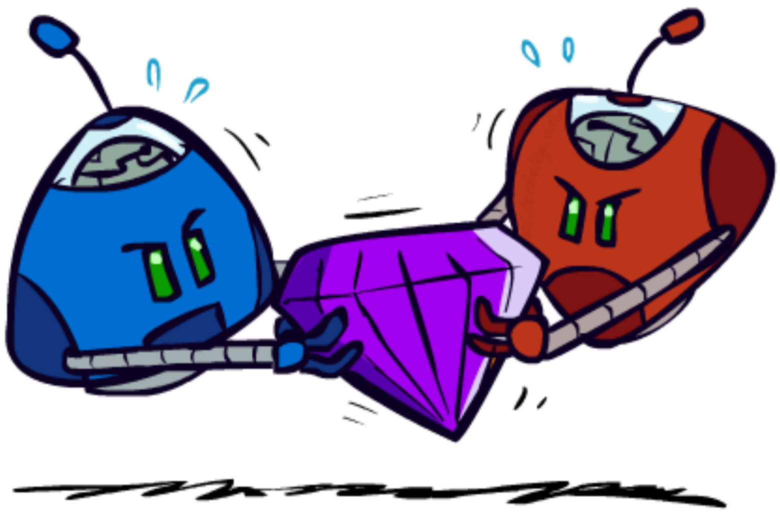
CSE 573: Artificial Intelligence

Adversarial Search

Dan Weld



Based on slides from

Dan Klein, Stuart Russell, Pieter Abbeel, Andrew Moore and Luke Zettlemoyer

(best illustrations from ai.berkeley.edu)

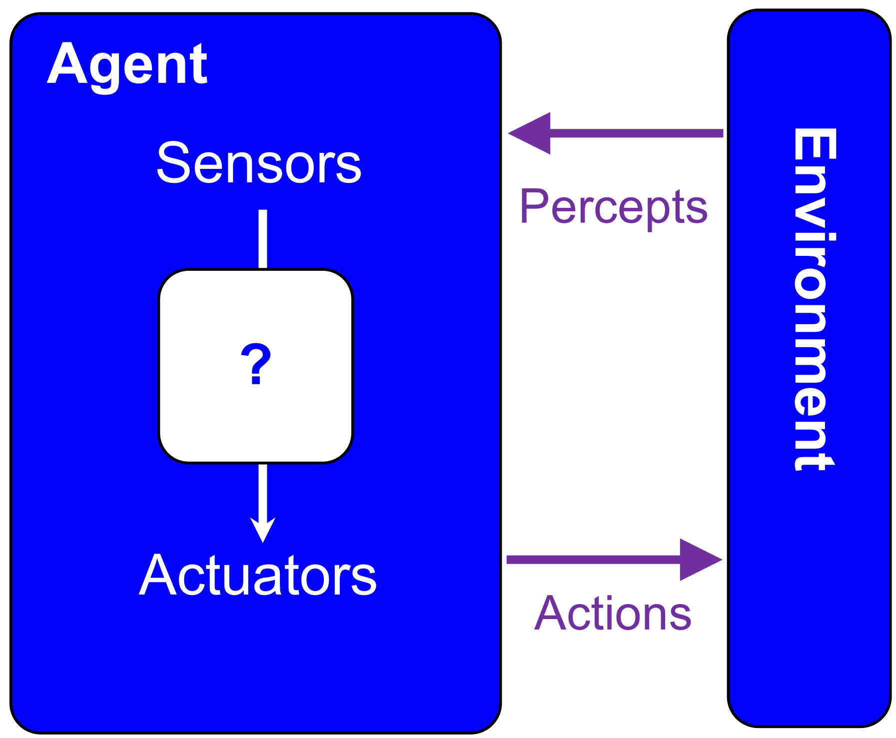
Outline

- Adversarial Search
  - Minimax search
  - α-β search
  - Evaluation functions
  - Expectimax

- Reminder:
  - Project 2 due in 7 days

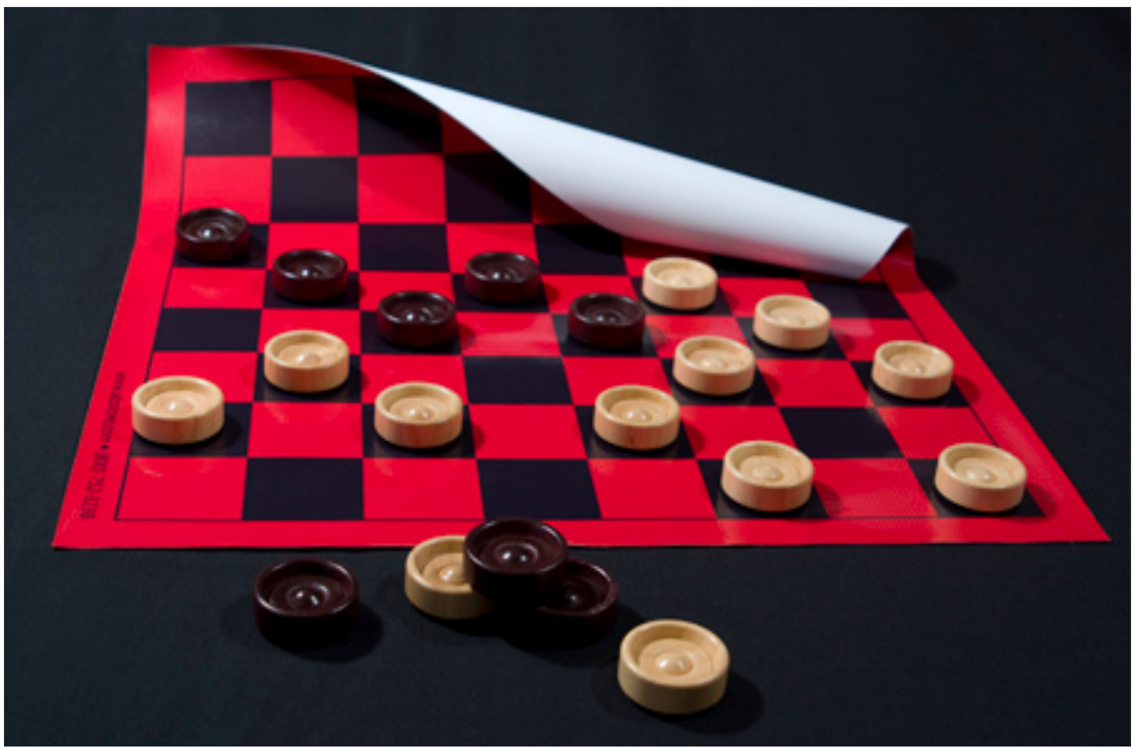
Types of Environments

- Fully observable *vs.* partially observable
- Single agent *vs.* ***multi-agent***
- Deterministic *vs.* stochastic
- Episodic *vs.* sequential
- Discrete *vs.* continuous



Game Playing State-of-the-Art

1994: Checkers. Chinook ended 40-year-reign of human world champion Marion Tinsley. Used search plus an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions. Checkers is now solved!



Game Playing State-of-the-Art

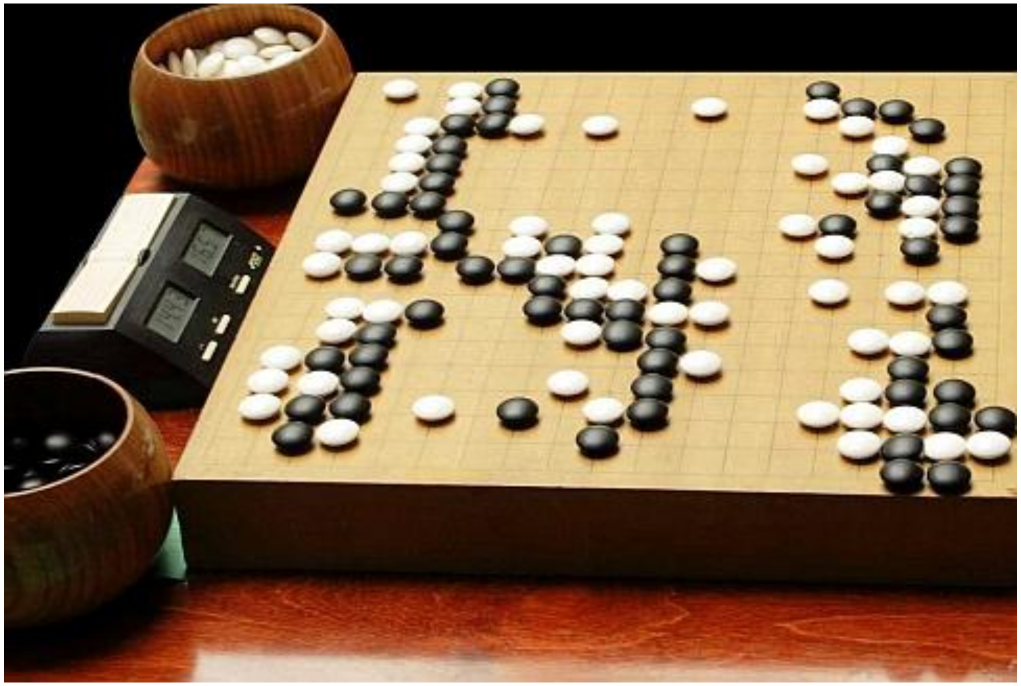
1997: Chess. Deep Blue defeated human world champion Gary Kasparov in a six-game match. Deep Blue examined 200 million positions per second, used very sophisticated evaluation and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic.

Game Playing State-of-the-Art

Go: b > 300! Programs use monte carlo tree search + pattern KBs

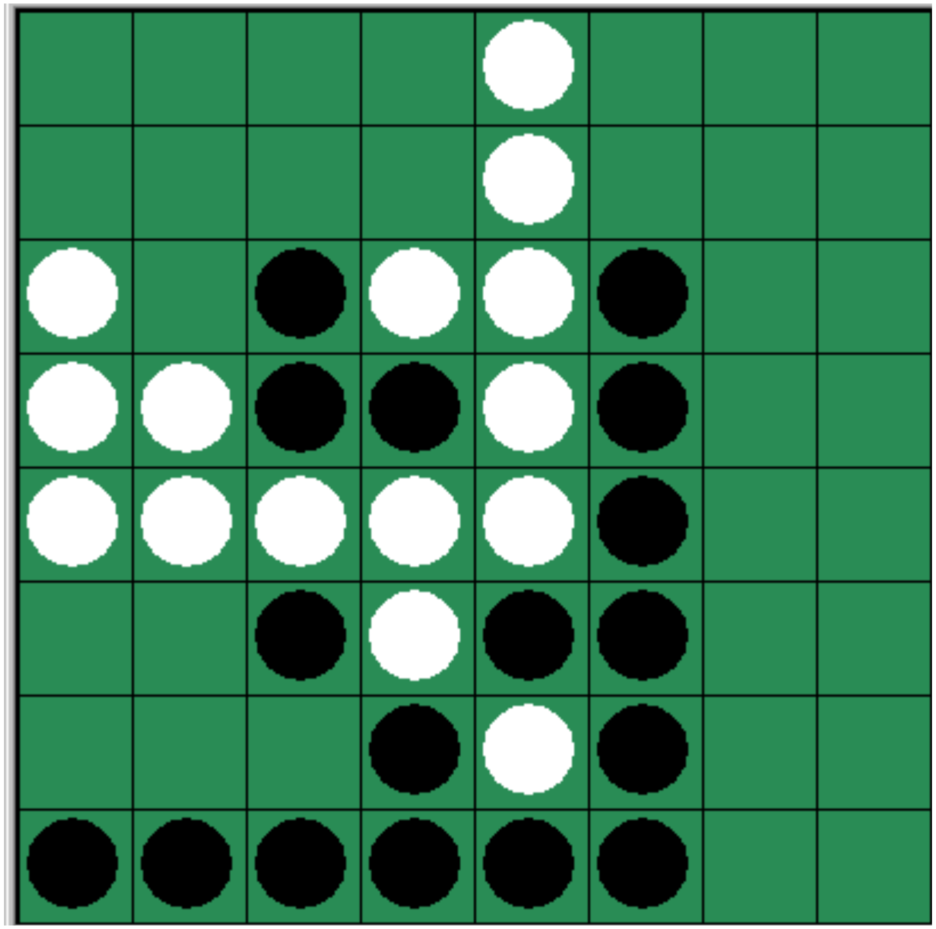
2015: AlphaGo beats European Go champion Fan Hui (2 dan) 5-0

2016: AlphaGo beats Lee Sedol (9 dan) 4-1



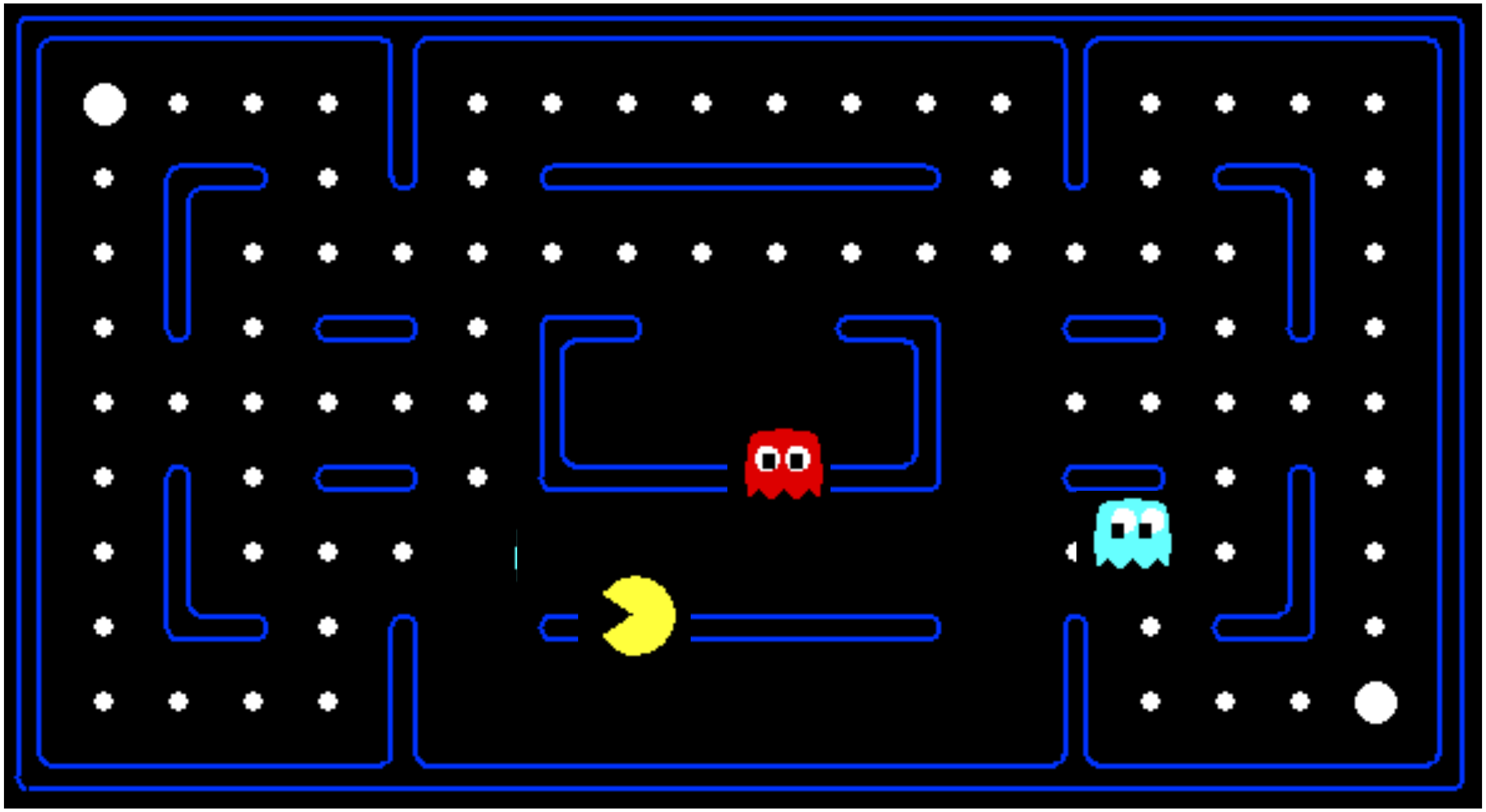
Game Playing State-of-the-Art

Othello: Human champions refuse to compete against computers.



Game Playing State-of-the-Art

- **Pacman:** … unknown …



Types of Games

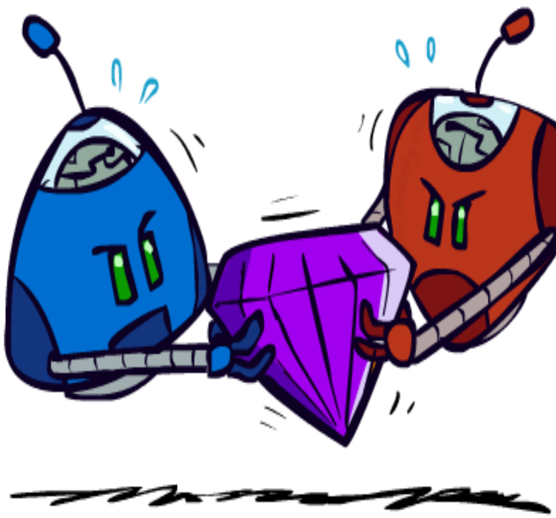
	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon, monopoly
imperfect information	stratego	bridge, poker, scrabble, nuclear war

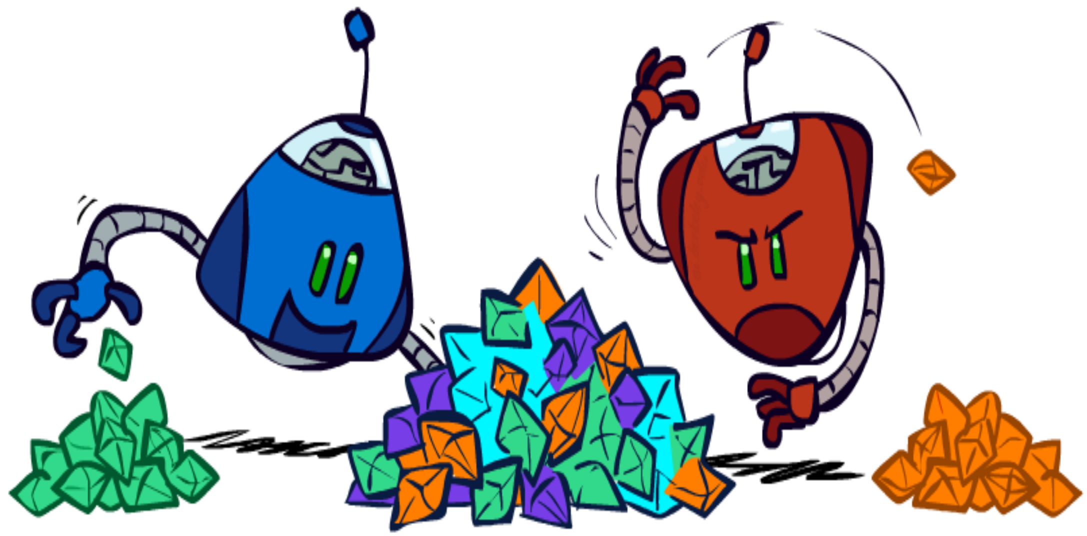
Number of Players? 1, 2, …?

Deterministic Games

- Many possible formalizations, one is:
  - States: S (start at s_0)
  - Players: P={1...N} (usually take turns)
  - Actions: A (may depend on player / state)
  - Transition Function: S x A → S
  - Terminal Test: S → {t,f}
  - Terminal Utilities: S x P→ R
- Solution for a player is a ***policy***: S → A

Zero-Sum Games

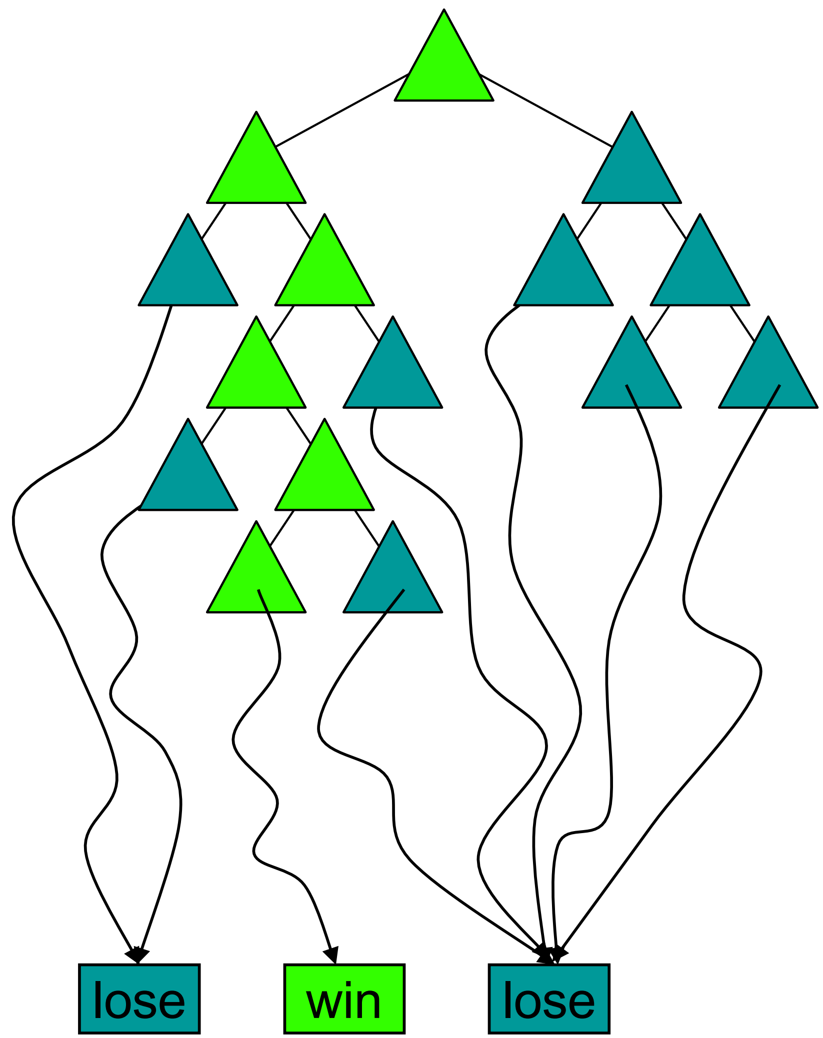




- Zero-Sum Games
  - Agents have opposite utilities (values on outcomes)
  - Lets us think of a single value that one maximizes and the other minimizes
  - Adversarial, pure competition
- General Games
  - Agents have independent utilities (values on outcomes)
  - Cooperation, indifference, competition, & more are possible
  - More later on non-zero-sum games

Deterministic Single-Player

- Deterministic, single player, perfect information:
  - Know the rules, action effects, winning states
  - E.g. Freecell, 8-Puzzle, Rubik's cube
- … it's just search!
- Slight reinterpretation:
  - Each node stores a value: the best outcome it can reach
  - This is the maximal outcome of its children (the max value)
  - Note that we don't have path sums as before (utilities at end)
- After search, can pick move that leads to best node

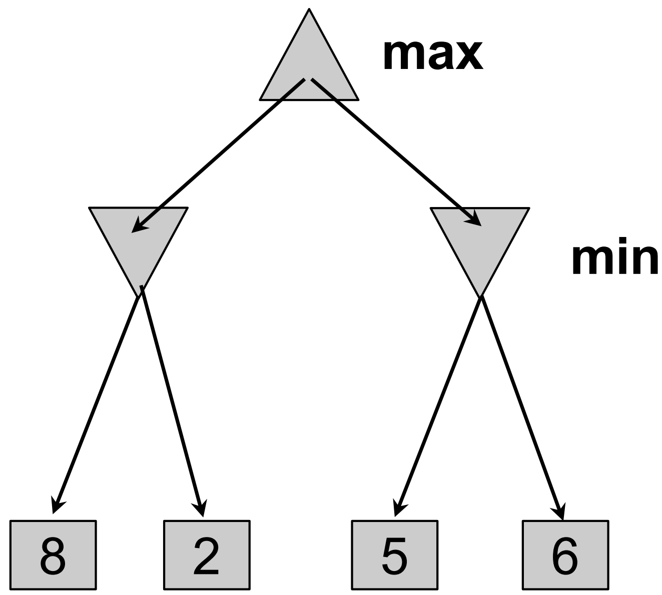


Deterministic Two-Player

- E.g. tic-tac-toe, chess, checkers
- Zero-sum games
  - One player maximizes result
  - The other minimizes result

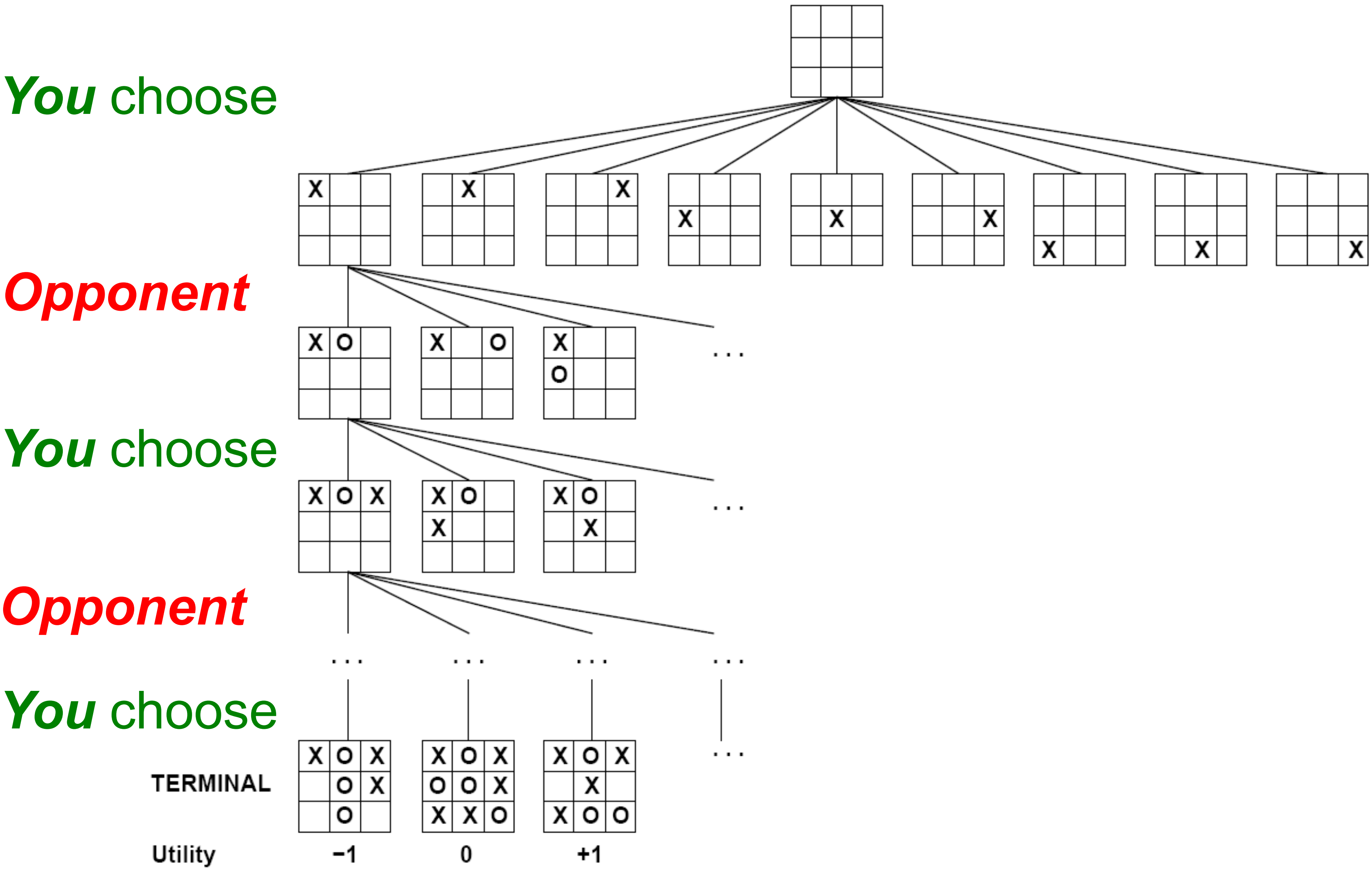
Deterministic Two-Player

- E.g. tic-tac-toe, chess, checkers
- Zero-sum games
  - One player maximizes result
  - The other minimizes result

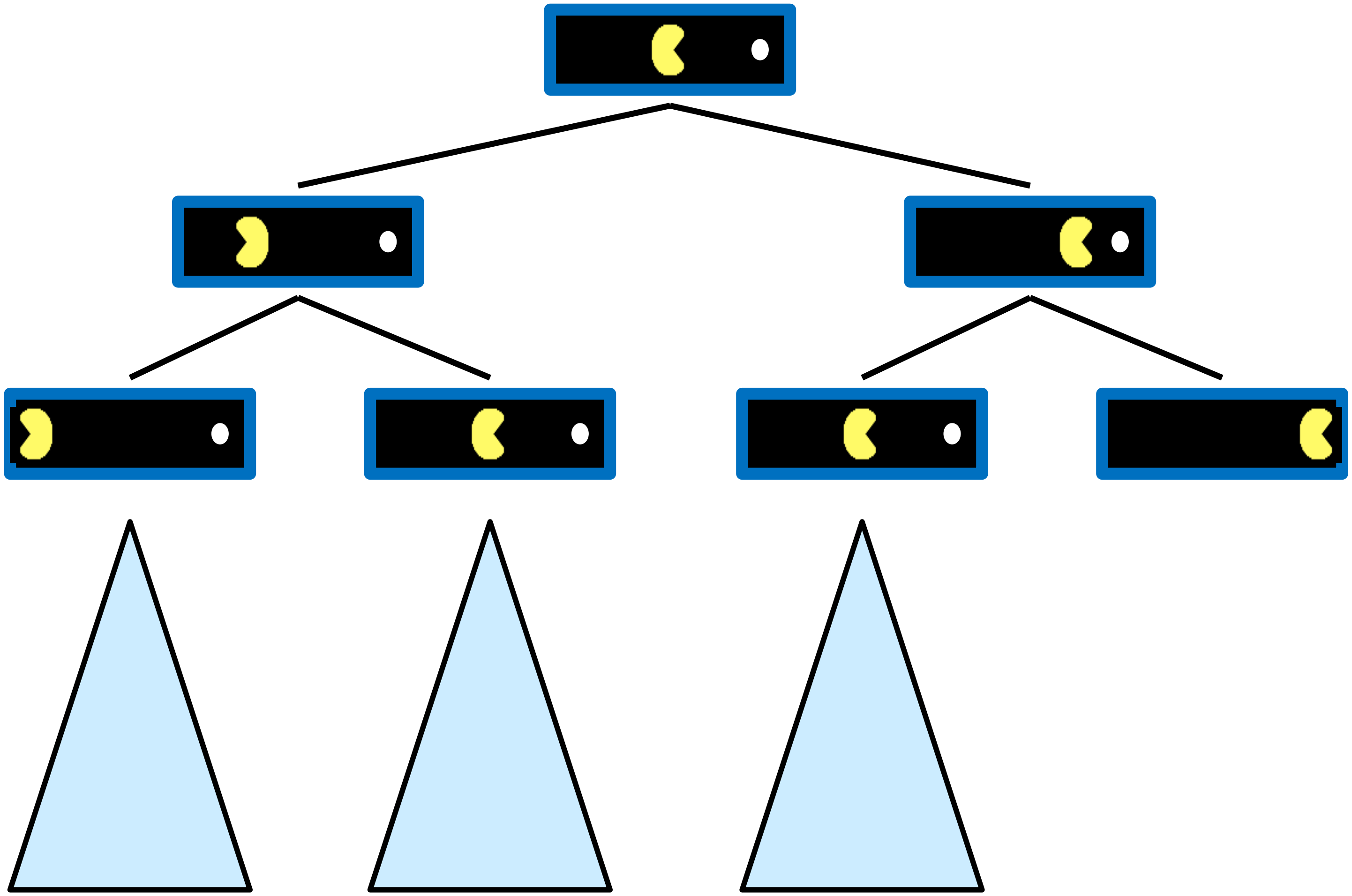


- ***Minimax search***
  - A state-space search tree
  - Players alternate
  - Choose move to position with highest minimax value ***= best achievable utility against best play***

Tic-tac-toe Game Tree



Previously: Single-Agent Trees



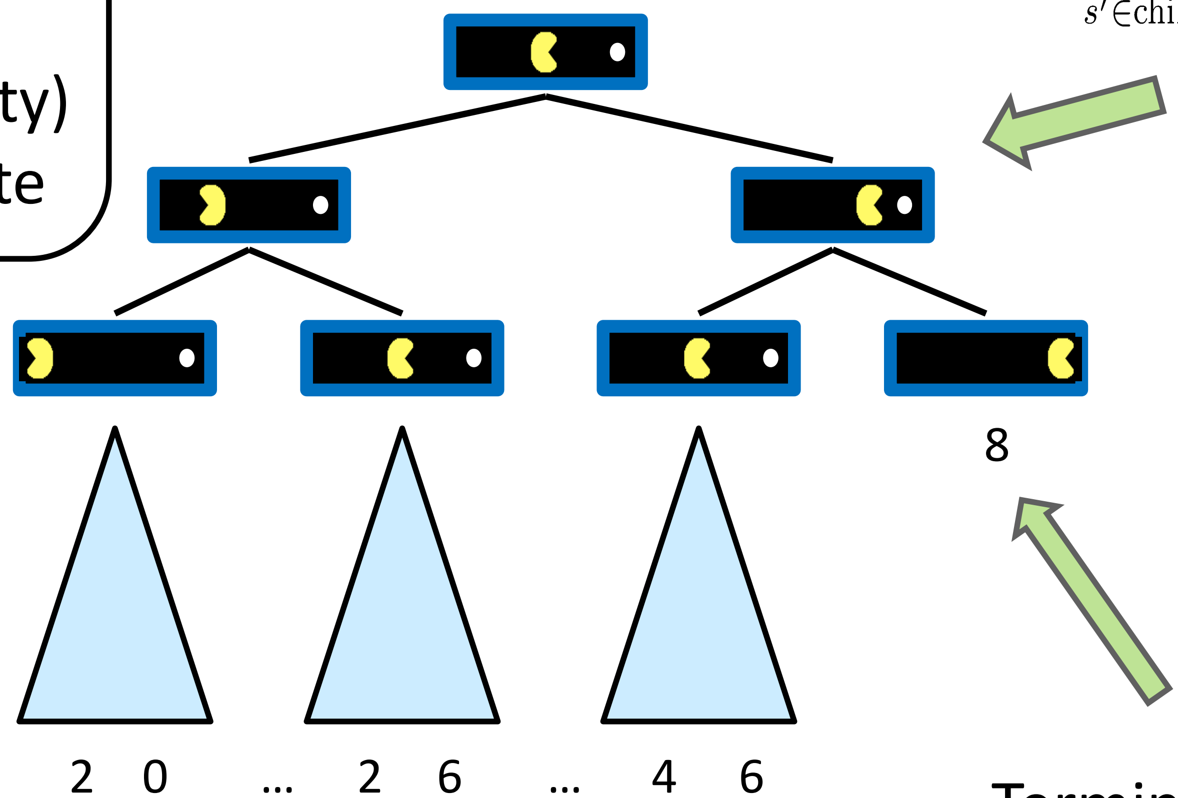
Slide from Dan Klein & Pieter Abbeel - ai.berkeley.edu

Previously: Value of a State

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

Non-Terminal States:

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

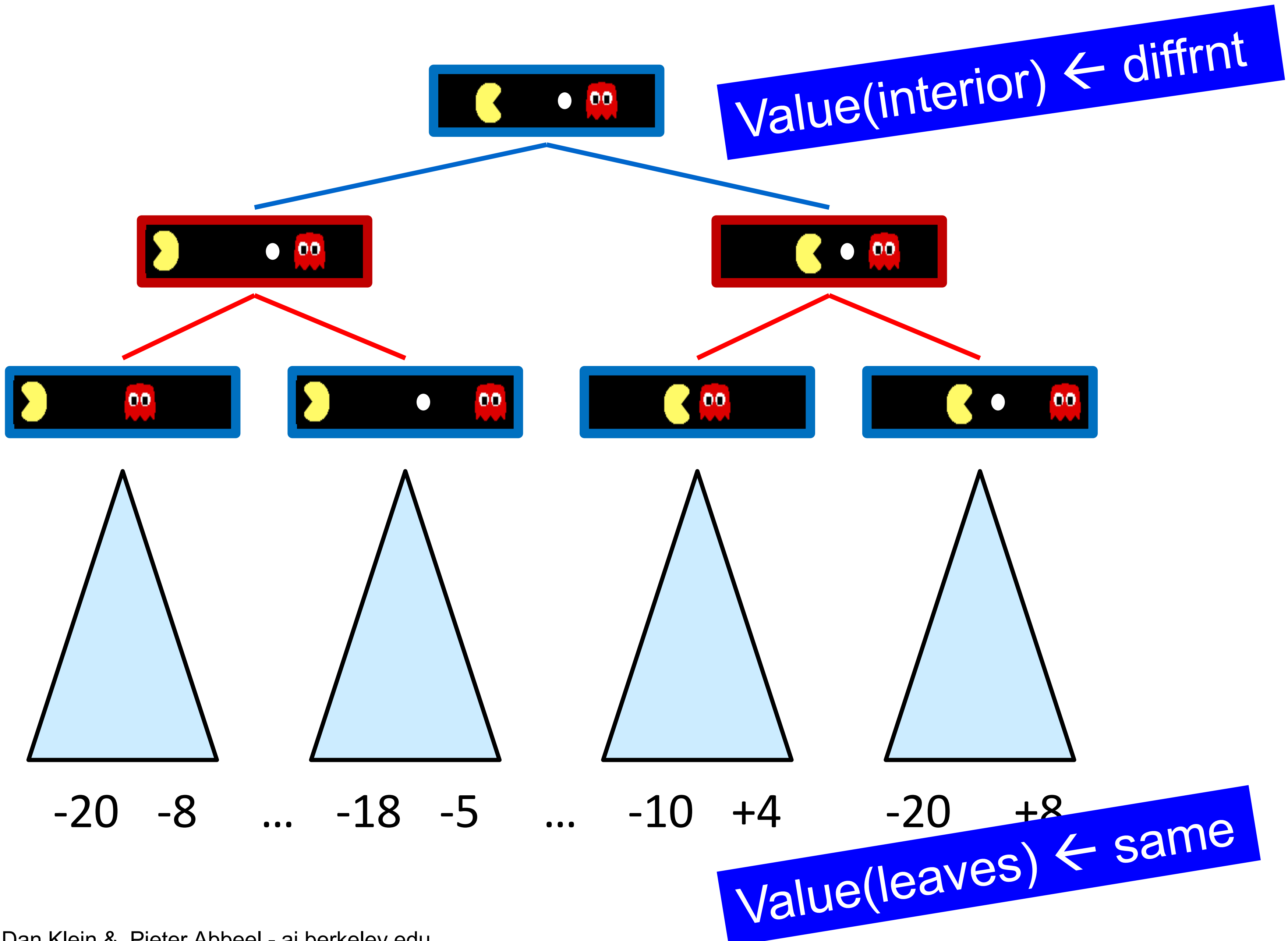


Terminal States:

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

Slide adapted from Dan Klein & Pieter Abbeel - ai.berkeley.edu

Adversarial Game Trees



Slide adapted from Dan Klein & Pieter Abbeel - ai.berkeley.edu

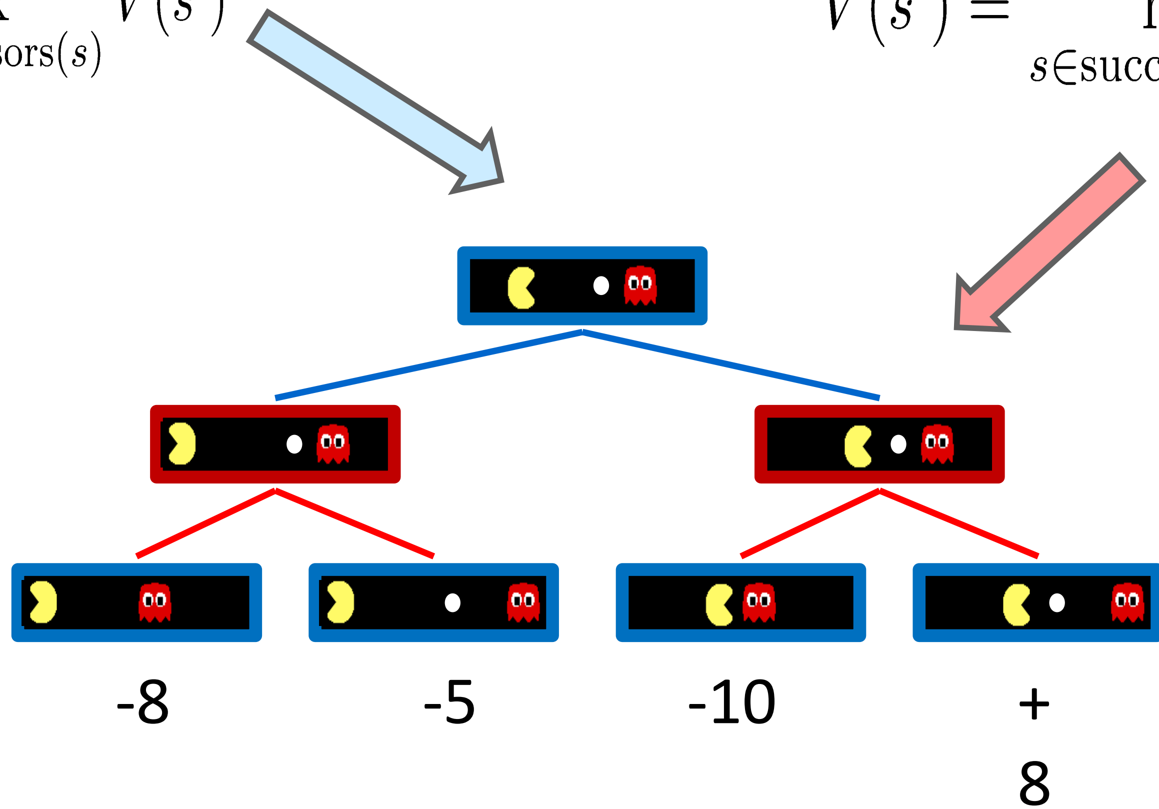
Minimax Values

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:

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

Minimax Implementation

Need ***Base case*** for recursion

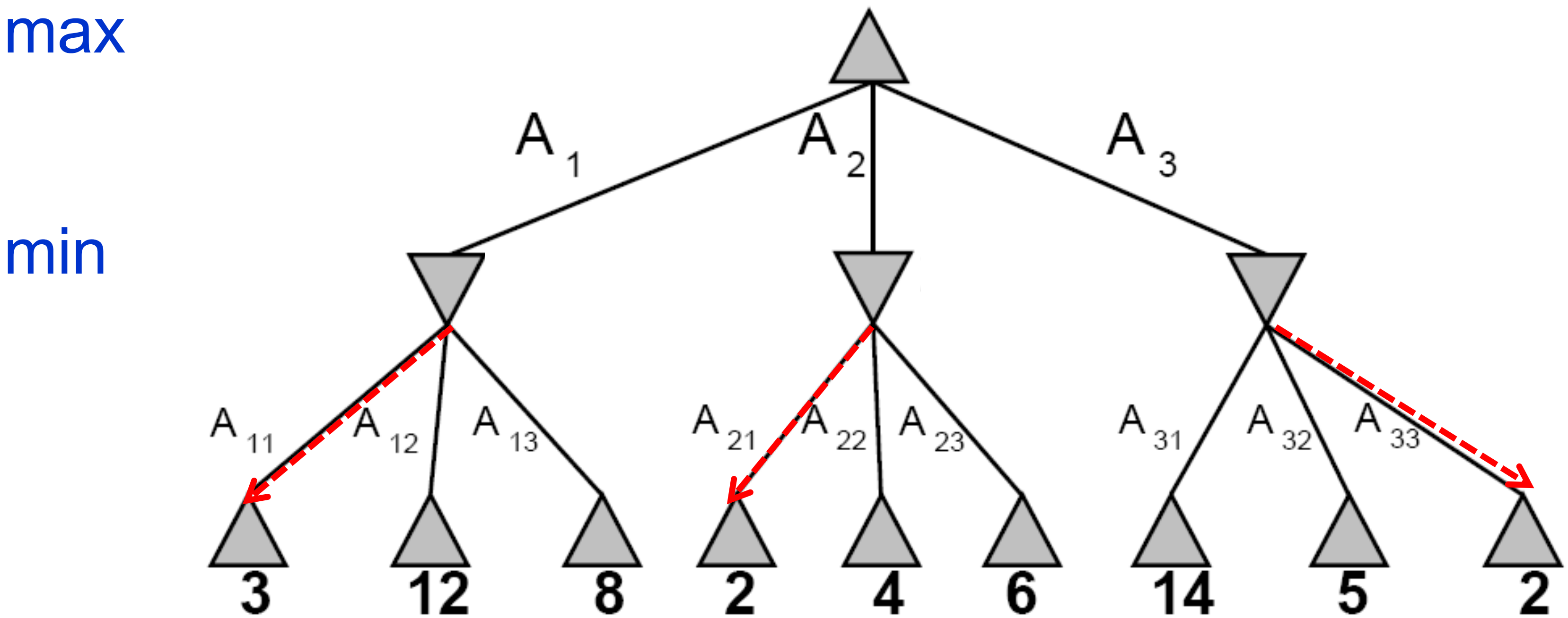
```
def max-value(state):
    if leaf?(state), return U(state)
    initialize v = -∞
    for each c in children(state)
        v = max(v, min-value(c))
    return v
```

```
def min-value(state):
    if leaf?(state), return U(state)
    initialize v = +∞
    for each c in children(state)
        v = min(v, max-value(c))
    return v
```

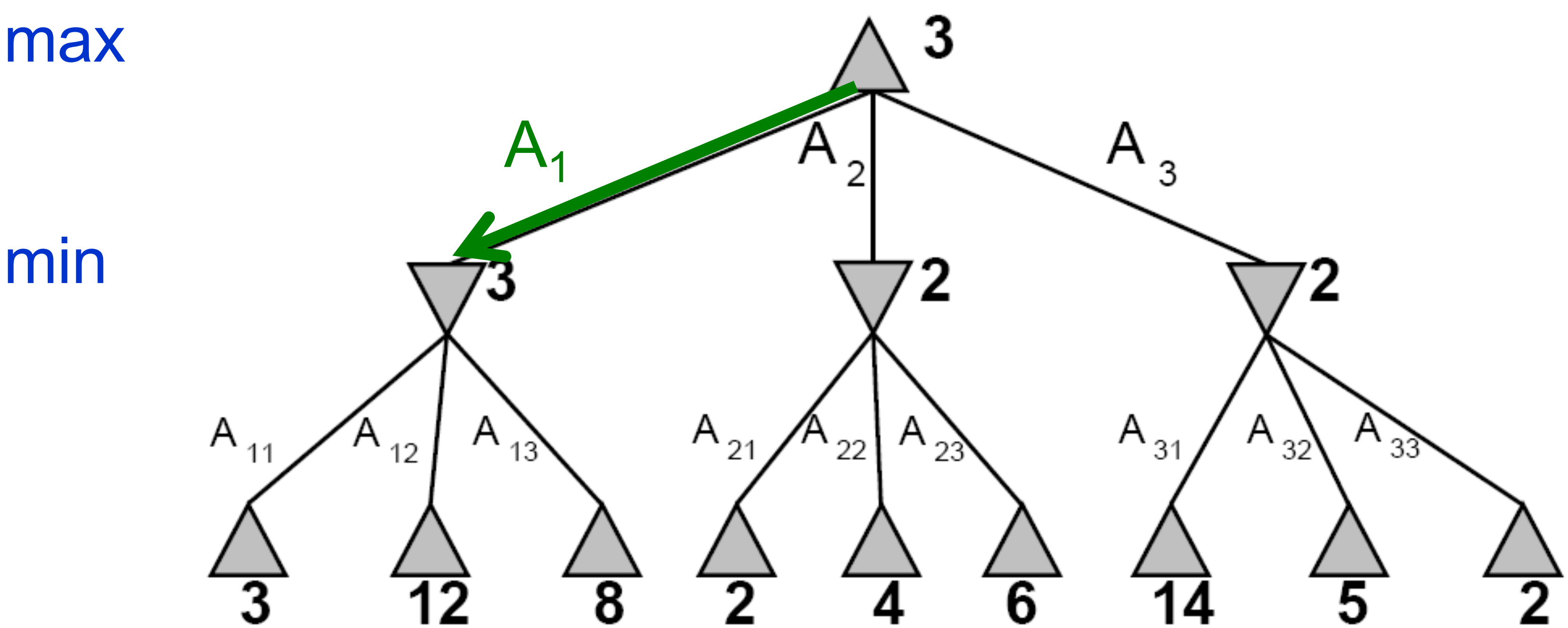
$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

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

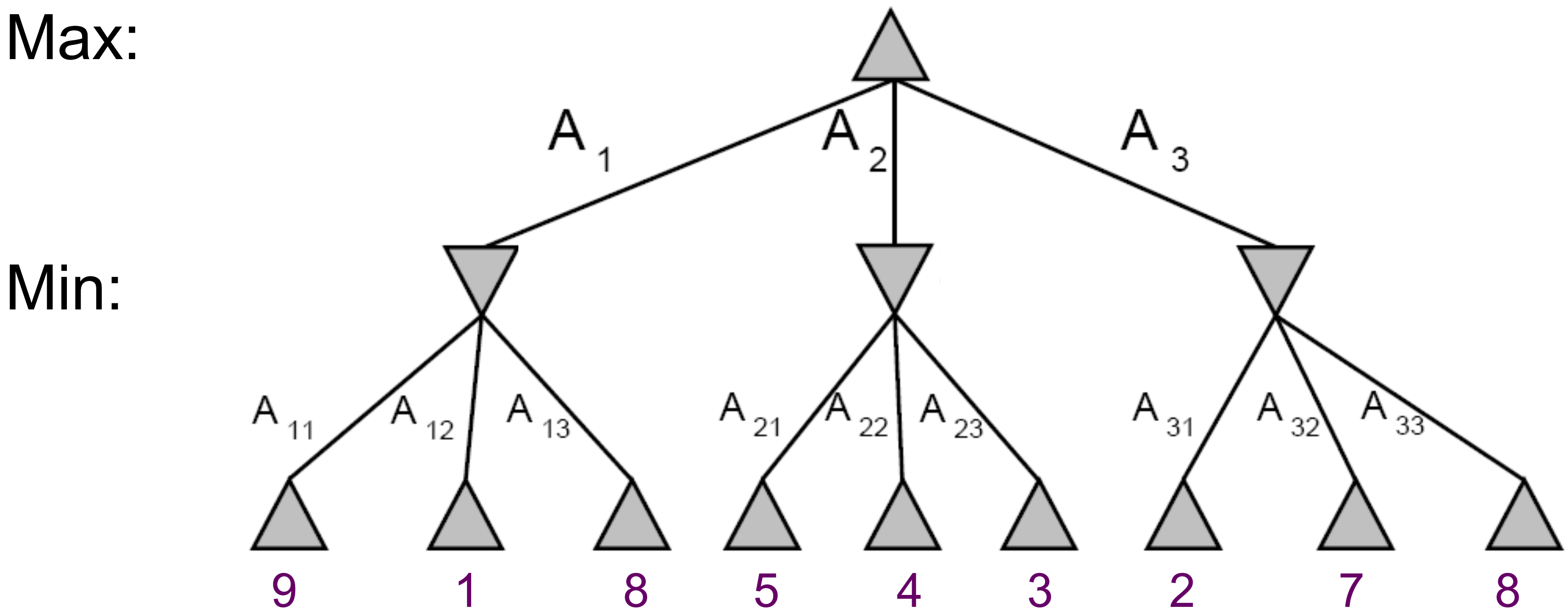
Concrete Minimax Example



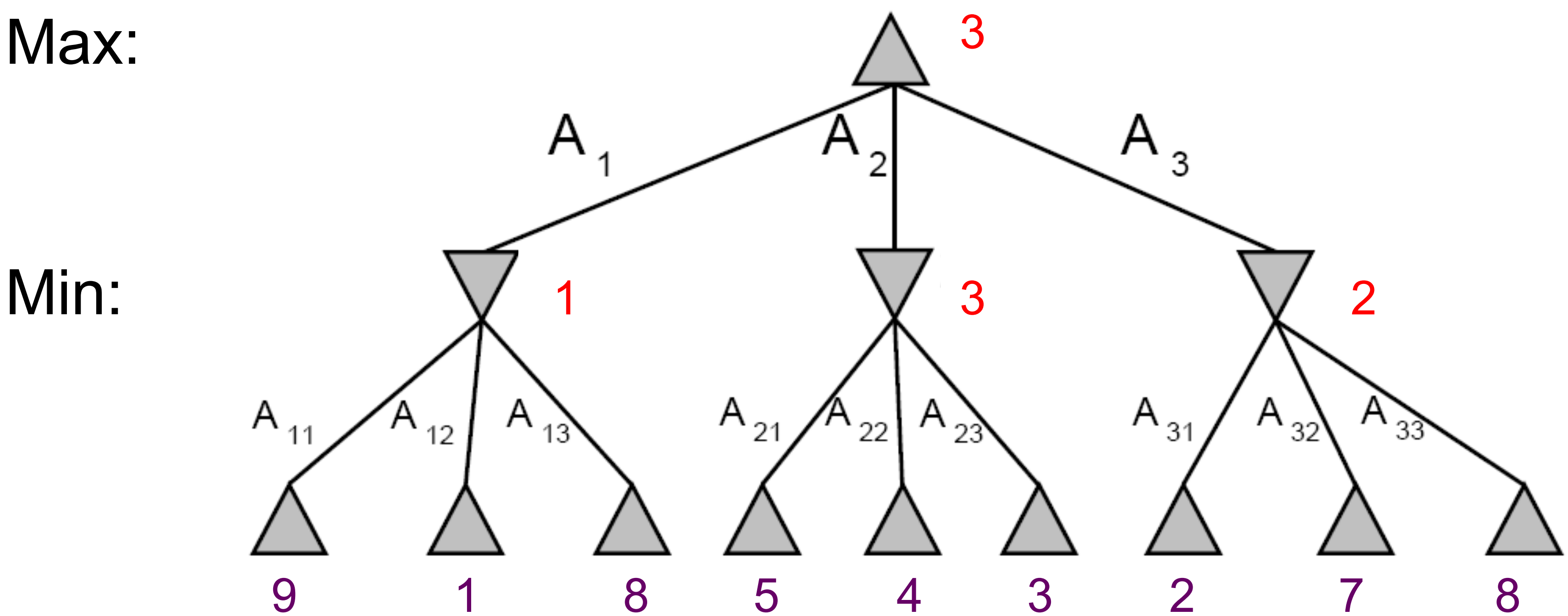
Minimax Example



Quiz



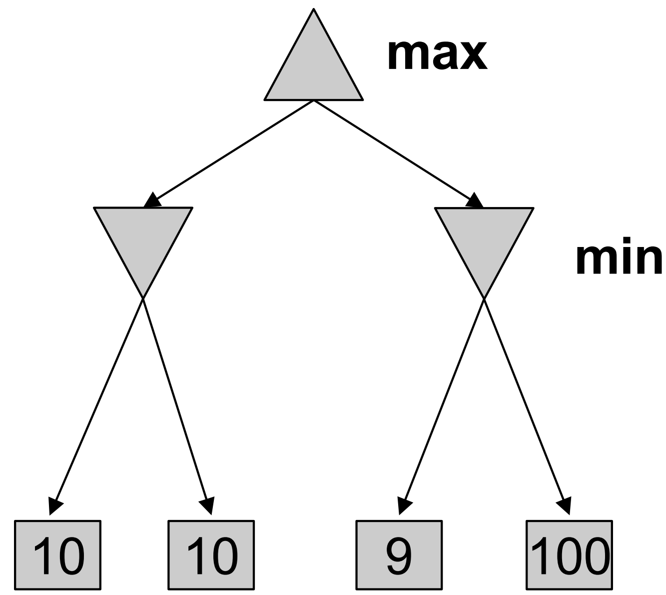
Answer



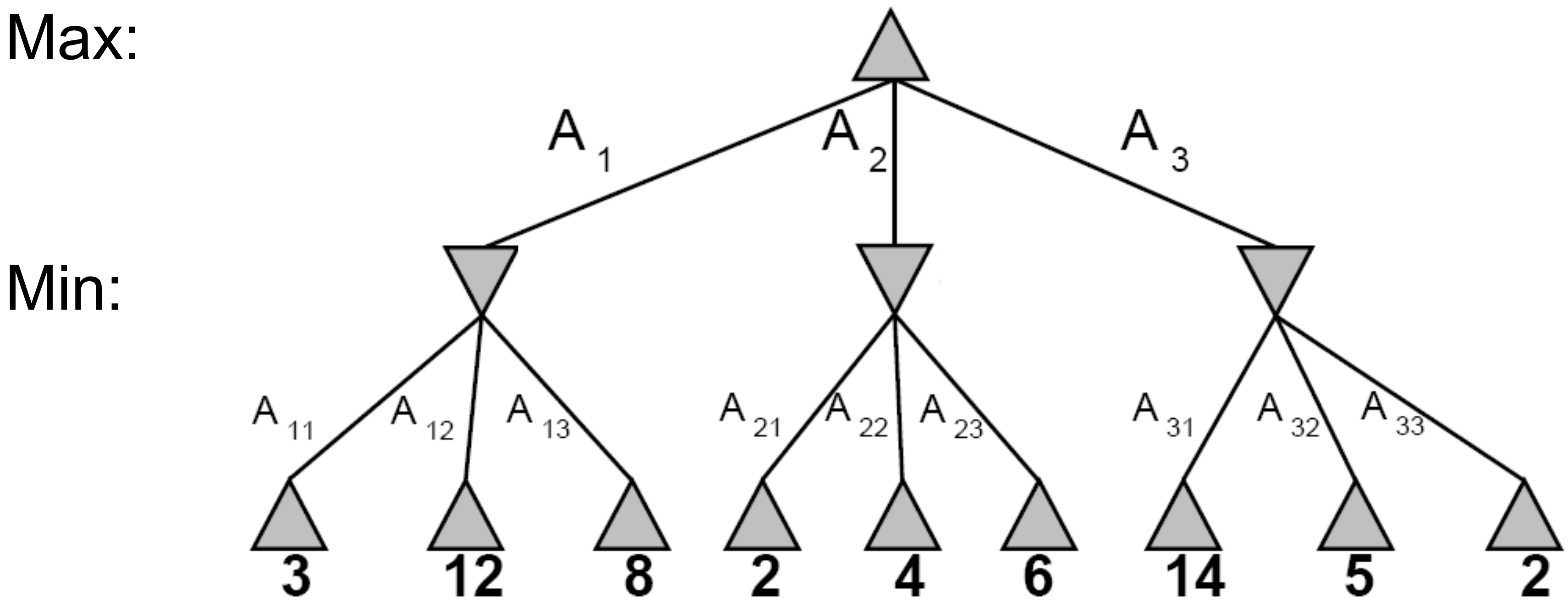
Max:
3
A 1
A 2
A 3
Min:
1
3
2
A 11
A 12
A 13
A 21
A 22
A 23
A 31
A 32
A 33
9
1
8
5
4
3
2
7
8

Minimax Properties

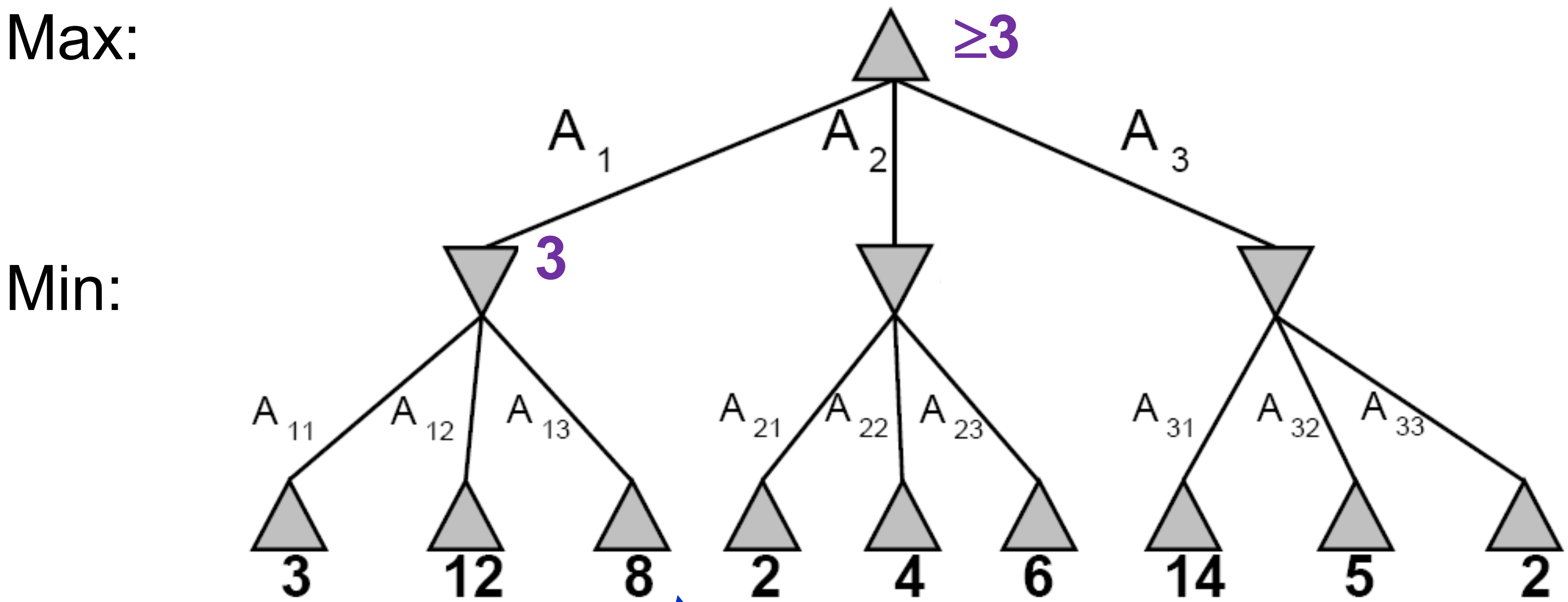
- Optimal?
  - Yes, against perfect player. Otherwise?
- Time complexity?
  - $O(b^m)$
- Space complexity?
  - $O(bm)$
- For chess, $b \sim 35$, $m \sim 100$
  - Exact solution is completely infeasible
  - **But**,… do we need to explore the whole tree?



Do We Need to Evaluate Every Node?

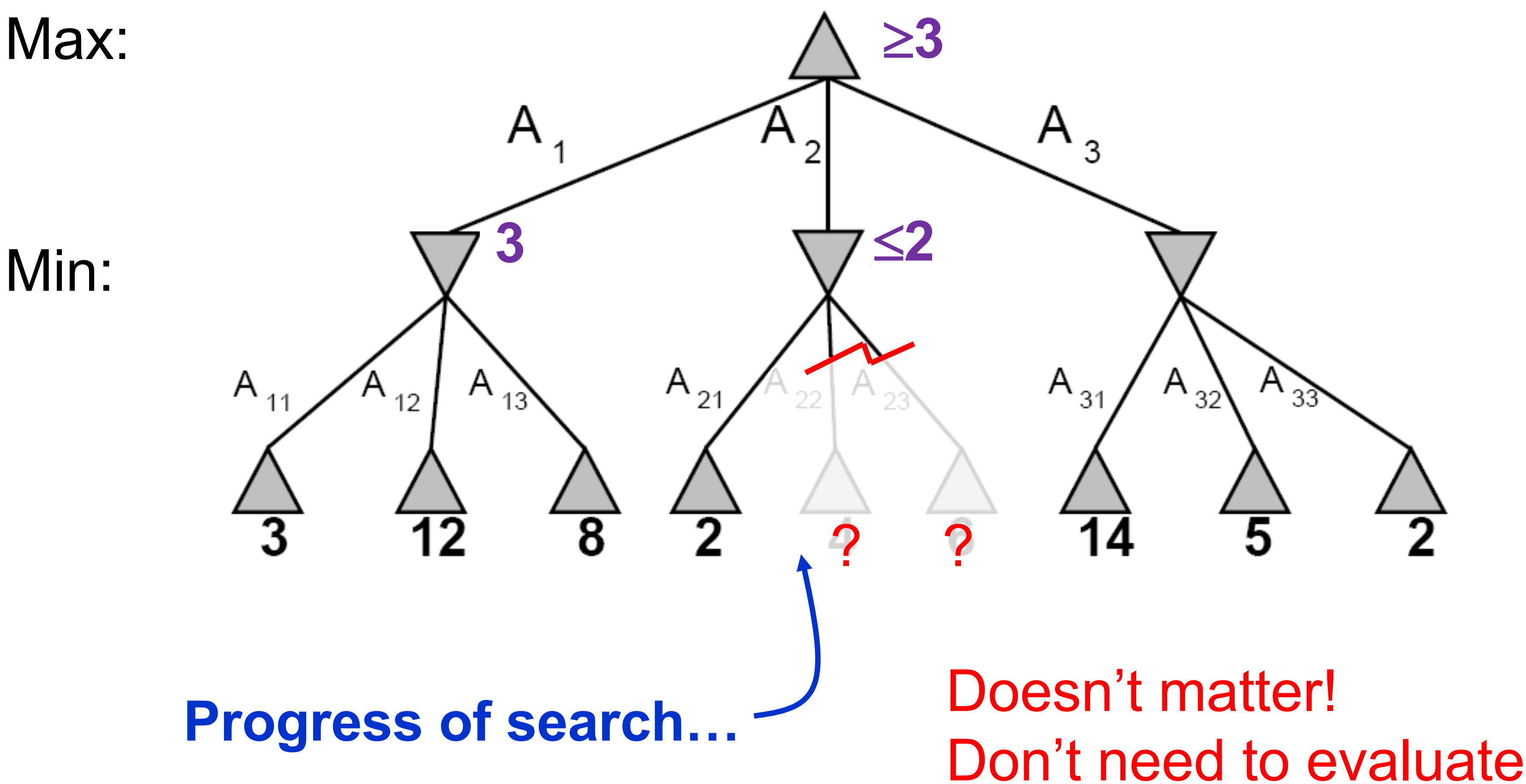


Do We Need to Evaluate Every Node?

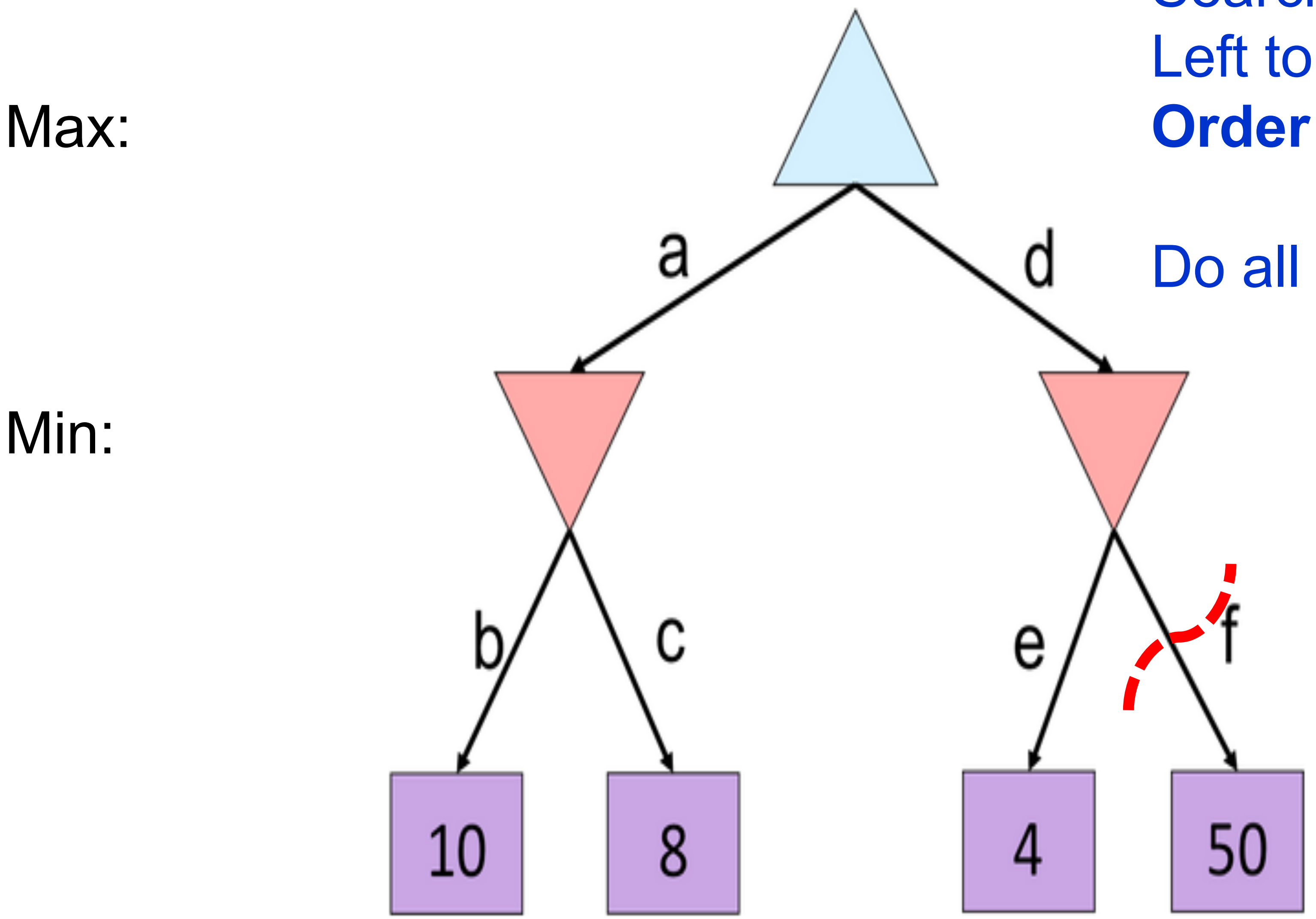


Progress of search…

α-β Pruning Example



Alpha-Beta Quiz



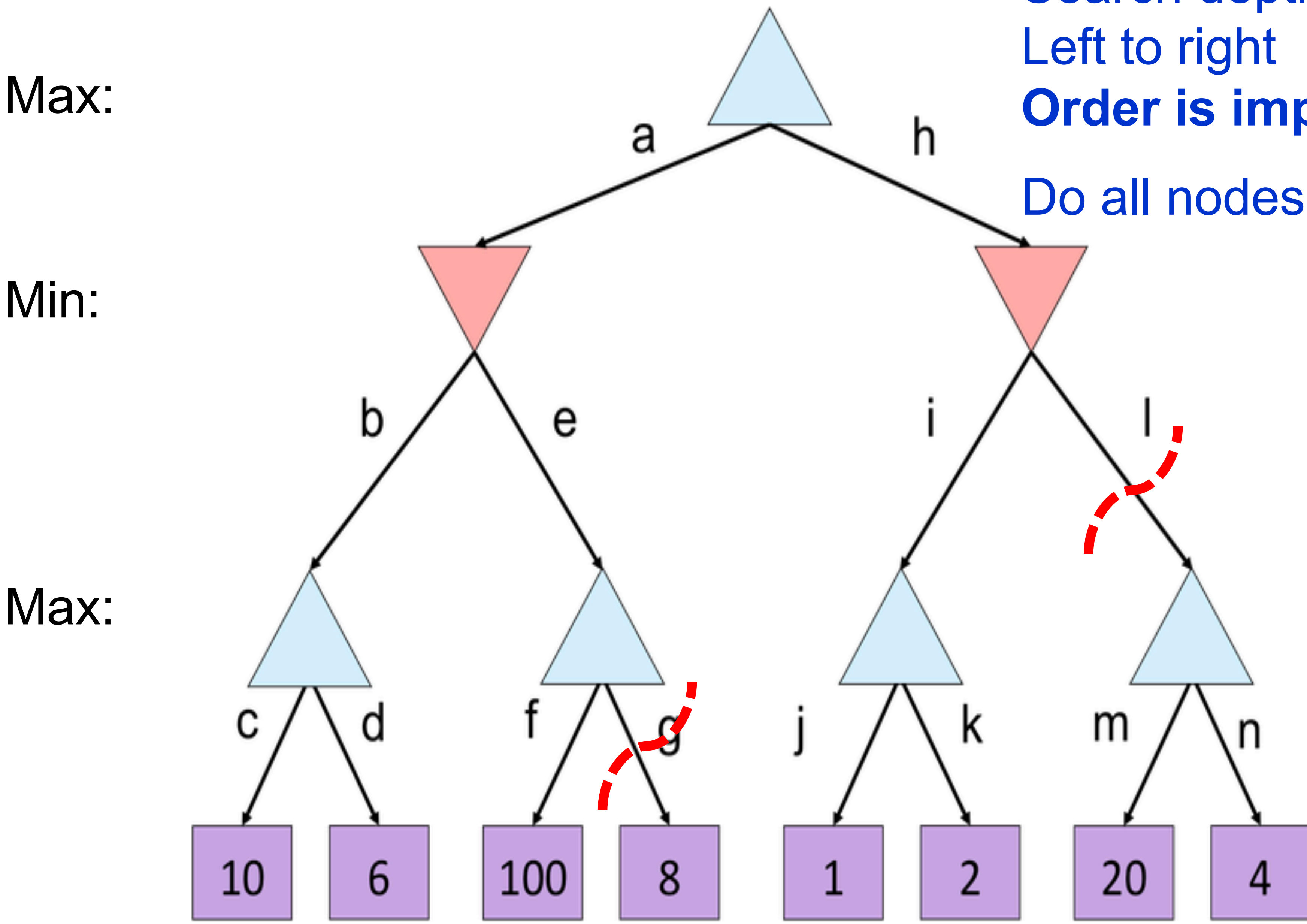
Search depth-first
Left to right
Order is important

Do all nodes matter?

Alpha-Beta Quiz 2

Search depth-first
Left to right
Order is important

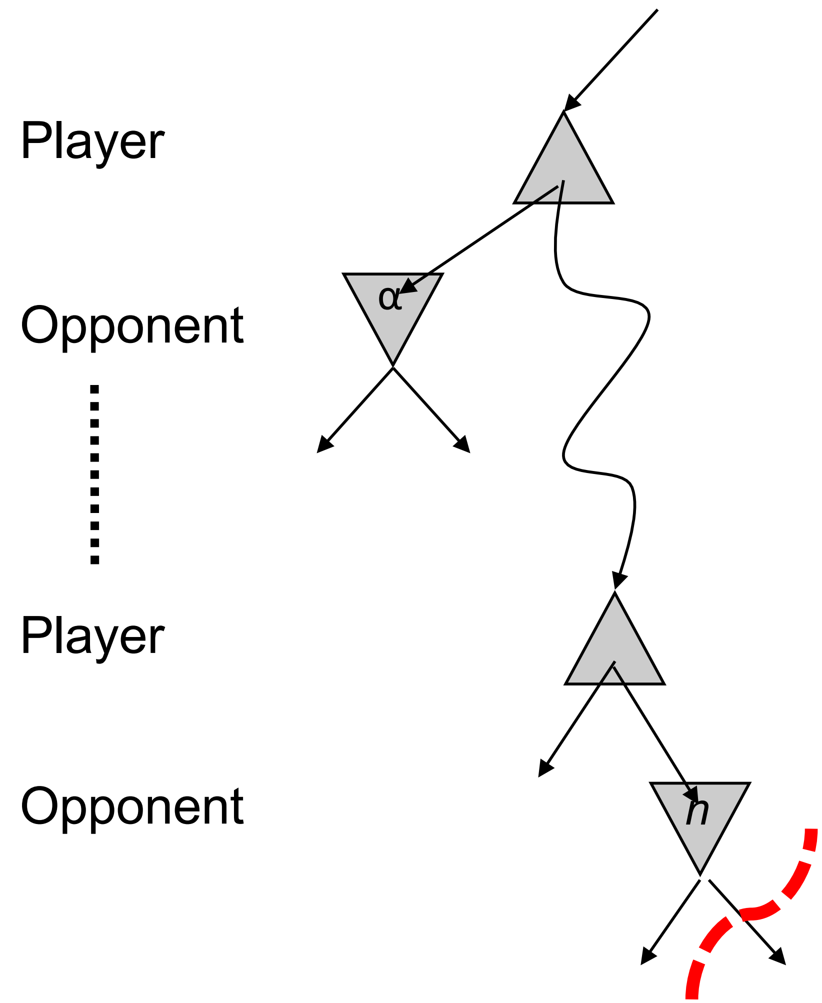
Do all nodes matter?



Slide adapted from Dan Klein & Pieter Abbeel - ai.berkeley.edu

α-β Pruning

- α is MAX's best choice on path to root
- If *n* becomes worse than α, MAX will avoid it, so can stop considering *n*'s other children
- Define β similarly for MIN



Min-Max Implementation

```
def max-val(state        ):
    if leaf?(state), return U(state)
    initialize v = -∞
    for each c in children(state):
        v = max(v, min-val(c        ))

    return v
```

```
def min-val(state        ):
    if leaf?(state), return U(state)
    initialize v = +∞
    for each c in children(state):
        v = min(v, max-val(c        ))

    return v
```

Slide adapted from Dan Klein & Pieter Abbeel - ai.berkeley.edu

Alpha-Beta Implementation

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

```
def max-val(state, α, β):
    if leaf?(state), return U(state)
    initialize v = -∞
    for each c in children(state):
        v = max(v, min-val(c, α, β))


    return v
```

```
def min-val(state , α, β):
    if leaf?(state), return U(state)
    initialize v = +∞
    for each c in children(state):
        v = min(v, max-val(c, α, β))


    return v
```

Alpha-Beta Implementation

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

```
def max-val(state, α, β):
    if leaf?(state), return U(state)
    initialize v = -∞
    for each c in children(state):
        v = max(v, min-val(c, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

```
def min-val(state, α, β):
    if leaf?(state), return U(state)
    initialize v = +∞
    for each c in children(state):
        v = min(v, max-val(c, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

Alpha-Beta Pruning Demo

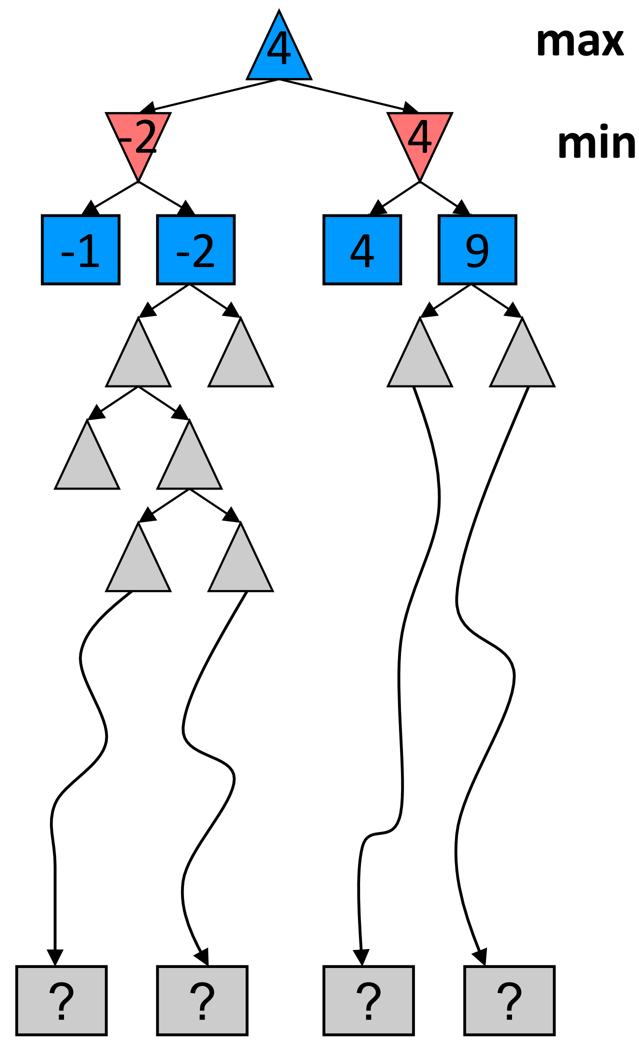
http://inst.eecs.berkeley.edu/~cs61b/fa14/ta-materials/apps/ab_tree_practice/

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 correct ***bounds***
- Good child ordering improves effectiveness of pruning
- With "perfect ordering":
  - Time complexity drops to $O(b^{m/2})$
  - ***Doubles*** solvable depth!
  - (But complete search of complex games, e.g. chess, is still hopeless…

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 3 min/move, can explore 1M nodes / sec
  - So can check 200M nodes per move
  - α-β reaches about depth 10 → decent chess program
- Guarantee of optimal play is gone
- More plies makes a BIG difference



Depth Matters

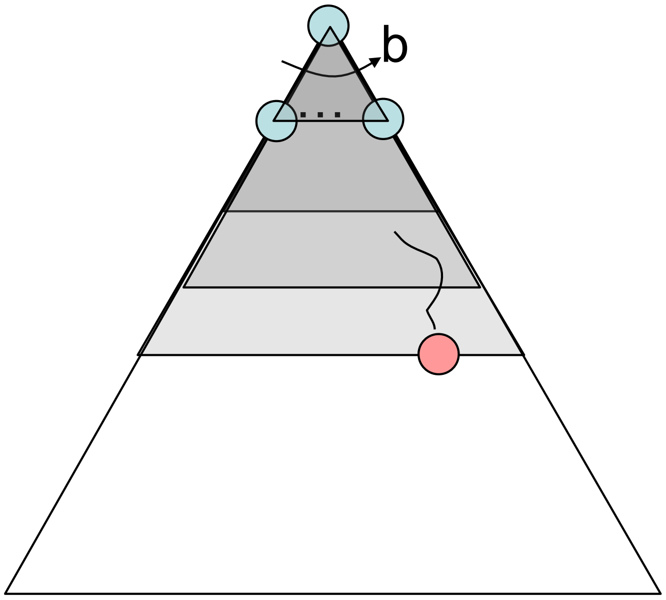
- Evaluation functions are always imperfect
- The deeper in the tree the evaluation function is buried, the less the quality of the evaluation function matters
- Good example of the tradeoff between complexity of ***features*** and complexity of ***computation***

Iterative Deepening

Iterative deepening uses DFS as a subroutine:

1. Do a DFS which only searches for paths of length 1 or less. (DFS gives up on any path of length 2)
2. If "1" ***fails***, do a DFS which only searches paths of length 2 or less.
3. If "2" ***fails***, do a DFS which only searches paths of length 3 or less.

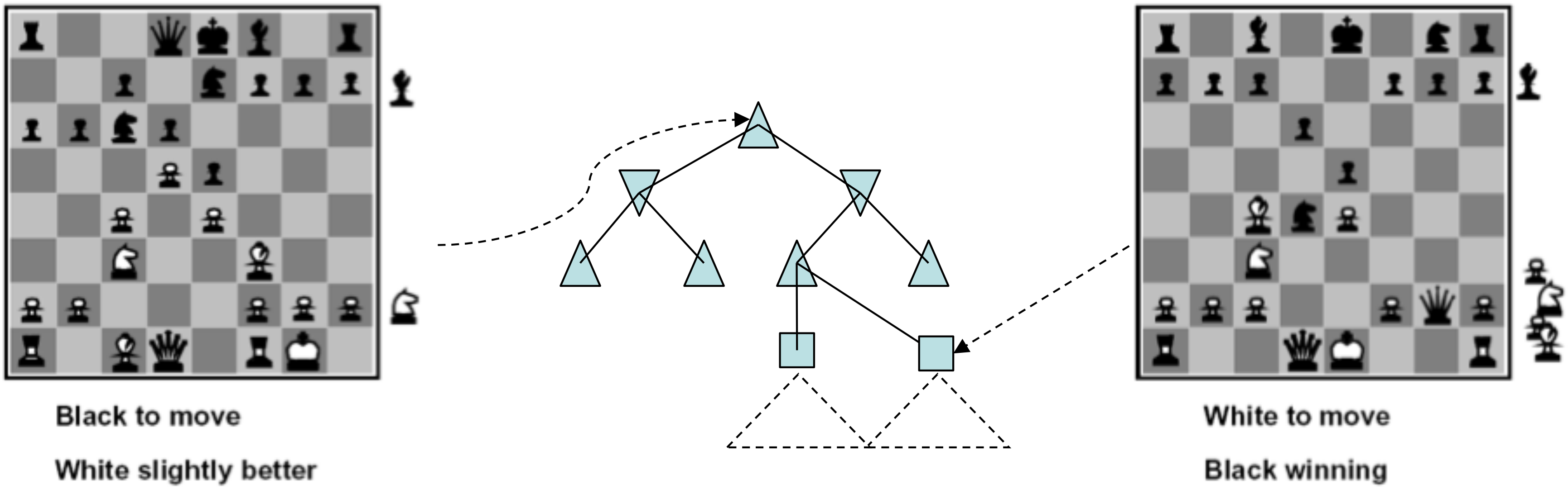
….and so on.



Can one adapt to games to make ***anytime algorithm*** ?

Heuristic Evaluation Function

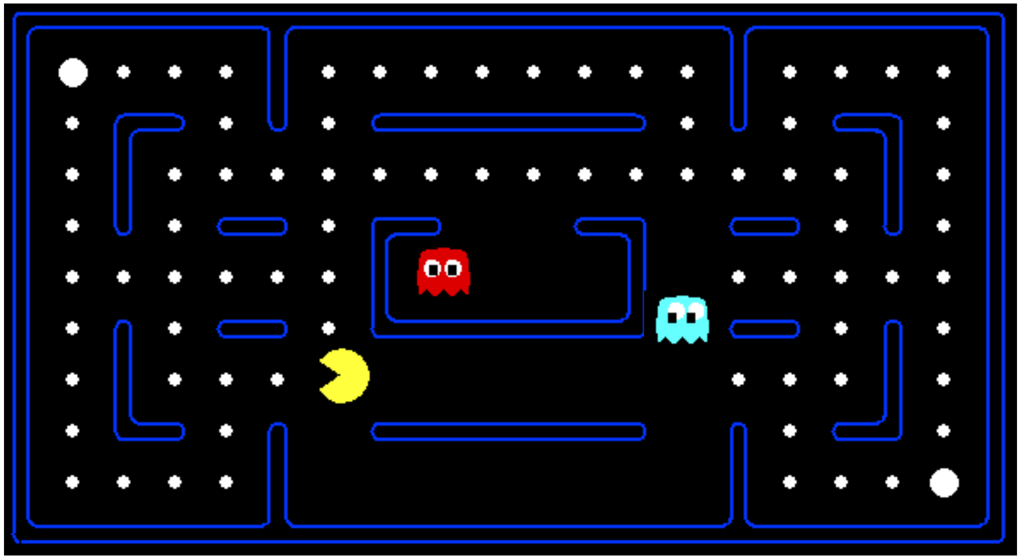
- Function which scores non-terminals



- Ideal function: returns the ***true utility*** of the position
- In practice: need a simple, fast ***approximation***
  - typically weighted linear sum of features:
  - e.g. $f_1(s)$ = (num white queens – num black queens), etc.

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

Evaluation for Pacman

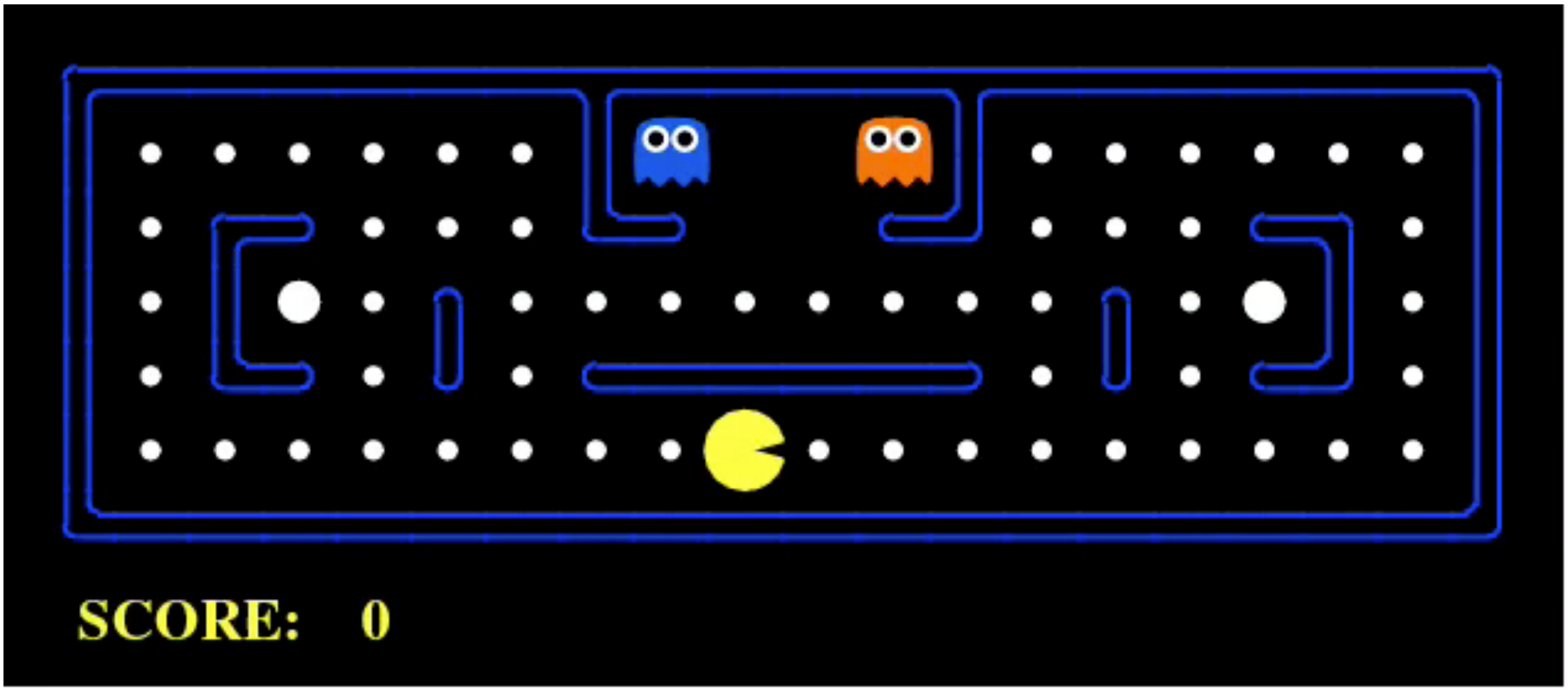


What features would be good for Pacman?

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

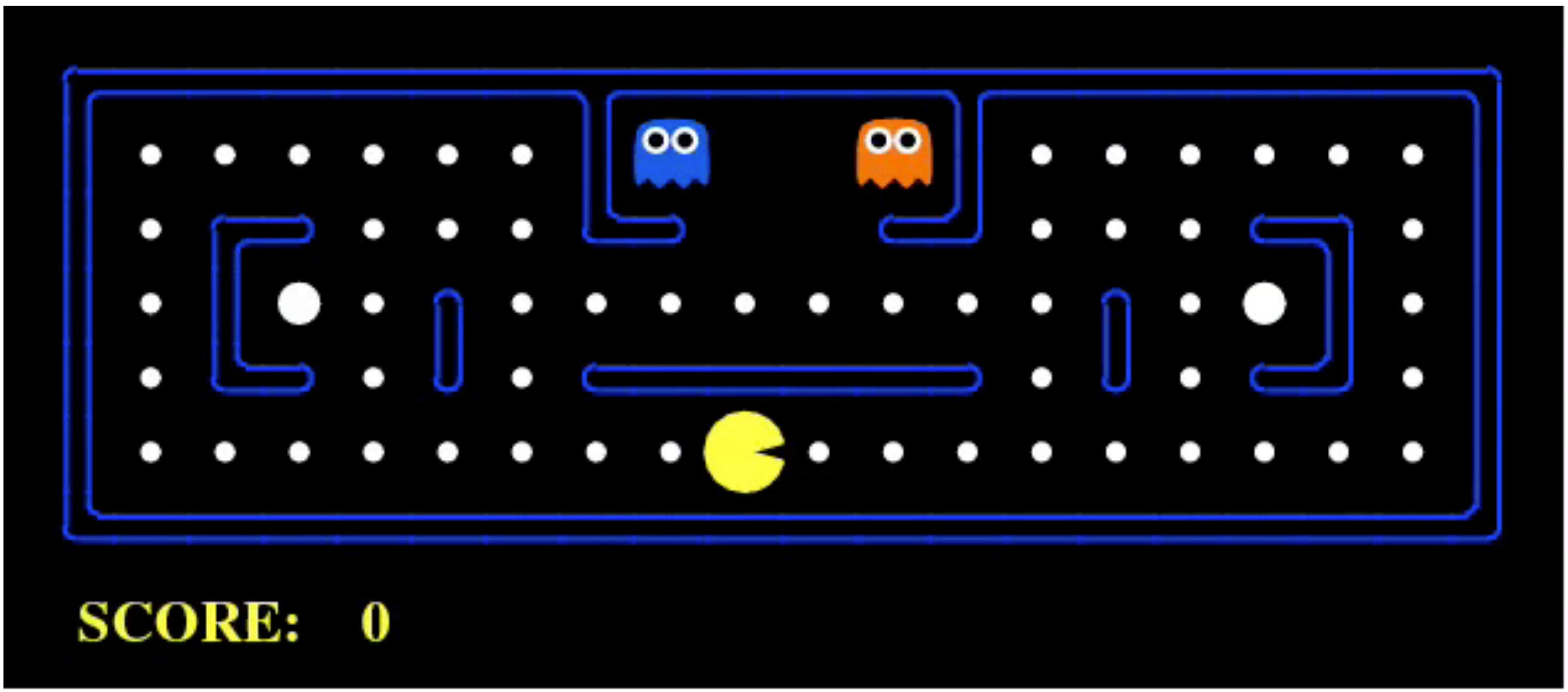
Which algorithm?

α-β, depth 4, simple eval fun



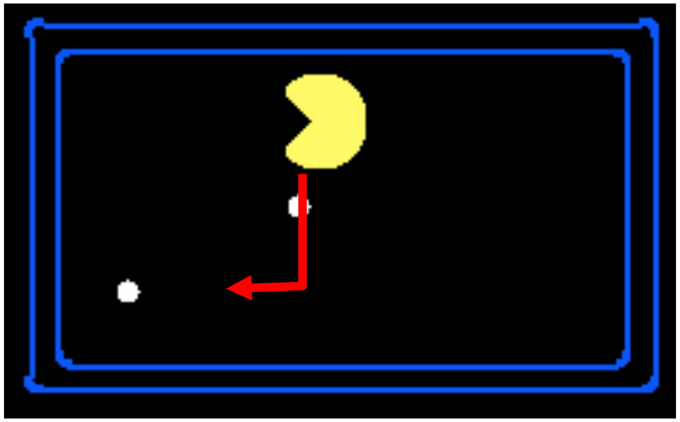
Which algorithm?

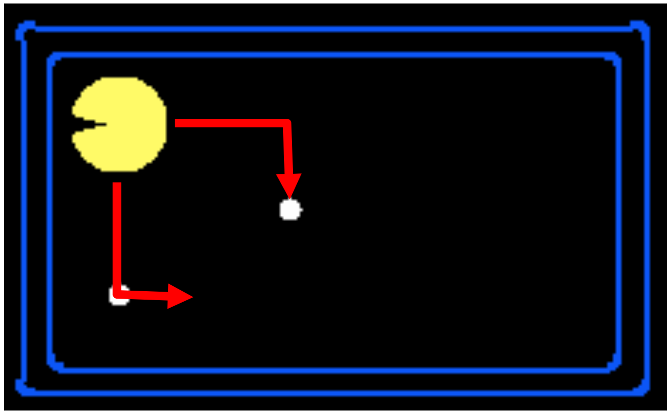
α-β, depth 4, better eval fun



Why Pacman Starves

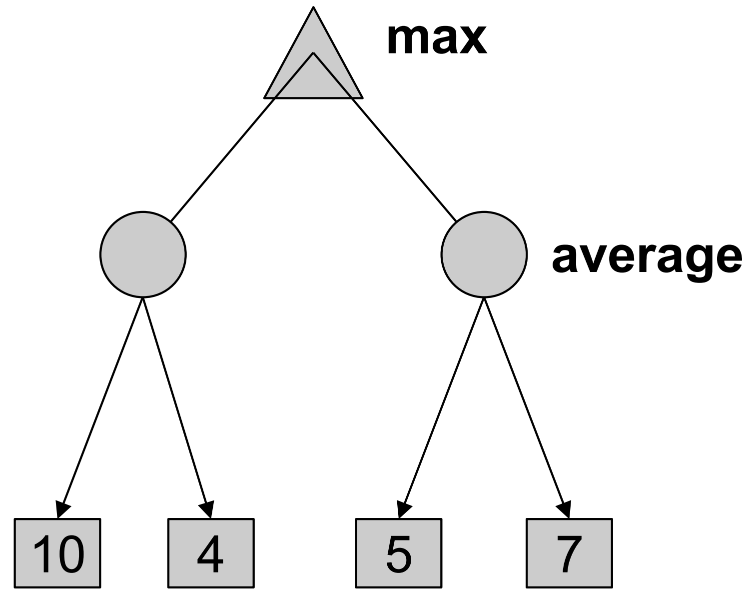
- He knows his score will go up by eating the dot now
- He knows his score will go up just as much by eating the dot later on
- There are no point-scoring opportunities after eating the dot
- Therefore, waiting seems just as good as eating





Stochastic Single-Player

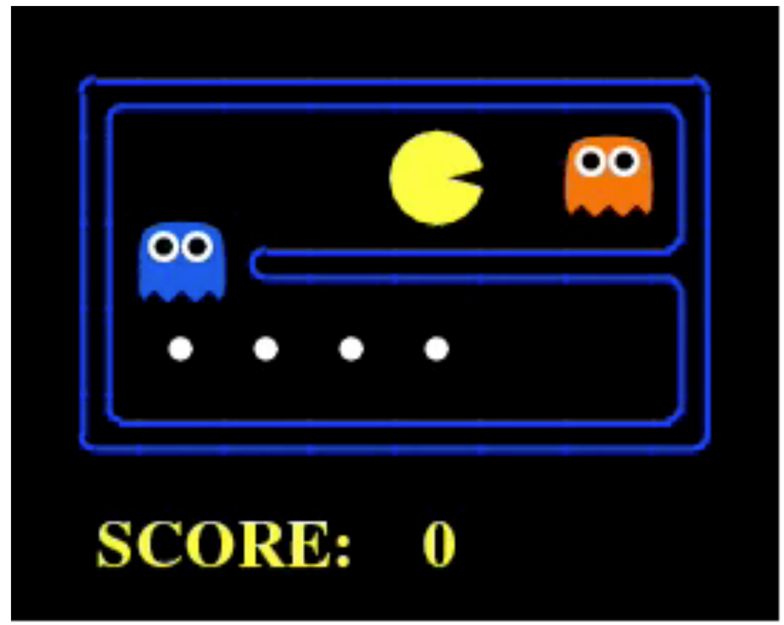
- What if we don't know what the result of an action will be? E.g.,
  - In solitaire, shuffle is unknown
  - In minesweeper, mine locations
- Can do **expectimax search**
  - Chance nodes, like actions except the environment controls the action chosen
  - Max nodes as before
  - Chance nodes take average (expectation) of value of children



Which Algorithms?

Expectimax

Minimax



3 ply look ahead, ghosts move randomly

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

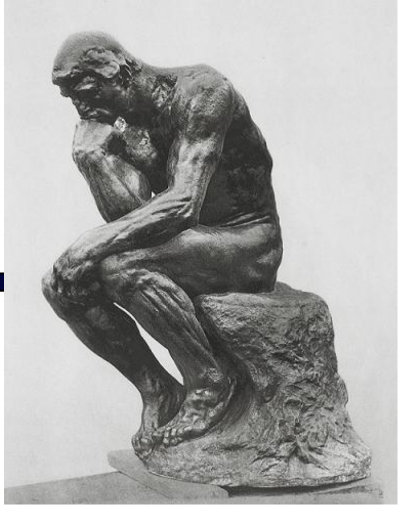
- A random variable represents an event whose outcome is unknown
- A probability distribution is an assignment of weights to outcomes

- 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) = 0.55, P(T=heavy) = 0.20

- 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) = 0.20,   P(T=heavy | Hour=8am) = 0.60
  - We'll talk about methods for reasoning and updating probabilities later

What are Probabilities?



- Objectivist / frequentist answer:
  - Averages over repeated *experiments*
  - E.g. empirically estimating P(rain) from historical observation
  - E.g. pacman's estimate of what the ghost will do, given what it has done in the past
  - Assertion about how future experiments will go (in the limit)
  - Makes one think of *inherently random* events, like rolling dice
- Subjectivist / Bayesian answer:
  - Degrees of belief about unobserved variables
  - E.g. an agent's belief that it's raining, given the temperature
  - E.g. pacman's belief that the ghost will turn left, given the state
  - Often *learn* probabilities from past experiences (more later)
  - New evidence *updates beliefs* (more later)

Uncertainty Everywhere

- Not just for games of chance!
  - I'm sick: will I sneeze this minute?
  - Email contains "FREE!": is it spam?
  - Tooth hurts: have cavity?
  - 60 min enough to get to the airport?
  - Robot rotated wheel three times, how far did it advance?
  - Safe to cross street? (Look both ways!)

- Sources of uncertainty in random variables:
  - Inherently random process (dice, etc)
  - Insufficient or weak evidence
  - Ignorance of underlying processes
  - Unmodeled variables
  - The world's just noisy – it doesn't behave according to plan!

Review: Expectations

- Real valued functions of random variables:

$$f : X \rightarrow R$$

- Expectation of a function of a random variable

$$E_{P(X)}[f(X)] = \sum_x f(x)P(x)$$

- Example: Expected value of a fair die roll

X	P	f
1	1/6	1
2	1/6	2
3	1/6	3
4	1/6	4
5	1/6	5
6	1/6	6

$$1 \times \frac{1}{6} + 2 \times \frac{1}{6} + 3 \times \frac{1}{6} + 4 \times \frac{1}{6} + 5 \times \frac{1}{6} + 6 \times \frac{1}{6}$$

$$= 3.5$$

Utilities

- Utilities are functions from outcomes (states of the world) to real numbers that describe an agent's preferences

- Where do utilities come from?
  - In a game, may be simple (+1/-1)
  - Utilities summarize the agent's goals
  - Theorem: any set of preferences between outcomes can be summarized as a utility function (provided the preferences meet certain conditions)

- In general, we hard-wire utilities and let actions emerge (why don't we let agents decide their own utilities?)

- More on utilities soon…