

CS 188: Artificial Intelligence

Spring 2011

Lecture 7: Minimax and Alpha-Beta Search

2/9/2011

Pieter Abbeel – UC Berkeley
Many slides adapted from Dan Klein

Announcements

- W1 out and due Monday 4:59pm
- P2 out and due next week Friday 4:59pm

Overview

- **Deterministic zero-sum games**
 - Minimax
 - Limited depth and evaluation functions
 - Alpha-Beta pruning
- **Stochastic games**
 - Expectimax
- **Non-zero-sum games**

3

Game Playing State-of-the-Art

- **Checkers:** Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used 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!
- **Chess:** Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997. 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.
- **Othello:** Human champions refuse to compete against computers, which are too good.
- **Go:** Human champions are beginning to be challenged by machines, though the best humans still beat the best machines. In go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves, along with aggressive pruning.
- **Pacman:** unknown

4

GamesCrafters

The GamesCrafters research and development group was formed in 2001 as a "hacking hole" to gather and engage top undergraduates as they explore the little area of computational game theory. At the core of the project is GAMESCRAFT, an open-source AI architecture developed for solving, playing, and analyzing two-person abstract strategy games (e.g., Tic-Tac-Toe or Chess). Given the description of a game as input, our system generates a command-line interface and GUI graphical application that will solve it (in the strong sense), and then play it perfectly. Programmers can easily prototype a new game with multiple rule variants, learn the strategy via colored-coded board moves (e.g., go, green, left, capture, yellow, lose, stop, no), and perform extended analysis.

The group is accessible to undergraduates at all levels. Those not yet ready to dive into code can create graphics, find bugs, or research the history of games for our website. Programmers can easily prototype a new game with multiple rule variants, design a fun interface, and perform extended analysis. Advanced students are encouraged to leave with the software core, and optimize the solvers, databases, hash functions, networking, user experience, etc.

Since this is not a class, but directed group study, students can re-register as often as they like, most stay for two or three semesters. This allows for a real community to be formed, with veterans providing continuity and mentoring as project leads, as well as allowing for more ambitious multi-term projects. Our alumni have told us how valuable this experience has been for them, providing them with a nurturing environment to mature as researchers, developers, and leaders.

Over the past six years, over two hundred undergraduates have implemented more than sixty-five games and several advanced software engineering projects. Our future research direction is "hunting big game", i.e., implementing, solving, and analyzing large games whose perfect strategy is yet unknown. This semester (Fall 2005), we're meeting in 606 Soda Hall, Mondays from 5-6PM. It is a "Directed Group Study" course worth two units led by Dr. Dan Garcia.

<http://gamescrafters.berkeley.edu/>

Dan Garcia.

5

Game Playing

- Many different kinds of games!
- Axes:
 - Deterministic or stochastic?
 - One, two, or more players?
 - Perfect information (can you see the state)?
- Want algorithms for calculating a **strategy** (**policy**) which recommends a move in each state

7

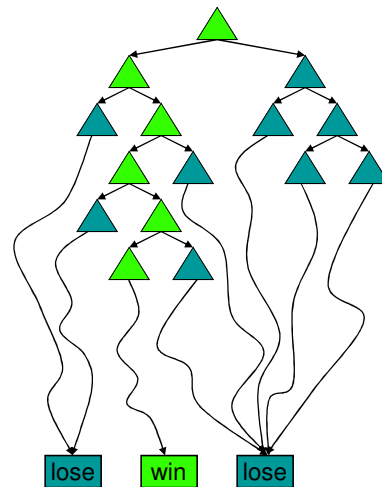
Deterministic Games

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

8

Deterministic Single-Player?

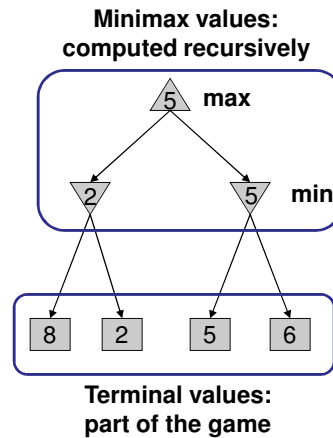
- Deterministic, single player, perfect information:
 - Know the rules
 - Know what actions do
 - Know when you win
 - E.g. Freecell, 8-Puzzle, Rubik's cube
- ... it's just search!
- Slight reinterpretation:
 - Each node stores a **value**: the best outcome it can reach
 - This is the maximal outcome of its children (the **max value**)
 - Note that we don't have path sums as before (**utilities at end**)
- After search, can pick move that leads to best node
- Often: not enough time to search till bottom before taking the next action



9

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



Terminology: ply = all players making a move, game to the right = 1 ply

10

Computing Minimax Values

- **Two recursive functions:**
 - **max-value** maxes the values of successors
 - **min-value** mins the values of successors

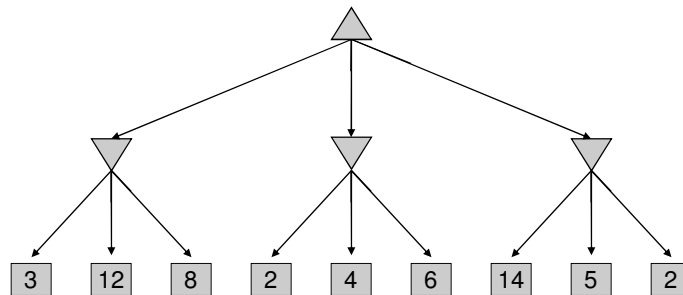
def value(state):

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 max = $-\infty$
For each successor of state:
 Compute value(successor)
 Update max accordingly
Return max

Minimax Example



12

Minimax Properties

- Optimal against a perfect player. Otherwise?

- Time complexity?

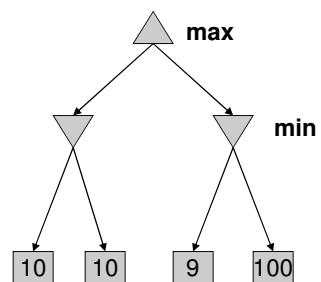
- $O(b^m)$

- Space complexity?

- $O(bm)$

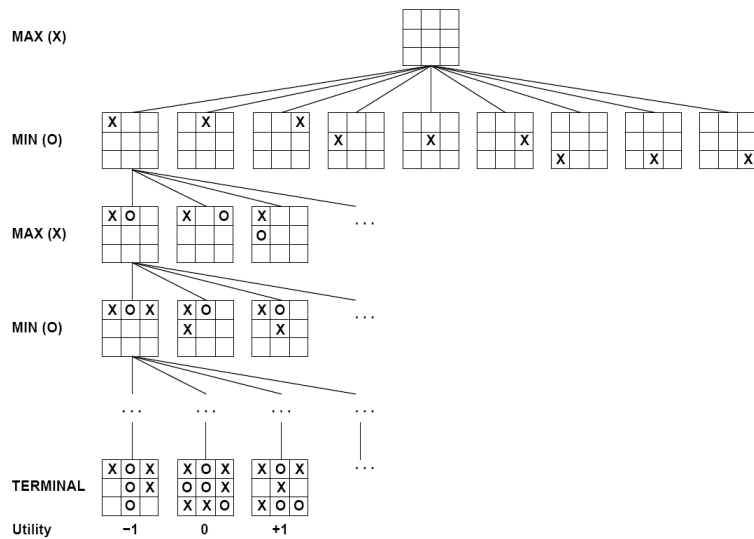
- For chess, $b \approx 35$, $m \approx 100$

- Exact solution is completely infeasible
- But, do we need to explore the whole tree?



15

Tic-tac-toe Game Tree



16

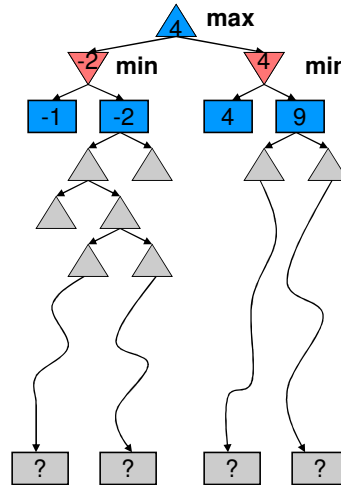
Speeding Up Game Tree Search

- Evaluation functions for non-terminal states
- Pruning: not search parts of the tree
 - Alpha-Beta pruning does so without losing accuracy, $O(b^d) \rightarrow O(b^{d/2})$

17

Resource Limits

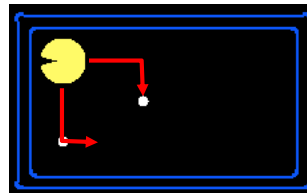
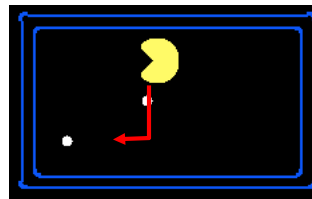
- Cannot search to leaves
- Depth-limited search
 - Instead, search a limited depth of tree
 - Replace terminal utilities with an eval function for non-terminal positions
- Guarantee of optimal play is gone



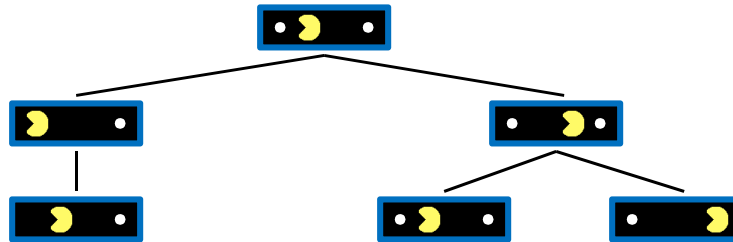
19

Why Pacman Can Starve

- He knows his score will go up by eating the dot now
- He knows his score will go up just as much by eating the dot later on
- There are no point-scoring opportunities after eating the dot
- Therefore, waiting seems just as good as eating



Why Pacman Starves



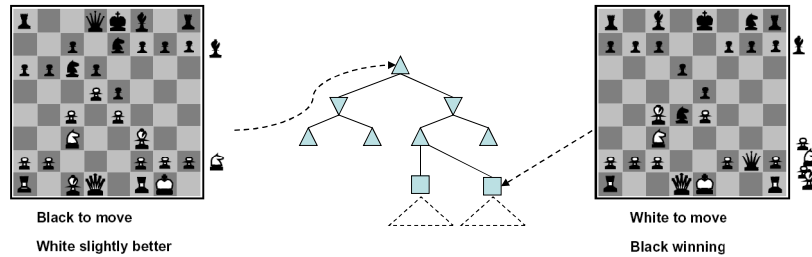
- He knows his score will go up by eating the dot now (west, east)
- He knows his score will go up just as much by eating the dot later (east, west)
- There are no point-scoring opportunities after eating the dot (within the horizon, two here)
- Therefore, waiting seems just as good as eating: he may go east, then back west in the next round of replanning!

Ghosts

- Game tree with 1 and multiple ghosts?

Evaluation Functions

- Function which scores non-terminals



- Ideal function: returns the utility of the position
- In practice: typically weighted linear sum of features:

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

- e.g. $f_1(s) = (\text{num white queens} - \text{num black queens})$, etc.

23

Evaluation Functions

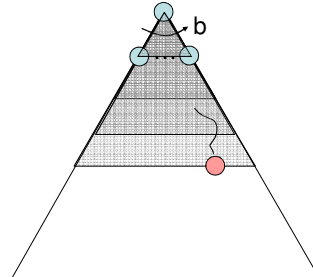
- With depth-limited search
 - Partial plan is returned
 - Only first move of partial plan is executed
 - When again maximizer's turn, run a depth-limited search again and repeat
- How deep to search?

25

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" failed, do a DFS which only searches paths of length 2 or less.
3. If "2" failed, do a DFS which only searches paths of length 3 or less.
....and so on.



Why do we want to do this for multiplayer games?

Note: wrongness of eval functions matters less and less the deeper the search goes

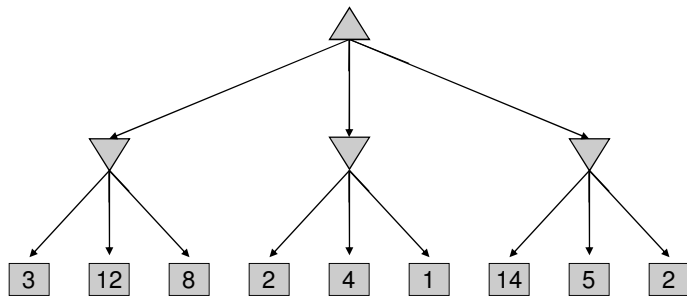
26

Speeding Up Game Tree Search

- Evaluation functions for non-terminal states
- Pruning: not search parts of the tree
 - Alpha-Beta pruning does so without losing accuracy, $O(b^d) \rightarrow O(b^{d/2})$

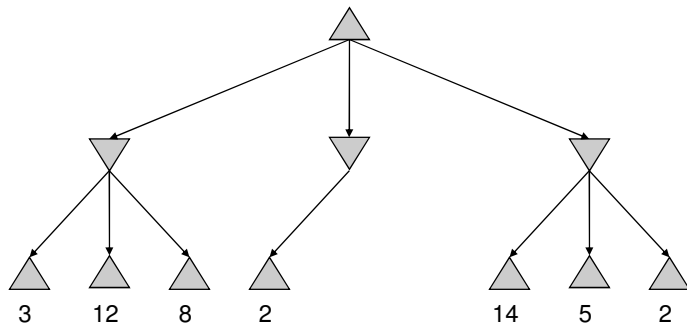
27

Minimax Example



28

Pruning

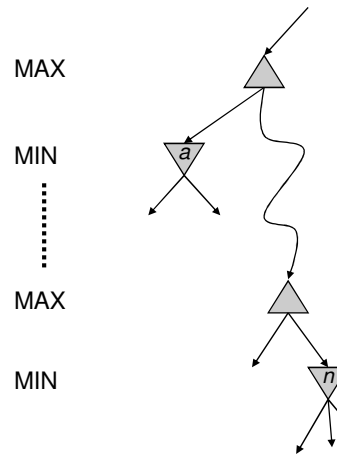


29

Alpha-Beta Pruning

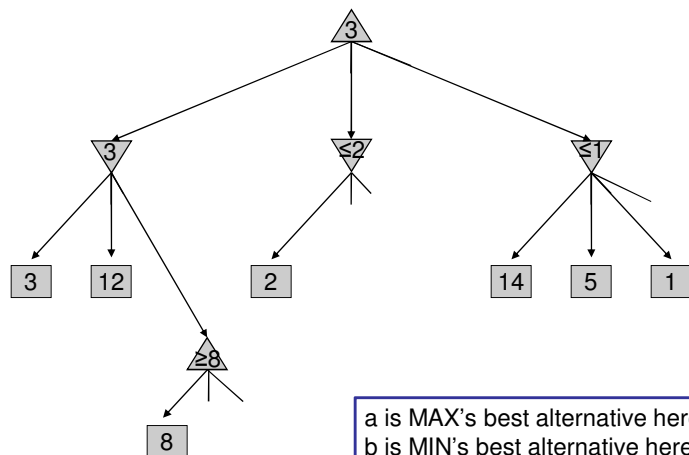
- General configuration

- We're computing the MIN-VALUE at n
- We're looping over n 's children
- n 's value estimate is dropping
- a is the best value that MAX can get at any choice point along the current path
- If n becomes worse than a , MAX will avoid it, so can stop considering n 's other children
- Define b similarly for MIN



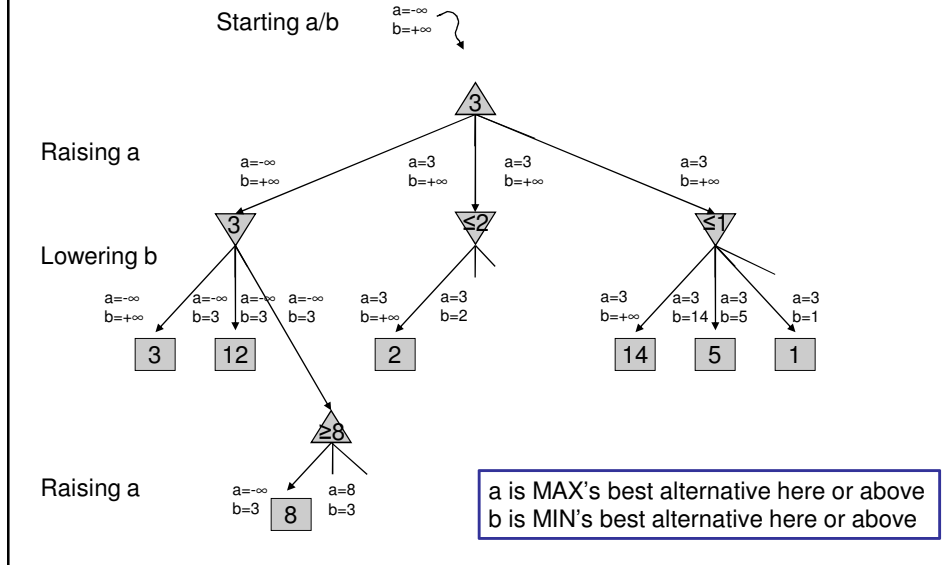
31

Alpha-Beta Pruning Example



a is MAX's best alternative here or above
 b is MIN's best alternative here or above

Alpha-Beta Pruning Example



Alpha-Beta Pseudocode

```

function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow -\infty$ 
  for  $a, s$  in SUCCESSORS(state) do  $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s))$ 
  return  $v$ 

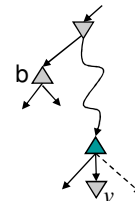
```

```

function MAX-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
  inputs: state, current state in game
          $\alpha$ , the value of the best alternative for MAX along the path to state
          $\beta$ , the value of the best alternative for MIN along the path to state

  if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow -\infty$ 
  for  $a, s$  in SUCCESSORS(state) do
     $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s, \alpha, \beta))$ 
    if  $v \geq \beta$  then return  $v$ 
     $\alpha \leftarrow \text{MAX}(\alpha, v)$ 
  return  $v$ 

```



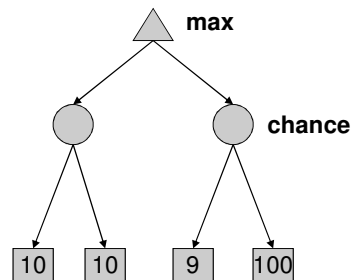
Alpha-Beta Pruning Properties

- This pruning has **no effect** on final result at the root
- Values of intermediate nodes might be wrong!
- 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)

35

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** to maximize average score
 - 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**



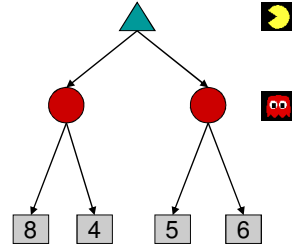
36

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)
```

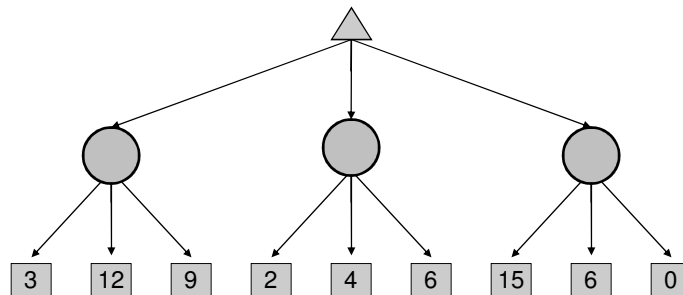
```
def maxValue(s)
  values = [value(s') for s' in successors(s)]
  return max(values)
```

```
def expValue(s)
  values = [value(s') for s' in successors(s)]
  weights = [probability(s, s') for s' in successors(s)]
  return expectation(values, weights)
```



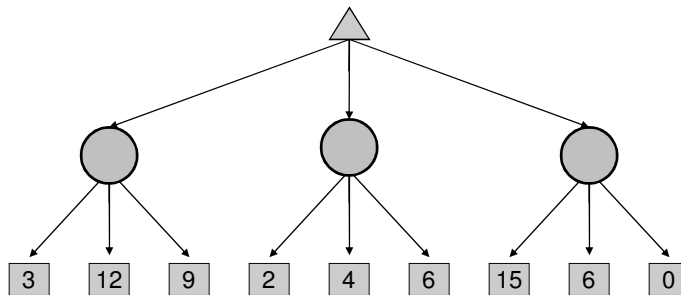
37

Expectimax Quantities



38

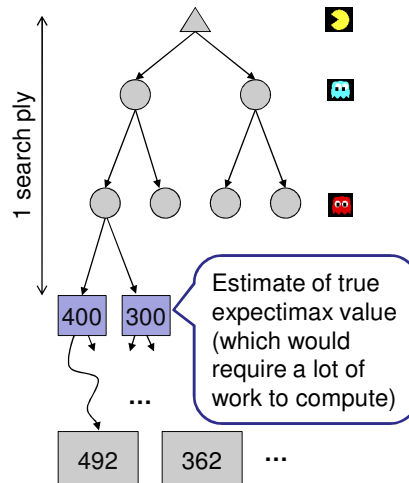
Expectimax Pruning?



39

Expectimax Search

- **Chance nodes**
 - Chance nodes are like min nodes, except the outcome is uncertain
 - Calculate **expected utilities**
 - Chance nodes average successor values (weighted)
- Each chance node has a **probability distribution over its outcomes (called a model)**
 - For now, assume we're given the model
- **Utilities for terminal states**
 - Static evaluation functions give us limited-depth search



Expectimax for Pacman

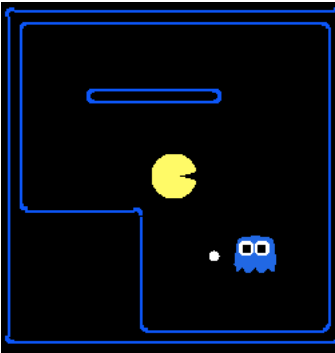
- Notice that we've gotten away from thinking that the ghosts are trying to minimize pacman's score
- Instead, they are now a part of the environment
- Pacman has a belief (distribution) over how they will act
- Quiz: Can we see minimax as a special case of expectimax?
- Quiz: what would pacman's computation look like if we assumed that the ghosts were doing 1-ply minimax and taking the result 80% of the time, otherwise moving randomly?
- If you take this further, you end up calculating belief distributions over your opponents' belief distributions over your belief distributions, etc...
 - Can get unmanageable very quickly!

41

Expectimax for Pacman

Results from playing 5 games

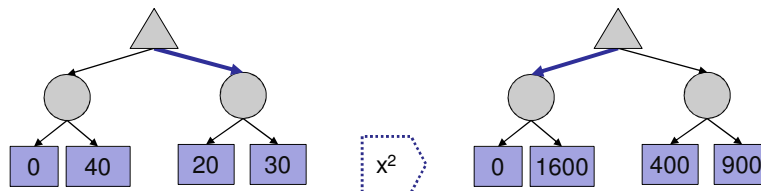
	Minimizing Ghost	Random Ghost
Minimax Pacman	Won 5/5 Avg. Score: 493	Won 5/5 Avg. Score: 483
Expectimax Pacman	Won 1/5 Avg. Score: -303	Won 5/5 Avg. Score: 503



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

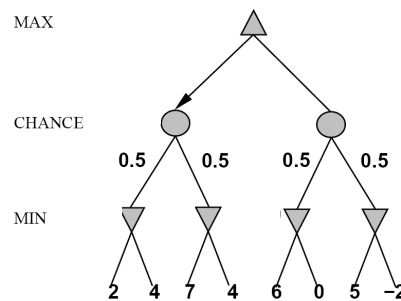
Expectimax Utilities

- For minimax, 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 expectimax, we need *magnitudes* to be meaningful



Stochastic Two-Player

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

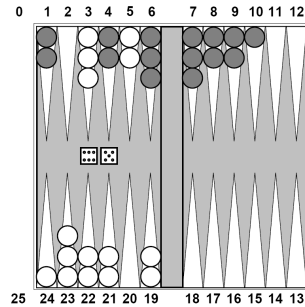


```

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 \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...
- TDGammon uses depth-2 search + very good evaluation function + reinforcement learning: world-champion level play
- 1st AI world champion in any game!



Non-Zero-Sum Utilities

- Similar to minimax:
 - Terminals have utility tuples
 - Node values are also utility tuples
 - Each player maximizes its own utility and propagate (or back up) nodes from children
 - Can give rise to cooperation and competition dynamically...

