Announcements

- Project 1 is due **Friday, February 2**, 11:59 PM PT

- HW2 is due **Thursday, February 8**, 11:59 PM PT
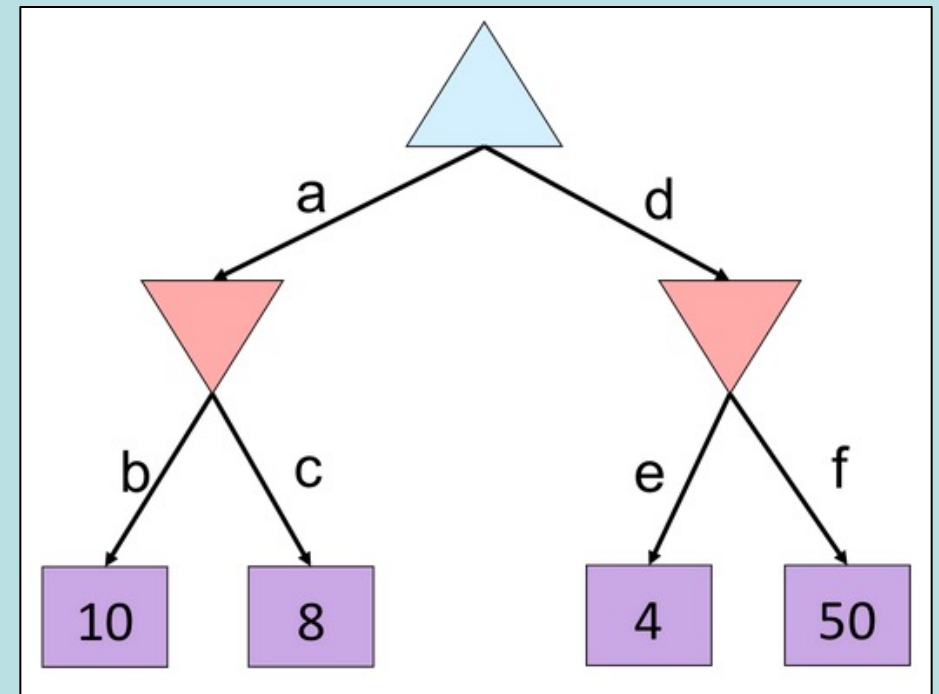
Pre-scan attendance QR code now!
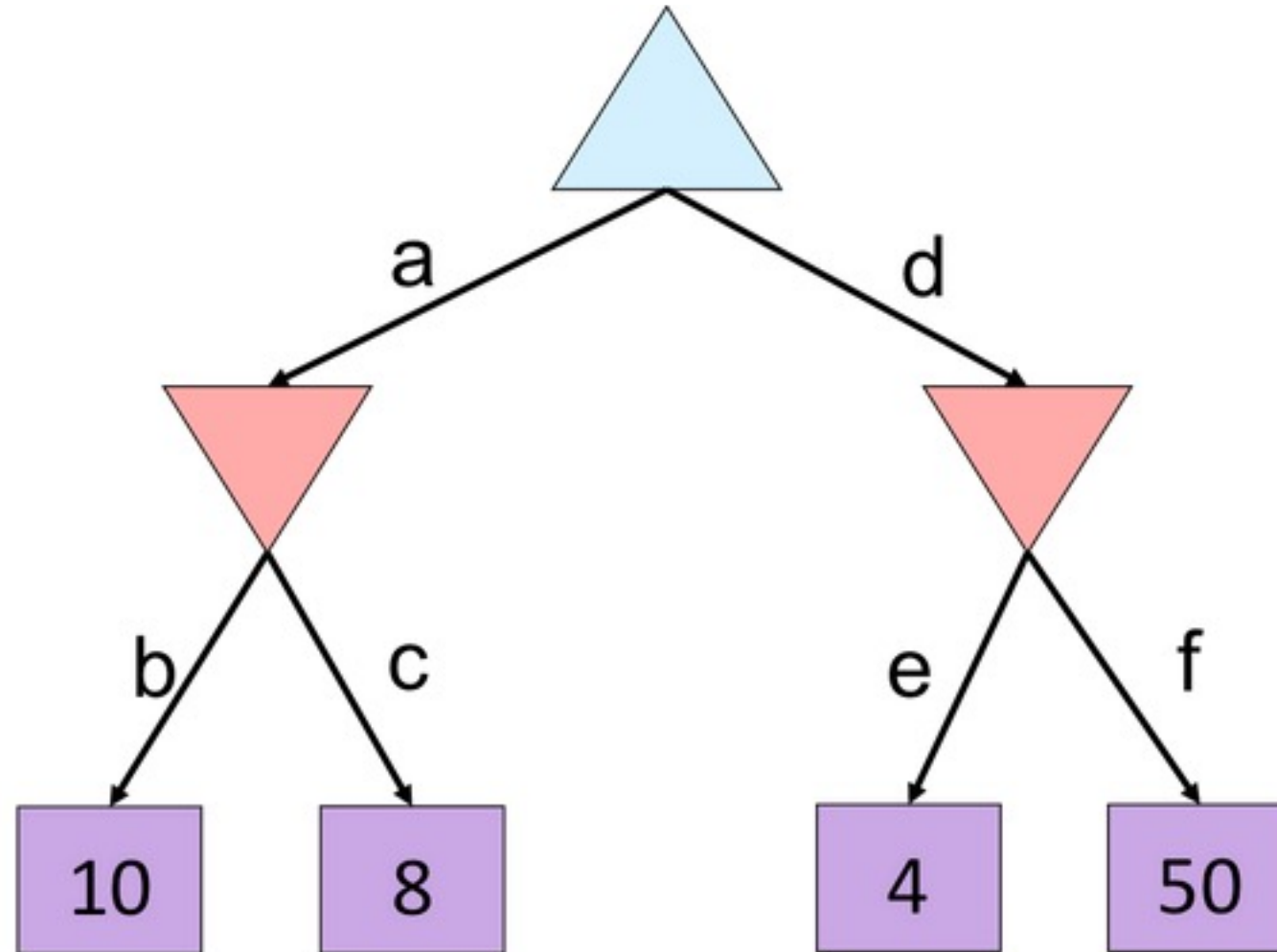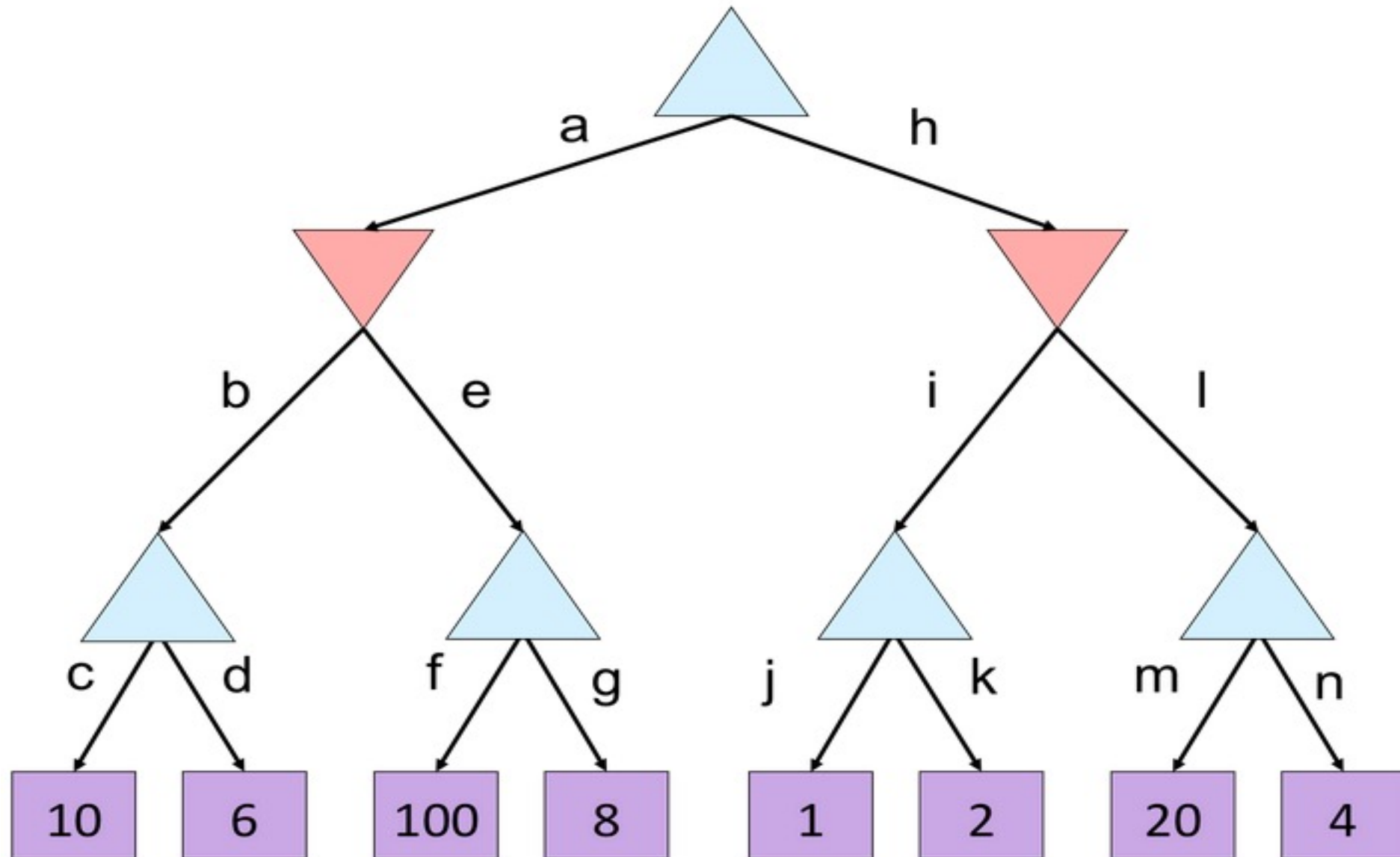
(Password appears later)

Alpha-Beta Pruning Quiz
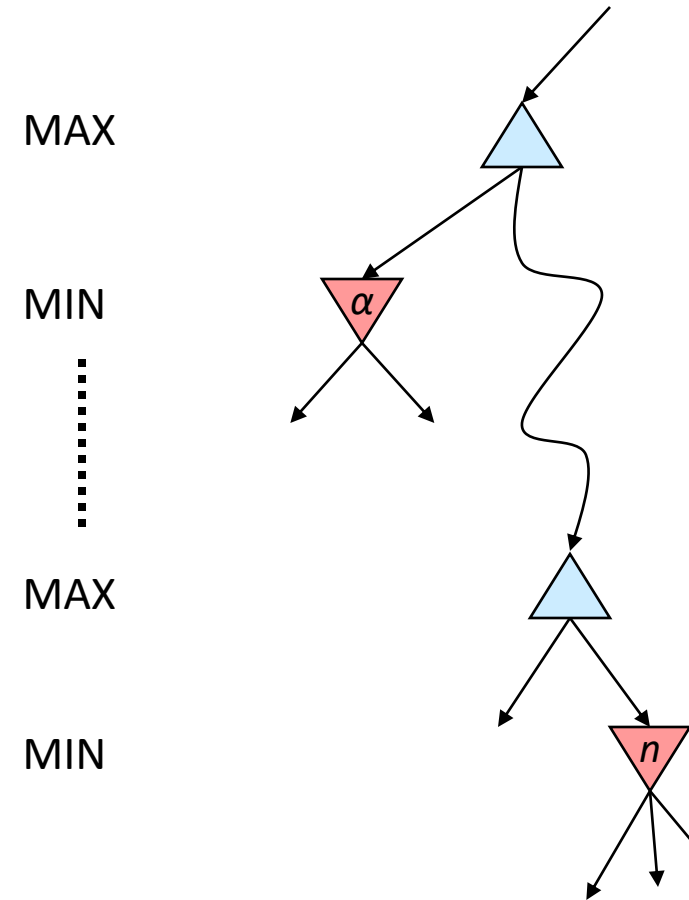
Which branches can we prune?

Alpha-Beta Pruning Quiz

Alpha-Beta Pruning

- **General case (pruning children of MIN node)**
  - We're computing the MIN-VALUE at some node *n*
  - We're looping over *n*'s children
  - *n*'s estimate of the childrens' min is dropping
  - Who cares about *n*'s value? MAX
  - Let **α** be the best value that MAX can get so far at any choice point along the current path from the root
  - If *n* becomes worse than **α**, MAX will avoid it, so we can prune *n*'s other children (it's already bad enough that it won't be played)
- **Pruning children of MAX node is symmetric**
  - Let **β** be the best value that MIN can get so far at any choice point along the current path from the root
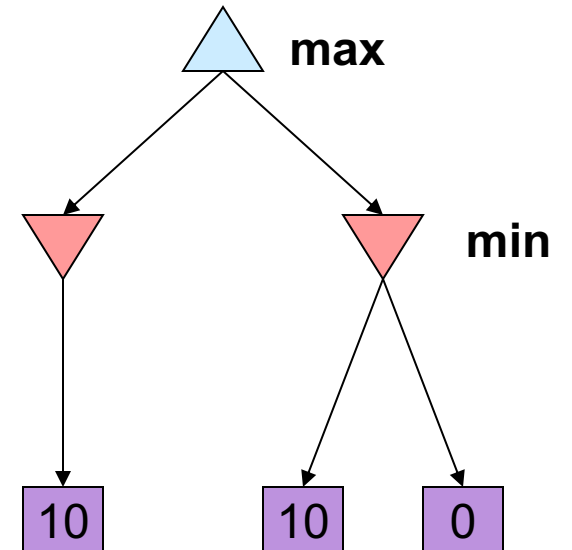
MAX

MIN α

MAX

MIN n

Alpha-Beta Pruning Implementation

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

```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

Alpha-Beta Pruning Properties

- This pruning has no effect on minimax value computed for the root!

- Values of intermediate nodes might be wrong
  - Important: children of the root may have the wrong value!

- 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...

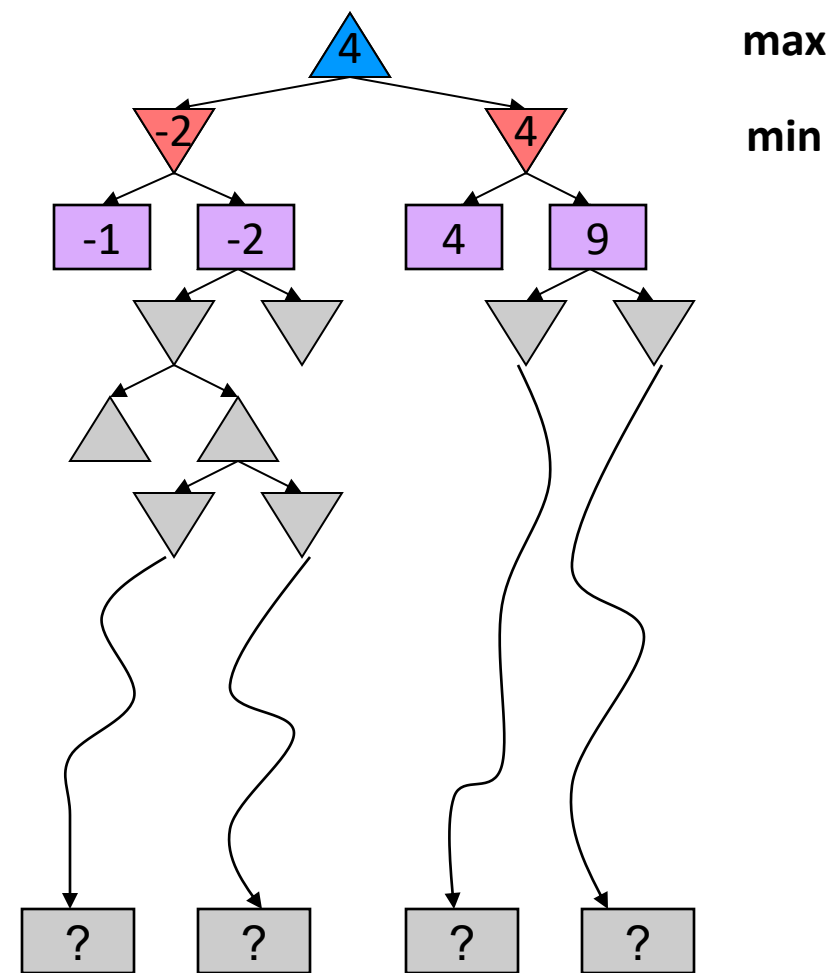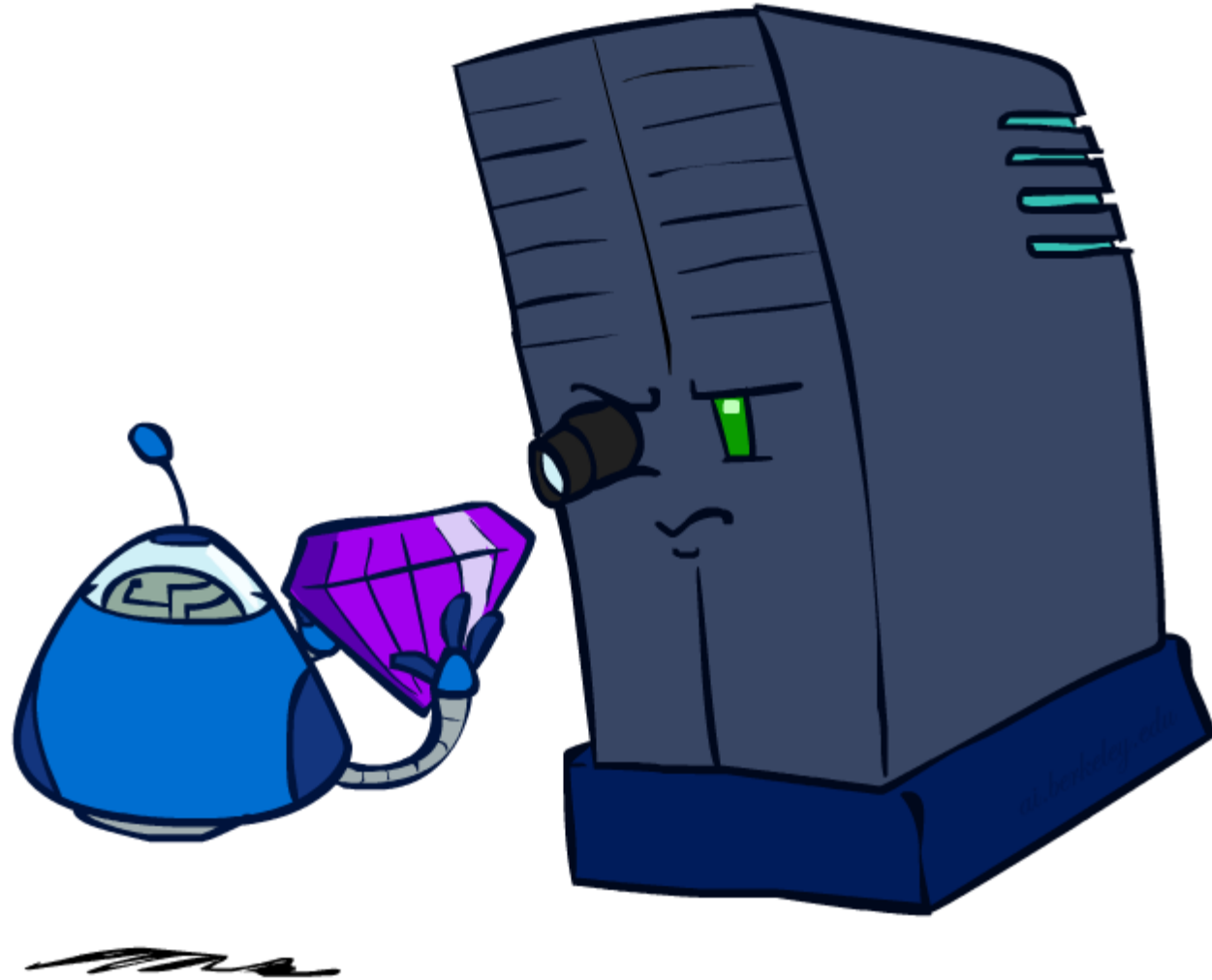- This is a simple example of metareasoning (computing about what to compute)

Resource Limits

Resource Limits

- **Problem: In realistic games, cannot search 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 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 (layers) makes a BIG difference**

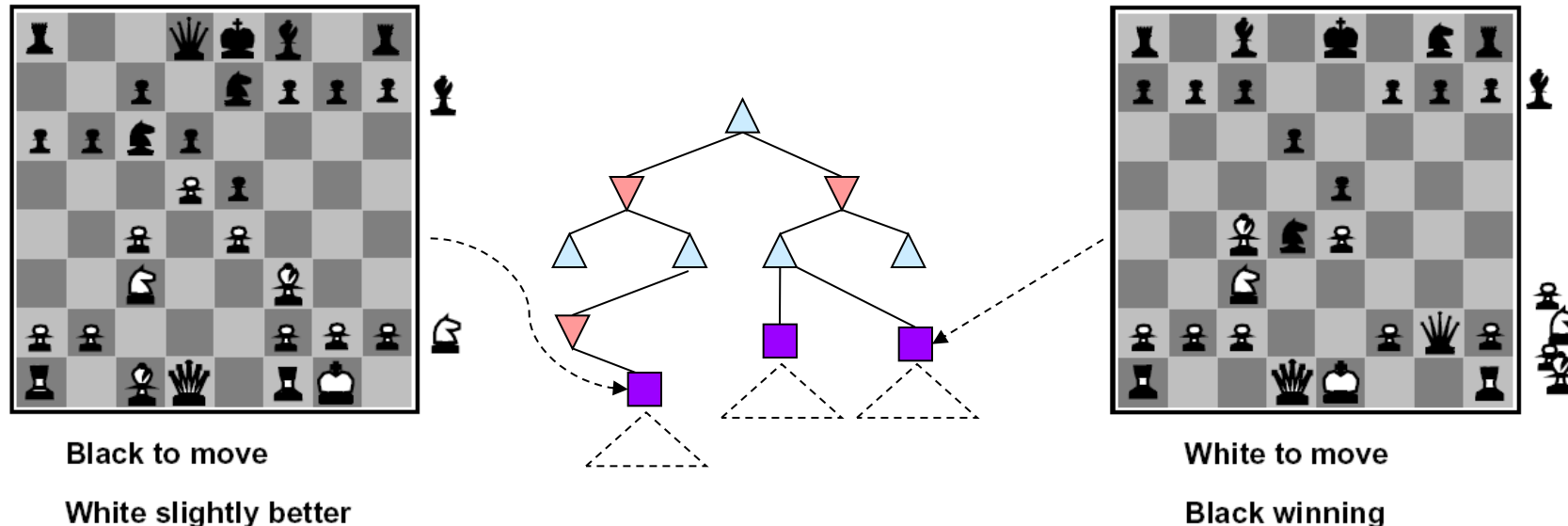- **Use iterative deepening for an anytime algorithm**

max
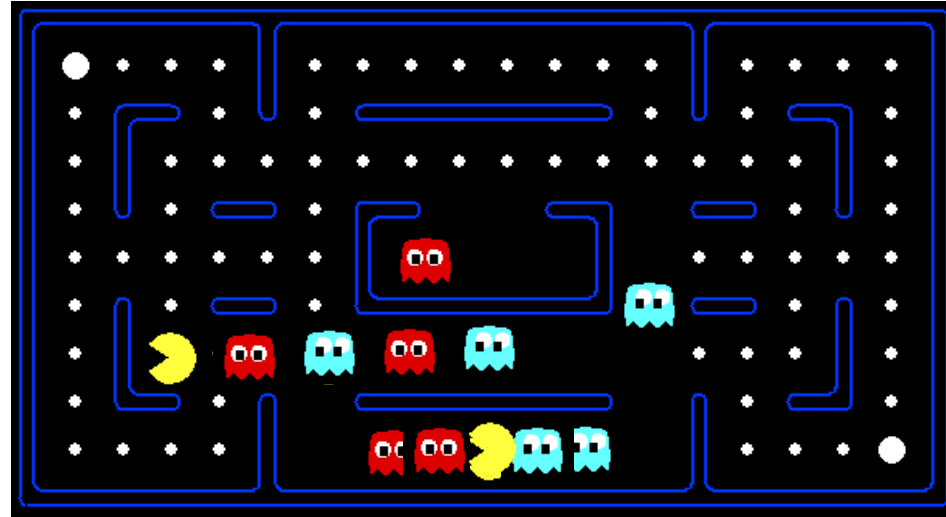
min

Evaluation Functions

Evaluation Functions

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

Black to move

White slightly better

White to move

Black winning

- 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.
- Or a more complex nonlinear function (e.g., NN) trained by self-play RL
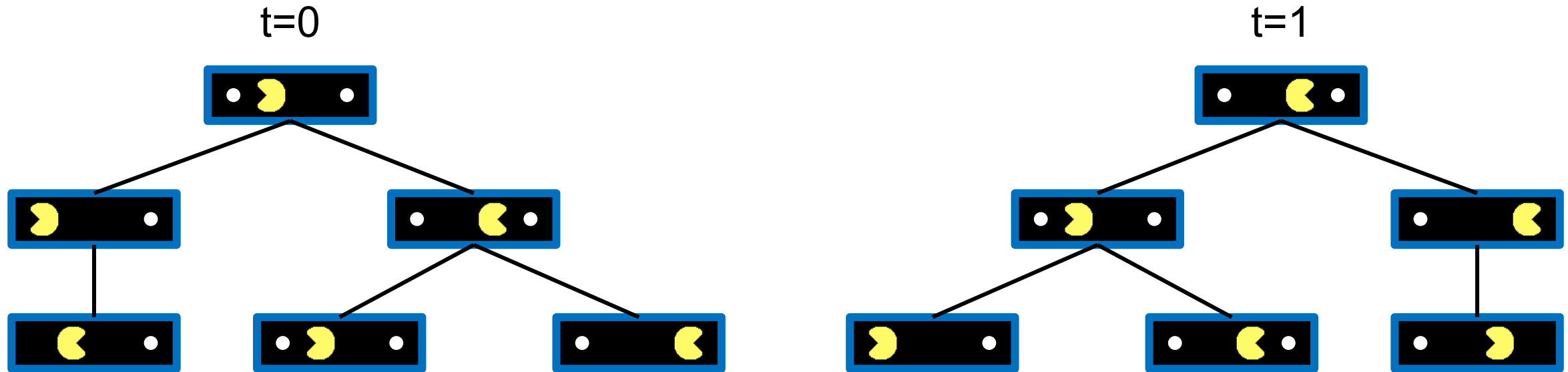
Evaluation for Pacman

[Demo: thrashing d=2, thrashing d=2 (fixed evaluation function), smart ghosts coordinate (L6D6,7,8,10)]

Video of Demo Thrashing (d=2)

Why Pacman Starves

t=0   t=1

- **A danger of replanning agents!**
  - 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, d=2)
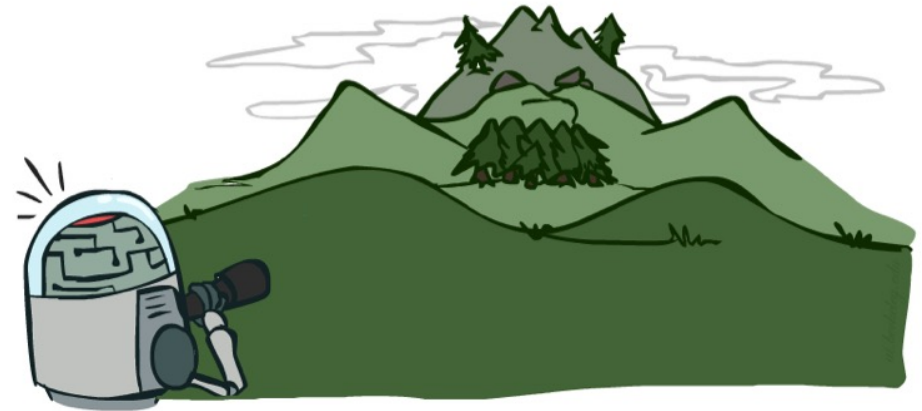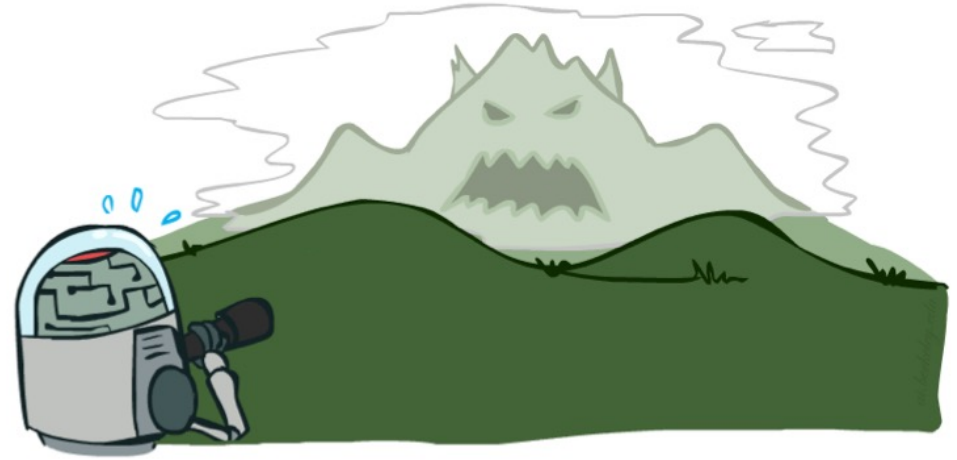 - Therefore, waiting seems just as good as eating: he may go east, then back west in the next round of replanning!

Video of Demo Thrashing -- Fixed (d=2)

[Demo: thrashing d=2, thrashing d=2 (fixed evaluation function) (L6D7)]

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

- An important example of the tradeoff between complexity of features and complexity of computation

Video of Demo Limited Depth (2)

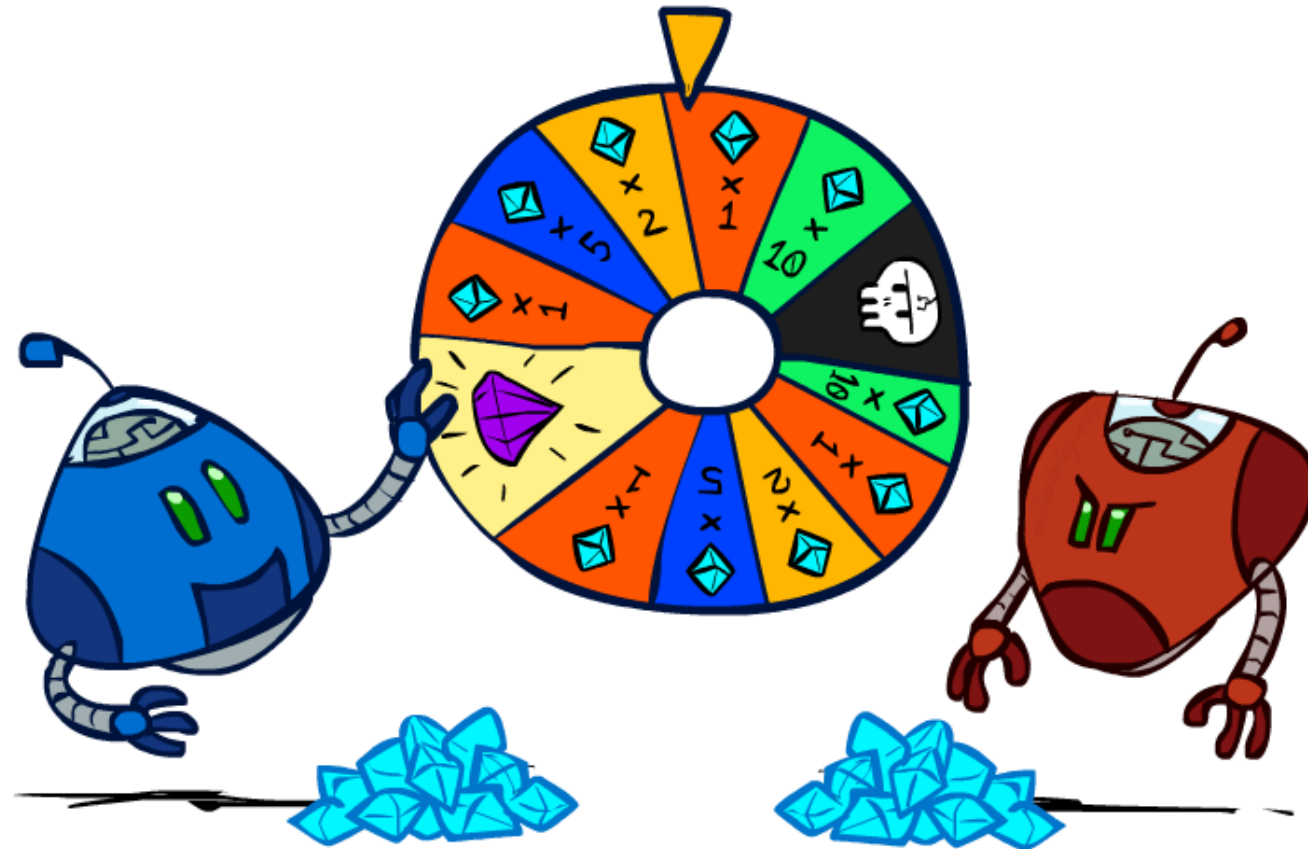# Synergies between Evaluation Function and Alpha-Beta?

- **Alpha-Beta: amount of pruning depends on expansion ordering**
  - Evaluation function can provide guidance to expand most promising nodes first (which later makes it more likely there is already a good alternative on the path to the root)
    - (somewhat similar to role of A* heuristic)

- **Alpha-Beta: (similar for roles of min-max swapped)**
  - Value at a min-node will only keep going down
  - Once value of min-node lower than better option for max along path to root, can prune
  - Hence: IF evaluation function provides upper-bound on value at min-node, and upper-bound already lower than better option for max along path to root THEN can prune
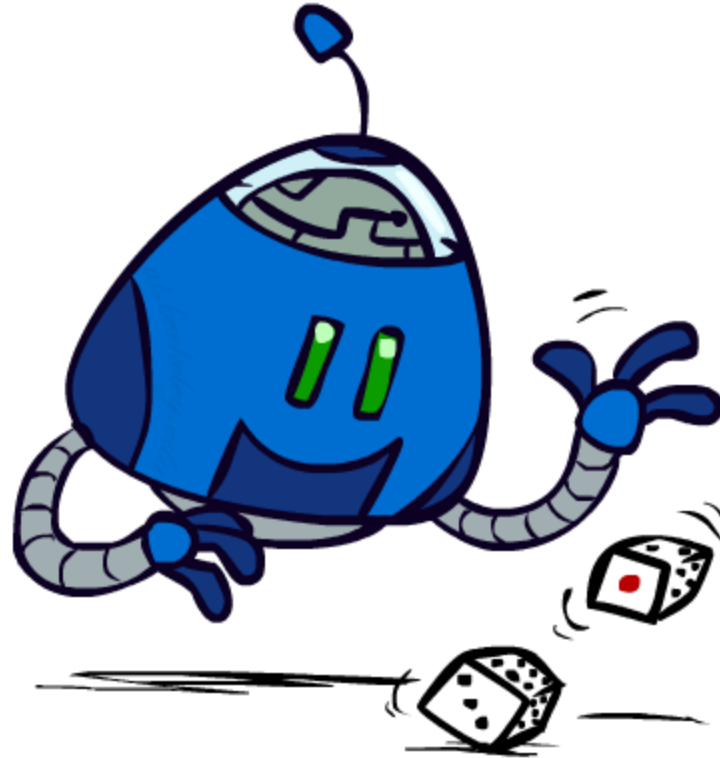
Summary

- Games are decision problems with ≥ 2 agents
    - Huge variety of issues and phenomena depending on details of interactions and payoffs
- For zero-sum games, optimal decisions defined by minimax
    - Simple extension to n-player "rotating" max with vectors of utilities
    - Implementable as a depth-first traversal of the game tree
    - Time complexity $O(b^m)$, space complexity $O(bm)$
- Alpha-beta pruning
    - Preserves optimal choice at the root
    - Alpha/beta values keep track of best obtainable values from any max/min nodes on path from root to current node
    - Time complexity drops to $O(b^{m/2})$ with ideal node ordering
- Exact solution is impossible even for "small" games like chess
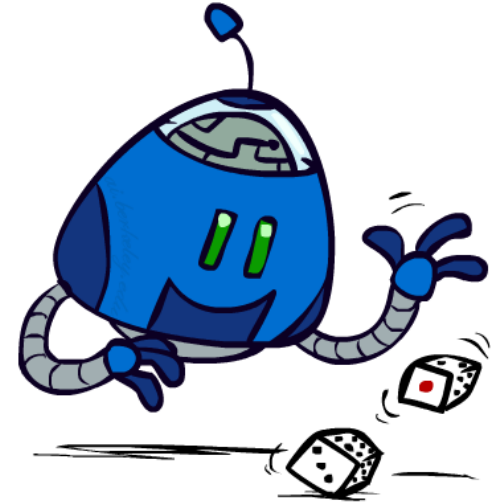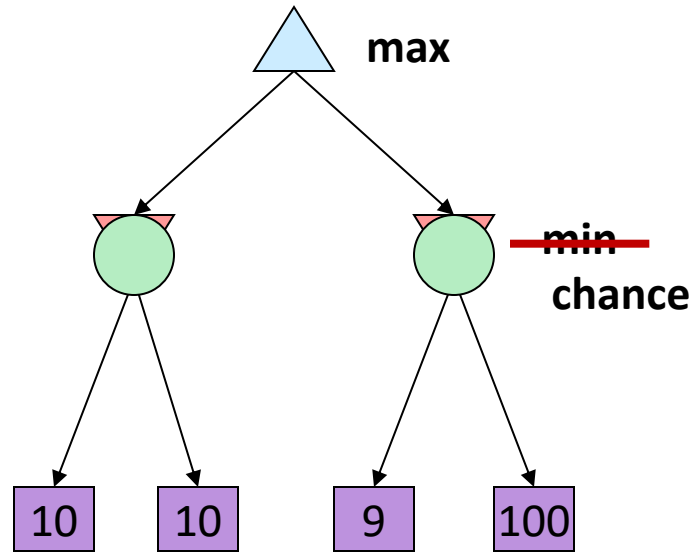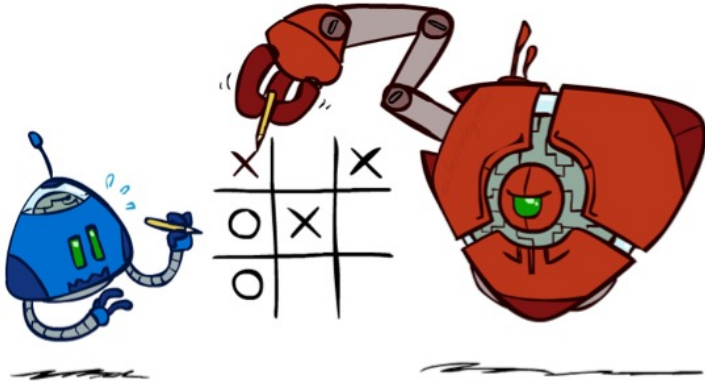
Next: Uncertainty!

Expectimax, Monte Carlo Tree Search
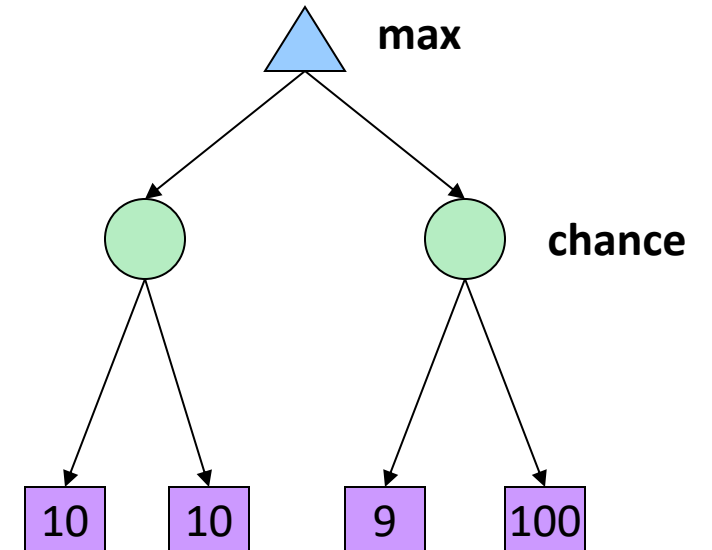
Uncertain Outcomes

Worst-Case vs. Average Case

Idea: Uncertain outcomes controlled by chance, not an adversary!

Expectimax Search

- **Why wouldn't we know what the result of an action will be?**
  - Explicit randomness: rolling dice
  - Unpredictable opponents: the ghosts respond randomly
  - Actions can fail: when moving a robot, wheels might slip

- **Values should now reflect average-case (expectimax) outcomes, not worst-case (minimax) outcomes**

- **Expectimax search: compute the average score under optimal play**
  - Max nodes as in minimax search
  - Chance nodes are like min nodes but the outcome is uncertain
  - Calculate their expected utilities
  - I.e. take weighted average (expectation) of children

- **Later, we'll learn how to formalize the underlying uncertain-result problems as Markov Decision Processes**

max

chance

| 10 | 10 | 9 | 100 |

Expectimax Pseudocode

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 EXP: return exp-value(state)

def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v

def exp-value(state):
    initialize v = 0
    for each successor of state:
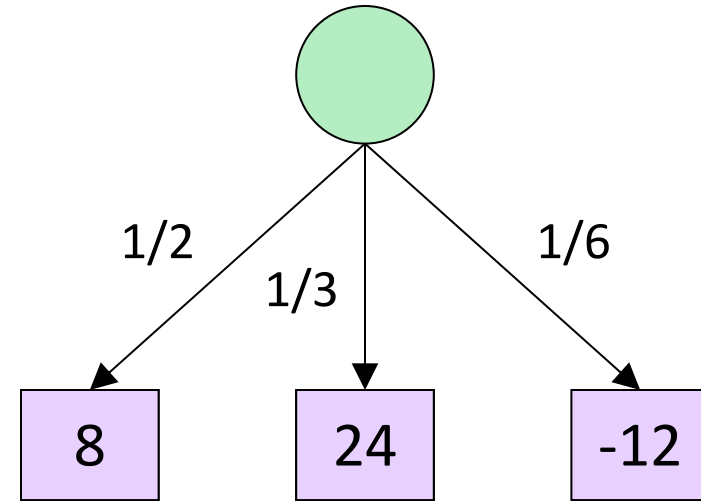   p = probability(successor)
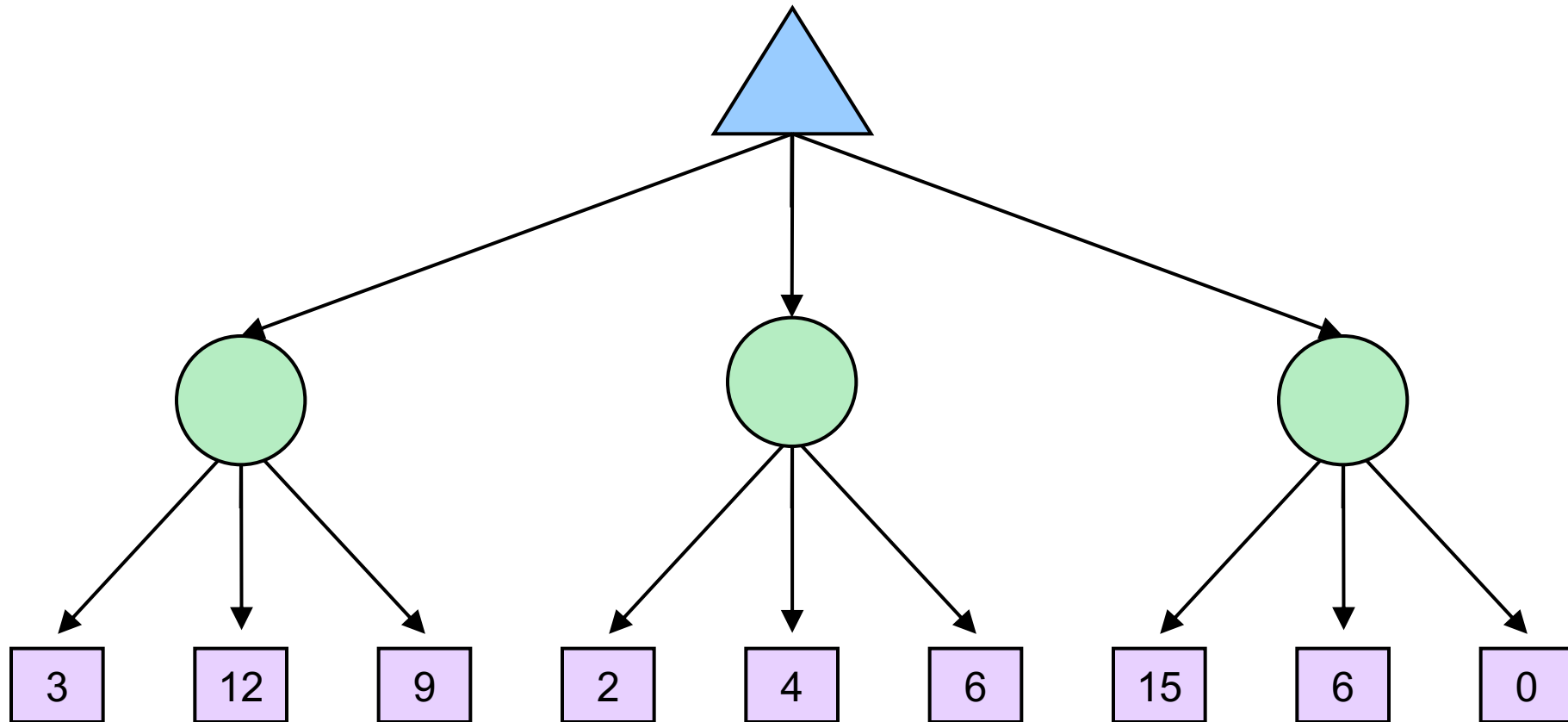   v += p * value(successor)
    return v

Expectimax Pseudocode

def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
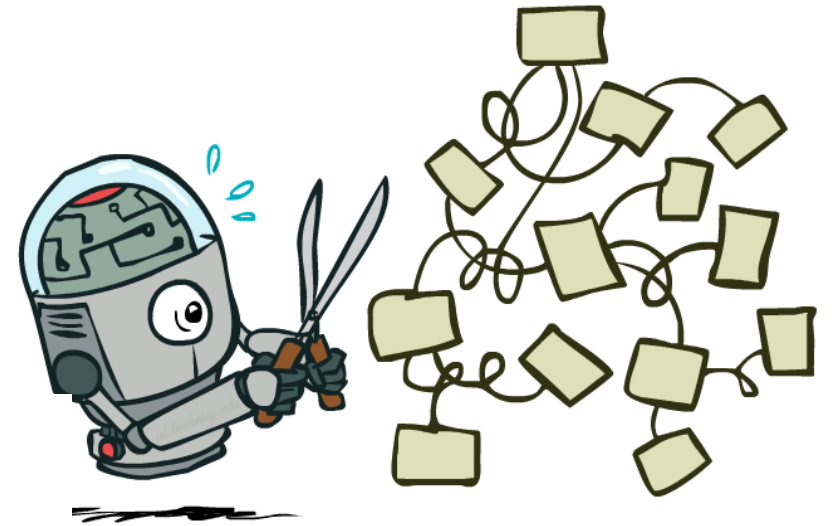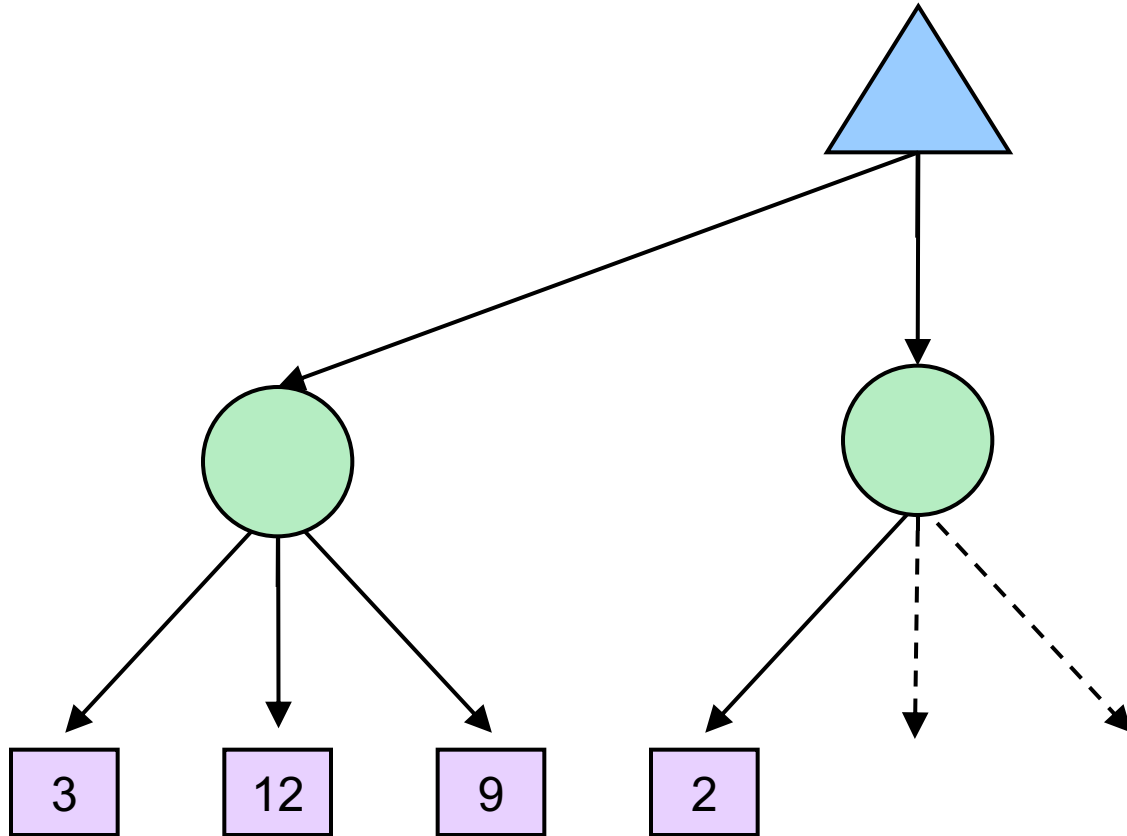   return v

$$v = (1/2)(8) + (1/3)(24) + (1/6)(-12) = 10$$
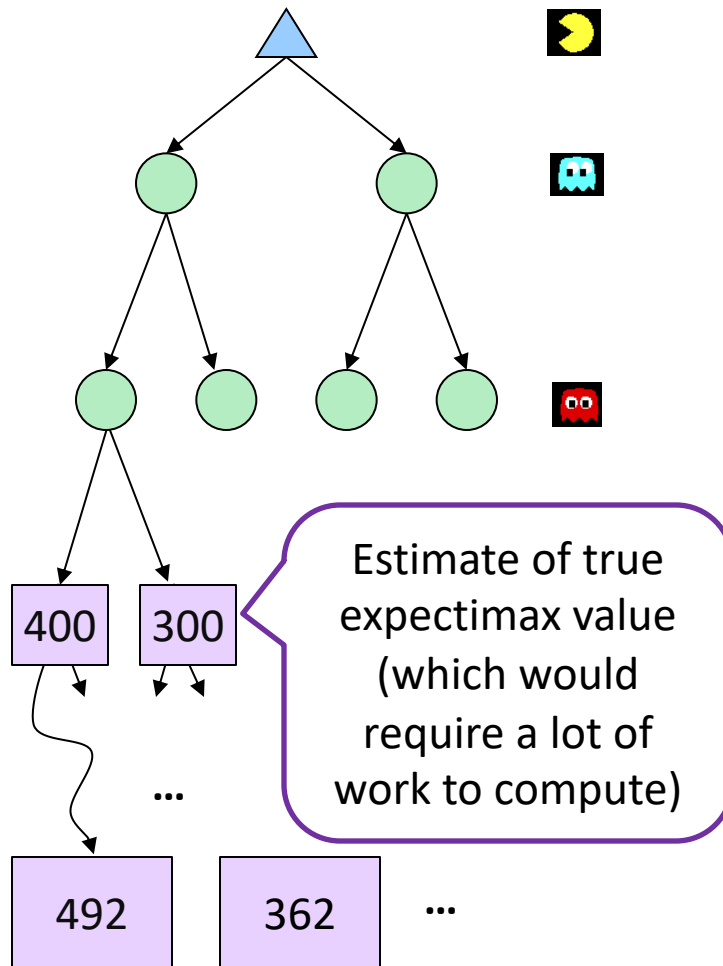
Expectimax Example
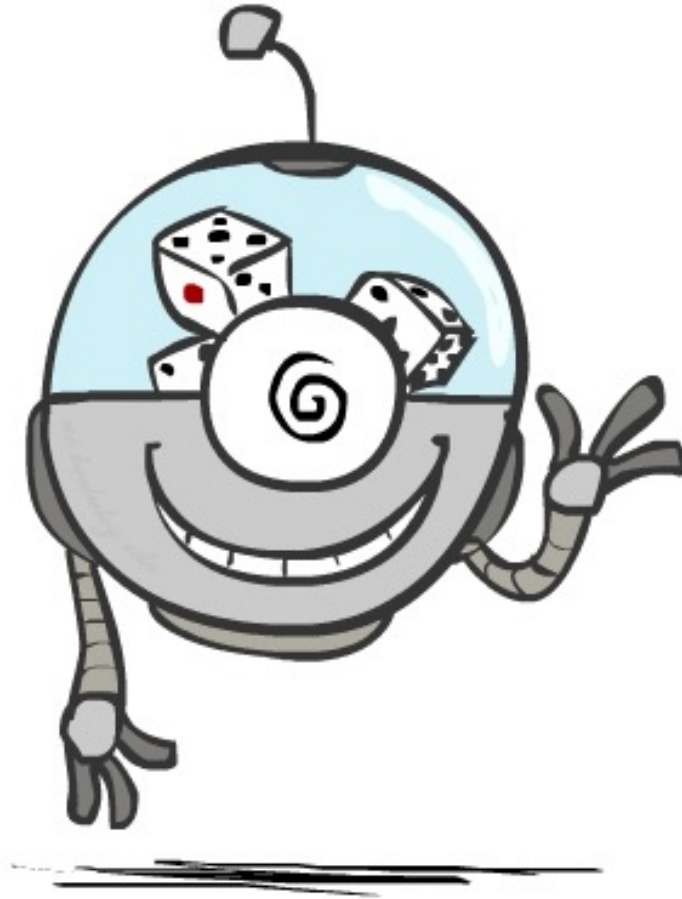
Expectimax Pruning?

Depth-Limited Expectimax

400 300

Estimate of true
expectimax value
(which would
require a lot of
work to compute)

...

492   362   ...

Probabilities

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) = 0.50, 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
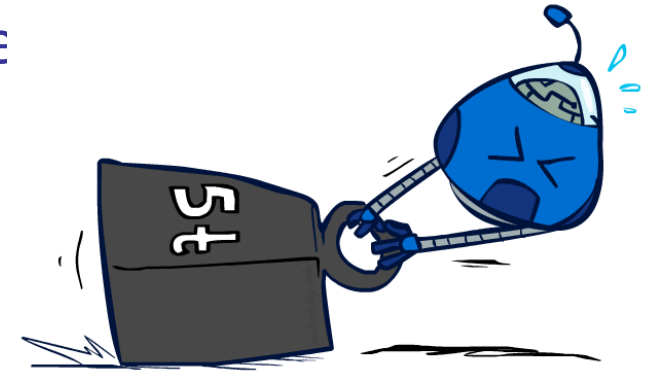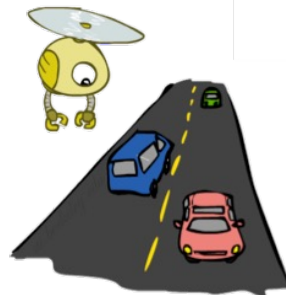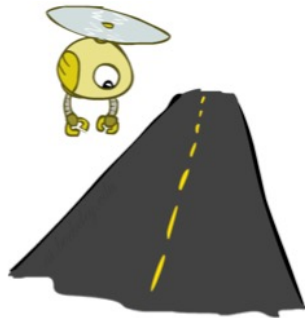
0.25

0.50

0.25

Reminder: Expectations

- The expected value of a function of a random variable is the average, weighted by the probability distribution over outcomes
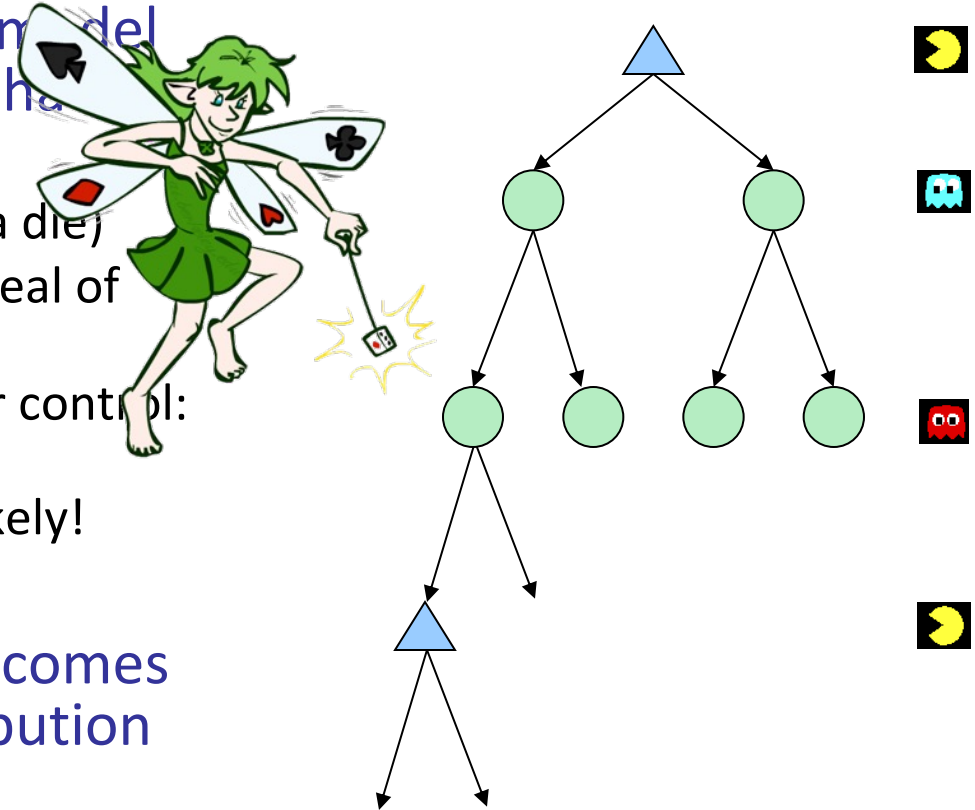
- Example: How long to get to the airport?

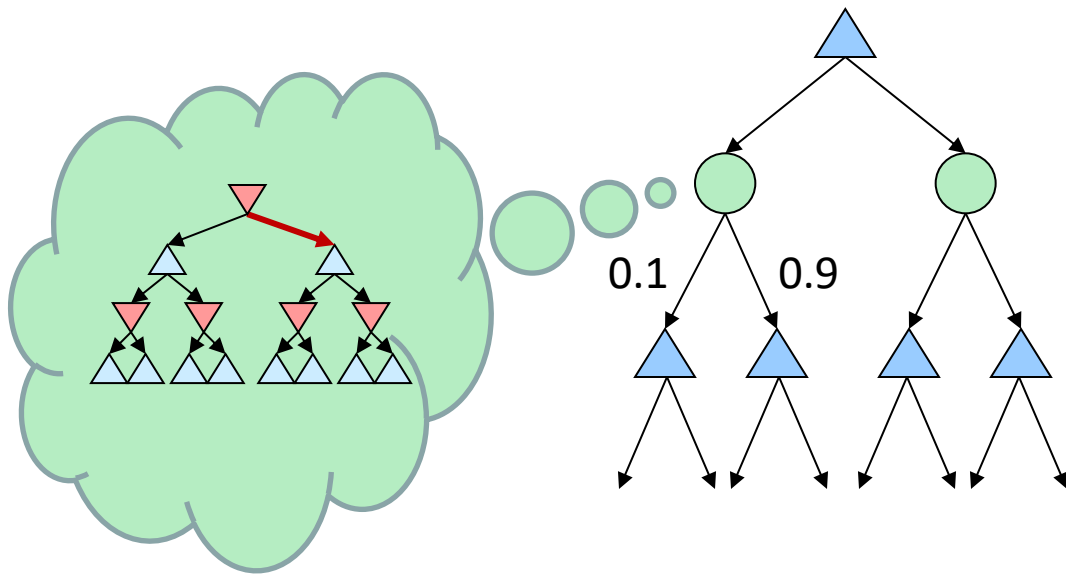| Time:	20 min		30 min		60 min		
	x	+	x	+	x	→	35 min
Probability:	0.25		0.50		0.25		

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!
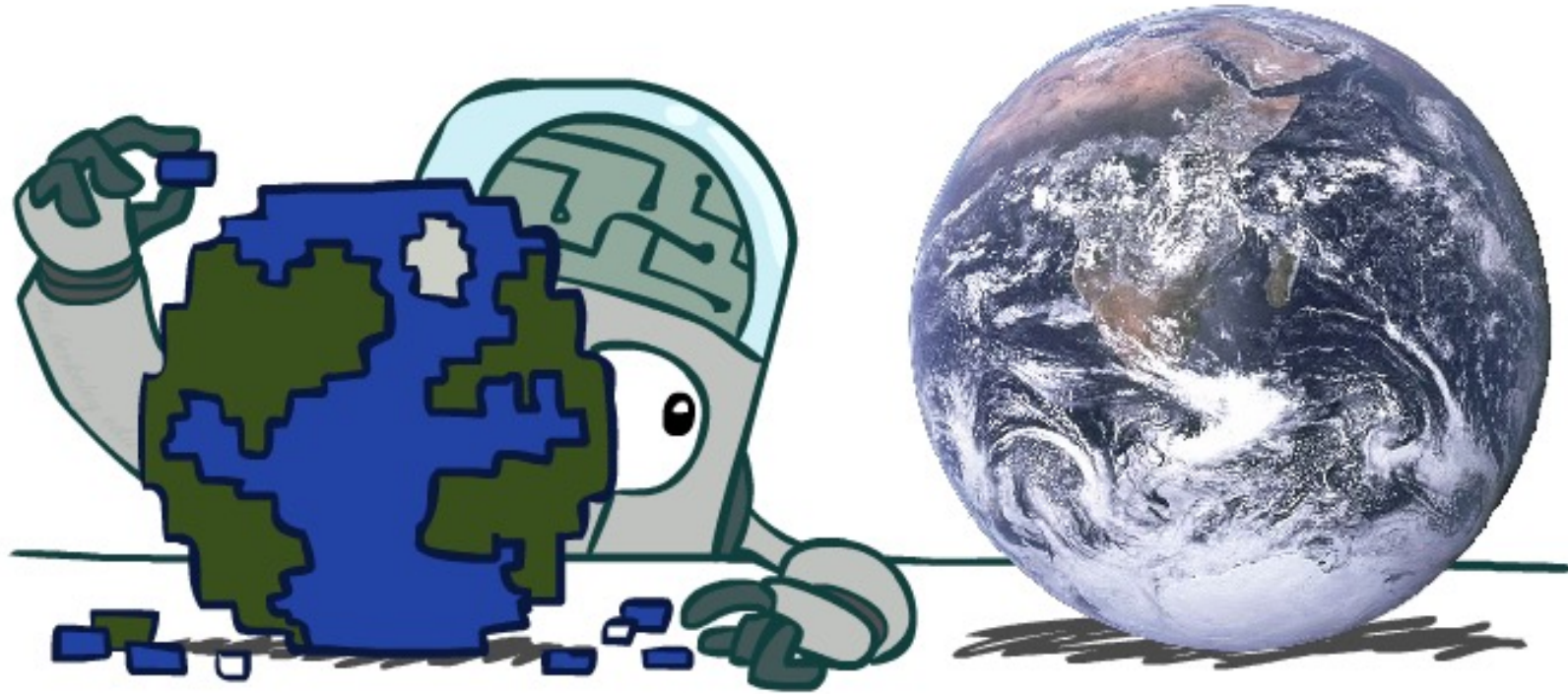
Quiz: Informed Probabilities

- 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

Modeling Assumptions

The Dangers of Optimism and Pessimism

Dangerous Optimism
Assuming chance when the world is adversarial

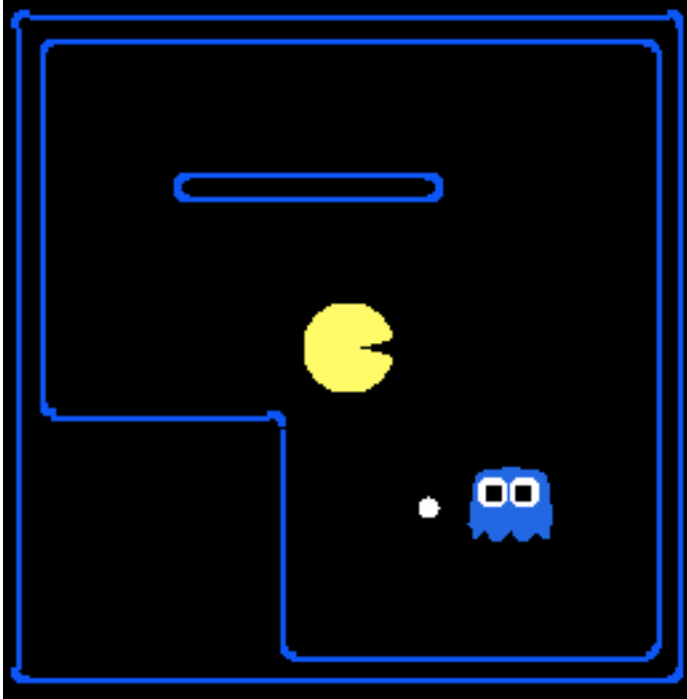### Dangerous Pessimism
Assuming the worst case when it's not likely

Video of Demo Minimax vs Expectimax (Min)

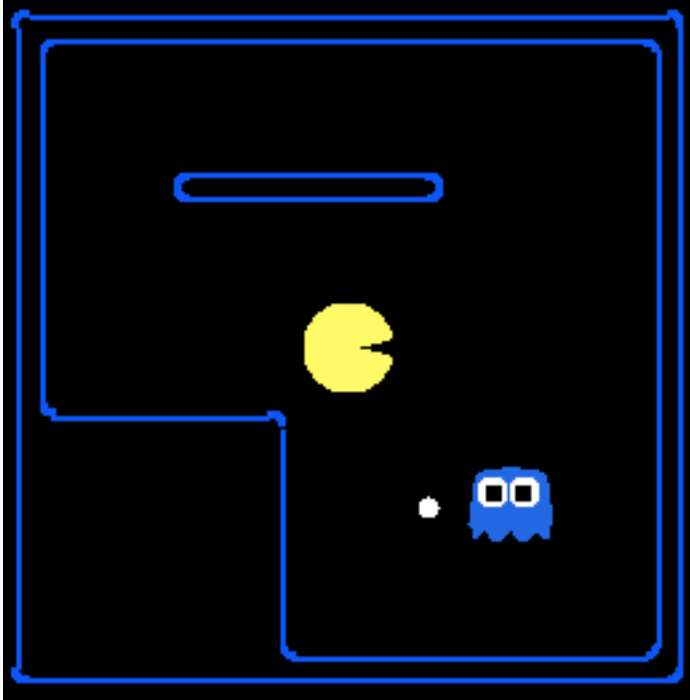# Video of Demo Minimax vs Expectimax (Exp)

Assumptions vs. Reality

	Adversarial Ghost	Random Ghost
Minimax Pacman		
Expectimax Pacman		

Results from playing 5 games

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

[Demos: world assumptions (L7D3,4,5,6)]

Assumptions vs. Reality

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

Results from playing 5 games

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

[Demos: world assumptions (L7D3,4,5,6)]

Video of Demo World Assumptions
Random Ghost – Expectimax Pacman

Video of Demo World Assumptions
Adversarial Ghost – Minimax Pacman

Video of Demo World Assumptions
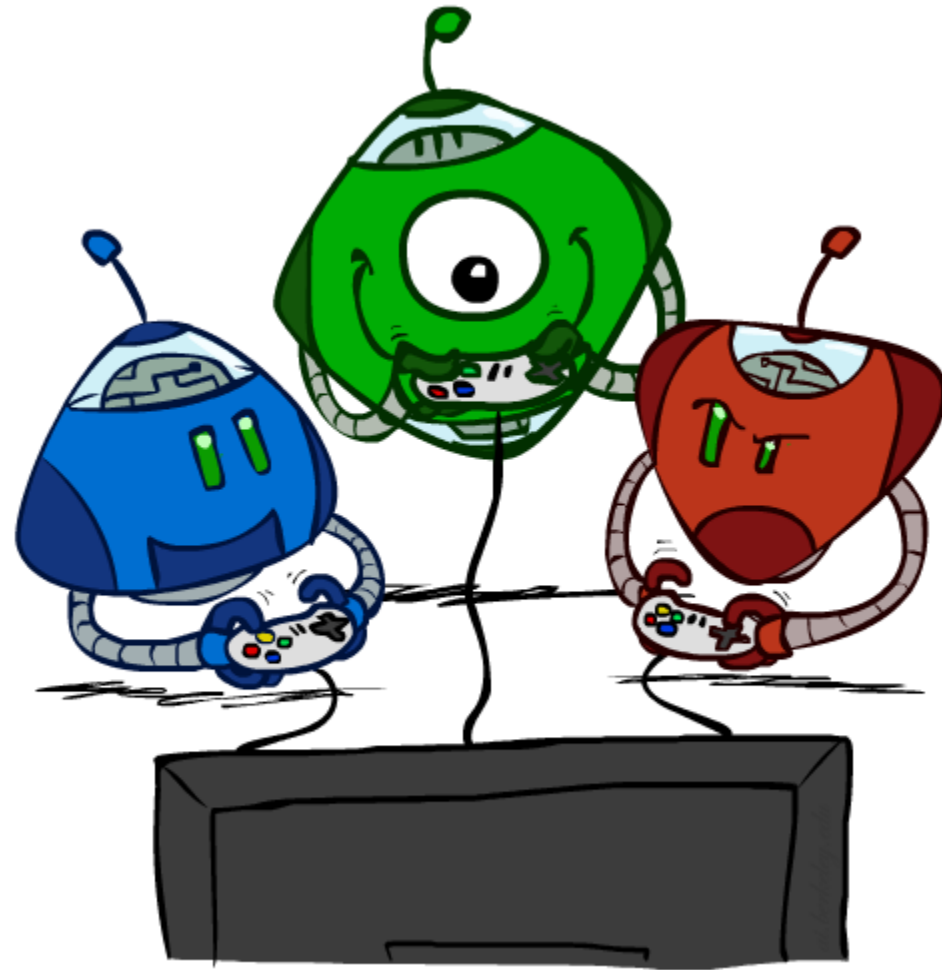# Adversarial Ghost – Expectimax Pacman

Video of Demo World Assumptions
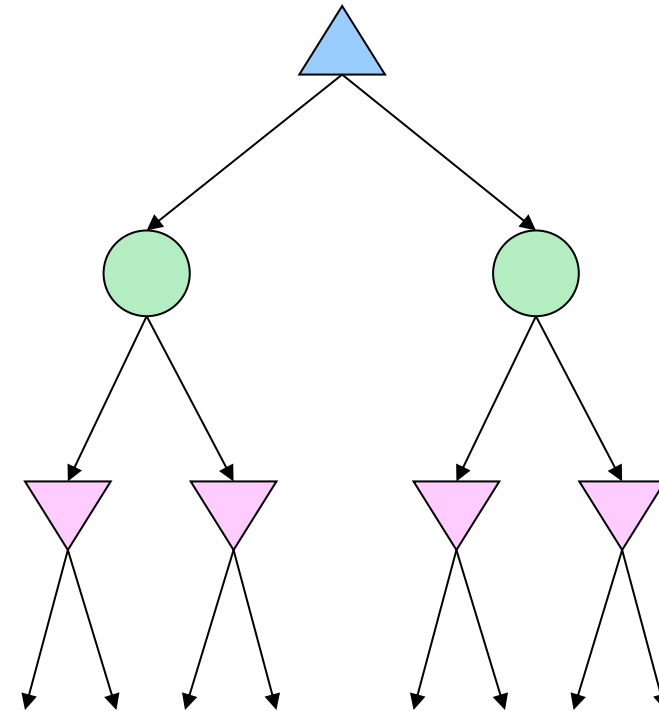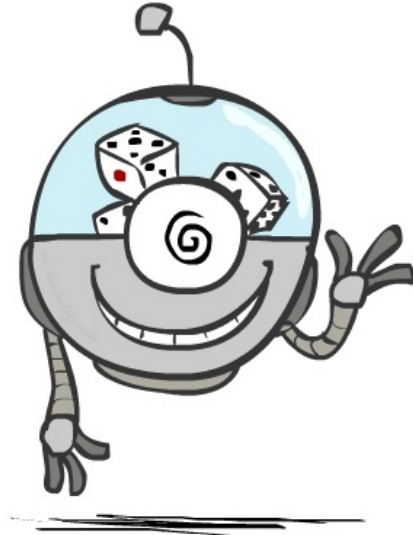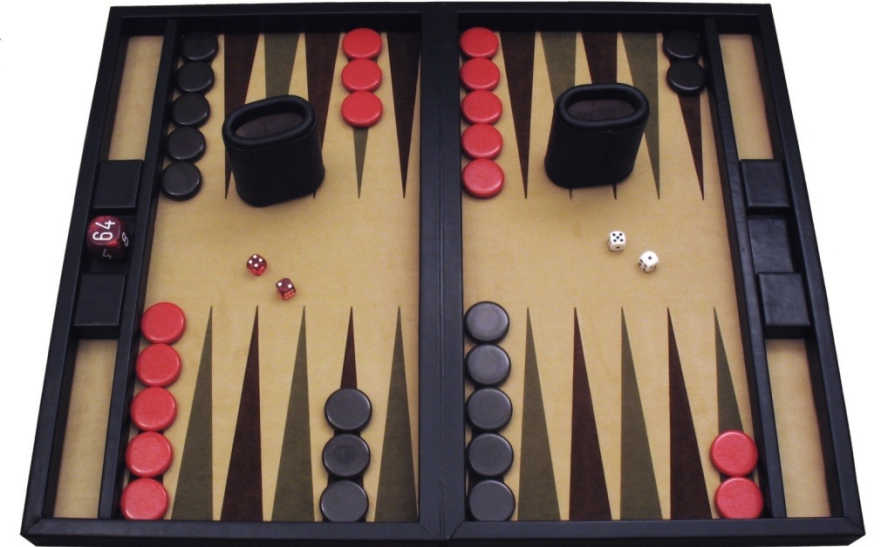# Random Ghost – Minimax Pacman

Other Game Types

Mixed Layer Types

- E.g. Backgammon
- Expectiminimax
  - Environment is an extra "random agent" player that moves after each min/max agent
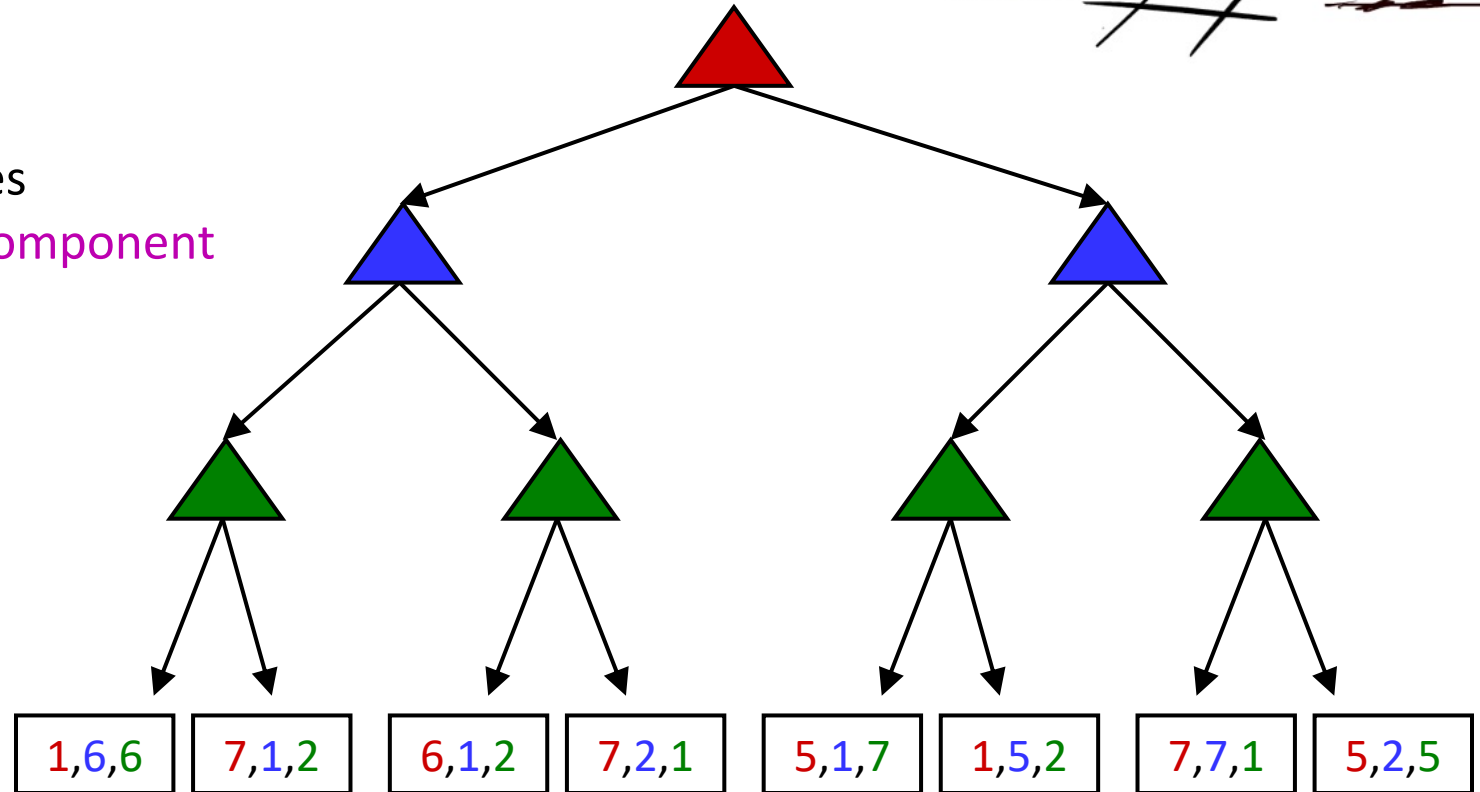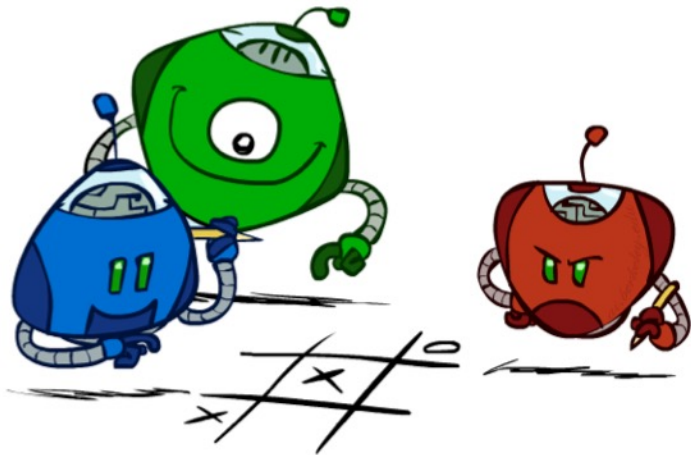 - Each node computes the appropriate combination of its children

Example: Backgammon

- Dice rolls increase b: 21 possible rolls with 2 dice
  - Backgammon $\approx$ 20 legal moves
  - Depth 2 = 20 x (21 x 20)^3 = 1.2 x 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...

- Historic AI: TDGammon uses depth-2 search + very good evaluation function + reinforcement learning: world-champion level play

- 1st AI world champion* in any game!

Image: Wikipedia

Multi-Agent Utilities

- **What if the game is not zero-sum, or has multiple players?**

- **Generalization of minimax:**
  - Terminals have utility tuples
  - Node values are also utility tuples
  - Each player maximizes its own component
  - Can give rise to cooperation and competition dynamically…

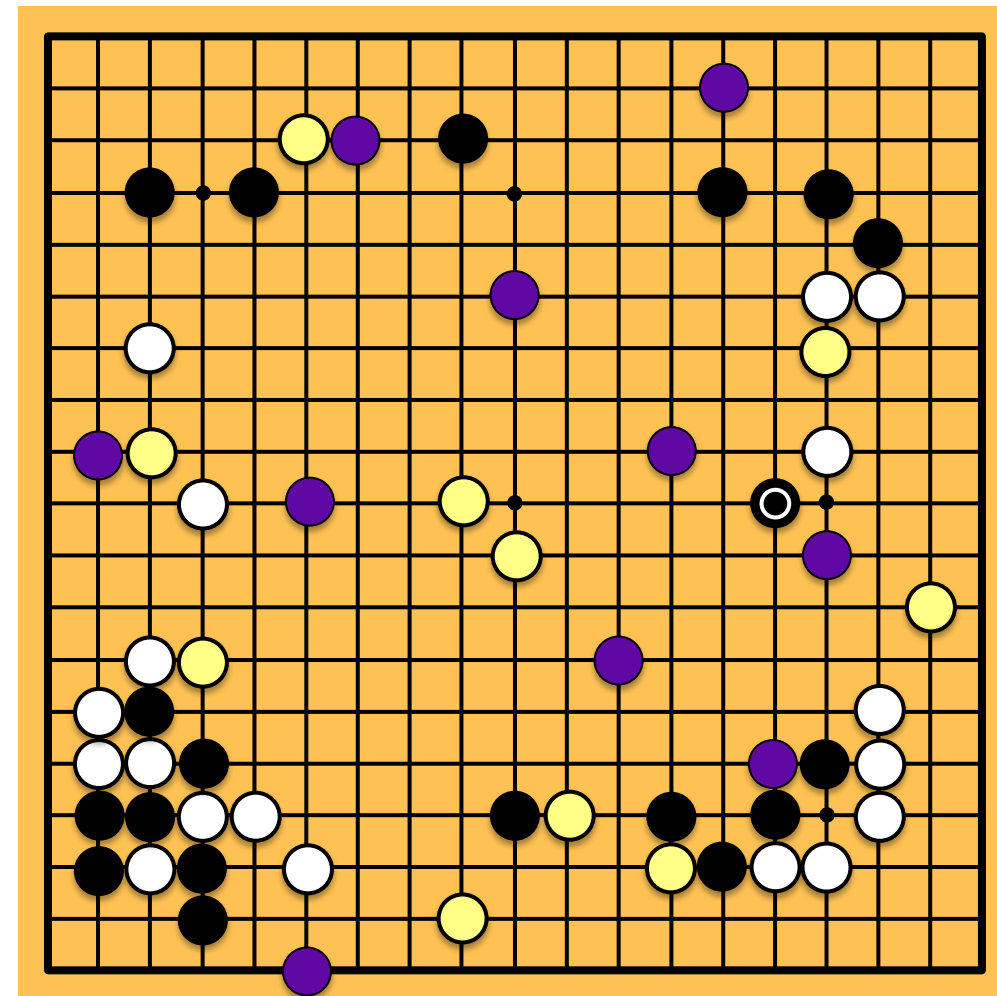| 1,6,6 | 7,1,2 | 6,1,2 | 7,2,1 | 5,1,7 | 1,5,2 | 7,7,1 | 5,2,5 |

Monte Carlo Tree Search

Monte Carlo Tree Search

- **Methods based on alpha-beta search assume a fixed horizon**
  - Pretty hopeless for Go, with $b > 300$

- **MCTS combines two important ideas:**

  - ***Evaluation by rollouts*** – play multiple games to termination from a state s (using a simple, fast rollout policy) and count wins and losses

  - ***Selective search*** – explore parts of the tree that will help improve the decision at the root, regardless of depth
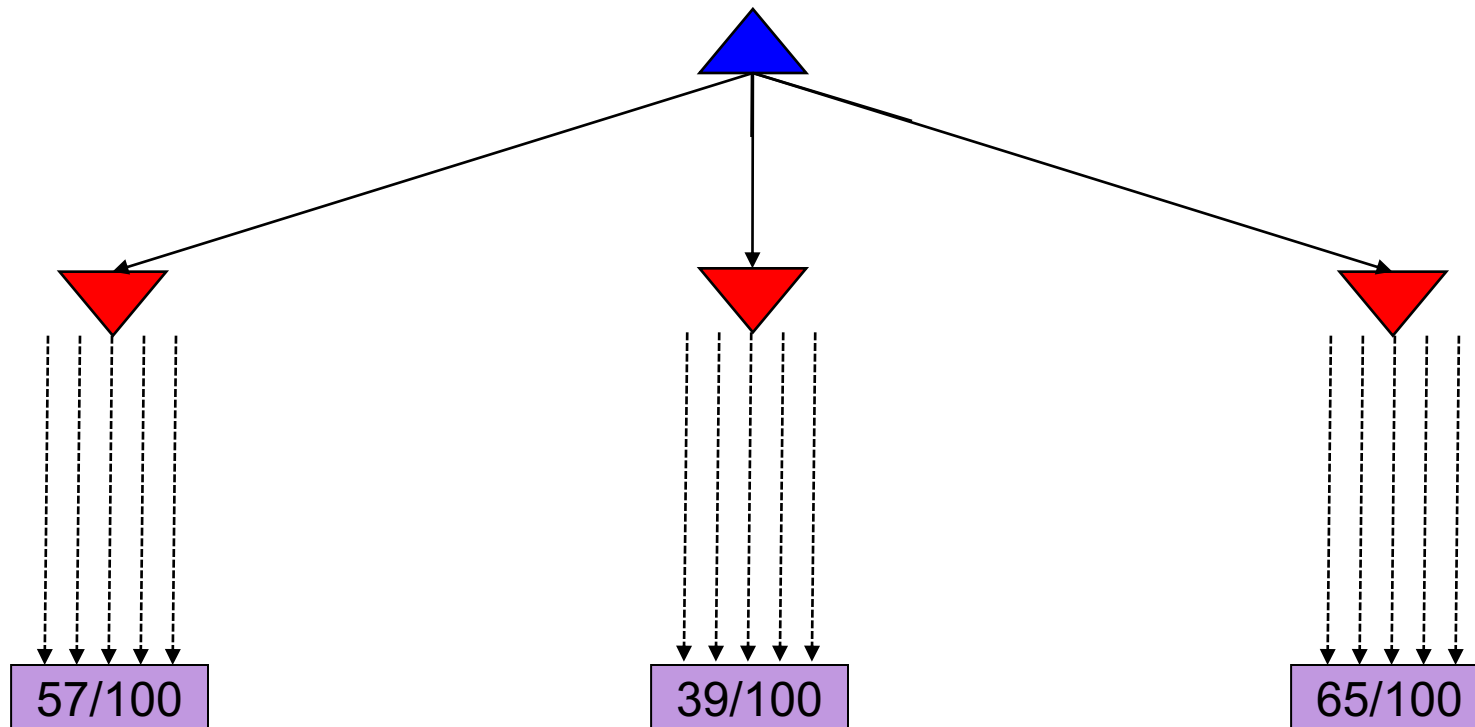
Rollouts

- **For each rollout:**
  - Repeat until terminal:
    - Play a move according to a fixed, fast rollout policy
  - Record the result
- **Fraction of wins correlates with the true value of the position!**
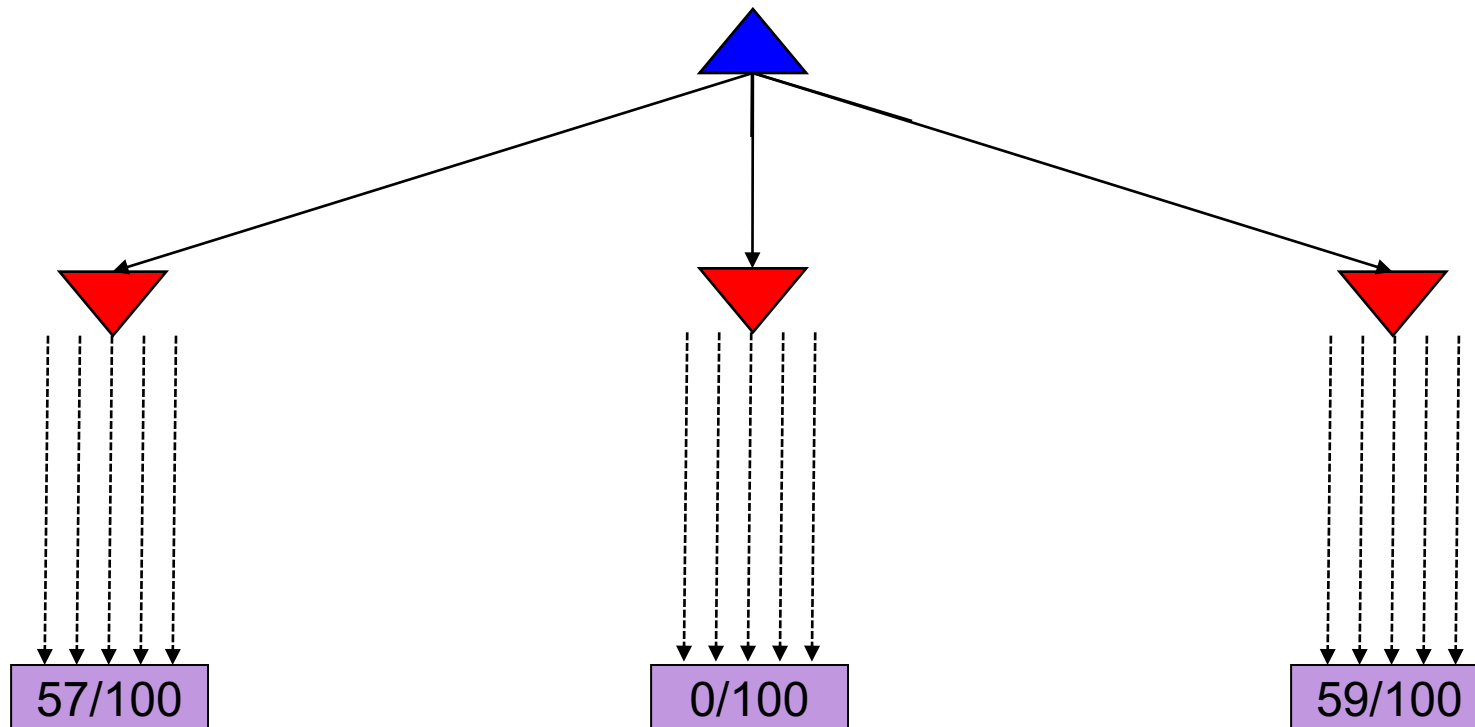- **Having a "better" rollout policy helps**

"Move 37"

MCTS Version 0

- Do *N* rollouts from each child of the root, record fraction of wins
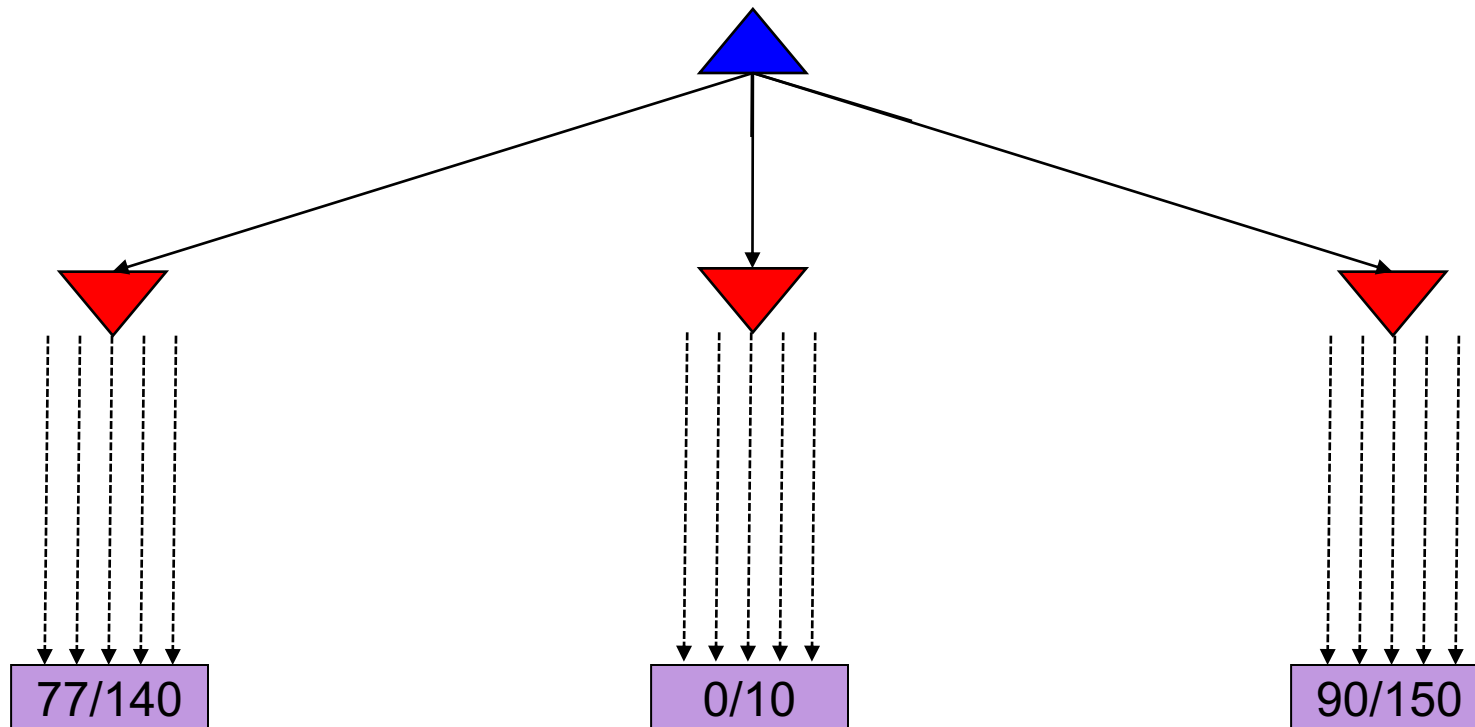- Pick the move that gives the best outcome by this metric

MCTS Version 0

- Do *N* rollouts from each child of the root, record fraction of wins
- Pick the move that gives the best outcome by this metric
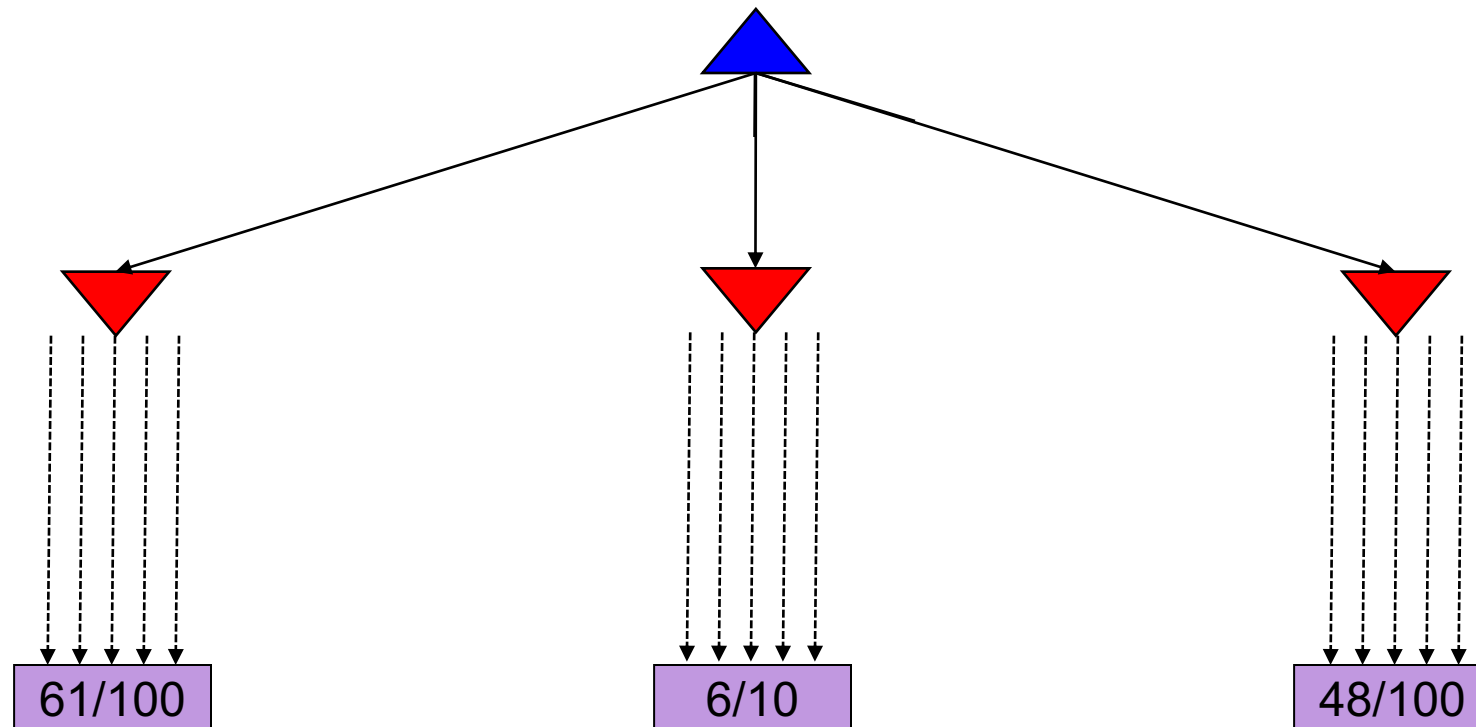
MCTS Version 0.9

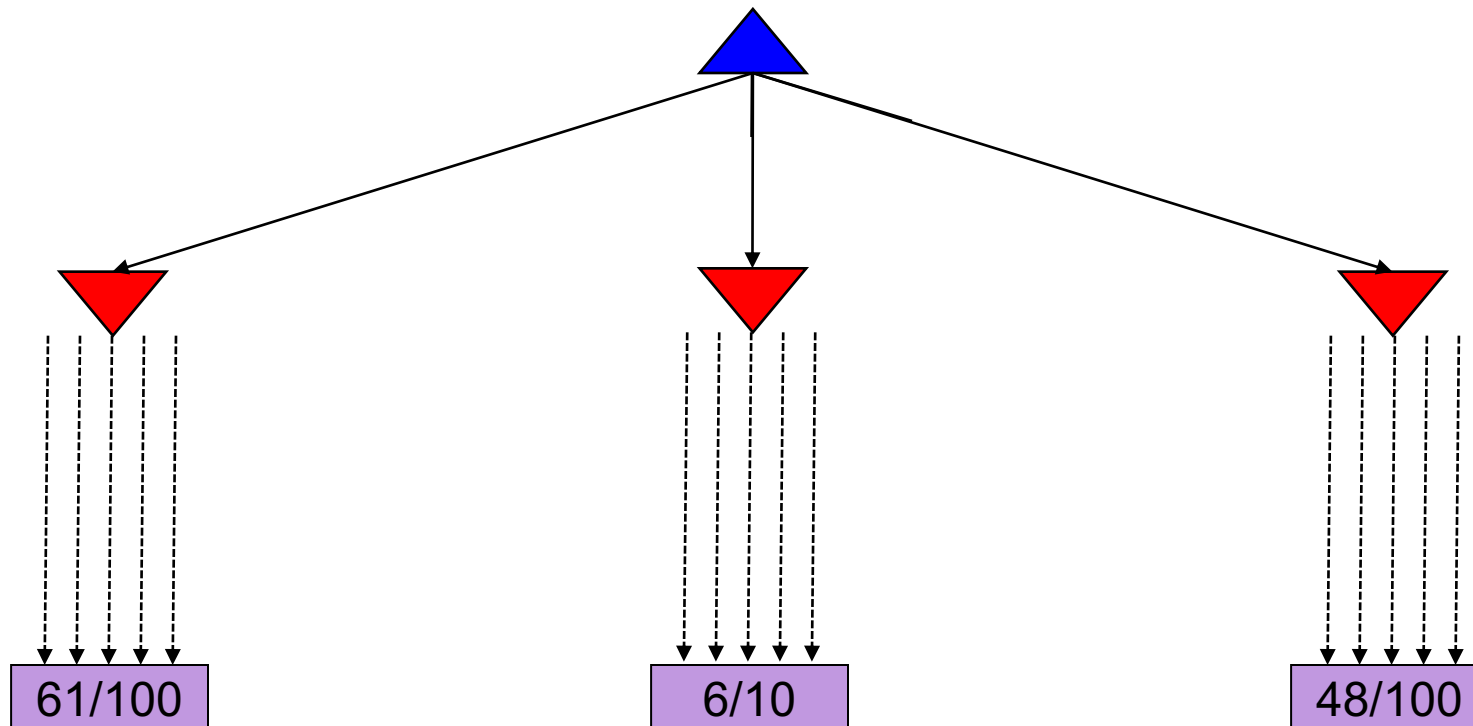- Allocate rollouts to more promising nodes

■ Allocate rollouts to more promising nodes

MCTS Version 1.0

- Allocate rollouts to more promising nodes
- Allocate rollouts to more uncertain nodes

UCB heuristics

■ UCB1 formula combines "promising" and "uncertain":

$$UCB1(n) = \frac{U(n)}{N(n)} + C \times \sqrt{\frac{\log N(\text{PARENT}(n))}{N(n)}}$$

pretty good heuristic function
when n gets really large, log(parent(n))/n -> 0
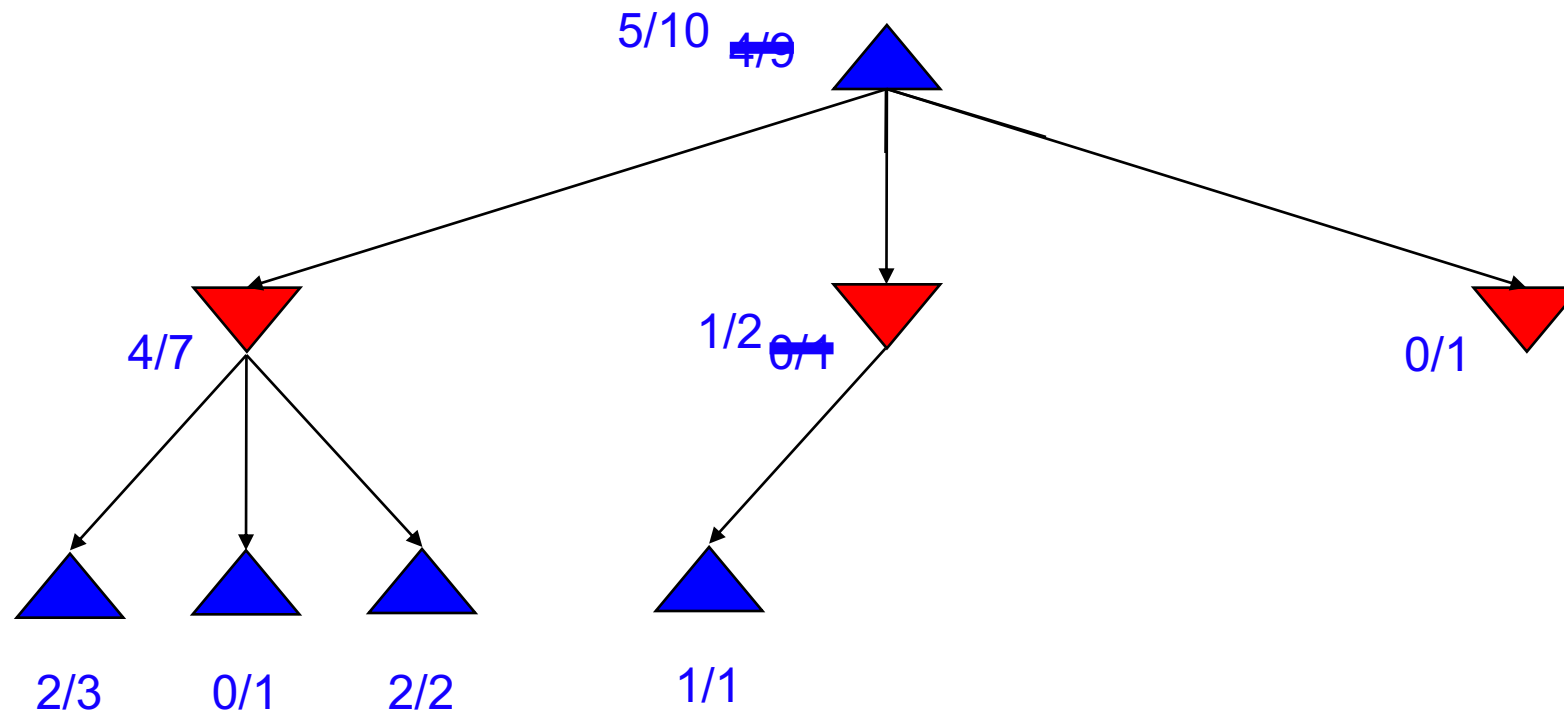we only care about the posibility th win

but when we donot have enough try
we have an ' bonus ' for giving new rollouts

■ $N(n)$ = number of rollouts from node n

■ $U(n)$ = total utility of rollouts (e.g., # wins) for Player(Parent(n))

■ A provably not terrible heuristic for ***bandit problems***

   ■ (which are not the same as the problem we face here!)

MCTS Version 2.0: UCT

- **Repeat until out of time:**
    - Given the current search tree, recursively apply UCB to choose a path down to a leaf (not fully expanded) node *n*
    - Add a new child *c* to *n* and run a rollout from *c*
    - Update the win counts from *c* back up to the root
- **Choose the action leading to the child with highest *N***

UCT Example

Why is there no min or max?

- "Value" of a node, $U(n)/N(n)$, is a weighted **sum** of child values!
- Idea: as $N \rightarrow \infty$, the vast majority of rollouts are concentrated in the best child(ren), so weighted average $\rightarrow$ max/min
- Theorem: as $N \rightarrow \infty$ UCT selects the minimax move
  - (but N never approaches infinity!)

So it's a bit more optimistic than minimax
and saving the computing resources

Summary

- **Games require decisions when optimality is impossible**
  - Bounded-depth search and approximate evaluation functions
- **Games force efficient use of computation**
  - Alpha-beta pruning, MCTS
- **Game playing has produced important research ideas**
  - Reinforcement learning (checkers)
  - Iterative deepening (chess)
  - Rational metareasoning (Othello)
  - Monte Carlo tree search (chess, Go)
  - Solution methods for partial-information games in economics (poker)
- **Video games present much greater challenges – lots to do!**
  - $b = 10^{500}$, $|S| = 10^{4000}$, $m = 10{,}000$, partially observable, often > 2 players

Next Time: Logic!