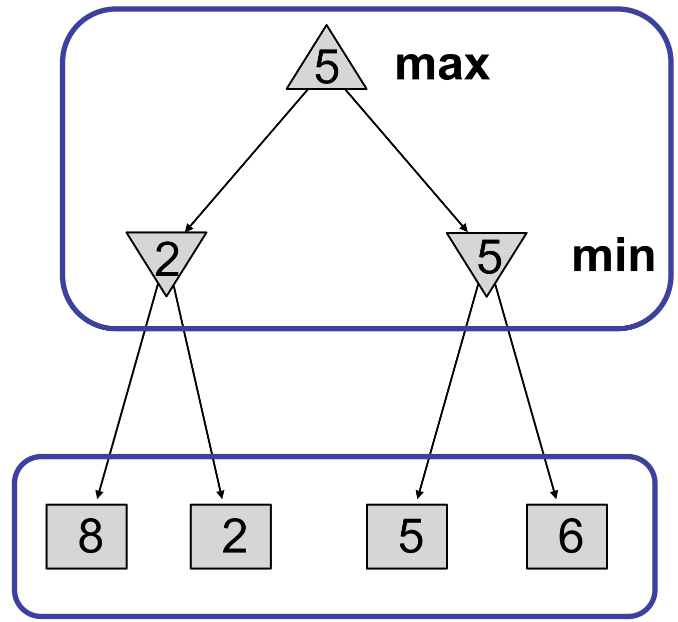
- Review adversarial search
  - Trace alpha/beta
- Review probabilities / expectations
- Expectimax search (one and two player)
- Rational preferences

Adversarial Games

- Deterministic, zero-sum games:
  - Tic-tac-toe, chess, checkers
  - One player maximizes result
  - The other minimizes result

- Minimax search:
  - A state-space search tree
  - Players alternate turns
  - Each node has a minimax value: best achievable utility against a rational adversary

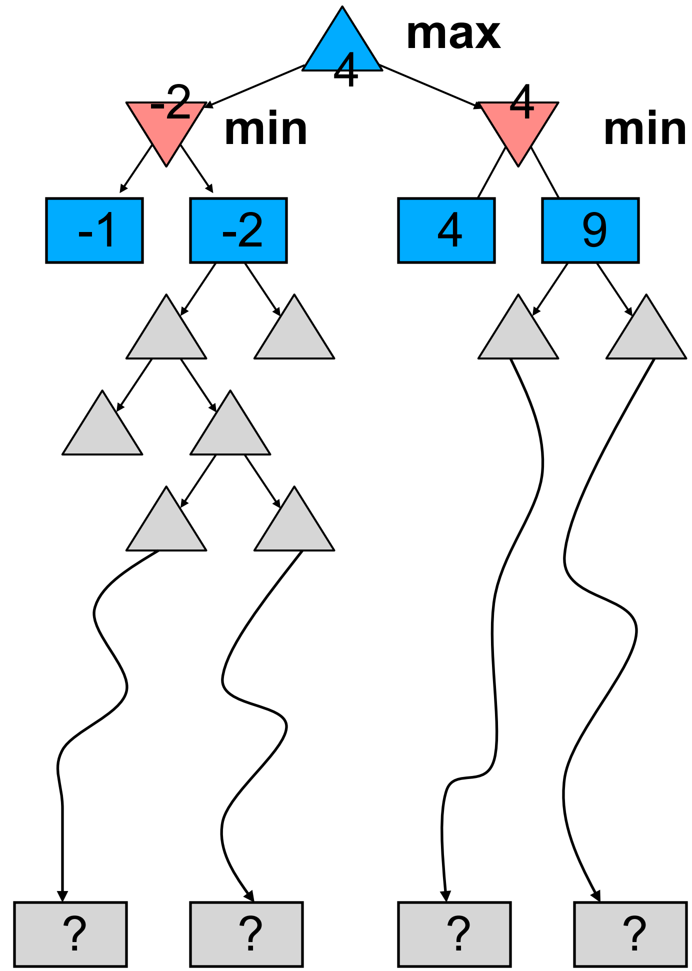
Minimax values: computed recursively



Terminal values: part of the game

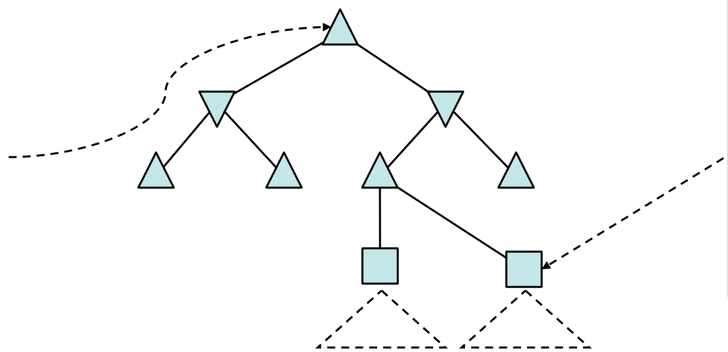
Recap: Resource Limits

- Cannot search to leaves
- Depth-limited search
  - Instead, search a limited depth of tree
  - Replace terminal utilities with an eval function for non-terminal positions
- Guarantee of optimal play is gone
- Replanning agents:
  - Search to choose next action
  - Replan each new turn in response to new state



Evaluation Functions

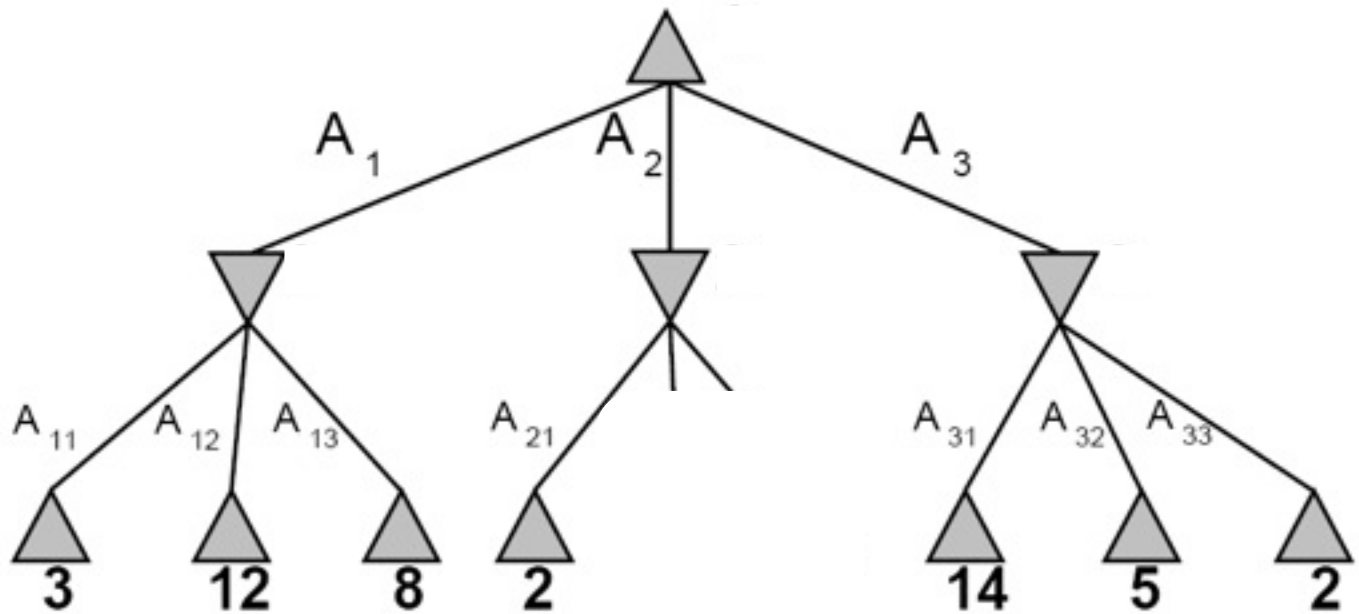
- Function which scores non-terminals



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

- Ideal function: returns the utility of the position
- Typically weighted linear sum of features:
  - number of pawns, rooks, etc.

Pruning for Minimax



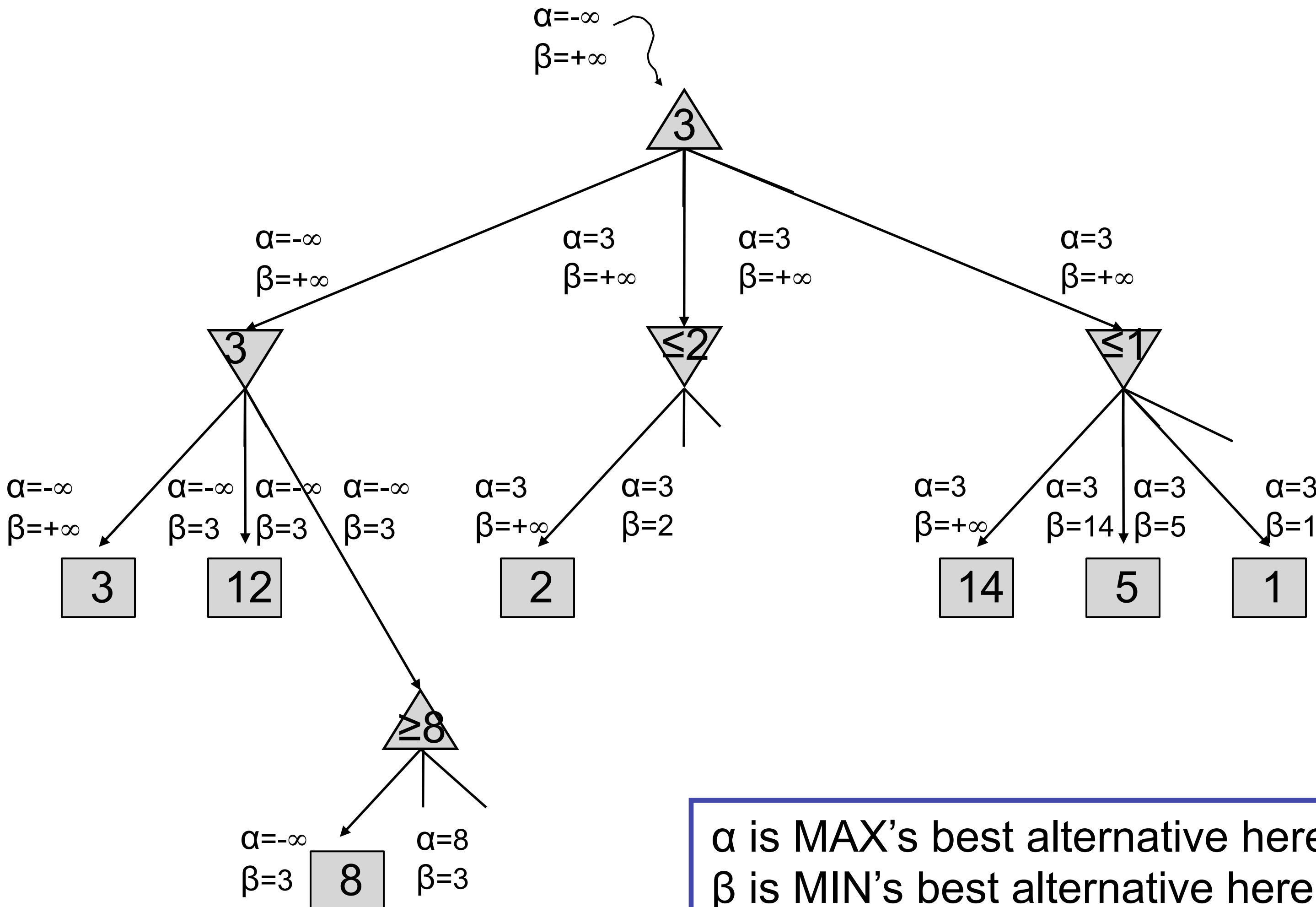
Alpha-Beta Pseudocode

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
```

Alpha-Beta Pruning Example



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

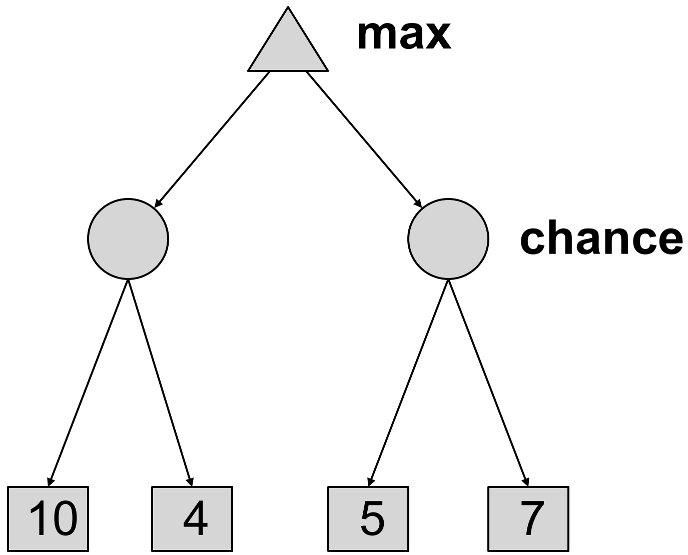
- Values of intermediate nodes might be wrong!
  - but, they are bounds

- 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…

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



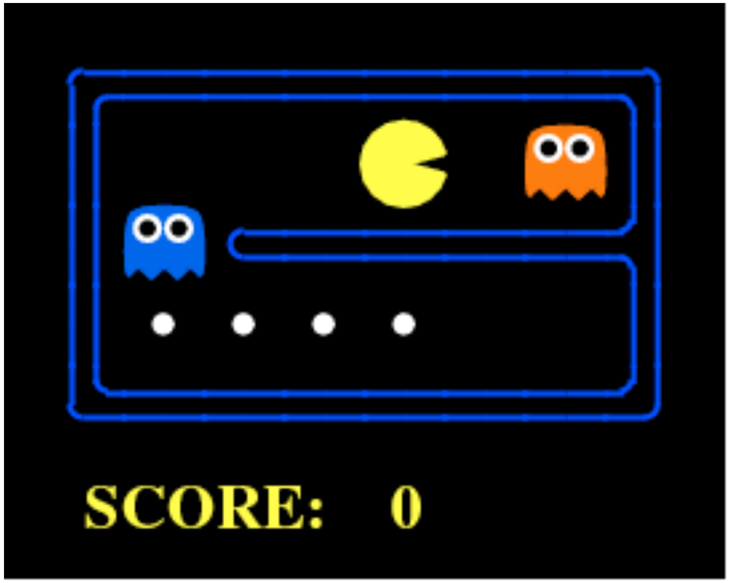
- Later, we'll learn how to formalize the underlying problem as a **Markov Decision Process**

Maximum Expected Utility

- Why should we average utilities? Why not minimax?

Which Algorithm?

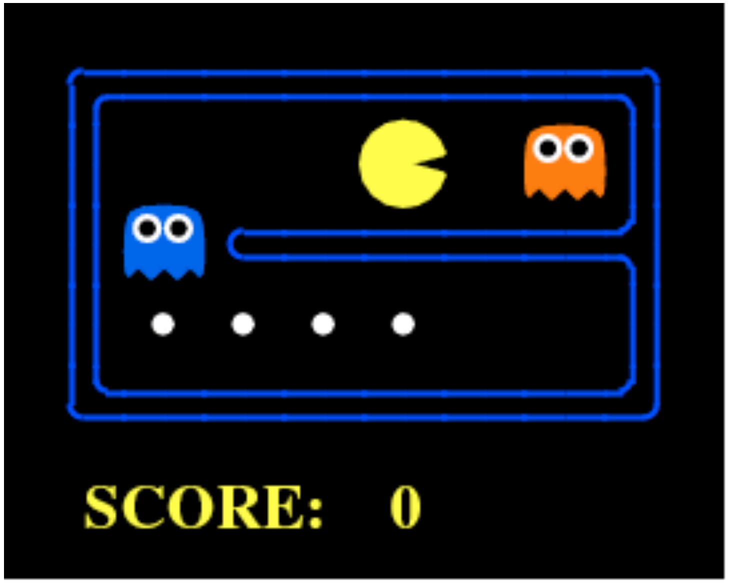
Minimax: no point in trying



3 ply look ahead, ghosts move randomly

Which Algorithm?

Expectimax: wins some of the time



3 ply look ahead, ghosts move randomly

Maximum Expected Utility

- Why should we average utilities? Why not minimax?
- Principle of maximum expected utility: an agent should chose 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…

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=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:
  - Averages over repeated *experiments*
  - E.g. empirically estimating P(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

- 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(none) = 20, L(light) = 30, L(heavy) = 60
  - What is my expected driving time?
    - Notation: $E_{P(T)}[\ L(T)\ ]$
    - Remember, P(T) = {none: 0.25, light: 0.5, heavy: 0.25}
    - E[L(T)] = L(none) * P(none) + L(light) * P(light) + L(heavy) * P(heavy)
    - E[L(T)] = (20 * 0.25) + (30 * 0.5) + (60 * 0.25) = 35

Utilities

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

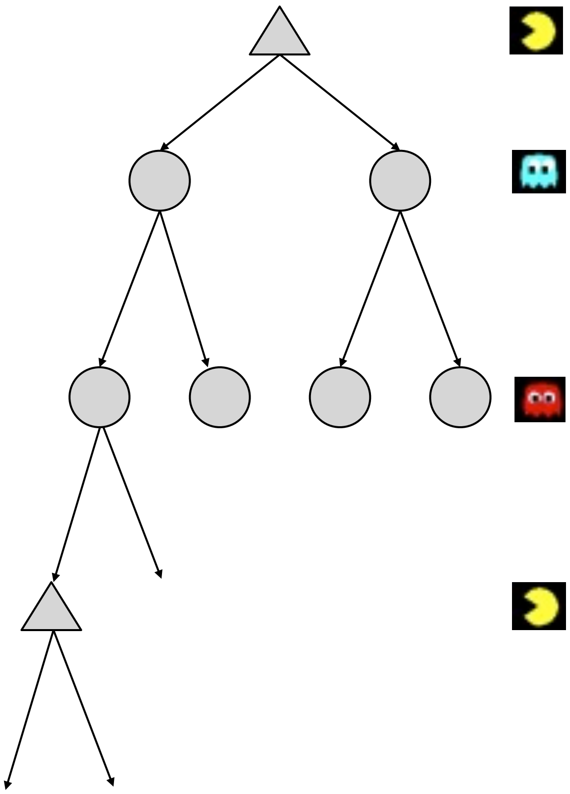
- Where do utilities come from?
  - In a game, may be simple (+1/-1)
  - Utilities summarize the agent's goals
  - Theorem: any set of preferences between outcomes can be summarized as a utility function (provided the preferences meet certain conditions)

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

- More on utilities soon…

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

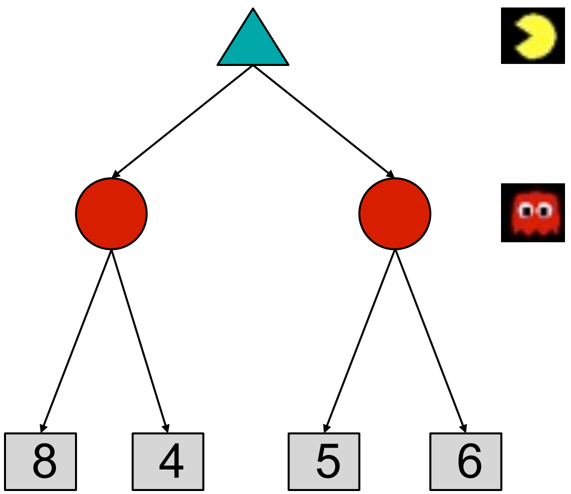


Expectimax Pseudocode

```
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)
```



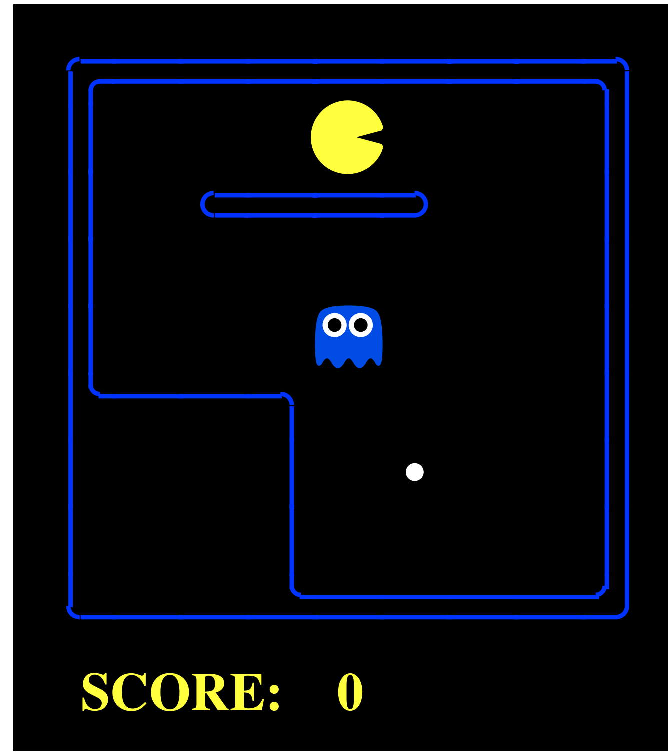
Expectimax for Pacman

- 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

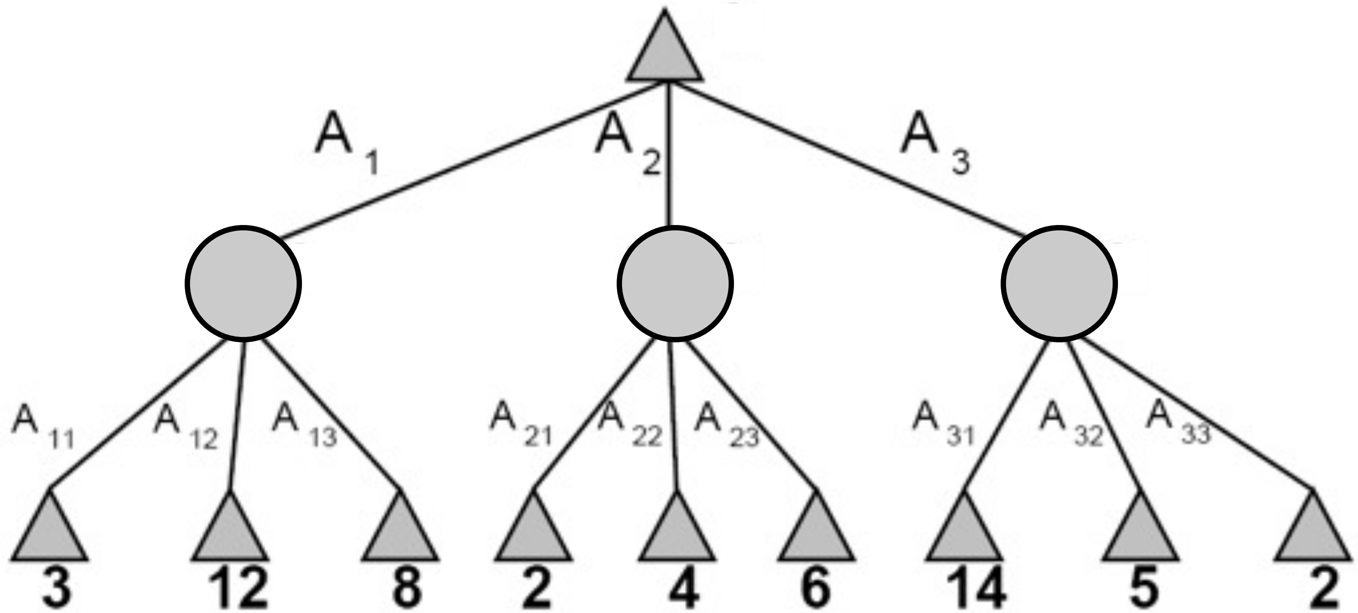
Results from playing 5 games

	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

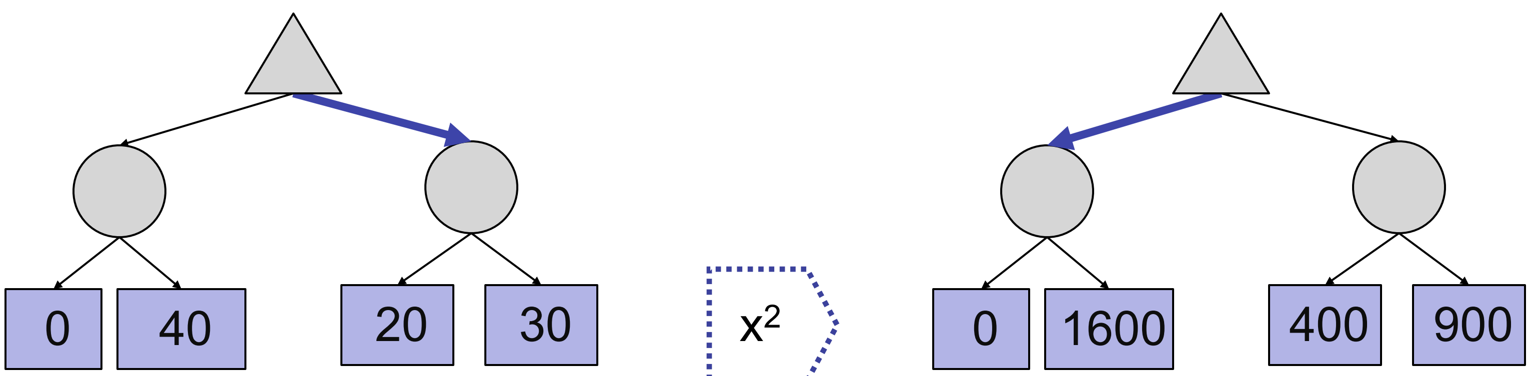
Expectimax Pruning?



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

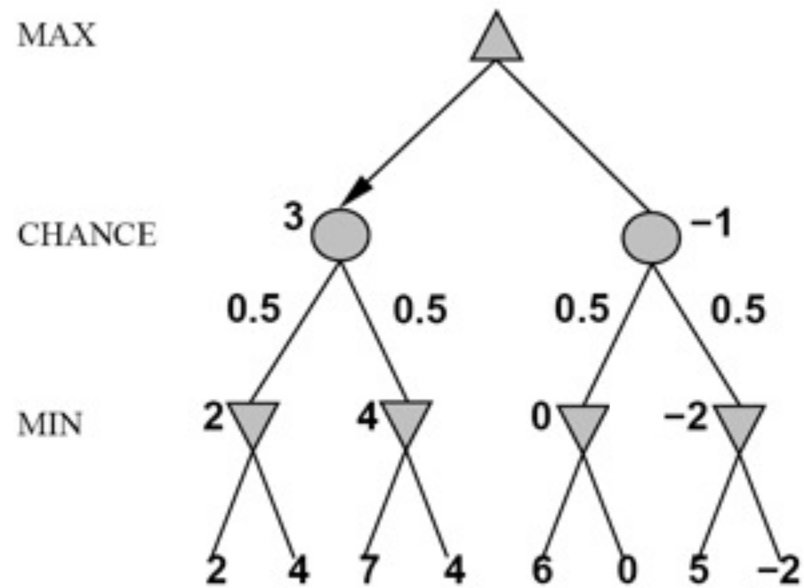
Expectimax Evaluation

- 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



Mixed Layer Types

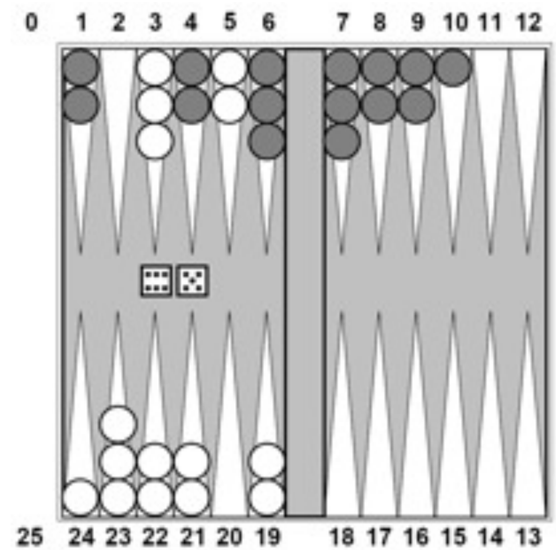
- E.g. Backgammon
- Expectiminimax
  - Environment is an extra player that moves after each agent
  - Chance nodes take expectations, otherwise like minimax



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*)

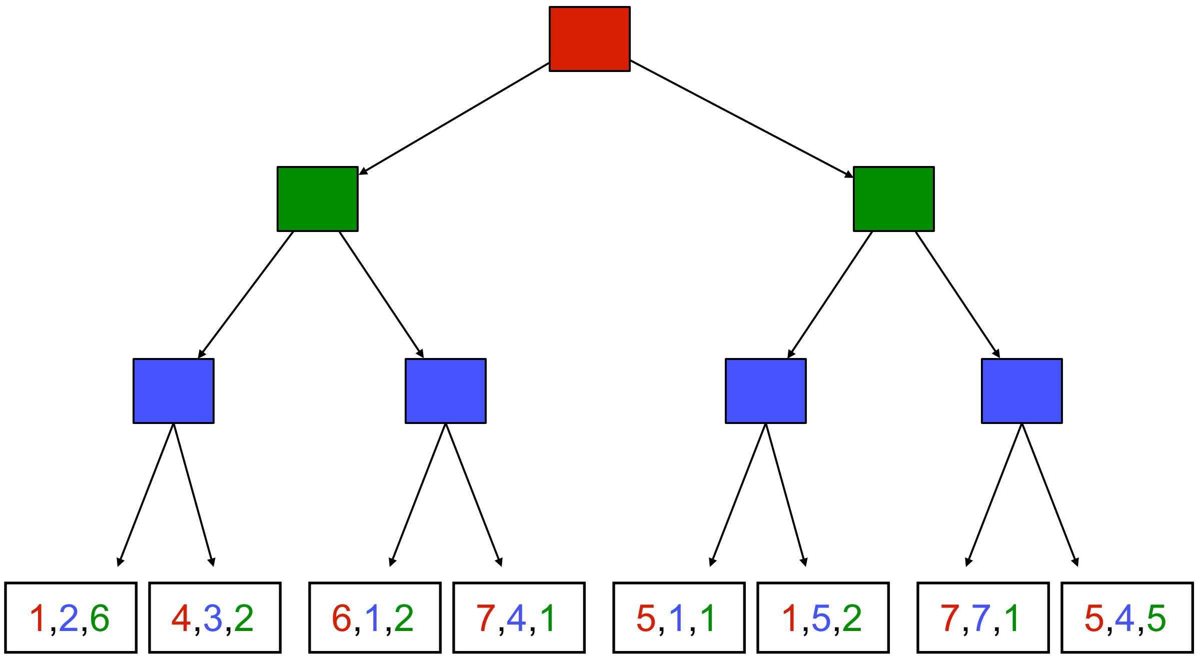
Stochastic Two-Player

- Dice rolls increase *b*: 21 possible rolls with 2 dice
  - Backgammon ≈ 20 legal moves
  - Depth 4 = $20 \text{ x } (21 \text{ x } 20)^3$ $1.2 \text{ x } 10^9$
- 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



Non-Zero-Sum Games

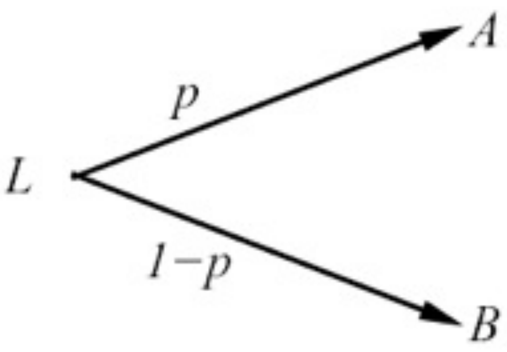
- Similar to minimax:
  - Utilities are now tuples
  - Each player maximizes their own entry at each node
  - Propagate (or back up) nodes from children
  - Can give rise to cooperation and competition dynamically…



Preferences

- An agent chooses among:
  - Prizes: A, B, etc.
  - Lotteries: situations with uncertain prizes

$$L = [p, A;\ (1-p), B]$$



- Notation:

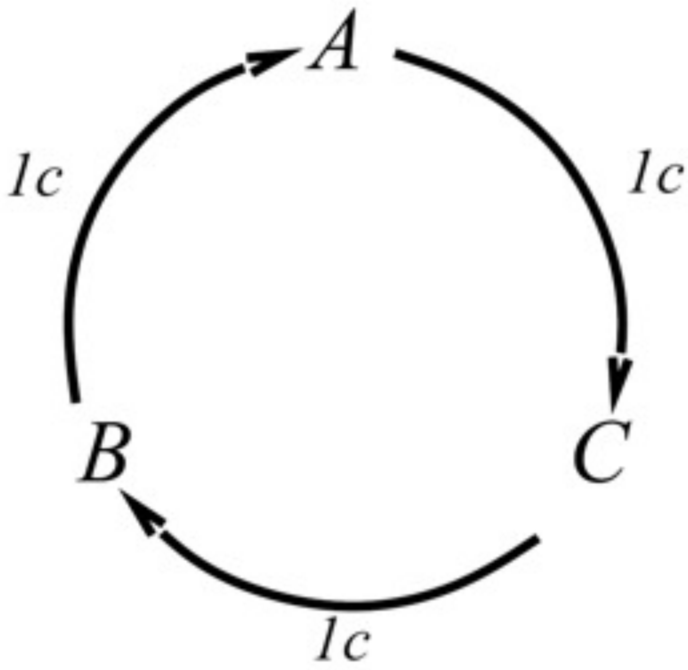
$A \succ B$ A preferred over B

$A \sim B$ indifference between A and B

$A \succeq B$ B not preferred over A

Rational Preferences

- We want some constraints on preferences before we call them rational
- For example: an agent with intransitive preferences can be induced to give away all 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

- Preferences of a rational agent must obey constraints.
  - 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:

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

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

- Maximum expected likelihood (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 tictactoe, reflex vacuum cleaner

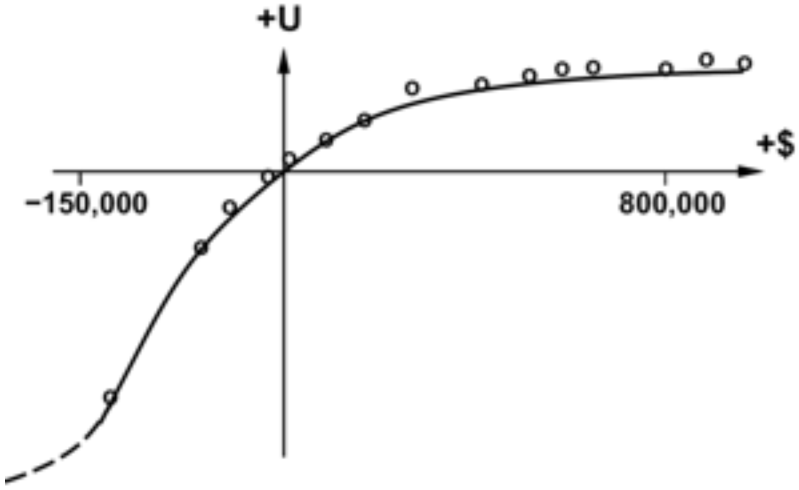
Utility Scales

- Normalized utilities: $u_+ = 1.0$, $u_- = 0.0$
- Micromorts: one-millionth chance of death, useful for paying to reduce product risks, etc.
- QALYs: quality-adjusted life years, useful for medical decisions involving substantial risk
- Note: behavior is invariant under positive linear transformation

$$U'(x) = k_1 U(x) + k_2 \quad \text{where } k_1 > 0$$

Money

- Money does not behave as a utility function
- Given a lottery L:
  - Define expected monetary value EMV(L)
  - Usually U(L) < U(EMV(L))
  - I.e., people are risk-averse
- Utility curve: for what probability p am I indifferent between:
  - A prize x
  - A lottery [p,$M; (1-p),$0] for large M?
- Typical empirical data, extrapolated with risk-prone behavior:



Example: Human Rationality?

- Famous example of Allais (1953)

  - A: [0.8,$4k; 0.2,$0]
  - B: [1.0,$3k; 0.0,$0]

  - C: [0.2,$4k; 0.8,$0]
  - D: [0.25,$3k; 0.75,$0]

- Most people prefer B > A, C > D
- But if U($0) = 0, then
  - B > A $\Rightarrow$ U($3k) > 0.8 U($4k)
  - C > D $\Rightarrow$ 0.8 U($4k) > U($3k)