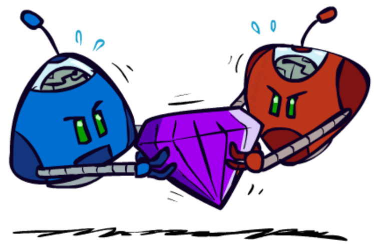
CSE 473: Artificial Intelligence

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)

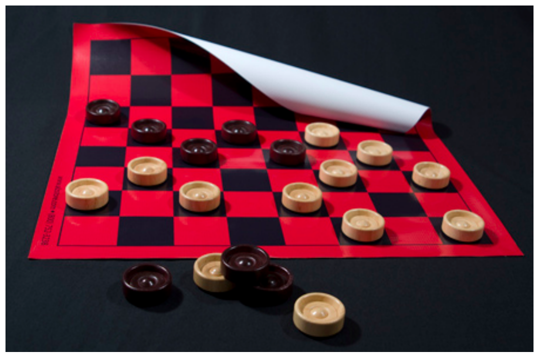
Outline

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

- Reminder:
  - Project 2 due in 7 days

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

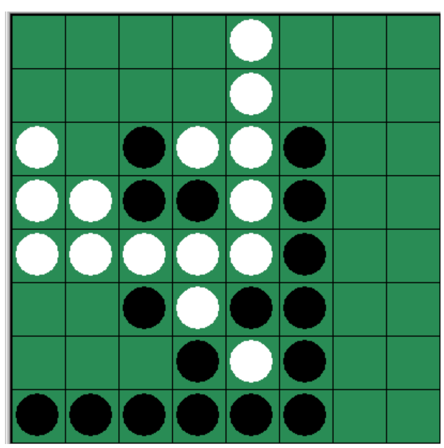
Go: b > 300! Programs use monte carlo tree search + pattern KBs

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

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

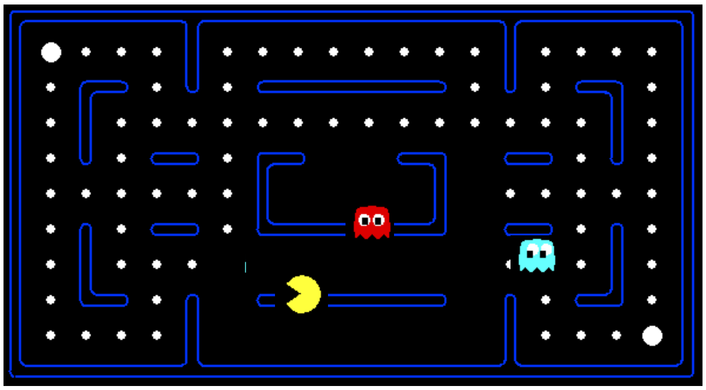
Game Playing State-of-the-Art

Othello: Human champions refuse to compete against computers.



Game Playing State-of-the-Art

- **Pacman:** … unknown …



Types of Games

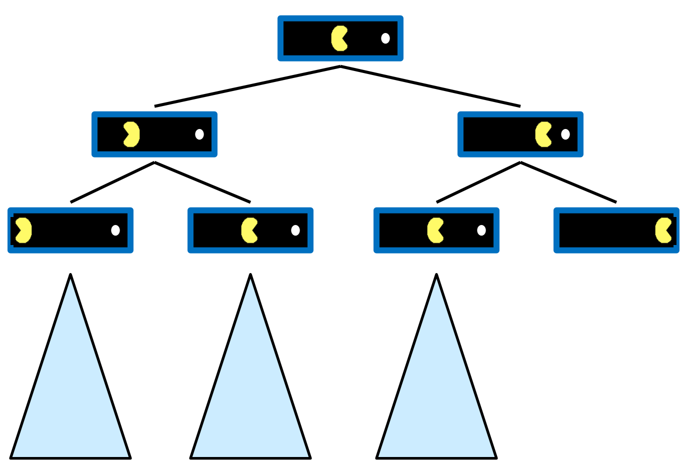
	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon, monopoly
imperfect information	stratego	bridge, poker, scrabble, nuclear war

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

Previously: Single-Agent Trees



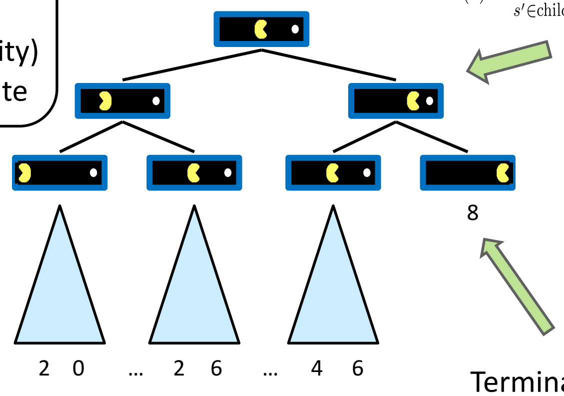
Slide from Dan Klein & Pieter Abbeel - ai.berkeley.edu

Previously: Value of a State

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

Non-Terminal States:

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

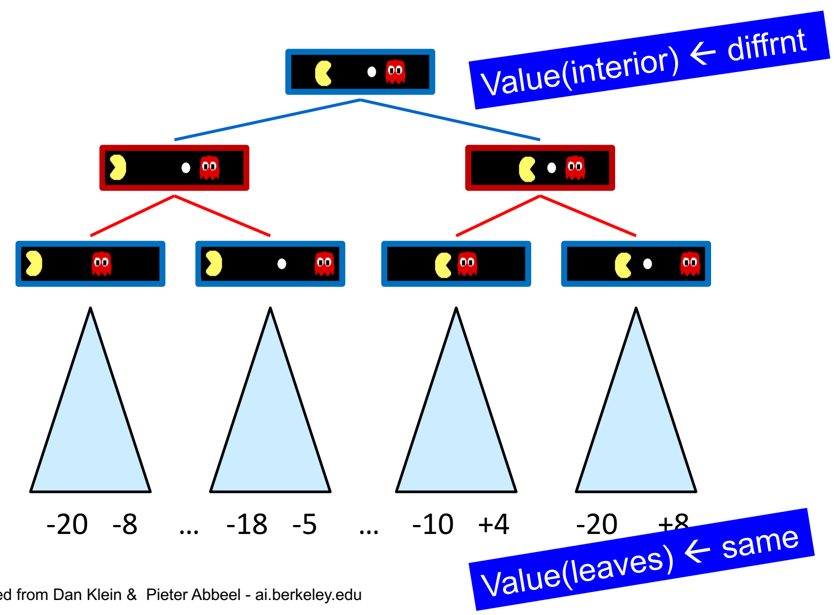


Terminal States:

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

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

Adversarial Game Trees



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

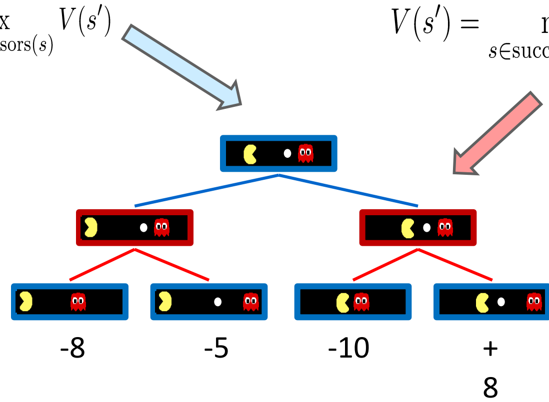
Minimax Values

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:

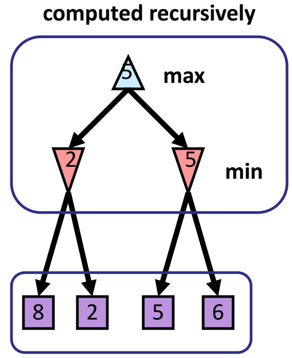
$$V(s) = \text{known}$$

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

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 values: computed recursively



Terminal values: part of the game

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

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
```

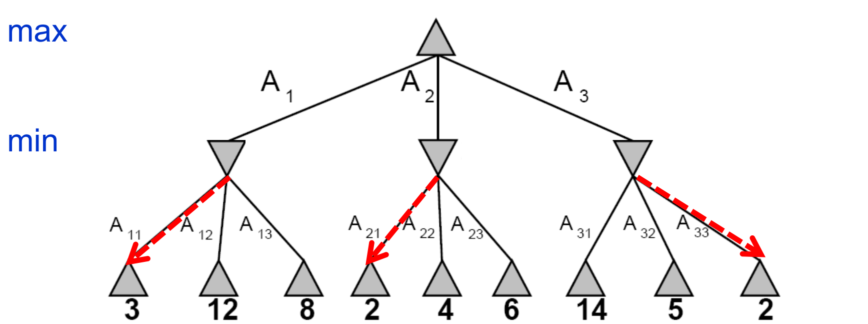
```
def min-value(state):
    if leaf?(state), return U(state)
    initialize v = +∞
    for each c in children(state)
        v = min(v, max-value(c))
    return v
```

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

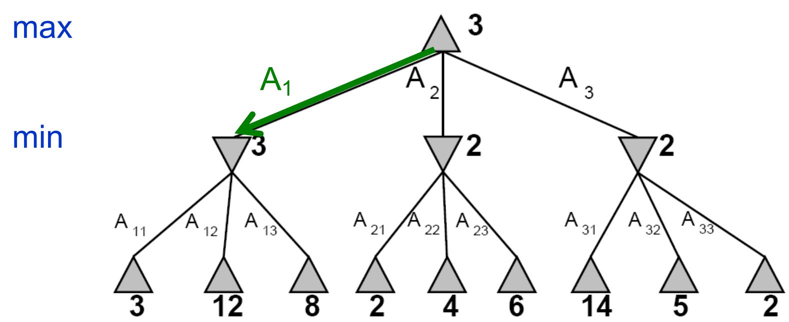
$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$

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

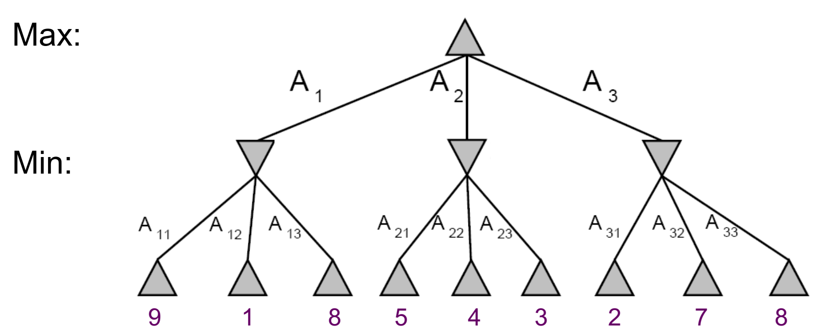
Concrete Minimax Example



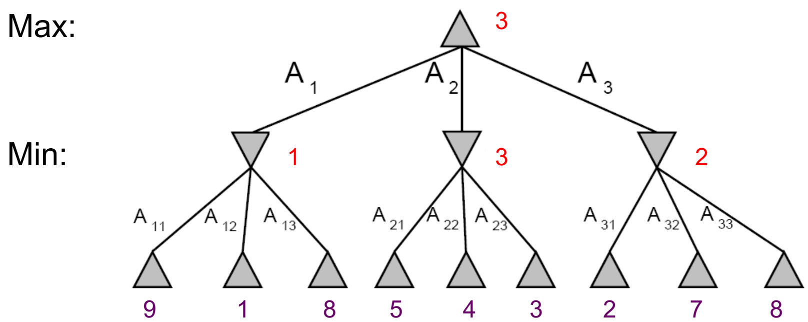
Minimax Example



Quiz

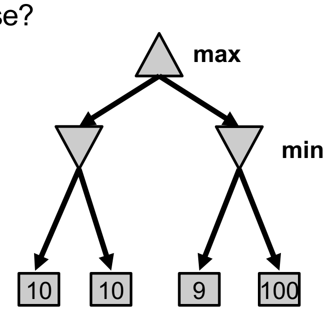


Answer

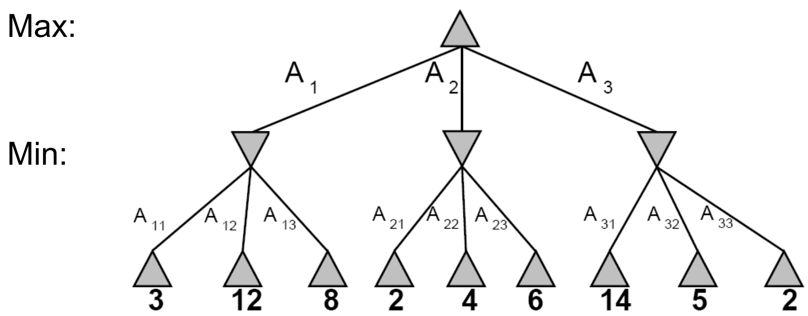


Minimax Properties

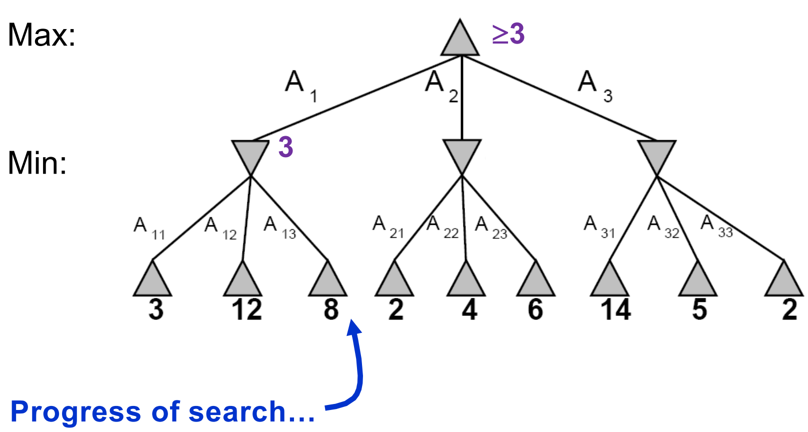
- Optimal?
  - Yes, against perfect player. Otherwise?
- Time complexity?
  - $O(b^m)$
- Space complexity?
  - $O(bm)$
- For chess, b ~ 35, m ~ 100
  - Exact solution is completely infeasible
  - But, do we need to explore the whole tree?



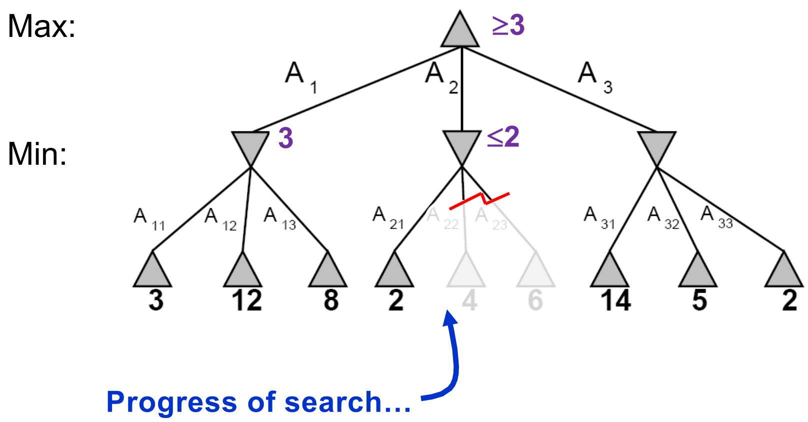
Do We Need to Evaluate Every Node?



Do We Need to Evaluate Every Node?

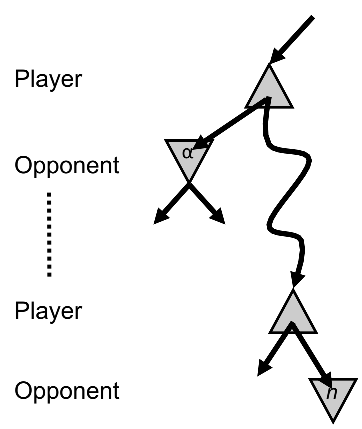


α-β Pruning Example



α-β Pruning

- General configuration
  - 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
```

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))
        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
```

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