

# Search Overview

- Introduction to Search
- Blind Search Techniques
- Heuristic Search Techniques
- Game Playing search
  - Perfect play
  - Resource limits
  - $\alpha$ - $\beta$  pruning
  - Games of chance
- Constraint Satisfaction Problems
- Stochastic Algorithms

## Games vs. search problems

- “Unpredictable” opponent
  - ⇒ solution  $\equiv$  contingency plan
- Time limits
  - ⇒ unlikely to find goal
  - ⇒ must approximate
- Plan of attack:
  - algorithm for perfect play  
[von Neumann, 1944]
  - finite horizon, approximate evaluation  
[Zuse, 1945; Shannon, 1950; Samuels, 1952–57]
  - pruning to reduce costs  
[McCarthy, 1956]

# Types of games

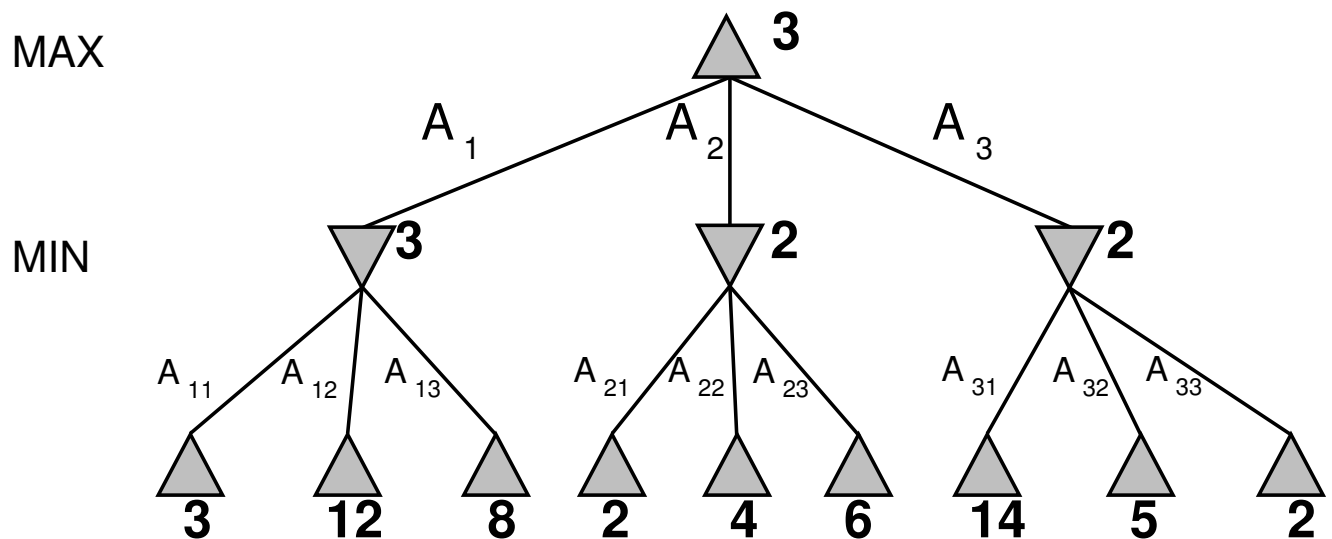
|                       | deterministic                   | chance                                 |
|-----------------------|---------------------------------|--|
| perfect information   | chess, checkers,<br>go, othello | backgammon<br>monopoly                 |
| imperfect information |                                 | bridge, poker, scrabble<br>nuclear war |

# Minimax

- Perfect play for deterministic, perfect-information games

Idea: choose move leading to position with highest *minimax value*  
≡ best achievable payoff against best play

- Eg, 2-ply game:



## Minimax algorithm

```
function Minimax-Decision(game) returns an operator  
  for each op in Operators[game] do  
    Value[op] ← Minimax-Value(Apply(op, game), game)  
  end  
  return the op with the highest Value[op]
```

---

```
function Minimax-Value(state, game) returns a utility value  
  if Terminal-Test[game](state) then  
    return Utility[game](state)  
  else if max is to move in state then  
    return the highest Minimax-Value of Successors(state)  
  else  
    return the lowest Minimax-Value of Successors(state)
```

## Properties of minimax

**Complete: ??**

**Optimal: ??**

**Time complexity: ??**

**Space complexity: ??**

## Properties of minimax

**Complete:** Yes, if tree is finite

[chess has specific rules for this]

**Optimal:** Yes, against an optimal opponent.

Otherwise??

**Time complexity:**  $O(b^m)$

**Space complexity:**  $O(bm)$

(depth-first exploration)

For chess:

$b \approx 35$ ,  $m \approx 100$  for “reasonable” games

$\Rightarrow$  exact solution completely infeasible

## Resource Limits

- Chess has  $\approx 10^{40}$  pos'ns;  
 $10^{10^{50}}$  possible games

<http://mathworld.wolfram.com/Chess.html>

- Suppose we have 10 seconds/move.

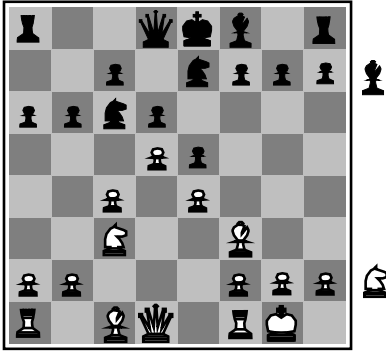
If explore  $10^9$  nodes/second

$\Rightarrow$   $10^{10}$  nodes per move

Not NEARLY enough!

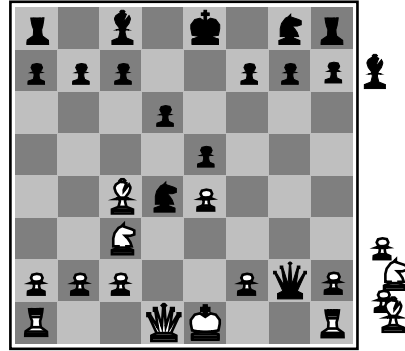
- Standard approach:
  - *cutoff test*  
eg, depth limit  
(perhaps add *quiescence search*)
  - *evaluation function*  
= *estimated* desirability of position

# Evaluation Functions



Black to move

White slightly better



White to move

Black winning

- Typically *linear* weighted sum of features

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

Eg, chess:

Approximation ...  $w_1 = 9$

$f_1(s) = \#WhiteQueens - \#BlackQueens$

$w_2 = 5$

$f_2(s) = \#WhiteRooks - \#BlackRooks$

...

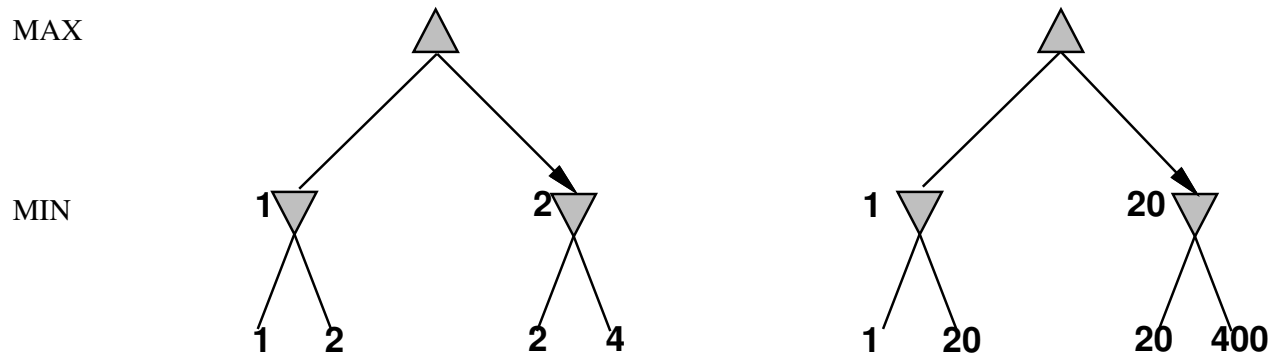
$w_5 = 0.3$

$f_5(s) = \text{White'sControlOfCenter}$

...

- Which features  $f_i(\cdot)$ ?  
 What values for  $w_i$ ?  
 ⇒ Machine Learning!

## Digression: Exact values don't matter



- Behaviour is preserved under any *monotonic* transformation of Eval
- Only the order matters: payoff in deterministic games acts as *ordinal utility* function

## Cutting off search

- MinimaxCutoff  $\equiv$  MinimaxValue except
  1. Terminal? is replaced by Cutoff?
  2. Utility is replaced by Eval

- Does it work in practice?

$$b^m = 10^6, \quad b = 35 \quad \Rightarrow \quad m = 4$$

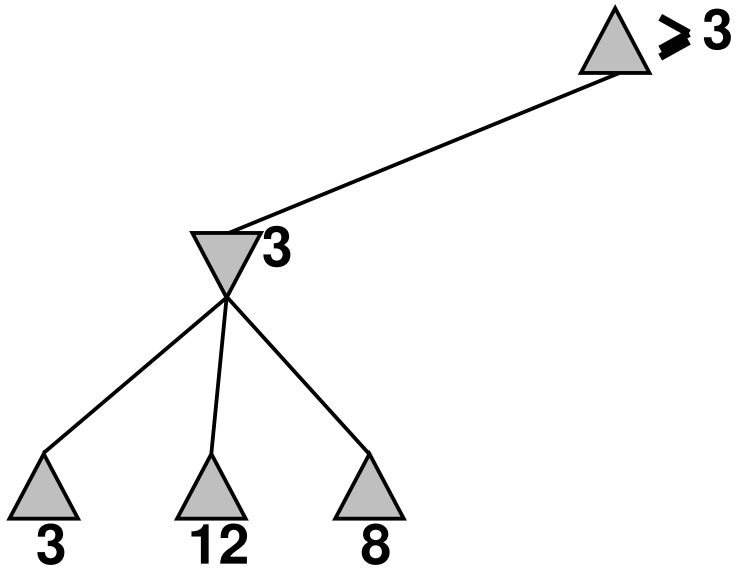
4-ply lookahead is a hopeless chess player!

- 4-ply  $\approx$  human novice  
8-ply  $\approx$  typical PC, human master  
12-ply  $\approx$  Deep Blue, Kasparov
- to do better . . .

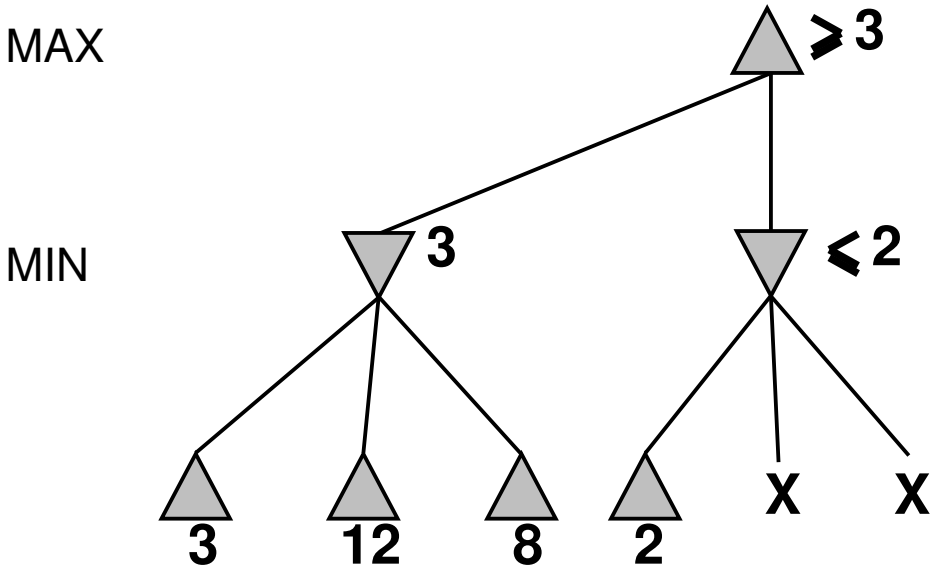
$\alpha$ - $\beta$  pruning example

MAX

MIN



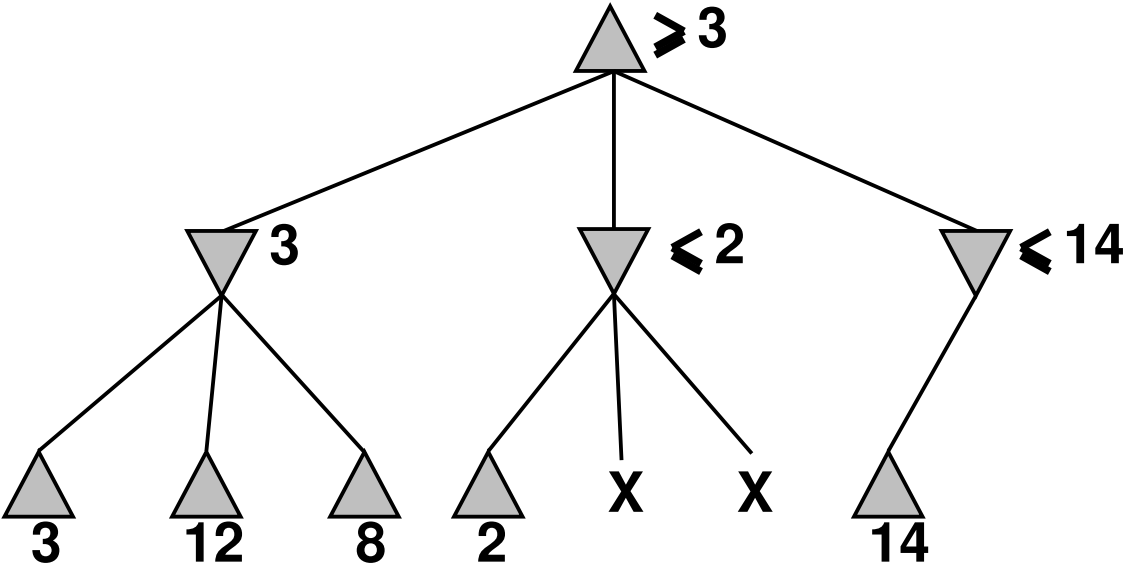
$\alpha$ - $\beta$  pruning example



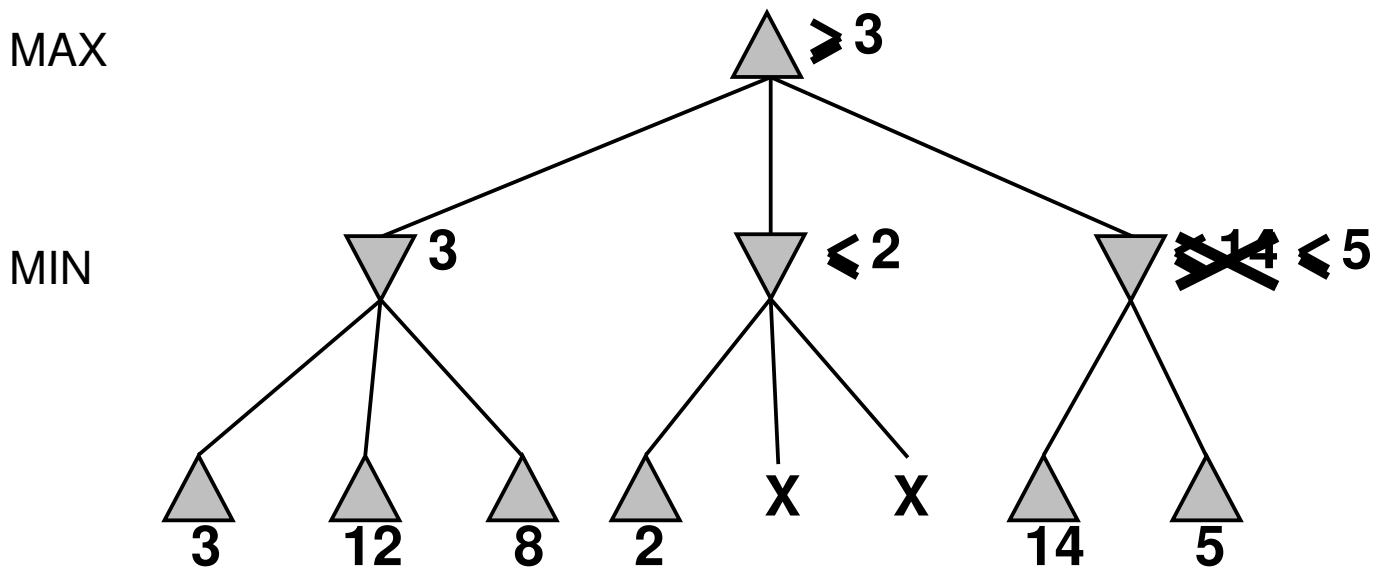
$\alpha$ - $\beta$  pruning example

MAX

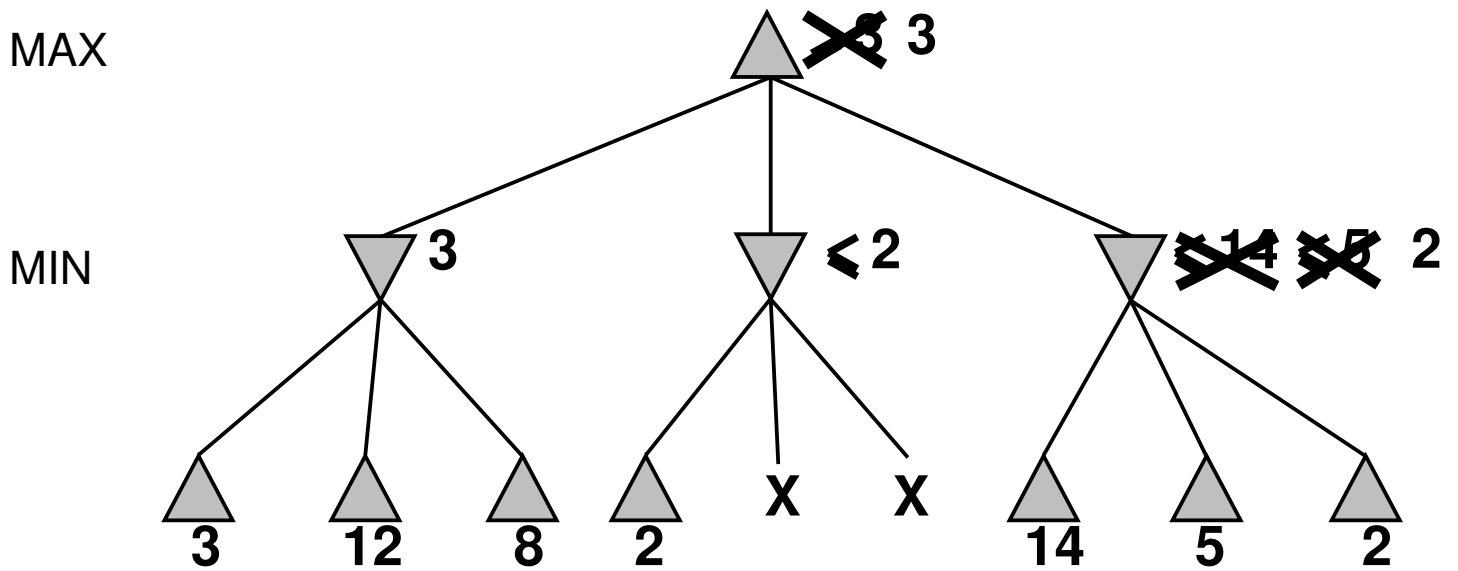
MIN



$\alpha$ - $\beta$  pruning example



$\alpha$ - $\beta$  pruning example

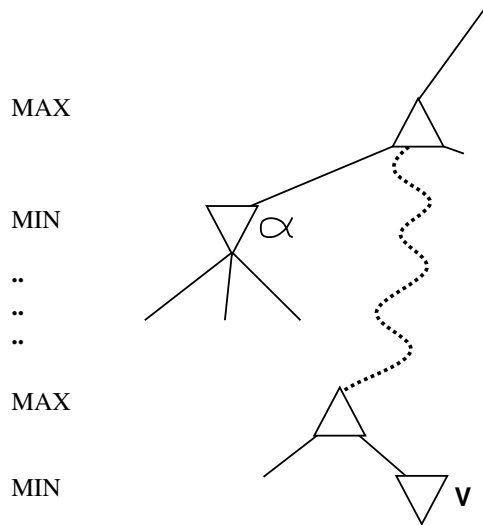


## Properties of $\alpha-\beta$

- Pruning *does not* affect final result
- Good move ordering improves effectiveness of pruning
- With “perfect ordering” :
  - time complexity =  $O(b^{m/2})$
  - $\Rightarrow$  *doubles* depth of search
  - $\Rightarrow$  can easily reach depth 8
  - $\Rightarrow$  play good chess!
- Shows value of “*metareasoning*” :

Reasoning about which computations are relevant

# Why is it called $\alpha$ - $\beta$ ?



- $\alpha$  = best value (to max) found so far, off current path
- If  $V$  is worse than  $\alpha$ , max will avoid it  $\Rightarrow$  prune that branch
- Define  $\beta$  similarly for min

## The $\alpha$ - $\beta$ algorithm

- Basically Minimax + keep track of  $\alpha$ ,  $\beta$   
+ prune

```
function Max-Value(state, game,  $\alpha$ ,  $\beta$ ) returns the minimax value of state  
inputs: state, current state in game  
         game, game description  
          $\alpha$ , the best score for max along the path to state  
          $\beta$ , the best score for min along the path to state  
  
if Cutoff-Test(state) then return Eval(state)  
for each s in Successors(state) do  
     $\alpha \leftarrow$  Max( $\alpha$ , Min-Value(s, game,  $\alpha$ ,  $\beta$ ))  
    if  $\alpha \geq \beta$  then return  $\beta$   
end  
return  $\alpha$ 
```

---

```
function Min-Value(state, game,  $\alpha$ ,  $\beta$ ) returns the minimax value of state  
  
if Cutoff-Test(state) then return Eval(state)  
for each s in Successors(state) do  
     $\beta \leftarrow$  Min( $\beta$ , Max-Value(s, game,  $\alpha$ ,  $\beta$ ))  
    if  $\beta \leq \alpha$  then return  $\alpha$   
end  
return  $\beta$ 
```

## Deterministic games in practice

**Checkers:** *Chinook* ended 40-yr-reign of human world champion Marion Tinsley [1994].

- Endgame database for *perfect play* for *all* positions involving  $\leq 8$  pieces on board. . . . . 443,748,401,247 positions!

**Chess:** *Deep Blue* defeated human world champion Gary Kasparov in 6-game match [1997].

- 200 million positions/sec
- very sophisticated evaluation
- undisclosed methods for extending some lines of search, up to 40 ply!

**Othello:** human champions refuse to compete against computers, who are too good!

**Go:** human champions refuse to compete against computers, who are too bad!

$b > 300 \Rightarrow$  most programs use *pattern knowledge bases* to suggest plausible moves



## Algorithm for nondeterministic games

- Expectiminimax gives perfect play

Just like Minimax, but also handles chance nodes:

...

**if** *state* is chance node **then**

**return** average of ExpectiMinimax-Value  
        of Successors(*state*)

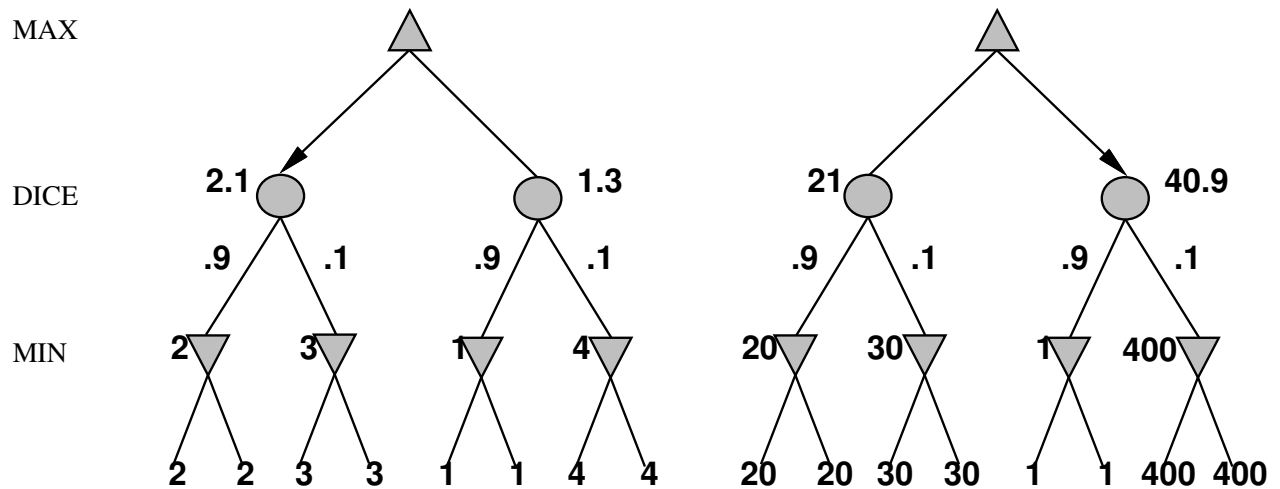
...

- A version of  $\alpha$ - $\beta$  pruning is possible  
(needs bounded leaf values)

## Nondeterministic games in practice

- Dice rolls increase  $b$ :  
21 possible rolls with 2 dice  
  
Backgammon  $\approx$  20 legal moves  
(6,000 with 1-1 roll)  
  
depth 4  $\Rightarrow 20 \times (21 \times 20)^3 \approx 1.2 \times 10^9$
- As depth increases,  
probability of reaching given node shrinks  
 $\Rightarrow$  value of lookahead is diminished
- $\alpha$ - $\beta$  pruning is much less effective
- TDGammon uses  
depth-2 search + very good Eval  
 $\approx$  world-champion level

## Digression: Exact values DO matter



- Behaviour is preserved only by *positive linear* transformation of Eval

⇒ Eval should be proportional to expected payoff

## Summary

- Games are fun to work on!  
... but dangerous. . .
- Illustrate several important points about AI
  - perfection is unattainable  
⇒ must approximate
  - good idea to  
think about what to think about
  - uncertainty constrains assignment of  
values to states
- Games are to AI  
as  
grand prix racing is to automobile design