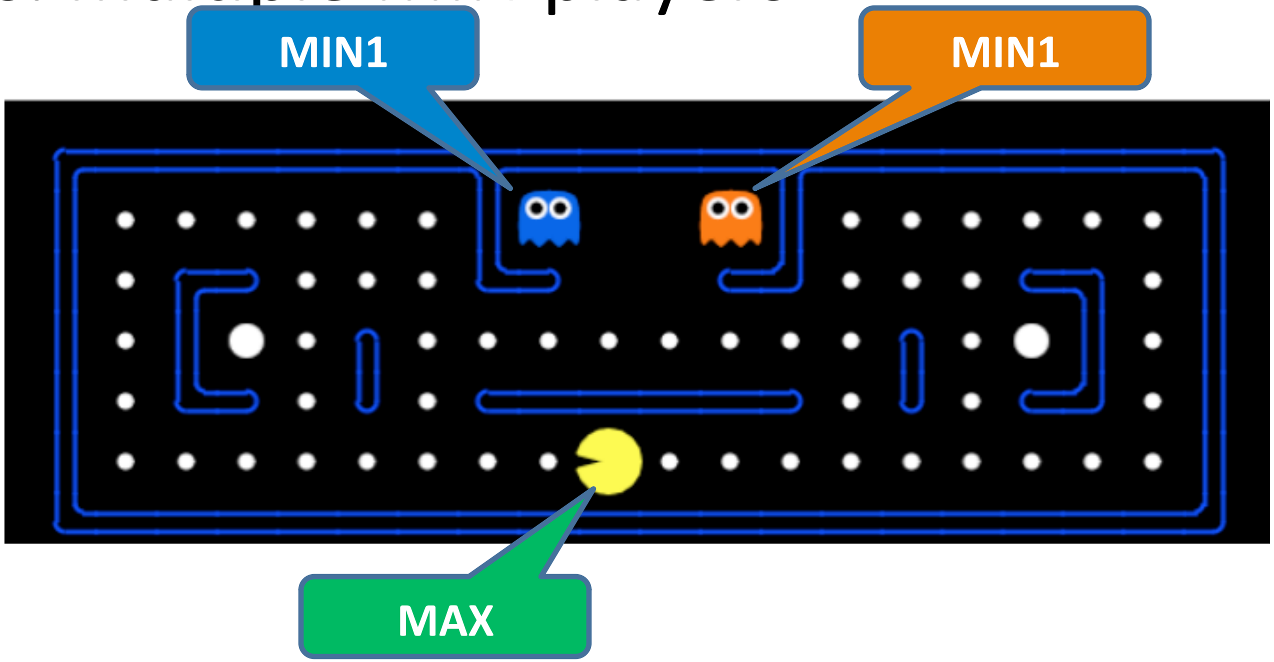
Adversarial Search: More on Eval Functions, ExpectiMax

With slides from
Dan Klein, Percy Liang, Luke Zettlemoyer

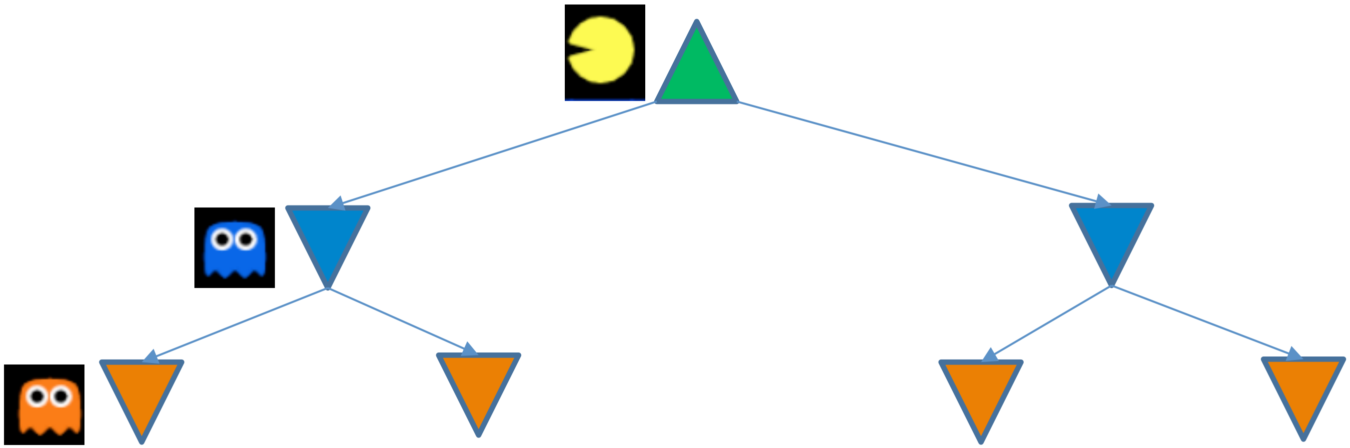
Multiplayer Games: 1 vs. All

- MAX vs. multiple MIN players

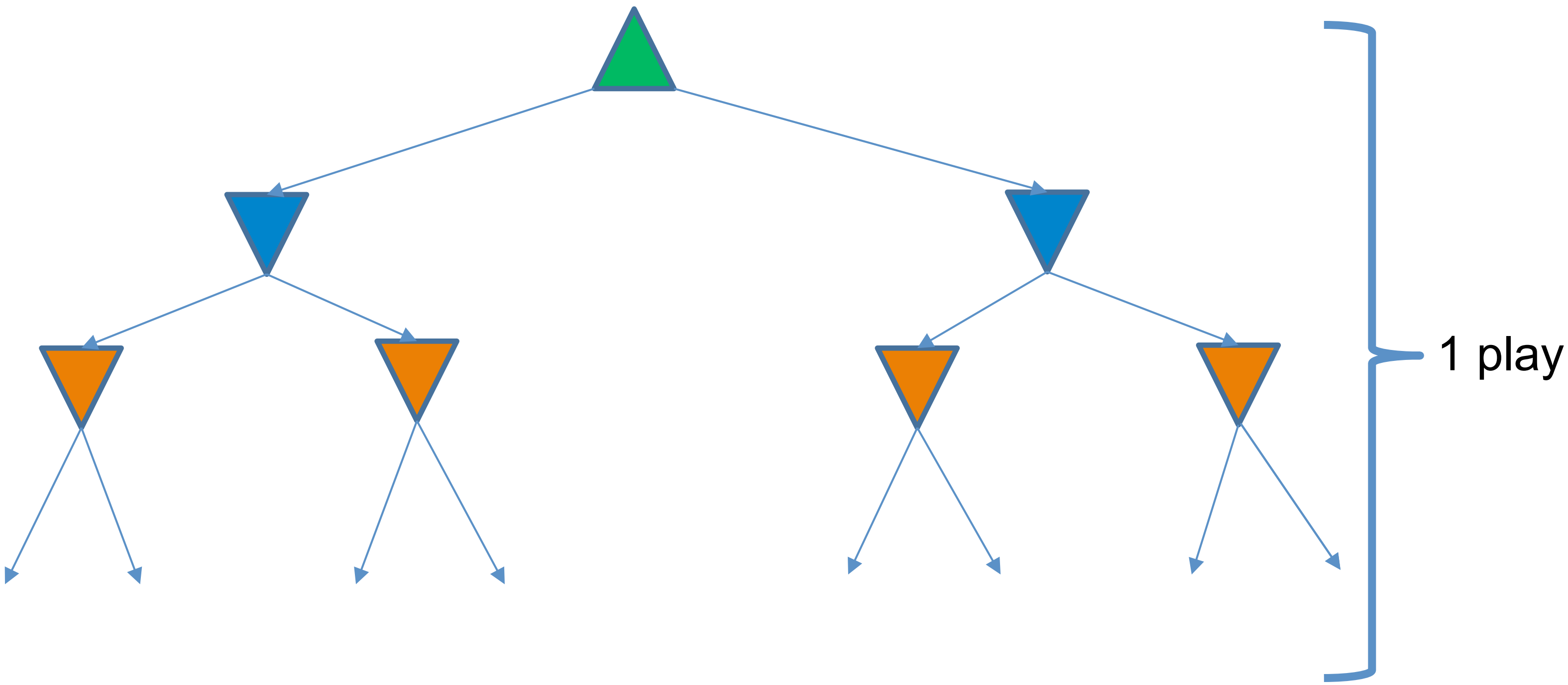


- How does the algorithm change?

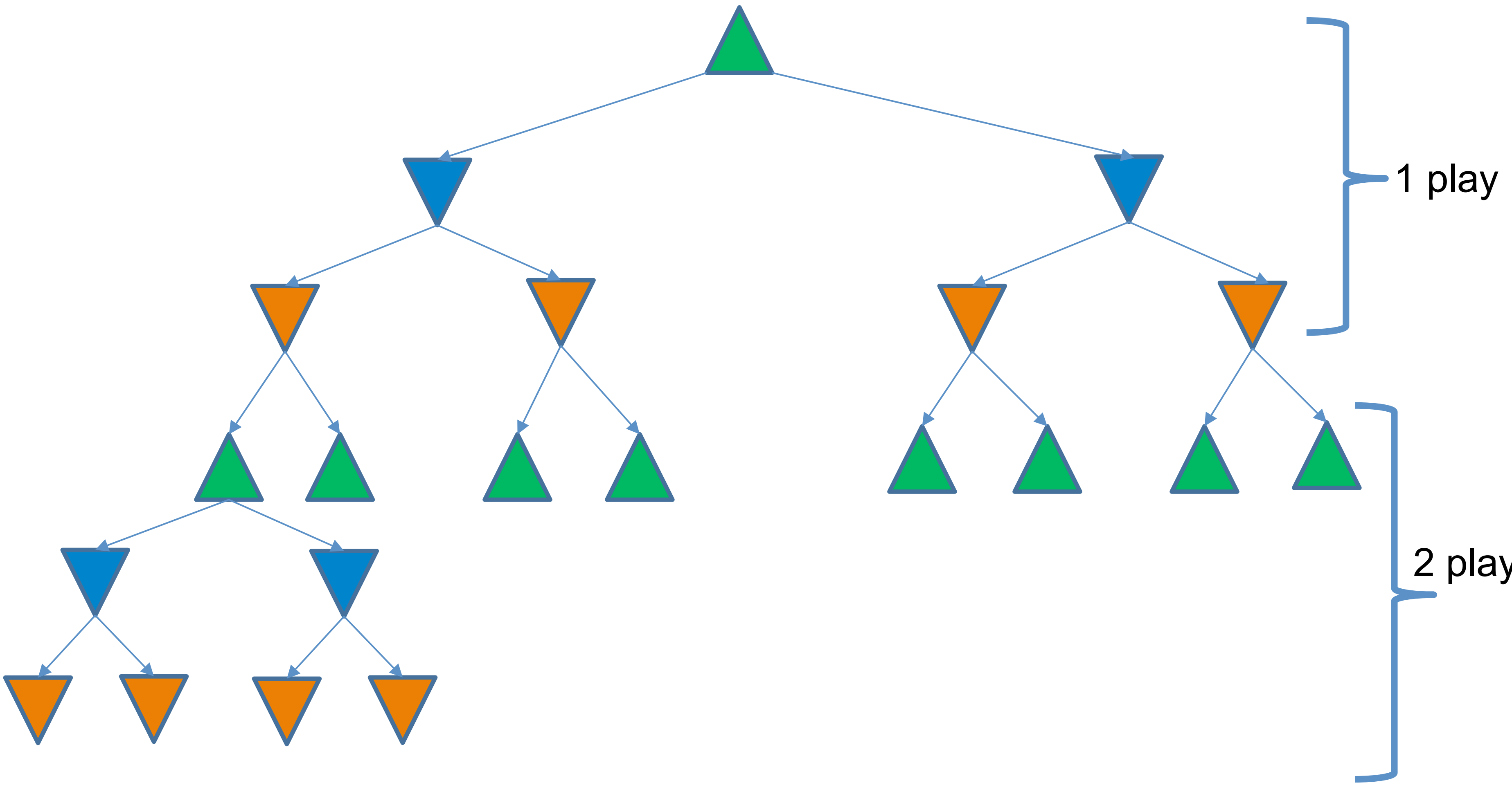
Multiplayer Game Tree: 1 Max, 2 MINs



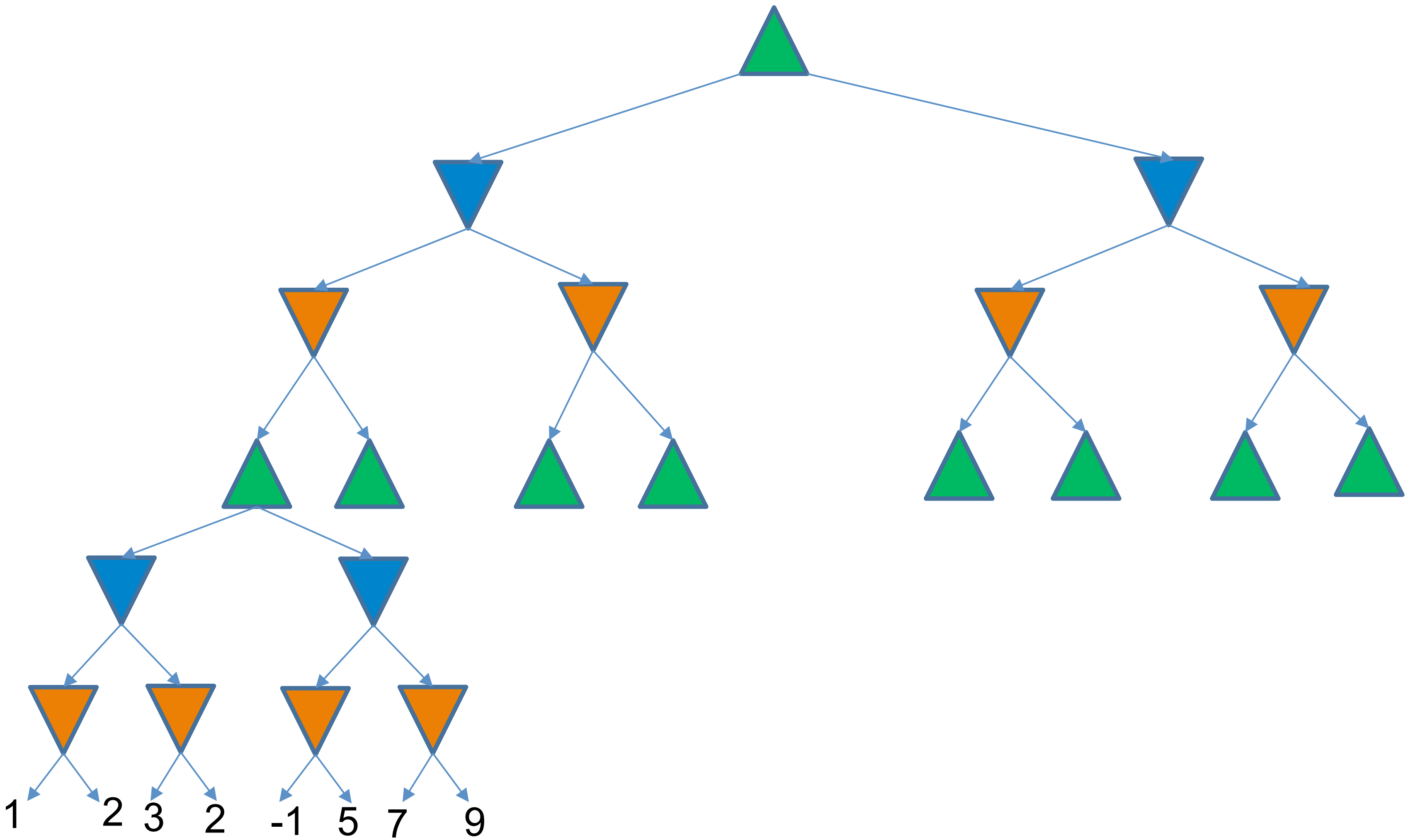
Multiplayer Game Tree: depth 1



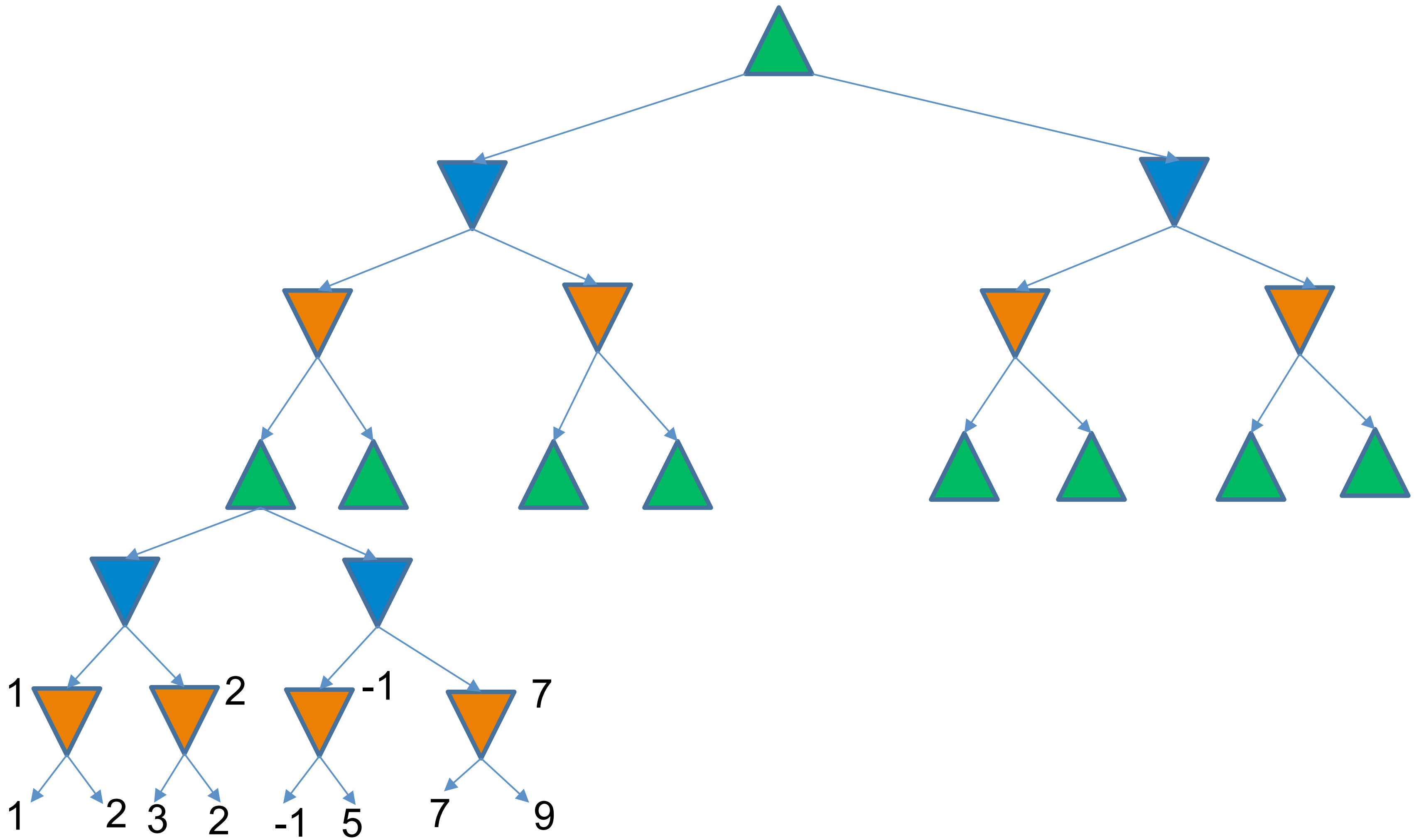
Multiplayer Game Tree: depth 2



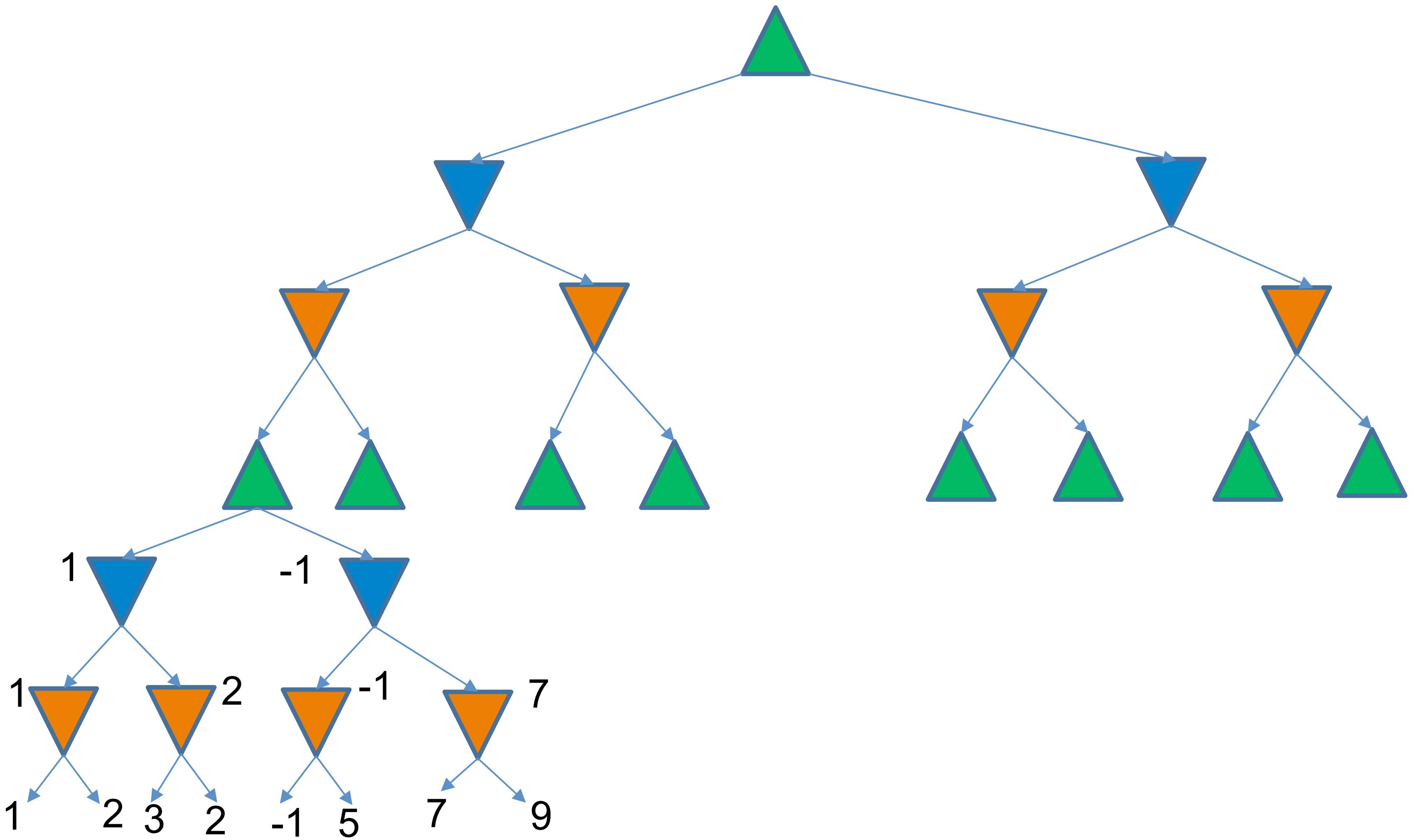
Multiplayer MiniMax: depth 2



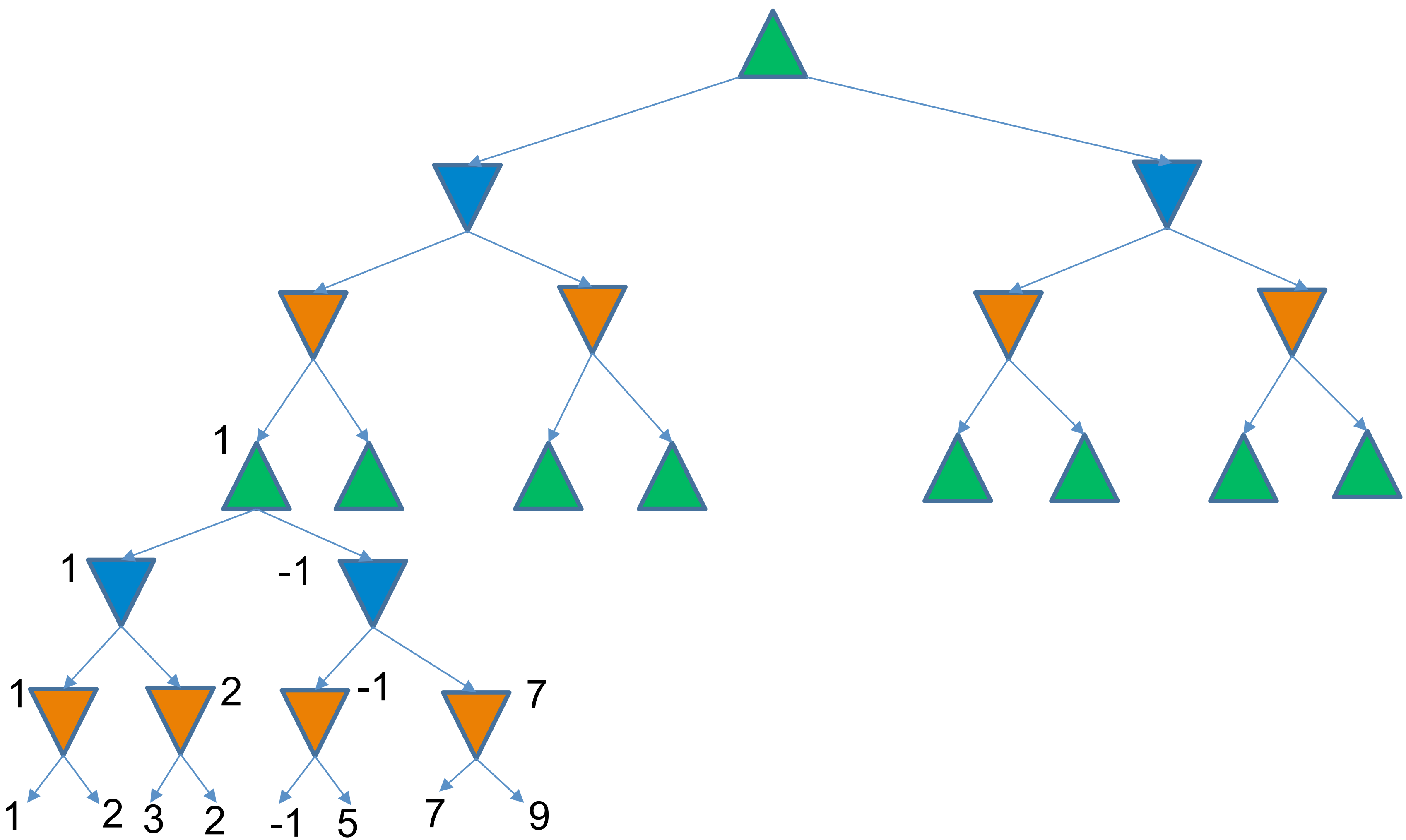
Multiplayer MiniMax: depth 2



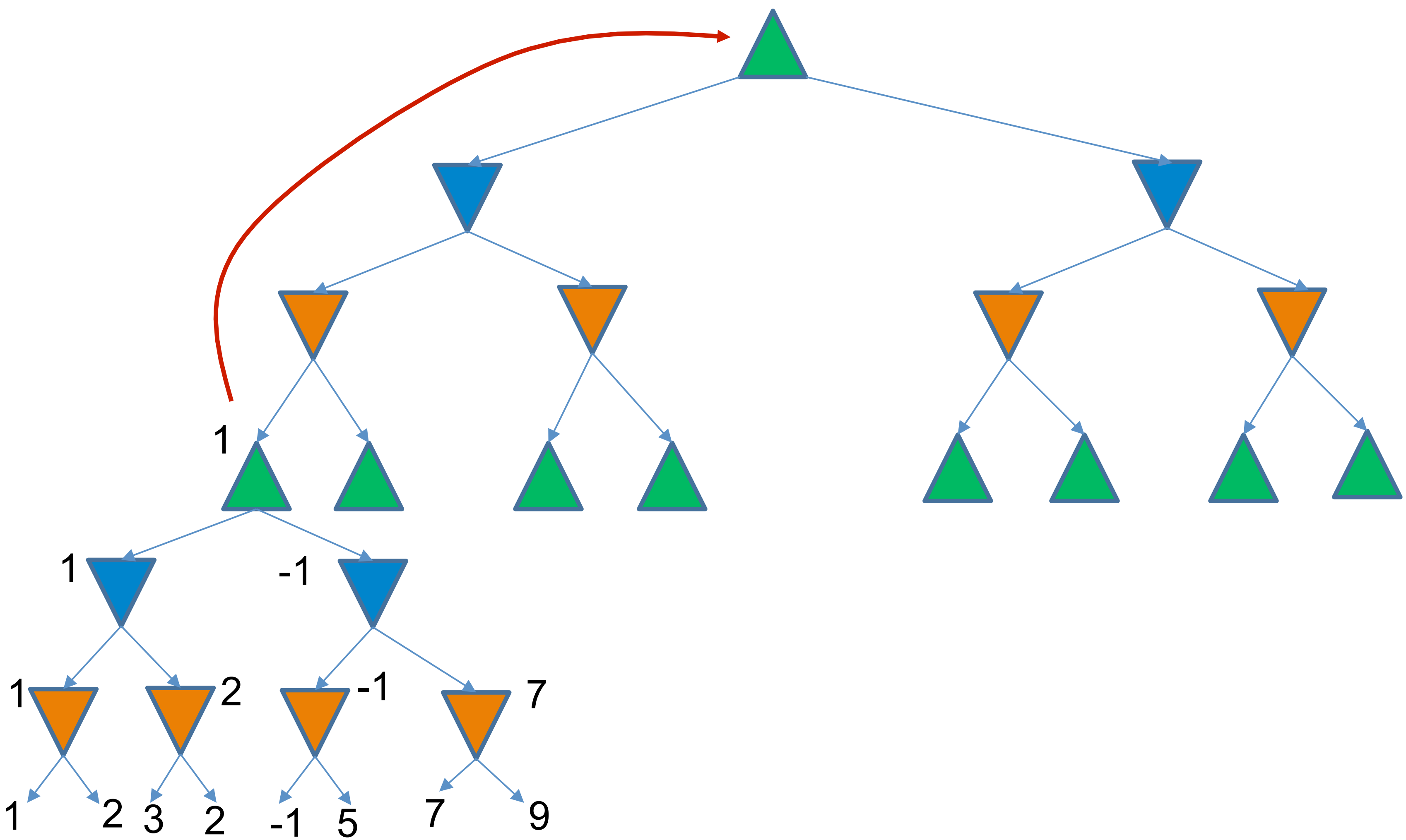
Multiplayer MiniMax: depth 2



Multiplayer MiniMax: depth 2



Multiplayer MiniMax: depth 2



MiniMax Algorithm: Multi-Min version

```
function MINIMAX-DECISION(state) returns an action
  inputs: state, current state in game
  v←MAX-VALUE(state)
  return the action in SUCCESSORS(state) with value v
```

```
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, 1))
  return v
```

Note:
MAX=Agent1
MIN1=Agent2
MIN2=Agent3
….
MINK=AgentN

```
function MIN-VALUE(state, agentIndex) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← ∞
  for a,s in SUCCESSORS(agentIndex, state) do
      if (agentIndex >= numAgents) then
          v_temp = MAX-VALUE(s)
      else #another ghost plays
          v_temp = MIN-VALUE(s, agentIndex+1))
      v ← MIN (v, v_temp)
return v
```

problem: collusion possible

- Previous slide (standard minimax analysis) assumes that each player operates to maximize only their own utility
- In practice, players **could make alliances**
  - Ex: C strong, A and B both weak
  - May be best for A and B to attack C rather than each other
- If game is not zero-sum (i.e., utility(A) = - utility(B) then alliances can be useful even with 2 players
  - e.g., both cooperate to maximum the sum of the utilities
- Ignore this, **assume non-cooperative games**

Minimax Algorithm: Analysis

- Complete depth-first exploration of the game tree
- Performance:
  - Max depth = d, b legal moves at each point
  - E.g., Chess: d ~ 100, b ~35

Criterion	Minimax
Time	O(b^m) ☹
Space	O(bm) ☺

Resource Limits

- Problem: In realistic games, cannot search to leaves!

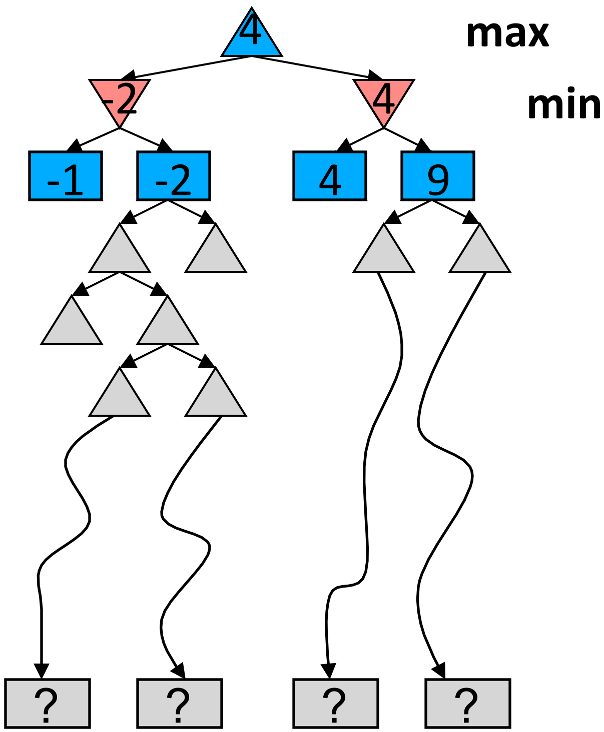
- Solution: ? Hint: 2 complementary approaches?
  1. ?
  2. ?

- Example:
  - Suppose we have 100 seconds, can explore 10K nodes / sec
  - So can check 1M nodes per move
  - α-β reaches about depth 8 – decent chess program

- Guarantee of optimal play is gone

- More plies makes a BIG difference

- Use iterative deepening for an anytime algorithm



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 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…

AlphaBeta Experiment

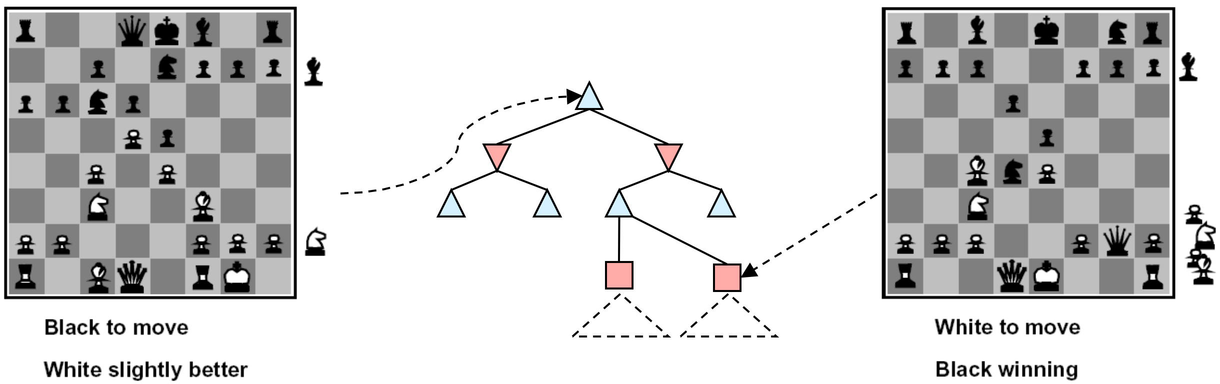
- time python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4 –q
- time python pacman.py -p AlphaBetaAgent -l minimaxClassic -a depth=4 -q

More Alpha-Beta Pruning Examples

- Generic game tree visualization: http://homepage.ufp.pt/jtorres/ensino/ia/alfabeta.html
- Example Play: Connect Four: https://gimu.org/connect-four-js/

(Better) Evaluation Functions

- Evaluation functions score non-terminals in depth-limited search



- Ideal function: returns the actual minimax value of the position
- In practice: typically weighted linear sum of features:

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

- e.g. $f_1(s)$ = (num white queens – num black queens), etc.

Construction of an Evaluation Function

- Usually a weighted sum of "features":

$$e(s) = \sum_{i=1}^{n} w_i f_i(s)$$

- Features may include
  - Number of pieces of each type
  - Number of possible moves
  - Number of squares controlled

More Examples of Evaluation Functions

- Reversi
  - Number squares held?
  - Better: number of squares held that cannot be flipped
  - Prefer valuable squares
    - NxN array w[i,j] of position values
    - Highest value: corners, edges
    - Lowest value: next to corner or edge
    - s[i,j] = +1 player, 0 empty, -1 opponent

$$score = \sum_{i,j} w[i,j]s[i,j]$$

Eval Function Approximation

Key idea: parameterized evaluation functions

Eval$(s; \mathbf{w})$ depends on weights $\mathbf{w} \in \mathbb{R}^d$

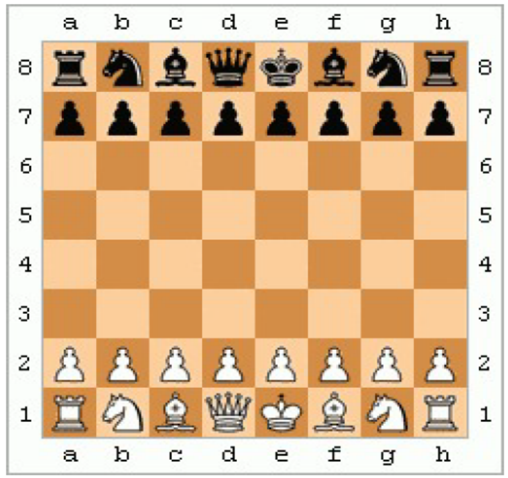
Feature vector: $\phi(s) \in \mathbb{R}^d$

$$\phi_1(s) = K - K'$$
$$\phi_2(s) = Q - Q'$$
$$\cdots$$

Linear evaluation function:

$$\text{Eval}(s; \mathbf{w}) = \mathbf{w} \cdot \phi(s)$$

Chess Example: Revisited



Example: chess

$$\begin{aligned}
\text{Eval}(s) &= \text{material} + \text{mobility} + \text{king-safety} + \text{center-control} \\
\text{material} &= 10^{100}(K - K') + 9(Q - Q') + 5(R - R') + \\
&\qquad 3(B - B' + N - N') + 1(P - P') \\
\text{mobility} &= 0.1(\text{num-legal-moves} - \text{num-legal-moves}') \\
\ldots
\end{aligned}$$

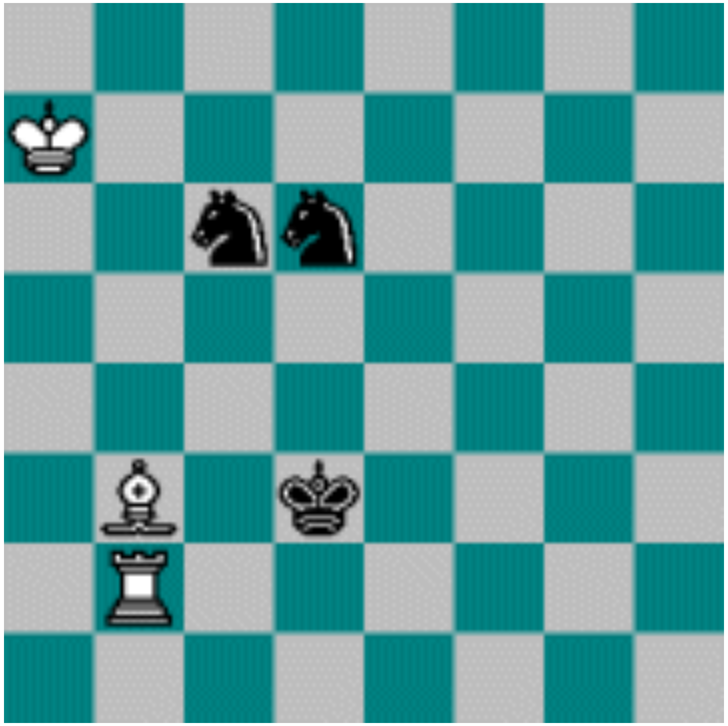
More Evaluation Functions

- Chess:
  - eval(s) =
    w1 * material(s) +
    w2 * mobility(s) +
    w3 * king safety(s) +
    w4 * center control(s) + ...
  - In practice MiniMax improves accuracy of heuristic eval function
  - But one can construct pathological games where more search hurts performance!
    (Nau 1981)

Idea: End-Game Databases

- Ken Thompson - all 5 piece end-games
- Lewis Stiller - all 6 piece end-games
  - Refuted common chess wisdom: many positions thought to be ties were really forced wins -- 90% for white
  - Is perfect chess a win for white?

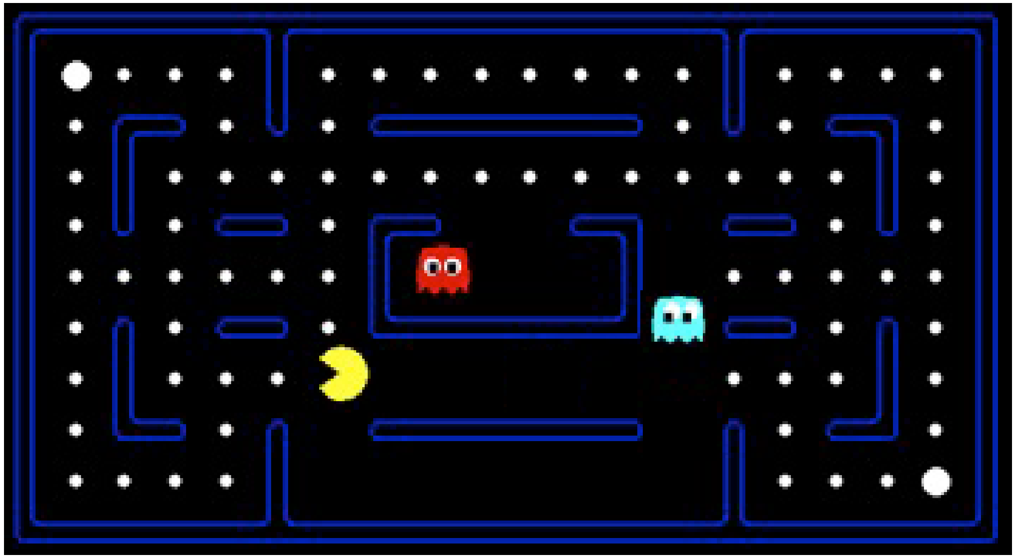
The MONSTER



White wins in 255 moves

(Stiller, 1991)

Evaluation for Pacman



What features would be good for Pacman?

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

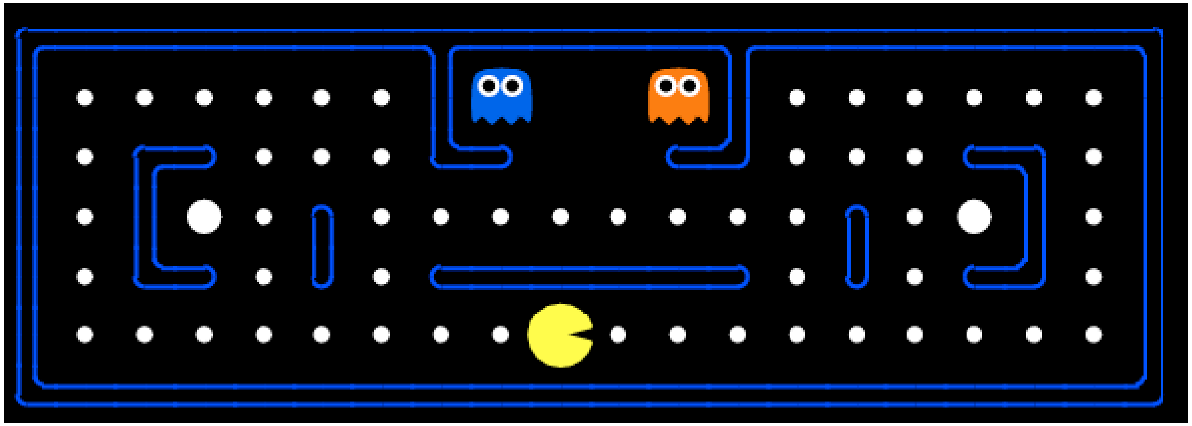
Pacman Eval Functions

- Use a linear combination of features.

  Eval(state) = f1*w1 + f2*w2

- Example:
  - f1: distance to nearest food
    w1: ?
  - f2: distance to nearest ghost
    w2:?

Pacman Eval Functions Demo



- Alpha-beta pruning, simple eval function:

```
python pacman.py -p AlphaBetaAgent -a depth=4 -l minimaxClassic  -q
python pacman.py -p AlphaBetaAgent -a depth=4 -l smallClassic  -q
```

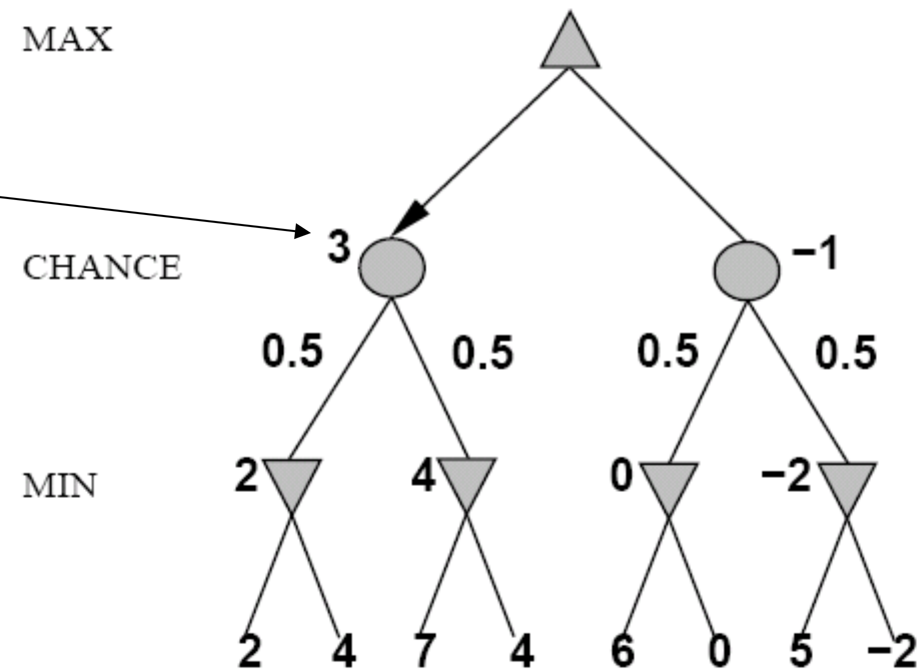
- Alpha-beta, better eval function:

```
python pacman.py -p AlphaBetaAgent -a depth=4,evalFn=better -l minimaxClassic  -q
python pacman.py -p AlphaBetaAgent -a depth=4,evalFn=better -l smallClassic  -q
```

Expected Minimax (ExpectiMax)

$$v = \sum_{chance\ nodes} P(n) \times Minimax(n)$$

$$3 = 0.5 \times 4 + 0.5 \times 2$$



Interleave chance nodes with min/max nodes

Again, the tree is constructed bottom-up

Idea: 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 (More Later)

- 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) = 0.50, P(T=heavy) = 0.25

- 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) = 0.25, P(T=heavy | Hour=8am) = 0.60
  - We'll talk about methods for reasoning and updating probabilities later

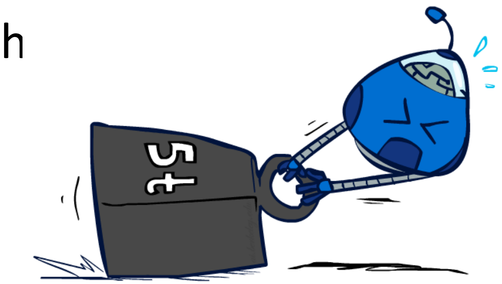
0.25

0.50

0.25

Reminder: Expectations

- The expected value of a function of a random variable is th average, weighted by the probability distribution over outcomes

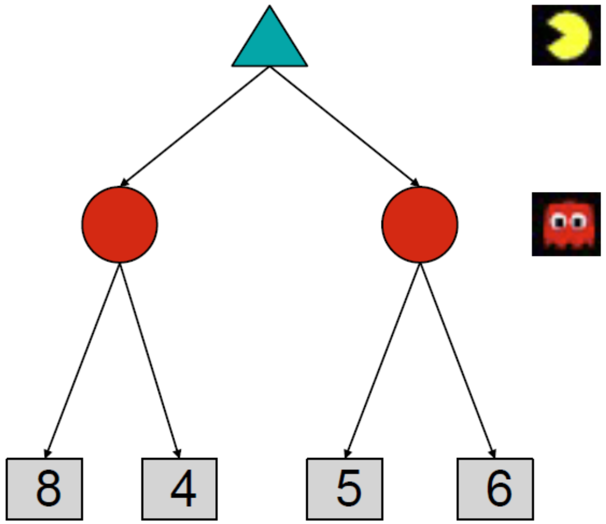


- Example: How long to get to the airport?

Time:	20 min		30 min		60 min		
	x	+	x	+	x	⇒	35 min
Probability:	0.25		0.50		0.25		

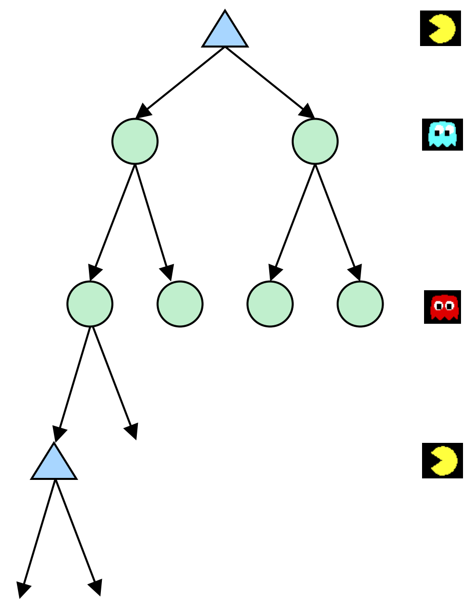
ExpectiMax: Revised Pseudocode

```
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)
```



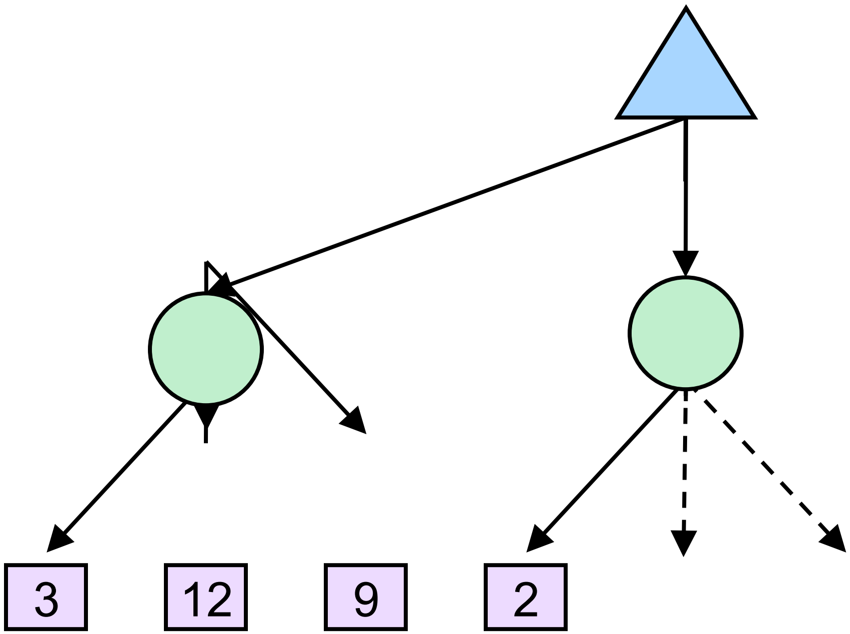
What Probabilities to Use?

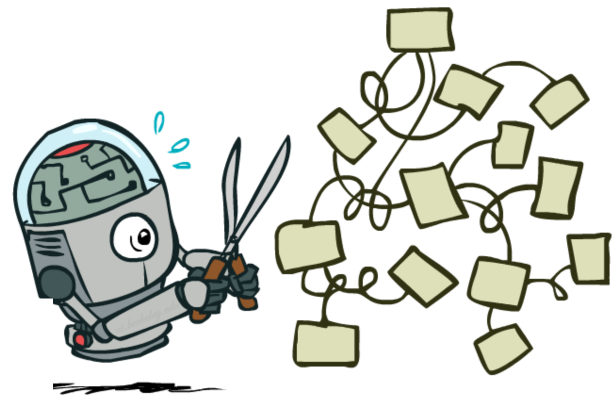
- In expectimax search, we have a probabilistic model of how the opponent (or environment) will behave in any state
  - Model could be a simple uniform distribution (roll a die)
  - Model could be sophisticated and require a great deal of computation
  - We have a chance node for any outcome out of our control: opponent or environment
  - The model might say that adversarial actions are likely!
- For now, assume each chance node magically comes along with probabilities that specify the distribution over its outcomes



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

Expectimax Pruning?



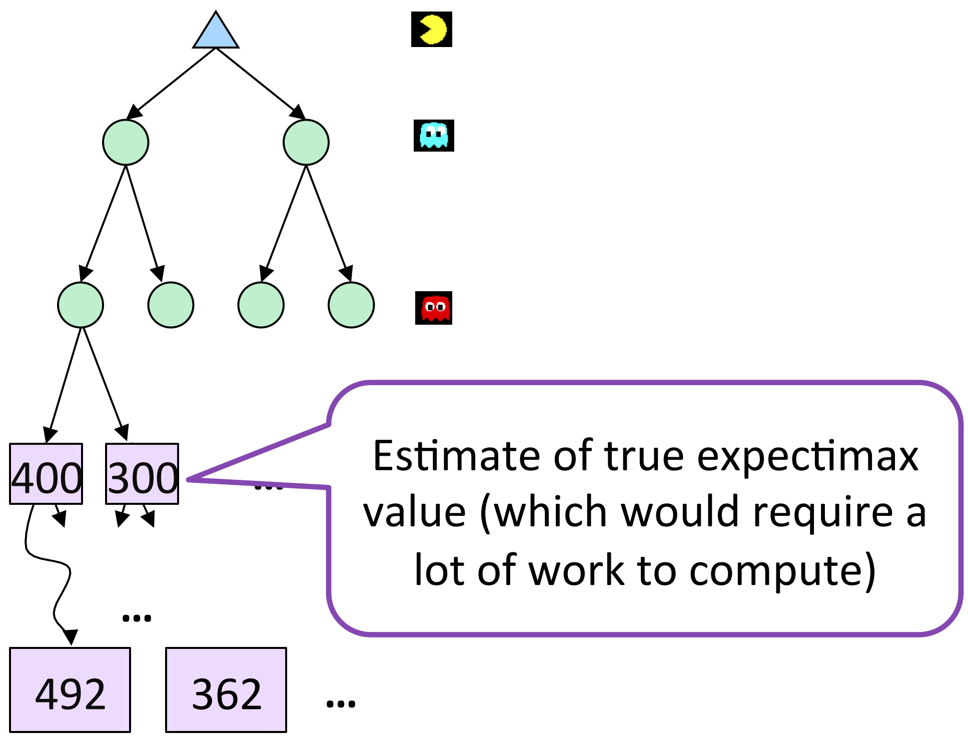


Theoretically possible, but:

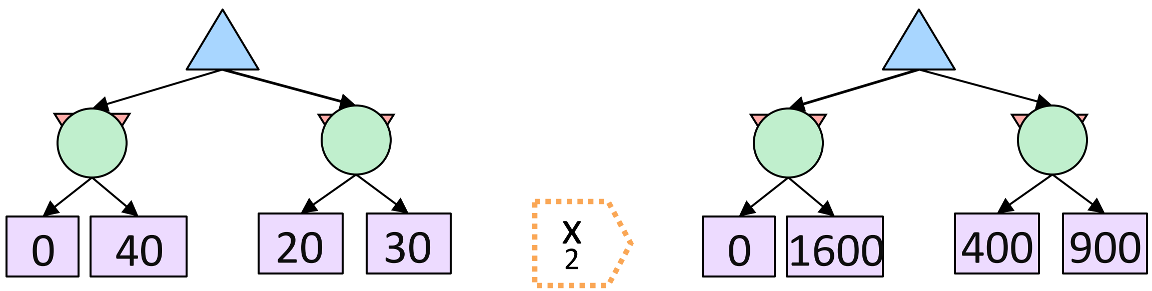
- Either accept that with some probability we will discard optimal solution
- OR: no guarantee that pruning will save much time

First algorithm (I know of) was published in 2009, makes a lot of assumptions: http://ieeexplore.ieee.org/document/5286476/

Depth-Limited Expectimax



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

Expectimax for Pacman

- Notice that we stepped 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 model MiniMax as special case of ExpectiMax?

ExpectiMax for Pacman

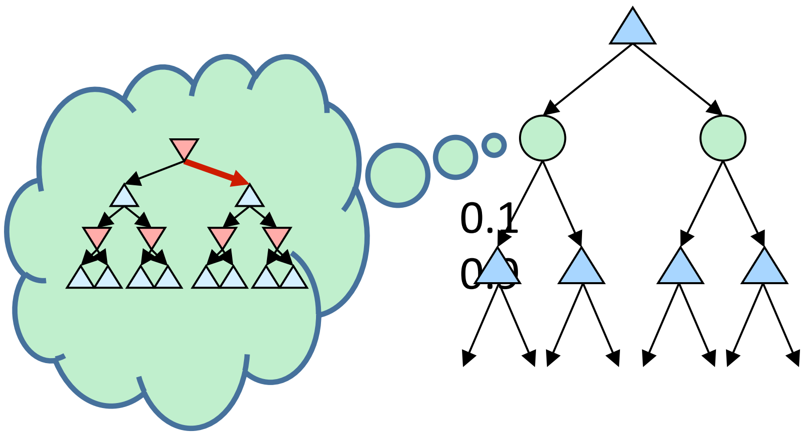
Results from playing 5 games

	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

Quiz: Informed Probabilities

- Let's say you know that your opponent is actually running a depth 2 minimax, using the result 80% of the time, and moving randomly otherwise
- Question: What tree search should you use?



- Answer: Expectimax!
  - To figure out EACH chance node's probabilities, you have to run a simulation of your opponent
  - This kind of thing gets very slow very quickly
  - Even worse if you have to simulate your opponent simulating you…
  - … except for minimax, which has the nice property that it all collapses into one game tree

Project 2: Assigned, Due Wed 2/15

- http://www.mathcs.emory.edu/~eugene/cs325/p2/
  - Use Piazza for discussions
  - Questions?