Expectimax

Lirong Xia

Rensselaer

Project 2

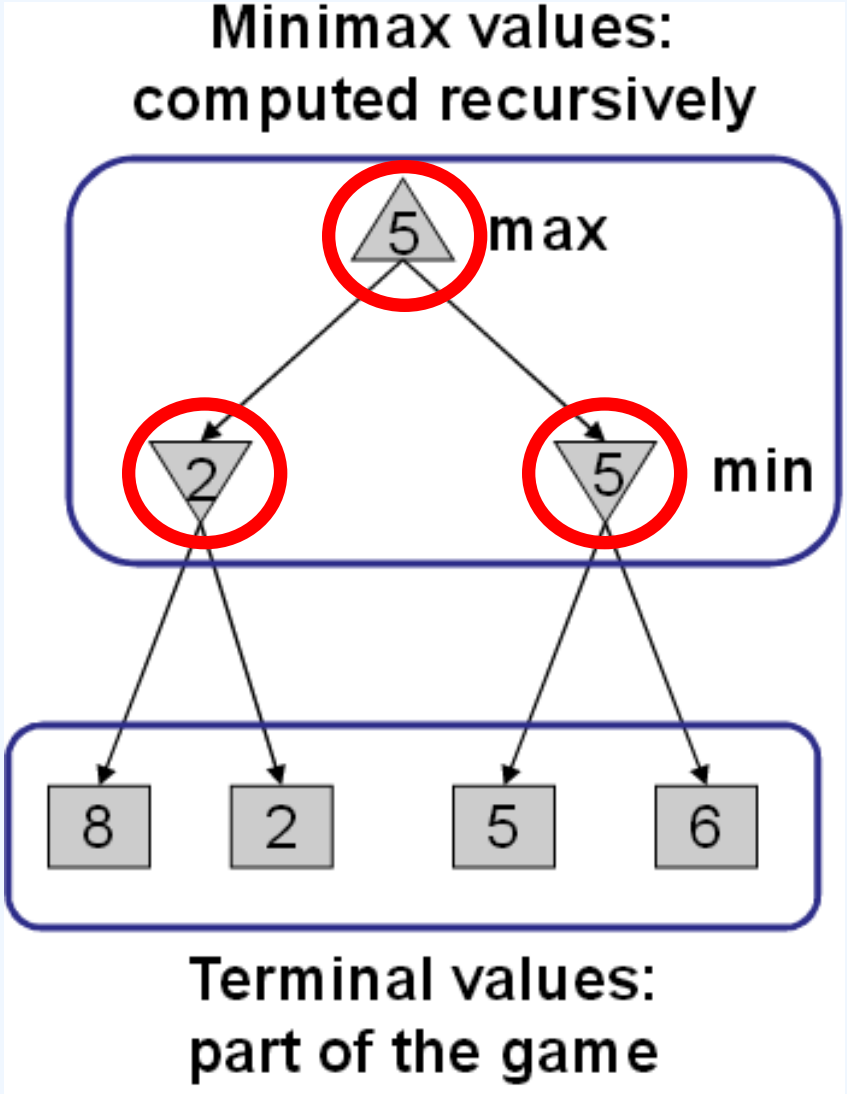
- MAX player: Pacman
- Question 1-3: Multiple MIN players: ghosts
- Extend classical minimax search and alpha-beta pruning to the case of multiple MIN players
- Important: A single search ply is considered to be one Pacman move and all the ghosts' responses
  - so depth 2 search will involve Pacman and each ghost moving two times.
- Question 4-5: Random ghosts

Last class

- Minimax search
  - with limited depth
  - evaluation function
- Alpha-beta pruning

Adversarial Games

- Deterministic, zero-sum games:
  - Tic-tac-toe, chess, checkers
  - The MAX player maximizes result
  - The MIN player minimizes result
- Minimax search:
  - A search tree
  - Players alternate turns
  - Each node has a minimax value: best achievable utility against a rational adversary



Computing Minimax Values

- This is DFS

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

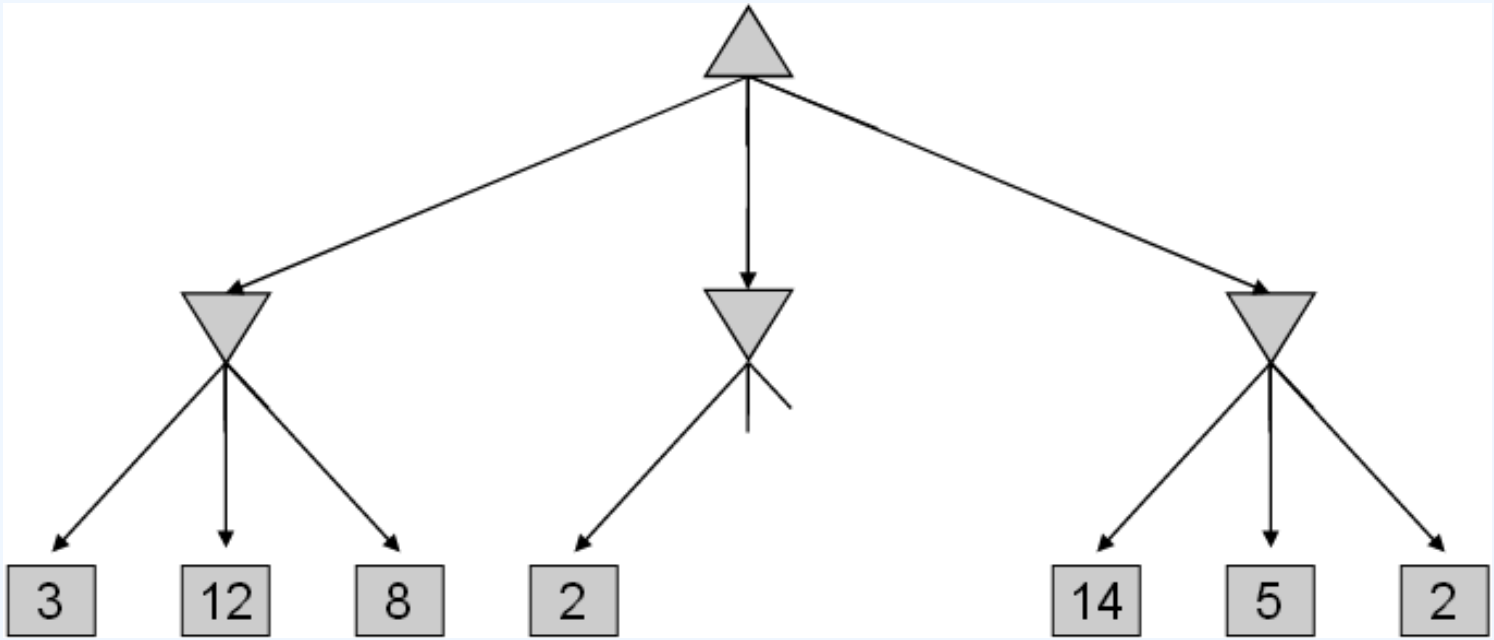
- Def value (state):
  If the state is a terminal state: return the state's utility
  If the next agent is MAX: return max-value(state)
  If the next agent is MIN: return min-value(state)

- Def max-value(state):
  Initialize max = -∞
  For each successor of state:
    Compute value(successor)
    Update max accordingly
  return max
- Def min-value(state): similar to max-value

Minimax with limited depth

- Suppose you are the MAX player
- Given a depth d and current state
- Compute value(state, d) that reaches depth d
  - at depth d, use a evaluation function to estimate the value if it is non-terminal

Pruning in Minimax Search



Alpha-beta pruning

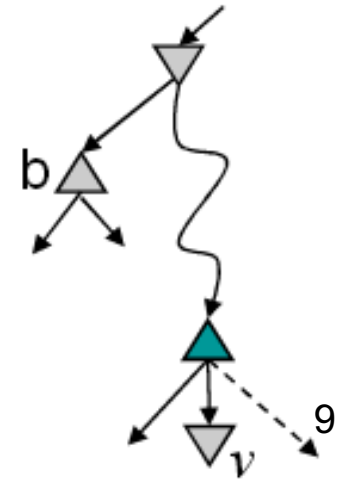
- Pruning = cutting off parts of the search tree (because you realize you don't need to look at them)
  - When we considered A* we also pruned large parts of the search tree
- Maintain α = value of the best option for the MAX player along the path so far
- β = value of the best option for the MIN player along the path so far
- Initialized to be α = -∞ and β = +∞
- Maintain and update α and β for each node
  - α is updated at MAX player's nodes
  - β is updated at MIN player's nodes

Alpha-Beta Pseudocode

```
function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do v ← MAX(v, MIN-VALUE(s))
    return v
```

```
function MAX-VALUE(state, α, β) returns a utility value
    inputs: state, current state in game
            α, the value of the best alternative for MAX along the path to state
            β, the value of the best alternative for MIN along the path to state

    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
```



Today's schedule

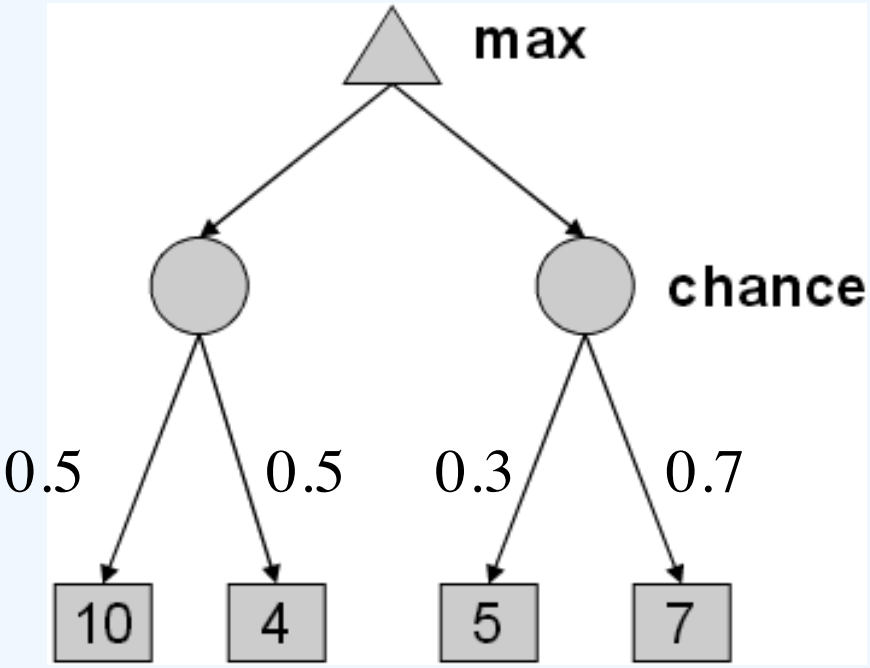
- Basic probability
- Expectimax search

Going beyond the MIN node

- In minimax search we (MAX) assume that the opponents (MIN players) act optimally
- What if they are not optimal?
  - lack of intelligence
  - limited information
  - limited computational power
- Can we take advantage of non-optimal opponents?
  - why do we want to do this?
  - you are playing chess with your roommate as if he/she is Kasparov

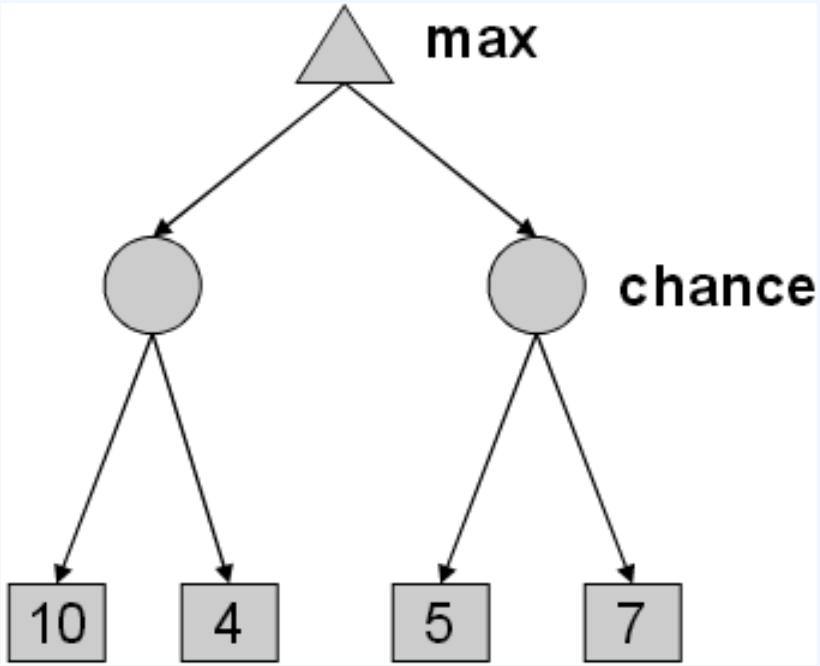
Modeling a non-optimal opponent

- Depends on your knowledge
- Model your belief about his/he action as a probability distribution



Expectimax Search Trees

- Expectimax search
  - Max nodes (we) as in minimax search
  - Chance nodes
    - Need to compute chance node values as expected utilities
- Later, we'll learn how to formalize the underlying problem as a Markov decision Process



Maximum Expected utility

- Principle of maximum expected utility
  - an agent should choose the action that maximizes its expected utility, given its knowledge
  - in our case, the MAX player should choose a chance node with the maximum expected utility
- General principle for decision making
- Often taken as the definition of rationality
- We'll see this idea over and over in this course!

Reminder: Probabilities

- A random variable represents an event whose outcome is unknown
- A probability distribution is an assignment of weights to outcomes
  - weights sum up to 1
- 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) = 0.50, p(T=heavy) = 0.25,
- 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

Reminder: Expectations

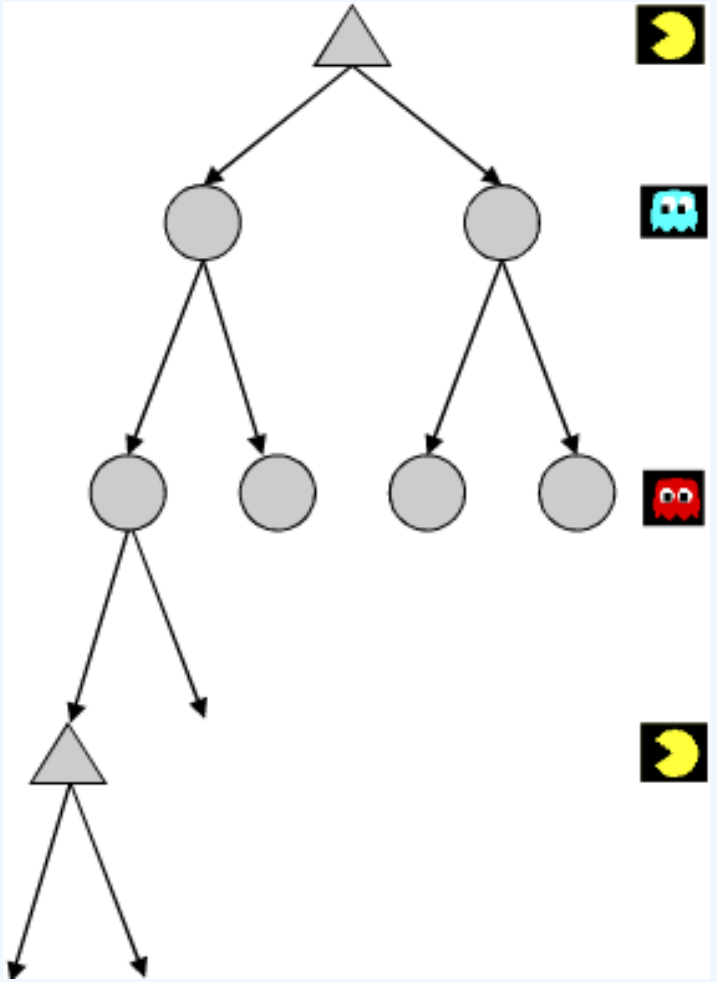
- We can define function f(X) or a random variable X
- The expected value of a function is its average value, weighted by the probability distribution over inputs
- Example: how long to get to the airport?
  - Length of driving time as a function of traffic:
    L(none) = 20, L(light) = 30, L(heavy) = 60
  - What is my expected driving time?
    - Notation: E[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?
  - Utilities summarize the agent's goals
  - Evaluation function
    - You will be asked to design evaluation functions in Project 2

Expectimax Search

- In expectimax search, we have a probabilistic model of how the opponent (or environment) will behave in any state
  - could be simple: uniform distribution
  - could be sophisticated and require a great deal of computation
  - We have a chance node for every situation out of our control: opponent or environment
- For now, assume for any state we magically have a distribution to assign probabilities to opponent actions / environment outcomes



Having a probabilistic belief about an agent's action does not mean that agent is flipping any coins!

Expectimax Pseudocode

- Def value(s):

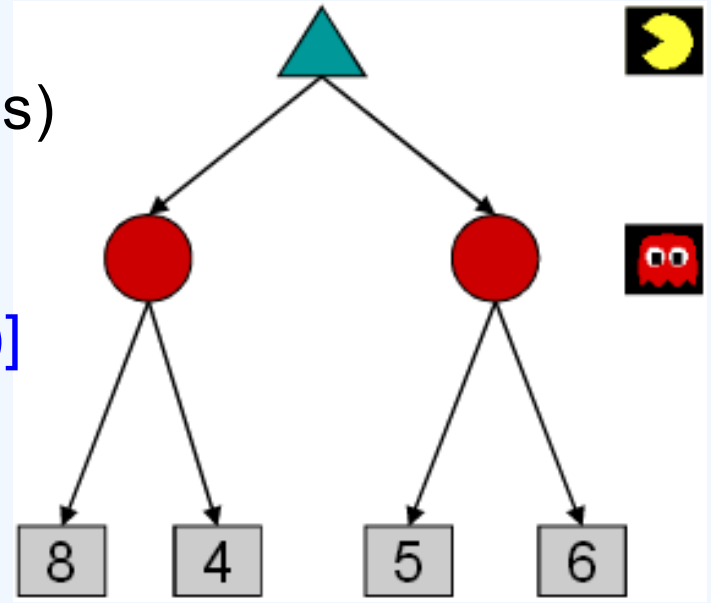
  If s is a max node return maxValue(s)
  If s is a chance node return expValue(s)
  If s is a terminal node return evaluations(s)

- Def maxValue(s):

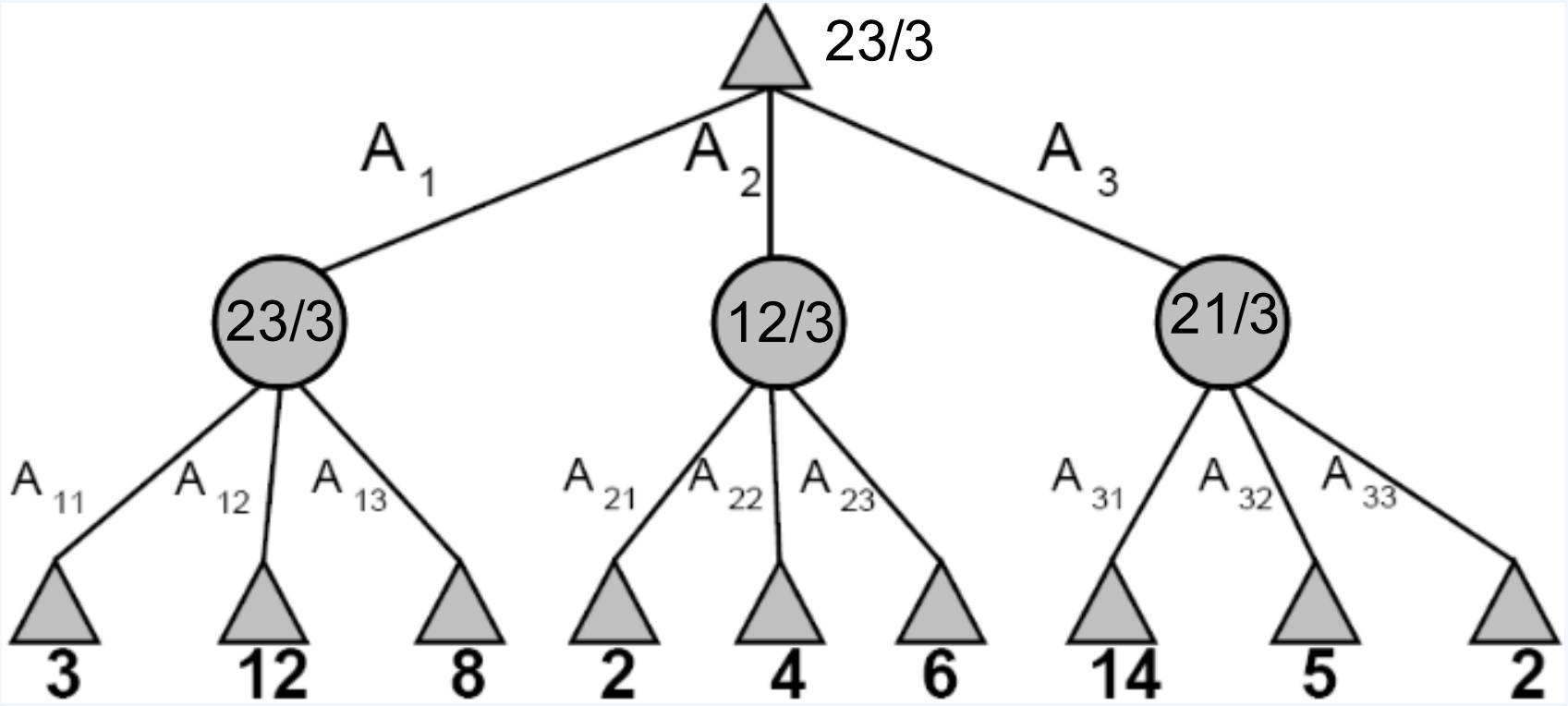
  values = [value(s') for s' in successors(s)]
  return max(values)

- 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 Example



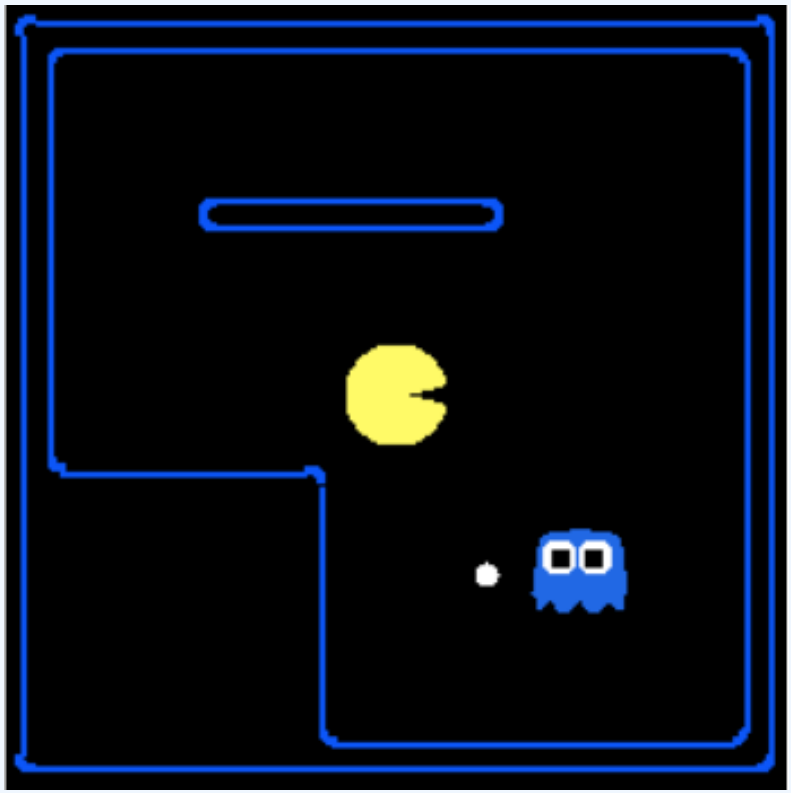
Expectimax for Pacman

- 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?
- Food for thought: what would pacman's computation look like if we assumed that the ghosts were doing 1-depth 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

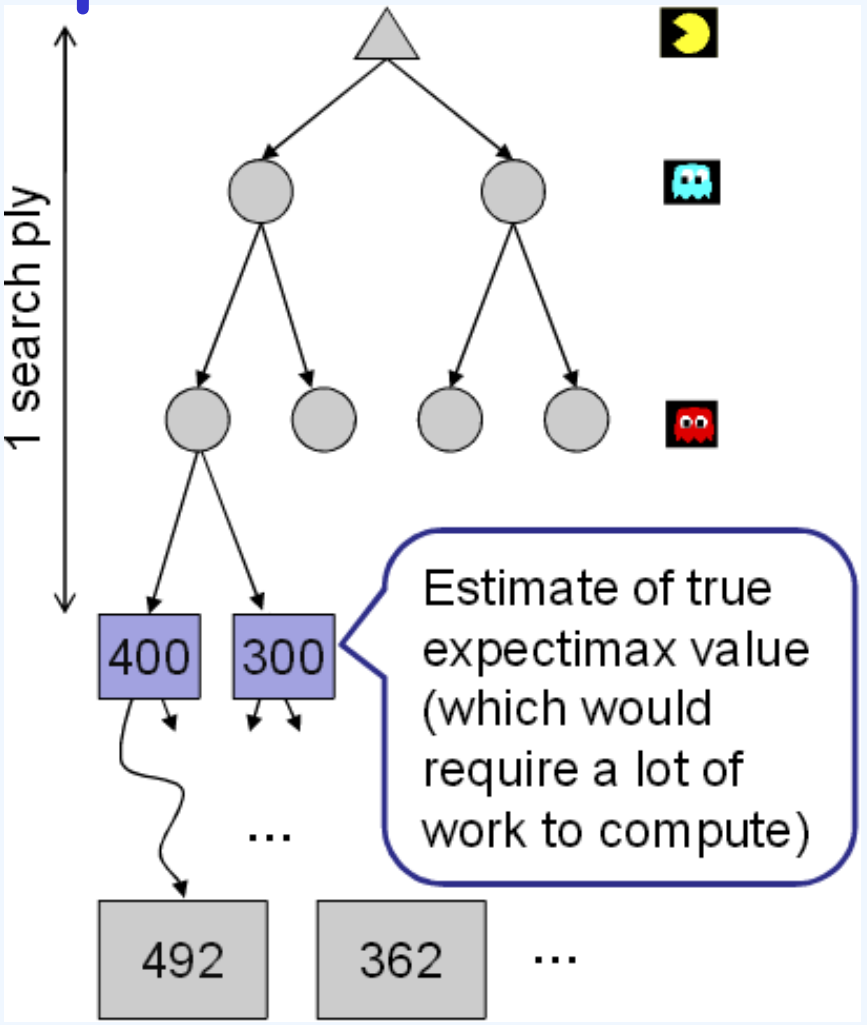


Pacman used depth 4 search with an eval function that avoids trouble

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

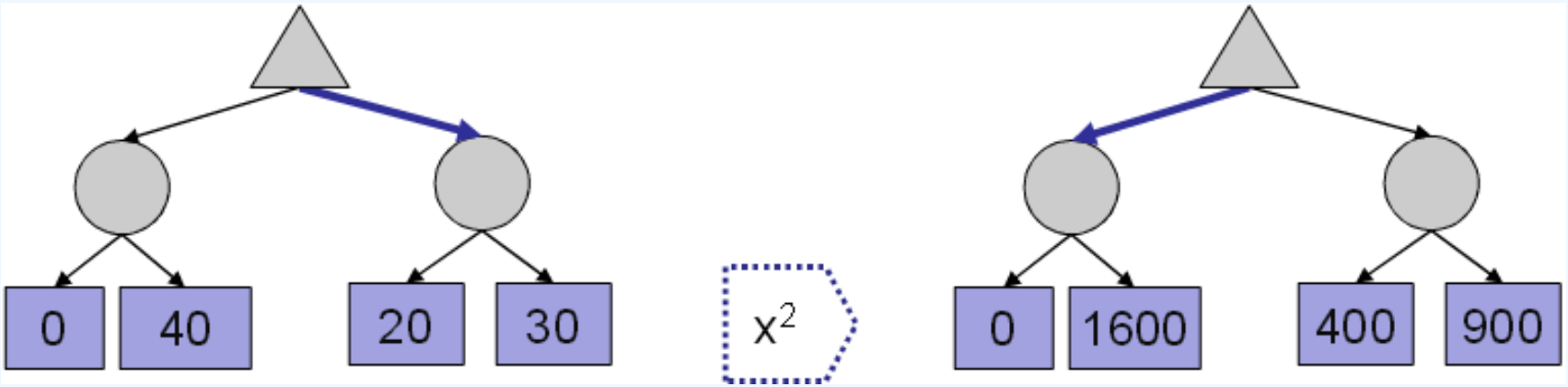
Expectimax Search with limited depth

- 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



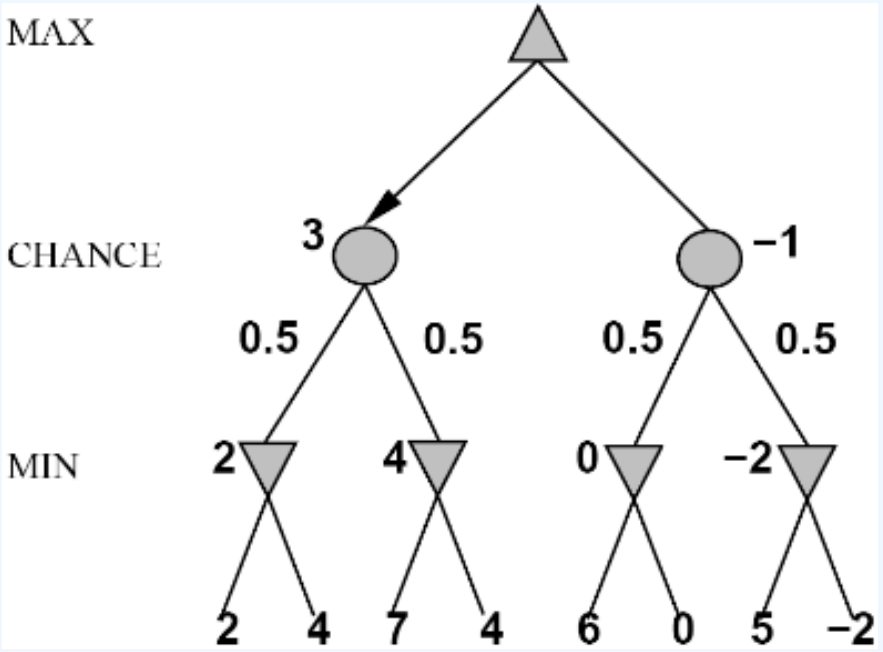
Expectimax Evaluation

- 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
  - We call this insensitivity to monotonic transformations
- For expectimax, we need magnitudes to be meaningful



Mixed Layer Types

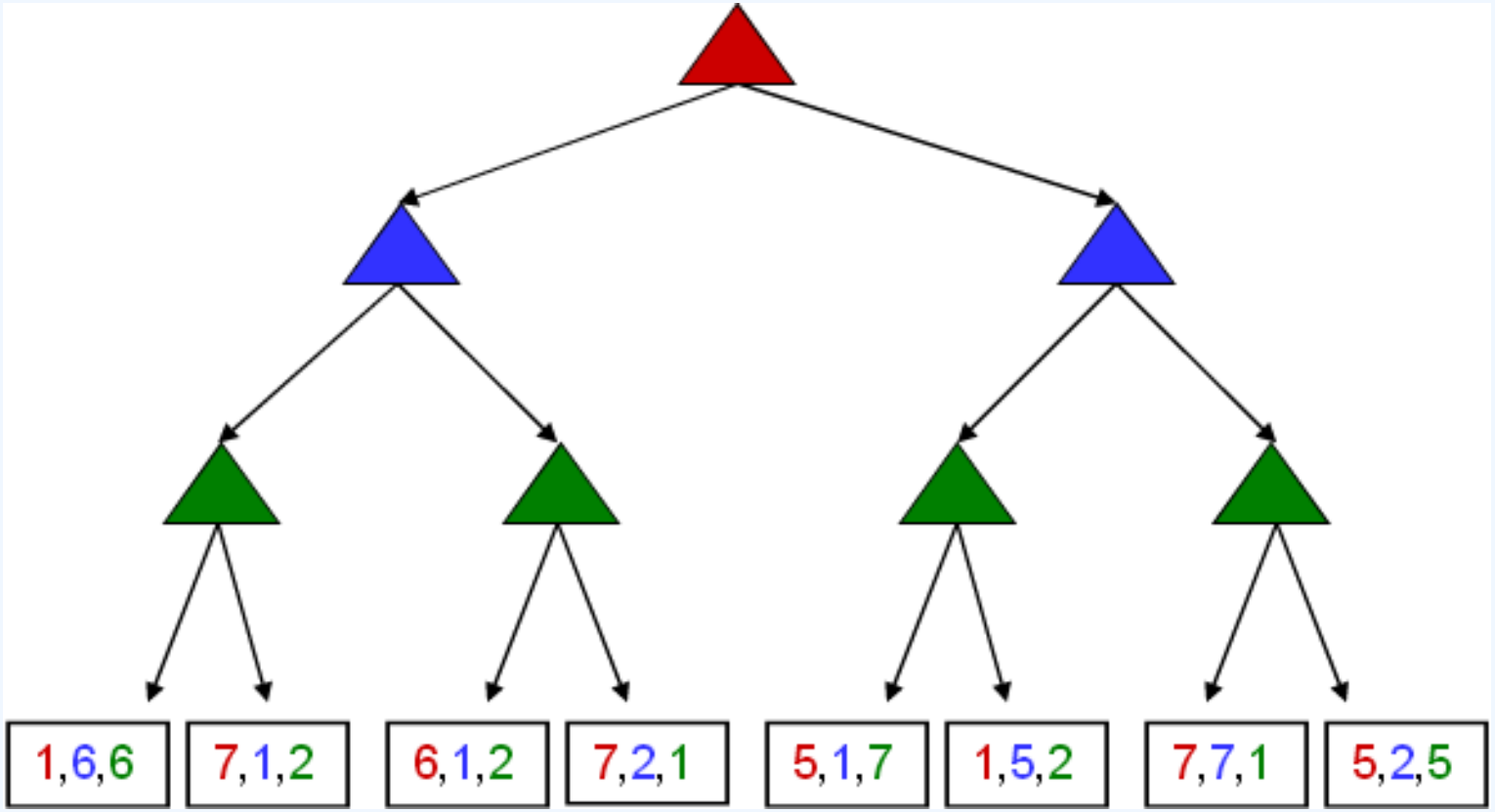
- E.g. Backgammon
- Expectiminimax
  - MAX node takes the max value of successors
  - MIN node takes the min value of successors
  - Chance nodes take expectations, otherwise like minimax



ExpectiMinimax-Value(*state*):

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*)

Multi-Agent Utilities



- Similar to minimax:
  - Terminals have utility tuples
  - Node values are also utility tuples
  - Each player maximizes its own utility

Recap

- Expecitmax search
  - search trees with chance nodes
  - c.f. minimax search
- Expectimax search with limited depth
  - use an evaluation function to estimate the outcome (Q4)
  - design a better evaluation function (Q5)
  - c.f. minimax search with limited depth