Game Playing

Introduction to Artificial Intelligence

CSE 150

Lecture 5

April 17, 2007

Tax Day!

Adapted from slides adapted from slides of Russell's by Kriegman

CSP Summary

- CSPs are a special kind of search problem:
  States defined by values of a fixed set of variables
  Goal test defined by constraints on variable values
- Backtracking = depth-first search with
  1) fixed variable order
  2) only legal successors
  3) Stop search path when constraints are violated.
- Forward checking prevents assignments that guarantee a later failure
- Heuristics:
  – Variable selection: most constrained variable
  – Value selection: least constraining value
- Iterative min-conflicts is usually effective in practice

READING

- Finish Chapter 5, Read Chapter 6!

This lecture

- Introduction to Games
- Perfect play (Min-Max)
  - Tic-Tac-Toe
  - Checkers
  - Chess
  - Go
- Resource limits
- α-β pruning
- Games with chance
  - Backgammon

Types of Games

	Deterministic	Chance
Perfect Information	Chess, Checkers Go, Othello	Backgammon, Monopoly
Imperfect Information	Battleship	Bridge, Poker, Scrabble, Nuclear war

Games vs. search problems

Similar to state-space search but:

1. There's an *Opponent* whose moves must be considered

2. Because of the combinatorial explosion, we can't search the entire tree:

   Have to *commit* to a move based on an *estimate* of its goodness.

   Heuristic mechanisms we used to decide which node to open next will be used to decide which MOVE to make next

Games vs. search problems

We will follow some of the history:

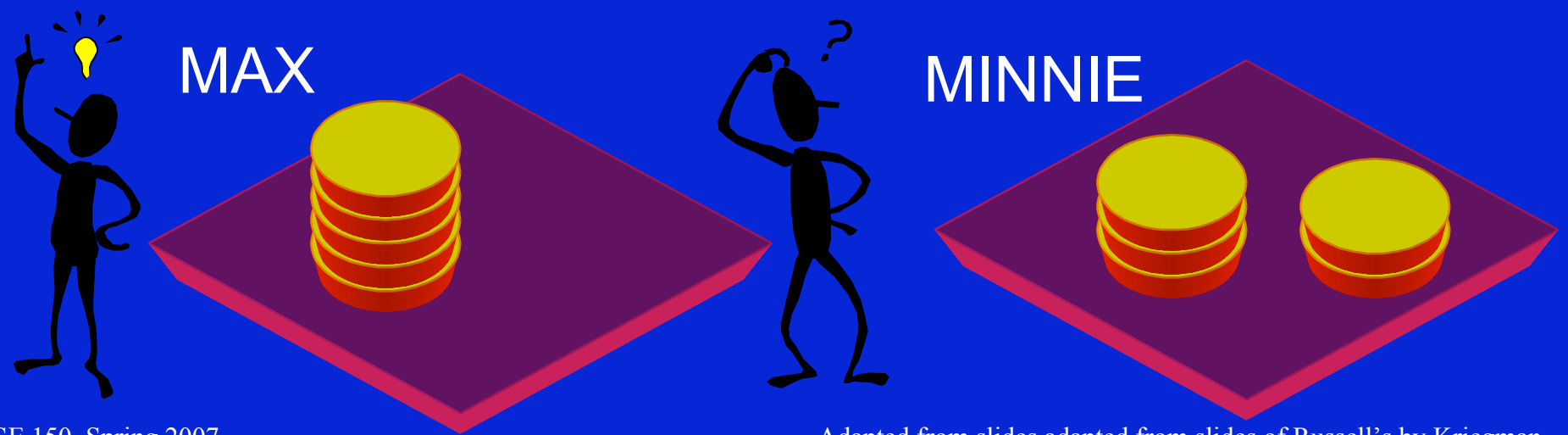
- algorithm for perfect play [Von Neumann, 1944]
- finite horizon, approximate evaluation [Zuse, 1945; Shannon, 1950; Samuel, 1952-57]
- pruning to reduce costs [McCarthy, 1956]

Deterministic Two-person Games

- Two players move in turn
- Each Player knows what the other has done and can do
- Either one of the players wins (and the other loses) or there is a draw.

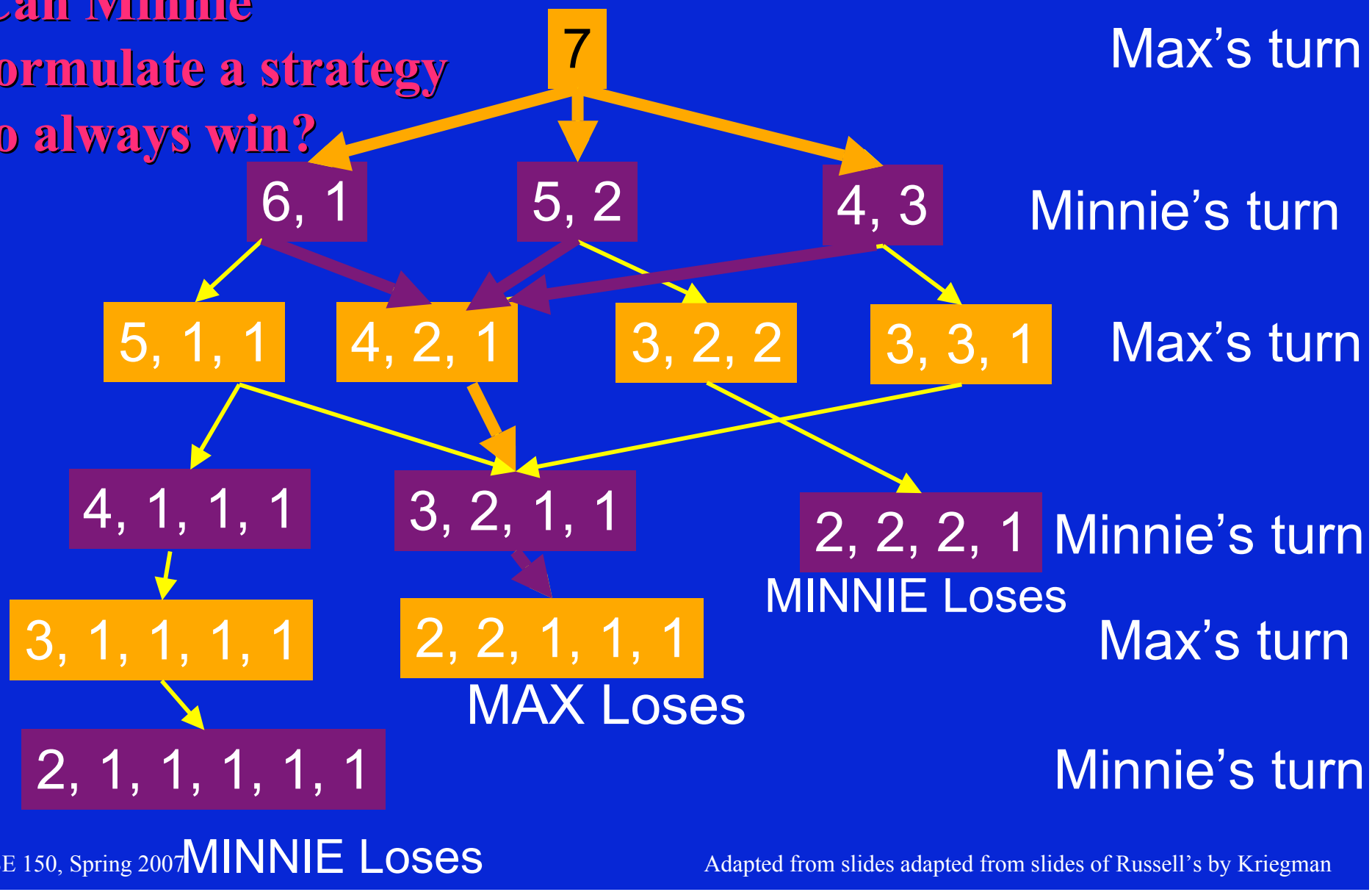
Games as Search: Grundy's Game
[Nilsson, Chapter 3]

- Initially a stack of pennies stands between two players
- Each player divides one of the current stacks into two unequal stacks.
- The game ends when every stack contains one or two pennies
- The first player who cannot play loses



Grundy's Game: States

Can Minnie formulate a strategy to always win?



Adapted from slides adapted from slides of Russell's by Kriegman

When is there a winning strategy for Max?

- When it is MIN's turn to play, a win must be obtainable for MAX from **all** the results of the move (sometimes called an AND node)

- When it is MAX's turn, a win must be obtainable from **at least one** of the results of the move (sometimes called an OR node).

Minimax

- **Initial state**: Board position and whose turn it is.
- **Operators:** Legal moves that the player can make
- **Terminal test**: A test of the state to determine if game is over; set of states satisfying terminal test are **terminal states.**
- **Utility function:** numeric value for outcome of game (leaf nodes).
  - E.g. +1 win, -1 loss, 0 draw.
  - E.g. # of points scored if tallying points as in backgammon.

- Assumption: Opponent will always chose operator (move) maximize own utility.

Minimax

- **Initial state**: Board position and whose turn it is.
- **Operators:** Legal moves that the player can make
- **Terminal test**: A test of the state to determine if game is over; set of states satisfying terminal test are **terminal states.**
- **Utility function:** numeric value for outcome of game (leaf nodes).
  - E.g. +1 win, -1 loss, 0 draw.
  - E.g. # of points scored if tallying points as in backgammon.

- Assumption: Opponent will always chose operator (move) maximize own utility.

Minimax

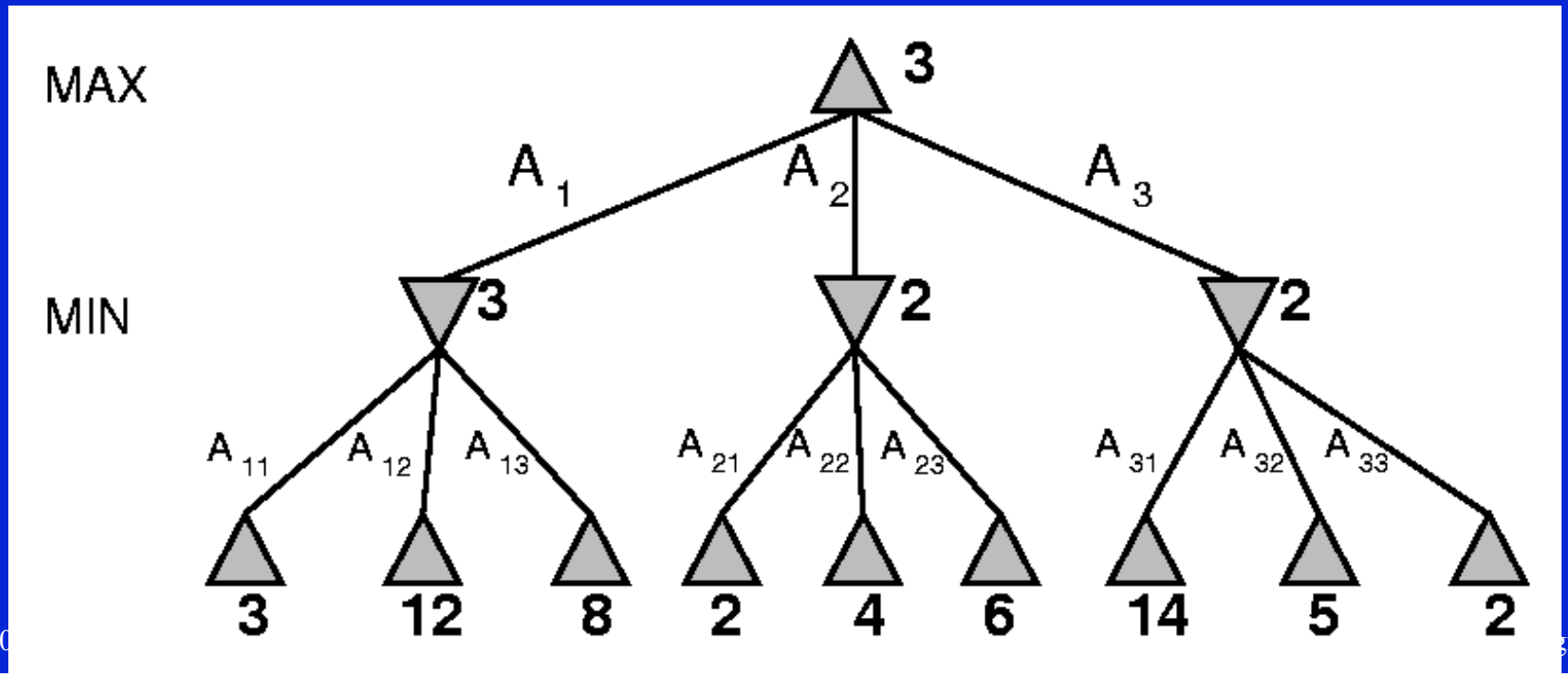
Perfect play for deterministic, perfect-information games

Max's strategy: choose action with highest minimax value

→best achievable payoff against best play by min.

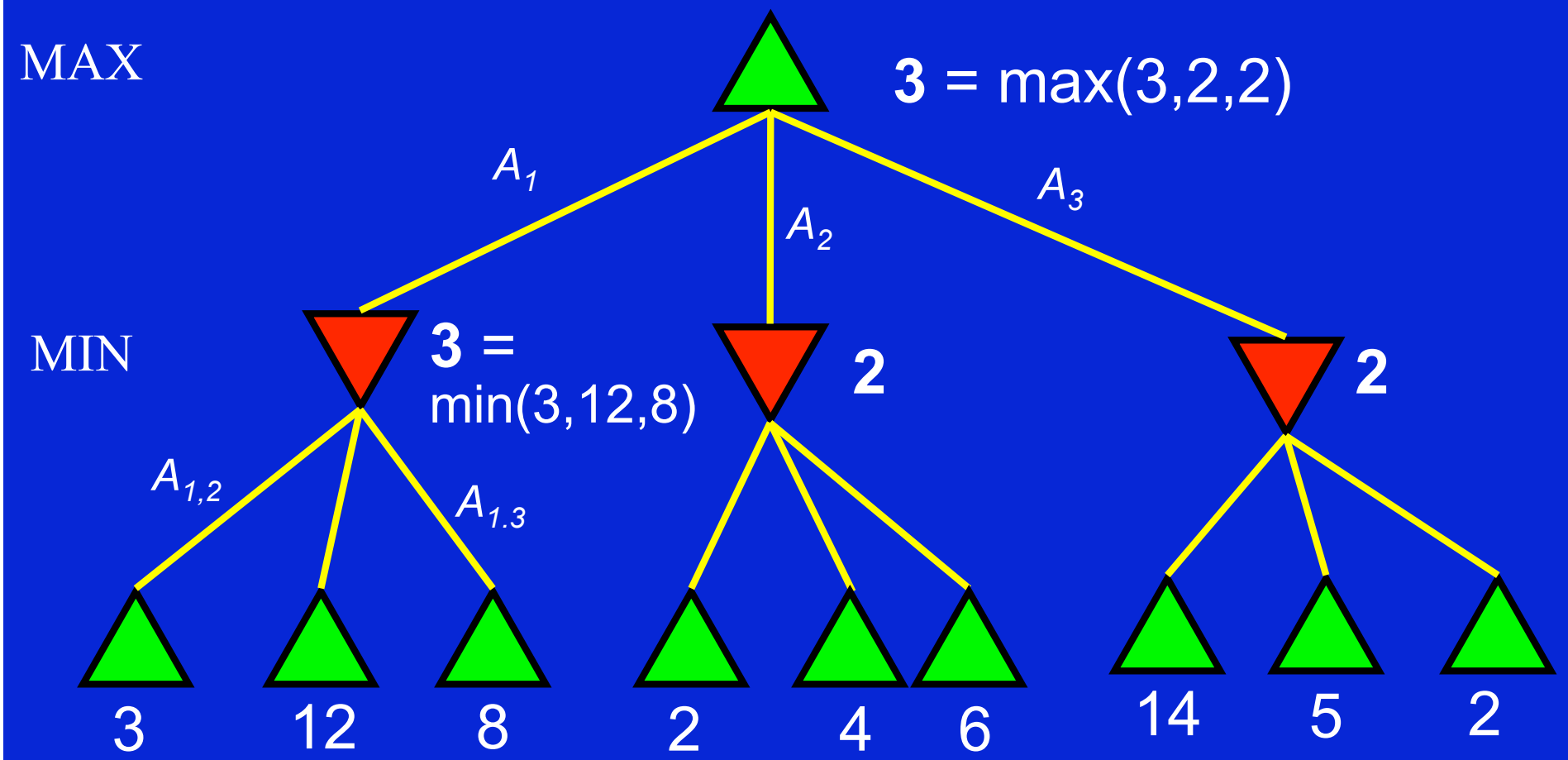
Consider a 2-ply (two step) game:

Max wants largest outcome --- Minnie wants smallest



Minimax Example

Game tree for a fictitious 2-ply game



Leaf nodes: value of utility function

Adapted from slides adapted from slides of Russell's by Kriegman

Minimax Algoritm

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: Yes, if tree is finite
- Optimal: Yes, against an optimal opponent. Otherwise??
- Time complexity: $O(b^m)$
- Space complexity: $O(bm)$

Note: For chess, $b \approx 35$, $m \approx 100$ for a "reasonable game."

→ Solution is completely infeasible

Actually only board 10^{40} board positions, not 35^{100}: Checking for previously visited states becomes important! ("transposition table")

Resource limits

Suppose we have 100 seconds, explore 10^4 nodes/second

→ 10^6 nodes per move

Standard approach:

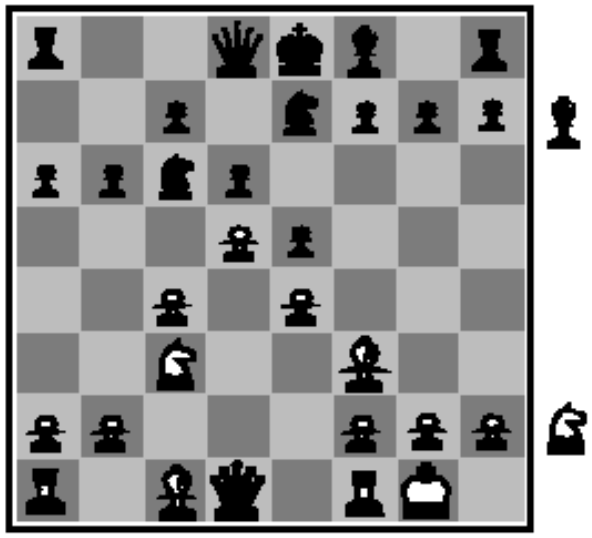
- *cutoff test*

  e.g., depth limit

- *evaluation function* = estimated desirability of position

  (an approximation to the utility used in minimax).

Evaluation Function



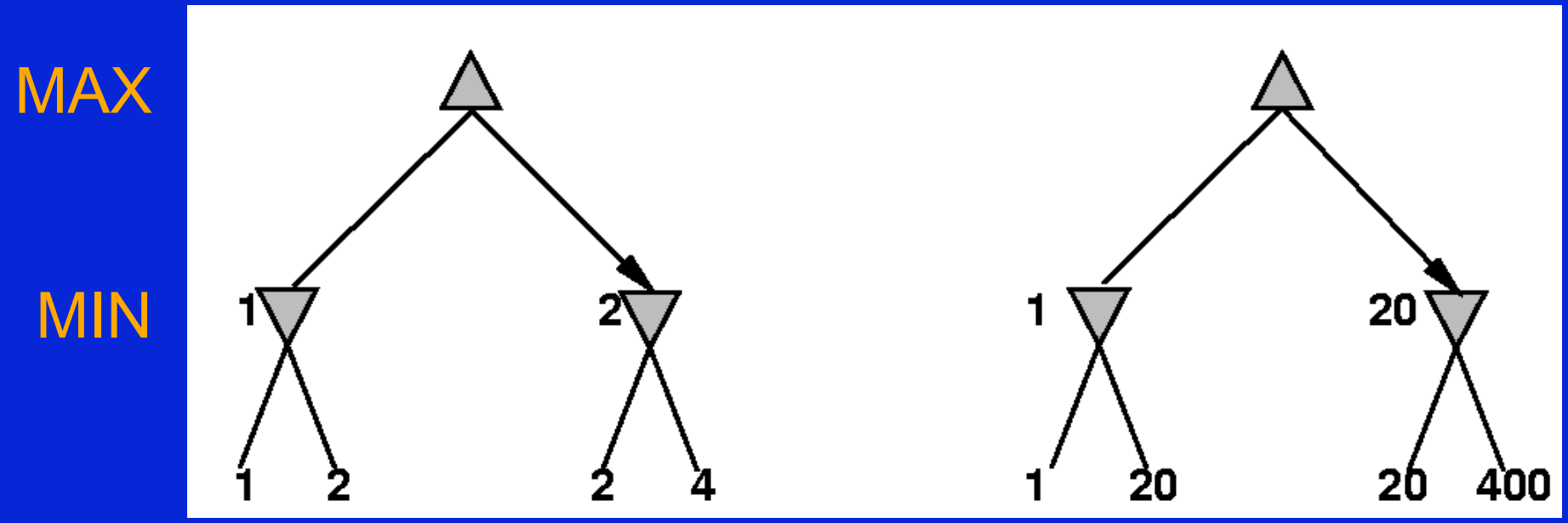
For chess, typically *linear* weighted sum of features

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

e.g., $w_1 = 9$ with
$f_1(s)$ = (number of white queens) - (number of black queens)

$w_2 = 5$ with
$f_2(s)$ = (number of white rooks) - (number of black rooks)

Digression: Exact Values Don't Matter



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

Cutting off search

- **Replace MinimaxValue** with **MinimaxCutoff .**
- **MinimaxCutoff** is identical to **MinimaxValue** except
  1. **Terminal?** is replaced by **Cutoff?**
  2. **Utility** is replaced by **Eval**

Does it work in practice?

$b^m = 10^6$, $b=35 \rightarrow m=4$

No: 4-ply lookahead is a hopeless chess player!

4-ply human novice

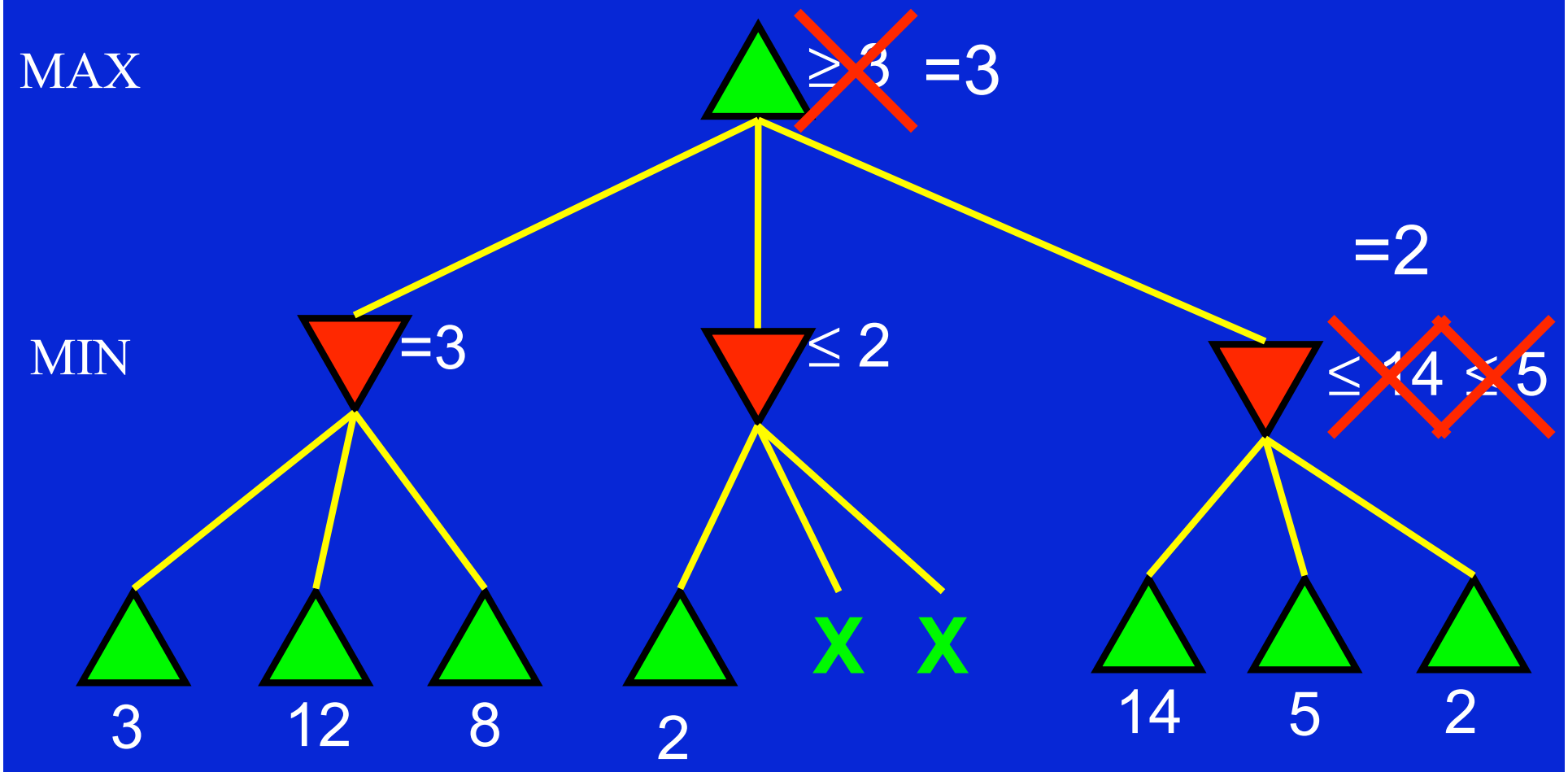
8-ply typical PC, human master

12-ply Deep Blue, Kasparov

Another idea

- α–β Pruning
- Essential idea is to stop searching down a branch of tree when you can determine that it's a dead end.

α–β Pruning Example



Adapted from slides adapted from slides of Russell's by Kriegman

The α–β Algorithm

Basically MINIMAX and keep track of α, β and prune

```
function MAX-VALUE(state, game, α, β) returns the minimax value of state
   inputs: state, current state in game
           game, game description
           α, the best score for MAX along the path to state
           β, 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
       α ← MAX(α, MIN-VALUE(s, game, α, β))
       if α ≥ β then return β
   end
   return α
```

```
function MIN-VALUE(state, game, α, β) returns the minimax value of state

   if CUTOFF-TEST(state) then return EVAL(state)
   for each s in SUCCESSORS(state) do
       β ← MIN(β, MAX-VALUE(s, game, α, β))
       if β ≤ α then return α
   end
   return β
```

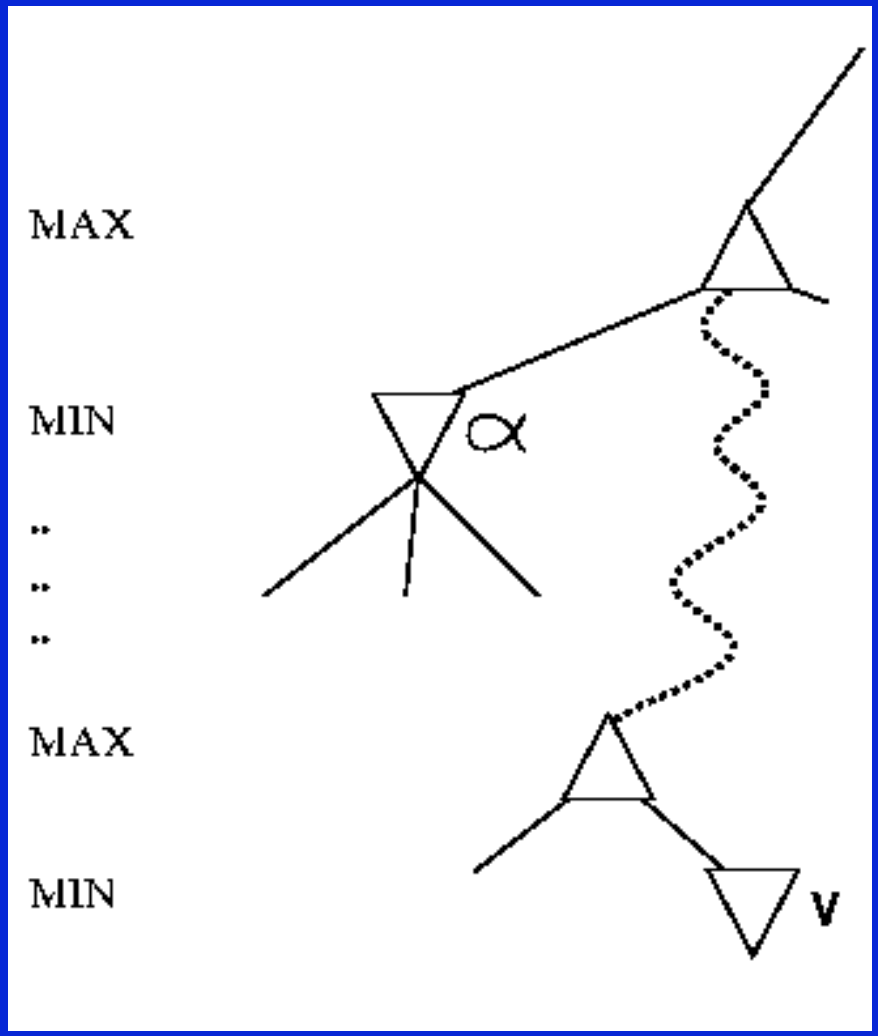
How is the search tree pruned?

- α is the best value (to MAX) found so far.
- If $V \leq \alpha$, then the subtree containing V will have utility no greater than V

  → **Prune that branch**

- β can be defined similarly from MIN's perspective

• Note: The book's figure 6.5 makes little sense to me w.r.t. the algorithm in 6.7



Properties of α–β

Pruning **does not** affect final result.

Good move ordering improves effectiveness of pruning.

With ``perfect ordering,'' time complexity = $O(b^{m/2})$
- doubles depth of search
- can easily reach depth 8 and play good chess

A simple example of the value of reasoning about which computations are relevant (a form of *metareasoning*).

Deeper searches

The ***horizon*** problem:

Sometimes distaster (or Shangri-la) is just beyond the depth bound.

One fix is ***quiesence*** search:
If a piece is about to be taken, the board is not "quiet" - keep searching beyond that point. Deep blue used this.

Another (new) and useful heuristic:

the ***null move*** heuristic: Let the opponent take ***two*** moves and then search under the node.

Stored knowledge

- There is a big fan-out at the beginning of the game:
  - Deep blue stored thousands of opening moves
  - And thousands of *solved* end games (all games with five pieces or less)
  - And 700,000 grandmaster games
  - Nowadays, a PC with good heuristics is as good as the best player.

Deterministic games in practice

Checkers: In 1994 ***Chinook*** ended 40-year-reign of human world champion Marion Tinsley. Used 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.

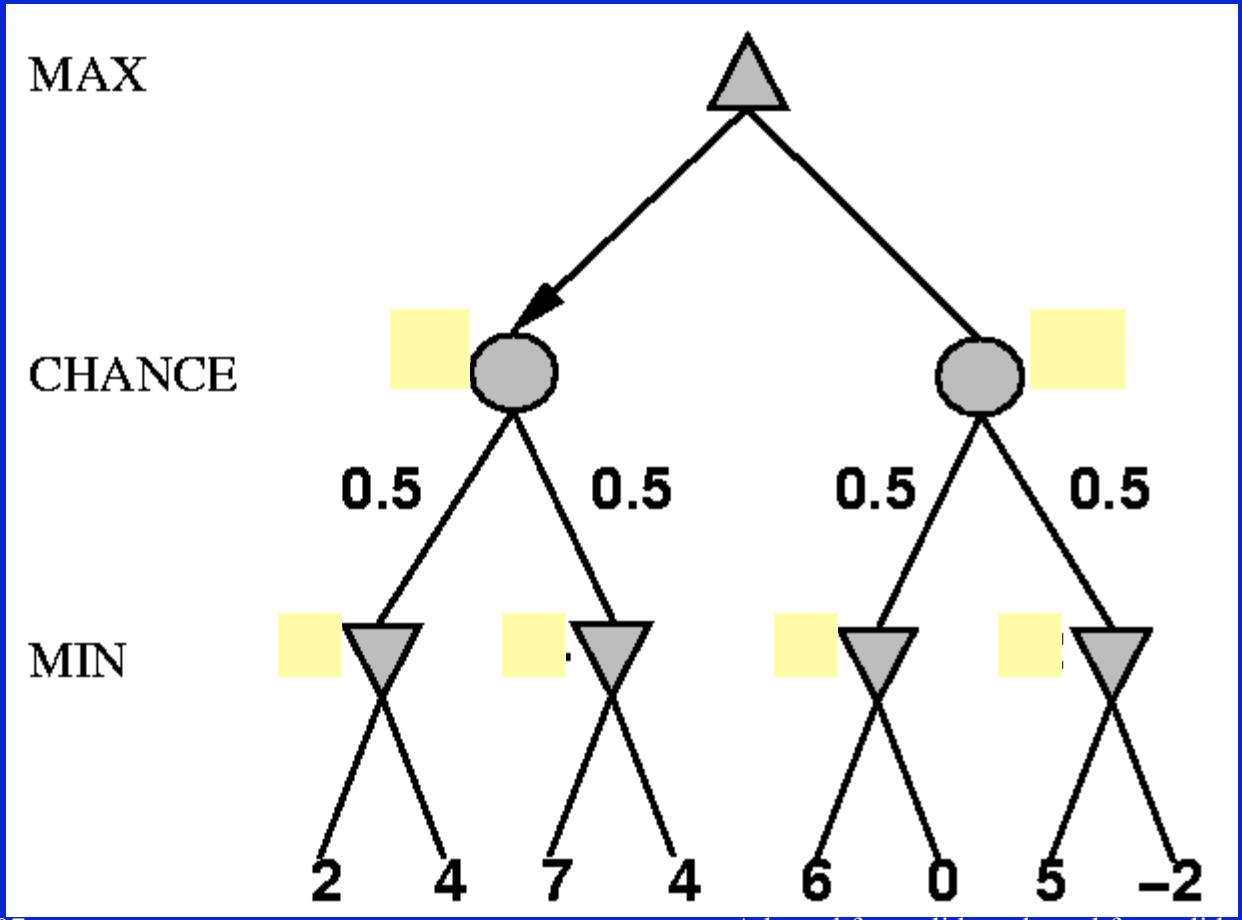
Chess: ***Deep Blue*** defeated human world champion Gary Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.

Othello: Human champions refuse to compete against computers, who are too good.

Go: Human champions refuse to compete against computers, who are too bad. In Go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.

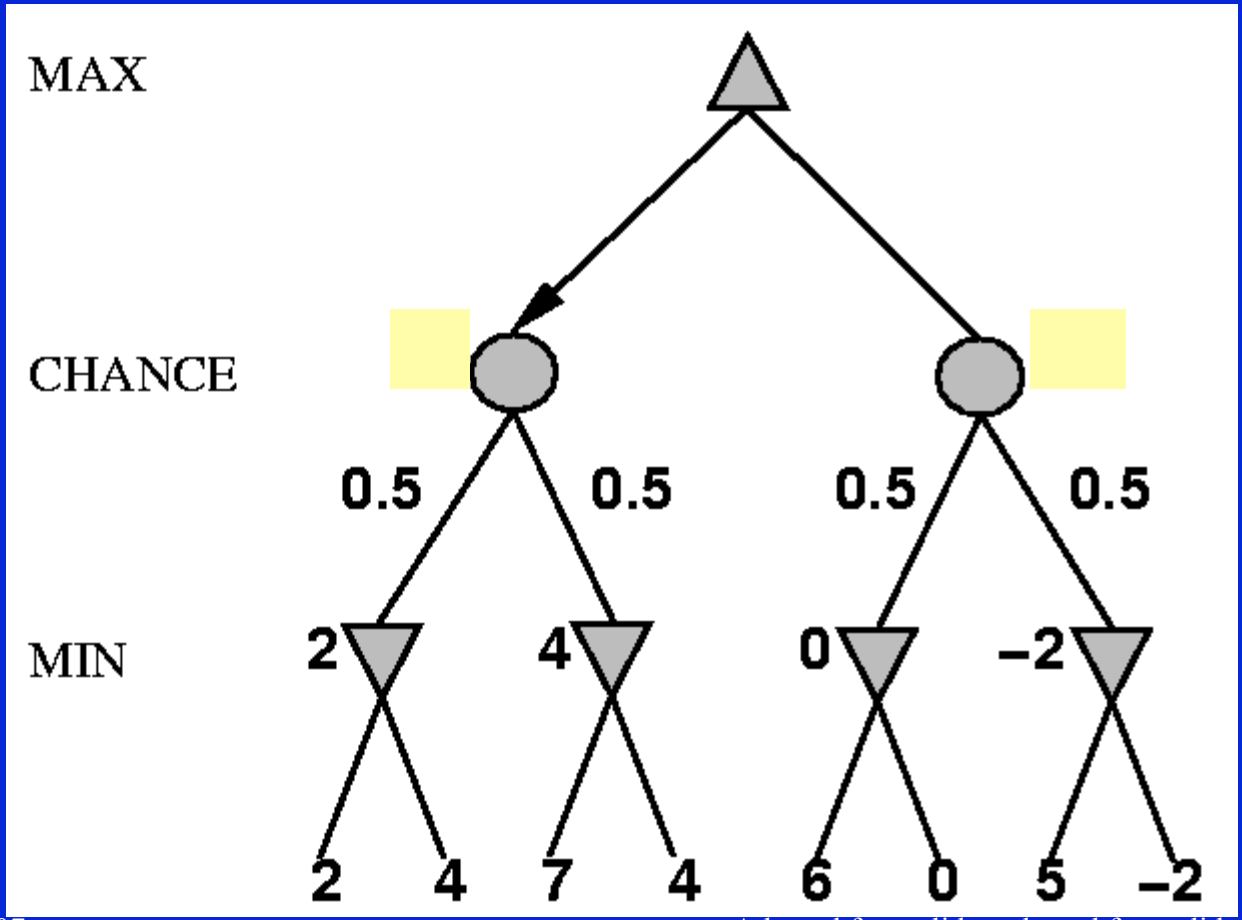
Nondeterministic games

- E.g., in backgammon, the dice rolls determine the legal moves
- Consider simpler example with coin-flipping instead of dice-rolling:



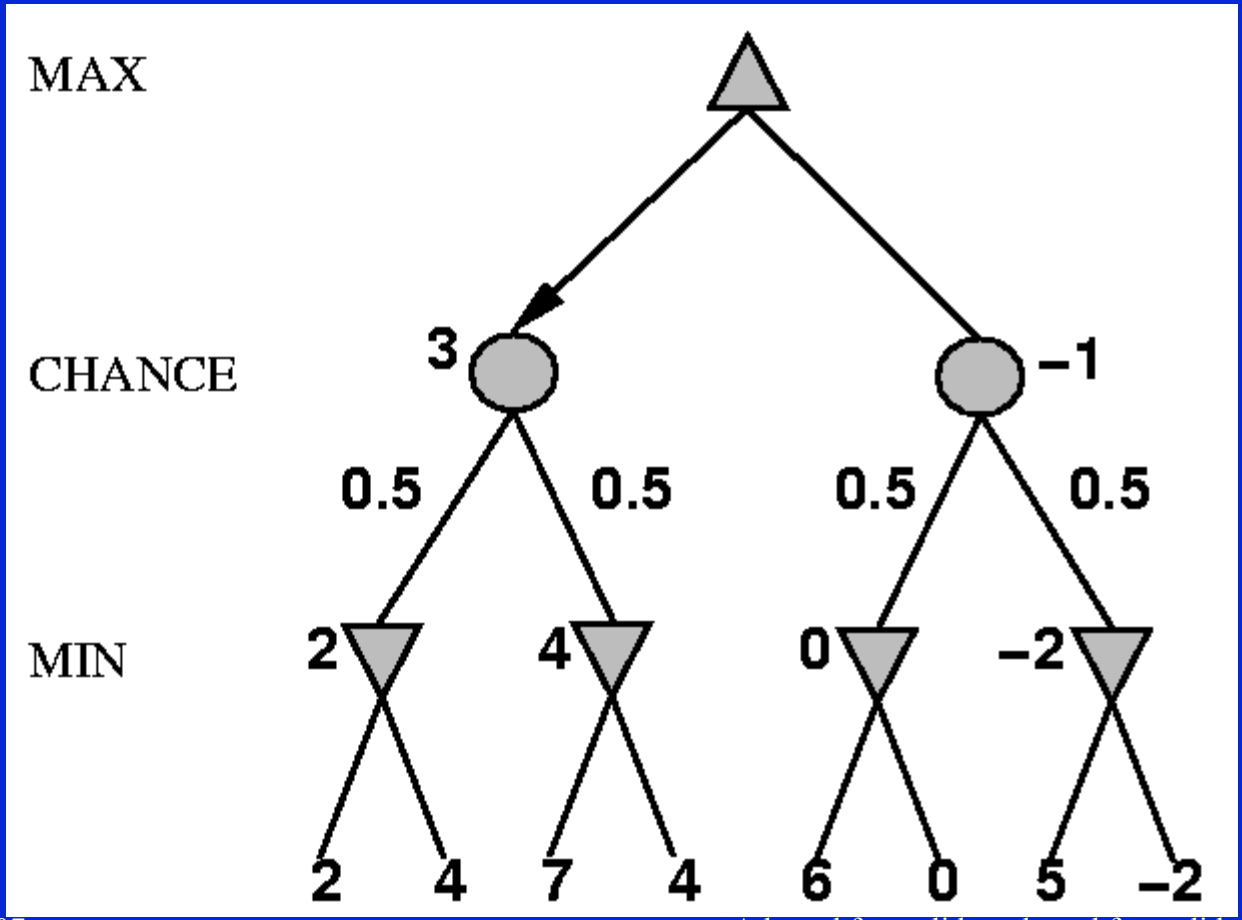
Nondeterministic games

1. For mind node, compute min of children



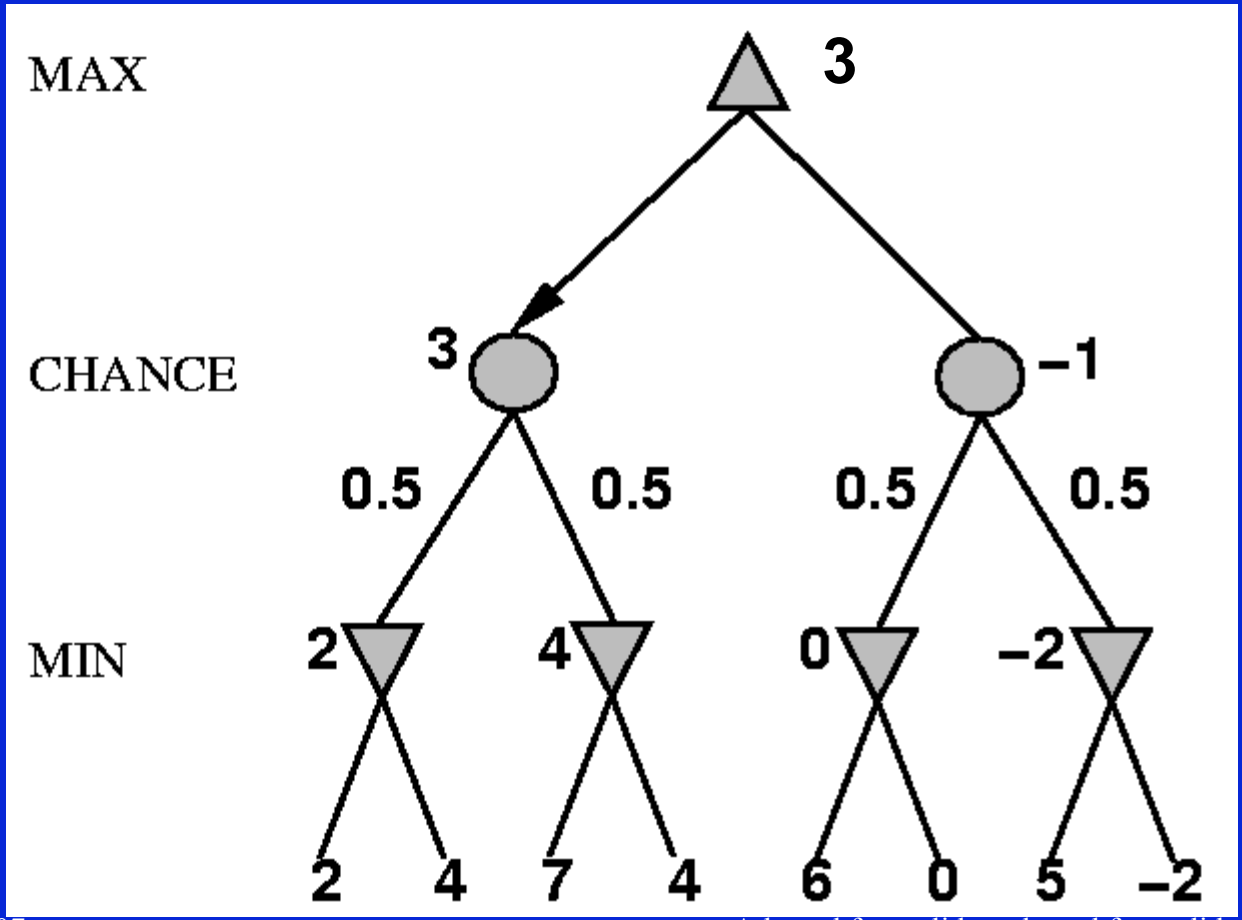
Nondeterministic games

1. For mind node, compute min of children
2. For chance node, compute weighted average of children



Nondeterministic games

1. For mind node, compute min of children
2. For chance node, compute weighted average of children
3. For max node, compute max of children



Algorithm for Nondeterministic Games

Expectiminimax maximize expected value (average) of utility against a perfect opponent.

Pseudocode is just like **Minimax**, except we must also handle chance nodes:

```
...
if state is a chance node then
  return average of
                ExpectiMinimax-Value of Successors(state)
...
```

A version of α–β pruning is possible but only if the leaf values are bounded.

```
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]
```

```
Function EXPECTIMINIMAX-VALUE (state, game) returns a utility value
If TERMINAL-TEST[game](state) then
        return  UTILITY[game](state)
else if state is a chance node then
   return average of  EXPECTIMINMAX-VALUE of Successors(state)
else if MAX is to move in state then
        return the highest EXPECTIMINIMAX-VALUE of Successors[state]
else return the lowest EXPECTIMINIMAX-VALUE of Successors[state]
```

Nondeterministic Games in Practice

- Dice rolls increase branching factor *b:*
  - 21 possible rolls with 2 dice
  - Backgammon 20 legal moves

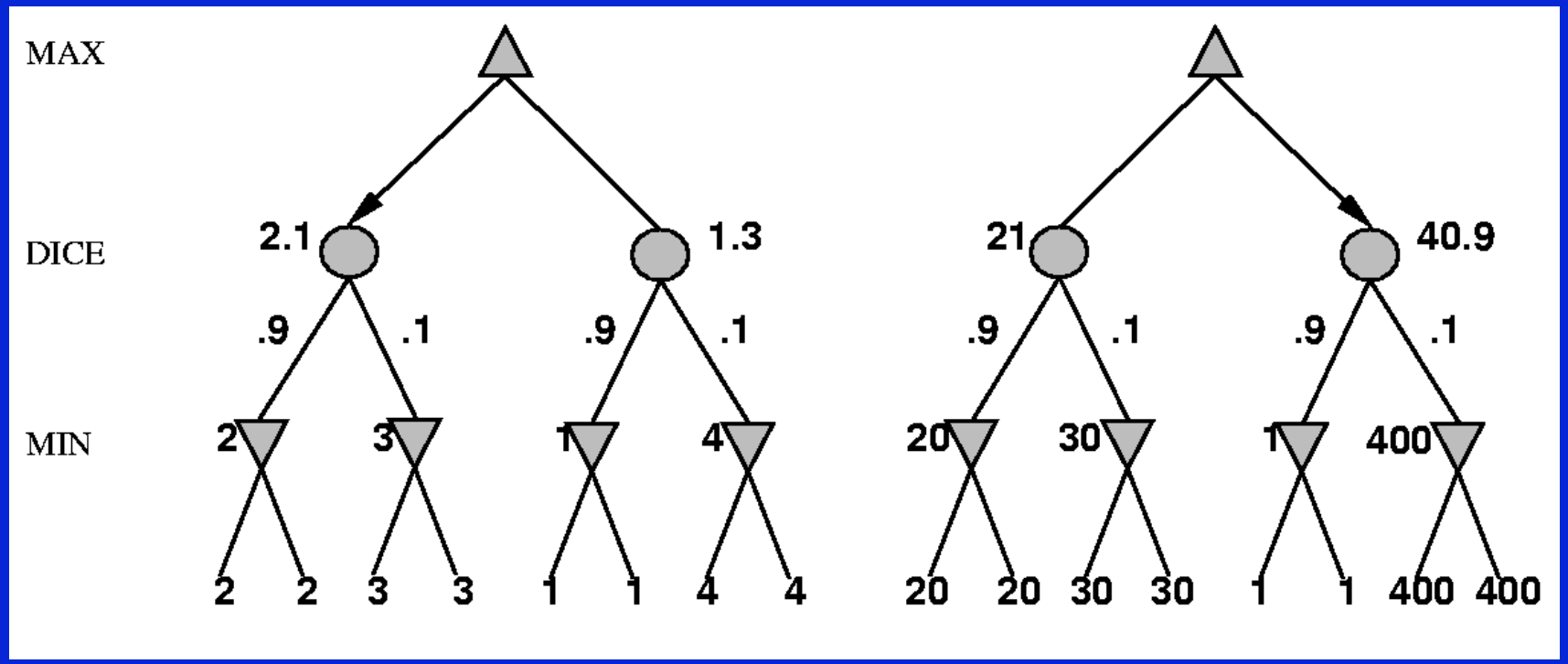
- depth 4 → $20 * (21 * 20)^3 \approx 1.2 \text{ X } 10^9$

- As depth increases, probability of reaching a given node shrinks
  → value of look ahead is diminished

- α–β pruning is much less effective

- **TDGammon** uses depth-2 search + very good Eval
  Plays at world-champion level

Digression: Exact Values DO Matter



- Behavior is preserved only by *positive linear* transformation of **Eval.**
- Hence **Eval** should be proportional to the expected payoff.

Games Summary

- Games are fun to work on! (and dangerous)
- They illustrate several important points about AI
  - perfection is unattainable → must approximate
  - good idea to think about what to think about
  - uncertainty constrains the assignment of values to states
- "Games are to AI as grand prix racing is to automobile design"