07-280 AI & ML I
Adversarial Search

Instructors:
Pat Virtue & Nihar Shah

Minimax

Why not use A* search?

Why the focus on games?

States

Actions

Values

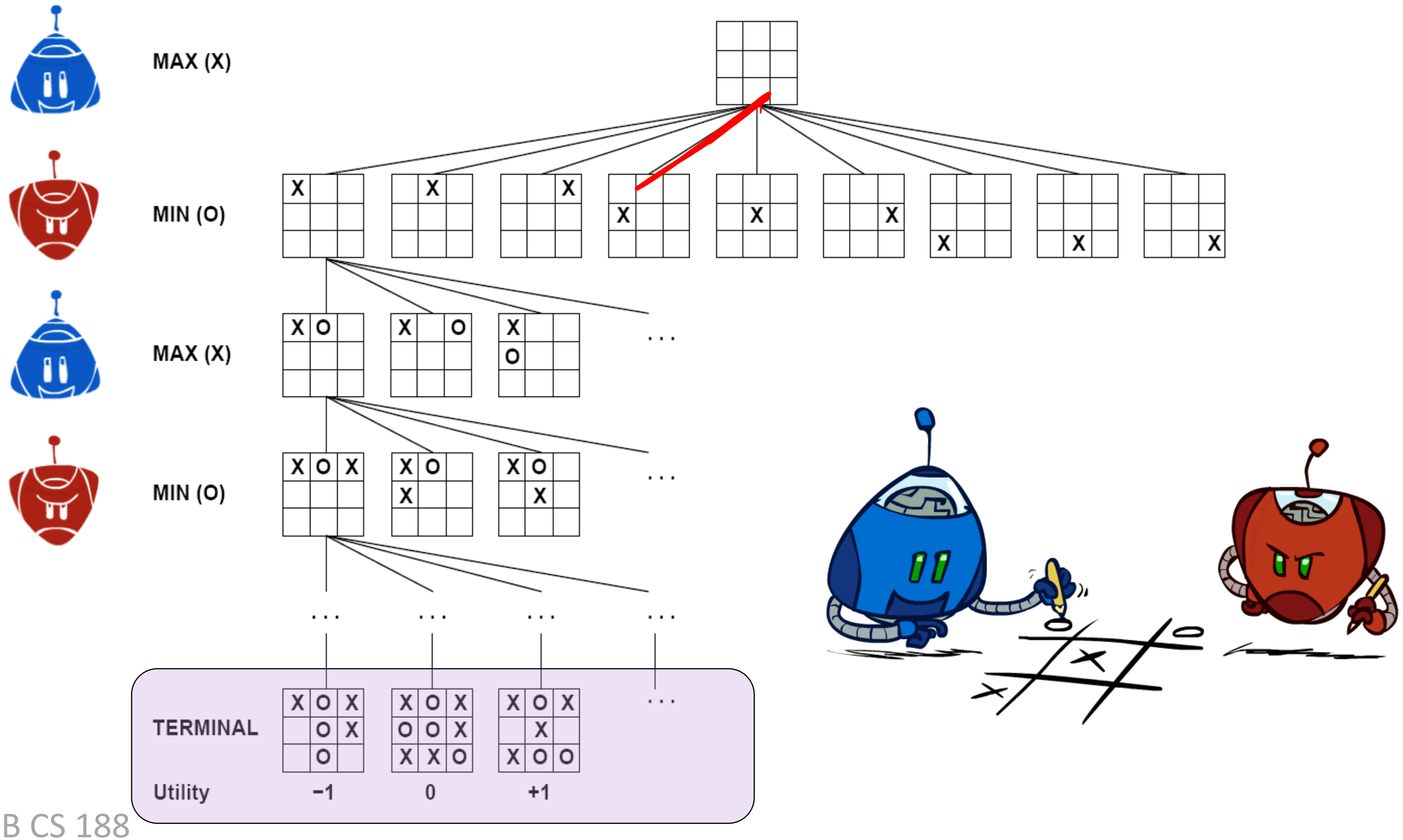


Image credit: Ketrina Yim, UCB CS 188

Intelligence and Uncertainty

Uncertainty can have lots of sources, including anything we attribute to random chance

- Hidden information
  - Cards in another player's hand
- Noise
  - Sensor noise
- Way to complicated to model
  - Leaves blowing in the wind
- Infinite number of possible configurations
- More possibilities than any computer can compute in a reasonable time
  - Tic-tac-toe → Checkers → Chess

Outline

Pre-reading

- History
- Standard/Zero-Sum Games
- Minimax
- Expected Value
- Expectimax

Evaluation Functions

Pruning

Modeling Assumptions

Image credit: Ketrina Yim, UCB CS 188 and well, Pat Virtue

Pre-reading

Game Playing State-of-the-Art

Checkers:

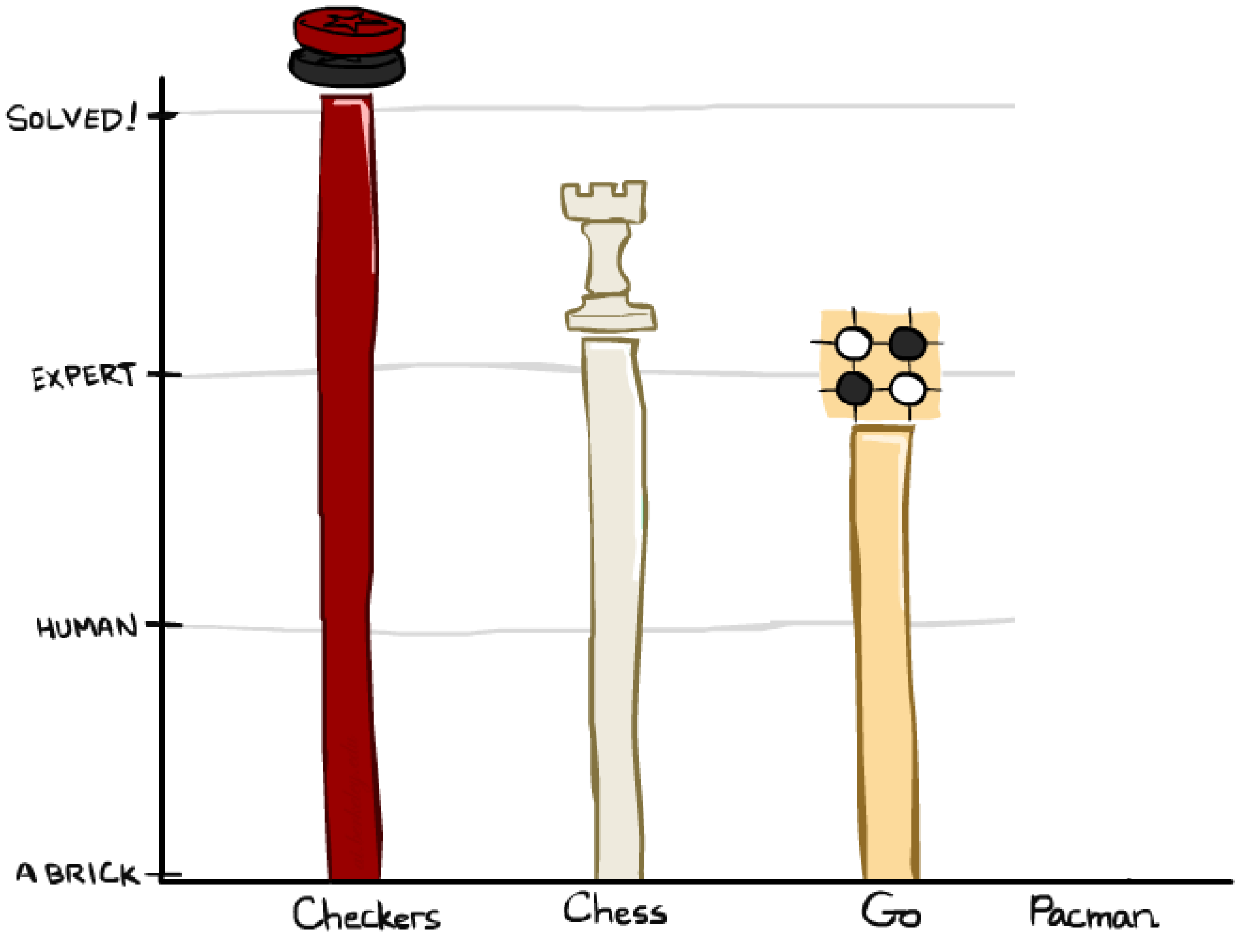
- 1950: First computer player.
- 1959: Samuel's self-taught program.
- 1994: First computer world champion: Chinook ended 40-year-reign of human champion Marion Tinsley using complete 8-piece endgame.
- 2007: Checkers solved! Endgame database of 39 trillion states

Chess:

- 1945-1960: Zuse, Wiener, Shannon, Turing, Newell & Simon, McCarthy.
- 1960s onward: gradual improvement under "standard model"
- 1997: special-purpose chess machine Deep Blue defeats human champion Gary Kasparov in a six-game match. Deep Blue examined 200M positions per second and extended some lines of search up to 40 ply. Current programs running on a PC rate > 3200 (vs 2870 for Magnus Carlsen).

Go:

- 1968: Zobrist's program plays legal Go, barely (b>300!)
- 2005-2014: Monte Carlo tree search enables rapid advances: current programs beat strong amateurs, and professionals with a 3-4 stone handicap.



Slide/Image credit: Ketrina Yim, UCB CS 188

Game Playing State-of-the-Art

Checkers:

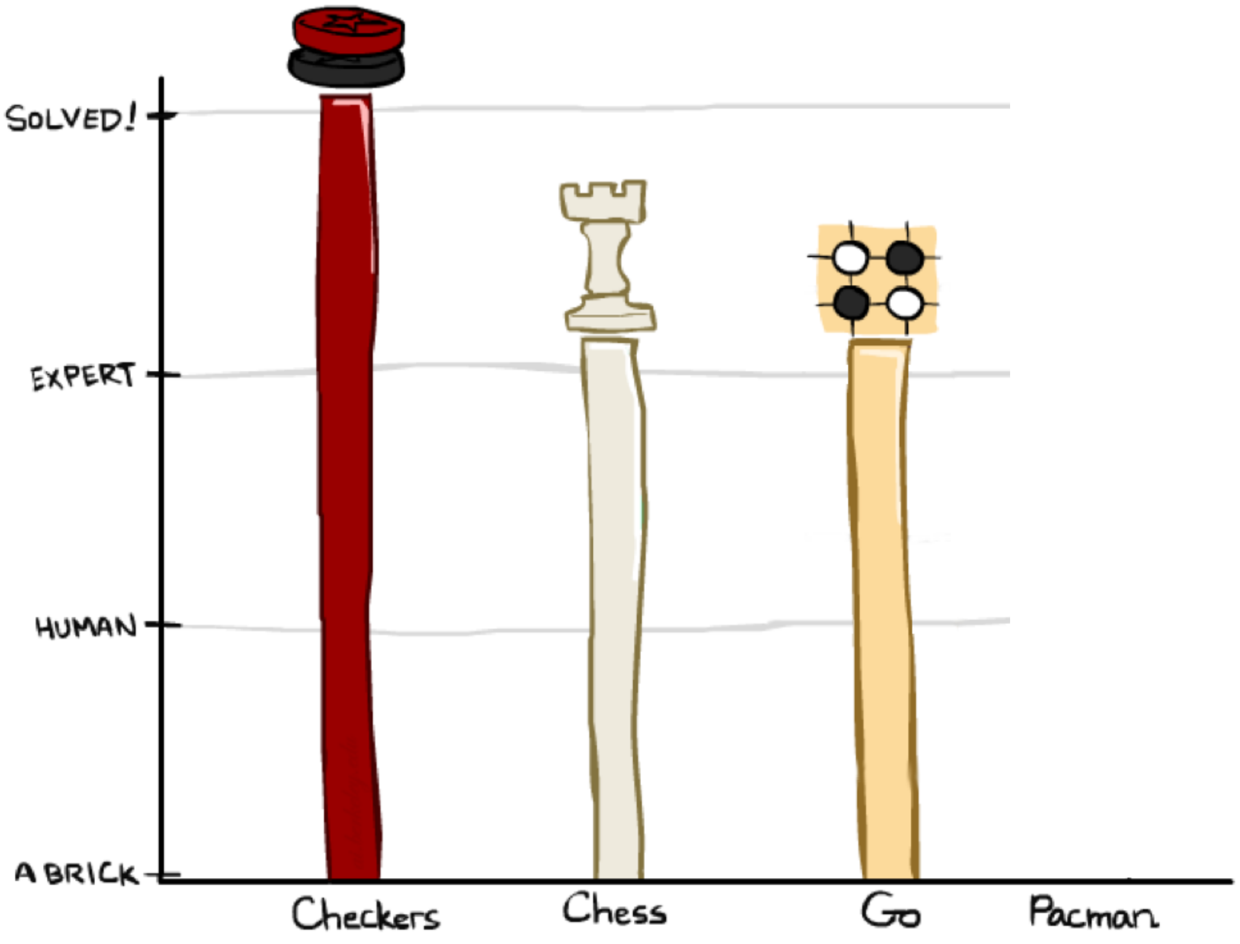
- 1950: First computer player.
- 1959: Samuel's self-taught program.
- 1994: First computer world champion: Chinook ended 40-year-reign of human champion Marion Tinsley using complete 8-piece endgame.
- 2007: Checkers solved! Endgame database of 39 trillion states

Chess:

- 1945-1960: Zuse, Wiener, Shannon, Turing, Newell & Simon, McCarthy.
- 1960s onward: gradual improvement under "standard model"
- 1997: special-purpose chess machine Deep Blue defeats human champion Gary Kasparov in a six-game match. Deep Blue examined 200M positions per second and extended some lines of search up to 40 ply. Current programs running on a PC rate > 3200 (vs 2870 for Magnus Carlsen).

Go:

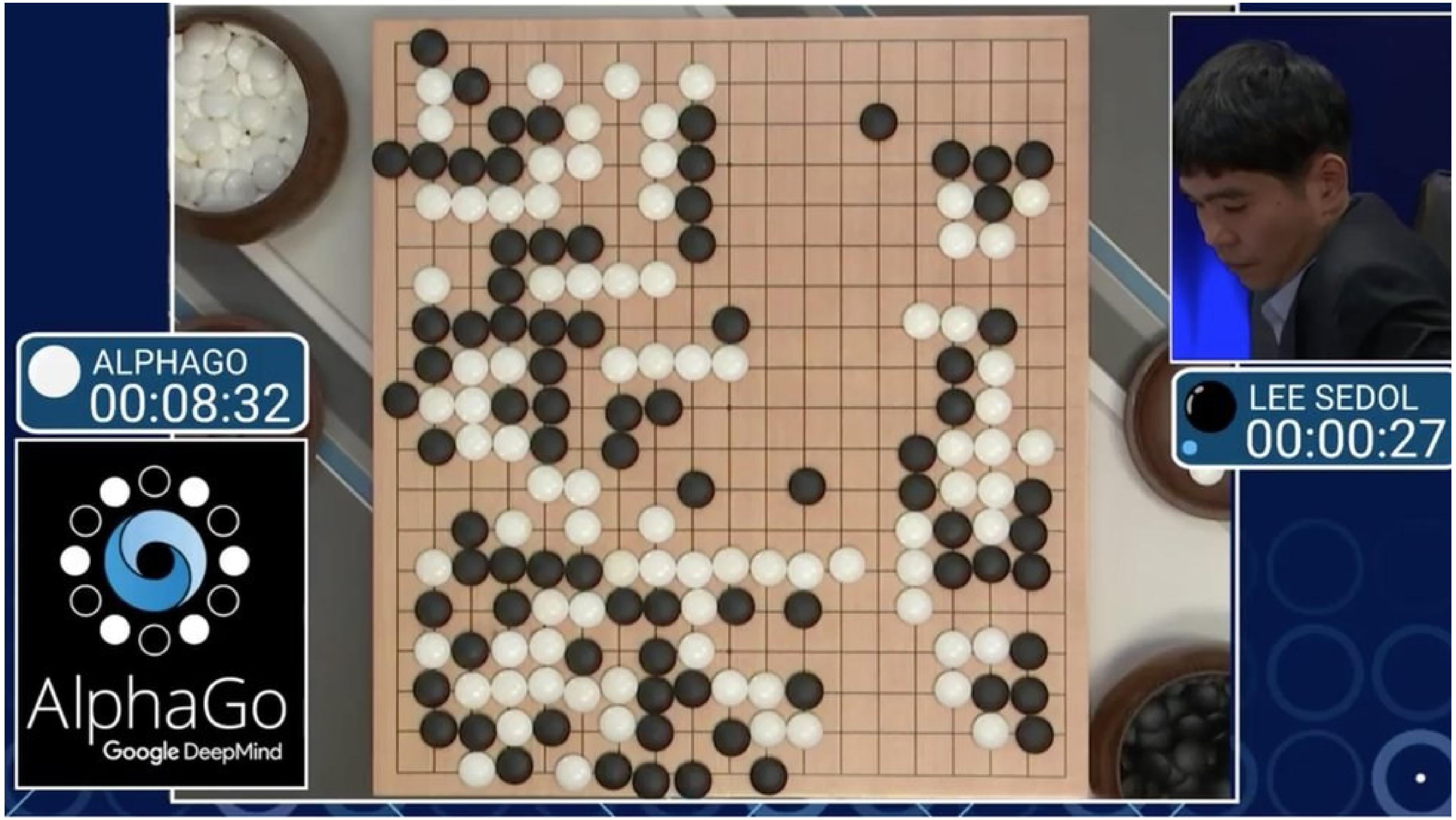
- 1968: Zobrist's program plays legal Go, barely (b>300!)
- 2005-2014: Monte Carlo tree search enables rapid advances: current programs beat strong amateurs, and professionals with a 3-4 stone handicap.
- 2015: AlphaGo from DeepMind beats Lee Sedol

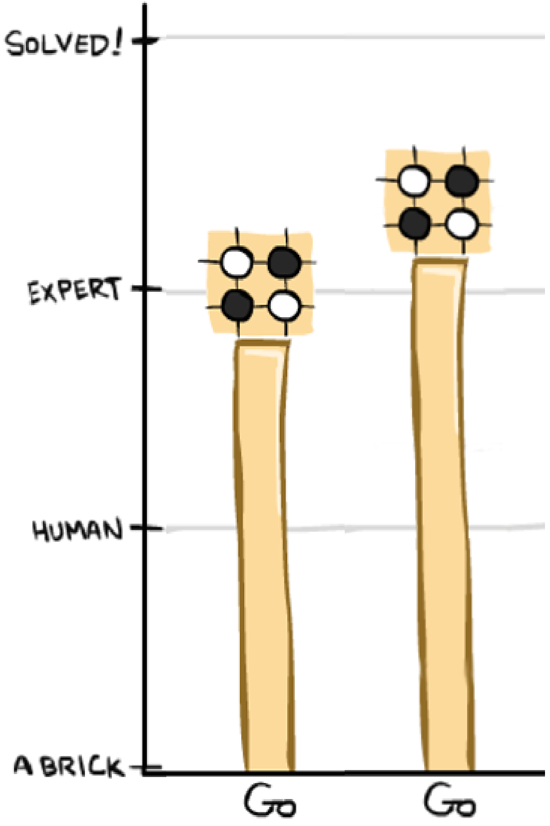


Slide/Image credit: Ketrina Yim, UCB CS 188

AlphaGo

2015: AlphaGo from DeepMind beats Lee Sedol





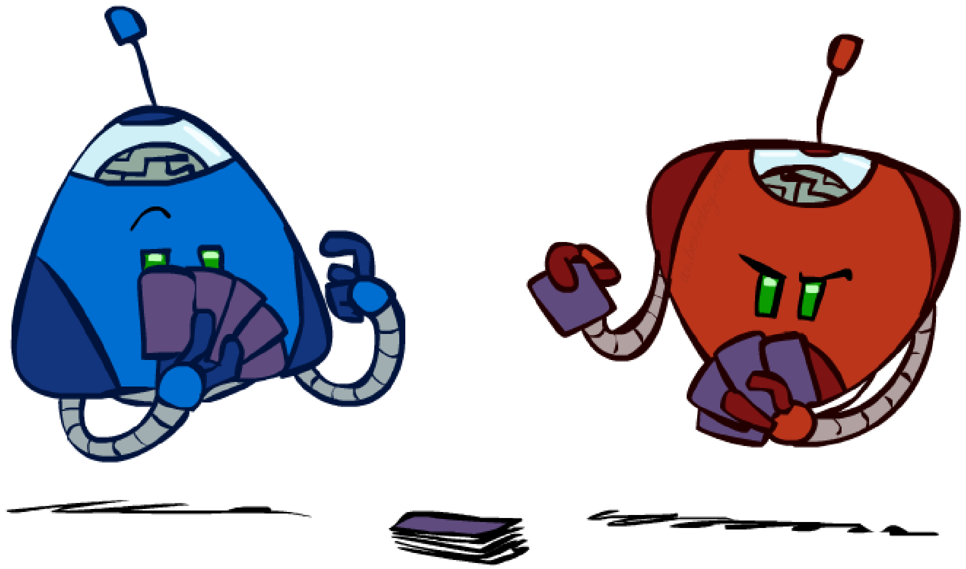
Slide/Image credit: Ketrina Yim, UCB CS 188

Types of Games

Many different kinds of games!

Axes:

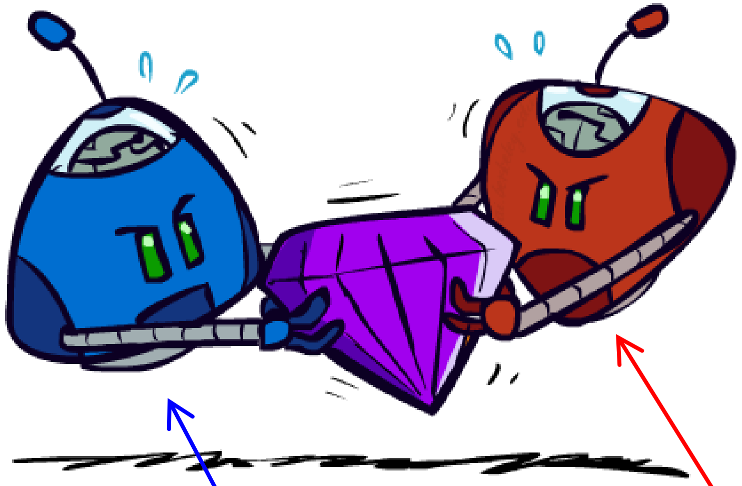
- Deterministic or stochastic?
- Perfect information (fully observable)?
- One, two, or more players?
- Turn-taking or simultaneous?
- Zero sum?

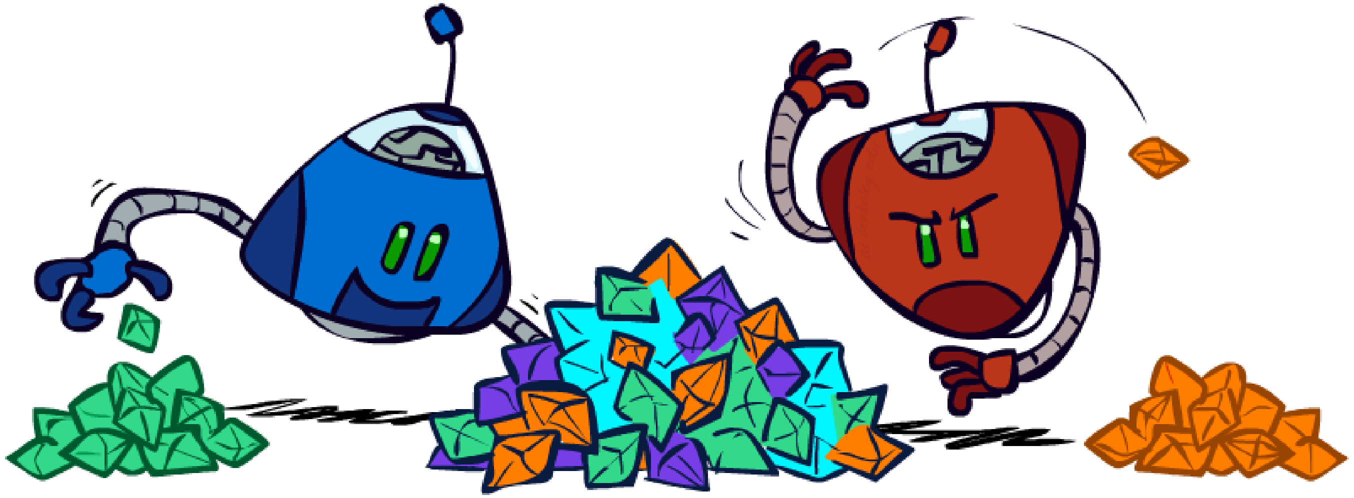


Want algorithms for calculating a ***contingent plan*** (a.k.a. strategy or policy) which recommends a move for every possible eventuality

Image/slide credit: Ketrina Yim, UCB CS 188

Zero-Sum Games





- Zero-Sum Games
  - Agents have ***opposite*** utilities
  - Pure competition:
    - One ***maximizes***, the other ***minimizes***

- General Games
  - Agents have ***independent*** utilities
  - Cooperation, indifference, competition, shifting alliances, and more are all possible

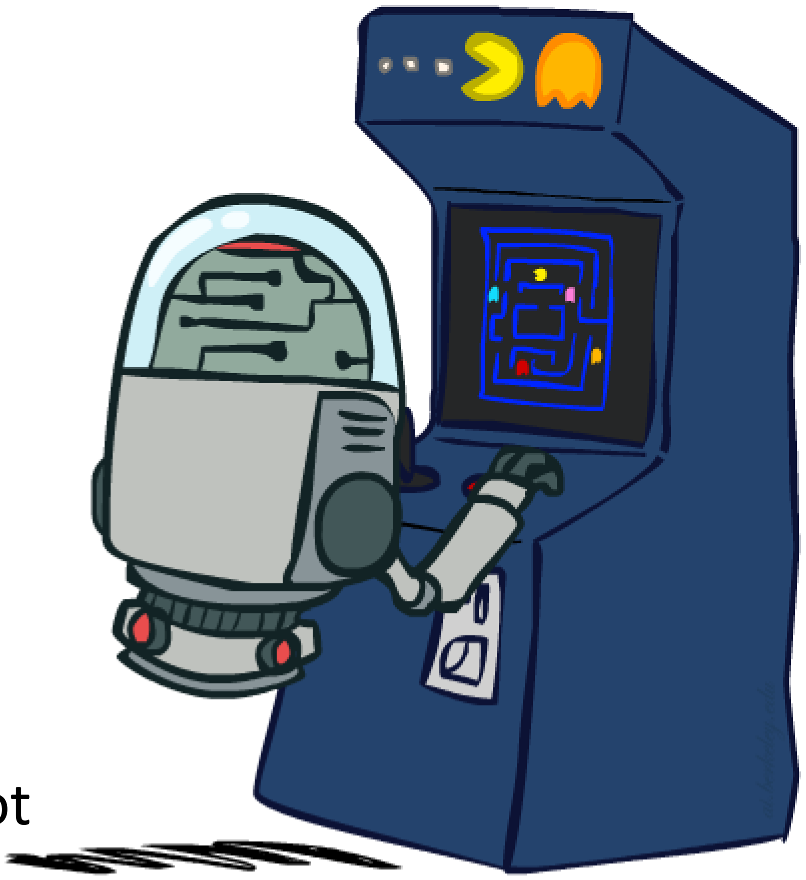
Image/slide credit: Ketrina Yim, UCB CS 188

"Standard" Games

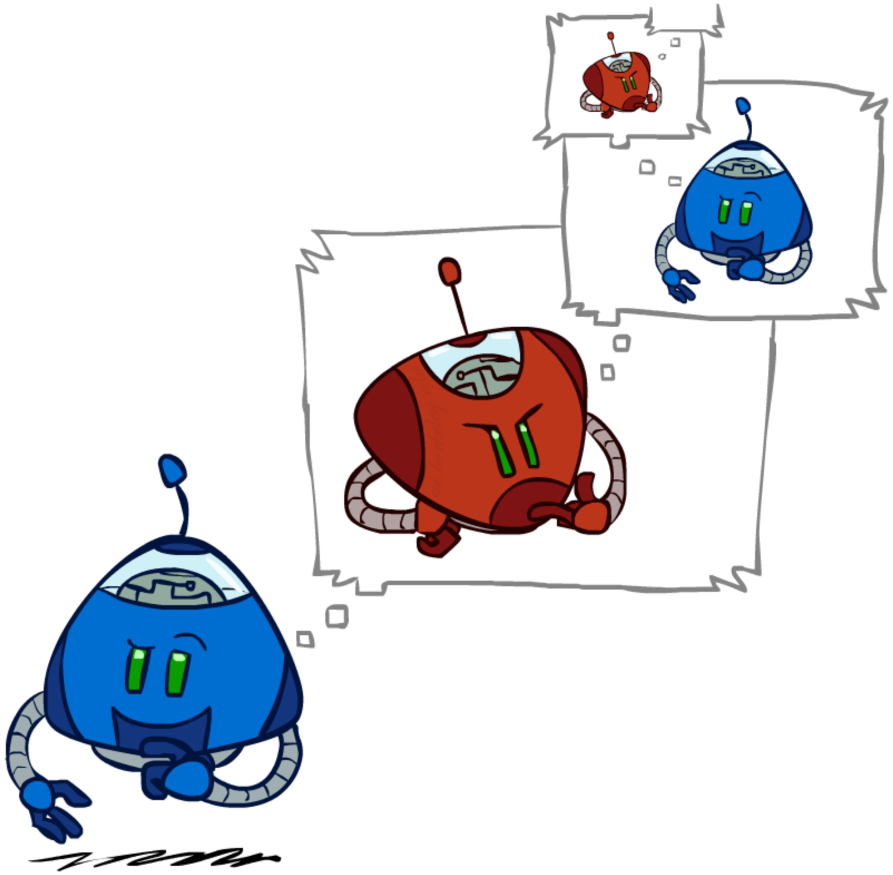
Standard games are deterministic, observable, two-player, turn-taking, zero-sum

Game formulation:

- Initial state: s_0
- Players: Player(s) indicates whose move it is
- Actions: Actions(s) for player on move
- Transition model: Result(s,a)
- Terminal test: Terminal-Test(s)
- Terminal values: Utility(s,p) for player p
  - Or just Utility(s) for player making the decision at root



Image/slide credit: Ketrina Yim, UCB CS 188

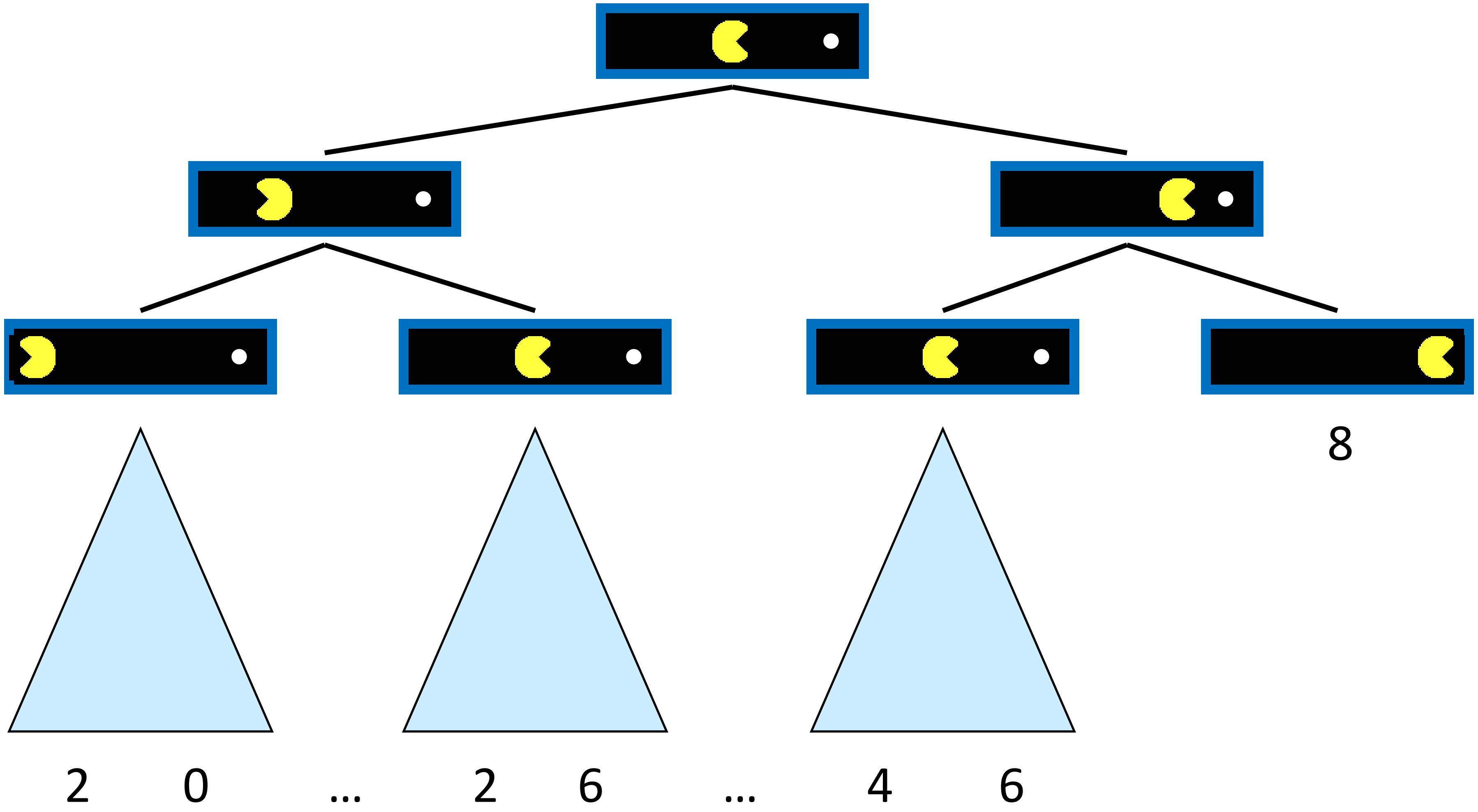


Pre-reading

Minimax Search

Image credit: Ketrina Yim, UCB CS 188

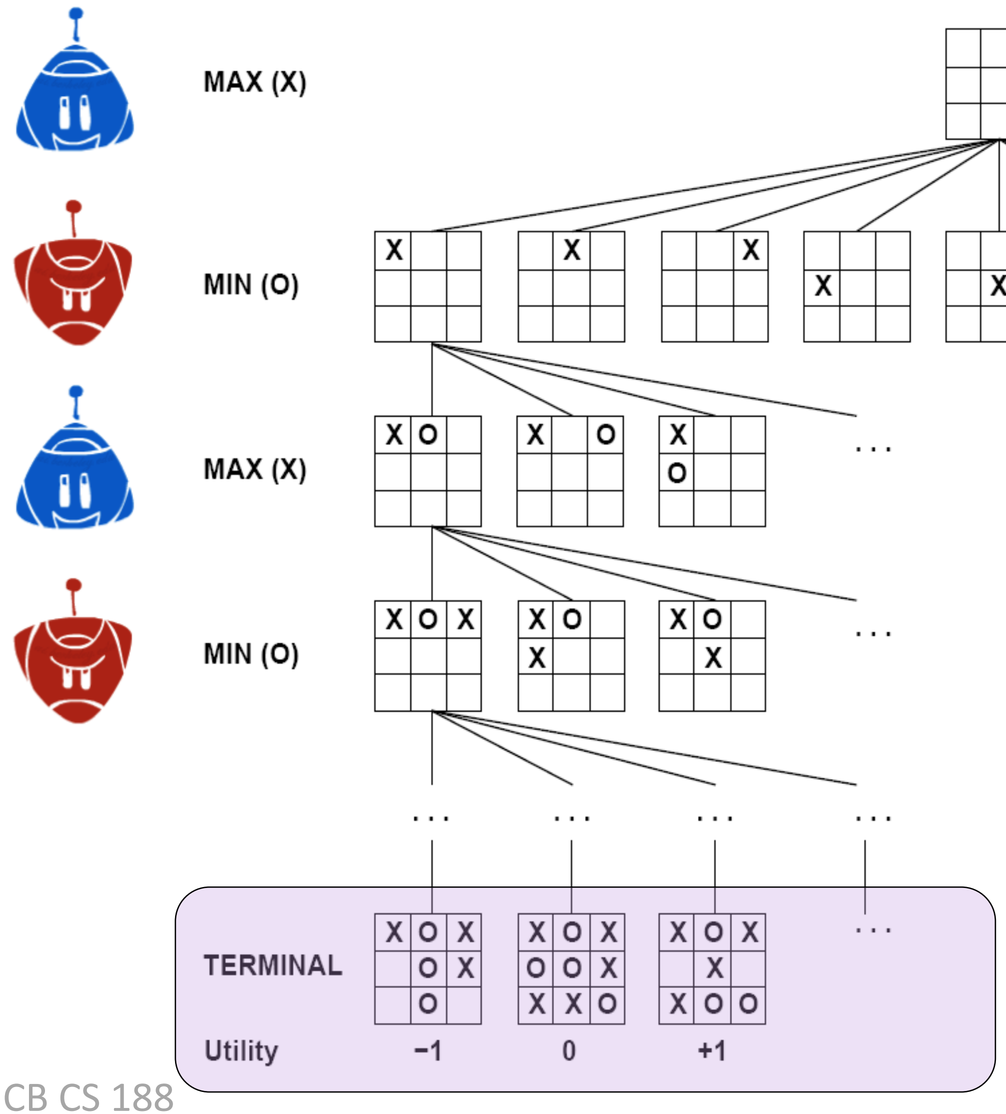
Single-Agent Trees

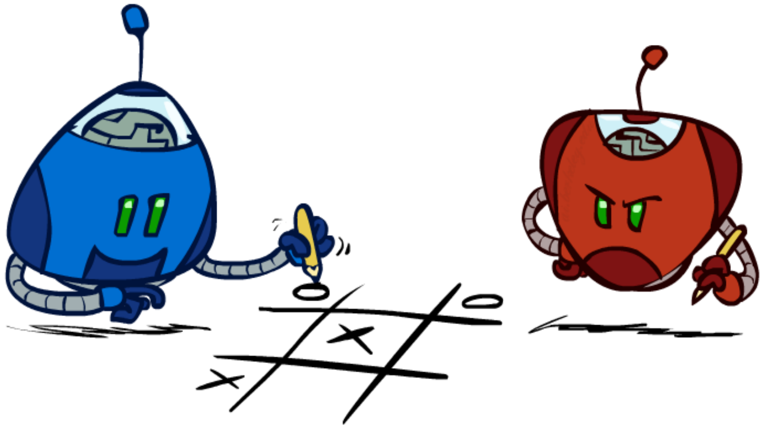


Slide credit: UCB CS 188

Minimax

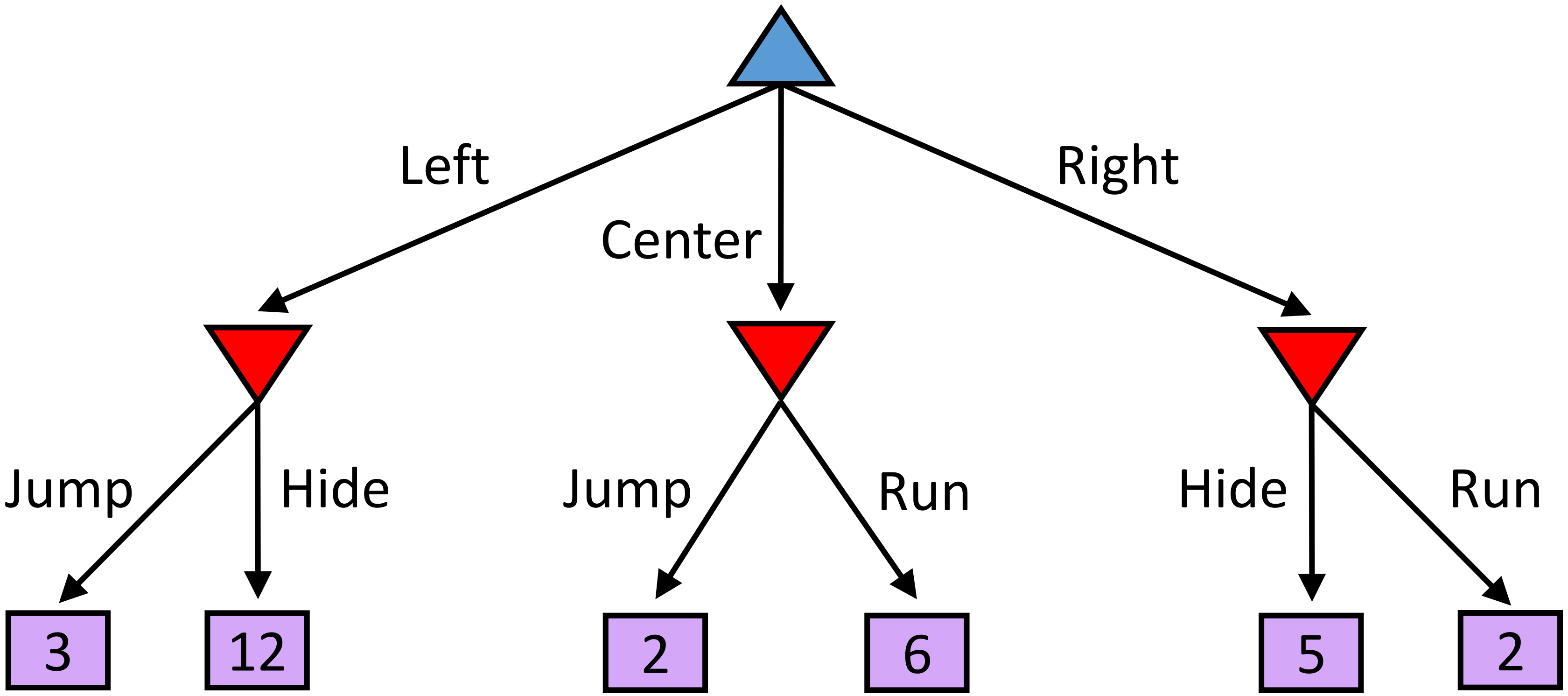
- States
- Actions
- Values





Image/slide credit: Ketrina Yim, UCB CS 188

Minimax Example



Generic Game Tree Pseudocode

```
function  minimax_decision( state )
    return  argmax a in state.actions  value( state.result(a) )

function  value( state )
    if  state.is_leaf
        return state.value

    if  state.player  is  MAX
        return  max a in state.actions  value( state.result(a) )

    if  state.player  is  MIN
        return  min a in state.actions  value( state.result(a) )
```

Slide credit: UCB CS 188

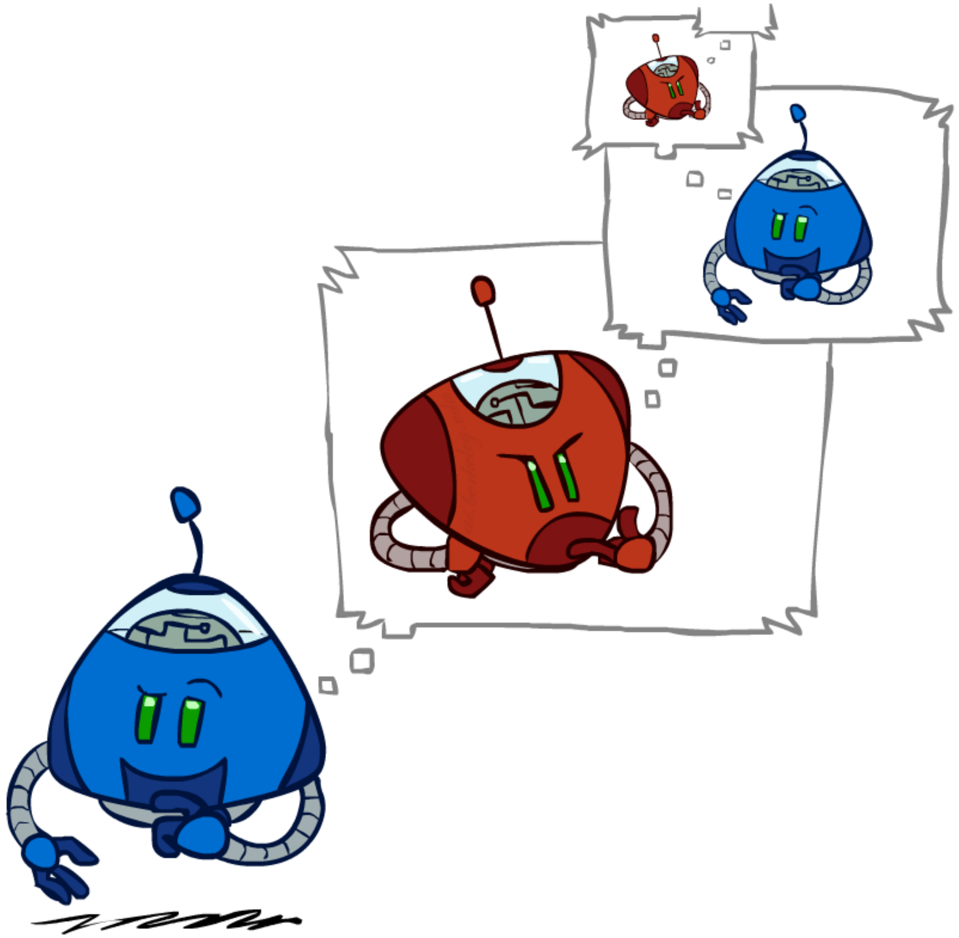
Minimax Efficiency

How efficient is minimax?

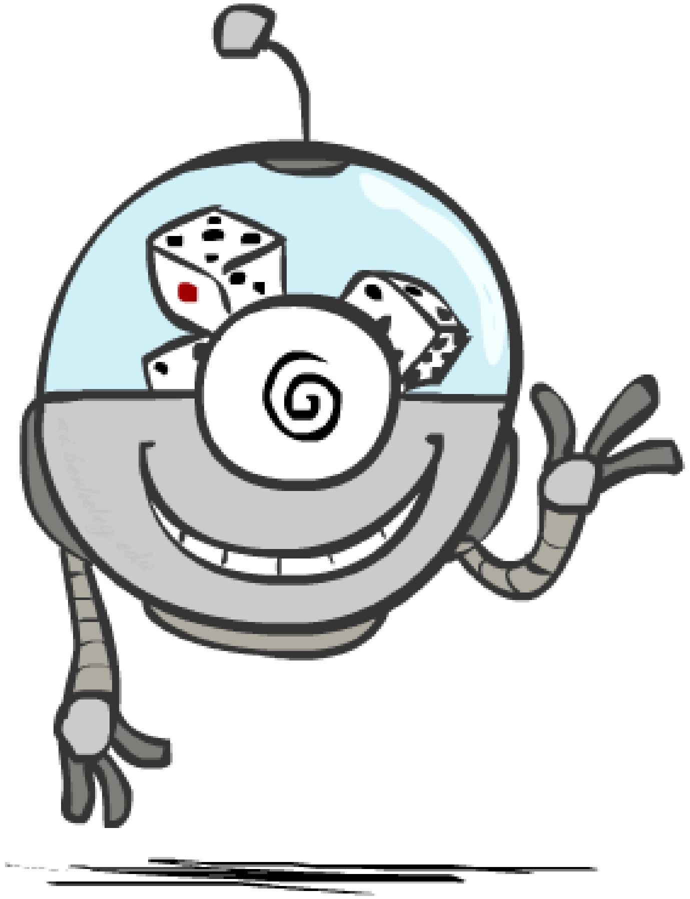
- Just like (exhaustive) DFS
- Time: $O(b^m)$
- Space: $O(bm)$

Example: For chess, $b \approx 35$, $m \approx 100$

- Exact solution is completely infeasible
- Humans can’t do this either, so how do we play chess?
- Bounded rationality – Herbert Simon



Image/slide credit: Ketrina Yim, UCB CS 188



Pre-reading

Probability and Expected Value

Image credit: Ketrina Yim, UCB CS 188

Probabilities

A random variable represents an event whose outcome is unknown

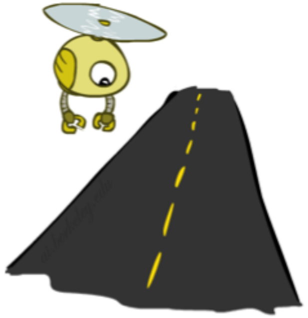
A probability distribution is an assignment of weights to outcomes

Example: Traffic on freeway

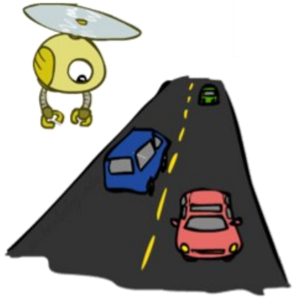
- Random variable: T = whether there's traffic
- Outcomes: T in {none, light, heavy}
- Distribution:

  P(T=none) = 0.25, P(T=light) = 0.50, P(T=heavy) = 0.25

Probabilities over all possible outcomes sum to one



0.25



0.50

0.25

Image/slide credit: Ketrina Yim, UCB CS 188

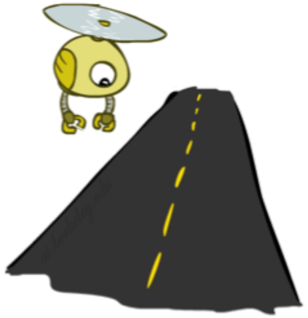
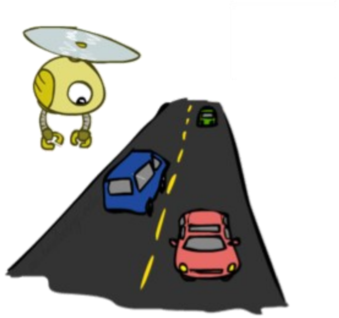
Expected Value

Expected value of a function of a random variable:

Average the values of each outcome,
weighted by the probability of that outcome

Example: How long to get to the airport?

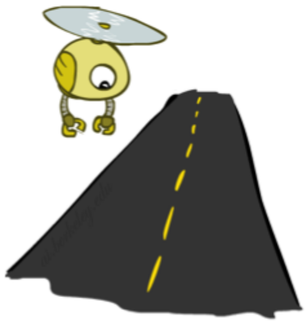
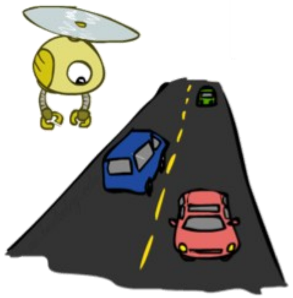
Time:	20 min		30 min		60 min		
	x	+	x	+	x	⇒	35 min
Probability:	0.25		0.50		0.25		

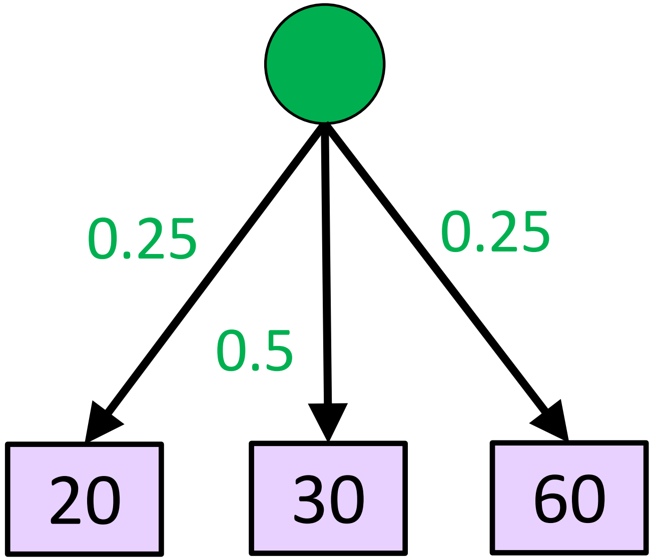




Image/slide credit: Ketrina Yim, UCB CS 188

Expected Value

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

Max node notation

$$V(s) = \max_a V(s'),$$

where $s' = result(s, a)$

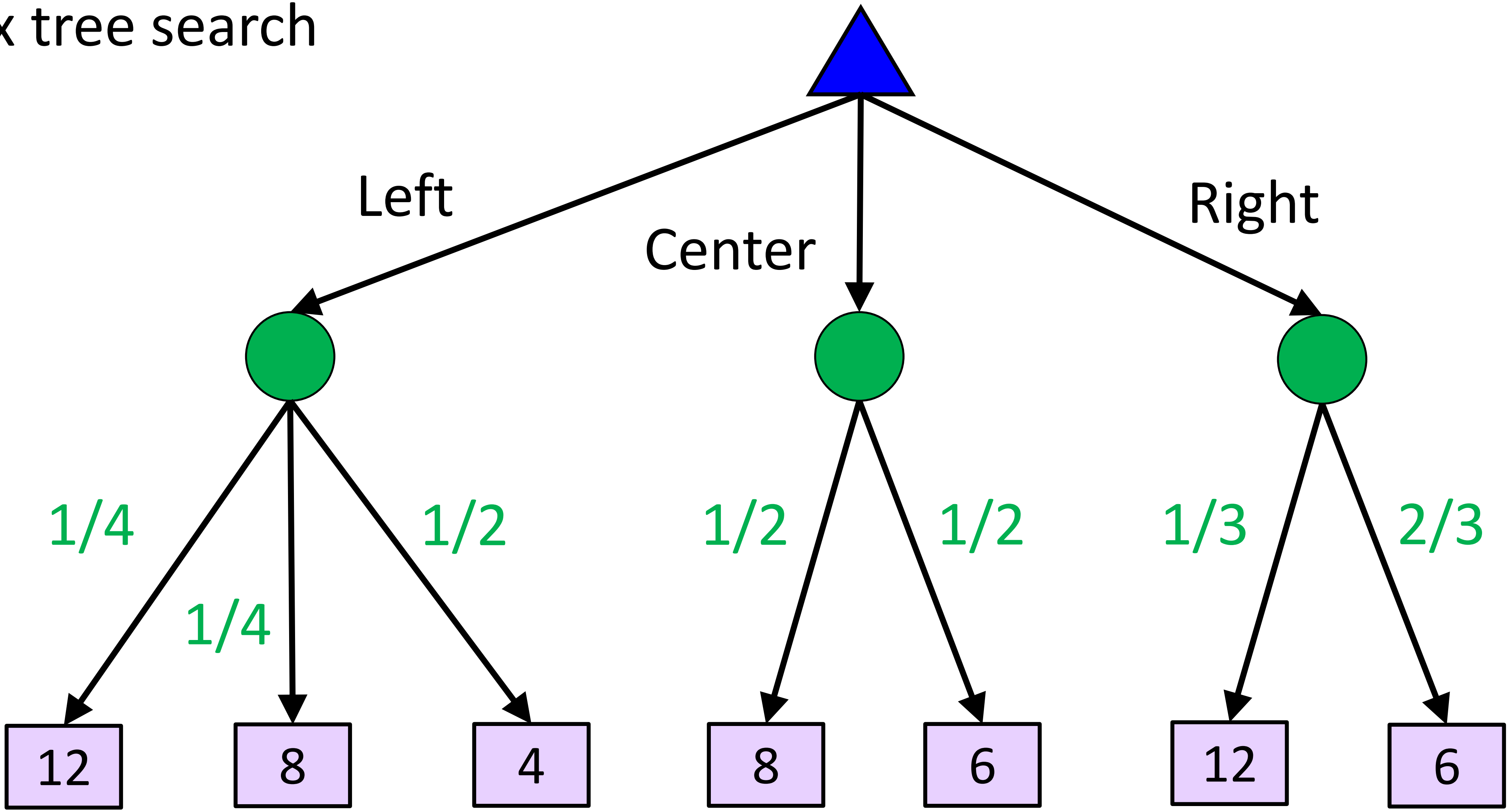
Chance node notation

$$V(s) = \sum_{s'} P(s')\, V(s')$$

Image credit: Ketrina Yim, UCB CS 188

Example

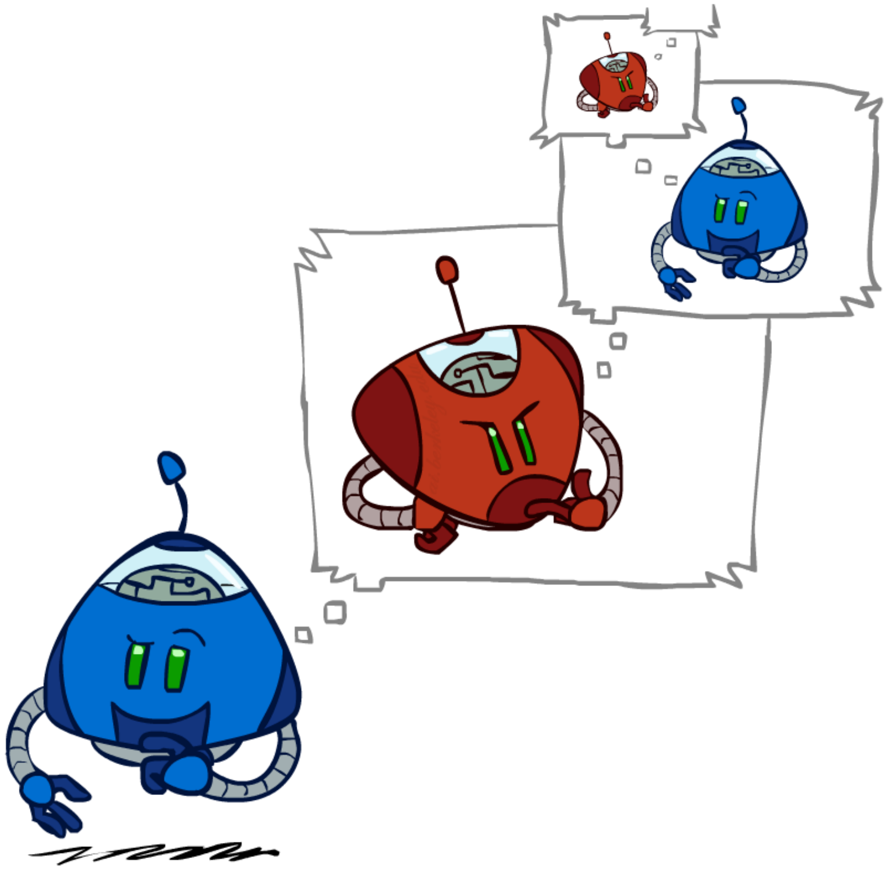
Expectimax tree search



Slide credit: UCB CS 188

Updated: Generic Game Tree Pseudocode

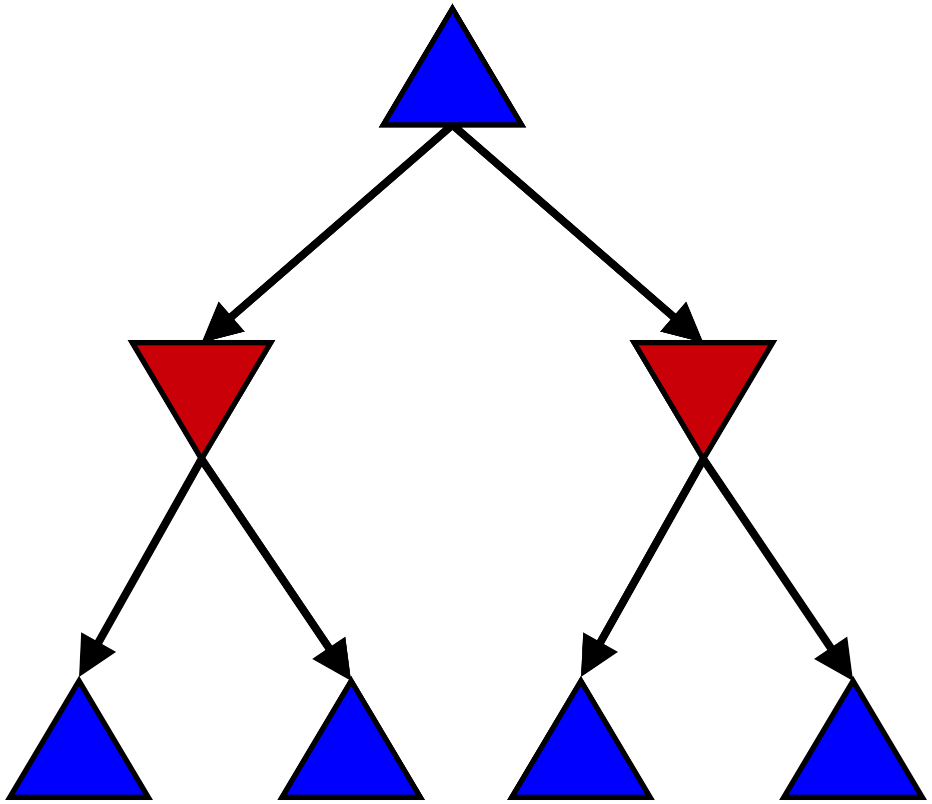
```
function value( state )
    if state.is_leaf
        return state.value

    if state.player is MAX
        return max_{a in state.actions} value( state.result(a) )

    if state.player is MIN
        return min_{a in state.actions} value( state.result(a) )

    if state.player is CHANCE
        return sum_{s in state.next_states} P( s ) * value( s )
```



Adversarial Search

Image credit: Ketrina Yim, UCB CS 188

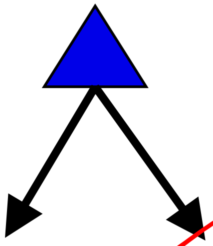
Minimax Code and Notation



$a \in A$

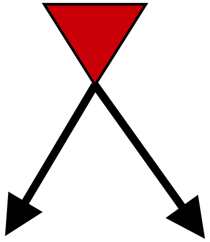
Minimax Notation

```
def max_value(state):

    if state.is_leaf:
        return state.value
    # TODO Also handle depth limit

    best_value = -10000000

    for action in state.actions:
        next_state = state.result(action)

        next_value = min_value(next_state)

        if next_value > best_value:
            best_value = next_value

    return best_value

def min_value(state):
```

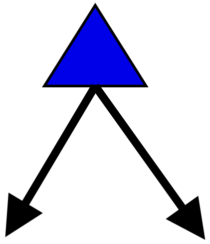


$$V(s) = \max_{a} V(s'),$$

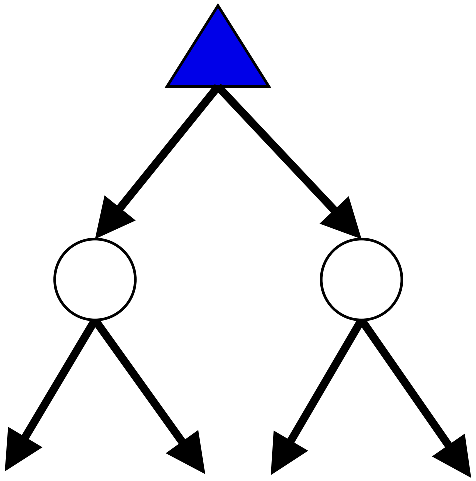
where $s' = result(s,a)$



Minimax Notation



$$V(s) = \max_{a} V(s'),$$
where $s' = result(s, a)$



$$\hat{a} = \operatorname*{argmax}_{a} V(s'),$$
where $s' = result(s, a)$

a for act in actions

$\hat{a}$ return best_act

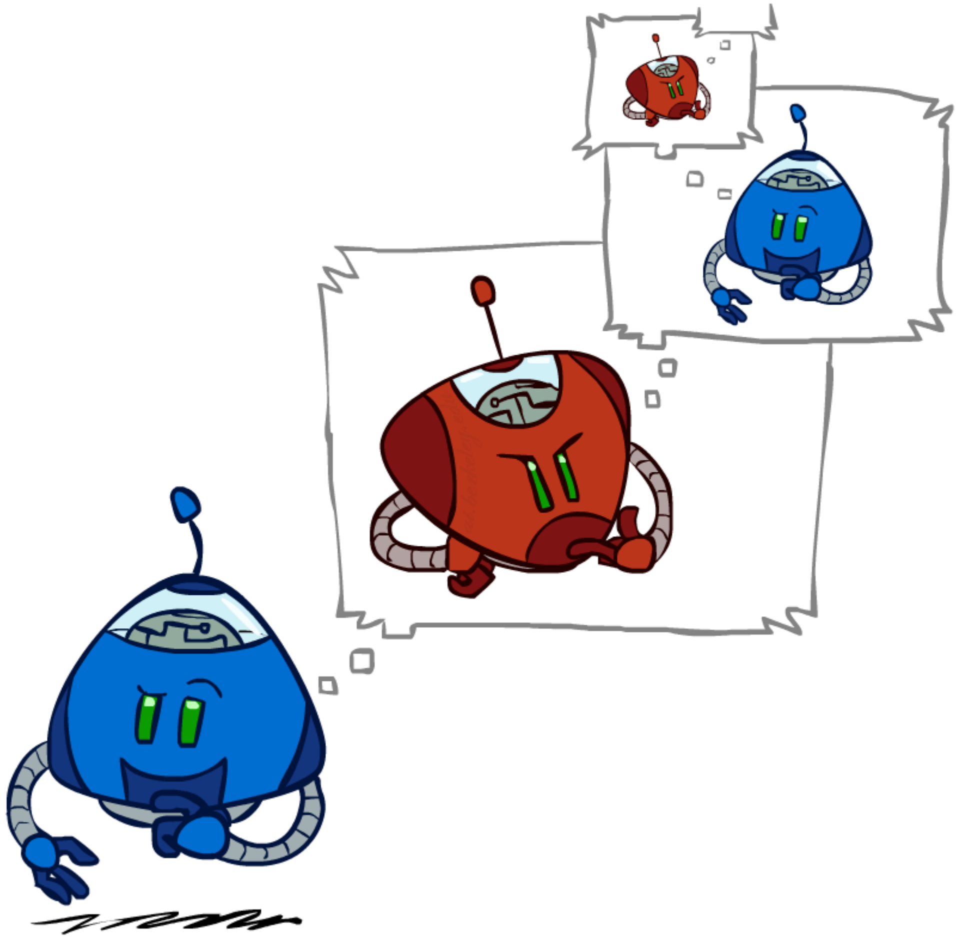
Minimax Efficiency

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
- Humans can't do this either, so how do we play chess?
- Bounded rationality – Herbert Simon



Image/slide credit: Ketrina Yim, UCB CS 188

Resource Limits

Image/slide credit: Ketrina Yim, UCB CS 188

Resource Limits

Problem: In realistic games, cannot search to leaves!

Solution 1: Bounded lookahead

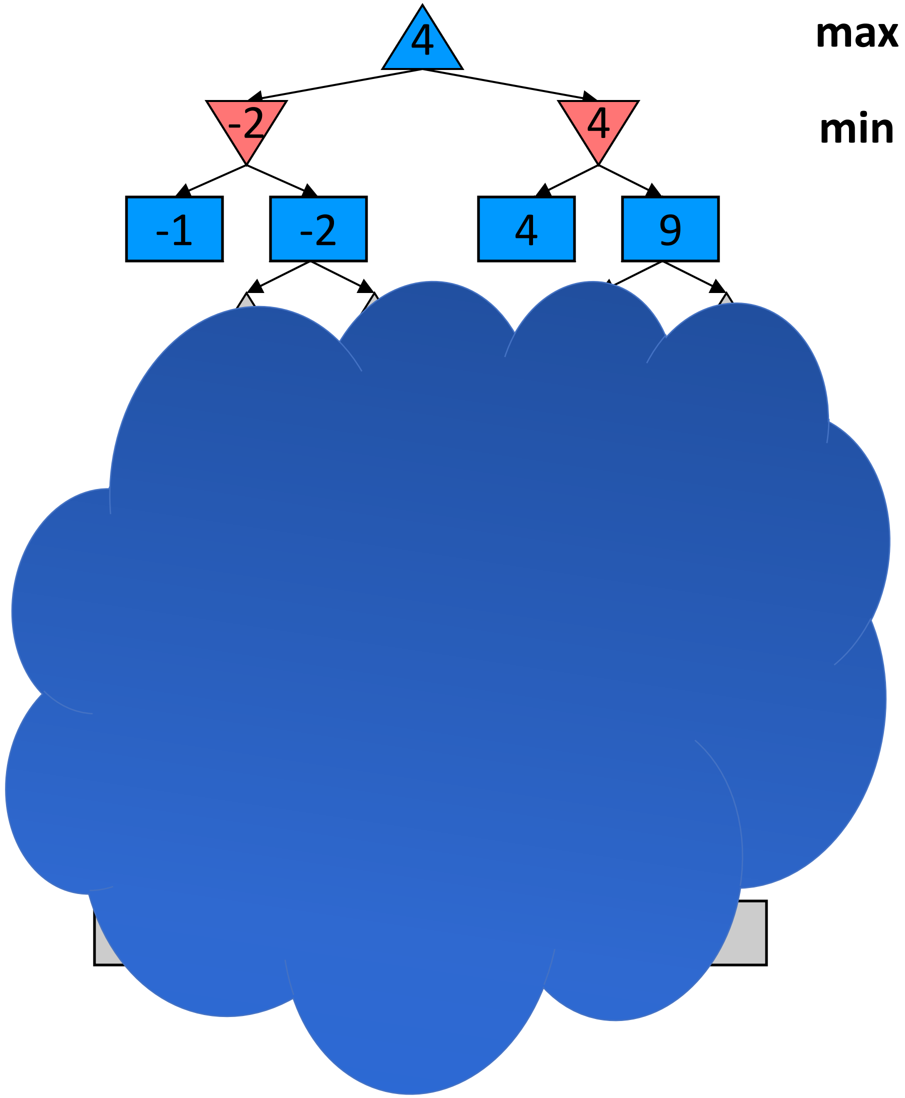
- Search only to a preset ***depth limit*** or ***horizon***
- Use an ***evaluation function*** for non-terminal positions

Guarantee of optimal play is gone

More plies make a BIG difference

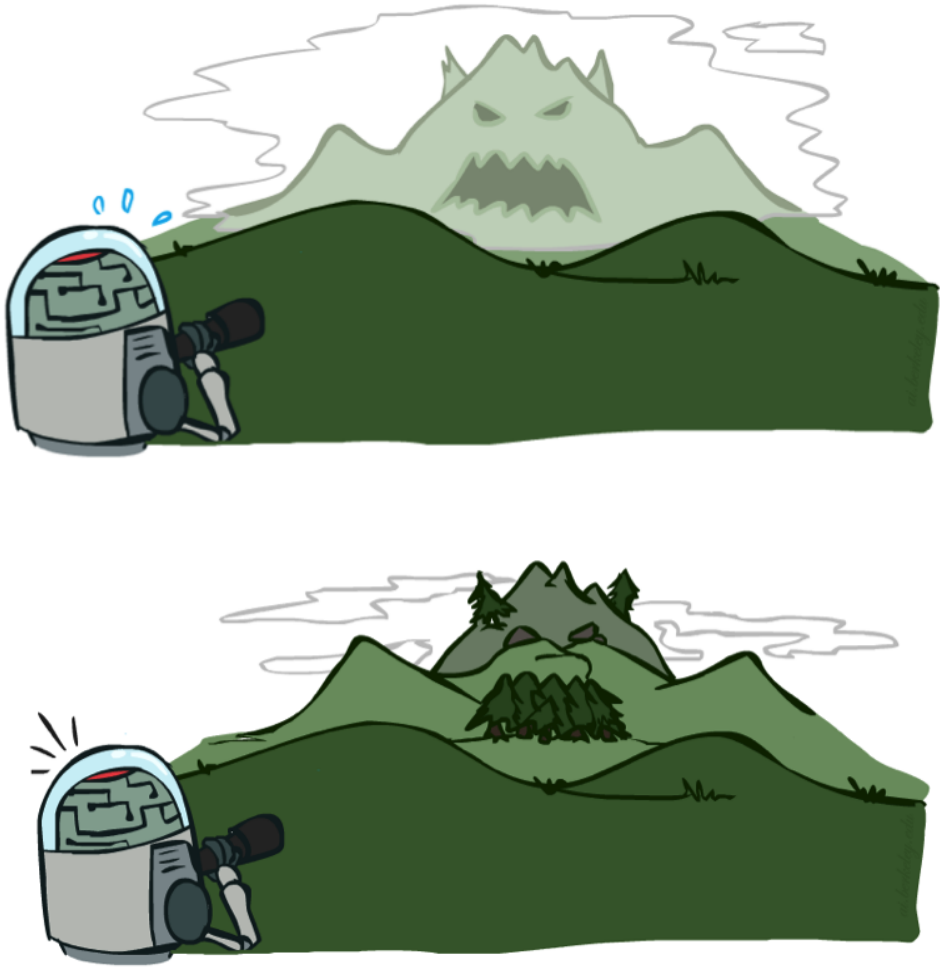
Example:

- Suppose we have 100 seconds, can explore 10K nodes / sec
- So can check 1M nodes per move
- For chess, b=~35 so reaches about depth 4 – not so good



Depth Matters

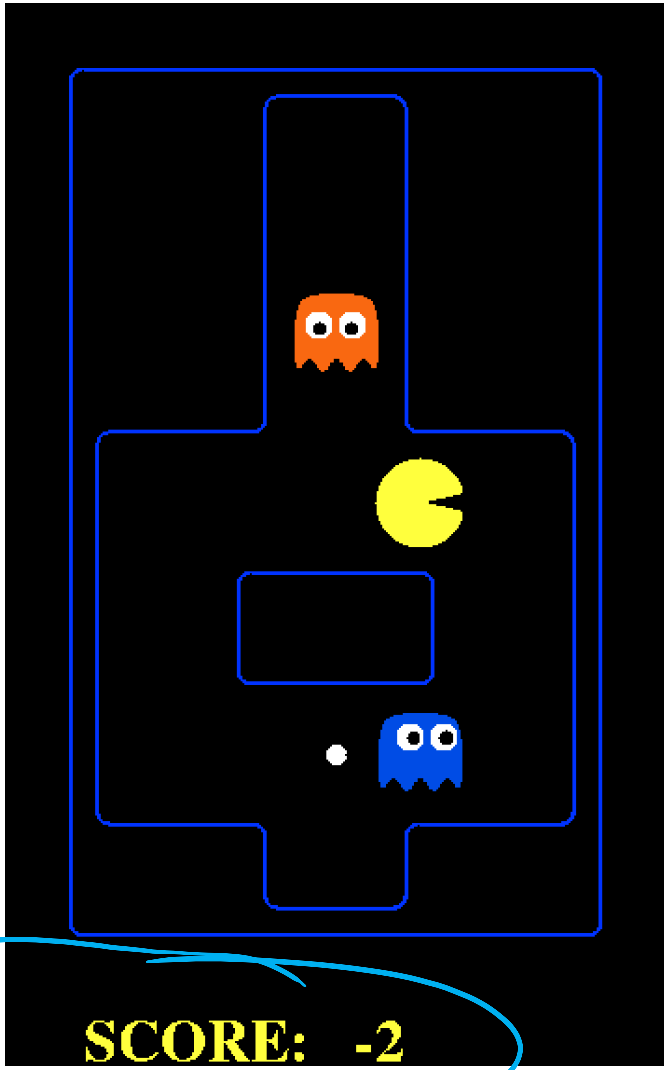
- Evaluation functions are always imperfect
- Deeper search => better play (usually)
- Or, deeper search gives same quality of play with a less accurate evaluation function
- An important example of the tradeoff between complexity of features and complexity of computation



Image/slide credit: Ketrina Yim, UCB CS 188

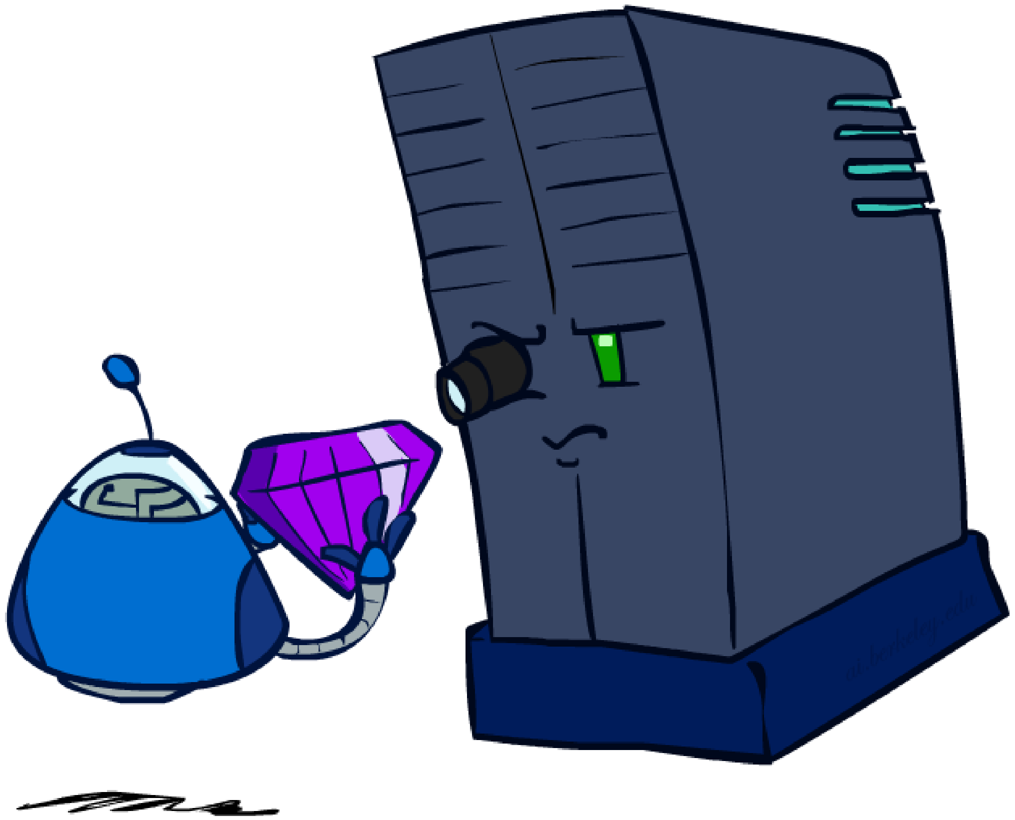
[Demo: depth limited (L6D4, L6D5)]

Demo Limited Depth (2 vs 10)



Slide credit: UCB CS 188

[Demo: depth limited (L6D4, L6D5)]

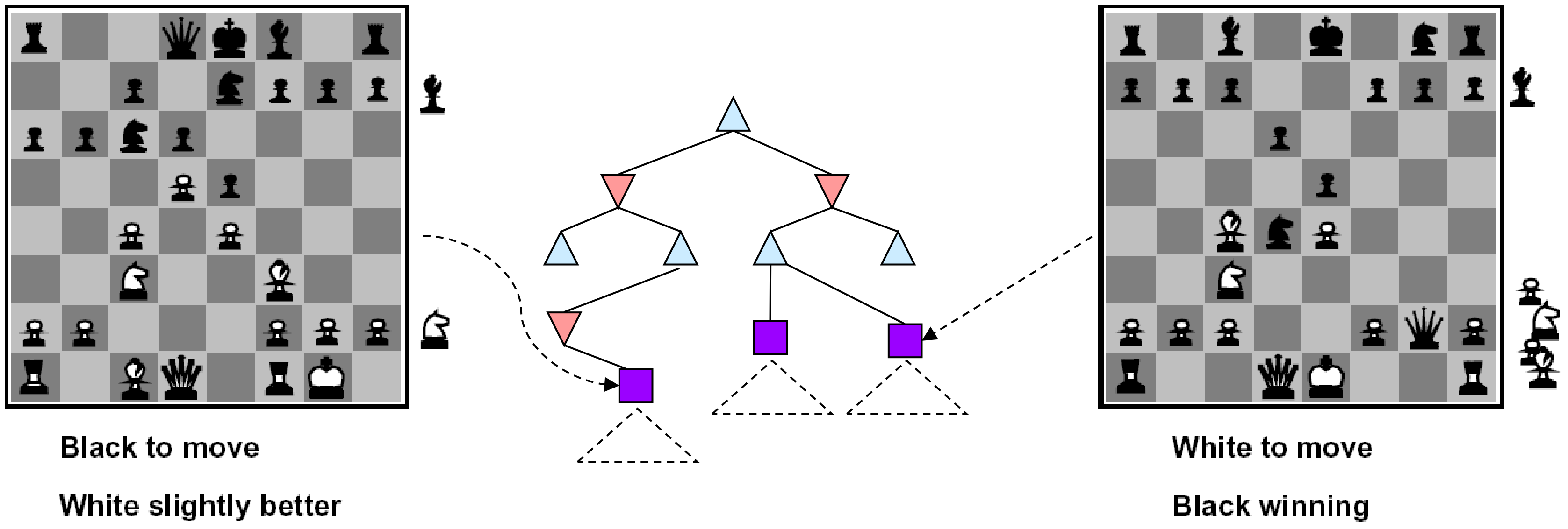


Evaluation Functions

Image/slide credit: Ketrina Yim, UCB CS 188

Evaluation Functions

Evaluation functions score non-terminals in depth-limited search



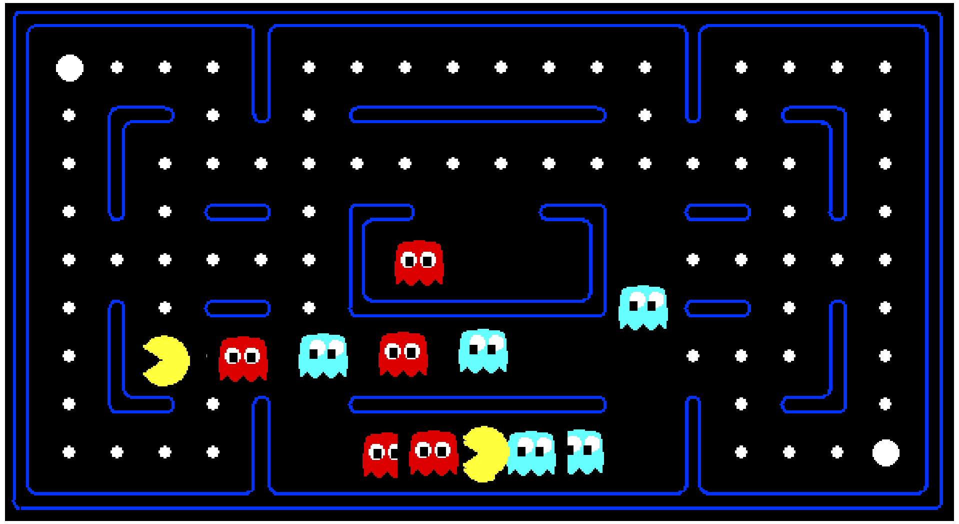
Ideal function: returns the actual minimax value of the position

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., $w_1 = 9$, $f_1(s)$ = (num white queens – num black queens), etc.

Slide credit: UCB CS 188

Evaluation for Pacman



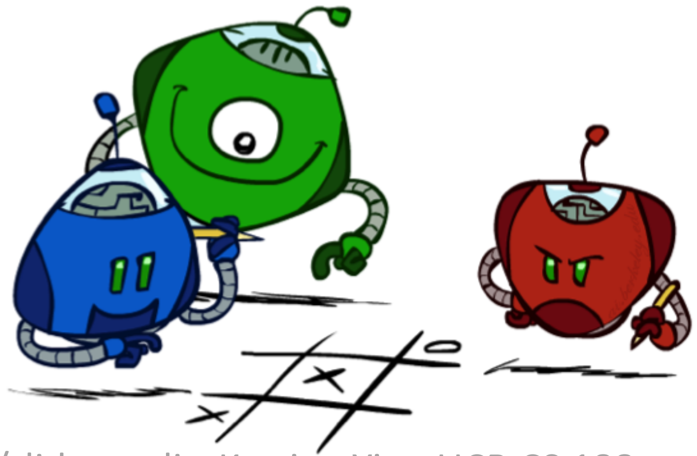
Slide credit: UCB CS 188

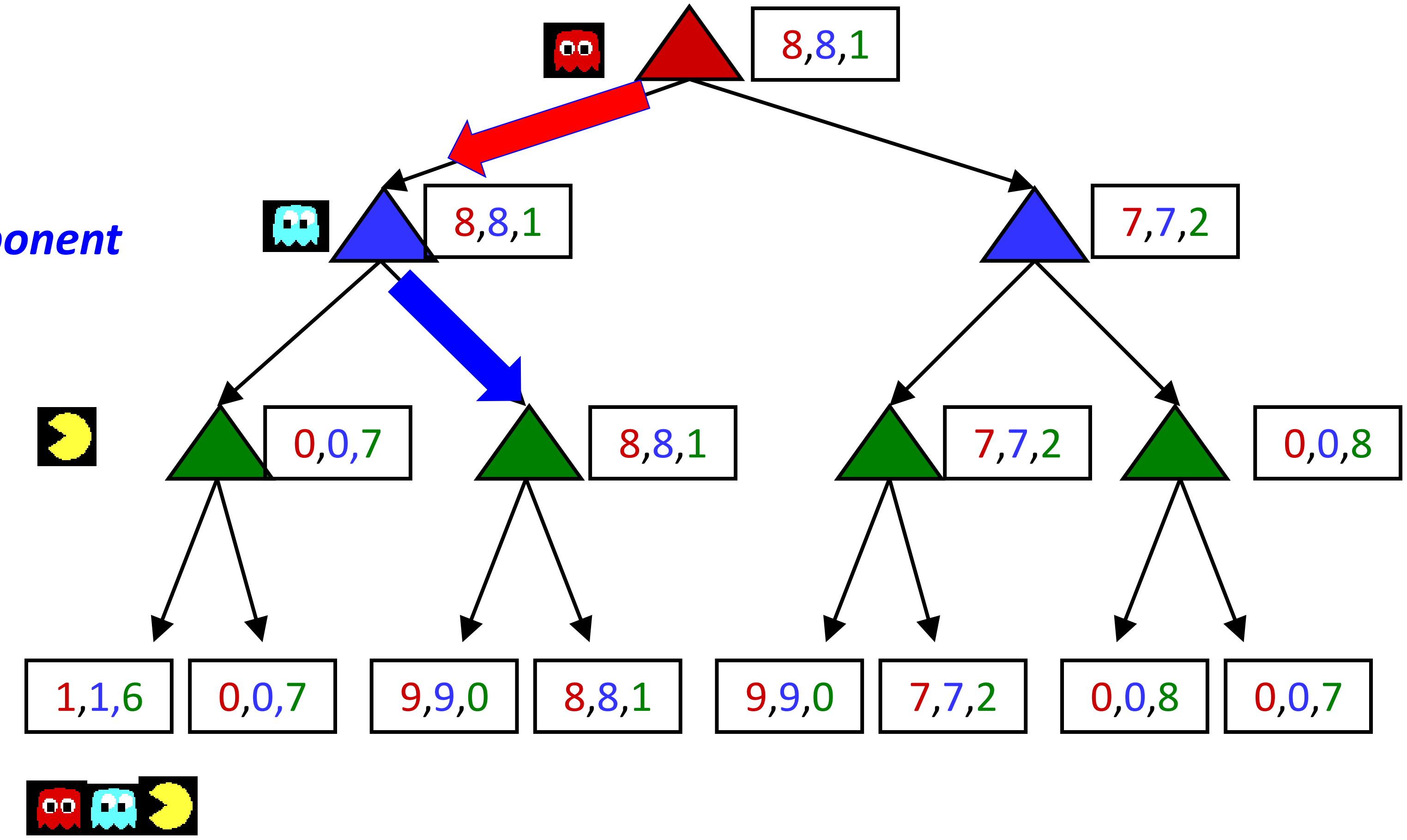
Generalized minimax

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

Generalization of minimax:

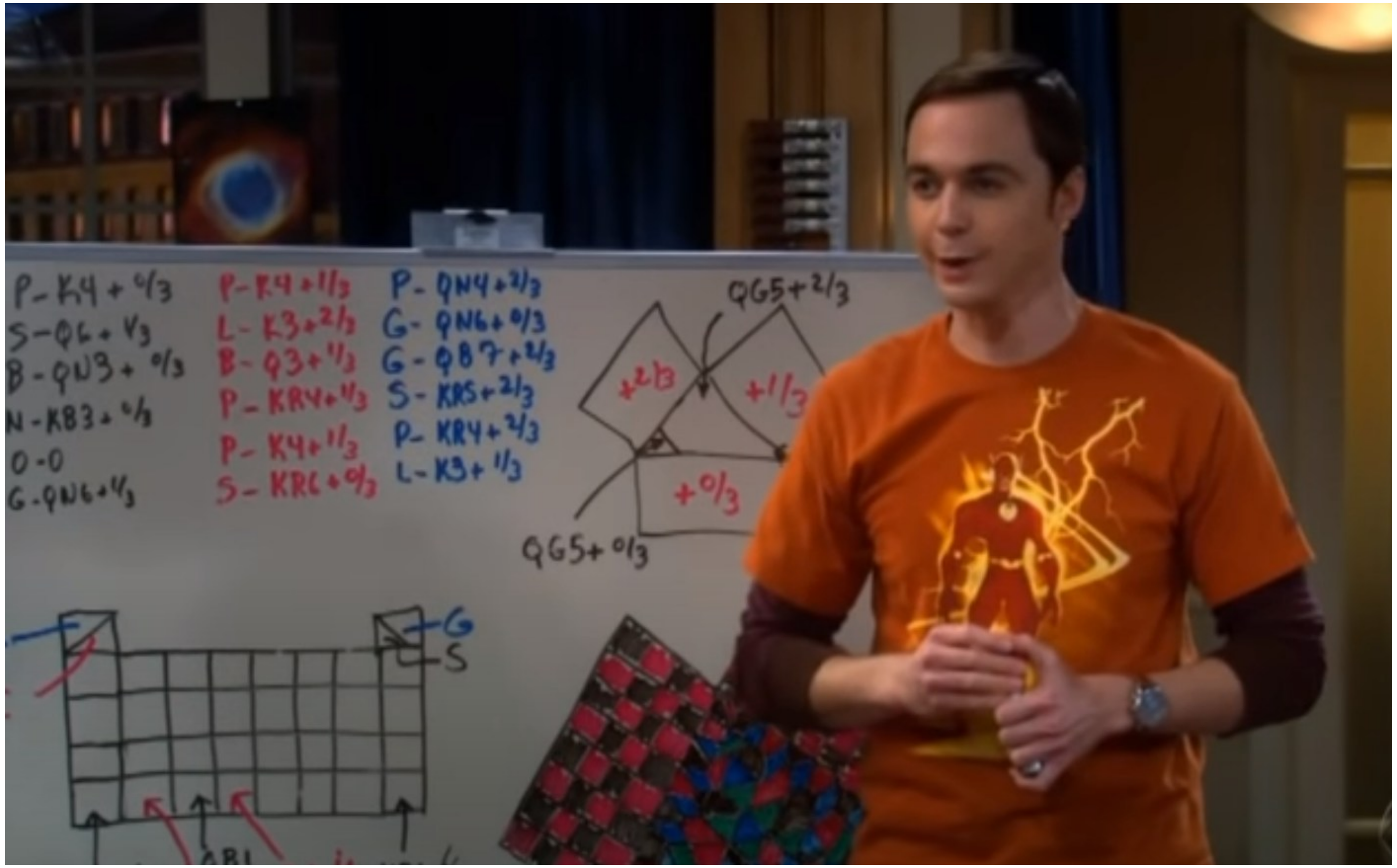
- Terminals have ***utility tuples***
- Node values are also utility tuples
- ***Each player maximizes its own component***
- Can give rise to cooperation and competition dynamically…





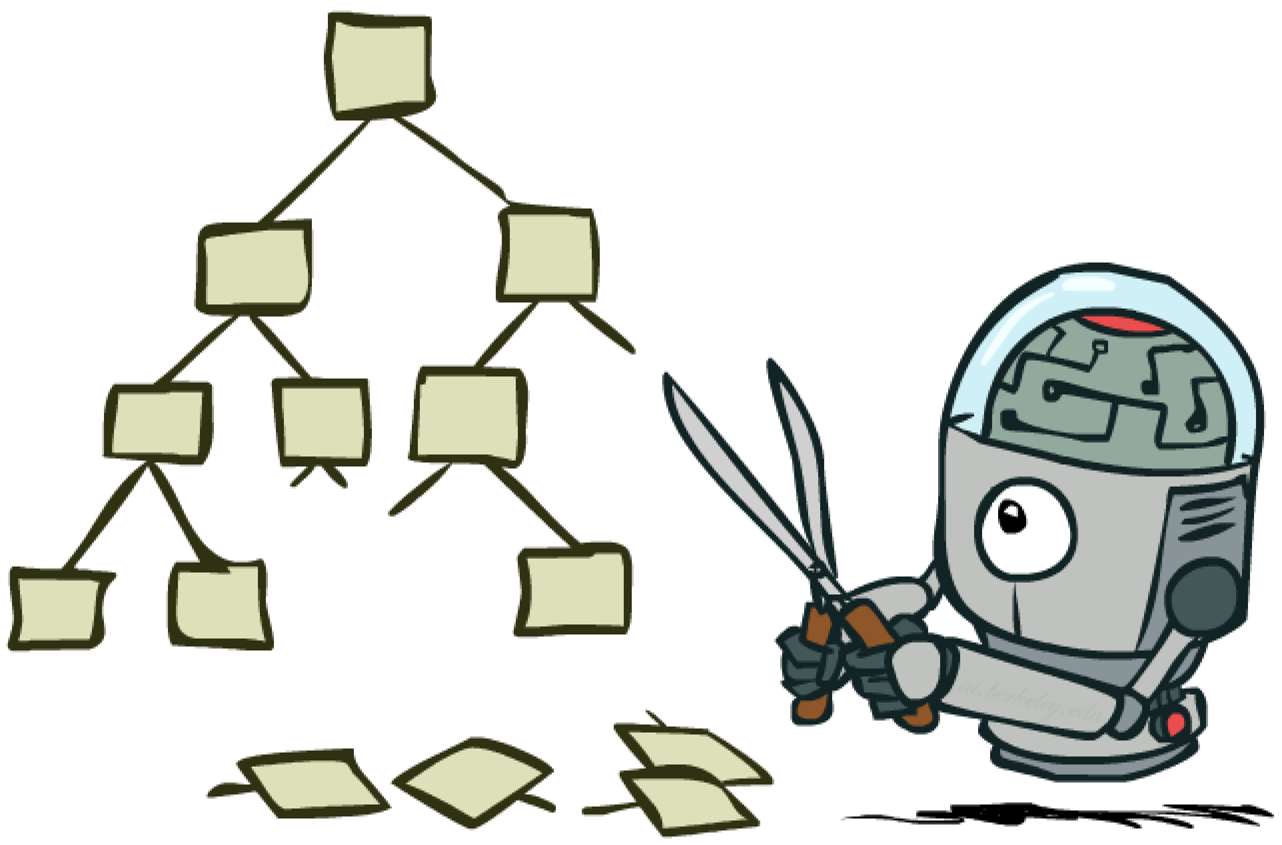
Image/slide credit: Ketrina Yim, UCB CS 188

Generalized minimax



[Three Person Chess](https://www.youtube.com/watch?v=HHVPutfveVs)

https://www.youtube.com/watch?v=HHVPutfveVs

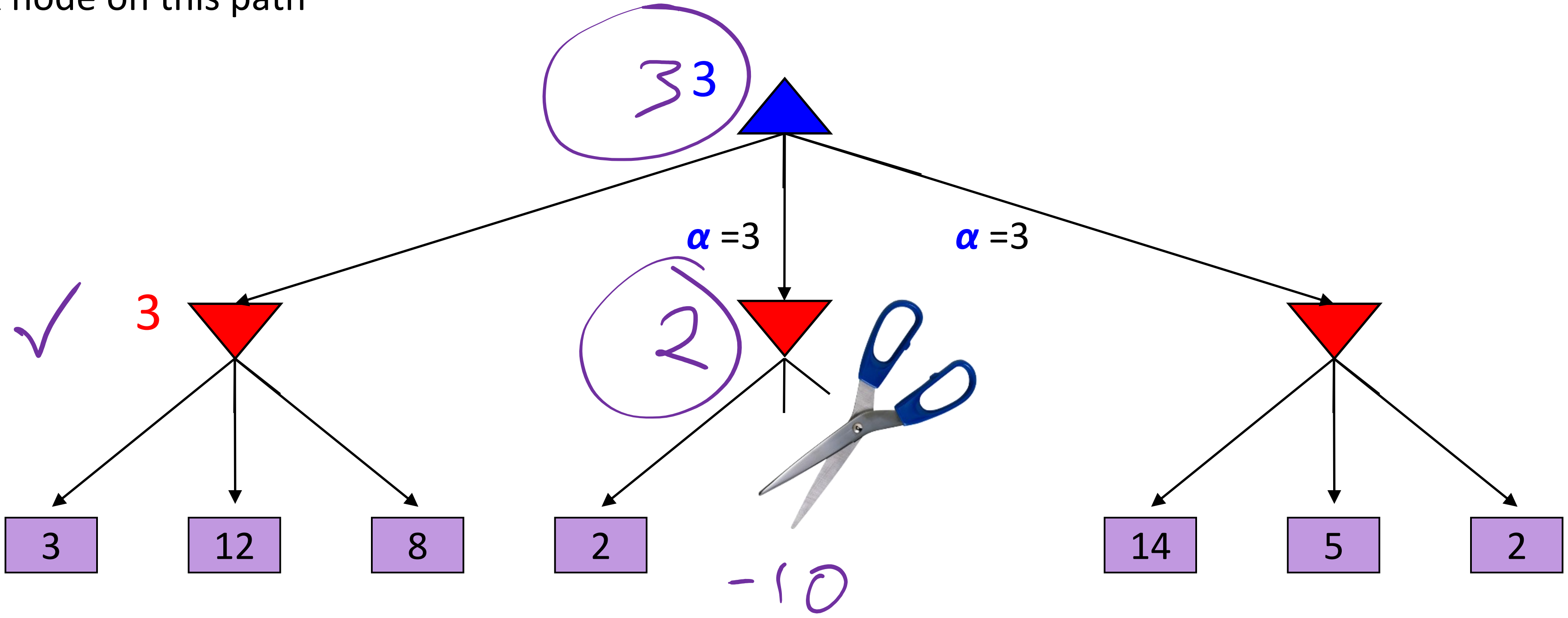


Game Tree Pruning

Image/slide credit: Ketrina Yim, UCB CS 188

Alpha-Beta Example

α = best option so far from any MAX node on this path



1

-10 20008

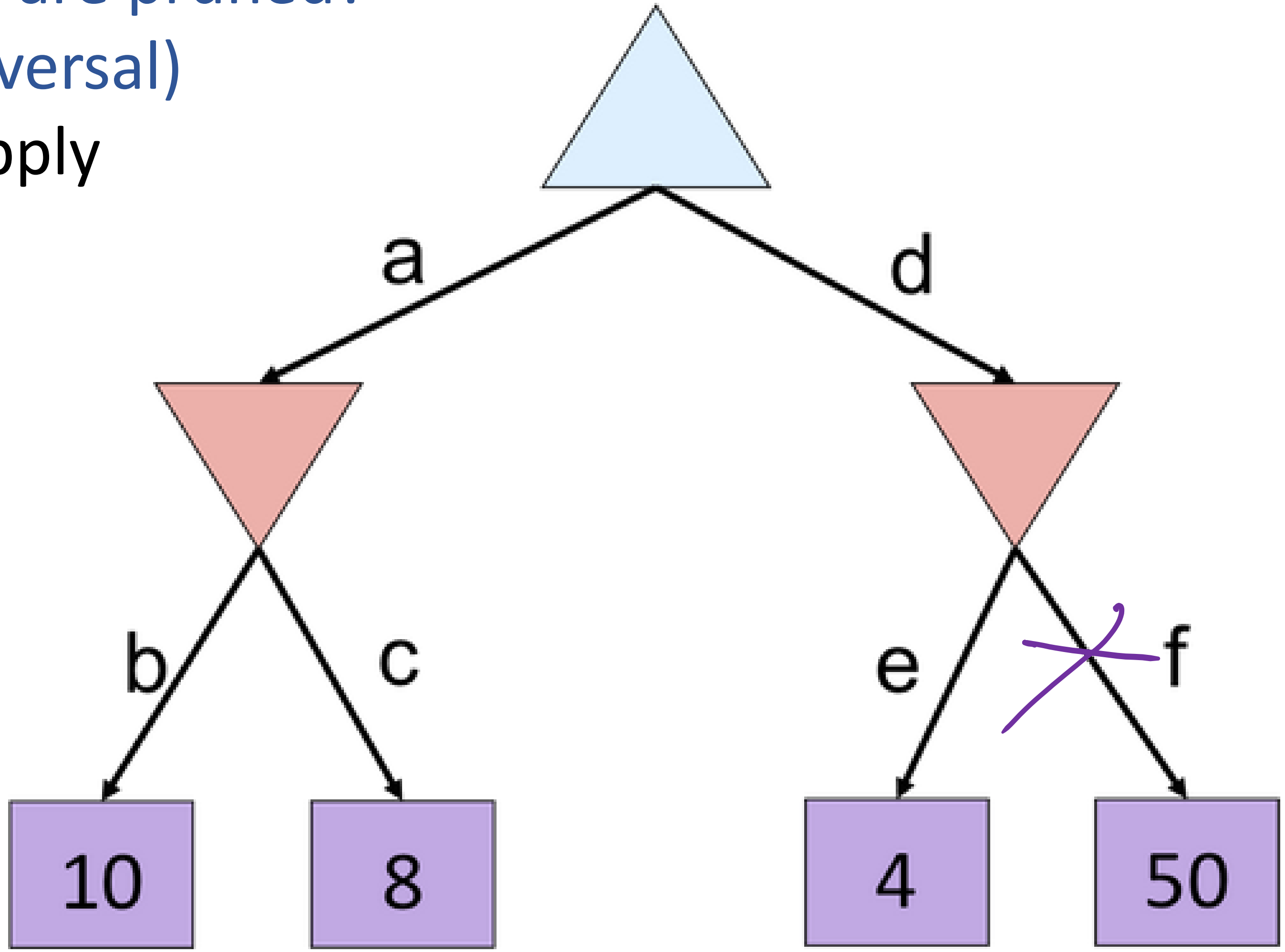
The order of generation matters: more pruning is possible if good moves come first

Slide credit: UCB CS 188

Poll 1

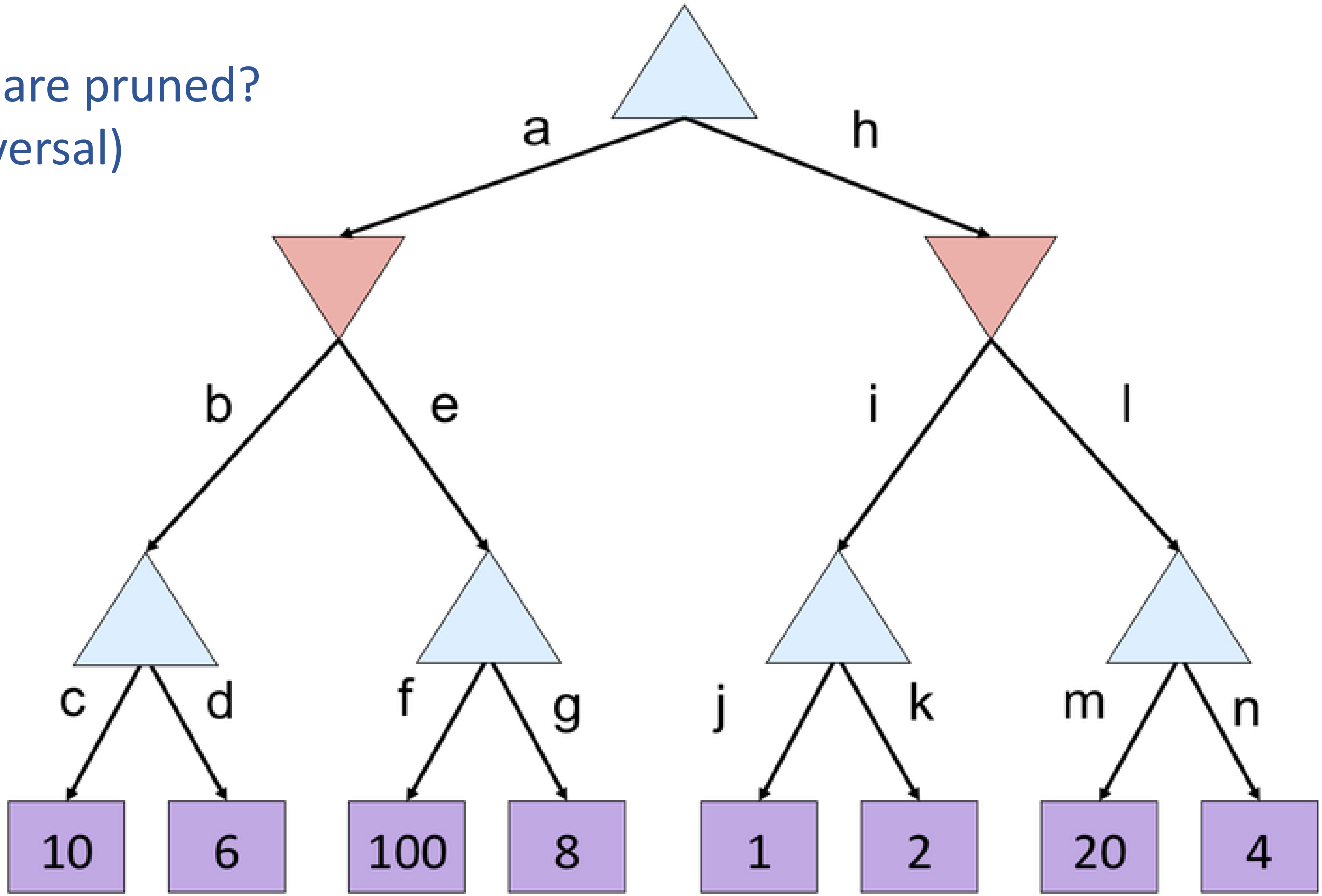
Which branches are pruned?
(Left to right traversal)

Select all that apply



Example

Which branches are pruned?
(Left to right traversal)

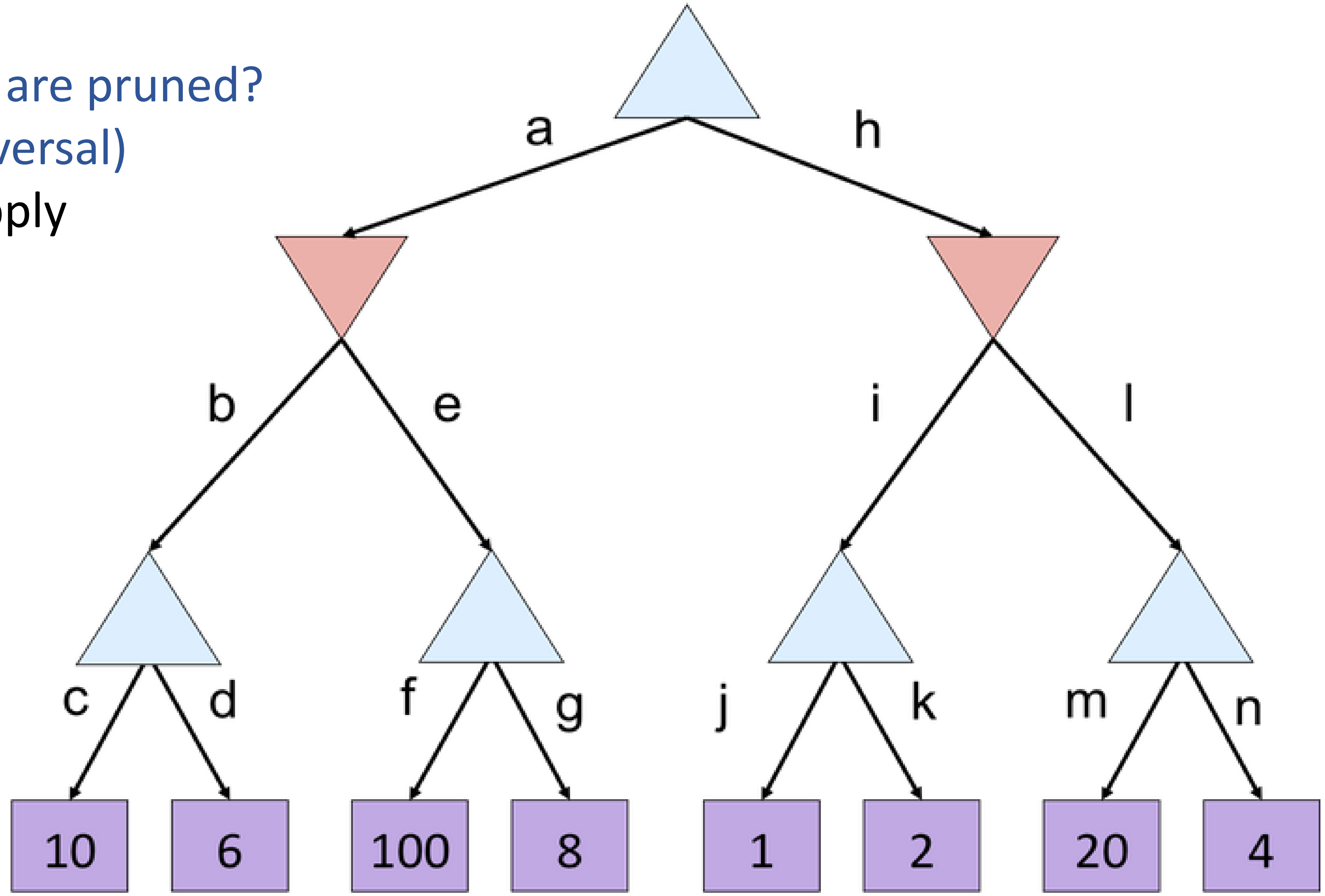


Slide credit: UCB CS 188

Poll 2

Which branches are pruned?
(Left to right traversal)

Select all that apply



Alpha-Beta Implementation

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

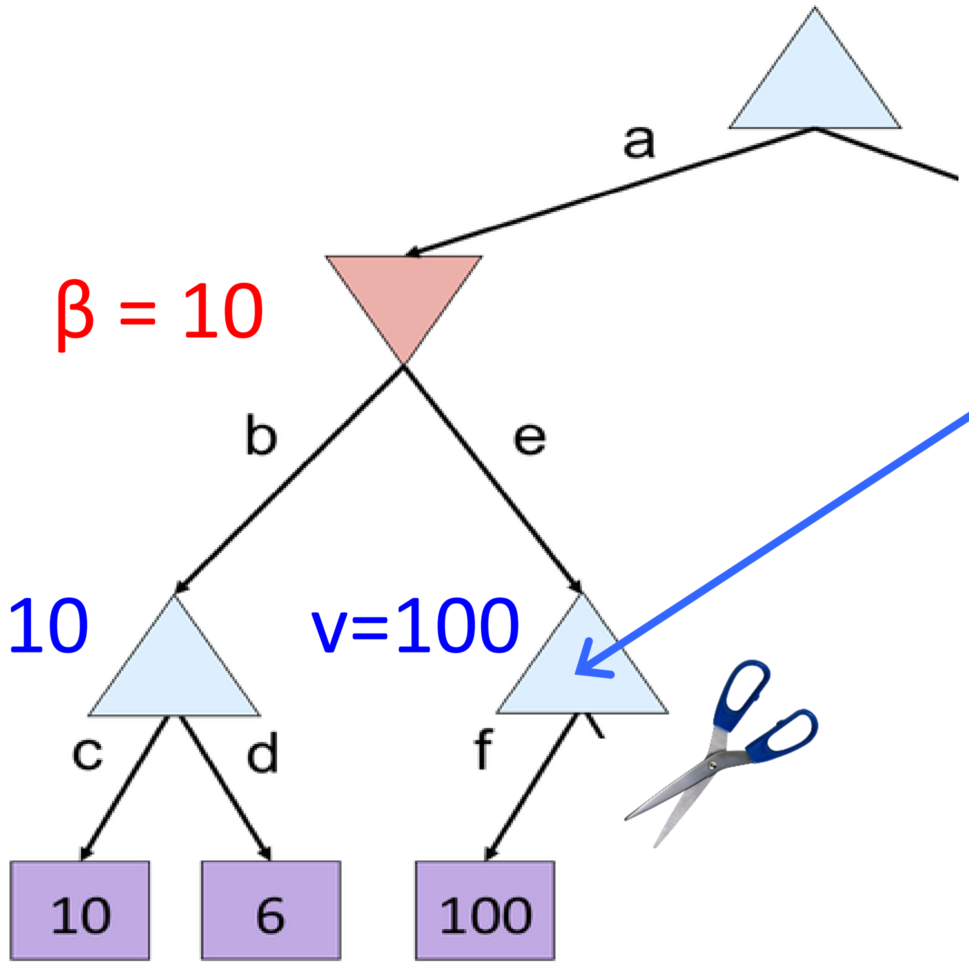
```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v ≥ β
            return v
        α = max(α, v)
    return v
```

```
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 version of Poll 1

α: MAX's best option on path to root

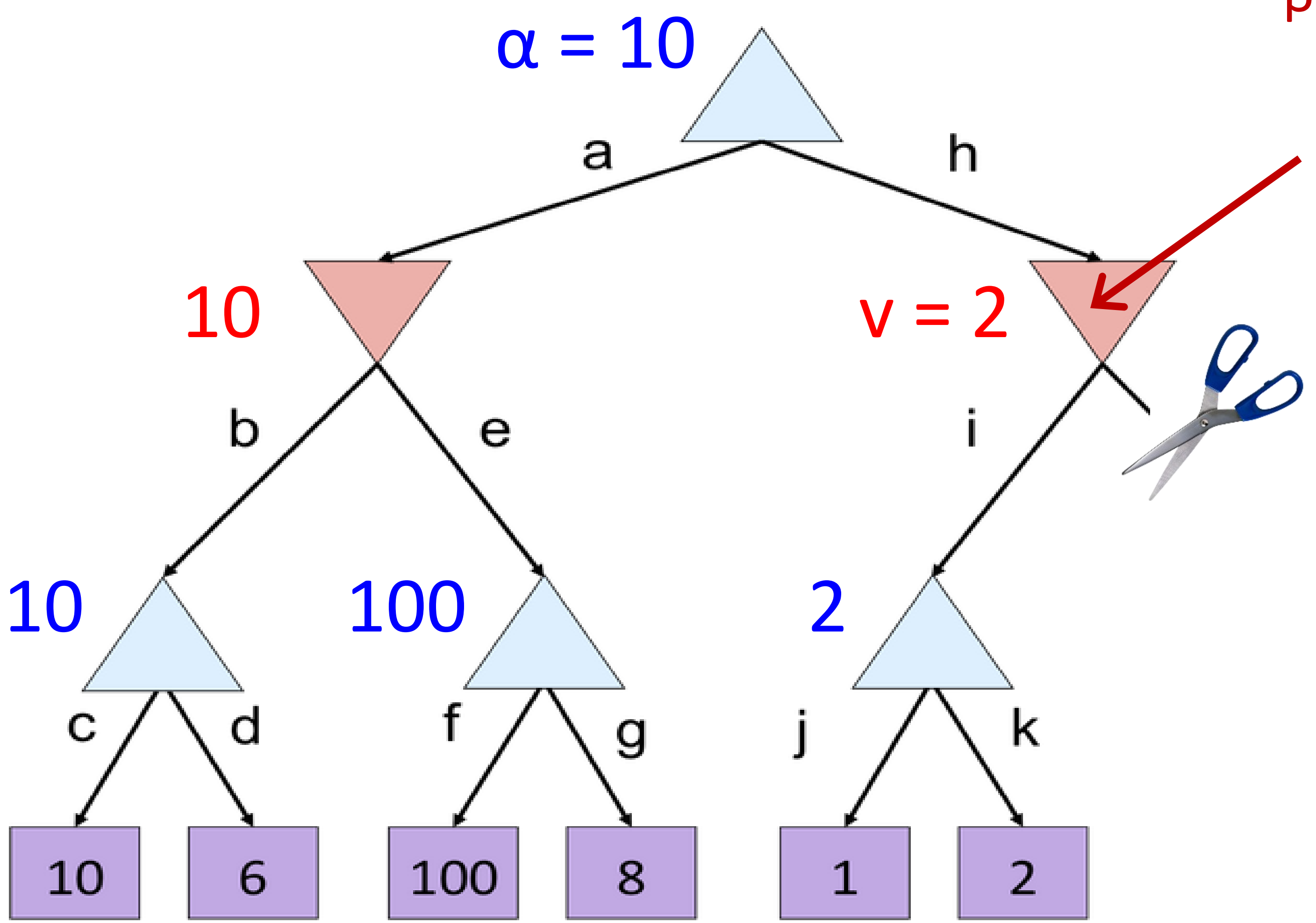
β: MIN's best option on path to root




```
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
```

Slide credit: UCB CS 188

Alpha-Beta version of Poll 1



α: MAX's best option on path to root

β: MIN's best option on path to root

```
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
```

Slide credit: UCB CS 188

Alpha-Beta Pruning Properties

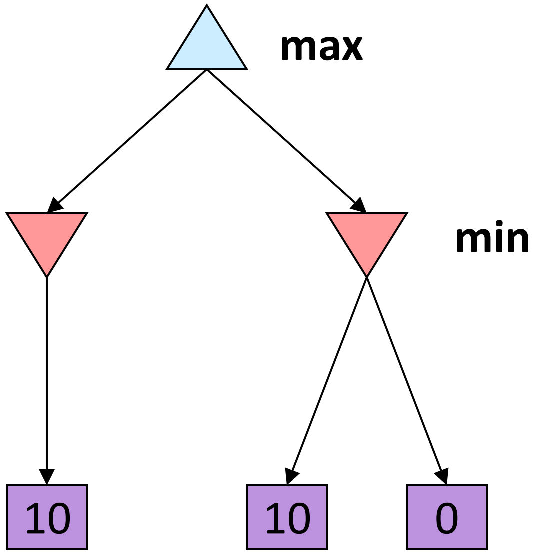
Theorem: This pruning has ***no effect*** on minimax value computed for the root!

Good child ordering improves effectiveness of pruning

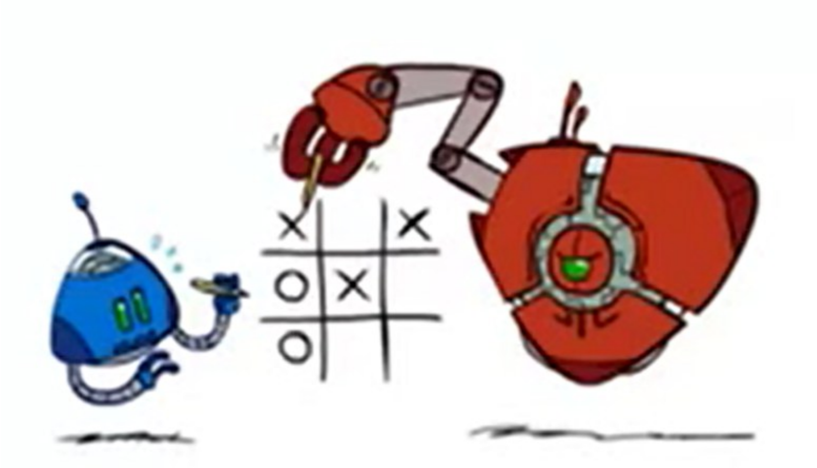
- Iterative deepening helps with this

With “perfect ordering”:

- Time complexity drops to $O(b^{m/2})$
- Doubles solvable depth!
- 1M nodes/move => depth=8, respectable



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

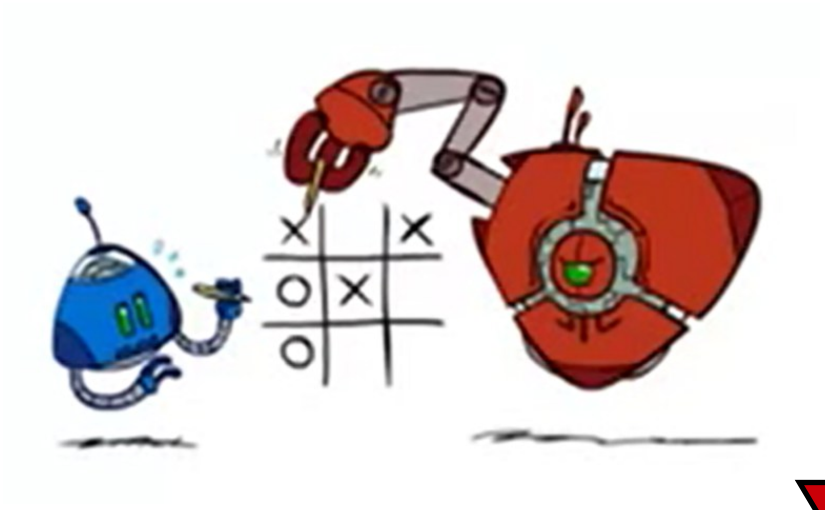


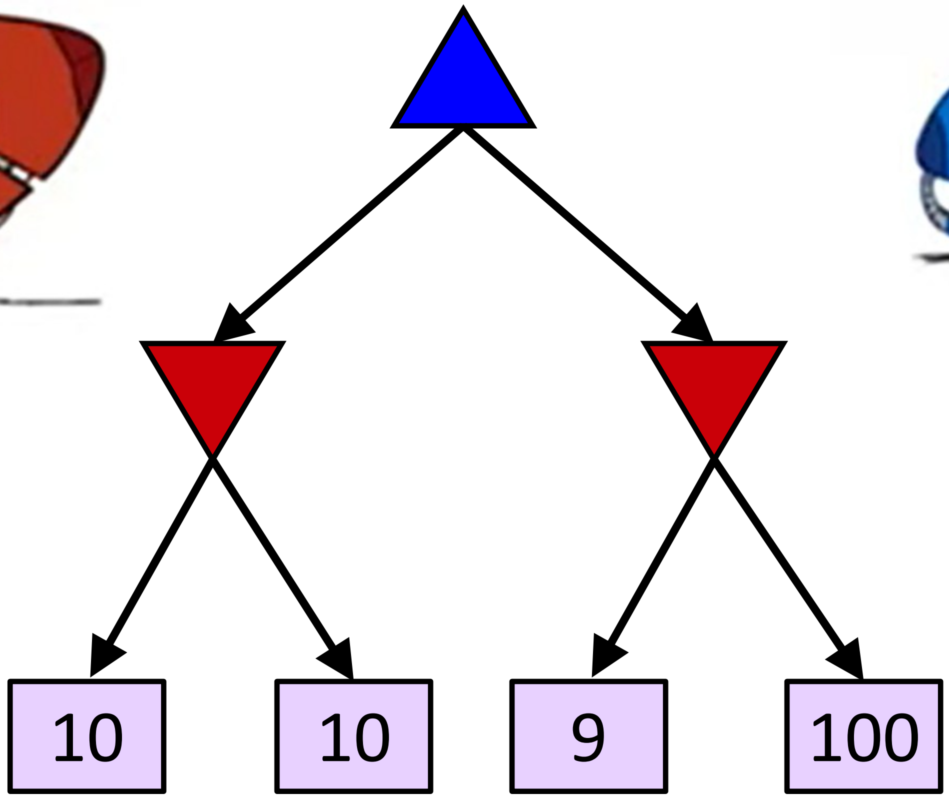
Modeling Assumptions

Image/slide credit: Ketrina Yim, UCB CS 188

Modeling Assumptions

Know your opponent





Image/slide credit: Ketrina Yim, UCB CS 188

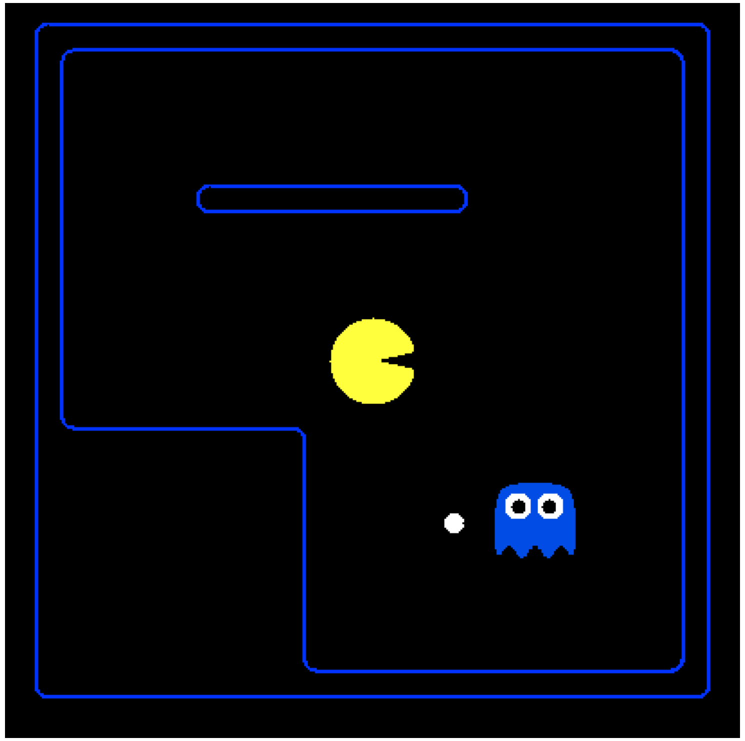
Minimax Demo

Points

+500 win

-500 lose

-1 each move



Fine print

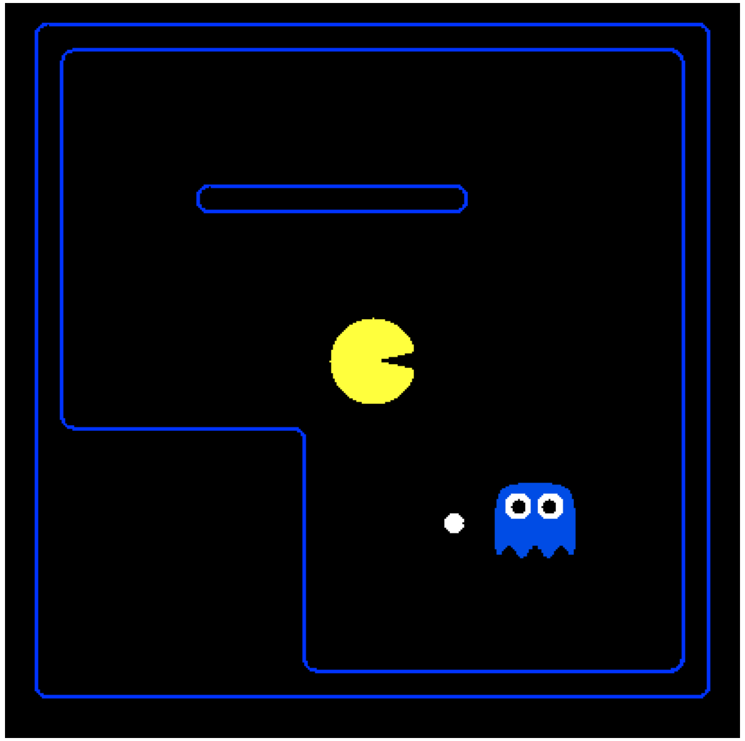
- Pacman: uses depth 4 minimax
- Ghost: uses depth 2 minimax

[Demo: minimax, minimax L7D4)]

Assumptions vs. Reality

Which pair performs the worst (for Pacman)?

Which is the best?



	Minimax Ghost	Random Ghost
Minimax Pacman		
Expectimax Pacman		

Results from playing 5 games

[Demo: random, minimax L7D6)]

Modeling Assumptions

Minimax autonomous vehicle?

Image: https://corporate.ford.com/innovation/autonomous-2021.html

Minimax Driver?

https://youtu.be/tzjgkMvTk5E?si=4twYU7UhYCkZqF73&t=89

Clip: How I Met Your Mother, CBS

Modeling Assumptions

Dangerous Pessimism

Assuming the worst case when it's not likely

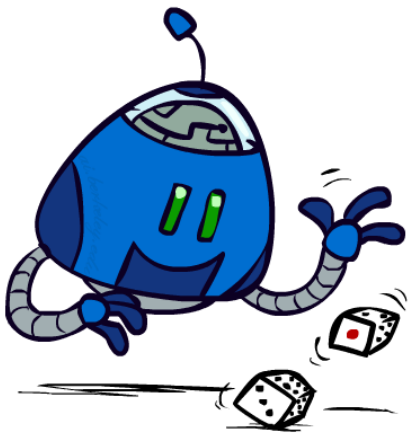
Dangerous Optimism

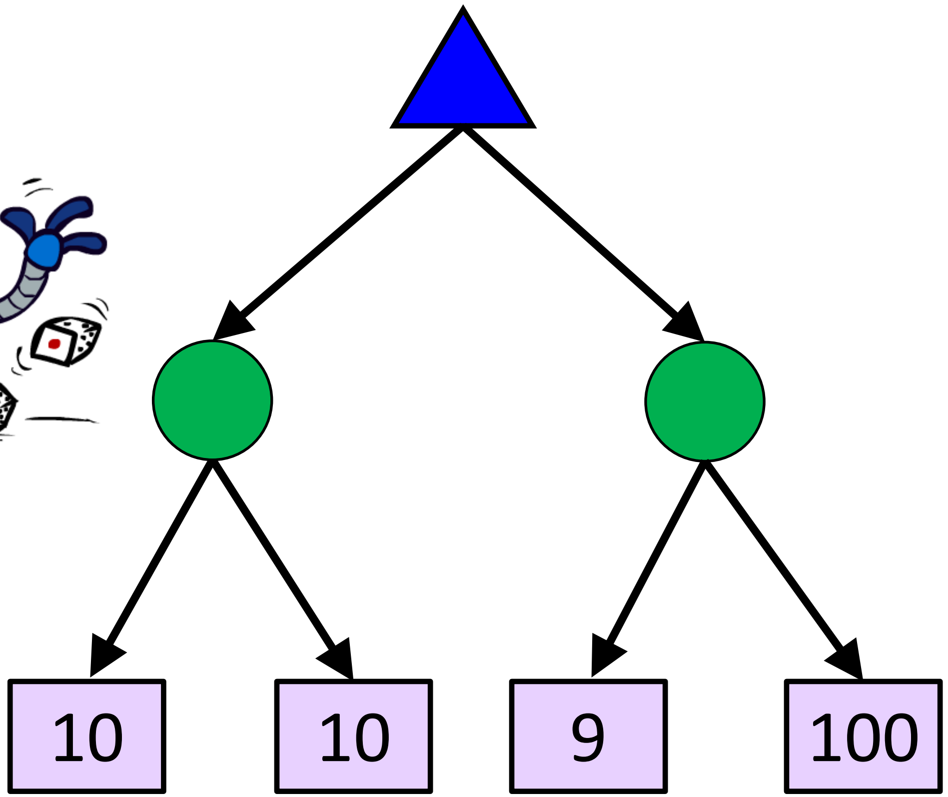
Assuming chance when the world is adversarial

Image/slide credit: Ketrina Yim, UCB CS 188

Modeling Assumptions

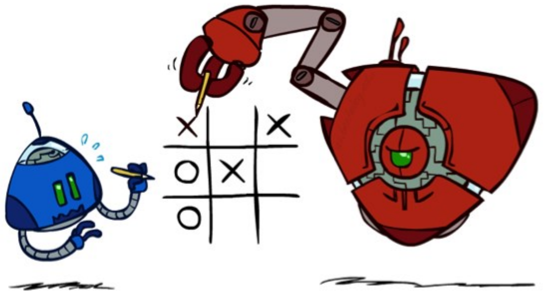
Chance nodes: Expectimax

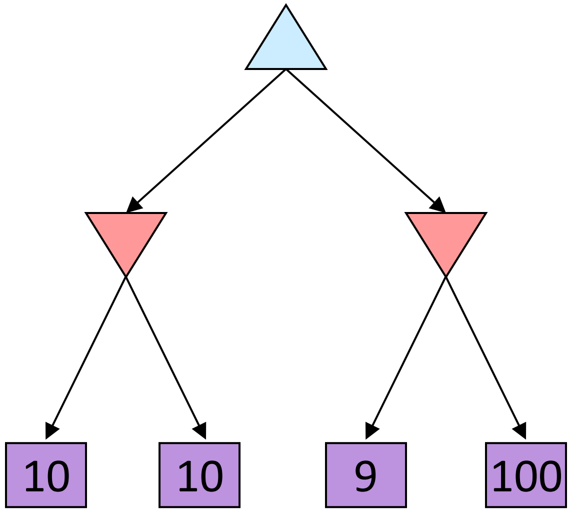




Image/slide credit: Ketrina Yim, UCB CS 188

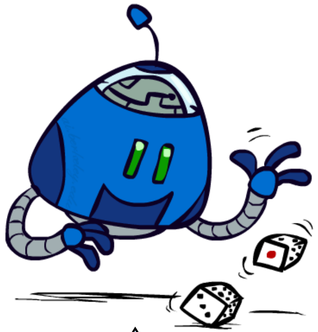
Chance outcomes in trees

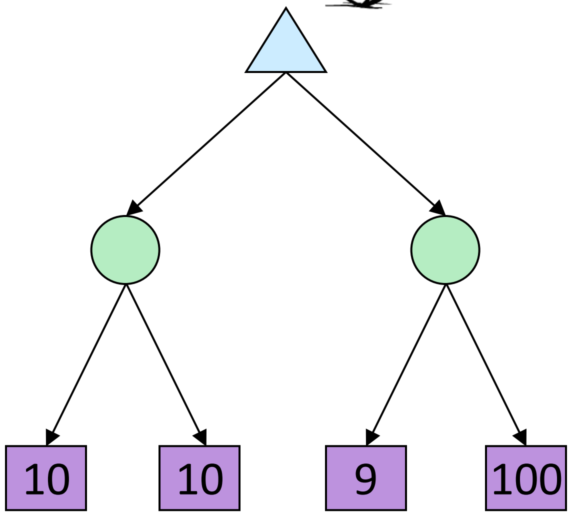




Tictactoe, chess

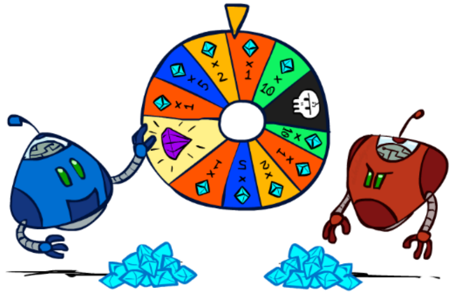
Minimax

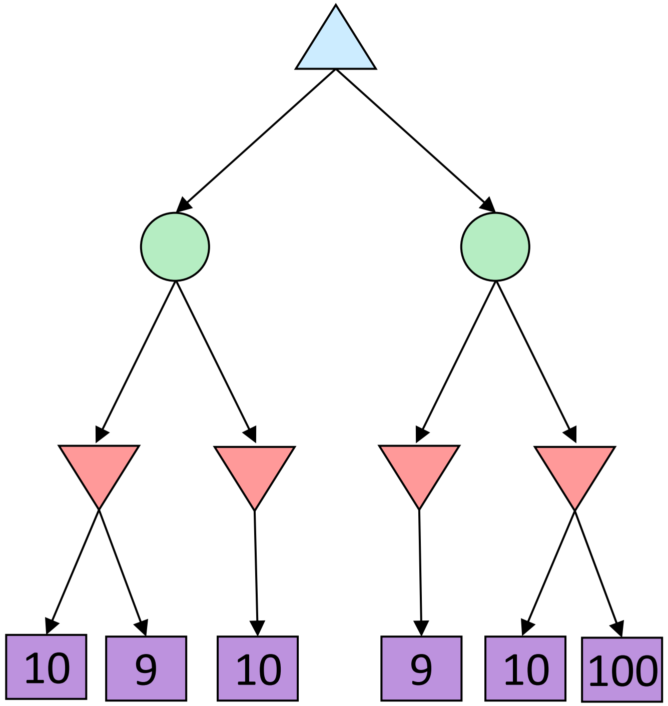




Tetris, investing

Expectimax



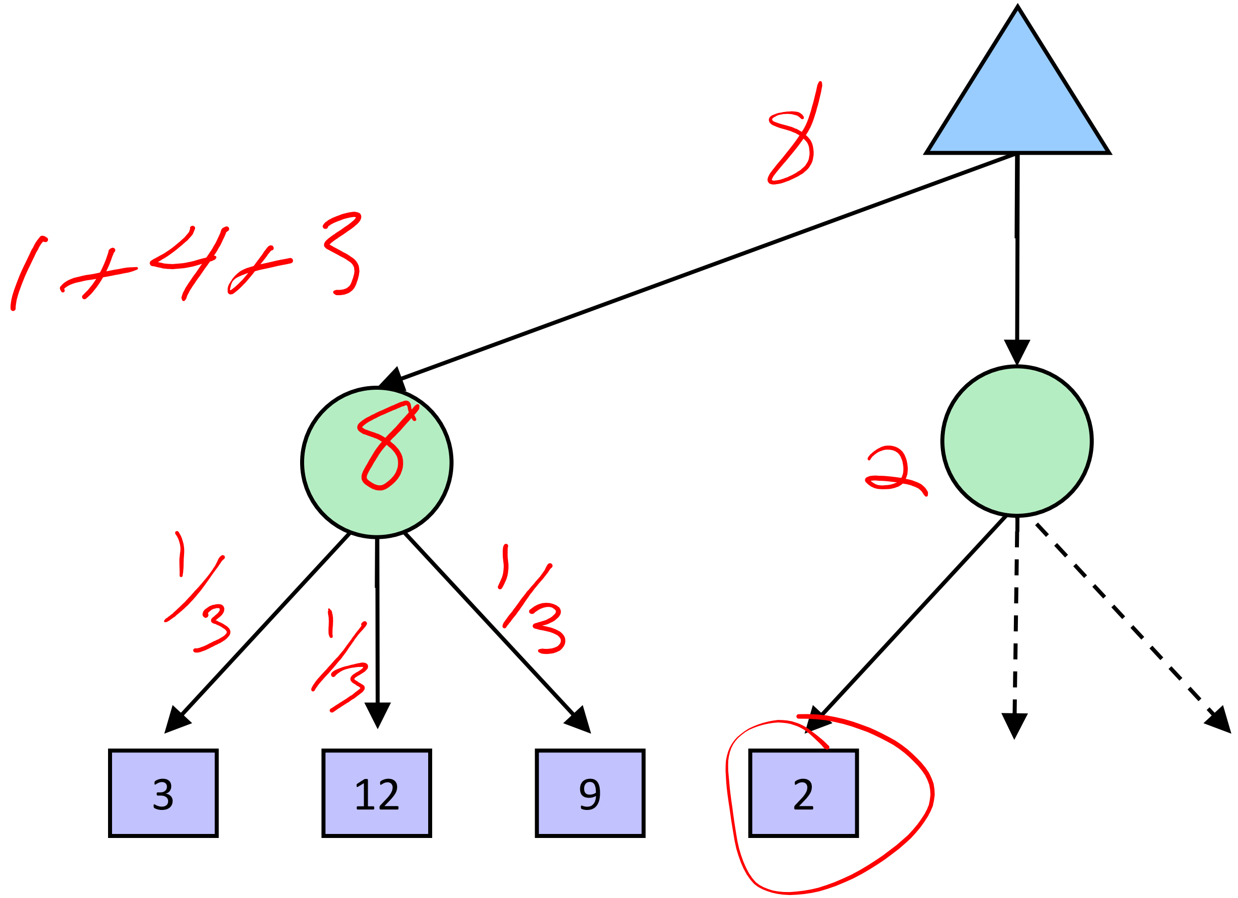


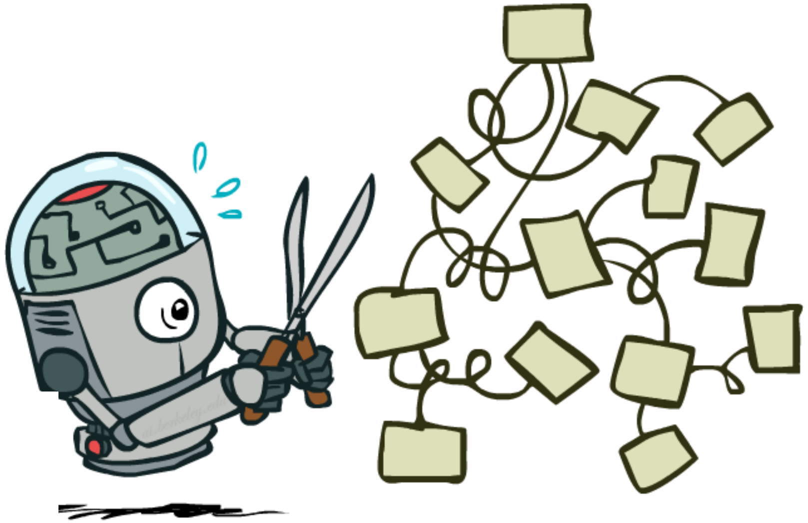
Backgammon, Monopoly

Expectiminimax

Image/slide credit: Ketrina Yim, UCB CS 188

Expectimax Pruning?



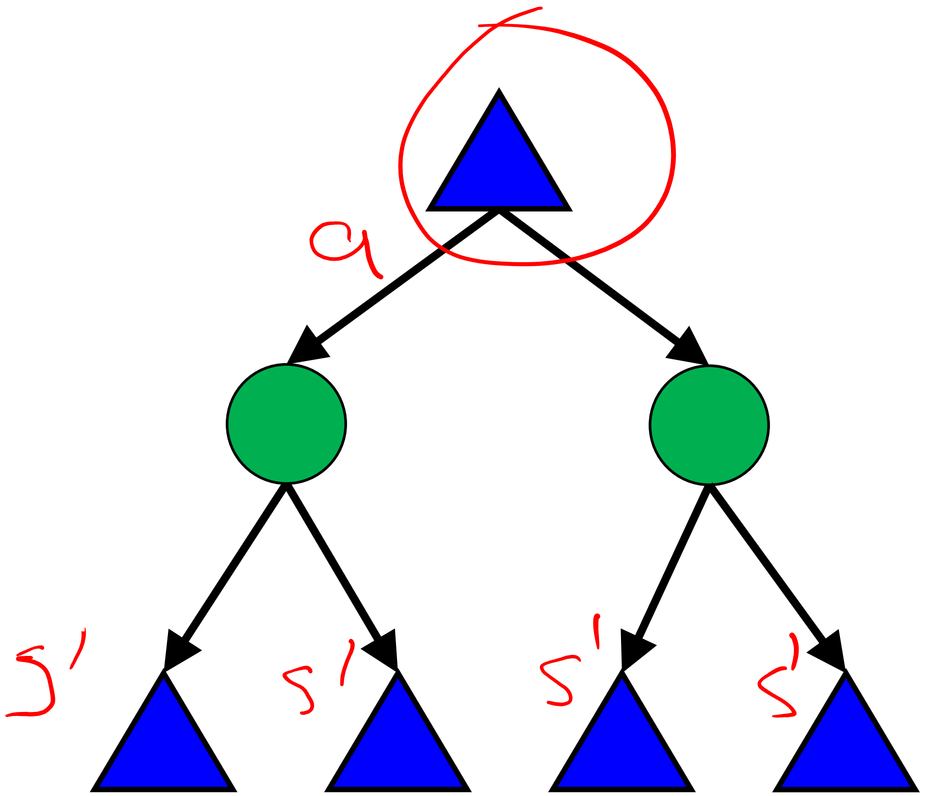


Expectimax Code

```
function  value( state )
    if  state.is_leaf
        return  state.value

    if  state.player  is  MAX
        return  max a in state.actions  value( state.result(a) )

    if  state.player  is  MIN
        return  min a in state.actions  value( state.result(a) )

    if  state.player  is  CHANCE
        return  sum s in state.next_states  P( s ) * value( s )
```

Slide credit: UCB CS 188

Preview: MDP/Reinforcement Learning Notation



$$V(s) = \max_a \sum_{s'} P(s')\, \underline{V(s')}$$

Preview: MDP/Reinforcement Learning Notation

Standard expectimax: $V(s) = \max_a \sum_{s'} P(s'|s,a)V(s')$

Bellman equations: $V(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')]$

Value iteration: $V_{k+1}(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_k(s')], \quad \forall\, s$

Q-iteration: $Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')], \quad \forall\, s,a$

Policy extraction: $\pi_V(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')], \quad \forall\, s$

Policy evaluation: $V_{k+1}^{\pi}(s) = \sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V_k^{\pi}(s')], \quad \forall\, s$

Policy improvement: $\pi_{new}(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^{\pi_{old}}(s')], \quad \forall\, s$

Preview: MDP/Reinforcement Learning Notation

Standard expectimax: $V(s) = \max_a \sum_{s'} P(s'|s,a)V(s')$

Bellman equations: $V(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')]$

Value iteration: $V_{k+1}(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_k(s')], \quad \forall\, s$

Q-iteration: $Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')], \quad \forall\, s,a$

Policy extraction: $\pi_V(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')], \quad \forall\, s$

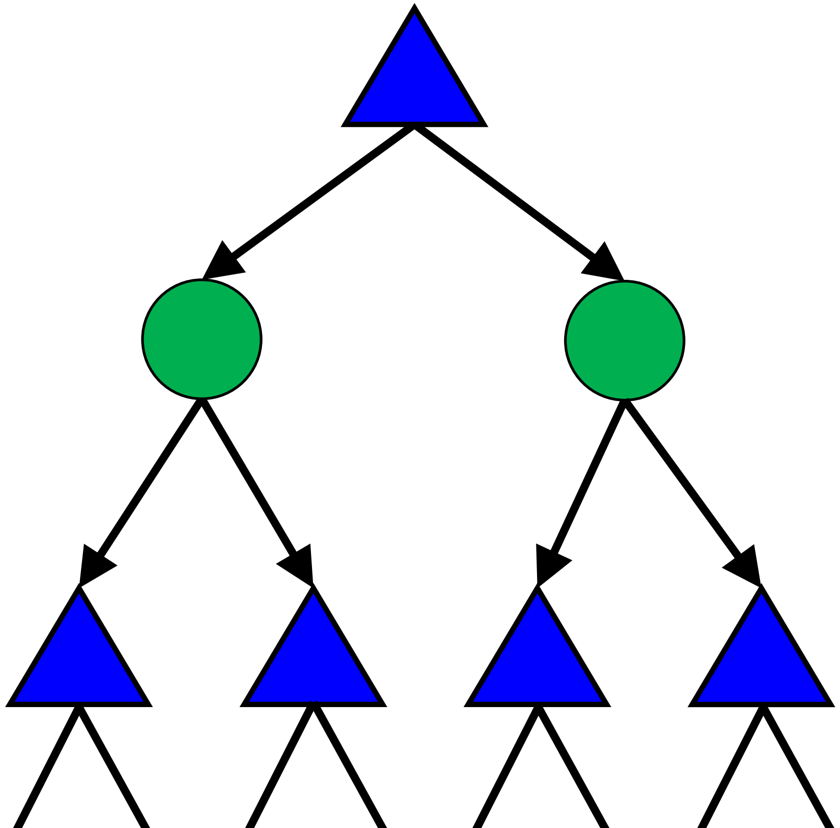
Policy evaluation: $V_{k+1}^{\pi}(s) = \sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V_k^{\pi}(s')], \quad \forall\, s$

Policy improvement: $\pi_{new}(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^{\pi_{old}}(s')], \quad \forall\, s$

Why Expectimax?

Pretty great model for an agent in the world

Choose the action that has the: highest expected value



Summary

Games require decisions when optimality is impossible

- Bounded-depth search and approximate evaluation functions

Games force efficient use of computation

- Alpha-beta pruning

Game playing has produced important research ideas

- Reinforcement learning (checkers)
- Iterative deepening (chess)
- Monte Carlo tree search (Go)
- Solution methods for partial-information games in economics (poker)

Video games present much greater challenges – lots to do!

- $b = 10^{500}$, $|S| = 10^{4000}$, $m = 10{,}000$