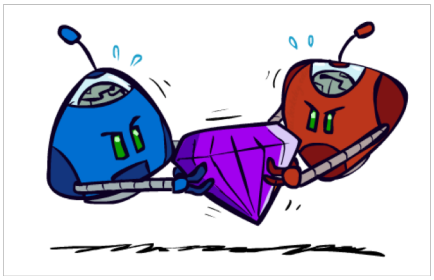
CSEP 573: Artificial Intelligence

Adversarial Search

Dan Weld



Based on slides from

Dan Klein, Stuart Russell, Pieter Abbeel, Andrew Moore and Luke Zettlemoyer

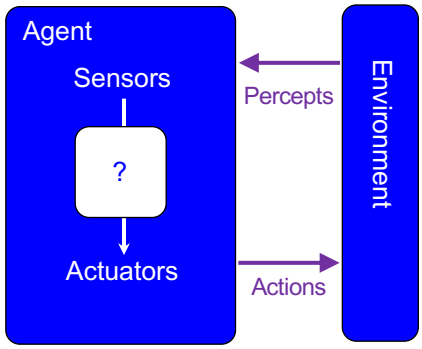
(best illustrations from ai.berkeley.edu)

Adversarial Search: Outline

- Minimax search
- α-β search
- Evaluation functions
- Expectimax
- More complex games

Types of Environments

- Fully observable *vs.* partially observable
- Single agent *vs.* ***multi-agent***
- Deterministic *vs.* ***stochastic***
- Episodic *vs.* sequential
- Discrete *vs.* continuous



Game Playing State-of-the-Art

1994: Checkers. Chinook ended 40-year-reign of human world champion Marion Tinsley. Used search plus 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. Checkers is now solved!

Game Playing State-of-the-Art

1997: Chess. Deep Blue defeated human world champion Gary Kasparov in a six-game match. Deep Blue examined 200 million positions per second, used very sophisticated evaluation and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic.

Game Playing State-of-the-Art

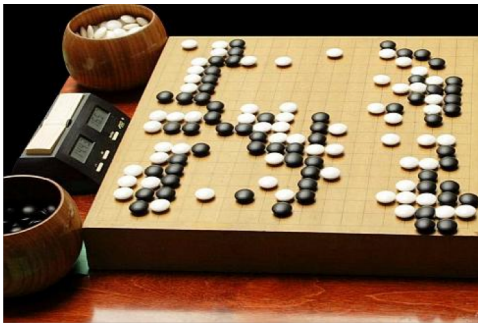
Go: b > 300! Programs use monte carlo tree search + pattern KBs

2015: AlphaGo beats European Go champion Fan Hui (2 dan) 5-0

2016: AlphaGo beats Lee Sedol (9 dan) 4-1

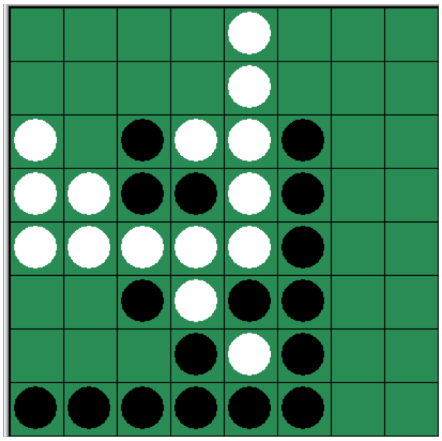
2018: AlphaZero

- Trained solely with self-play
- Training uses 5000 TPUs
- To play: 4 TPUs & 44-core CPU
- Also plays chess & shogi



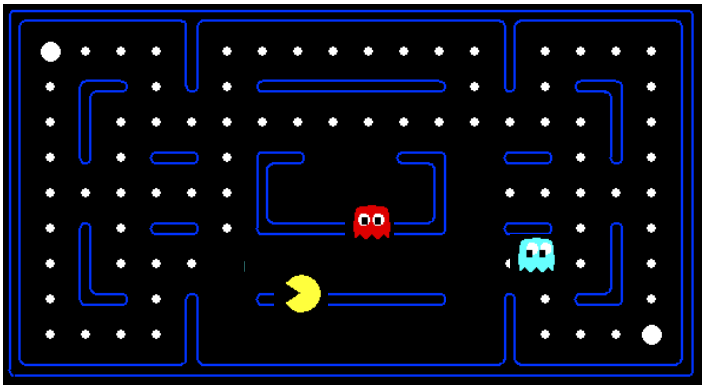
Game Playing State-of-the-Art

Othello: Human champions refuse to compete against computers.



Game Playing State-of-the-Art

- **Pacman:** … unknown …



Types of Games

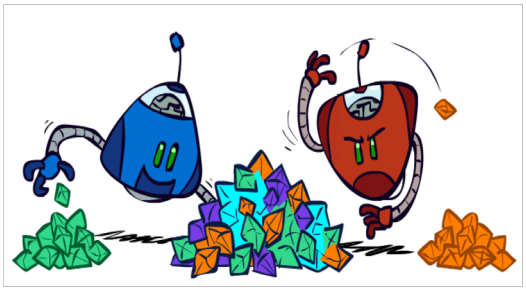
	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon, monopoly
imperfect information	stratego	bridge, poker, scrabble, nuclear war

Number of Players? 1, 2, …?

Deterministic Games

- Many possible formalizations, one is:
  - States: S (start at s_0)
  - Players: P={1...N} (usually take turns)
  - Actions: A (may depend on player / state)
  - Transition Function: S x A → S
  - Terminal Test: S → {t,f}
  - Terminal Utilities: S x P→ R
- Solution for a player is a ***policy***: S → A

Zero-Sum Games

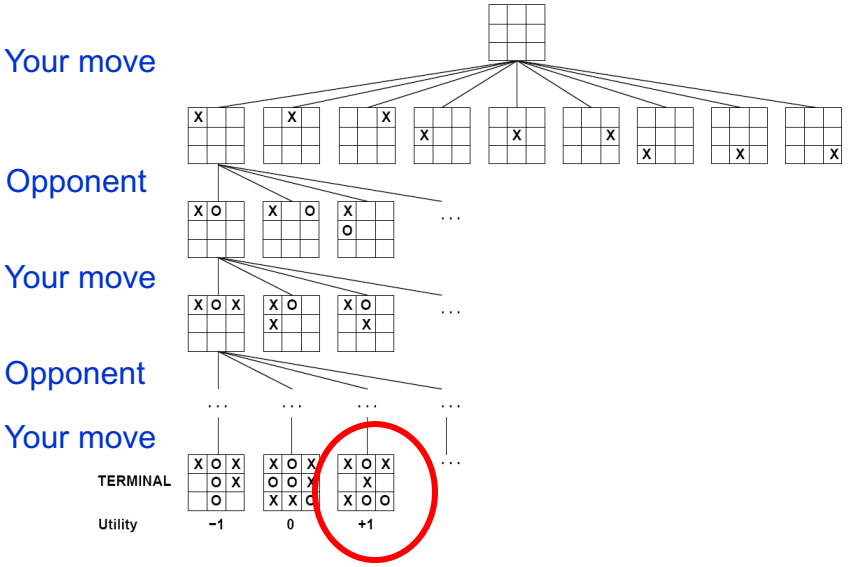


- Zero-Sum Games
  - Agents have opposite utilities (values on outcomes)
  - Lets us think of a single value that one maximizes and the other minimizes
  - Adversarial, pure competition
- General Games
  - Agents have independent utilities (values on outcomes)
  - Cooperation, indifference, competition, & more are possible
  - More later on non-zero-sum games

Deterministic Two-Player

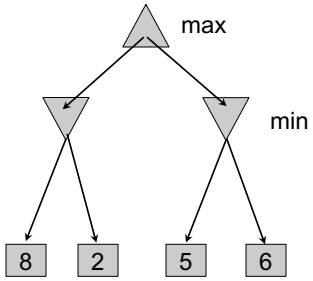
- E.g. tic-tac-toe, chess, checkers
- Zero-sum games
  - One player maximizes result
  - The other minimizes result

Depth First Search?



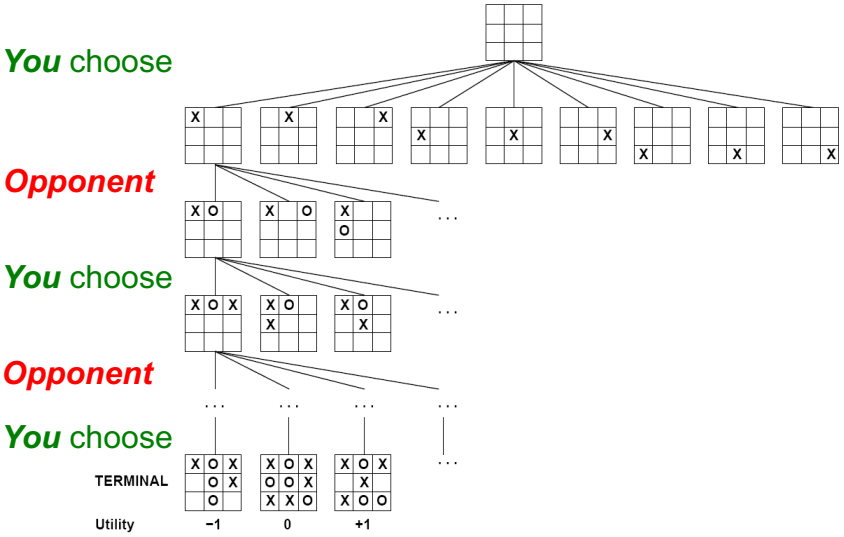
Deterministic Two-Player

- E.g. tic-tac-toe, chess, checkers
- Zero-sum games
  - One player maximizes result
  - The other minimizes result

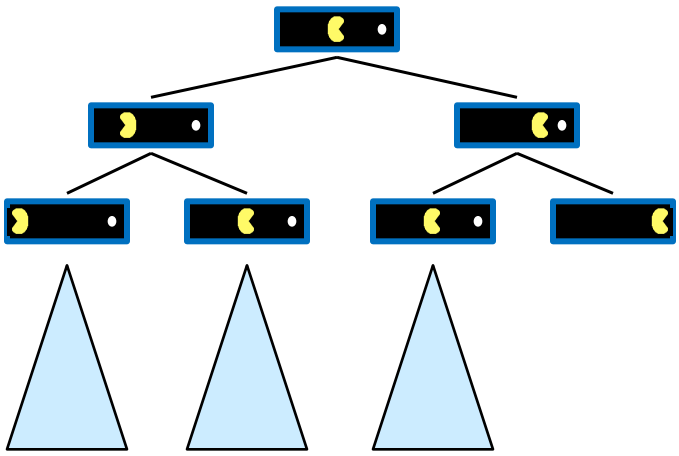


- ***Minimax search***
  - A state-space search tree
  - Players alternate
  - Choose move to position with highest minimax value
    = best achievable utility against best play

Tic-tac-toe Game Tree



Previously: Single-Agent Trees



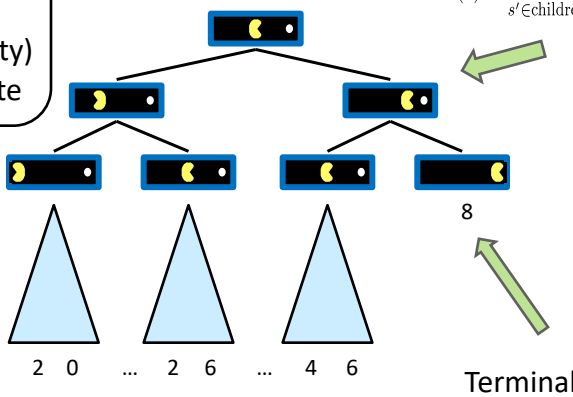
Slide from Dan Klein & Pieter Abbeel - ai.berkeley.edu

Previously: Value of a State

Value of a state: The best achievable outcome (utility) from that state

Non-Terminal States:

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

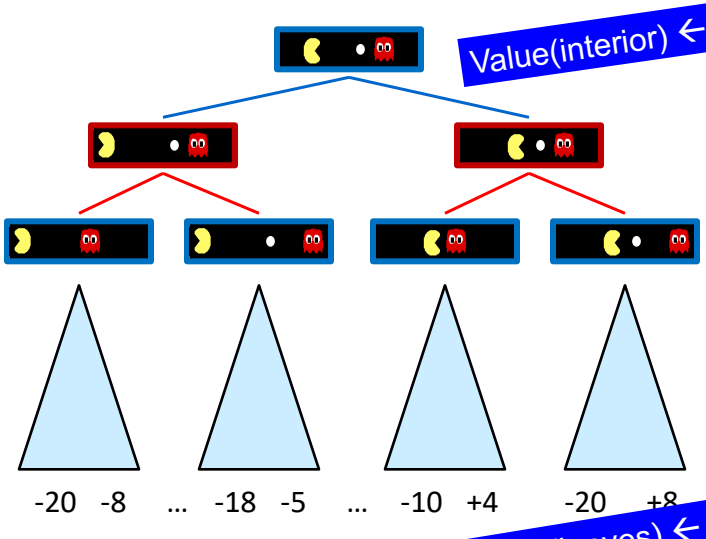


Terminal States:

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

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

Adversarial Game Trees



Value(interior) ← diffrnt

Value(leaves) ← same

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

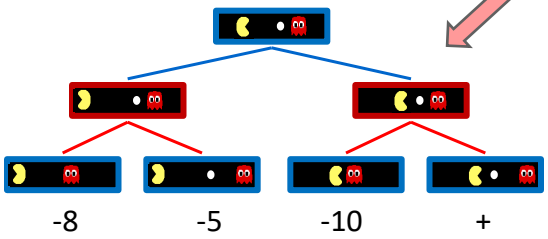
Minimax Values

States Under Agent's Control:

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

States Under Opponent's Control:

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



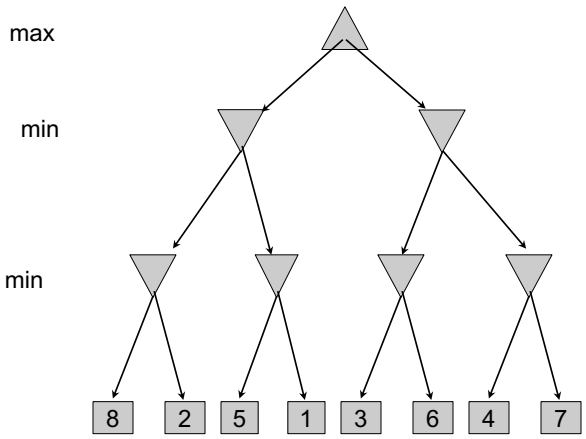
Terminal States:

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

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

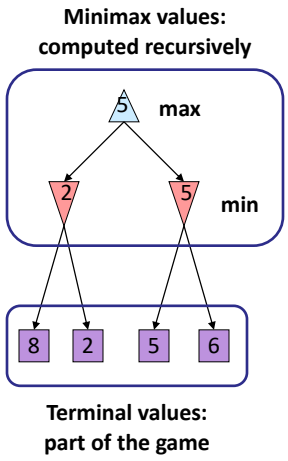
Three-Player

Eg playing against two ghosts?



Adversarial Search (Minimax)

- 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
  - Compute each node's minimax value: the best achievable utility against a rational (optimal) adversary



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

Minimax Implementation

Need ***Base case*** for recursion

```
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
```

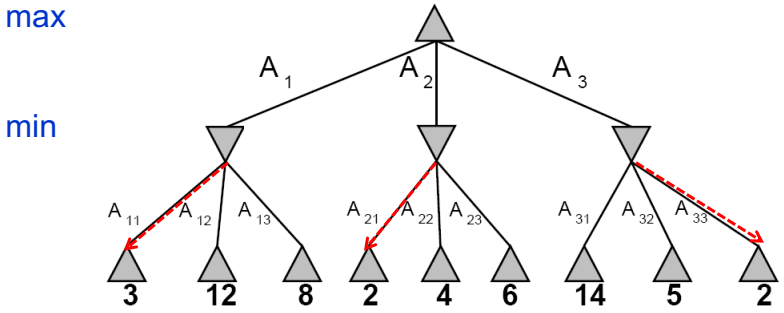
```
def min-value(state):
    if leaf?(state), return U(state)
    initialize v = +∞
    for each c in children(state)
        v = min(v, max-value(c))
    return v
```

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

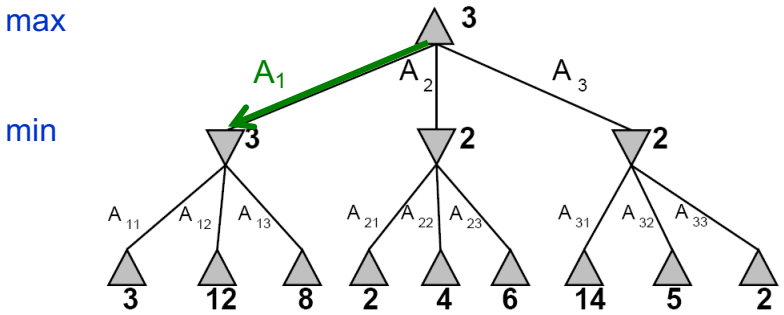
$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$

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

Concrete Minimax Example

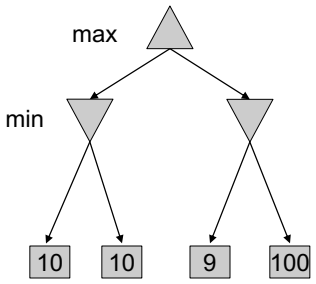


Minimax Example



Minimax Properties

- Optimal?
  - Yes, against perfect player (assuming finite depth)
  - Otherwise?
- Time complexity?
  - $O(b^m)$
- Space complexity?
  - O(bm)

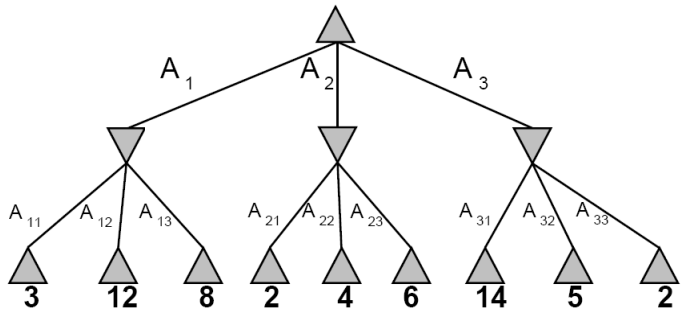


- For chess, b ~ 35, m ~ 100
  - Exact solution is completely infeasible >> | atoms |
  - **But**,… do we need to explore the whole tree?

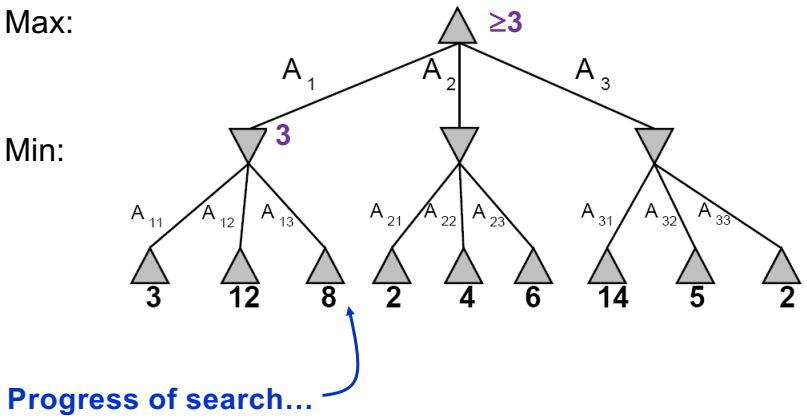
Do We Need to Evaluate Every Node?

Max:

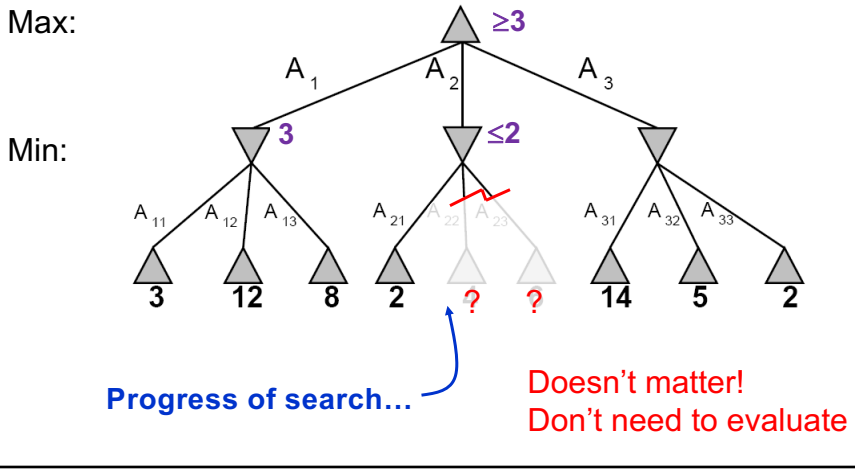
Min:



Do We Need to Evaluate Every Node?



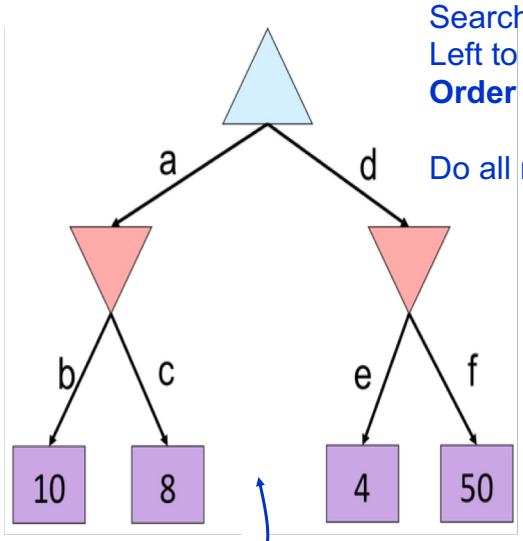
α-β Pruning Example



Alpha-Beta Quiz

Max:

Min:



Search depth-first
Left to right
Order is important

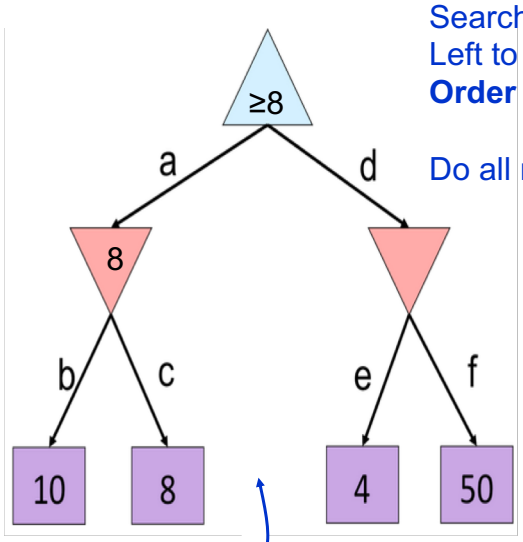
Do all nodes matter?

Progress of search…

Alpha-Beta Quiz

Max:

Min:



Search depth-first
Left to right
Order is important

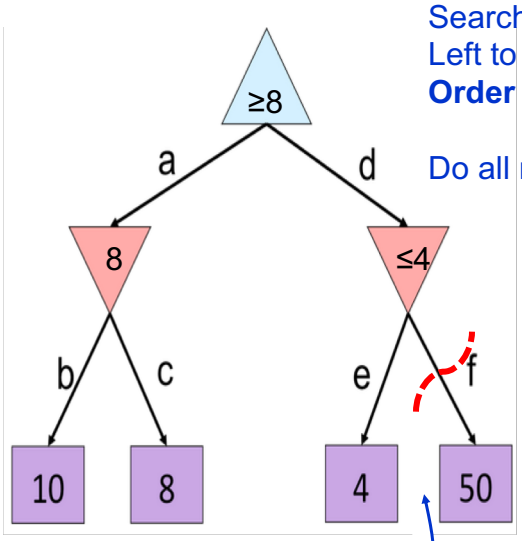
Do all nodes matter?

Progress of search…

Alpha-Beta Quiz

Max:

Min:



Search depth-first
Left to right
Order is important

Do all nodes matter?

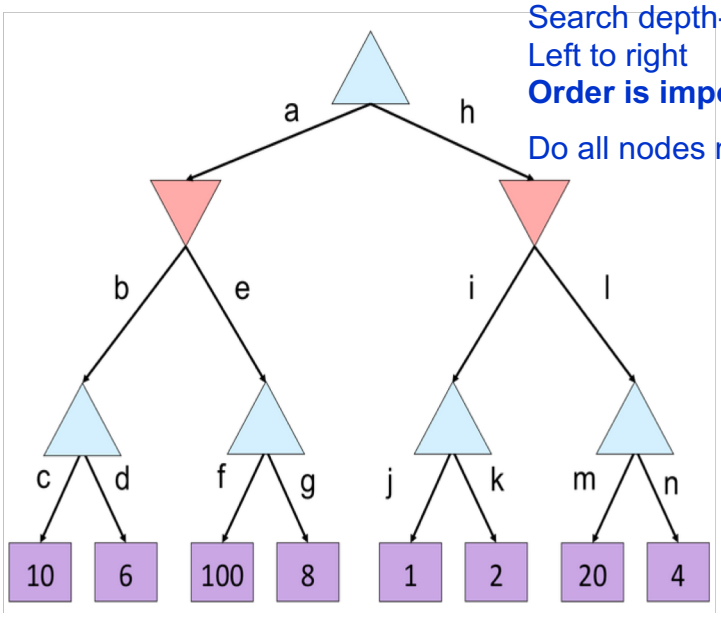
Slide adapted from Dan Klein & Pieter Abbeel - **Progress of search…**

Alpha-Beta Quiz 2

Max:

Min:

Max:



Search depth-first
Left to right
Order is important

Do all nodes matter?

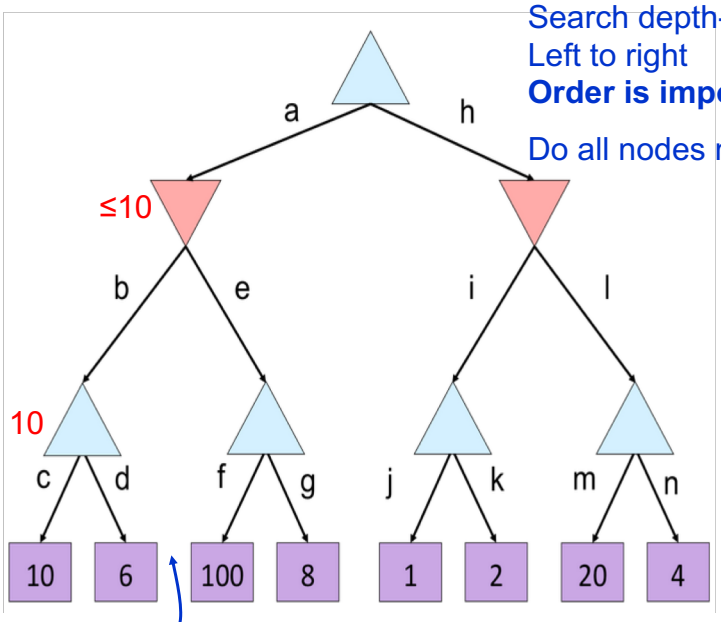
Slide adapted from Dan Klein & Pieter Abbeel - ai.berkeley.edu

Alpha-Beta Quiz 2

Max:

Min:

Max:



Search depth-first
Left to right
Order is important

Do all nodes matter?

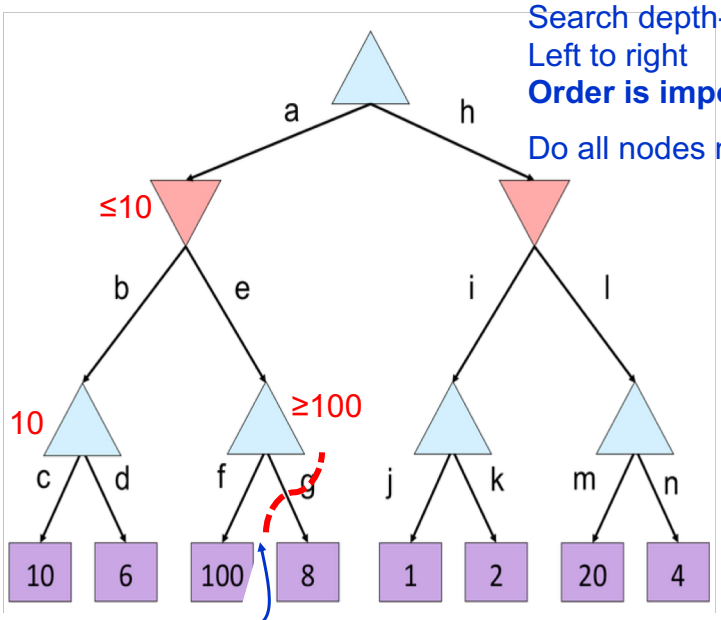
Slide adapted from Dan Klein & Pieter Abbeel - ai.berkeley.edu

Alpha-Beta Quiz 2

Max:

Min:

Max:

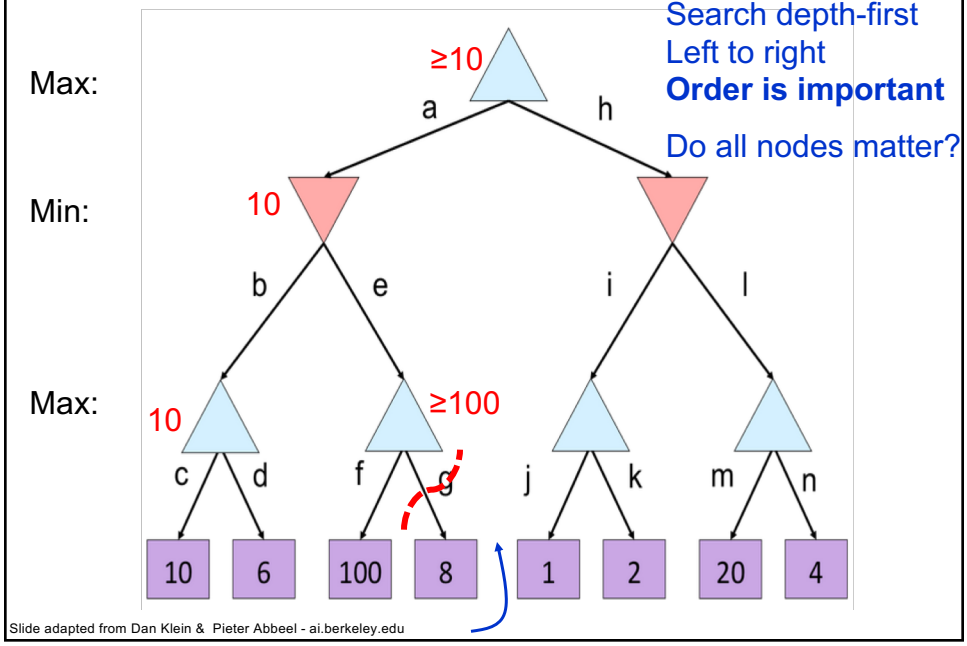


Search depth-first
Left to right
Order is important

Do all nodes matter?

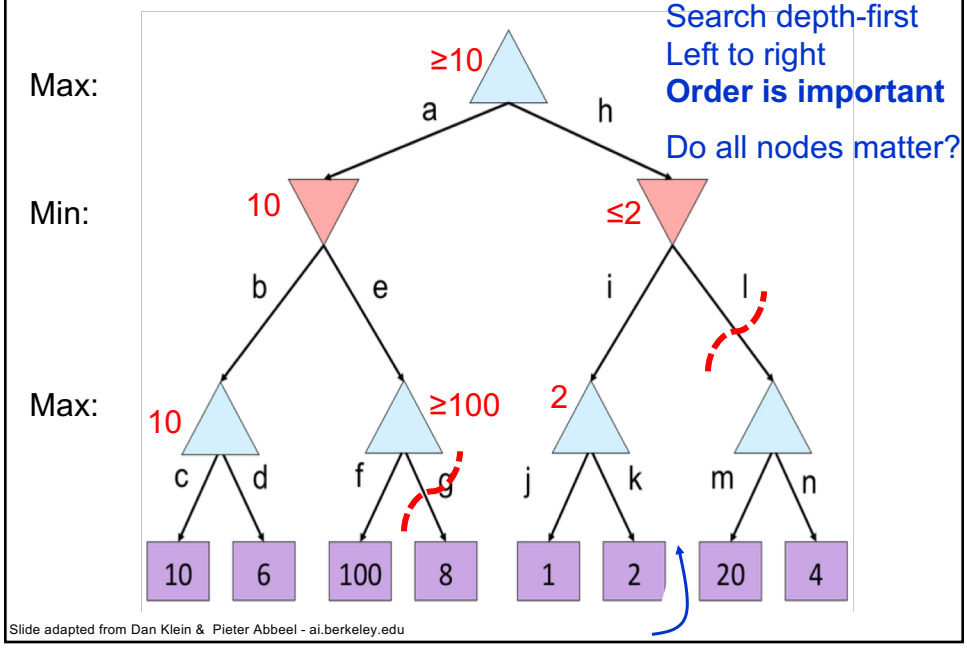
Slide adapted from Dan Klein & Pieter Abbeel - ai.berkeley.edu

Alpha-Beta Quiz 2



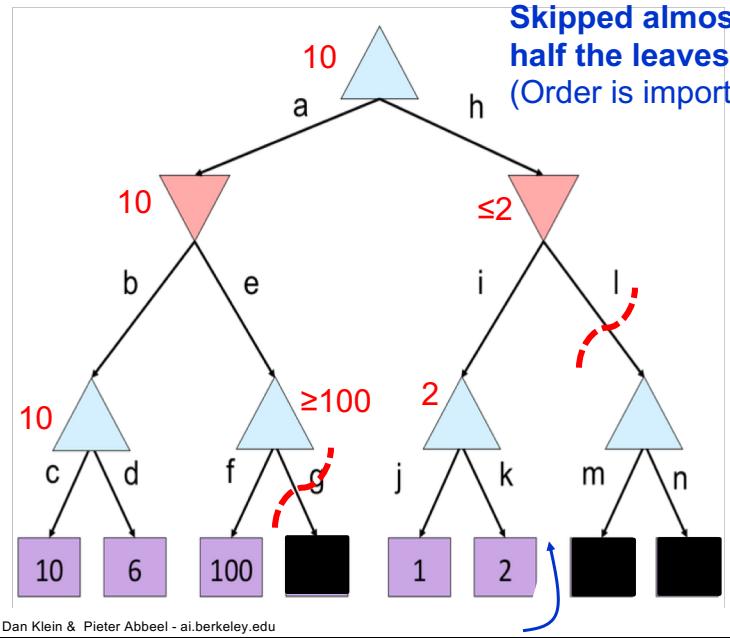
Slide adapted from Dan Klein & Pieter Abbeel - ai.berkeley.edu

Alpha-Beta Quiz 2



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

Alpha-Beta Quiz 2

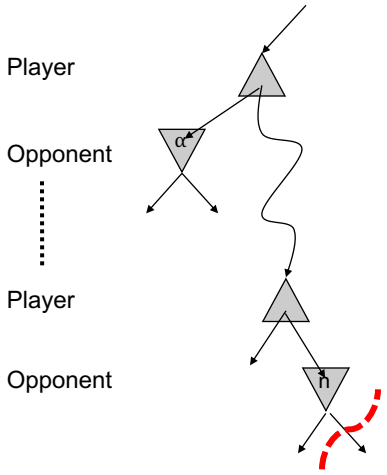


Skipped almost half the leaves
(Order is important)

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

α-β Pruning

- α is MAX's best choice on path to root
- If n becomes worse than α, MAX will avoid it, so can stop considering n's other children
- Define β similarly for MIN



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
```

At max node:
Prune if v≥β;
Update α

```
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
```

At min node:
Prune if α≤v;
Update β

Min-Max Implementation

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

    return v
```

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

    return v
```

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

Alpha-Beta Implementation

α: MAX's best option on path to root
β: MIN's best option on path to root

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

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

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

Alpha-Beta Implementation

α: MAX's best option on path to root
β: MIN's best option on path to root

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

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

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

Alpha-Beta Pruning Demo

http://inst.eecs.berkeley.edu/~cs61b/fa14/ta-materials/apps/ab_tree_practice/

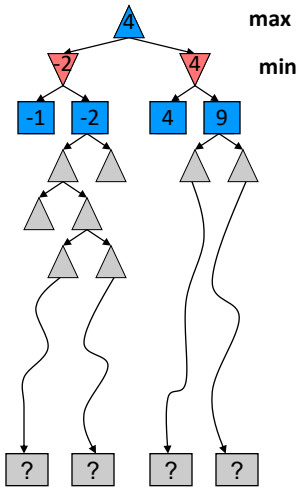
47

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 correct ***bounds***
- Good child ordering improves effectiveness of pruning
- With "perfect ordering":
  - Time complexity drops to $O(b^{m/2})$
  - ***Doubles*** solvable depth!
  - (But complete search of complex games, e.g. chess, is still hopeless…

Resource Limits

- Problem: In realistic games, one cannot search all way 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 3 min/move, can explore 1M nodes / sec
  - So can check 200M nodes per move
  - α-β reaches about depth 10 → decent chess program
- Guarantee of optimal play is gone
- More plies makes a BIG difference



Depth Matters

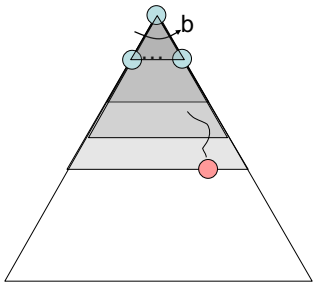
- Evaluation functions are always imperfect
- The deeper in the tree the evaluation function is buried, the less the quality of the evaluation function matters
- Good example of the tradeoff between complexity of ***features*** and complexity of ***computation***

Iterative Deepening

Iterative deepening uses DFS as a subroutine:

1. Do a DFS which only searches for paths of length 1 or less. (DFS gives up on any path of length 2)
2. If "1" ***fails***, do a DFS which only searches paths of length 2 or less.
3. If "2" ***fails***, do a DFS which only searches paths of length 3 or less.

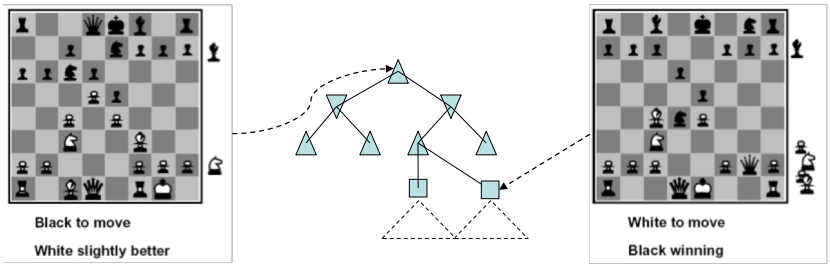
….and so on.



Can one adapt to games to make ***anytime algorithm ?***

Heuristic Evaluation Function

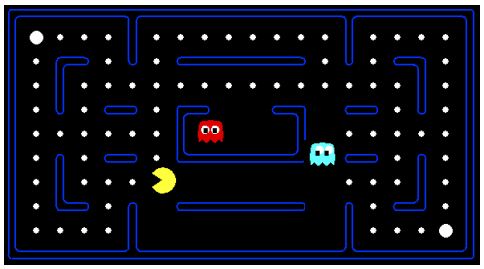
- Function which scores non-terminals



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

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

Evaluation for Pacman

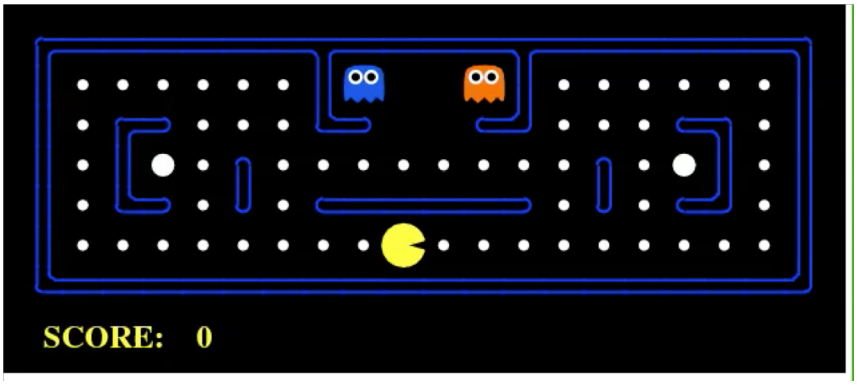


What features would be good for Pacman?

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

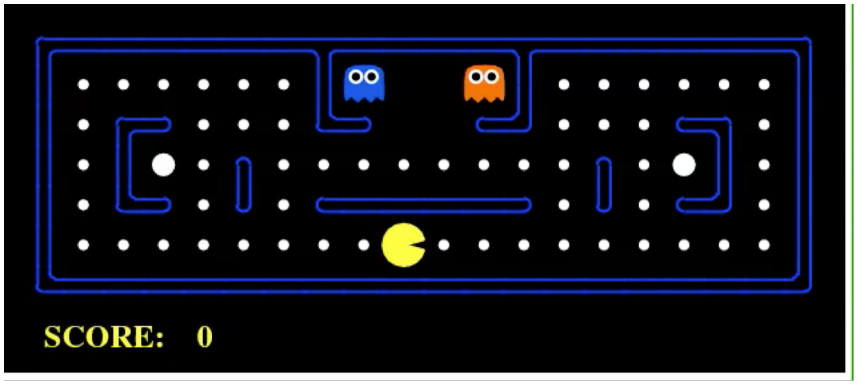
Which algorithm?

α-β, depth 4, simple eval fun



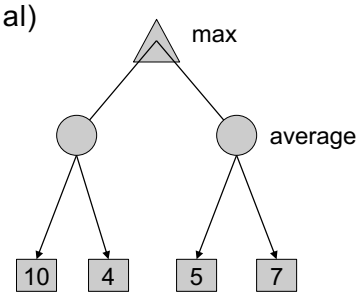
Which algorithm?

α-β, depth 4, better eval fun



Stochastic Single-Player

- What if we don't know what the result of an action will be? E.g.,
  - In solitaire, shuffle is unknown
  - In minesweeper, mine locations
  - Ghost are stupid (not super-rational)
- Can do ***expectimax*** search
  - Chance nodes, like actions except the ***environment*** controls the action chosen
  - Max nodes as before
  - Chance nodes take average (expectation) of value of children



Maximum Expected Utility

- Why should we average utilities? Why not take max?
- 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…

Uncertainty Everywhere

- Not just for games of chance!
  - I'm sick: will I sneeze this minute?
  - Email contains "FREE!": is it spam?
  - Tooth hurts: do I have a 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: Probabilities

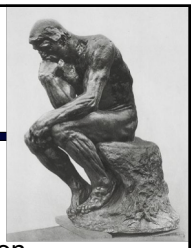
- A random variable represents an event whose outcome is unknown
- A probability distribution is an assignment of weights to outcomes

- 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)

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

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

$$= 3.5$$

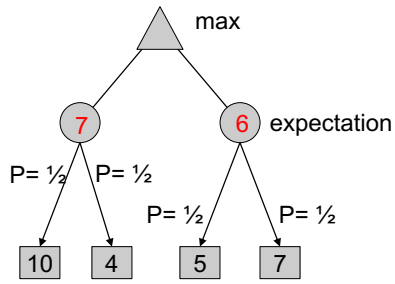
Utilities

- 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

- Max nodes as before
- Chance nodes are like actions
  except the ***environment*** controls the action chosen
- Chance nodes take average (expectation) of value of children

- We'll soon generalize this to a….
  Markov Decision Process (MDP)



Which Algorithm?

Minimax

No point in trying

3 ply look ahead, ghosts move randomly

Which Algorithm?

Expectimax

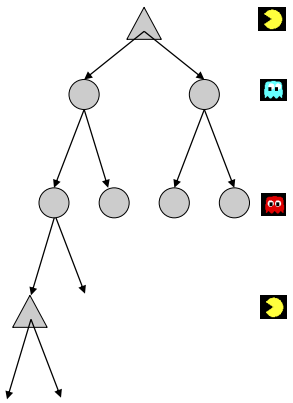
Wins some of the time

3 ply look ahead, ghosts move randomly

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)… or more complex
- We have a node for every outcome out of our control: opponent or environment



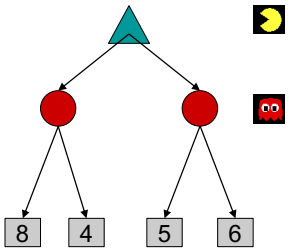
For now, assume ∀states we magically have a distribution to assign probabilities to enemy-actions / environment outcomes

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

- Note: 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?

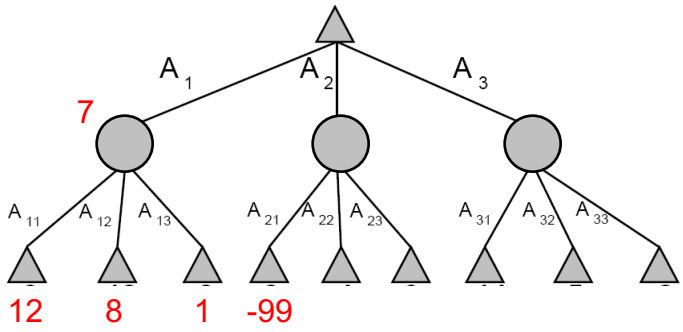
Expectimax for Pacman

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

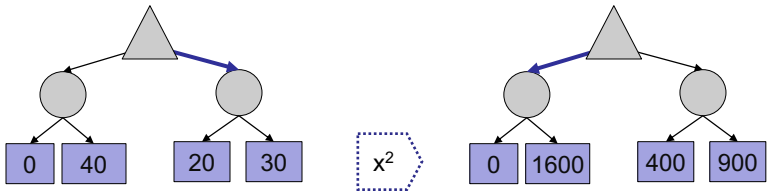
ExpectiMax Pruning?



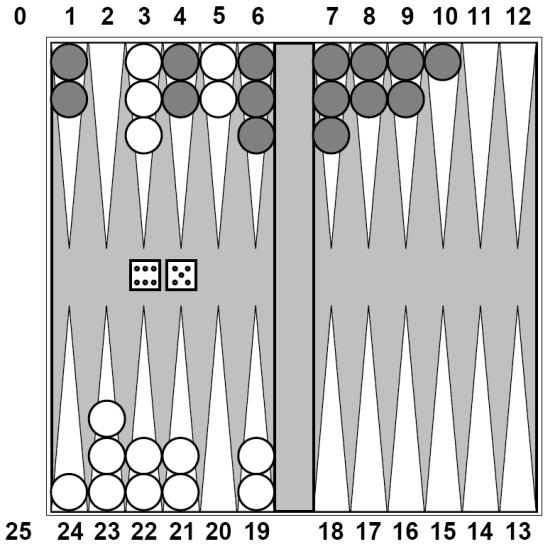
- Not easy
  - exact: need ***bounds*** on possible values
  - approximate: sample high-probability branches

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

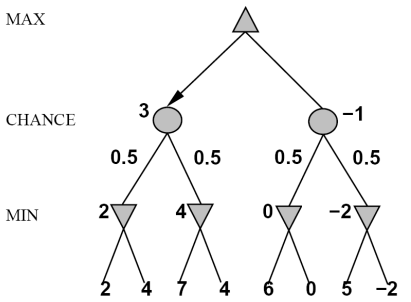


What about..



Mixed Layer Types

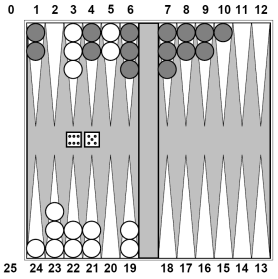
- E.g. Backgammon
- Expecti-Mini-Max
  - Environment is an extra player that moves after each agent
  - Chance nodes take expectations, otherwise like minimax




```
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 x $(21 \times 20)^3$ = 1.2 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…

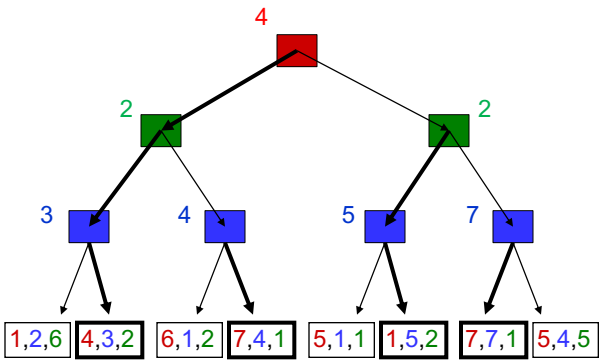


- TDGammon used depth-2 search + very good eval function
  - Learned via NN & reinforcement learning
  - World-champion level play (1992, Gerald Tesauro)

Multi-Player 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…



In this example… three agents