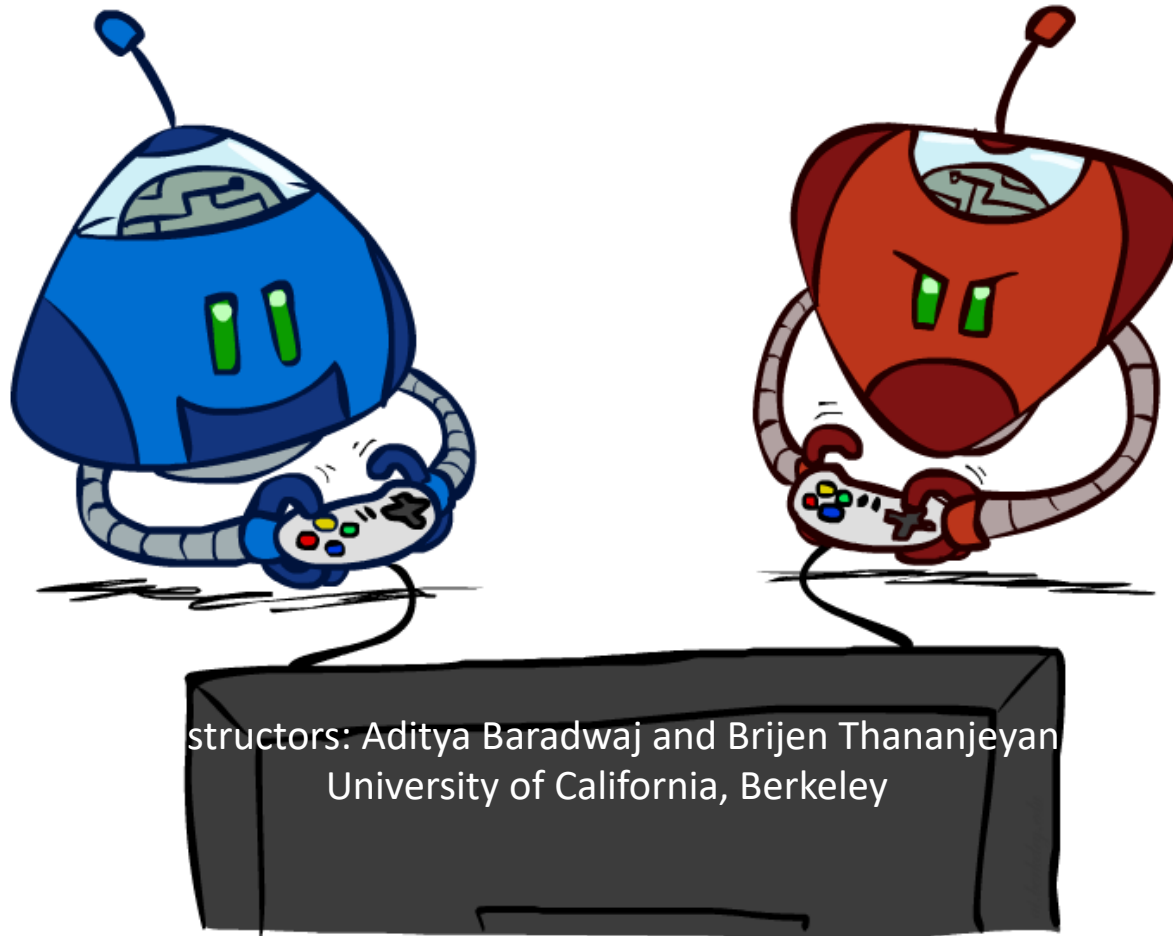


Announcements

- Homework 1
 - Due **tomorrow 6/28** at 11:59pm
 - Electronic HW1
 - Written HW1
- Project 1
 - Due Tuesday 7/2 at 4:00pm

CS 188: Artificial Intelligence

Adversarial Search and Game Trees

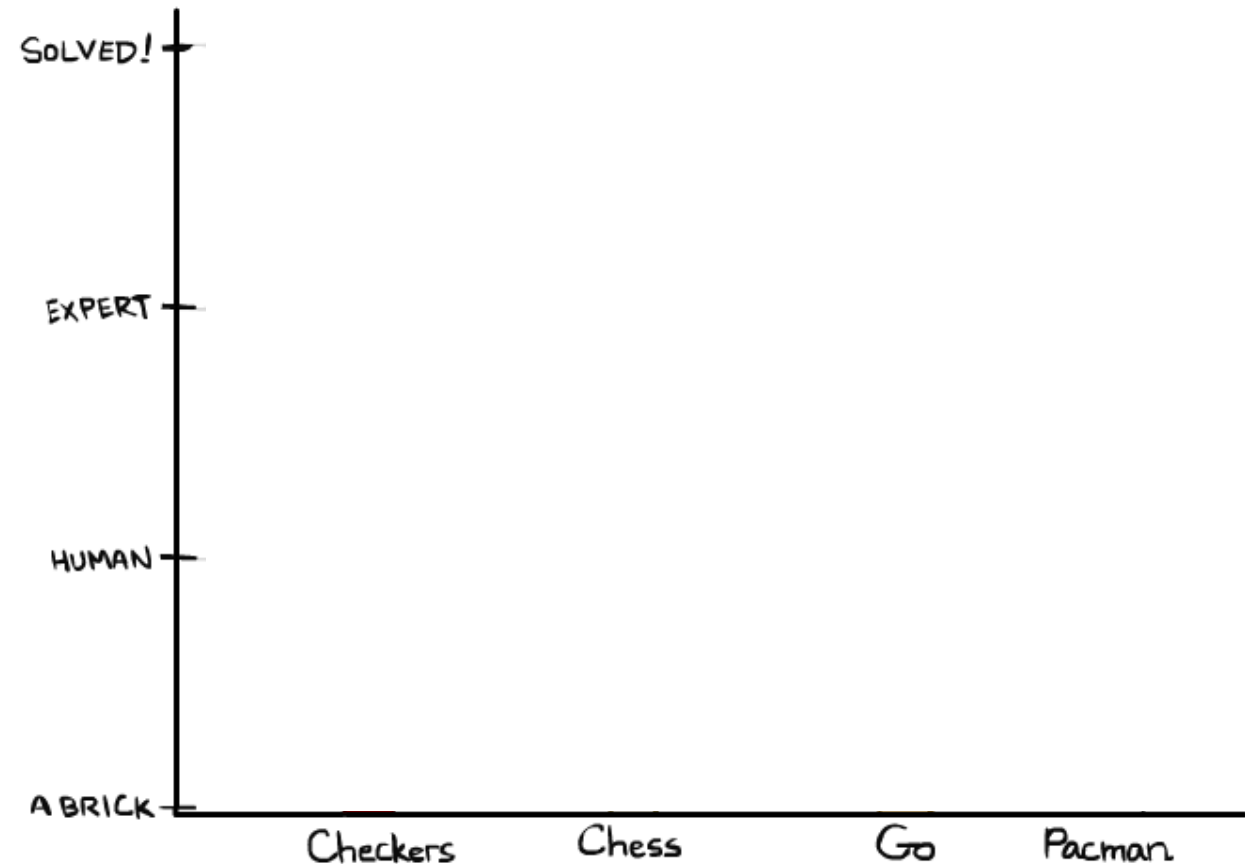


Instructors: Aditya Baradwaj and Brijen Thananjeyan
University of California, Berkeley

[Slides adapted from Dan Klein and Pieter Abbeel (ai.berkeley.edu).]

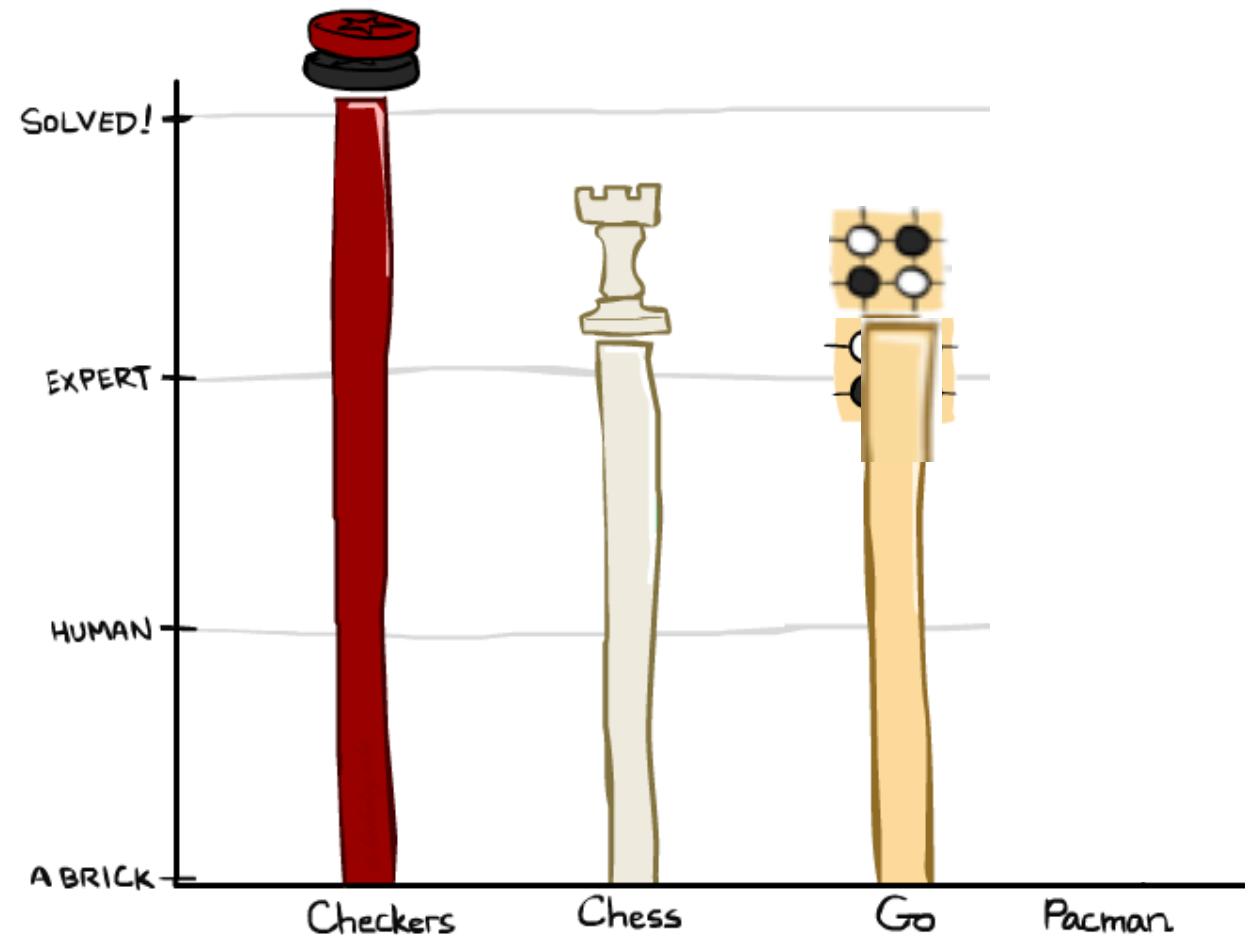
Game Playing State-of-the-Art

- **Checkers:** 1950: First computer player. 1994: First computer champion: Chinook ended 40-year-reign of human champion Marion Tinsley using complete 8-piece endgame. 2007: Checkers solved!
- **Chess:** 1997: Deep Blue defeats human champion Gary Kasparov in a six-game match. Deep Blue examined 200M 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.
- **Go:** Human champions are now starting to be challenged by machines. In go, $b > 300$! Classic programs use pattern knowledge bases, but big recent advances use Monte Carlo (randomized) expansion methods.



Game Playing State-of-the-Art

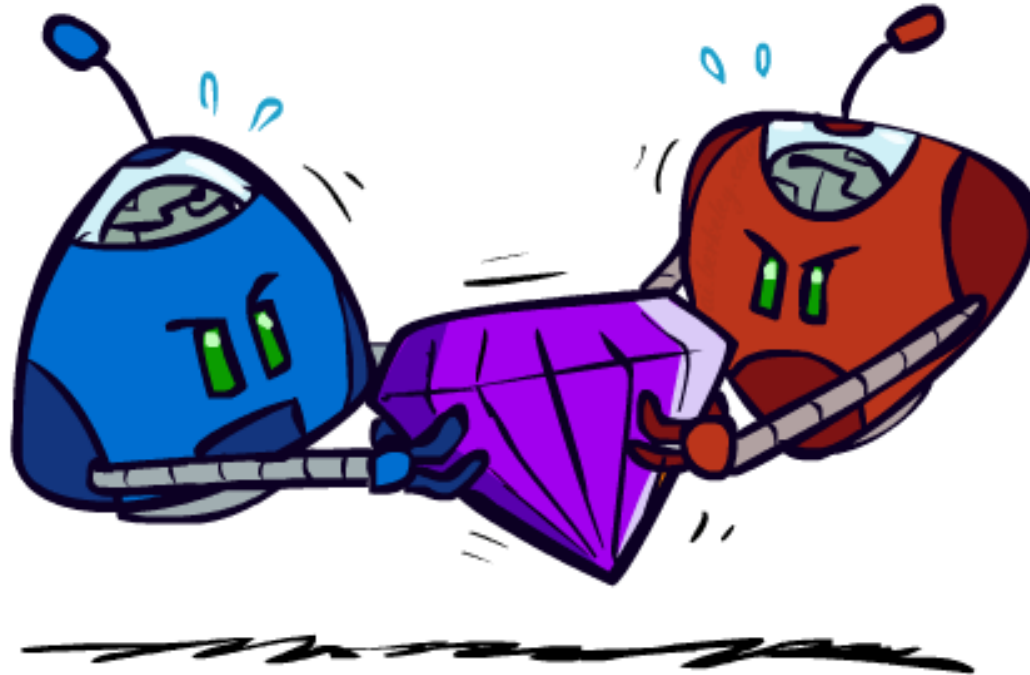
- **Checkers:** 1950: First computer player. 1994: First computer champion: Chinook ended 40-year-reign of human champion Marion Tinsley using complete 8-piece endgame. 2007: Checkers solved!
- **Chess:** 1997: Deep Blue defeats human champion Gary Kasparov in a six-game match. Deep Blue examined 200M 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.
- **Go:** 2016: Alpha GO defeats human champion. Uses Monte Carlo Tree Search, learned evaluation function.
- **Pacman**



Video of Demo Mystery Pacman

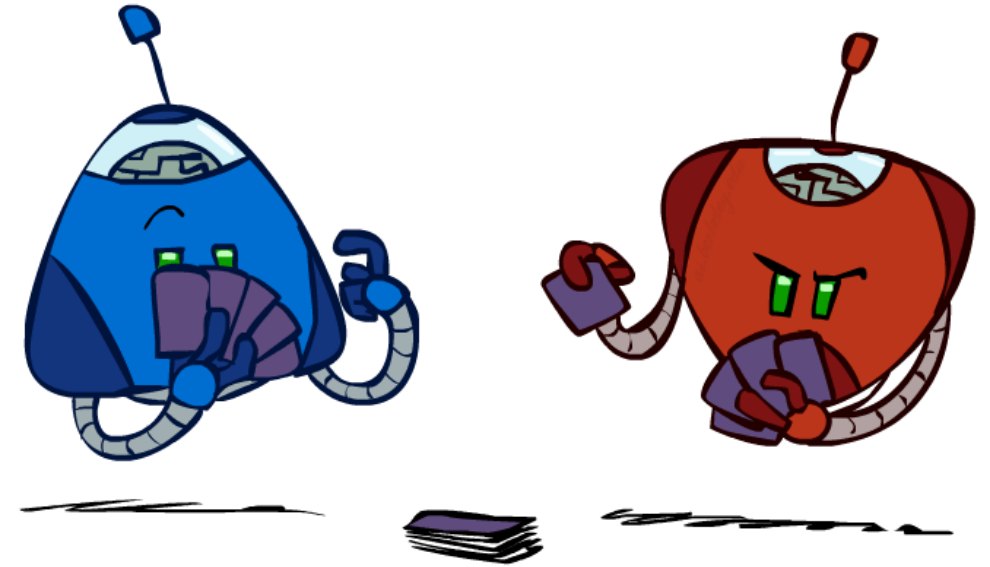


Adversarial Games



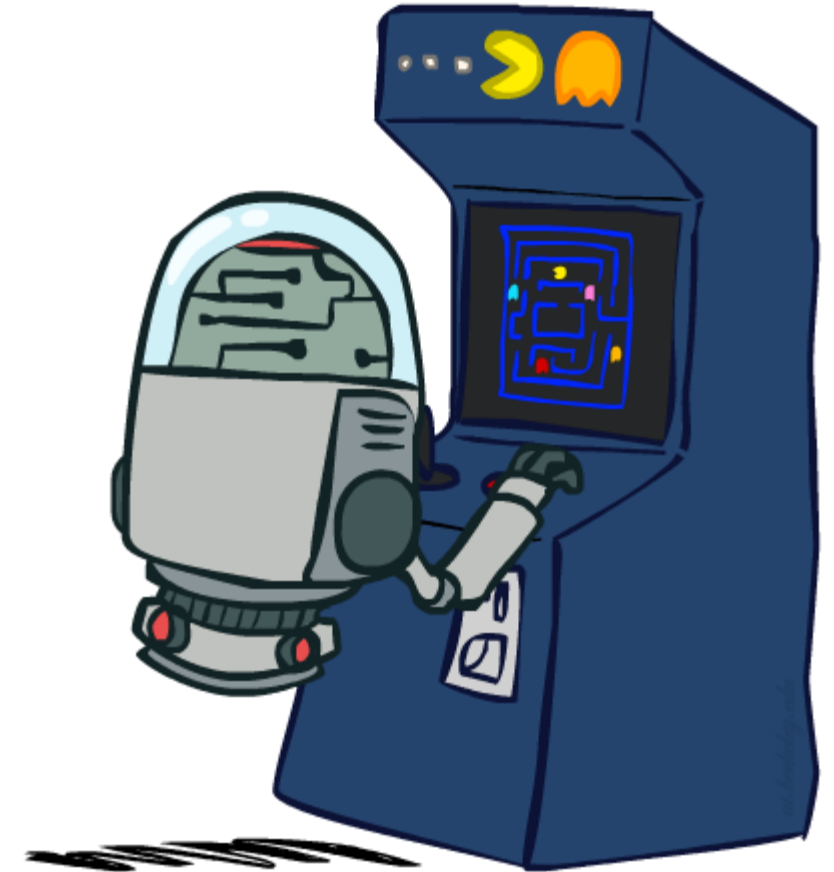
Types of Games

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

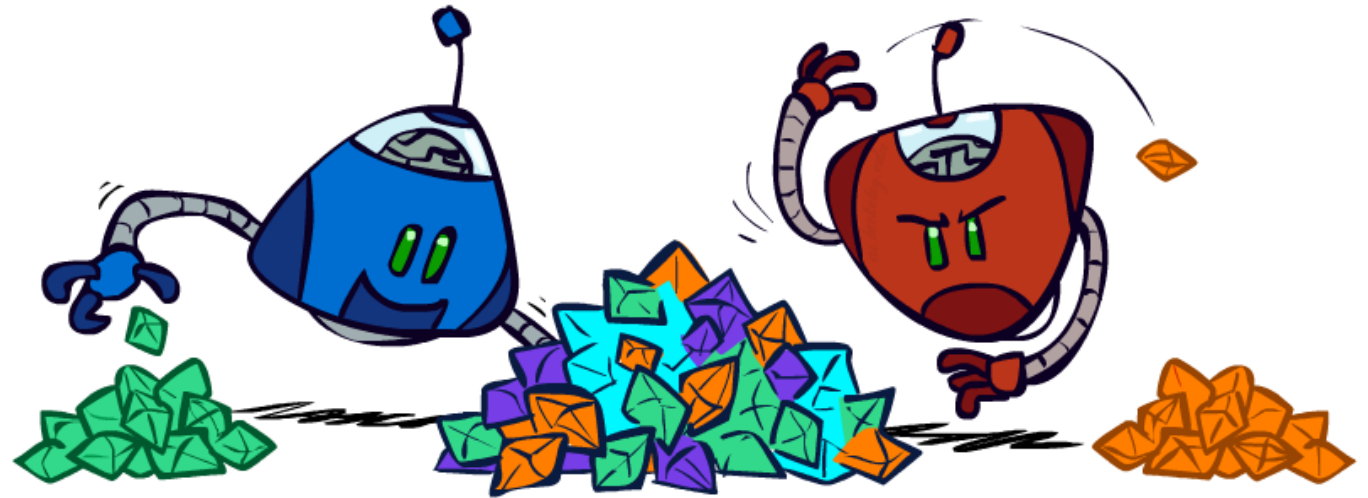
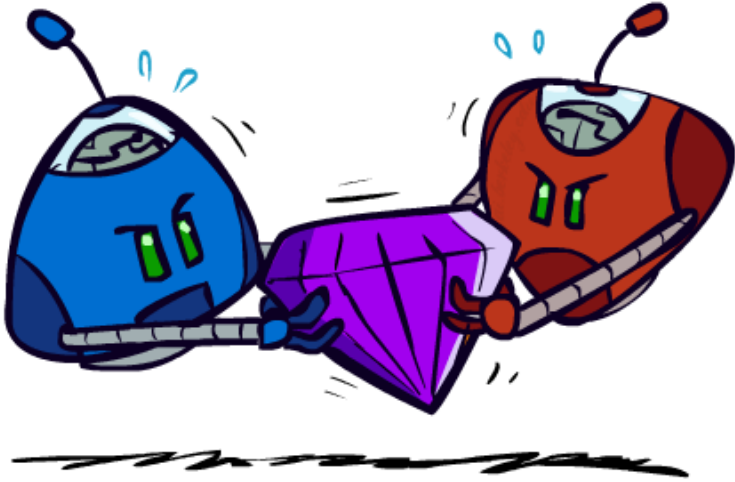


Deterministic Games

- 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**: $S \rightarrow A$



Zero-Sum Games



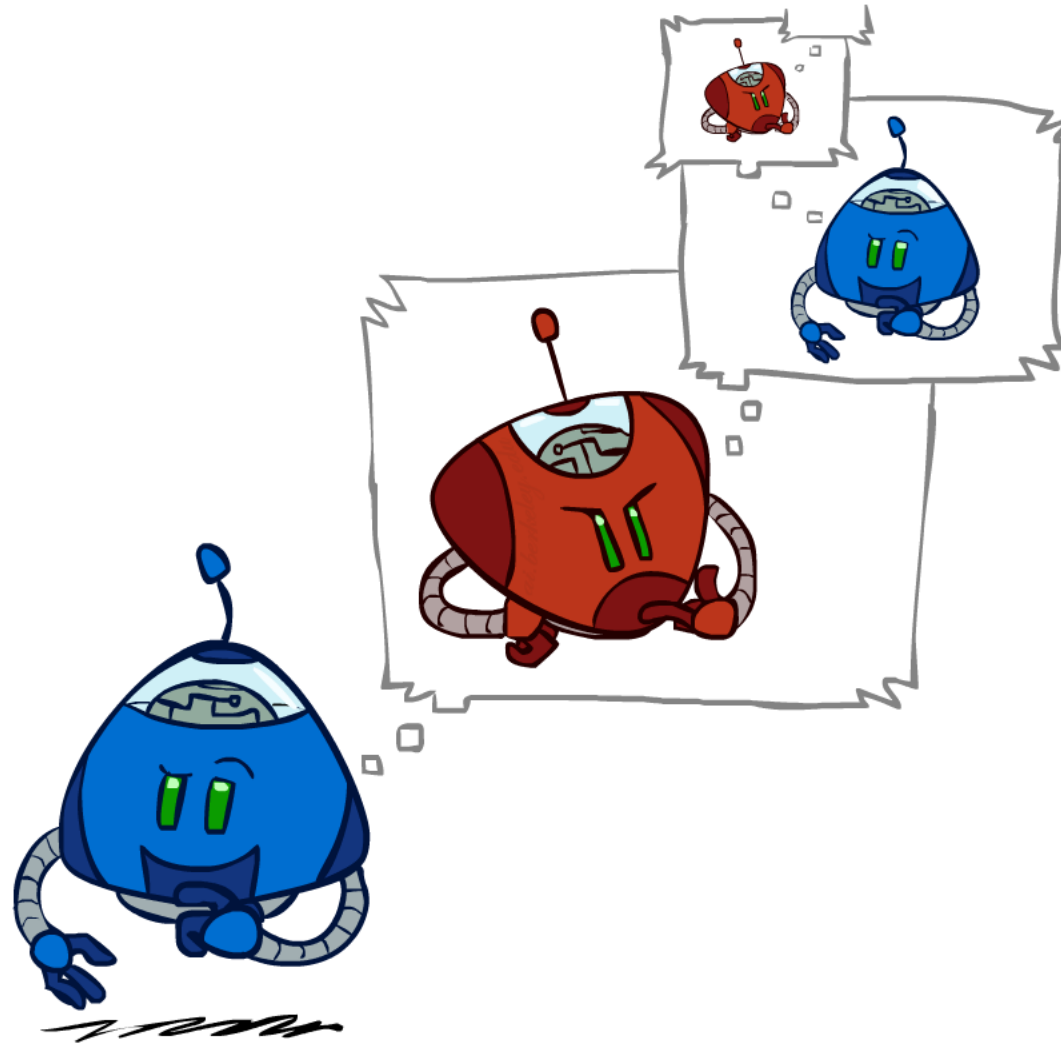
■ Zero-Sum Games

- Agents have opposite utilities (values on outcomes)
- Lets us think of a single value that one maximizes and the other minimizes
- Adversarial, pure competition

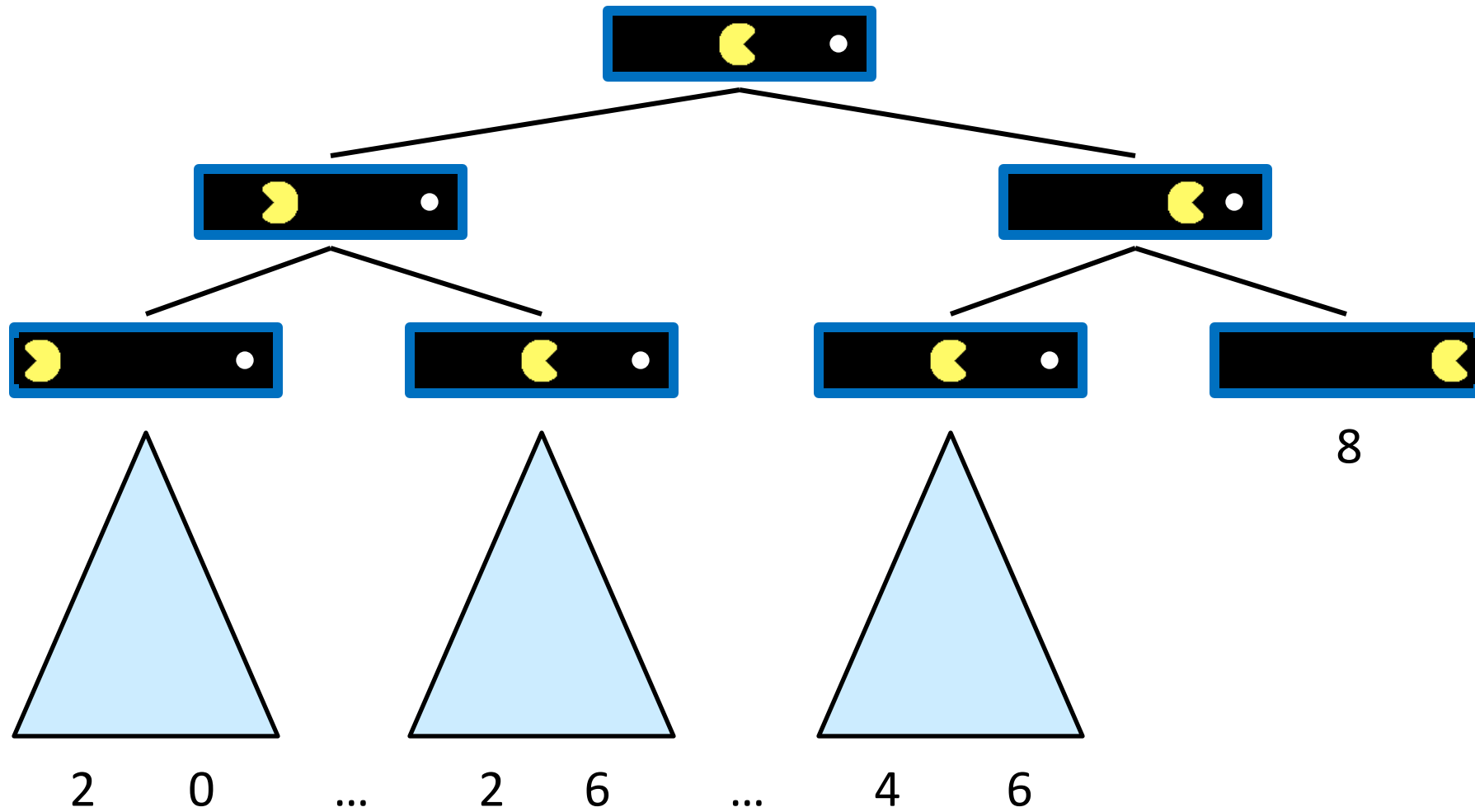
■ General Games

- Agents have independent utilities (values on outcomes)
- Cooperation, indifference, competition, and more are all possible
- More later on non-zero-sum games

Adversarial Search

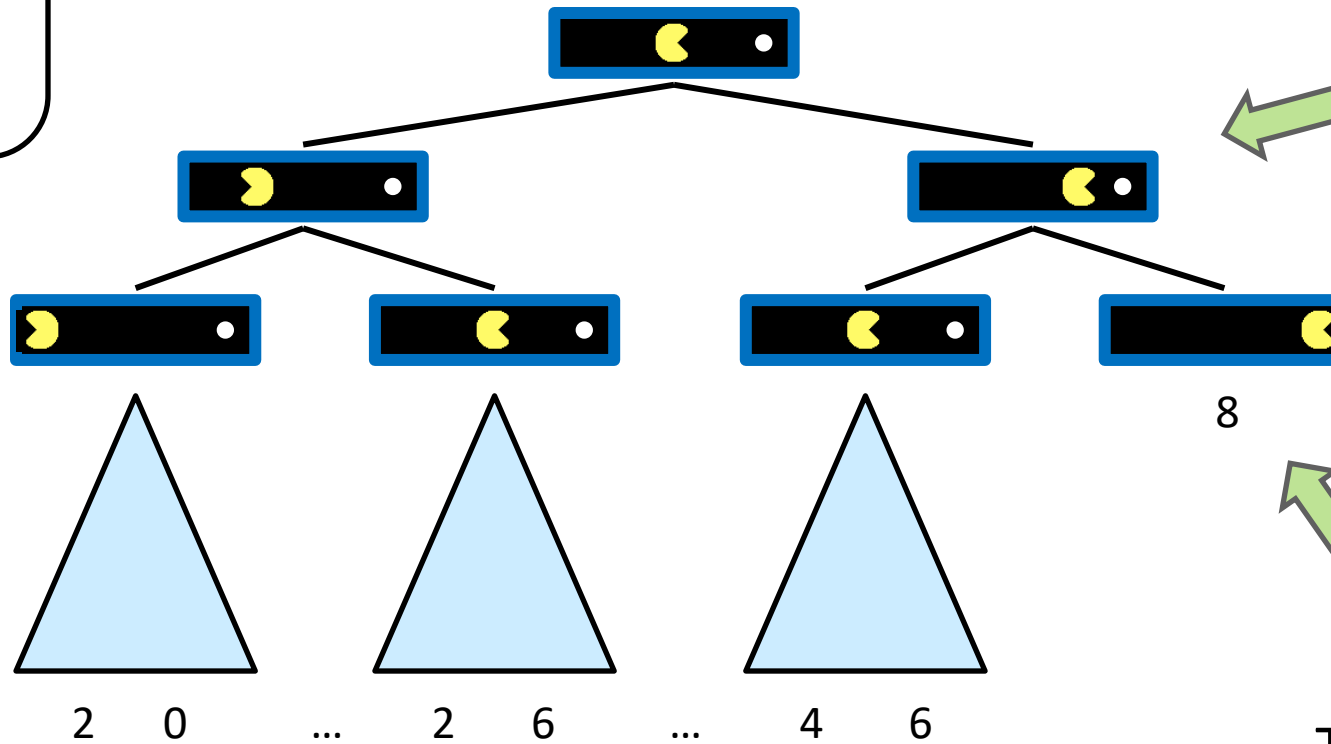


Single-Agent Trees



Value of a State

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



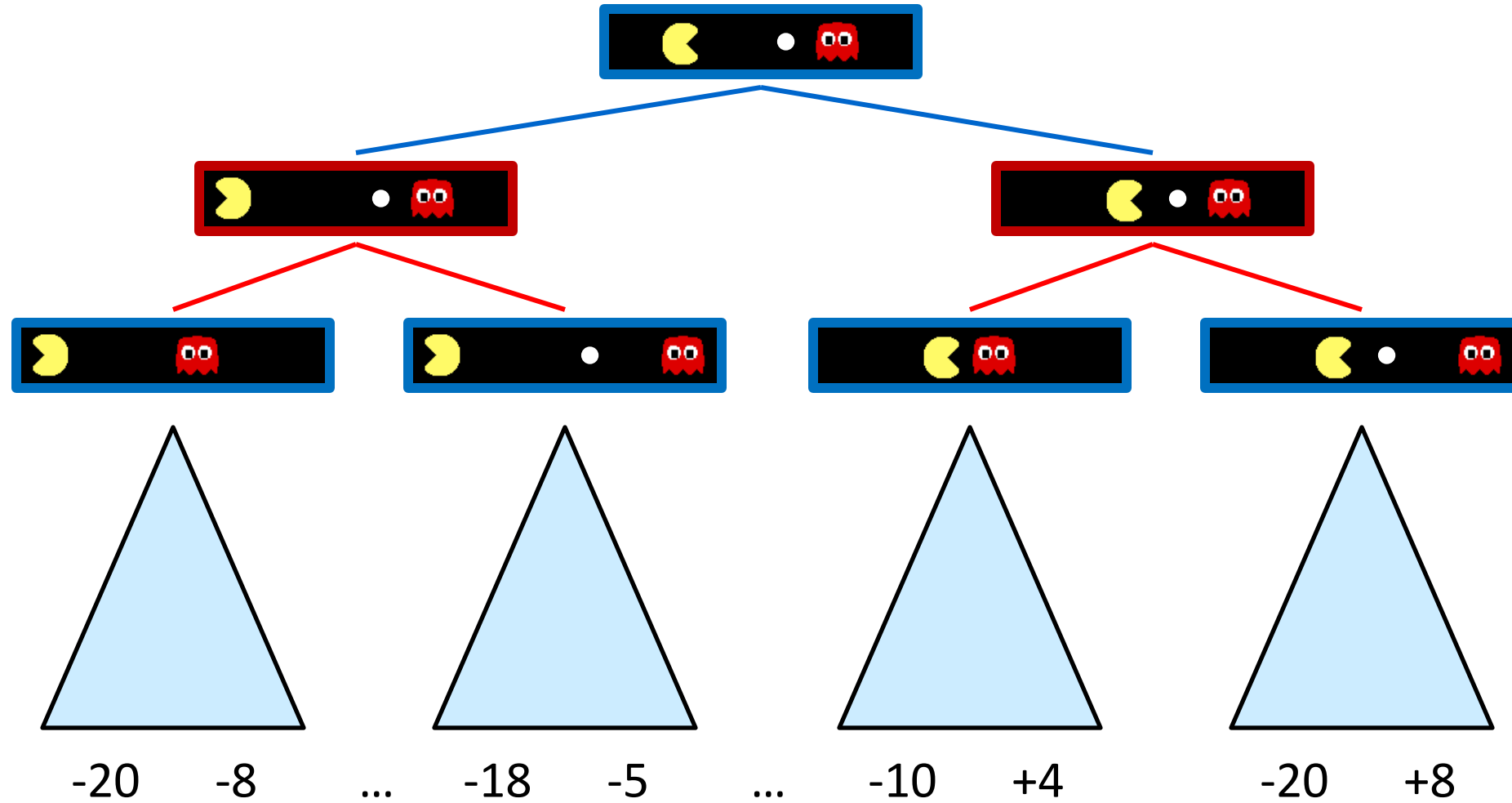
Non-Terminal States:

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

Terminal States:

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

Adversarial Game Trees



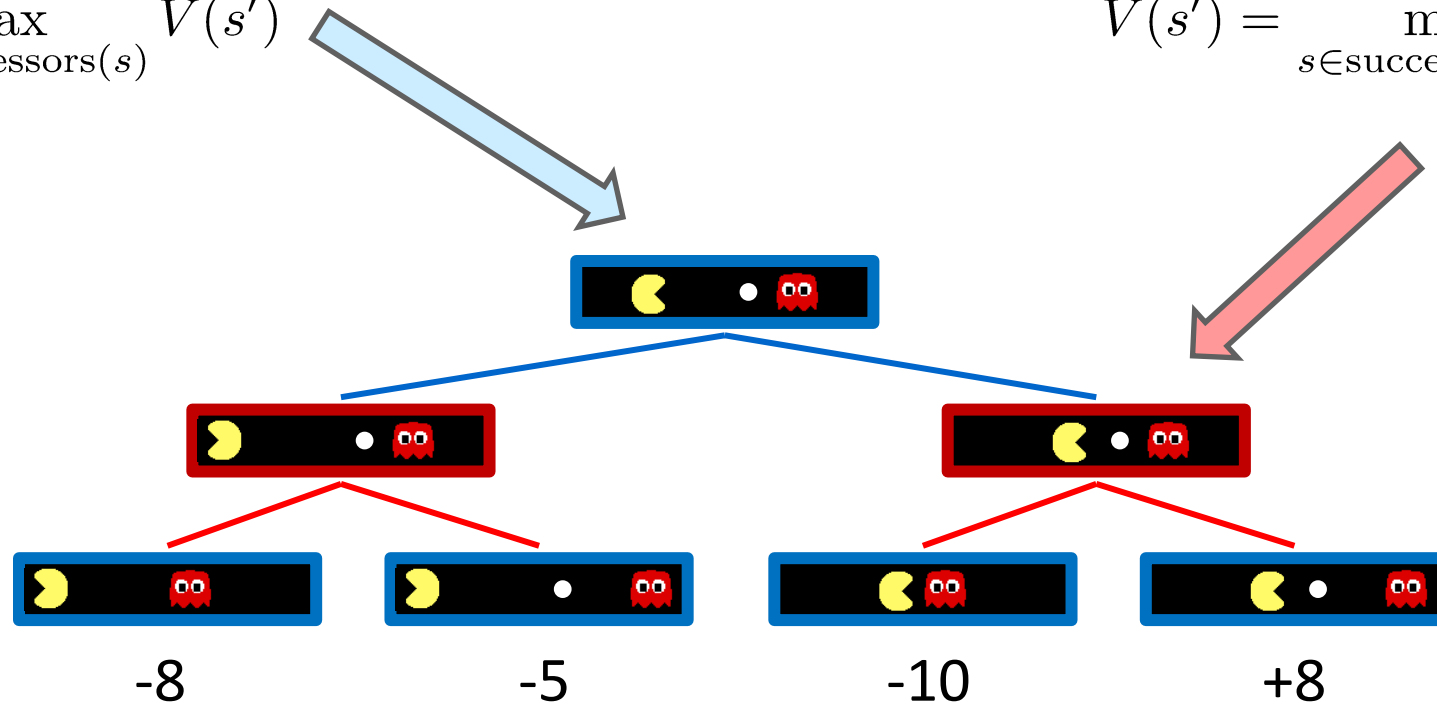
Minimax Values

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:

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

Tic-Tac-Toe Game Tree



MAX (X)



MIN (O)



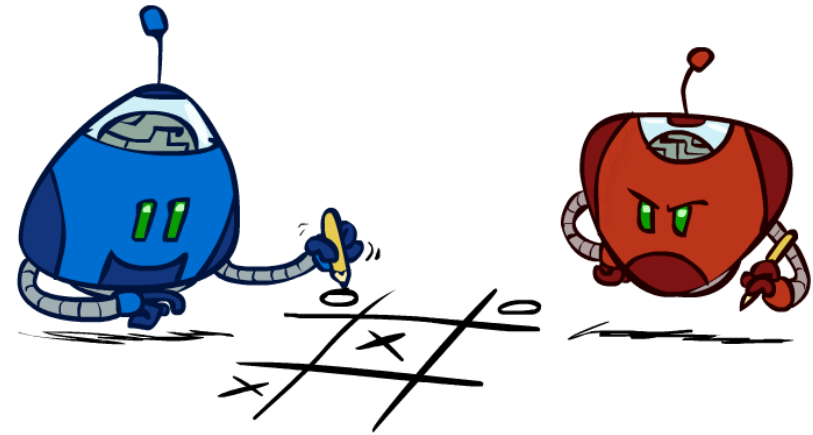
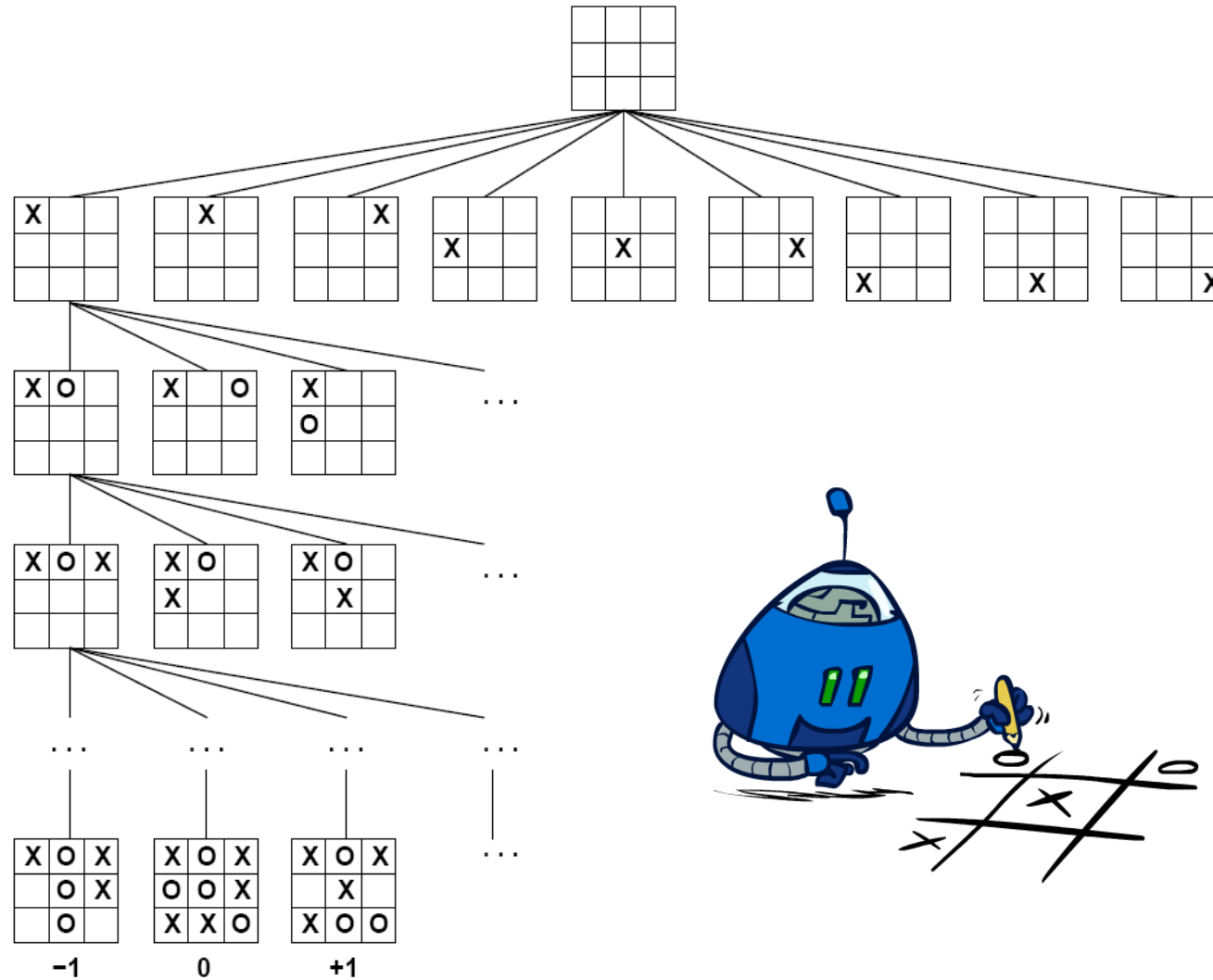
MAX (X)



MIN (O)

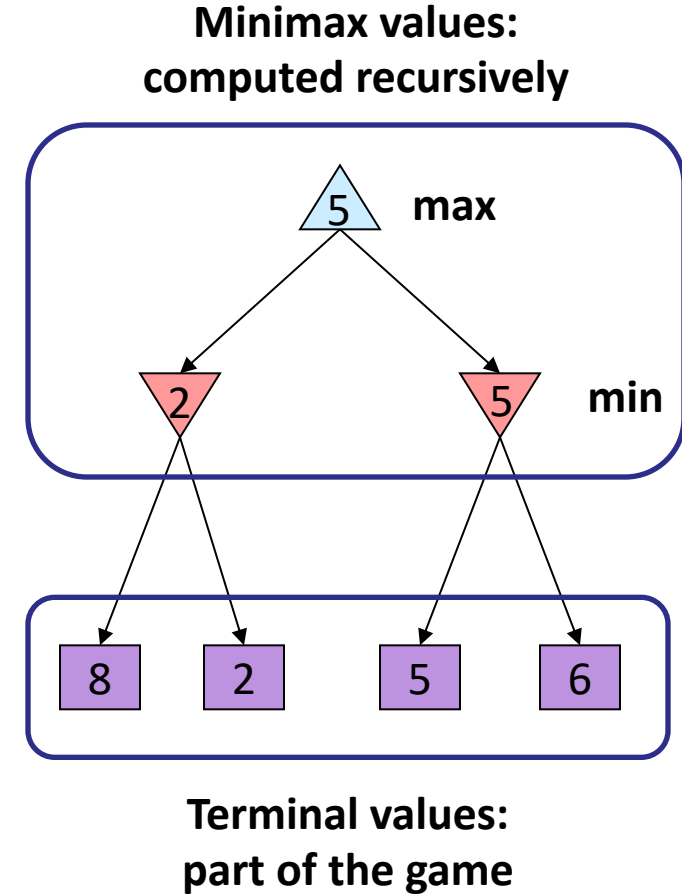
TERMINAL

Utility



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



Minimax Implementation

def max-value(state):

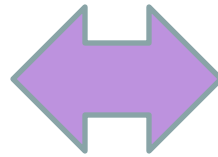
 initialize $v = -\infty$

 for each successor of state:

$v = \max(v, \text{min-value}(\text{successor}))$

 return v

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



def min-value(state):

 initialize $v = +\infty$

 for each successor of state:

$v = \min(v, \text{max-value}(\text{successor}))$

 return v

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

Minimax Implementation (Dispatch)

```
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 $v = -\infty$

for each successor of state:

$v = \max(v, \text{value}(\text{successor}))$

return v

```
def min-value(state):
```

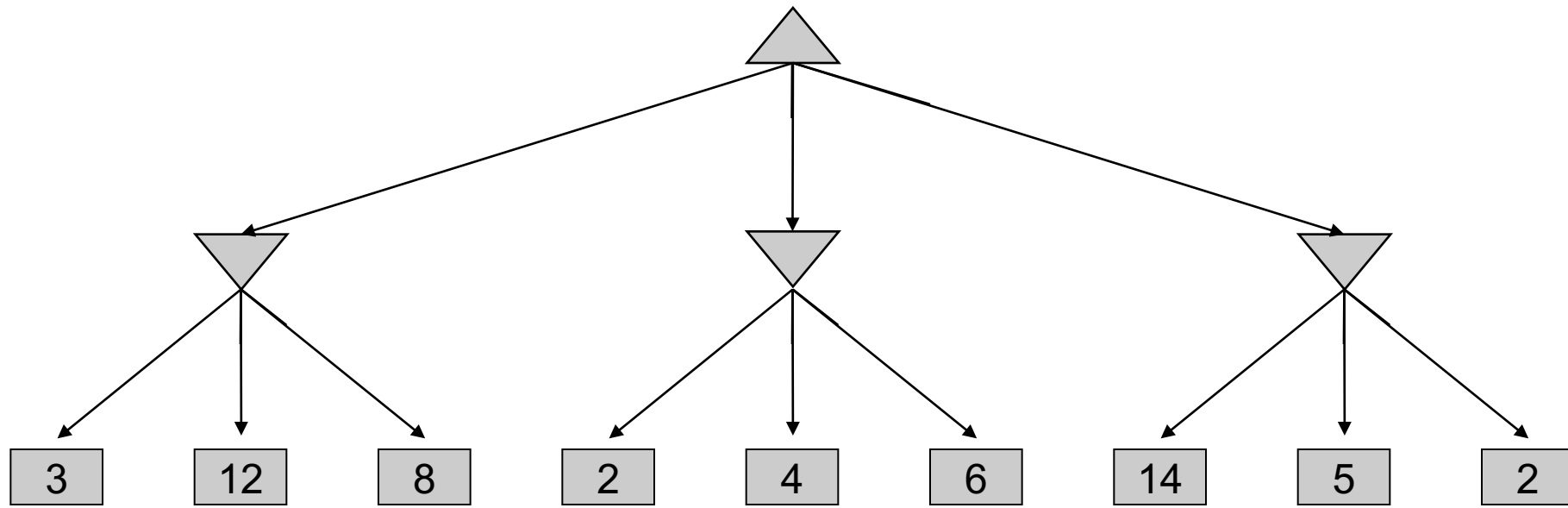
initialize $v = +\infty$

for each successor of state:

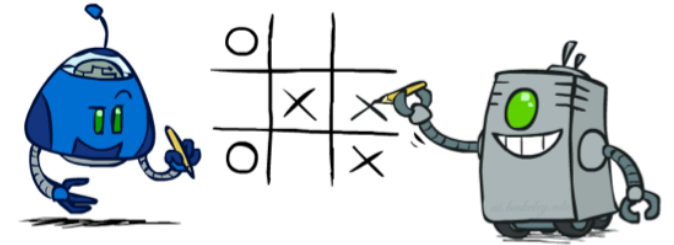
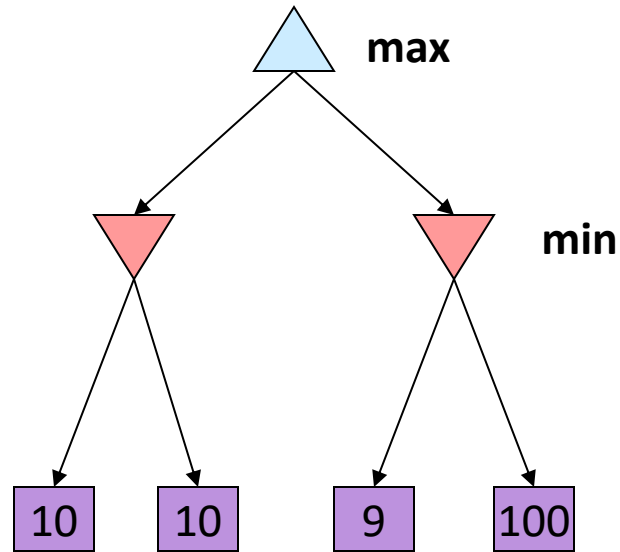
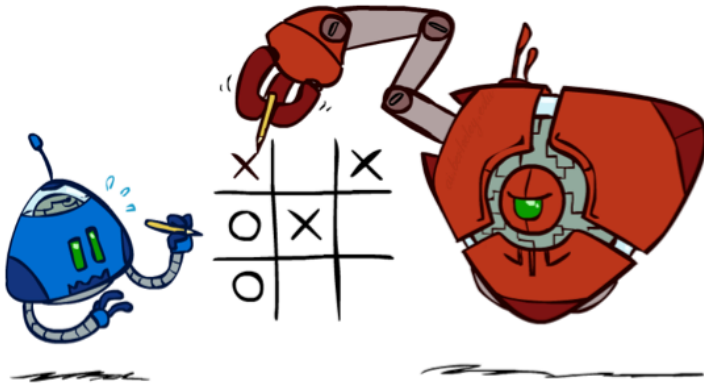
$v = \min(v, \text{value}(\text{successor}))$

return v

Minimax Example



Minimax Properties



Optimal against a perfect player. Otherwise?

Video of Demo Min vs. Exp (Min)

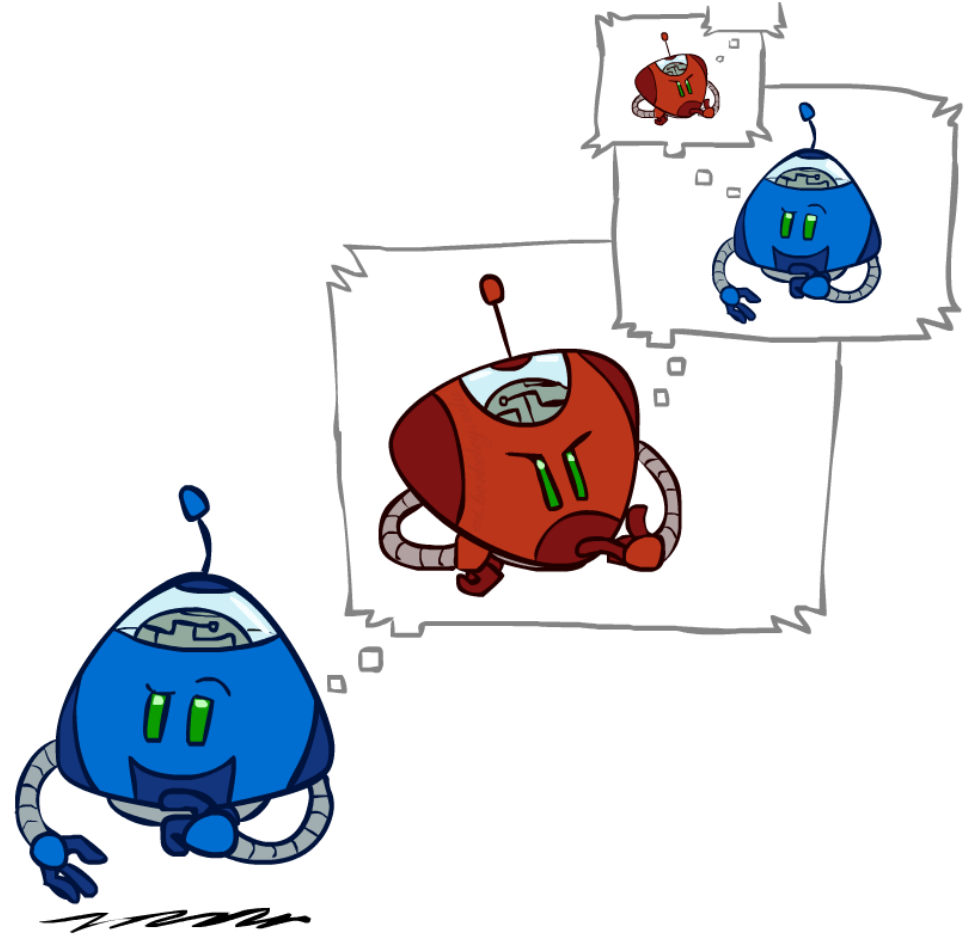


Video of Demo Min vs. Exp (Exp)

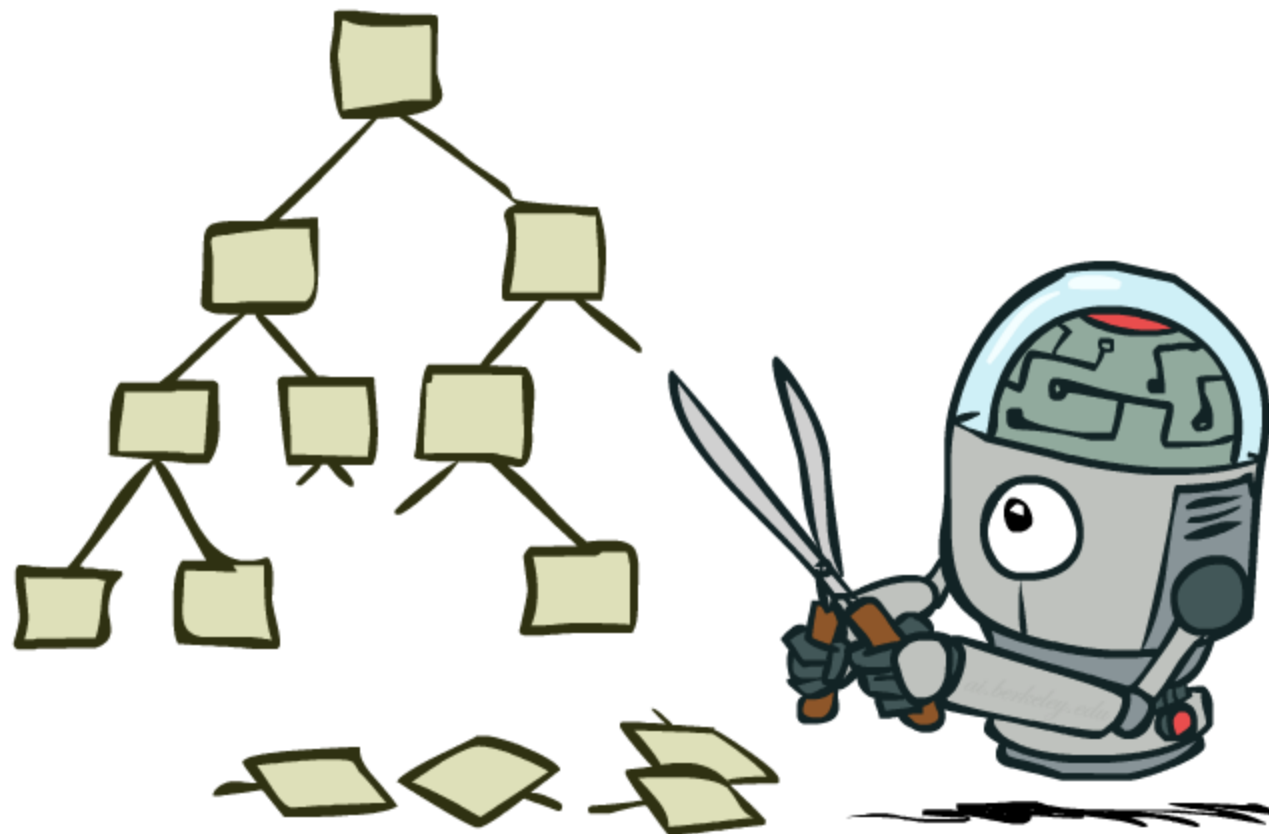


Minimax Efficiency

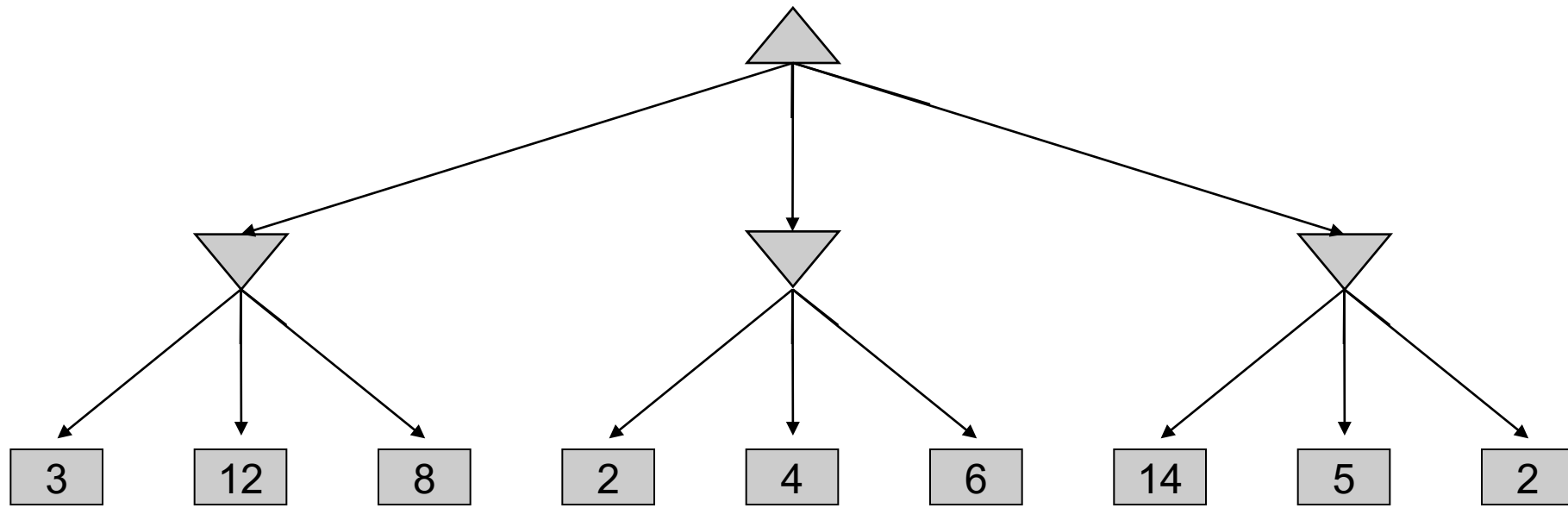
- How efficient is minimax?
 - Just like (exhaustive) DFS
 - Time: $O(b^m)$
 - Space: $O(bm)$
- Example: For chess, $b \approx 35$, $m \approx 100$
 - Exact solution is completely infeasible
 - But, do we need to explore the whole tree?



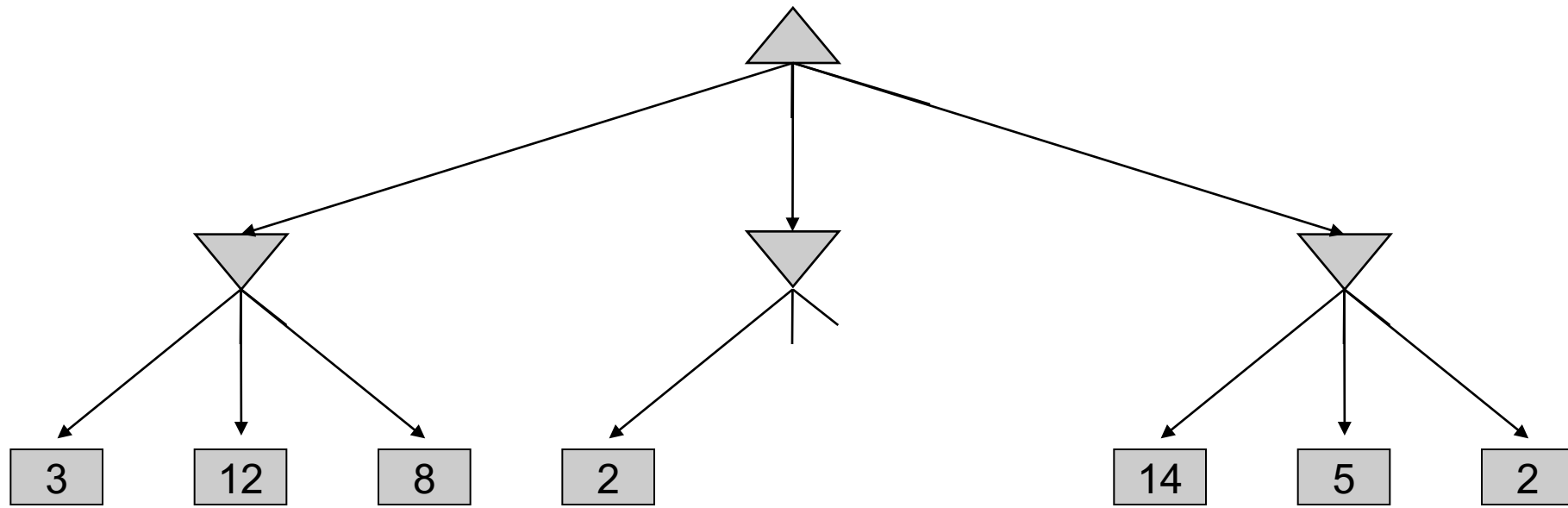
Game Tree Pruning



Minimax Example

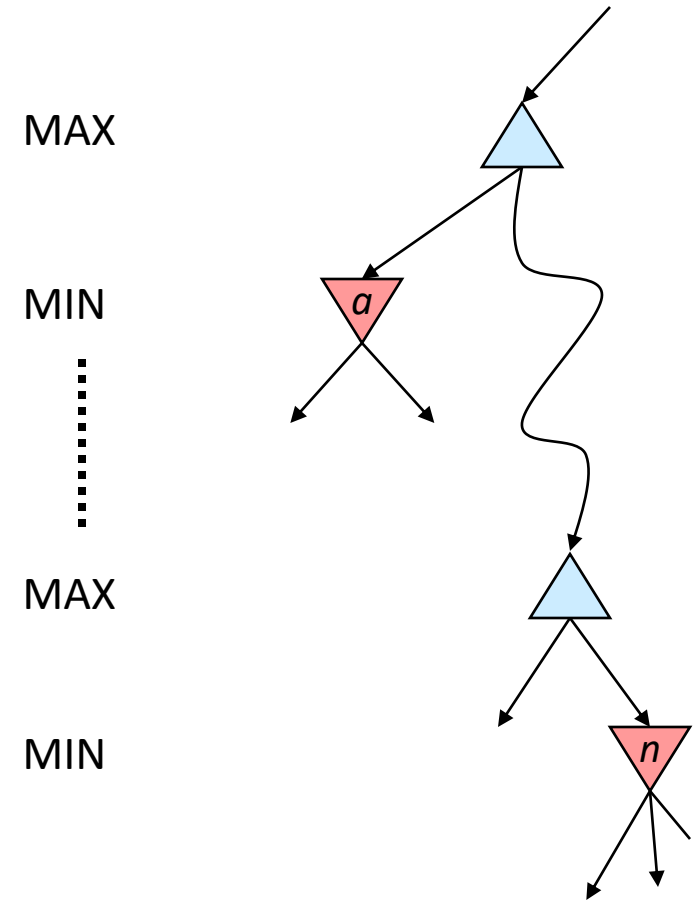


Minimax Pruning



Alpha-Beta Pruning

- General configuration (MIN version)
 - 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 a be the best value that MAX can get at any choice point along the current path from the root
 - If n becomes worse than a , MAX will avoid it, so we can stop considering n 's other children (it's already bad enough that it won't be played)
- MAX version is symmetric



Alpha-Beta Implementation

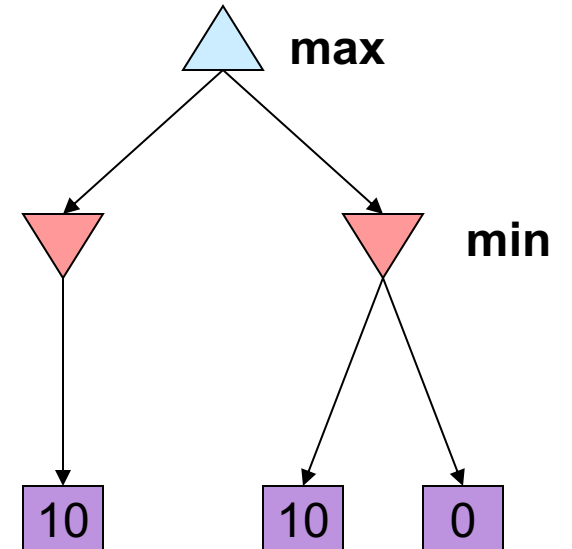
α : MAX's best option on path to root
 β : MIN's best option on path to root

```
def max-value(state,  $\alpha$ ,  $\beta$ ):  
    initialize  $v = -\infty$   
    for each successor of state:  
         $v = \max(v, \text{value}(\text{successor}, \alpha, \beta))$   
        if  $v \geq \beta$  return  $v$   
         $\alpha = \max(\alpha, v)$   
    return  $v$ 
```

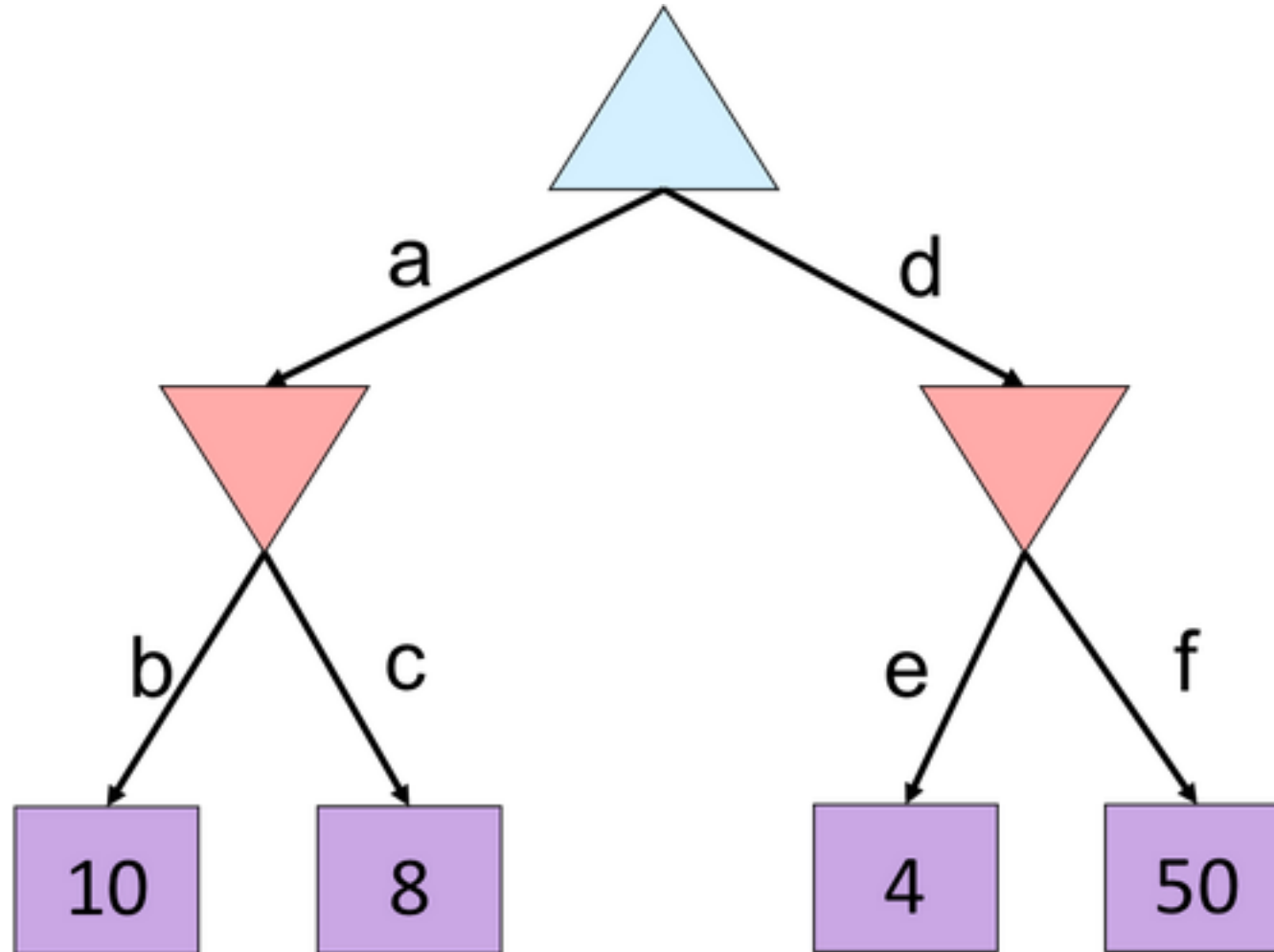
```
def min-value(state,  $\alpha$ ,  $\beta$ ):  
    initialize  $v = +\infty$   
    for each successor of state:  
         $v = \min(v, \text{value}(\text{successor}, \alpha, \beta))$   
        if  $v \leq \alpha$  return  $v$   
         $\beta = \min(\beta, v)$   
    return  $v$ 
```

Alpha-Beta Pruning Properties

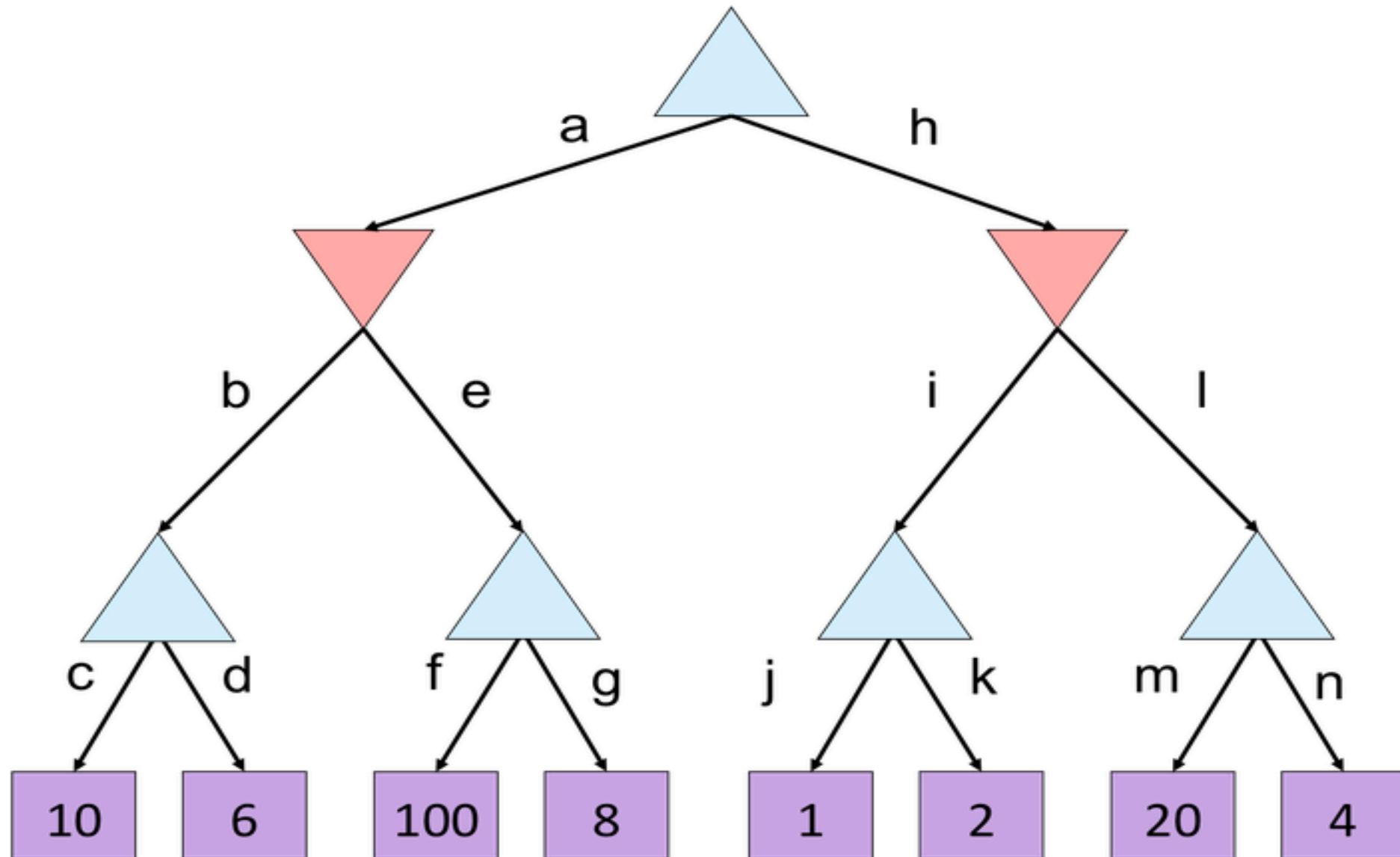
- This pruning has **no effect** on minimax value computed for the root!
- Values of intermediate nodes might be wrong
 - Important: children of the root may have the wrong value
 - So the most naïve version won't let you do action selection
- Good child ordering improves effectiveness of pruning
- With “perfect ordering”:
 - Time complexity drops to $O(b^{m/2})$
 - Doubles solvable depth!
 - Full search of, e.g. chess, is still hopeless...



Alpha-Beta Quiz



Alpha-Beta Quiz 2

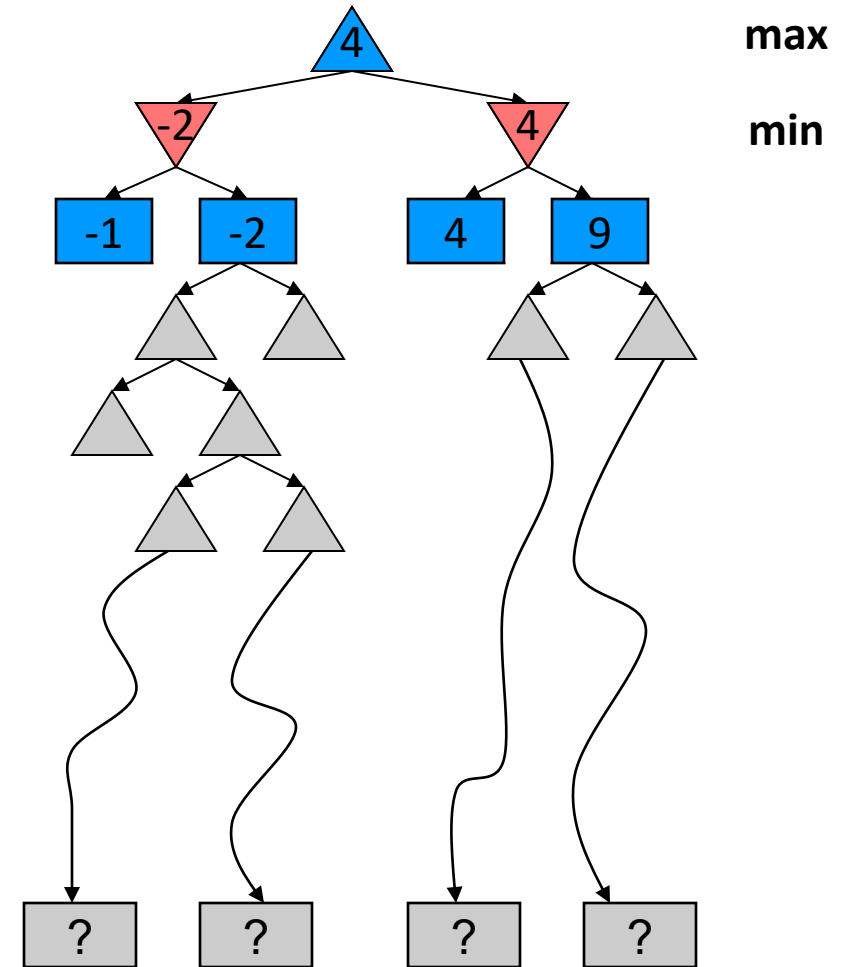


Resource Limits



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
- Use iterative deepening for an anytime algorithm

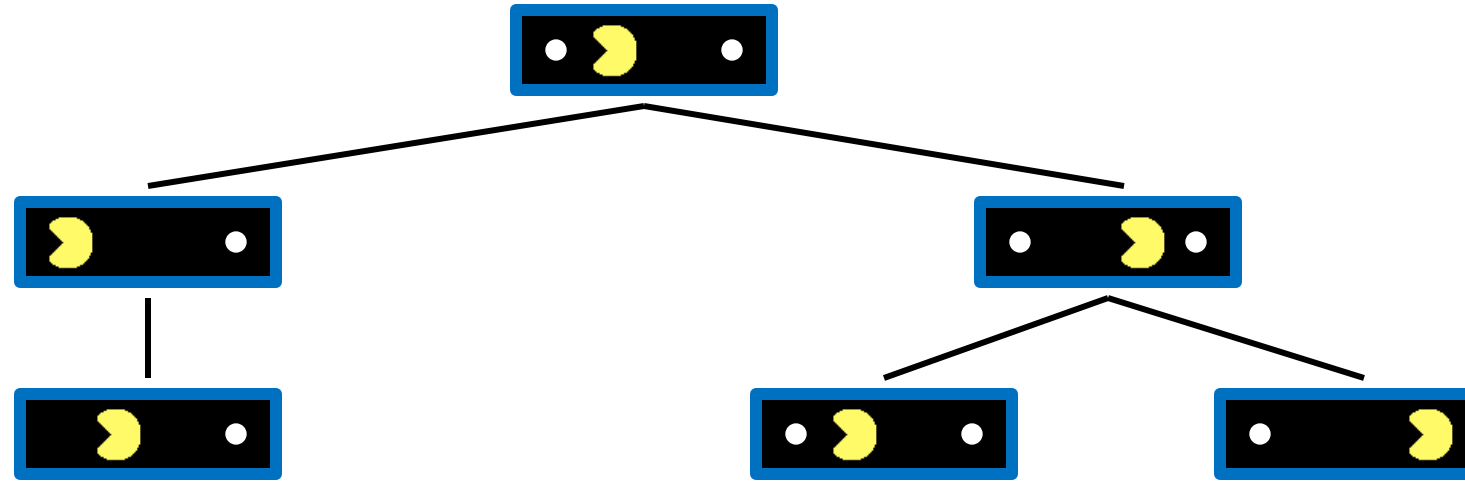


Video of Demo Thrashing (d=2)



[Demo: thrashing d=2, thrashing d=2 (fixed evaluation function) (L6D6)]

Why Pacman Starves



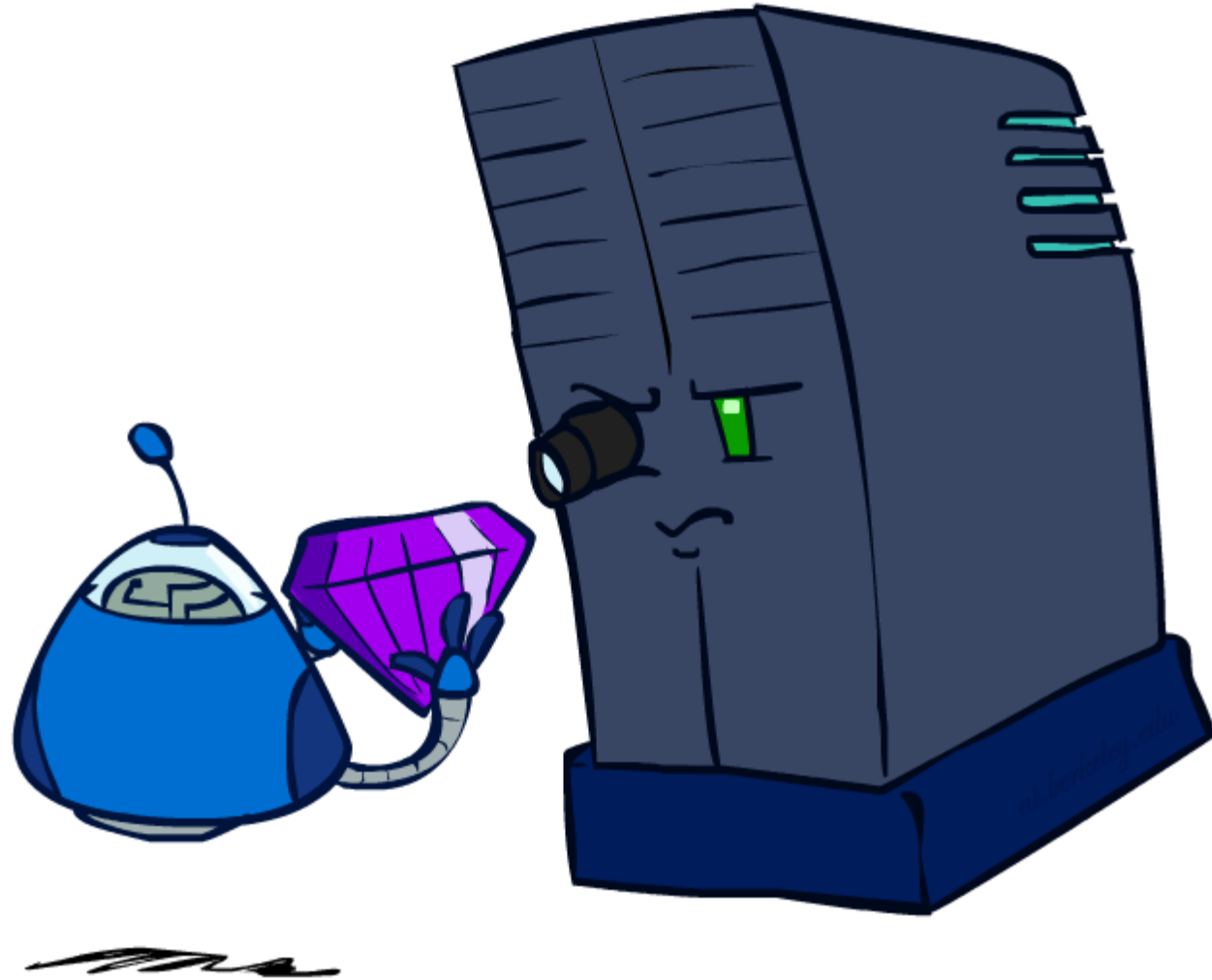
- A danger of replanning agents!
 - He knows his score will go up by eating the dot now (west, east)
 - He knows his score will go up just as much by eating the dot later (east, west)
 - There are no point-scoring opportunities after eating the dot (within the horizon, two here)
 - Therefore, waiting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

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



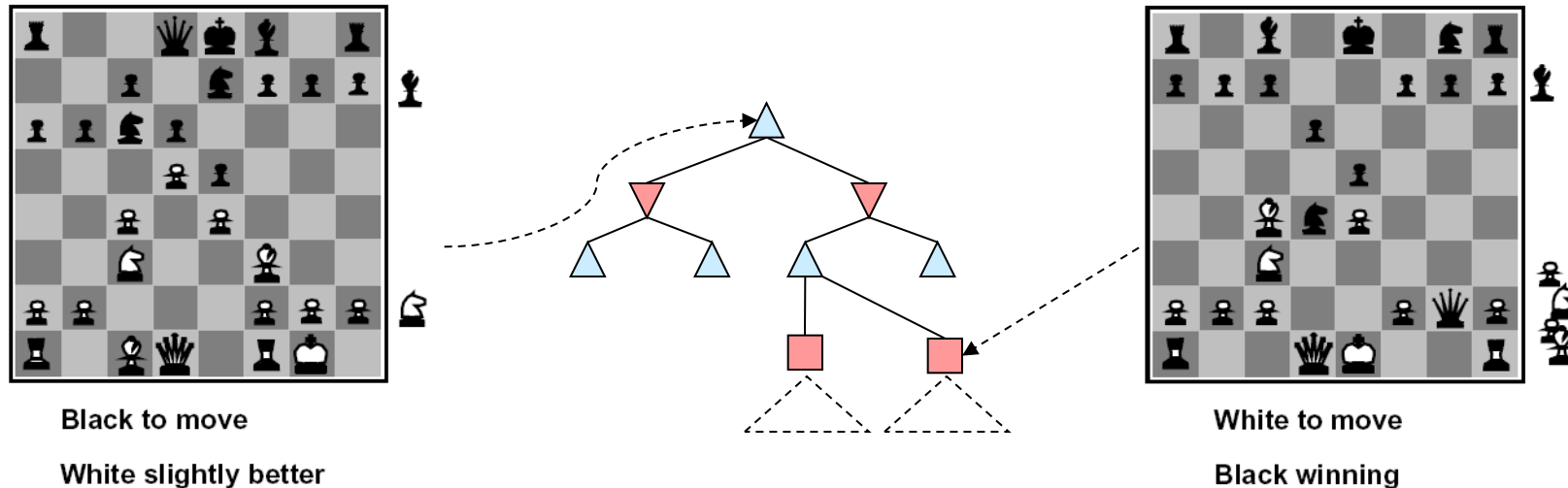
[Demo: thrashing $d=2$, thrashing $d=2$ (fixed evaluation function) (L6D7)]

Evaluation Functions



Evaluation Functions

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

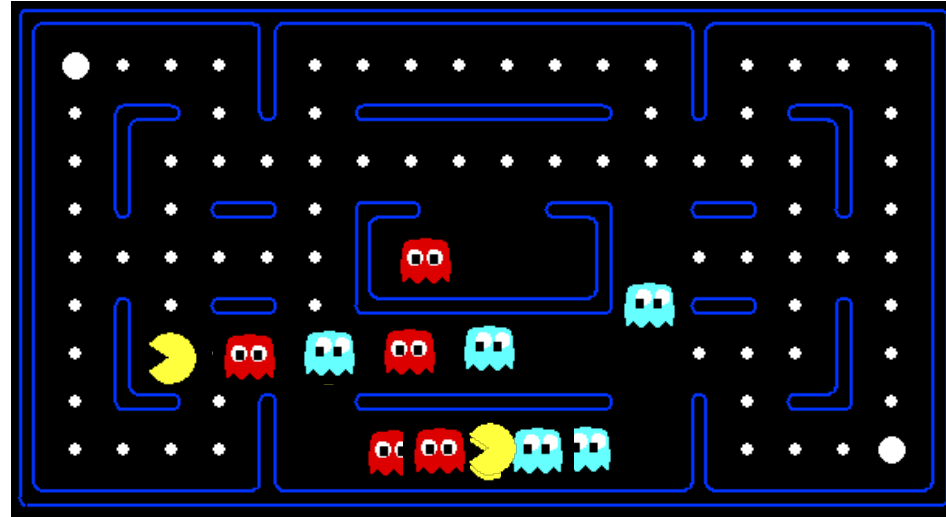


- Ideal function: returns the actual minimax value of the position
- In practice: typically weighted linear sum of features:

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

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

Evaluation for Pacman



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

Video of Demo Smart Ghosts (Coordination)

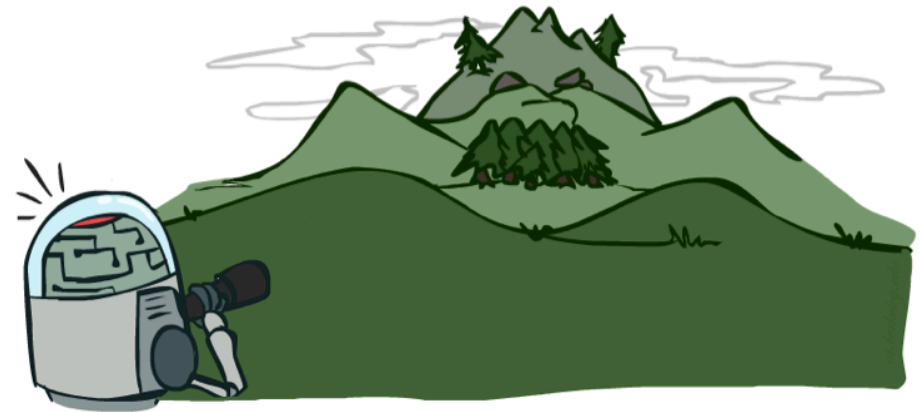
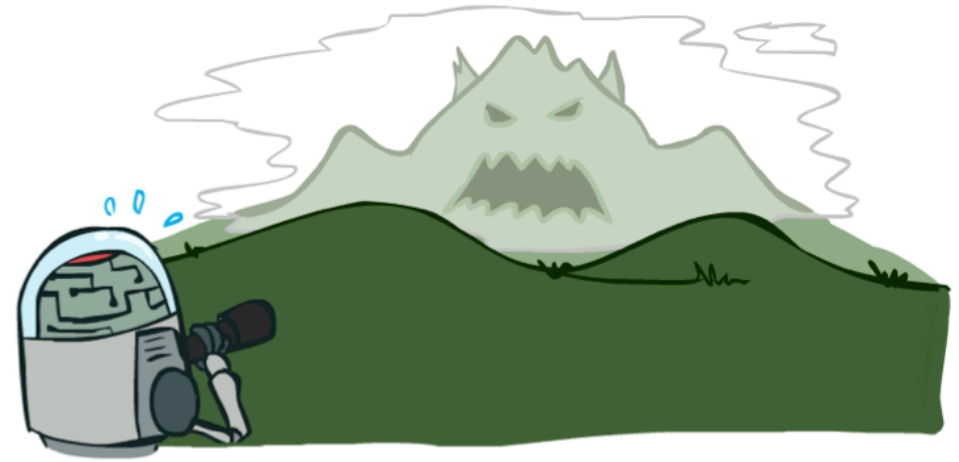


Video of Demo Smart Ghosts (Coordination) – Zoomed In



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



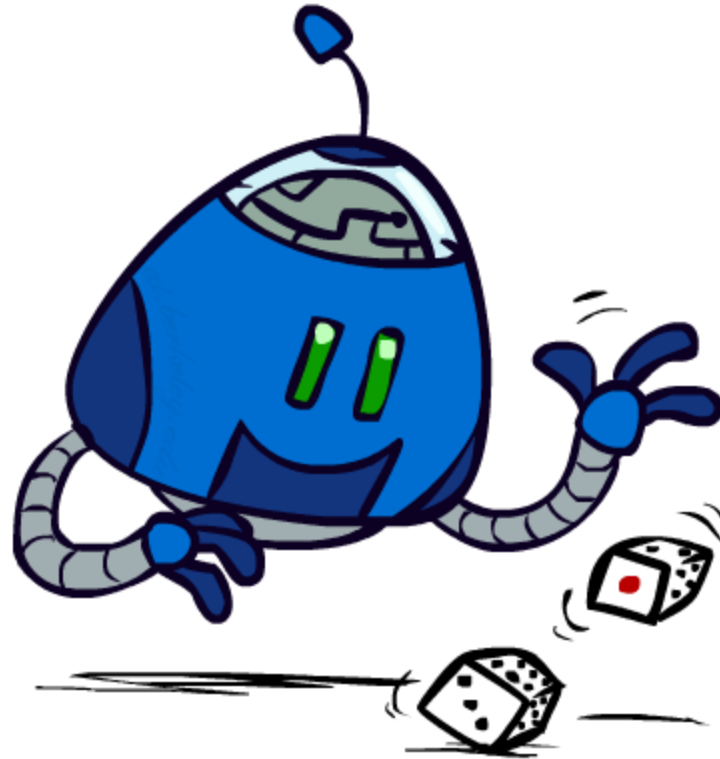
Video of Demo Limited Depth (2)



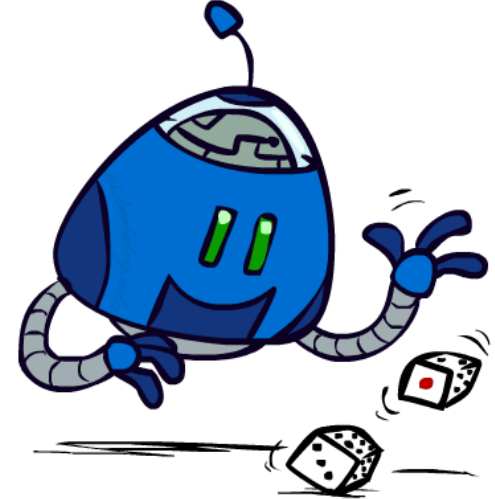
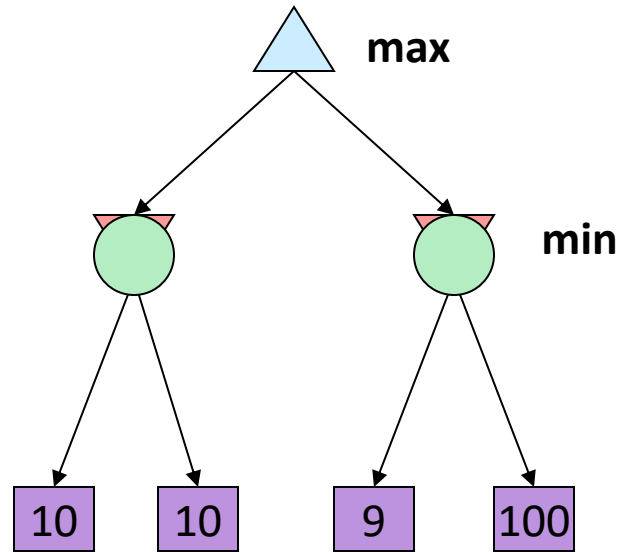
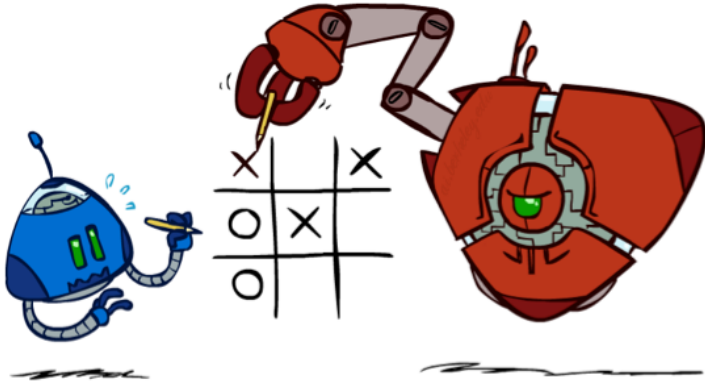
Video of Demo Limited Depth (10)



Uncertain Outcomes



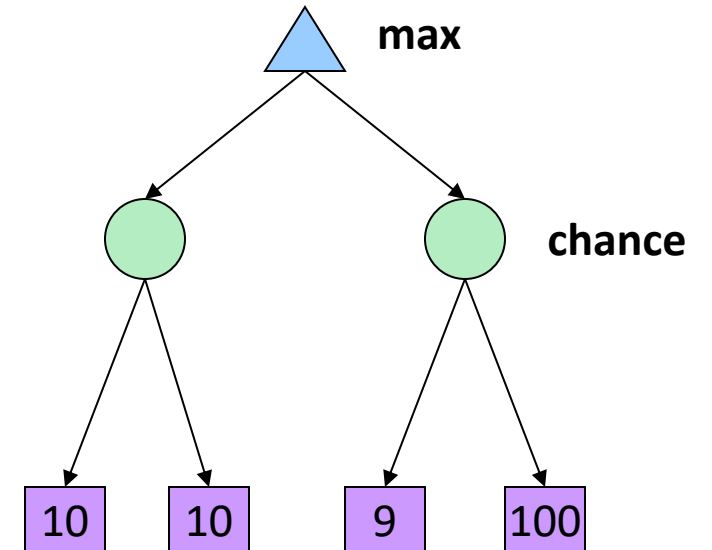
Worst-Case vs. Average Case



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**



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 $v = -\infty$

for each successor of state:

$v = \max(v, \text{value}(\text{successor}))$

return v

```
def exp-value(state):
```

initialize $v = 0$

for each successor of state:

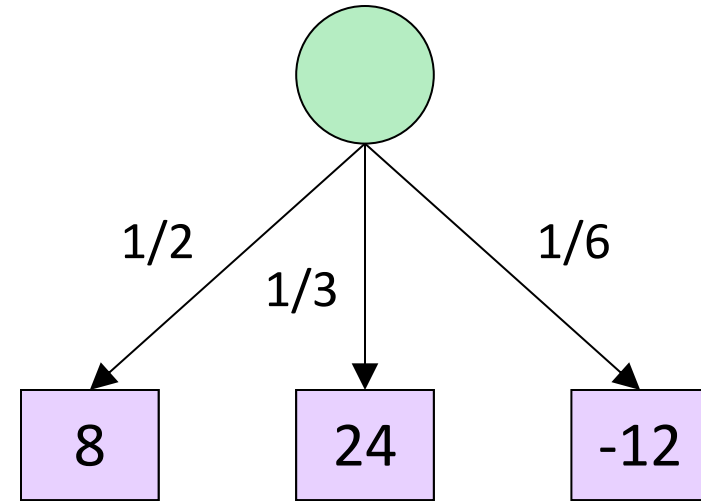
$p = \text{probability}(\text{successor})$

$v += p * \text{value}(\text{successor})$

return v

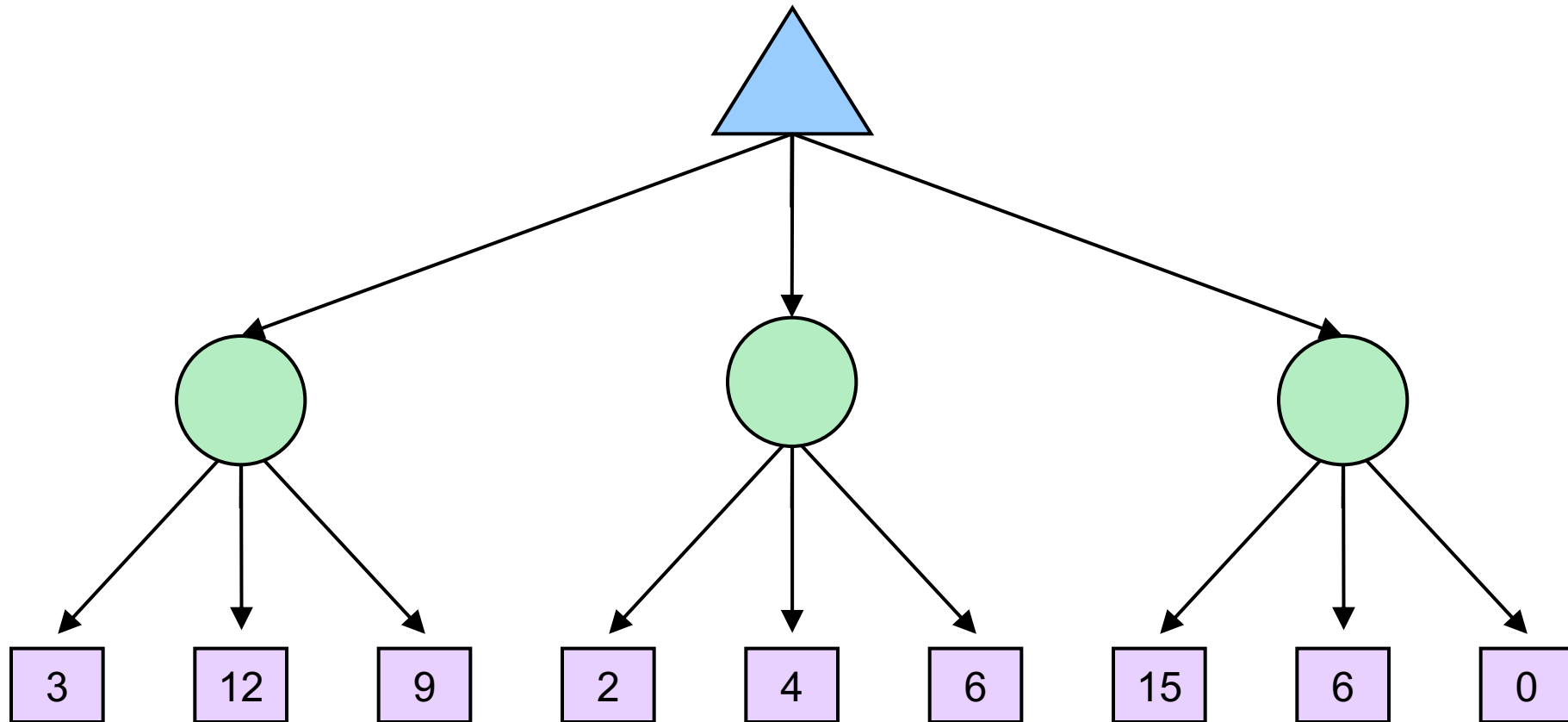
Expectimax Pseudocode

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

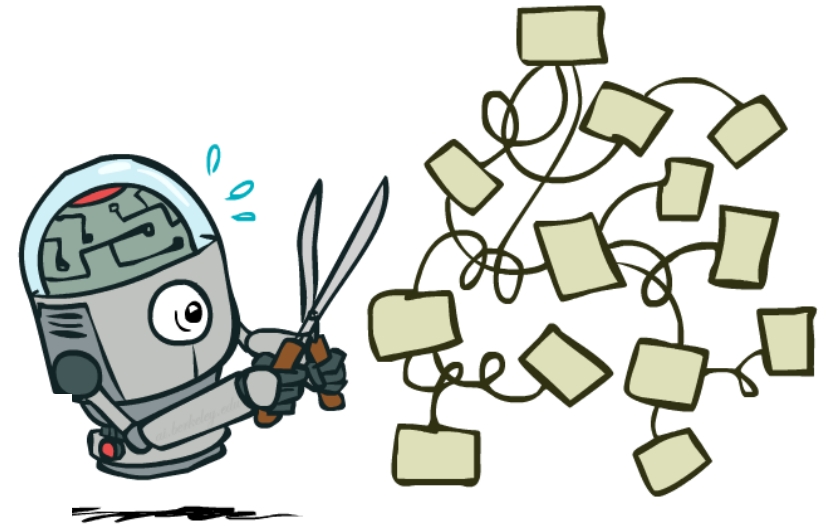
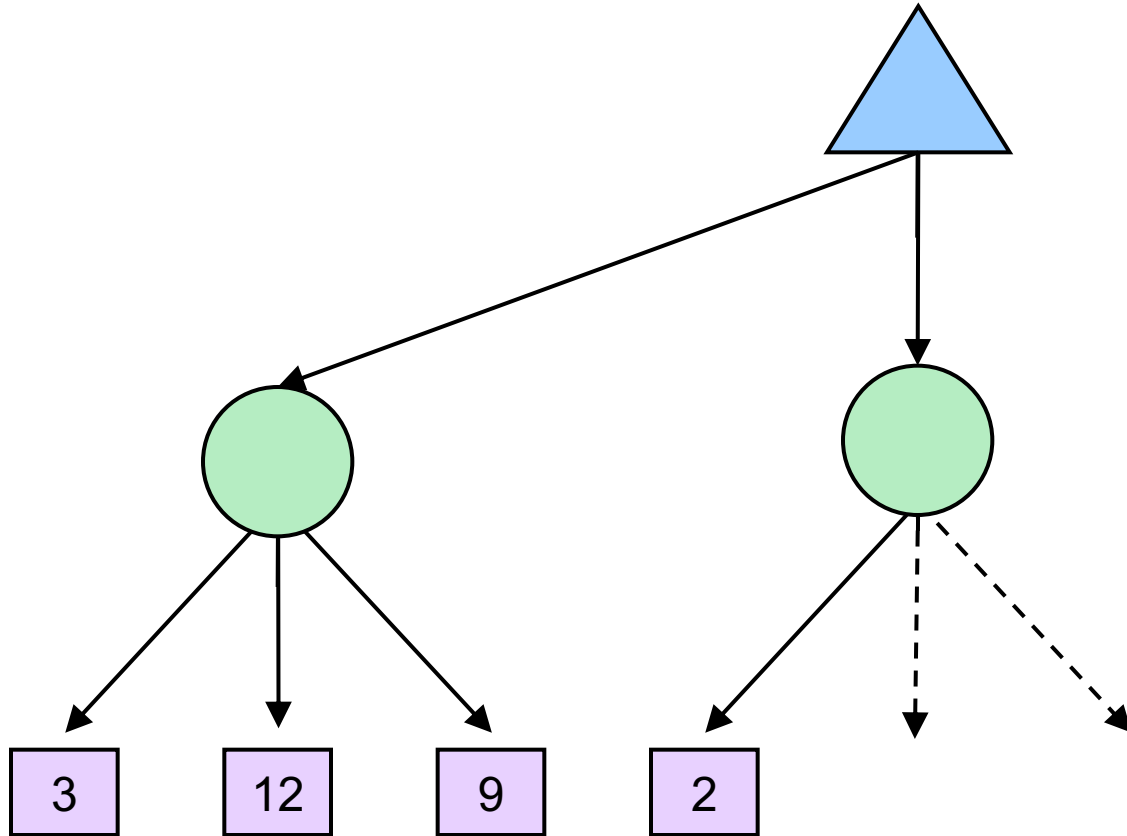


$$v = (1/2) (8) + (1/3) (24) + (1/6) (-12) = 10$$

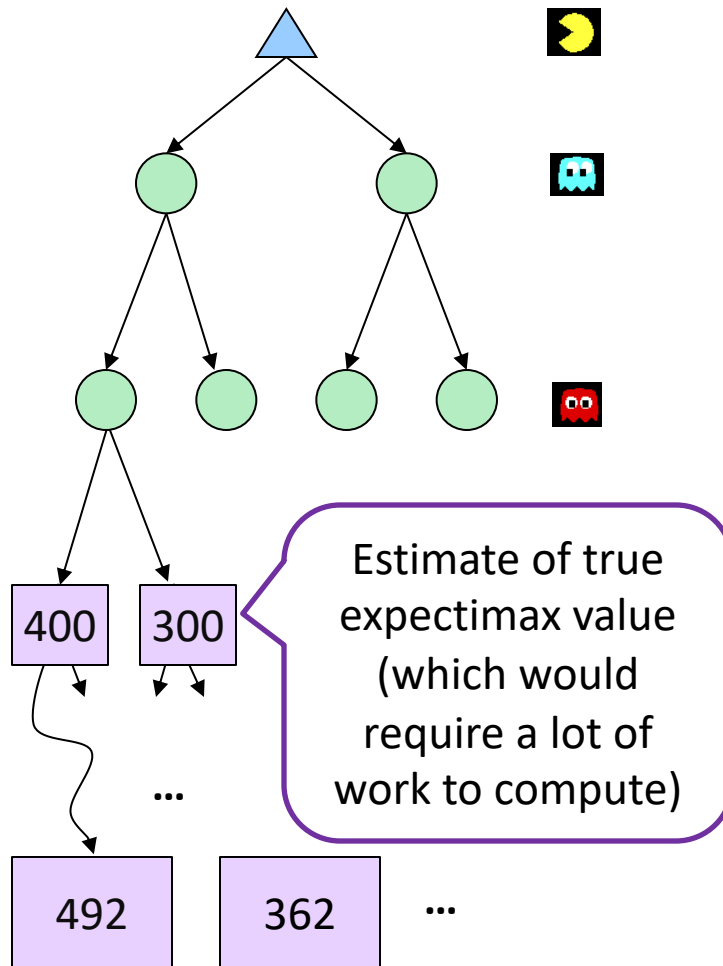
Expectimax Example



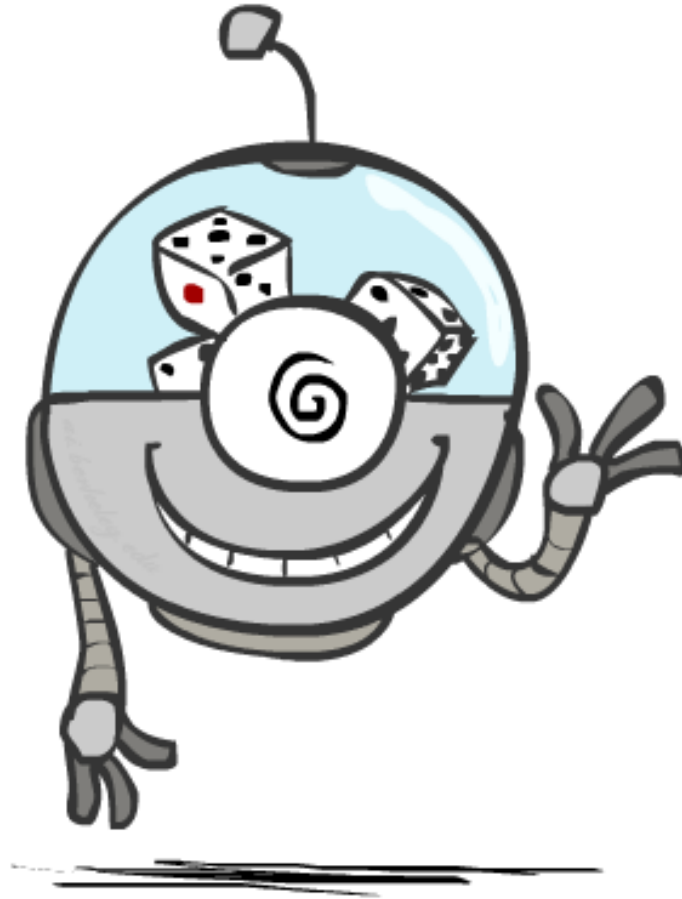
Expectimax Pruning?



Depth-Limited Expectimax



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: $P(T=\text{none}) = 0.25$, $P(T=\text{light}) = 0.50$, $P(T=\text{heavy}) = 0.25$
- Some laws of probability:
 - Probabilities are always non-negative
 - Probabilities over all possible outcomes sum to one



0.25



0.50



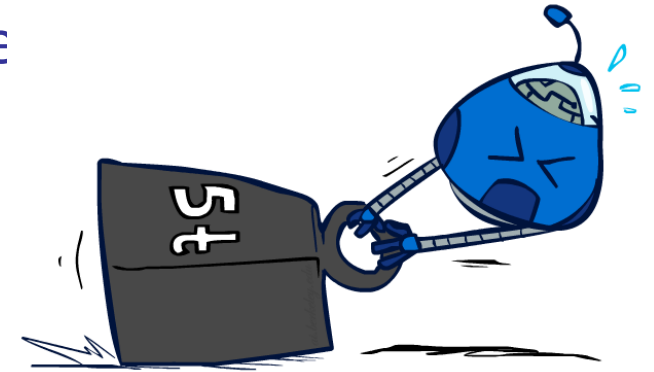
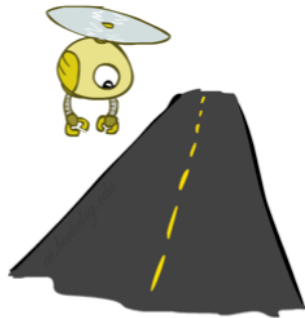
0.25

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?

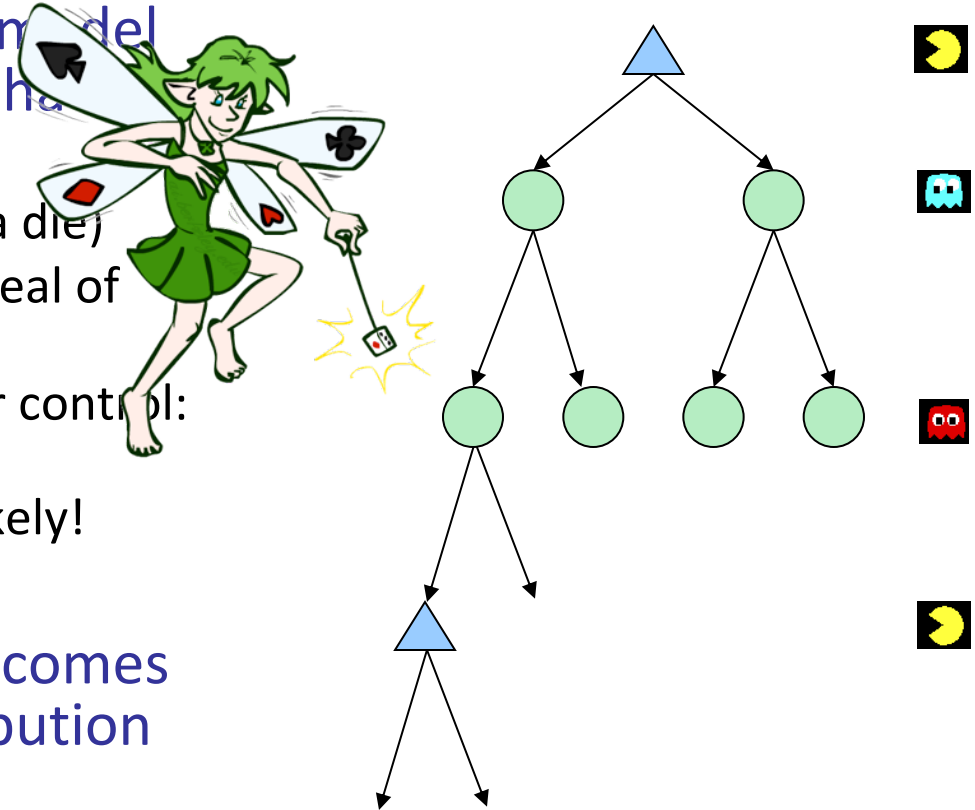
Time:	20 min		30 min		60 min		
	x		x		x		
Probability:	0.25	+	0.50	+	0.25		

35 min



What Probabilities to Use?

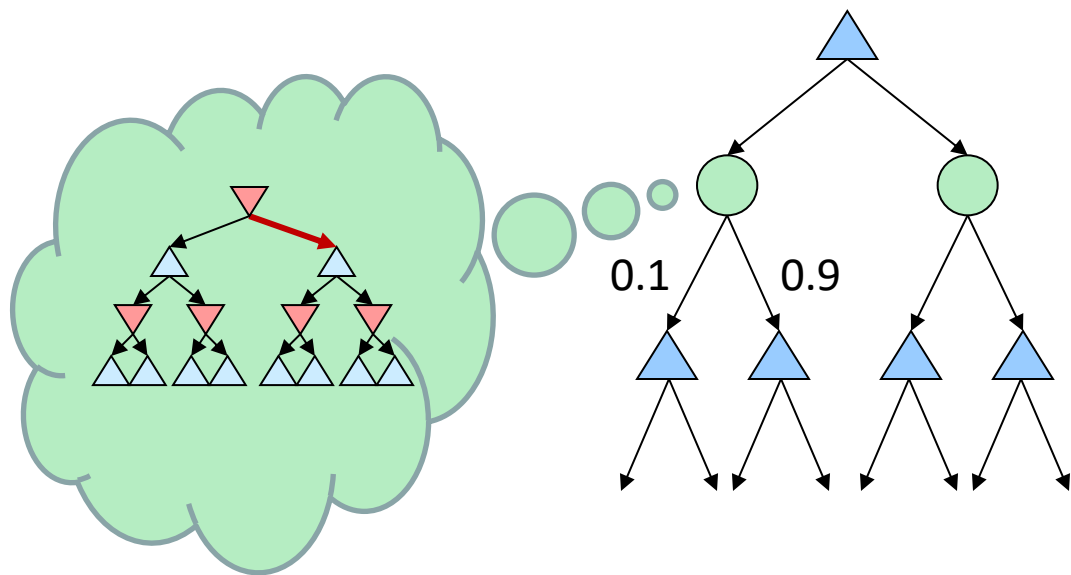
- In expectimax search, we have a probabilistic model of how the opponent (or environment) will behave in any state
 - Model could be a simple uniform distribution (roll a die)
 - Model could be sophisticated and require a great deal of computation
 - We have a chance node for any outcome out of our 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!

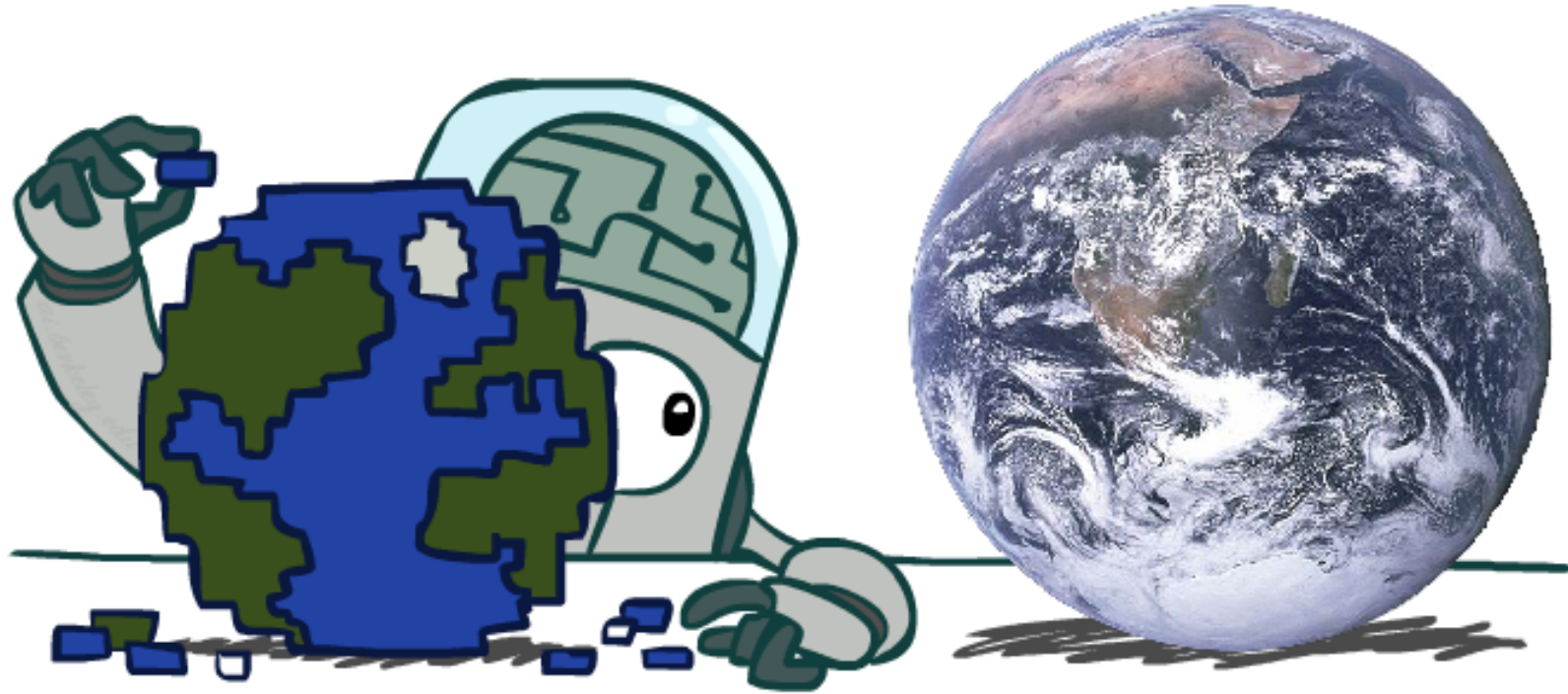
Quiz: Informed Probabilities

- 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

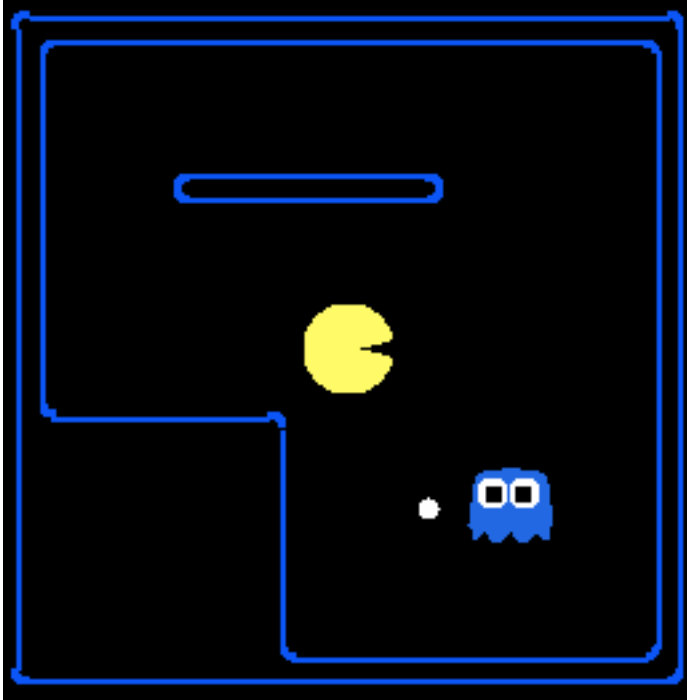


Dangerous Pessimism

Assuming the worst case when it's not likely



Assumptions vs. Reality



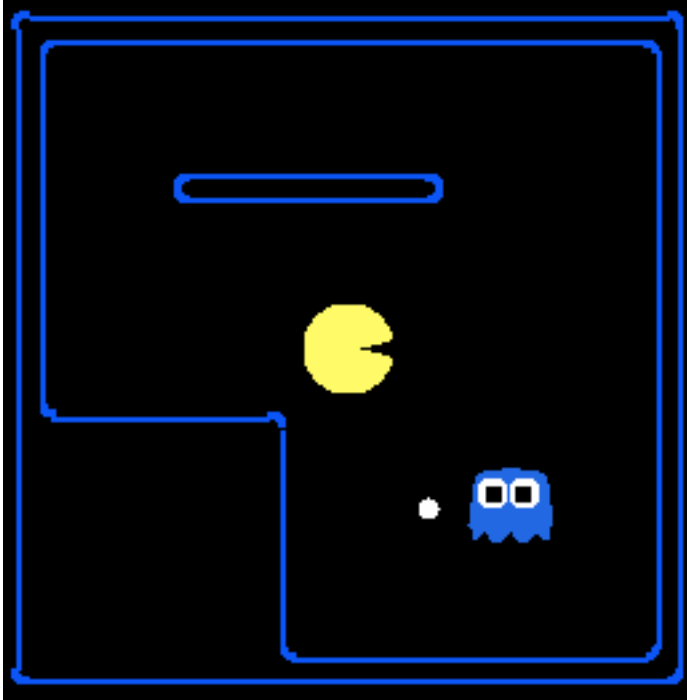
	Adversarial Ghost	Random Ghost
Minimax Pacman		
Expectimax Pacman		

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

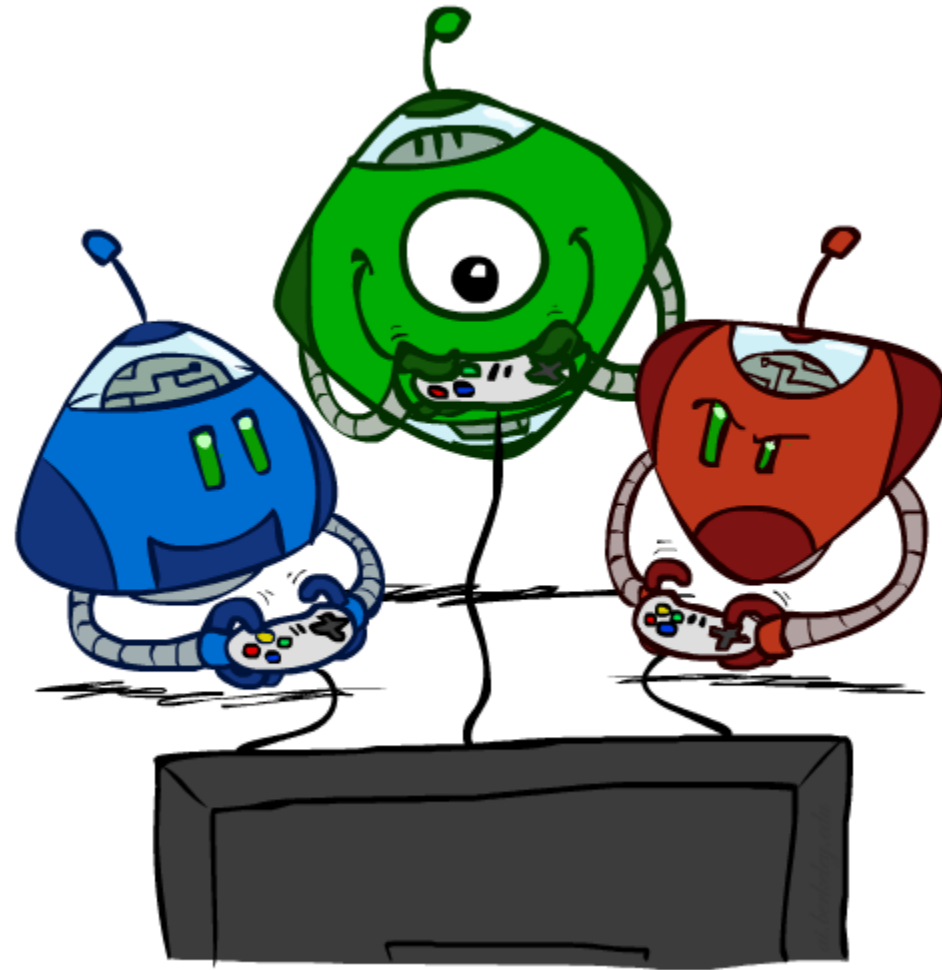


Video of Demo World Assumptions

Random Ghost – Minimax Pacman

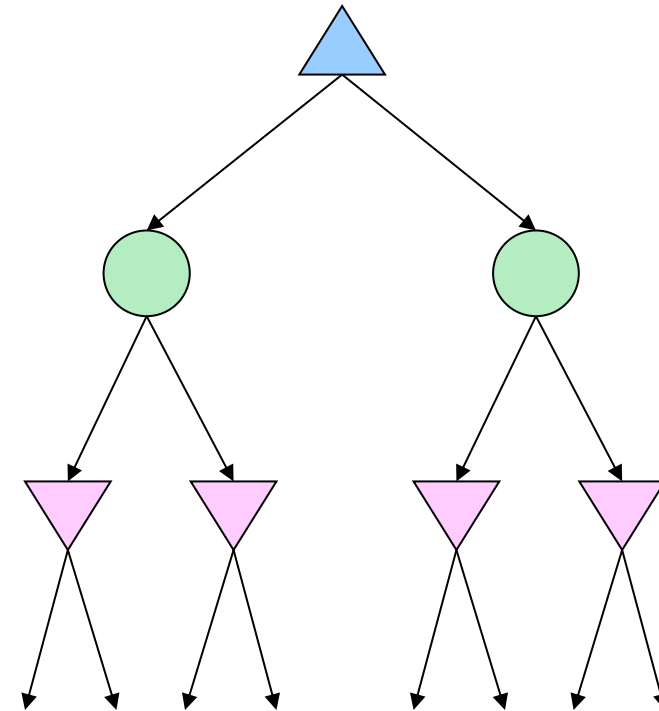
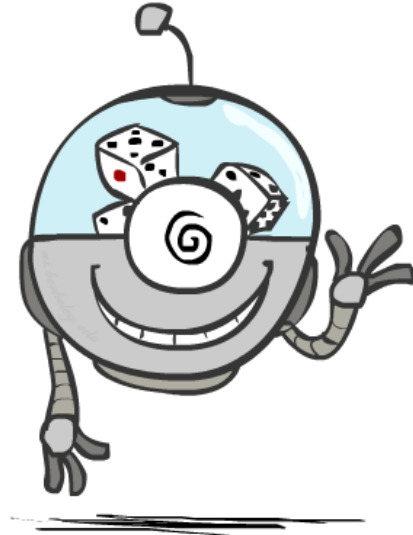


Other Game Types



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



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...

