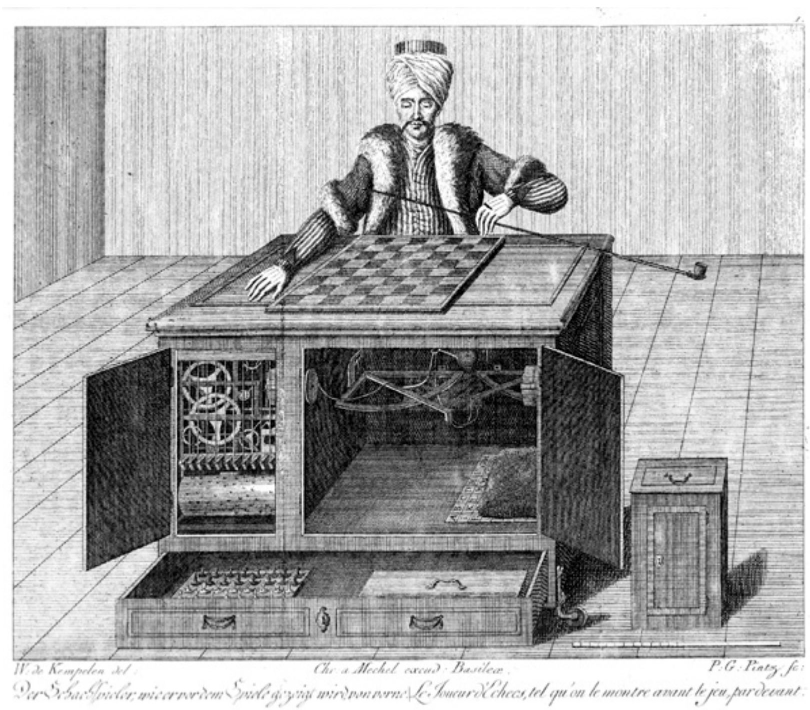
Game-playing AIs: Games and Adversarial Search

AIMA 5.1-5.5,

AIMA 16.1-16.3



Penn
University of Pennsylvania

Games: Outline of Unit

Part I: Games as Search

- **Motivation**
- **Game-playing AI successes**
- **Game Trees**
- **Evaluation Functions**

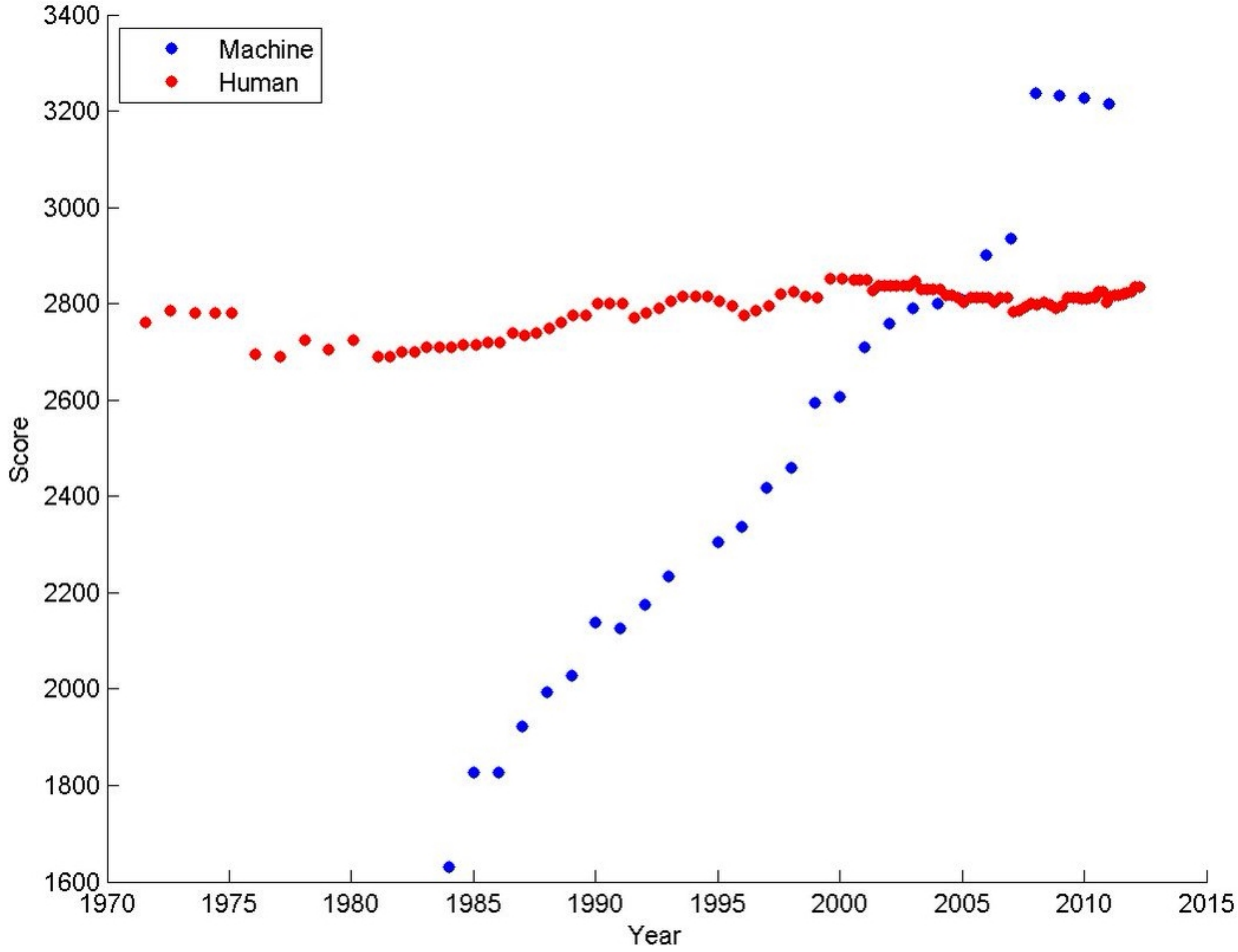
Part II: Adversarial Search

- **The Minimax Rule**
- **Alpha-Beta Pruning**

May 11, 1997


may 11th
game 6 : may 11 @ 3:00PM EDT | 19:00 GMT
kasparov 2.5 deep blue 2.5
Home
The match
The players
The technology
Community
Deep Blue Wins
3.5 to 2.5
KASPAROV VS DEEP BLUE
the rematch
With a dramatic victory in Game 6, Deep Blue won its six-game rematch with Champion Garry Kasparov
OVERVIEW
EVENT COVERAGE
MATCH NEWS
MAIN STORIES
Commentary
George Plimpton on chess, Kasparov, and the limitations of computers
Read the article
Commentary
Vishwanathan Anand on the legacy of Kasparov vs. Deep Blue
Read the article
Club Kasparov
Visit the virtual home of the world's greatest chess player.
KASPAROV
Guest essays
Thoughts on chess, computers, and what it all means
Read the essays...
Community
During the rematch, more than 20,000 people from 120 countries joined the community to talk about the match.
Clips from the rematch
Video footage from the games
Highlights from the games

Ratings of human & computer chess champions



https://srconstantin.wordpress.com/2017/01/28/performance-trends-in-ai/

Artificial intelligence (AI)

AlphaGo seals 4-1 victory over Go grandmaster Lee Sedol

DeepMind's artificial intelligence astonishes fans to defeat human opponent and offers evidence computer software has mastered a major challenge

Steven Borowiec

Tuesday 15 March 2016 06.12 EDT

This article is 6 months old

Shares
613

Save for later

The world's top Go player, Lee Sedol, lost the final game of the Google DeepMind challenge match. Photograph: Yonhap/Reuters

Google DeepMind's AlphaGo program triumphed in its final game against South Korean Go grandmaster Lee Sedol to win the series 4-1, providing further evidence of the landmark achievement for an artificial intelligence program.

Lee started Tuesday's game strongly, taking advantage of an early mistake by AlphaGo. But in the end, Lee was unable to hold off a comeback by his opponent, which won a narrow victory.

(from The Guardian)

The Simplest Game Environment

- ***Multiagent***
- ***Static:*** **No change while an agent is deliberating**
- ***Discrete:*** **A finite set of percepts and actions**
- ***Fully observable*** **: An agent's sensors give it the complete state of the environment.**
- ***Strategic:*** **The next state is determined by the current state and the action executed by the agent and the actions of one other agent.**

Key properties of our sample games

1. **Two players alternate moves**
2. **Zero-sum: one player's loss is another's gain**
3. **Clear set of legal moves**
4. **Well-defined outcomes (e.g. win, lose, draw)**

- **Examples:**
  - Chess, Checkers, Go,
  - Mancala, Tic-Tac-Toe, Othello …

More complicated games

- **Most card games (e.g. Hearts, Bridge, etc.) and Scrabble**
  - **Stochastic, not deterministic**
  - **Not fully observable: lacking in perfect information**
- **Real-time strategy games, e.g. Warcraft**
  - **Don’t take turns**
  - **No pause between actions**
- **Cooperative games**

Pac-Man

https://youtu.be/-CbyAk3Sn9I

Formalizing the Game setup

1. **Two players: *MAX* and *MIN; MAX* moves first.**
2. ***MAX* and *MIN* take turns until the game is over.**
3. **Winner gets award, loser gets penalty.**

- **Games as *search*:**
  - *Initial state*: e.g. board configuration of chess
  - *Successor function*: list of (move,state) pairs specifying legal moves.
  - *Terminal test*: Is the game finished?
  - *Utility function*: Gives numerical value of terminal states. e.g. win (+∞), lose (-∞) and draw (0)
  - *MAX* uses search tree to determine next move.

How to Play a Game by Searching

- **General Scheme**
  1. **Consider all legal successors to the current state ('board position')**
  2. **Evaluate each successor board position**
  3. **Pick the move which leads to the best board position.**
  4. **After your opponent moves,** repeat.

- **Design issues**
  1. Representing the 'board'
  2. Representing legal next boards
  3. Evaluating positions
  4. Looking ahead

Hexapawn: A very simple Game

- **Hexapawn is played on a 3x3 chessboard**

- **Only standard pawn moves:**
  1. A pawn moves forward one square onto an empty square
  2. A pawn "captures" an opponent pawn by moving diagonally forward one square, if that square contains an opposing pawn. The opposing pawn is removed from the board.

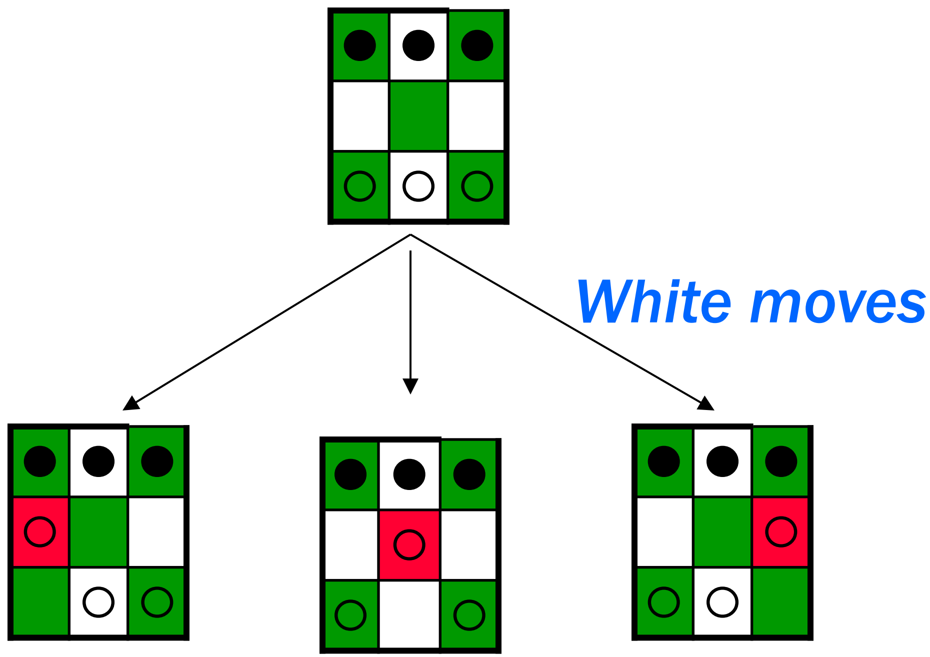
Hexapawn: A very simple Game

- **Hexapawn is played on a 3x3 chessboard**

- **Player P_1 wins the game against P_2 when:**
  - One of P_1's pawns reaches the far side of the board, or
  - P_2 cannot move because no legal move is possible.
  - P_2 has no pawns left.

(Invented by Martin Gardner in 1962, with learning "program" using match boxes. Reprinted in "The Unexpected Hanging..)

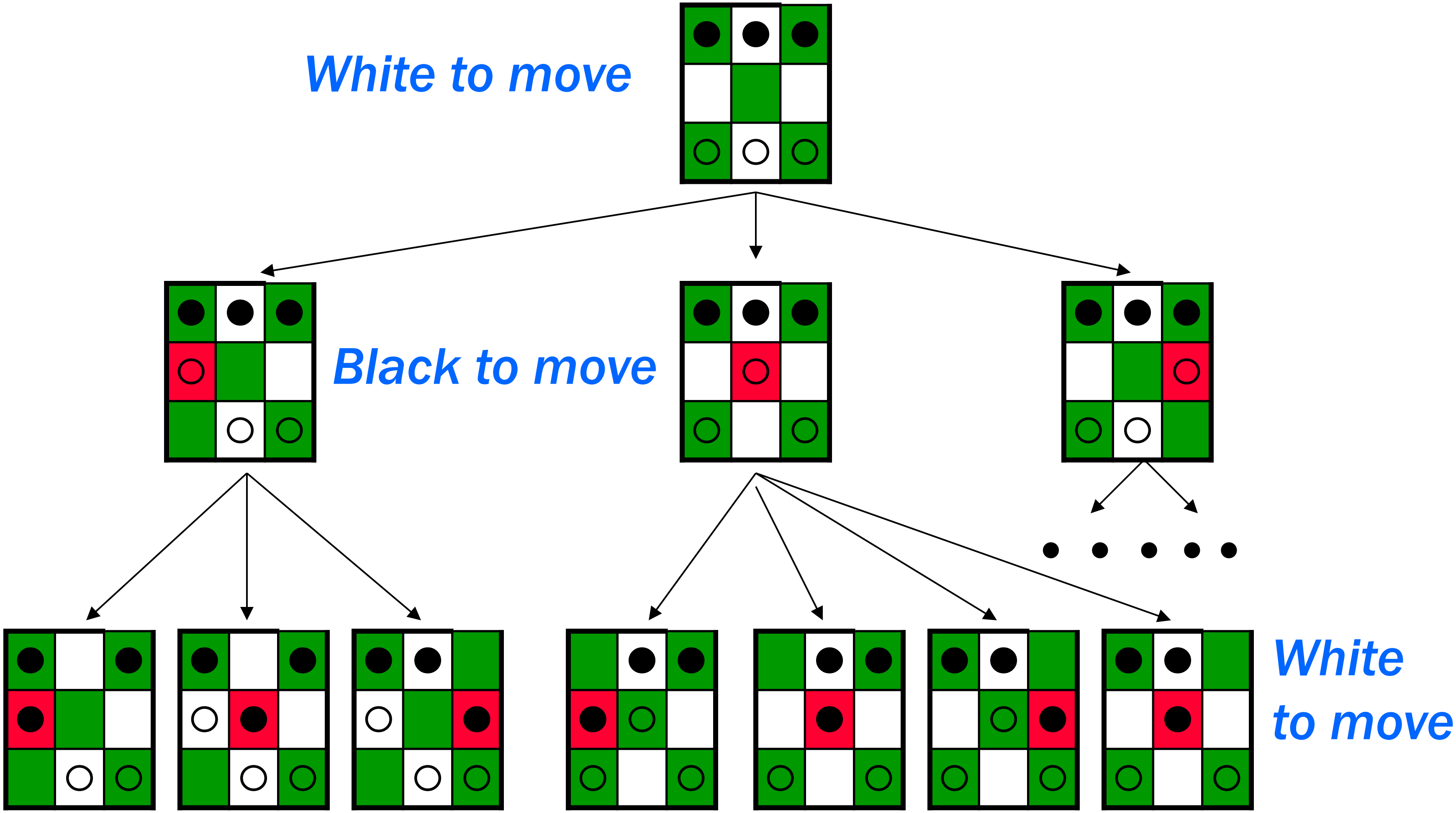
Hexapawn: Three Possible First Moves



Game Trees

- **Represent the game problem space by a tree:**
  - Nodes represent ‘board positions’; edges represent legal moves.
  - Root node is the first position in which a decision must be made.

Hexapawn: Simplified Game Tree for 2 Moves



Adversarial Search

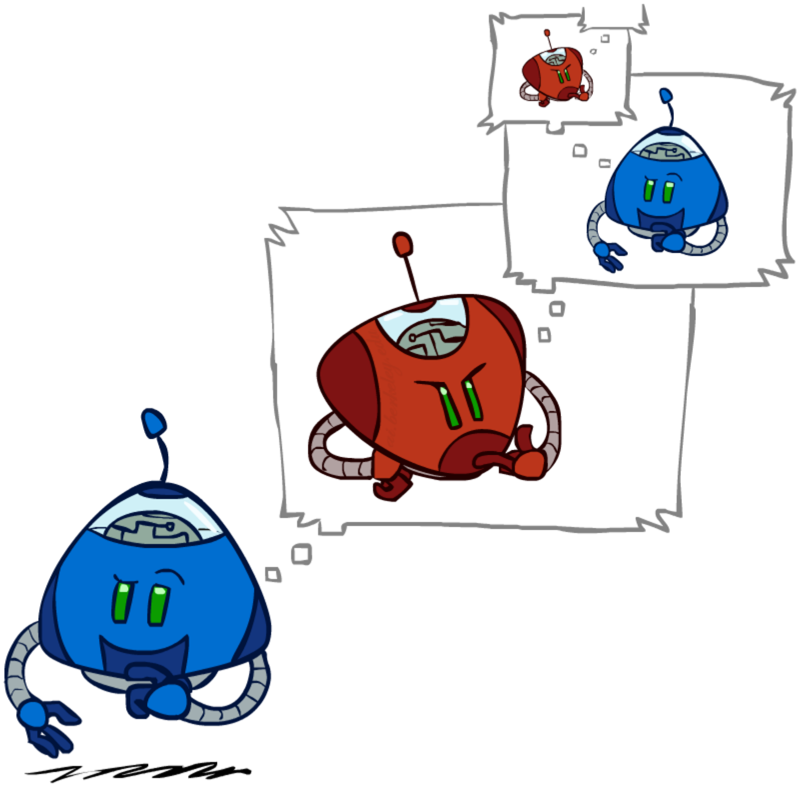


Image from Dan Klein

Battle of Wits

https://www.youtube.com/watch?v=rMz7JBRbmNo

MAX & MIN Nodes : An egocentric view

- **Two players: MAX, MAX's opponent MIN**
- ***All play is computed from MAX's vantage point.***
- **When MAX moves, MAX attempts to MAXimize MAX's outcome.**
- **When MAX's opponent moves, they attempt to MINimize MAX's outcome.**

WE TYPICALLY ASSUME MAX MOVES FIRST:

- **Label the root (level 0) MAX**
- **Alternate MAX/MIN labels at each successive tree level *(ply).***
- ***Even levels* represent turns for MAX**
- ***Odd levels* represent turns for MIN**

Game Trees

- Represent the game problem space by a tree:
  - Nodes represent ‘board positions’; edges represent legal moves.
  - Root node is the first position in which a decision must be made.

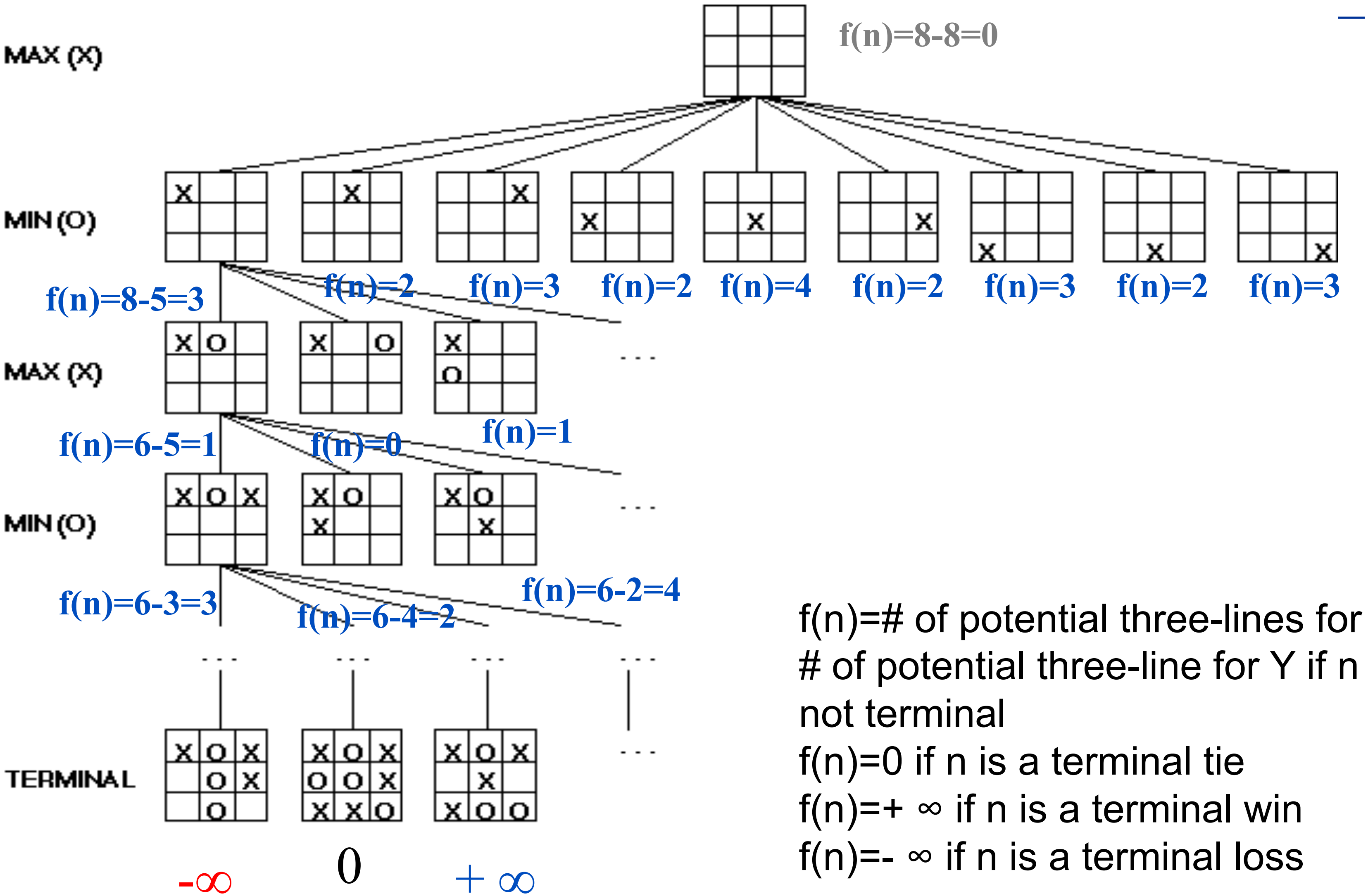
- **Evaluation function *f* assigns real-number scores to `board positions’ *without reference to path***
- **Terminal nodes represent ways the game could end, labeled with the desirability of that ending (e.g. win/lose/draw or a numerical score)**

Evaluation functions: *f(n)*

- **Evaluates how good a 'board position' is**
- **Based on *static features* of that board alone**
- **Zero-sum assumption lets us use one function to describe goodness for both players.**
  - ***f(n)>0*** if MAX is winning in position *n*
  - ***f(n)=0*** if position *n* is tied
  - ***f(n)<0*** if MIN is winning in position *n*
- **Build using expert knowledge,**
  - Tic-tac-toe***: f(n)=(# of 3 lengths open for MAX)- (# open for MIN)***

(AIMA 5.4.1)

A Partial Game Tree for Tic-Tac-Toe



f(n)=# of potential three-lines for X – # of potential three-line for Y if n is not terminal

f(n)=0 if n is a terminal tie

f(n)=+ ∞ if n is a terminal win

f(n)=- ∞ if n is a terminal loss

Chess Evaluation Functions

- **Alan Turing's**
  f(n)=(sum of A's piece values)
  -(sum of B's piece values)

Pawn	**1.0**
Knight	**3.0**
Bishop	**3.25**
Rook	**5.0**
Queen	**9.0**

Pieces values for a simple Turing-style evaluation function often taught to novice chess players

- **More complex: weighted sum of *positional* features:**

$$\sum w_i feature_i(n)$$

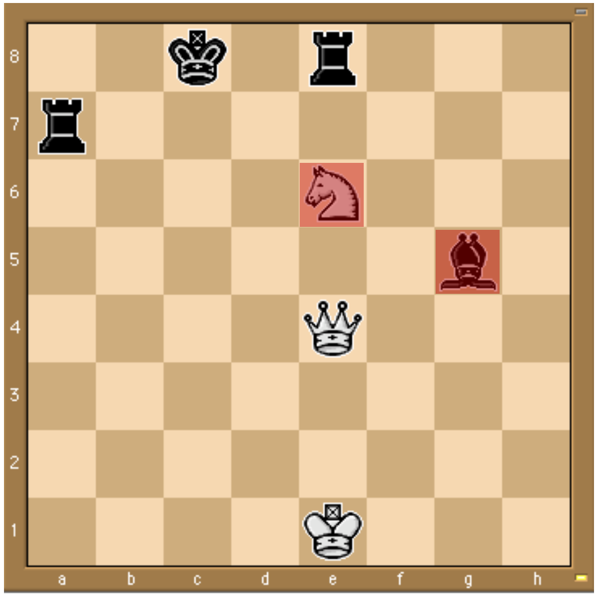
- **Deep Blue had > 8000 features**

Positive: rooks on open files, knights in closed positions, control of the center, developed pieces

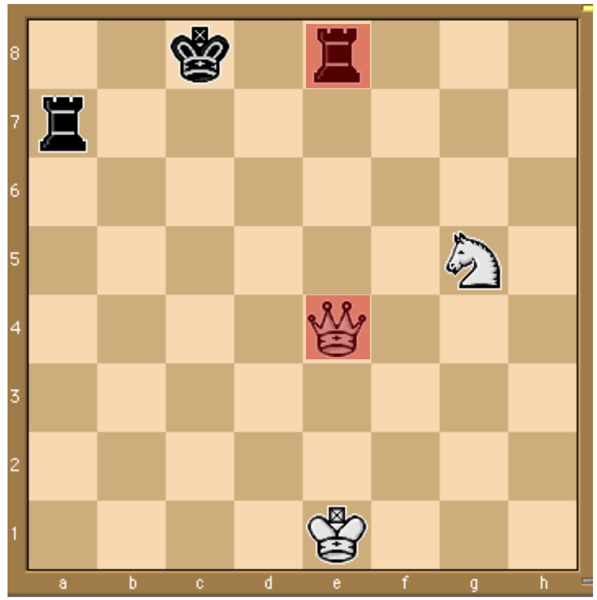
Negative: doubled pawns, wrong-colored bishops in closed positions, isolated pawns,pinned pieces

Examples of more complex features

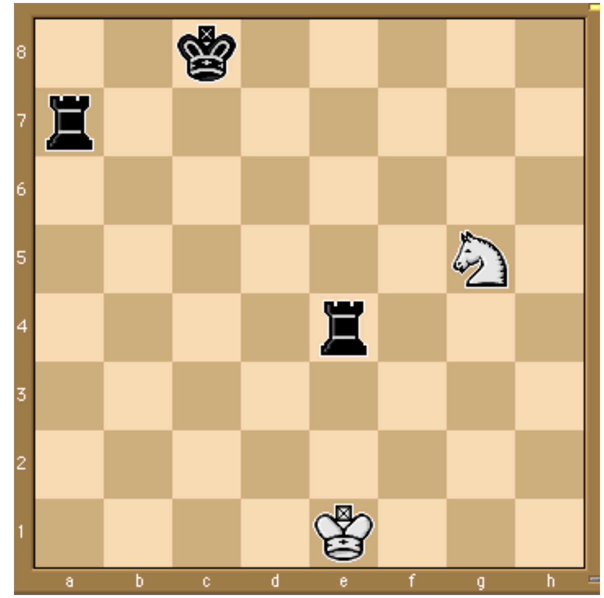
Some Chess Positions and their Evaluations



White to move
f(n)=(9+3)-(5+5+3.25)
=-1.25



… Nxg5??
f(n)=(9+3)-(5+5)
=2



Uh-oh: Rxg4+
f(n)=(3)-(5+5)
=-7

And black may force checkmate

So, considering our opponent's possible responses would be wise.

The Minimax Rule (AIMA 5.2)

The Minimax Rule: `Don't play hope chess'

Idea: Make the best move for MAX *assuming that MIN always replies with the best move for MIN*

Easily computed by a recursive process

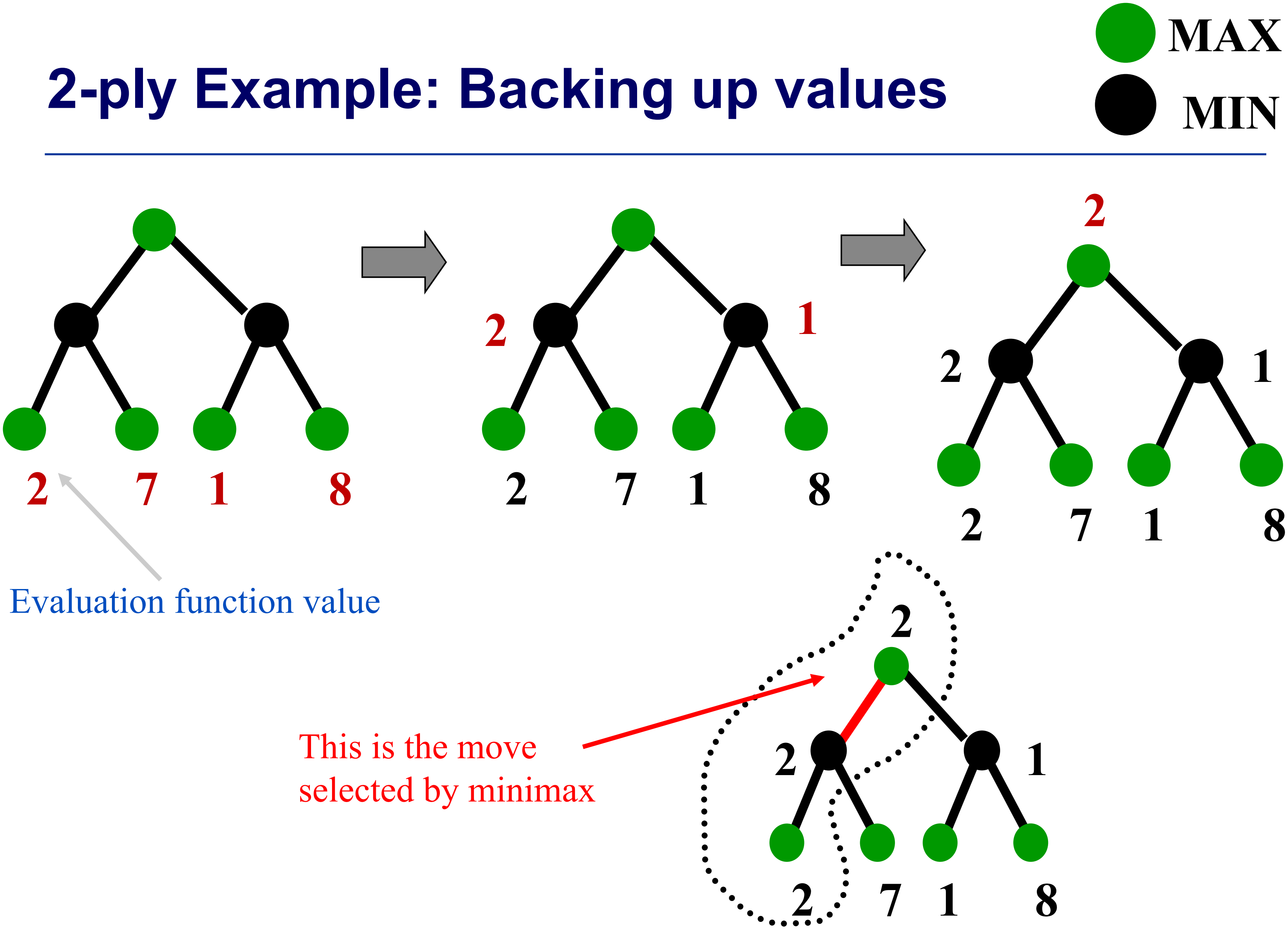
- The ***backed-up value*** of each node in the tree is determined by the values of its children:
  - For a **MAX** node, the backed-up value is the ***maximum*** of the values of its children *(i.e. the best for MAX)*
  - For a **MIN** node, the backed-up value is the ***minimum*** of the values of its children *(i.e. the best for MIN)*

The Minimax Procedure

Until game is over:

1. **Start with the current position as a MAX node.**
2. **Expand the game tree a fixed number of *ply*.**
3. **Apply the evaluation function to the leaf positions.**
4. **Calculate back-up values bottom-up.**
5. **Pick the move assigned to MAX at the root**
6. **Wait for MIN to respond**

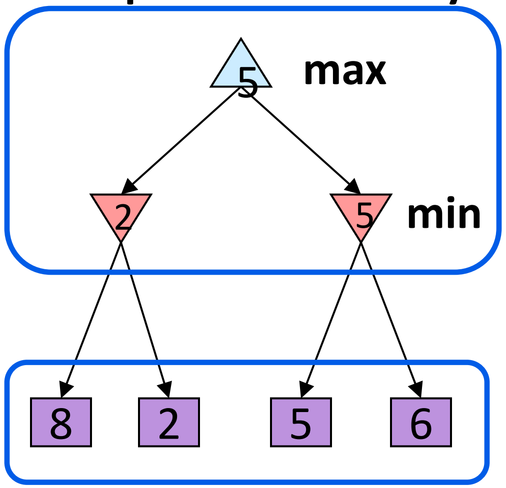
2-ply Example: Backing up values



Adversarial Search (Minimax)

- **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

Minimax Implementation

```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, min-value(successor))
    return v
```

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

```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, max-value(successor))
    return v
```

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

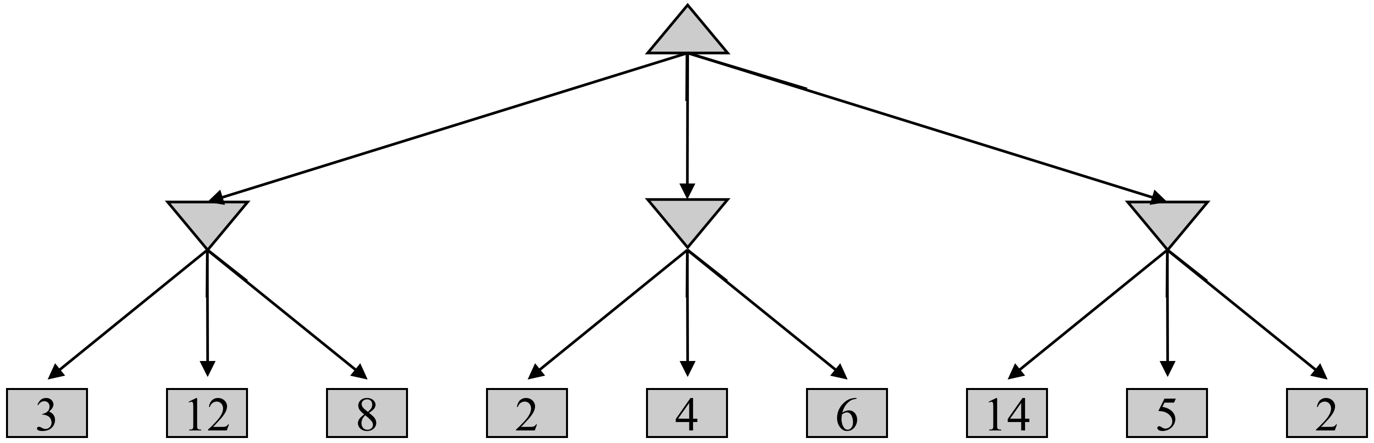
Minimax Implementation (Dispatch)

```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is MIN: return min-value(state)
```

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

```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor))
    return v
```

Minimax Example




```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, min-value(successor))
    return v
```

```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, max-value(successor))
    return v
```

Penn
UNIVERSITY of PENNSYLVANIA

What if MIN does not play optimally?

- **Definition of optimal play for MAX assumes MIN plays optimally:**
  - *Maximizes worst-case outcome* for MAX.
  - (Classic game theoretic strategy)
- **But if MIN does not play optimally, MAX will do even better. [Theorem-not hard to prove]**

Comments on Minimax Search

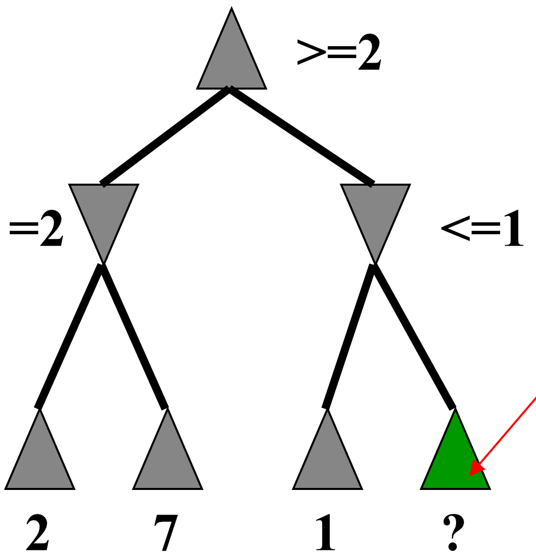
- **Depth-first search with fixed number of ply *m* as the limit.**
  - $O(b^m)$ time complexity – *As usual!*
  - $O(bm)$ space complexity
- **Performance will depend on**
  - the quality of the static evaluation function (expert knowledge)
  - depth of search (computing power and search algorithm)
- **Differences from normal state space search**
  - Looking to make *one* move only, despite deeper search
  - No cost on arcs – costs from backed-up static evaluation
  - MAX can't be sure how MIN will respond to his moves
- **Minimax forms the basis for other game tree search algorithms.**

Alpha-Beta Pruning (AIMA 5.3)

Many slides adapted from Richard Lathrop, USC/ISI, CS 271

Alpha-Beta Pruning

- **A way to improve the performance of the Minimax Procedure**
- **Basic idea: *"If you have an idea which is surely bad, don't take the time to see how truly awful it is"* ~ Pat Winston**



- We don't need to compute the value at this node.
- No matter what it is it can't effect the value of the root node.

Alpha-Beta Pruning

- **During Minimax, keep track of two additional values:**
  - **α: MAX's current *lower* bound on MAX's outcome**
  - **β: MIN's current *upper* bound on MIN's outcome**

- **MAX will never allow a move that could lead to a worse score (for MAX) than α**
- **MIN will never allow a move that could lead to a better score (for MAX) than β**

- **Therefore, stop evaluating a branch whenever:**
  - When evaluating a MAX node: a value $v \geq \beta$ is backed-up
    - —MIN will never select that MAX node
  - When evaluating a MIN node: a value $v \leq \alpha$ is found
    - —MAX will never select that MIN node

Alpha-Beta Implementation

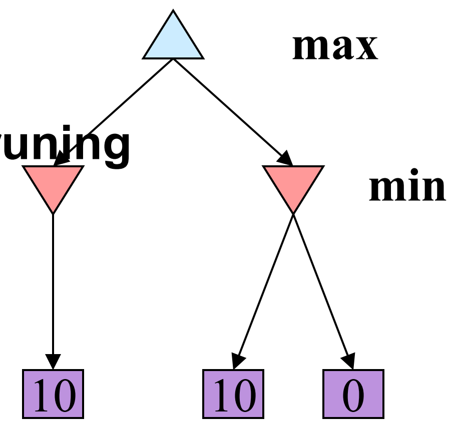
α: MAX's best option on path to root
β: MIN's best option on path to root

```
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 Pruning Properties

- **This pruning has no effect on minimax value computed for the root!**
- **Values of intermediate nodes might be wrong**
  - Important: children of the root may have the wrong value
  - So the most naïve version won't let you do action selection
- **Good child ordering improves effectiveness of pruning**
- **With "perfect ordering":**
  - Time complexity drops to $O(b^{m/2})$
  - Doubles solvable depth!
  - Full search of, e.g. chess, is still hopeless…
- **This is a simple example of metareasoning (computing about what to compute)**



Alpha-Beta Pruning

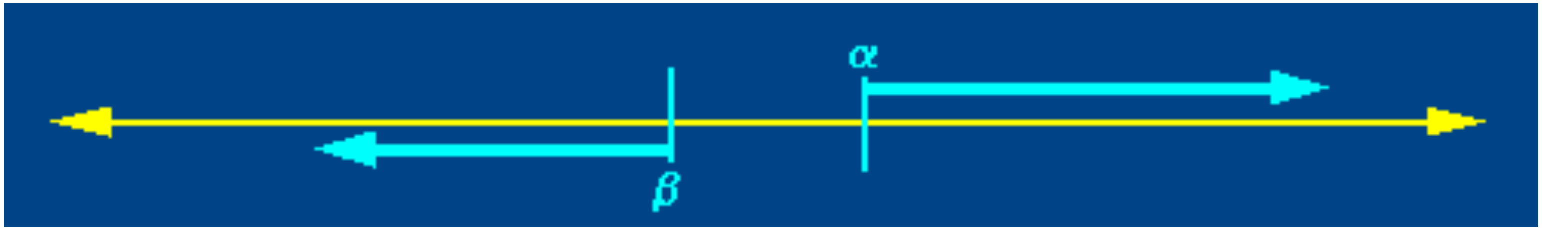
- **Based on observation that for all viable paths utility value f(*n*) will be $\alpha \le f(n) \le \beta$**

- **Initially, α = -infinity, β=infinity**

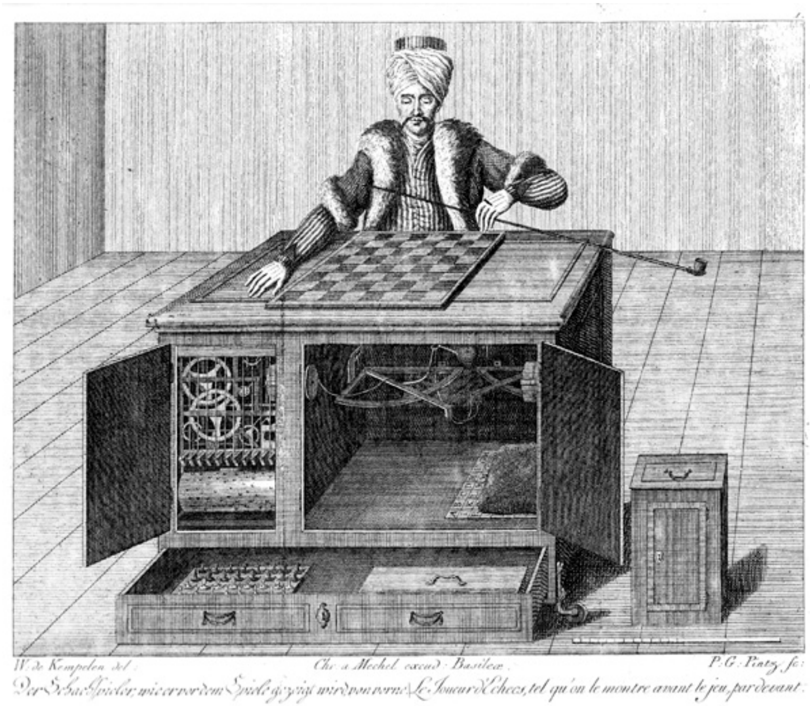
- **As the search tree is traversed, the possible utility value window shrinks as α increases, β decreases**

Alpha-Beta Pruning

- **Whenever the current ranges of alpha and beta no longer overlap, it is clear that the current node is a dead end**



Games and Adversarial Search II



Penn
UNIVERSITY of PENNSYLVANIA

Alpha-Beta Pruning (AIMA 5.3)

Some slides adapted from Richard Lathrop, USC/ISI, CS 271

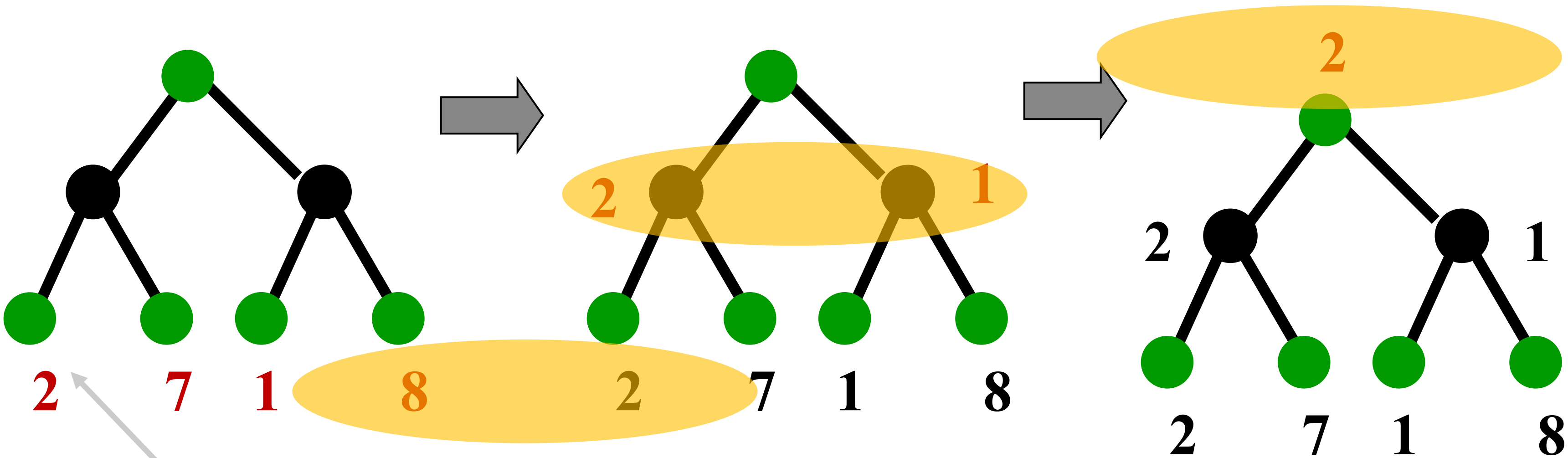
Review: The Minimax Rule

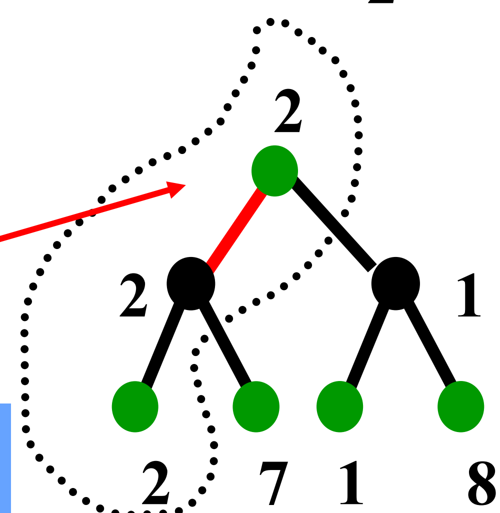
Idea: Make the best move for MAX *assuming that MIN always replies with the best move for MIN*

1. **Start with the current position as a MAX node.**
2. **Expand the game tree a fixed number of *ply*.**
3. **Apply the evaluation function to all leaf positions.**
4. **Calculate back-up values bottom-up**:
   - For a **MAX** node, return the ***maximum*** of the values of its children *(i.e. the best for MAX)*
   - For a **MIN** node, return the ***minimum*** of the values of its children *(i.e. the best for MIN*
5. **Pick the move assigned to MAX at the root**
6. **Wait for MIN to respond and REPEAT FROM 1**

MAX
MIN

2-ply Example: Backing up values





New point:
Actually calculated by DFS!

Minimax Algorithm

function MINIMAX-DECISION(*state*) **returns** *an action*
  inputs: *state*, current state in game
  $\boxed{v \leftarrow \text{MAX-VALUE}(\textit{state})}$
  return an *action* in SUCCESSORS(*state*) with value *v*

---

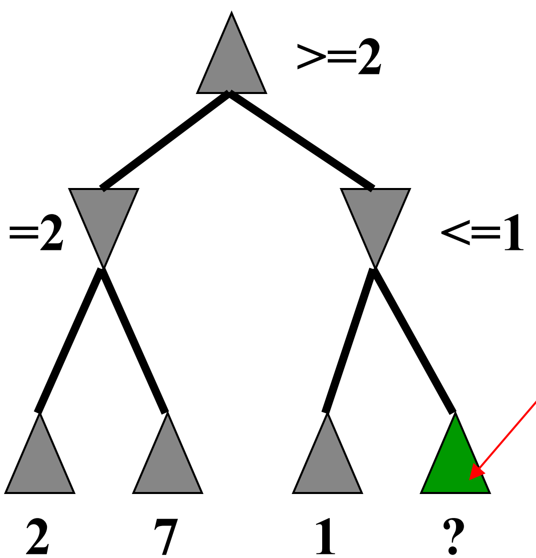
function MAX-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for *a,s* in SUCCESSORS(*state*) **do**
    $\boxed{v \leftarrow \text{MAX}(v, \textit{MIN-VALUE}(s))}$
  return *v*

---

function MIN-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow \infty$
  for *a,s* in SUCCESSORS(*state*) **do**
    $\boxed{v \leftarrow \text{MIN}(v, \textit{MAX-VALUE}(s))}$
  return *v*

Alpha-Beta Pruning

- **A way to improve the performance of the Minimax Procedure**
- **Basic idea: *"If you have an idea which is surely bad, don't take the time to see how truly awful it is"* ~ Pat Winston**



- Assuming left-to-right tree traversal:
- We don't need to compute the value at this node!
- No matter what it is it can't effect the value of the root node.

Alpha-Beta Pruning II

- **During Minimax, keep track of two additional values:**
  - **α: current *lower* bound on MAX's outcome**
  - **β: current *upper* bound on MIN's outcome**

- **MAX will never choose a move that could lead to a worse score (for MAX) than α**
- **MIN will never choose a move that could lead to a better score (for MAX) than β**

- **Therefore, stop evaluating a branch whenever:**
  - When evaluating a MAX node: a value $v \geq \beta$ is backed-up
    - —MIN will never select that MAX node
  - When evaluating a MIN node: a value $v \leq \alpha$ is found
    - —MAX will never select that MIN node

Alpha-Beta Pruning IIIa

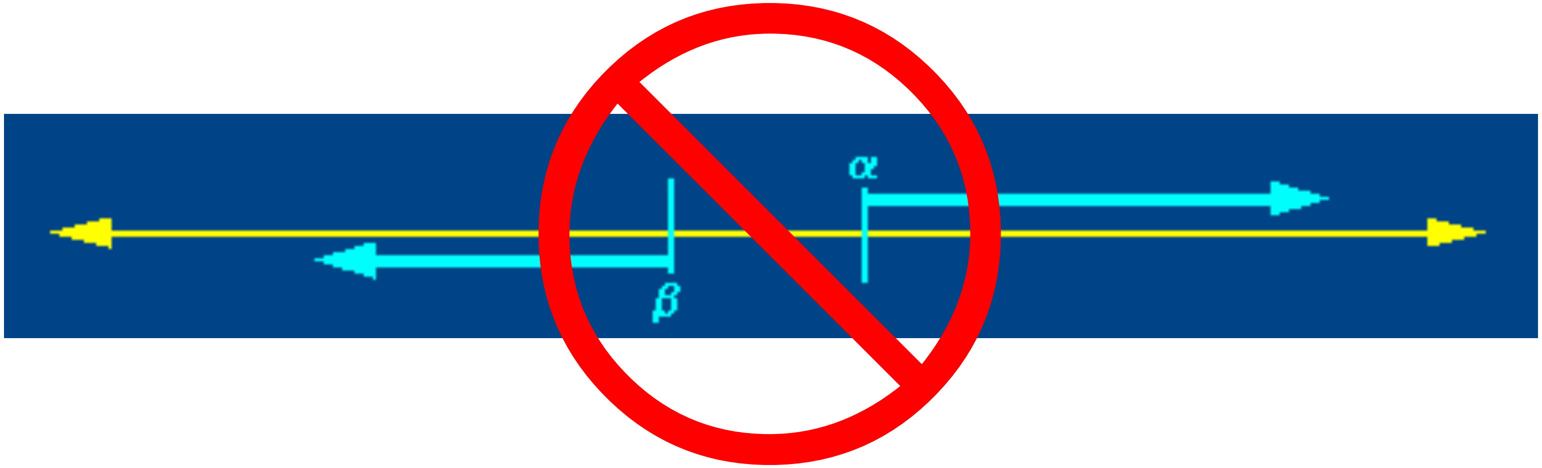
- **Based on observation that for all viable paths utility value f(*n*) will be $\alpha <= f(n) <= \beta$**

- **Initially, α = -infinity, β=infinity**

- **As the search tree is traversed, the possible utility value window shrinks as α increases, β decreases**

Alpha-Beta Pruning IIIb

- **Whenever the current ranges of alpha and beta no longer overlap ($\alpha \geq \beta$), it is clear that the current node is a dead end, so it can be pruned**



Alpha-beta Algorithm: In detail

- **Depth first search (usually bounded, with static evaluation)**
  - only considers nodes along a single path from root at any time

β
$\downarrow$

α = current *lower* bound on MAX's outcome
(initially, α = −infinity)

β = current *upper* bound on MIN's outcome
(initially, β = +infinity)

$\uparrow$
α

- **Pass current values of α and β *down* to child nodes during search.**
- **Update values of α and β during search:**
  - MAX updates α at MAX nodes
  - MIN updates β at MIN nodes
- **Prune remaining branches at a node whenever $\alpha \geq \beta$**

When to Prune

Prune whenever $\alpha \geq \beta$.

β
$\downarrow$

- Prune below a Max node when its α value becomes $\geq$ the β value of its ancestors.
  - **Max nodes update** α based on children's returned values.
  - Idea: Player MIN at node above won't pick that value anyway, since MIN can force a worse value.

$\uparrow$
α

- Prune below a Min node when its β value becomes $\leq$ the α value of its ancestors.
  - **Min nodes update** β based on children's returned values.
  - Idea: Player MAX at node above won't pick that value anyway; she can do better.

Pseudocode for Alpha-Beta Algorithm

function ALPHA-BETA-SEARCH(*state*) **returns** *an action*
  inputs: *state*, current state in game
  $\boxed{v \leftarrow \text{MAX-VALUE}(state, -\infty, +\infty)}$
  return an *action* in ACTIONS(*state*) with value *v*

Pseudocode for Alpha-Beta Algorithm

function ALPHA-BETA-SEARCH(*state*) **returns** *an action*
  inputs: *state*, current state in game
  $\boxed{v \leftarrow \text{MAX-VALUE}(state, -\infty, +\infty)}$
  return an *action* in ACTIONS(*state*) with value *v*

---

function $\boxed{\text{MAX-VALUE}(state, \alpha, \beta)}$ **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for *a* in ACTIONS(*state*) **do**
    $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{Result}(s,a), \alpha, \beta))$
    if $v \geq \beta$ **then return** *v*
    $\alpha \leftarrow \text{MAX}(\alpha, v)$
  return *v*

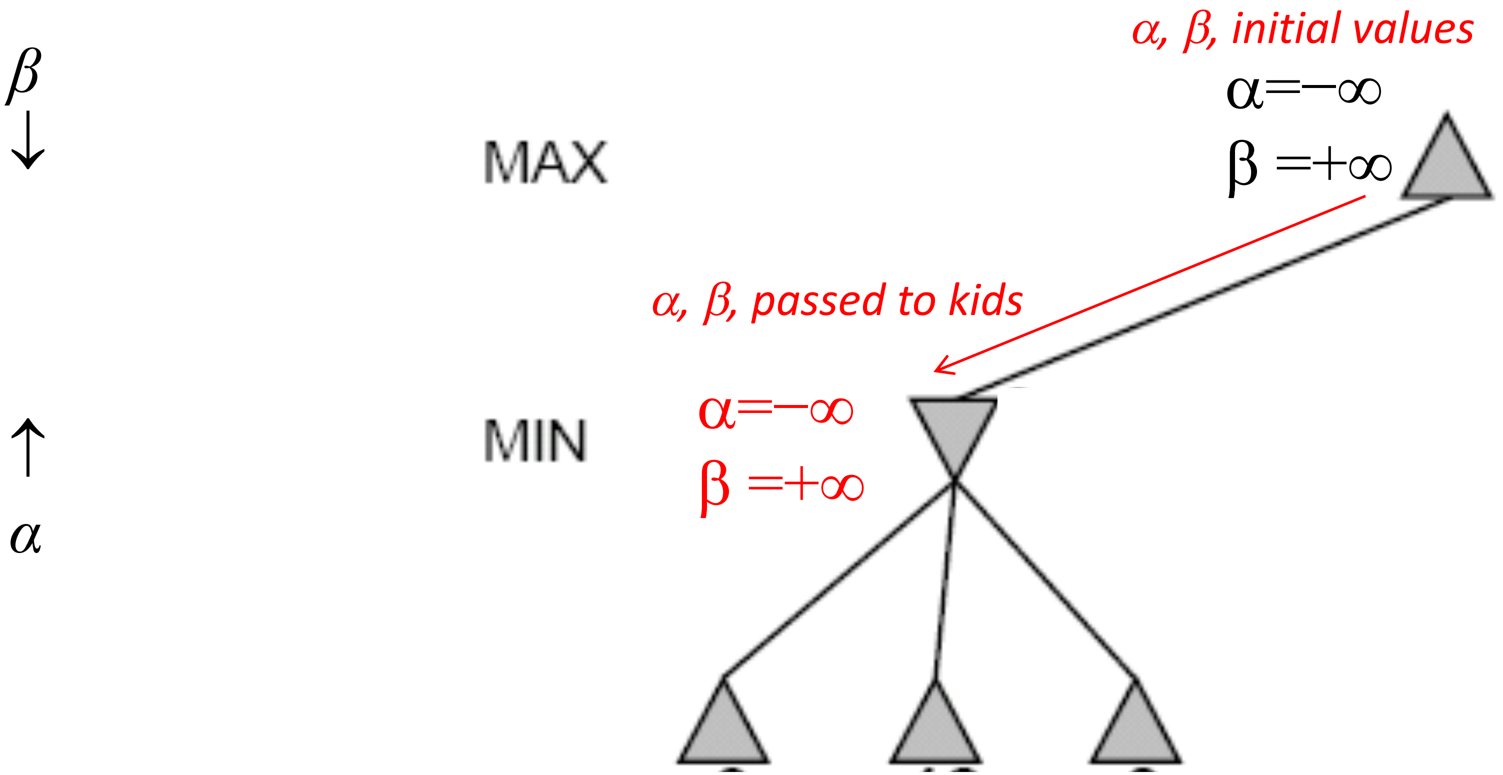
Alpha-Beta Algorithm II

---

function MIN-VALUE(*state*, α , β) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow +\infty$
  for *a,s* in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MIN(v,MAX-VALUE(s, α , β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β ,v)
  return v

An Alpha-Beta Example

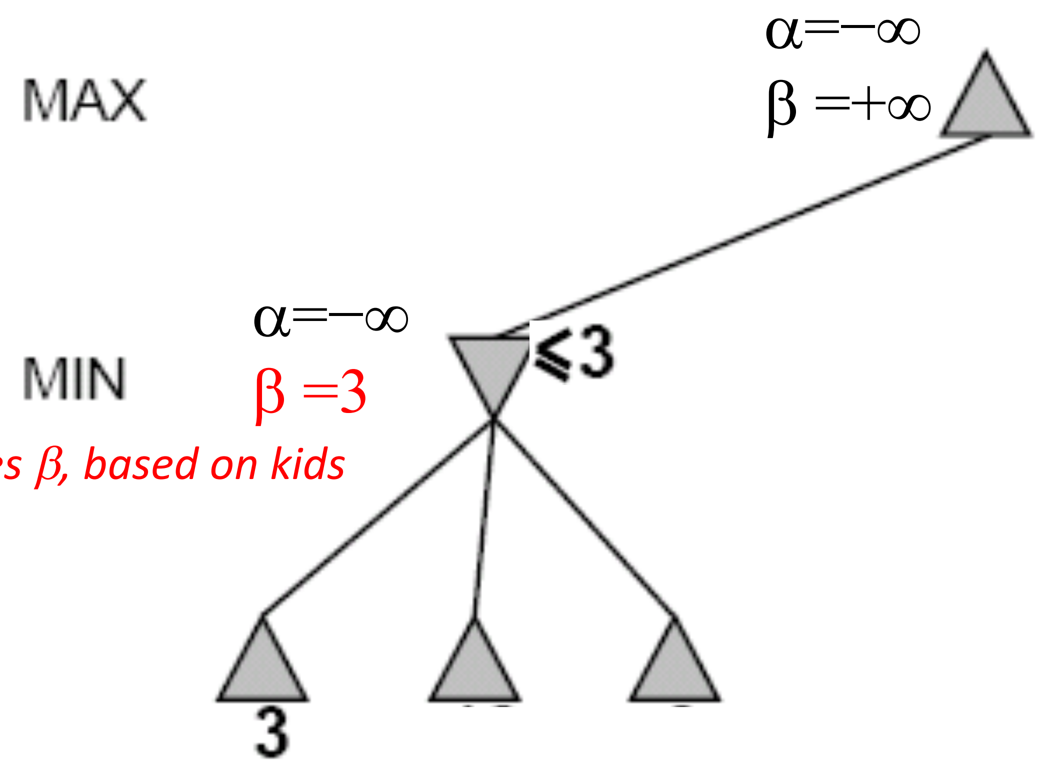
Do DF-search until first leaf



Alpha-Beta Example (continued)

β
$\downarrow$

$\uparrow$
α

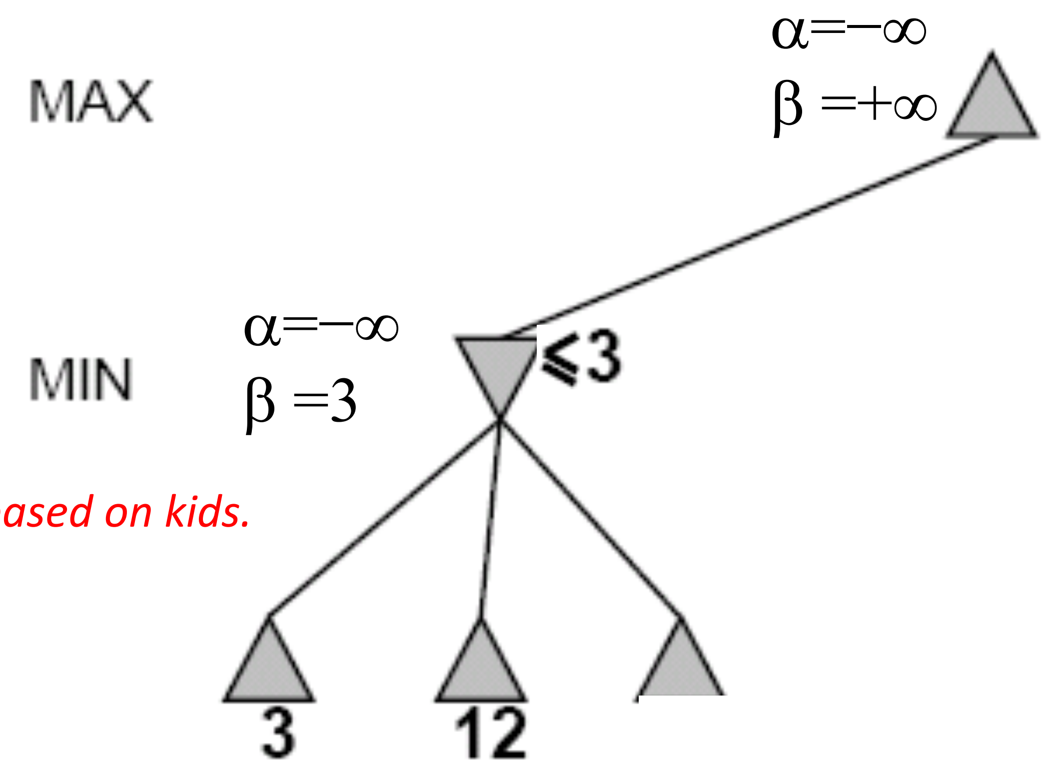


MIN updates β, based on kids

Alpha-Beta Example (continued)

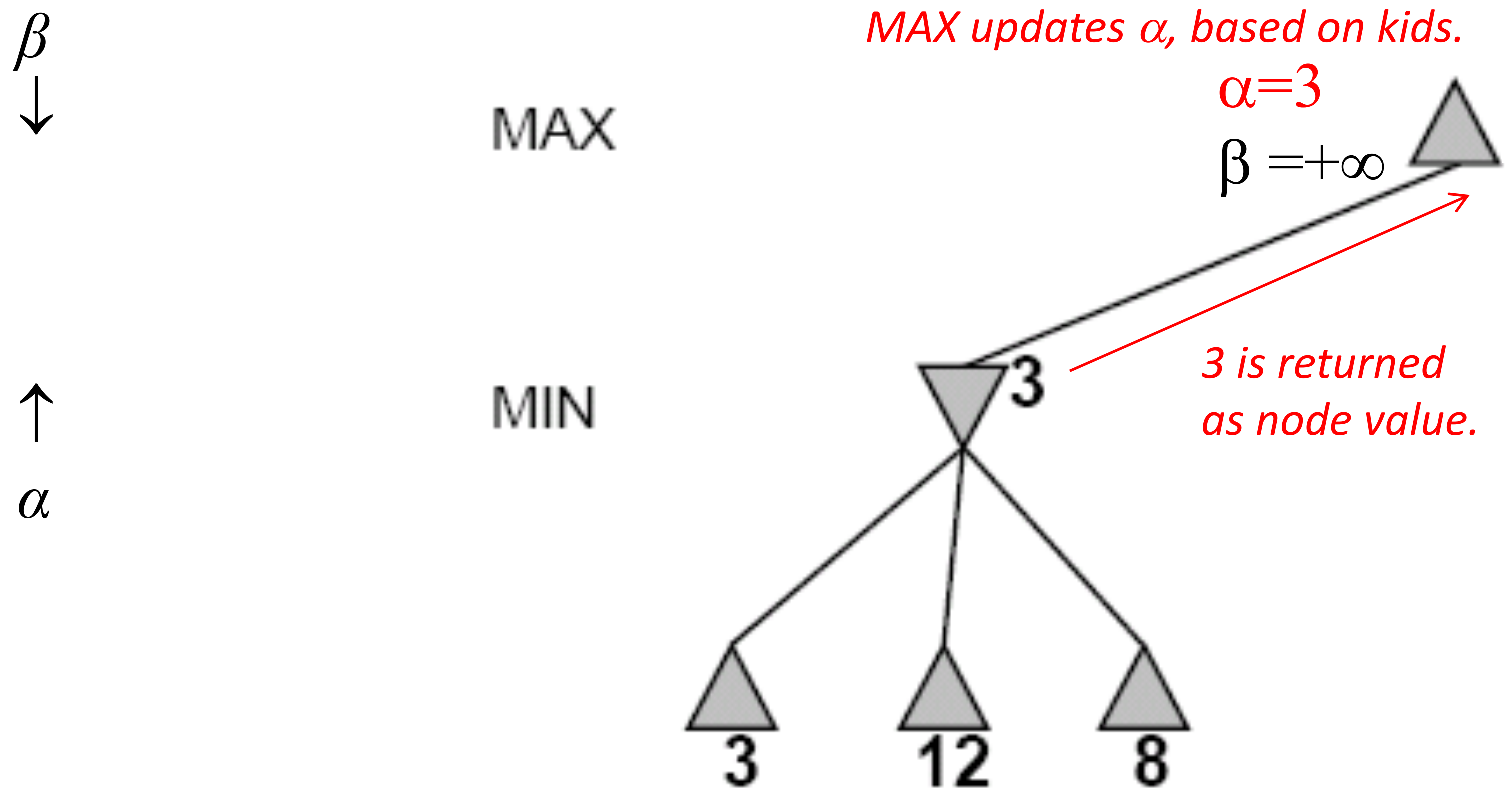
β
$\downarrow$

$\uparrow$
α

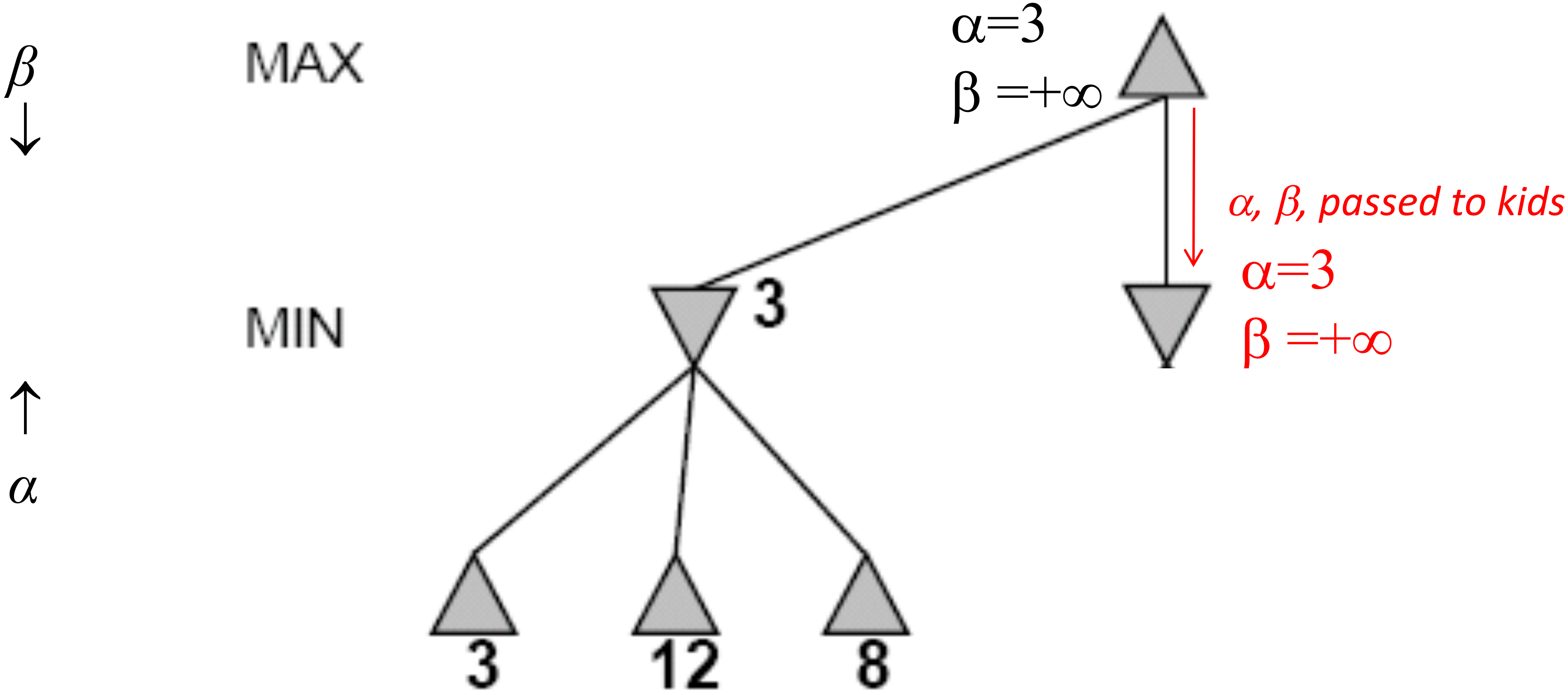


MIN updates β, based on kids.
No change.

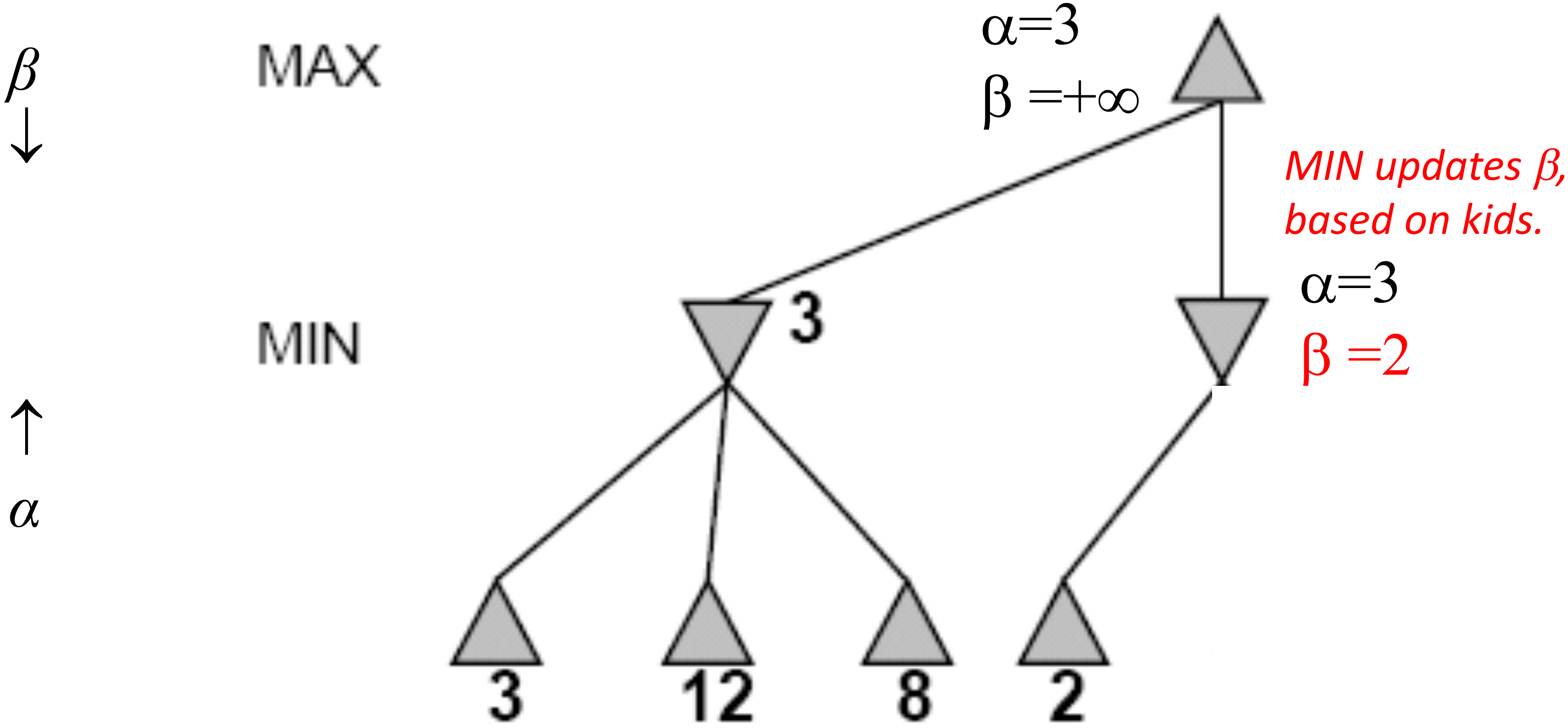
Alpha-Beta Example (continued)



Alpha-Beta Example (continued)



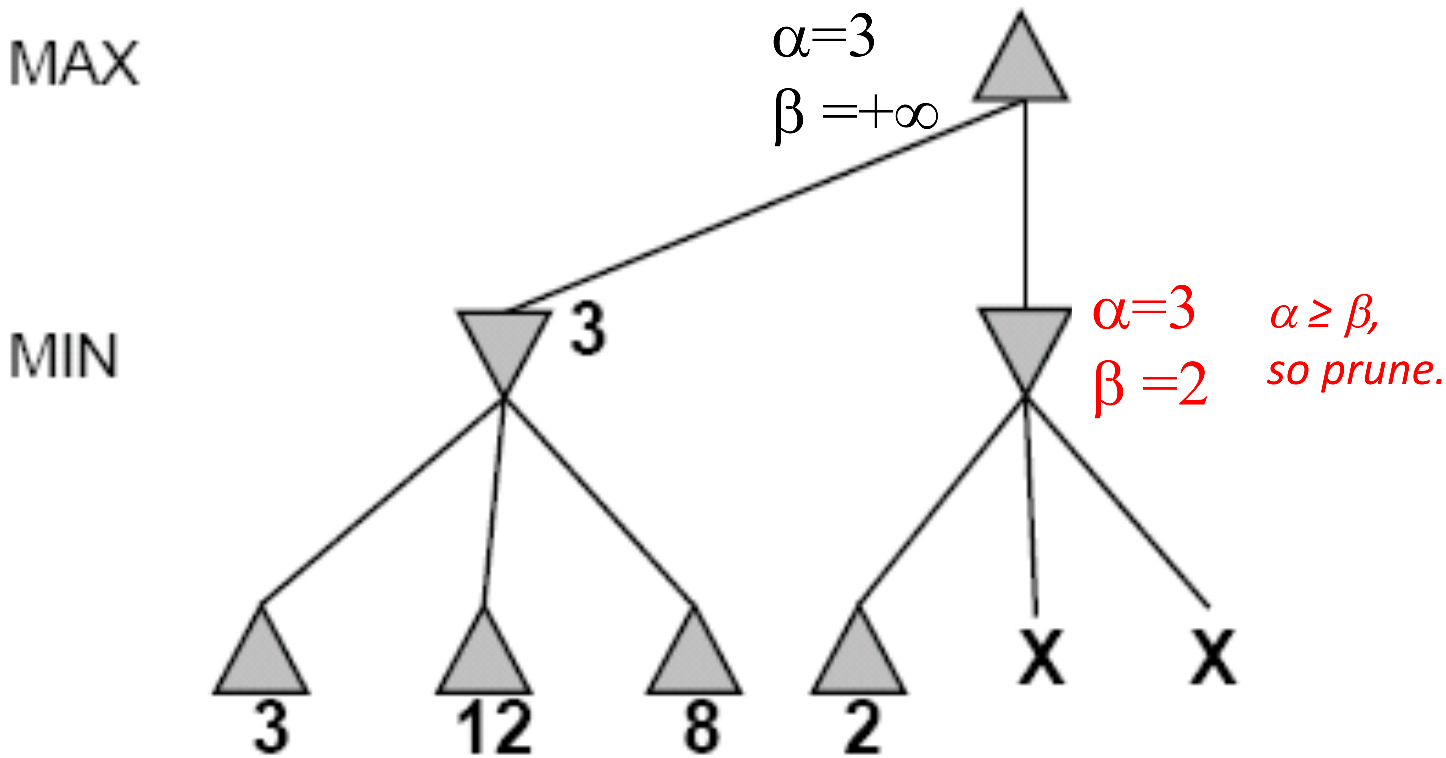
Alpha-Beta Example (continued)



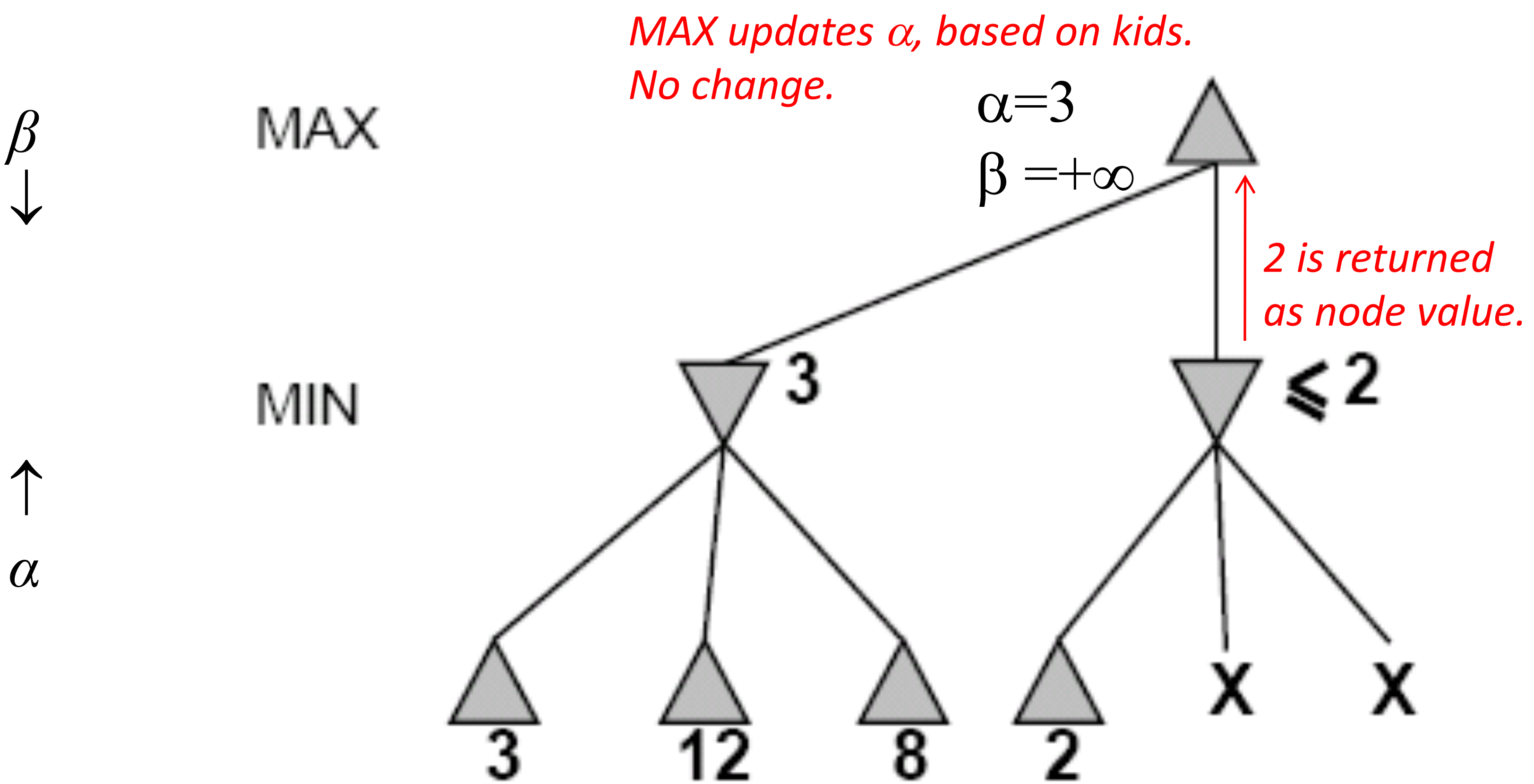
Alpha-Beta Example (continued)

β
$\downarrow$

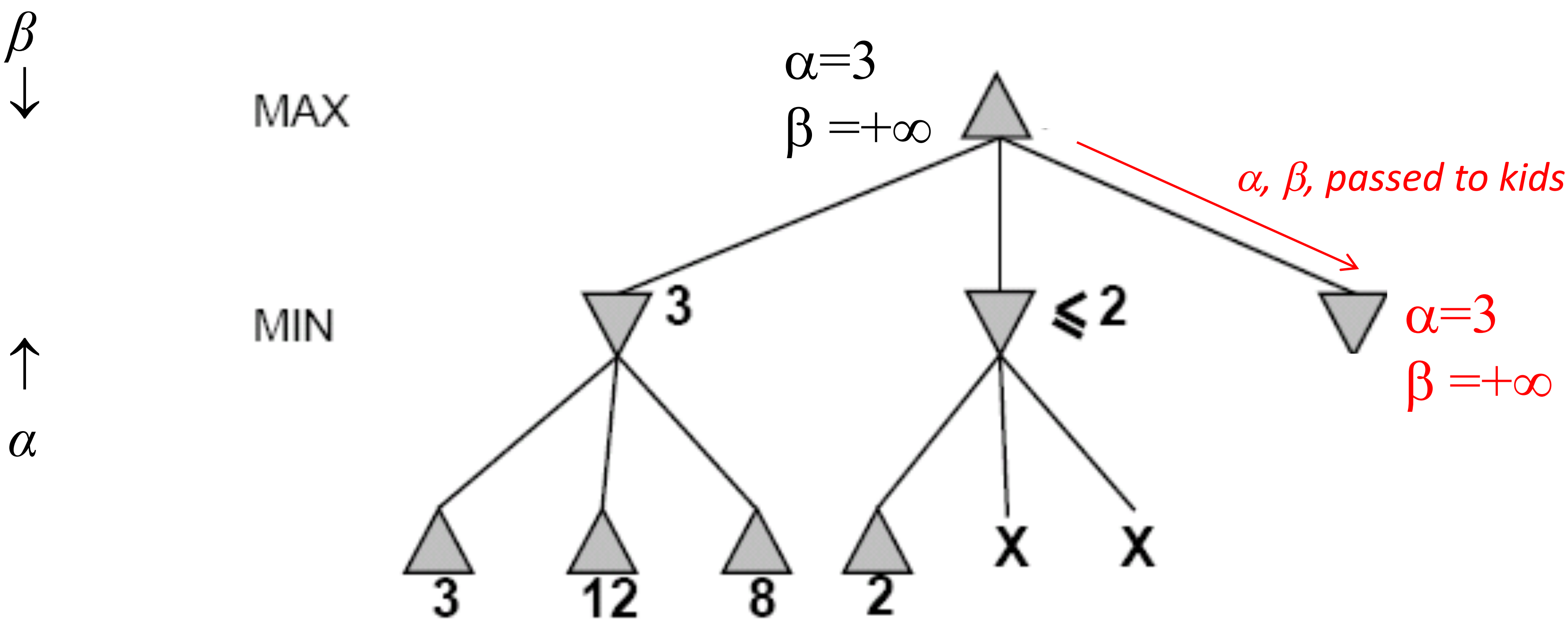
$\uparrow$
α



Alpha-Beta Example (continued)



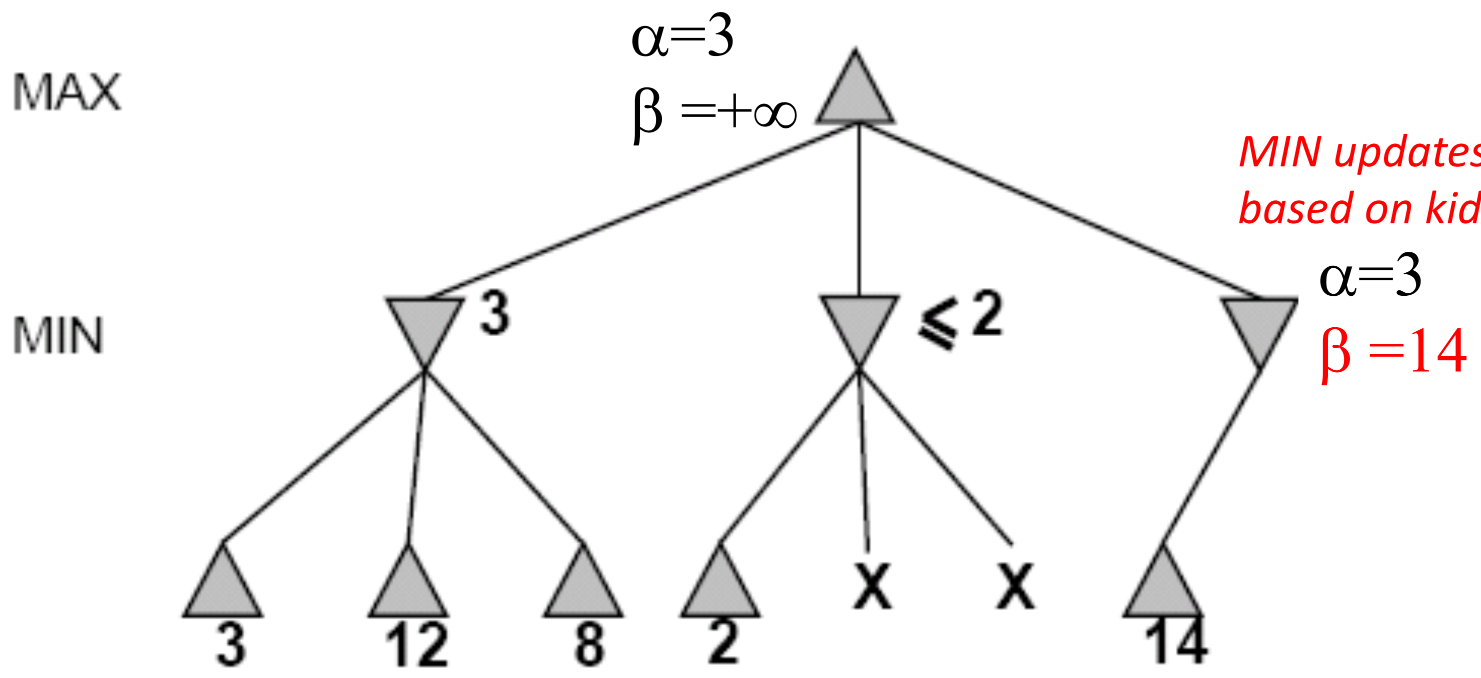
Alpha-Beta Example (continued)



Alpha-Beta Example (continued)

β ↓

↑ α

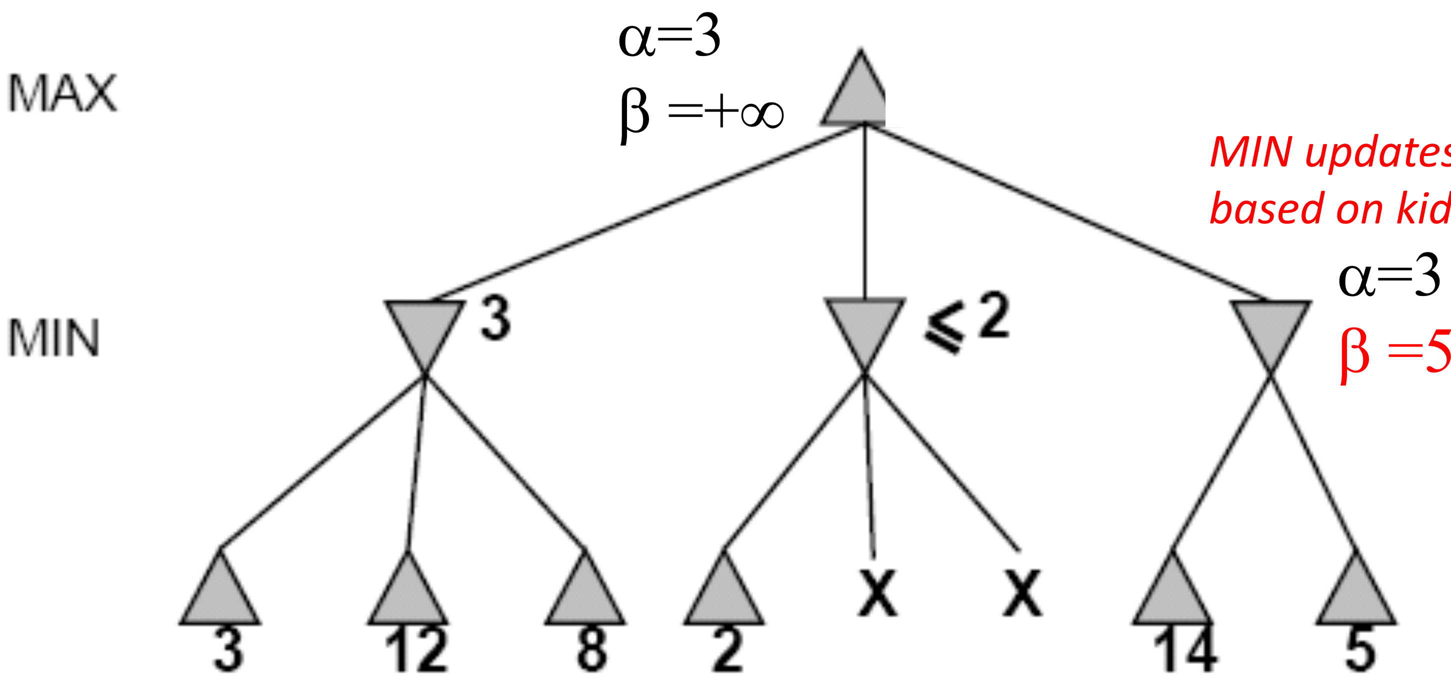


MIN updates β, based on kids.

Alpha-Beta Example (continued)

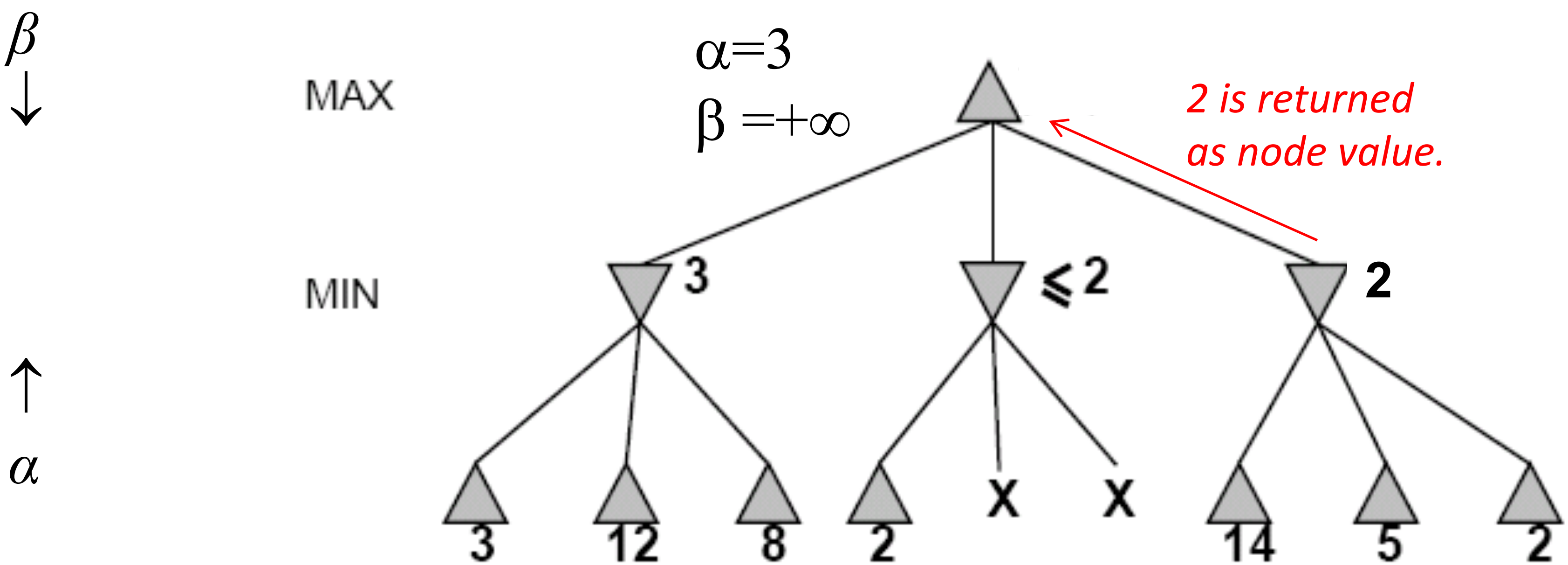
β ↓

↑ α



MIN updates β, based on kids.

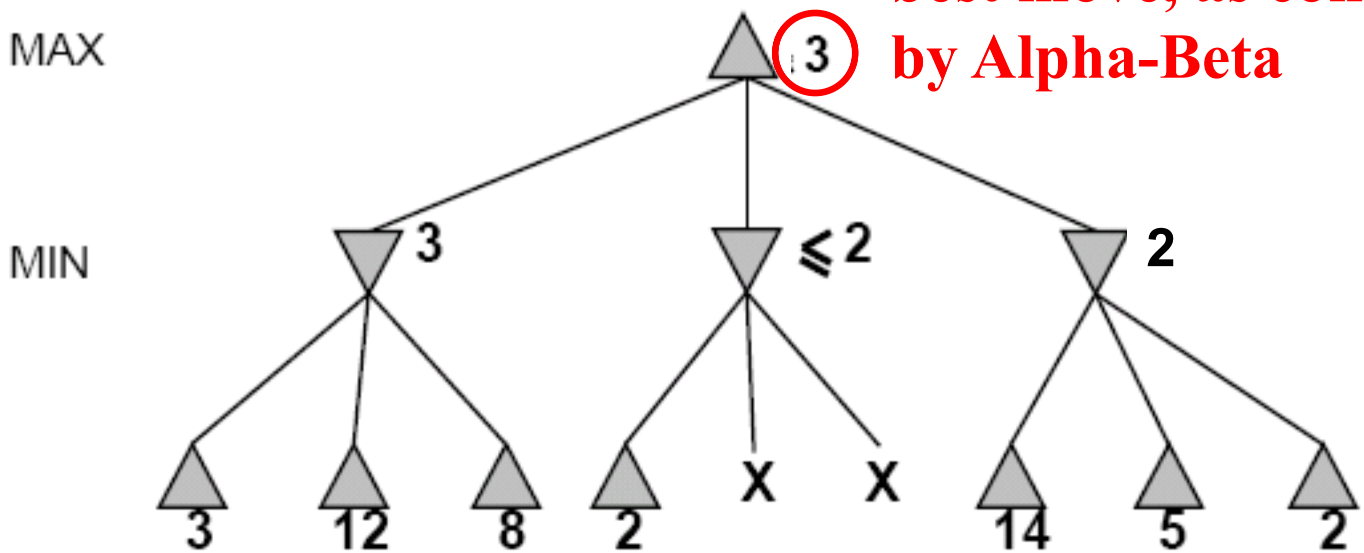
Alpha-Beta Example (continued)



Alpha-Beta Example (continued)

β ↓

↑ α



Max now makes it's best move, as computed by Alpha-Beta

Effectiveness of Alpha-Beta Pruning

- **Guaranteed to compute same root value as Minimax**
- **Worst case:** no pruning, same as Minimax ($O(b^d)$)
- **Best case:** when each player's best move is the first option examined, examines only $O(b^{d/2})$ nodes, allowing to search twice as deep!

When best move is the first examined, examines only O($b^{d/2}$) nodes….

- So: run Iterative Deepening search, sort by value returned on last iteration.
- So: expand captures first, then threats, then forward moves, etc.

- **O($b^{(d/2)}$)** is the same as having a branching factor of sqrt(b),
  - Since $(\text{sqrt}(b))^d = b^{(d/2)}$
  - e.g., in chess go from b ~ 35 to b ~ 6

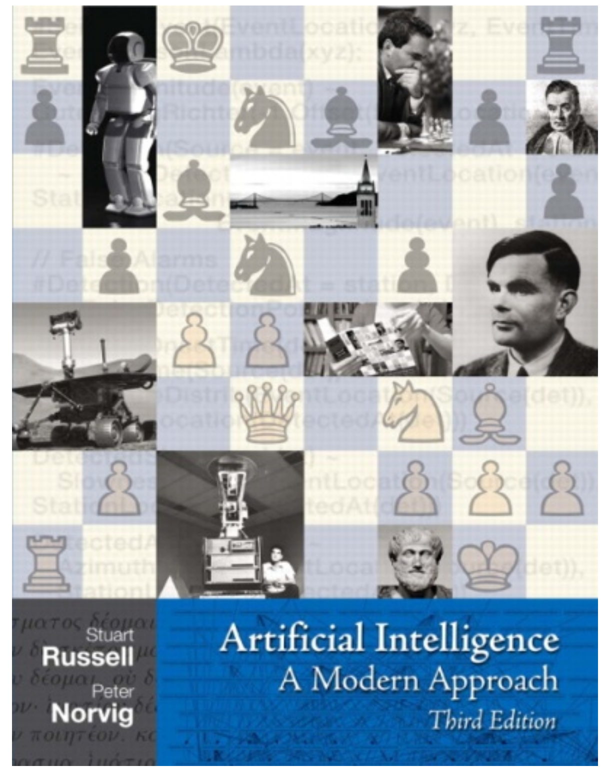
- **For Deep Blue, alpha-beta pruning reduced the average branching factor from 35-40 to 6, as expected, doubling search depth**

Real systems use a few more tricks

- **Expand the proposed solution a little farther**
  - Just to make sure there are no surprises
- **Learn better board evaluation functions**
  - E.g., for backgammon
- **Learn model of your opponent**
  - E.g., for poker
- **Do stochastic search**
  - E.g., for go

Expectimax

Read AIMA Chapter 16.1-16.3,

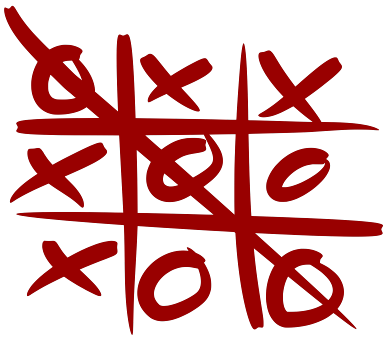


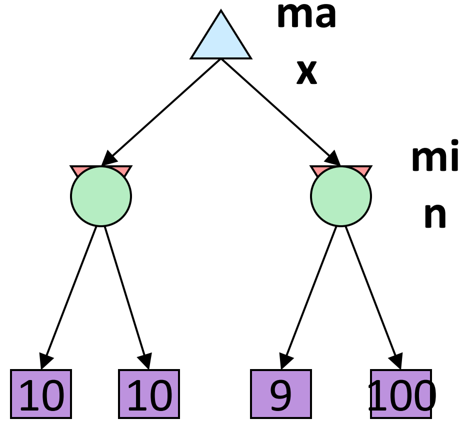
Slides courtesy of Dan Klein and Pieter Abbeel
University of California, Berkeley

CIS 421/521 - Intro to AI

Uncertain Outcomes

Worst-Case vs. Average Case

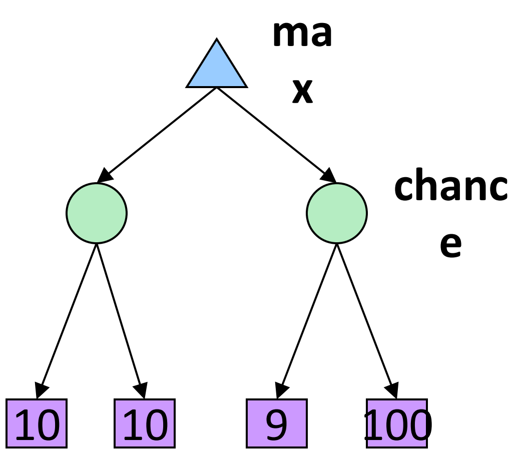




Idea: Uncertain outcomes controlled by chance, not an adversary!

Expectimax Search

- **Why wouldn't we know what the result of an action will be?**
  - Explicit randomness: rolling dice
  - Unpredictable opponents: the opponent isn't optimal
  - Actions can fail: when moving a robot, wheels might slip
- **Values should now reflect average-case (expectimax) outcomes, not worst-case (minimax) outcomes**
- **Expectimax search: compute the average score under optimal play**
  - Max nodes as in minimax search
  - Chance nodes are like min nodes but the outcome is uncertain
  - Calculate their expected utilities
  - I.e. take weighted average (expectation) of children
- **Later, we'll learn how to formalize the underlying uncertain-result problems as Markov Decision Processes**



Expectimax Pseudocode

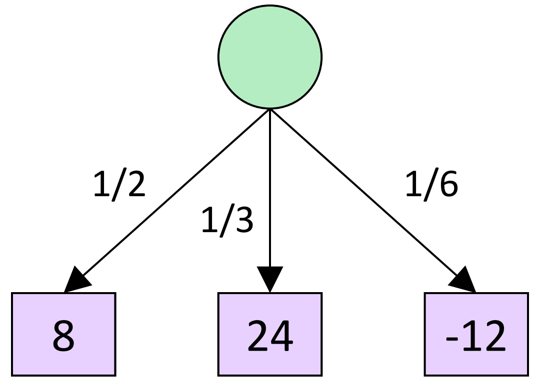
```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is EXP: return exp-value(state)
```

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

```
def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```

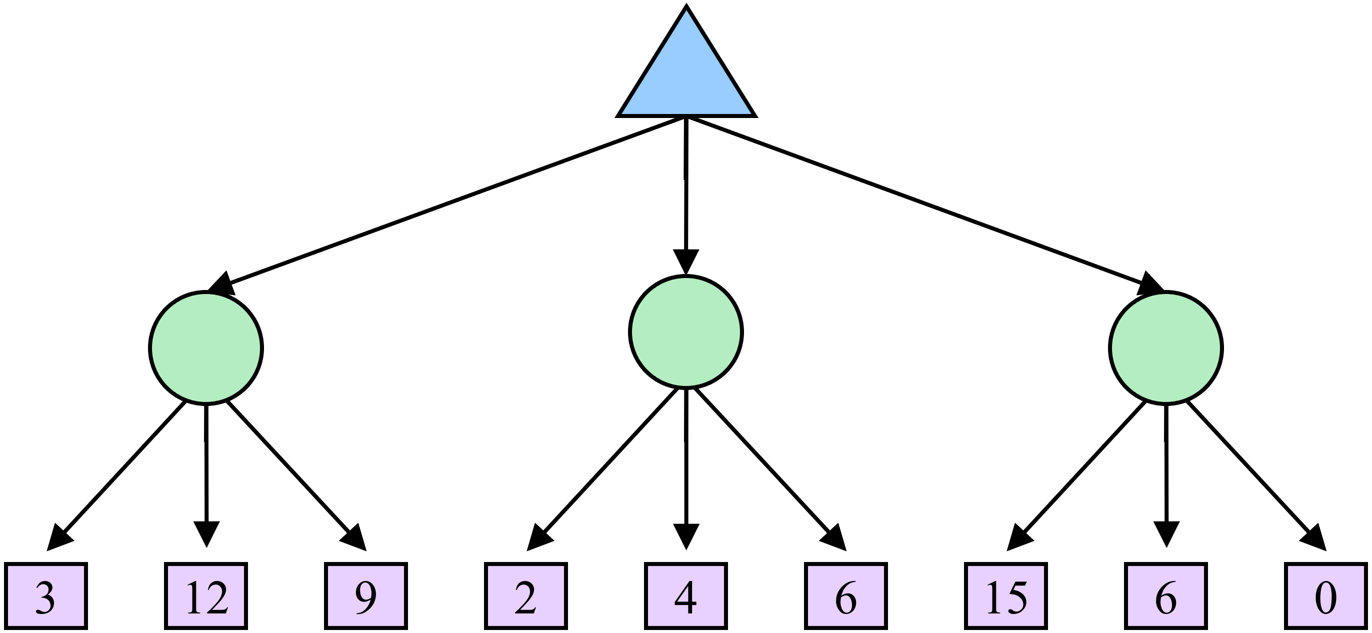
Expectimax Pseudocode

```
def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```

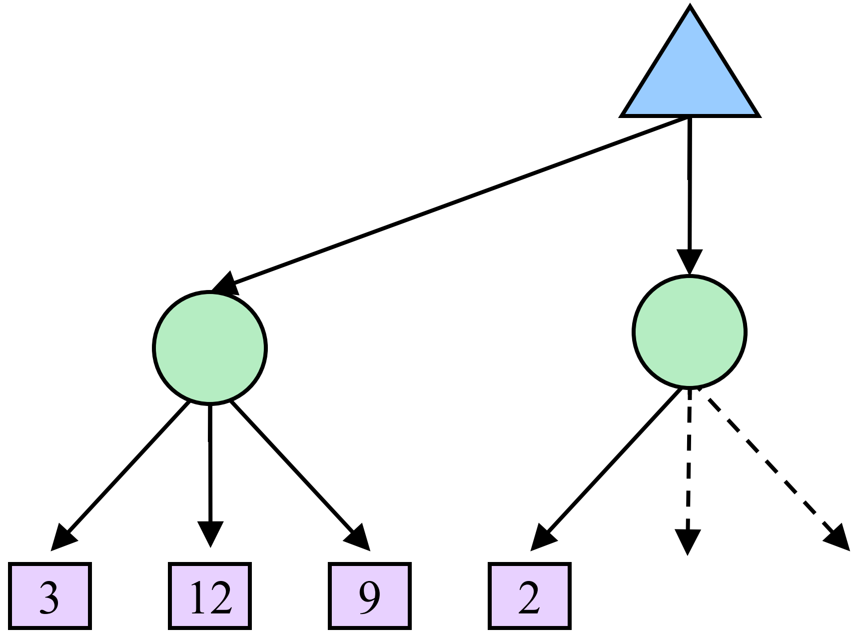


v = (1/2) (8) + (1/3) (24) + (1/6) (-12) = 10

Expectimax Example

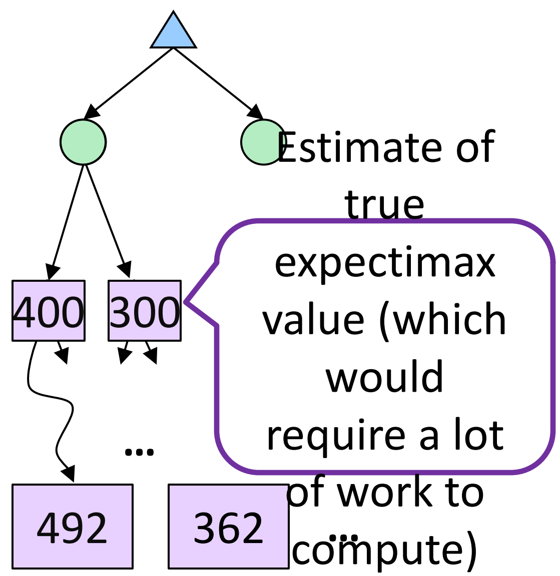


Expectimax Pruning?

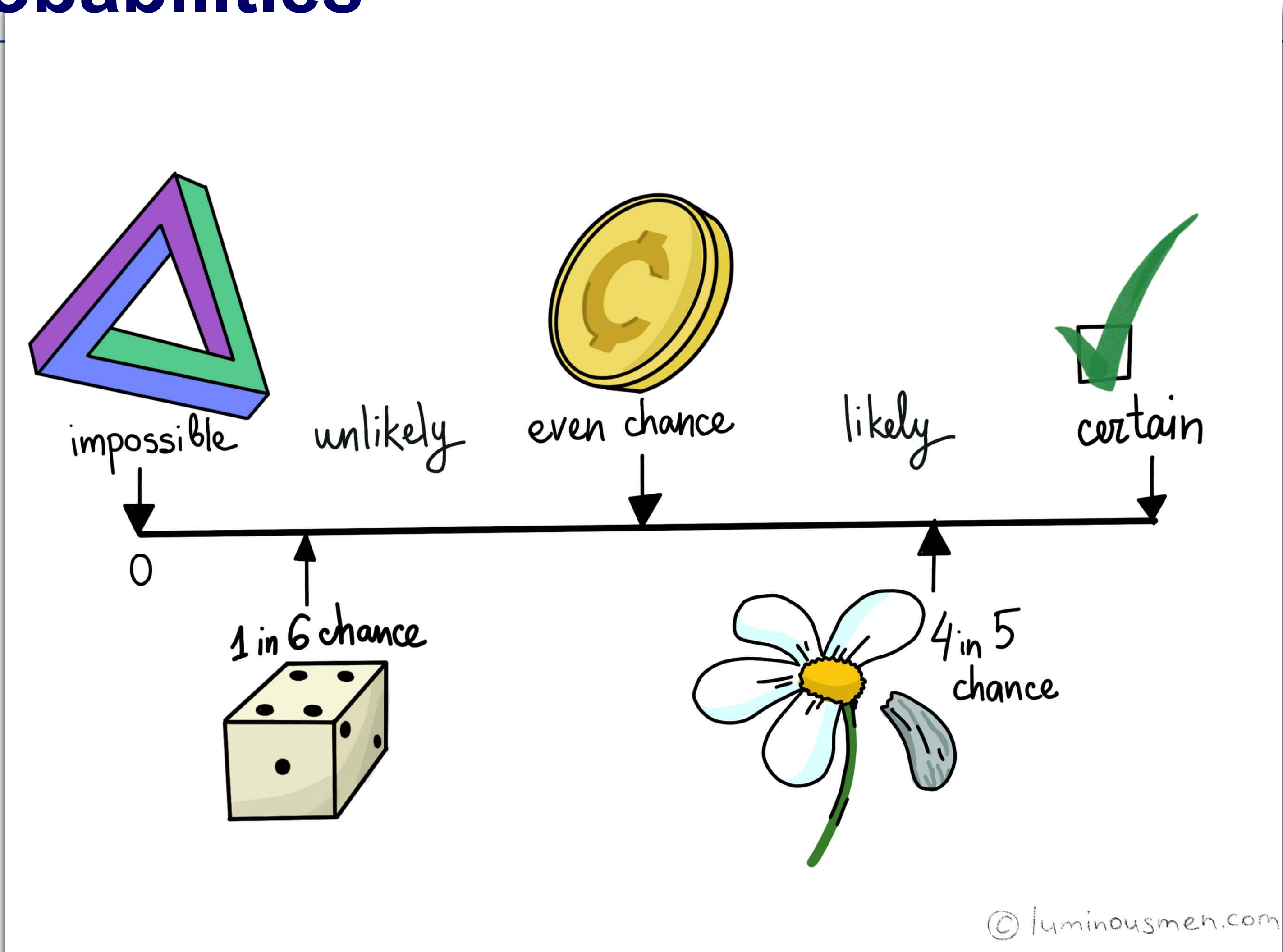


Penn
University of Pennsylvania

Depth-Limited Expectimax



Probabilities



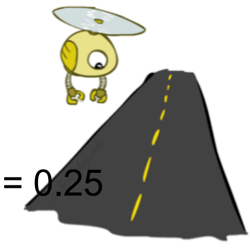
Probabilities

- **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

- **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) = 0.25, P(T=heavy | Hour=8am) = 0.60
  - We'll talk about methods for reasoning and updating probabilities later



0.25

0.50

0.25

Expectations

- **The expected value of a function of a random variable is the average, weighted by the probability distribution over outcomes**

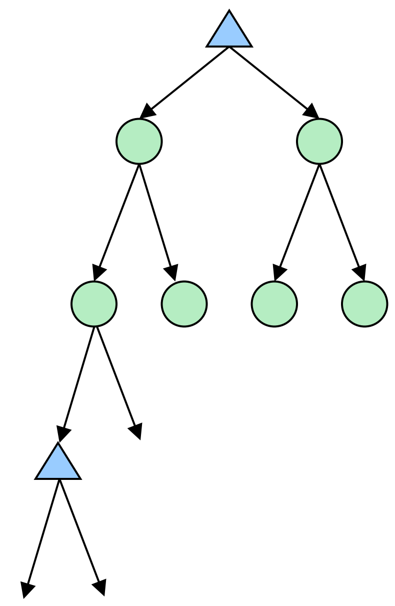
- **Example: How long to get to the airport?**

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

What Probabilities to Use?

- **In expectimax search, we have a probabilistic model of how the opponent (or environment) will behave in any state**
  - Model could be a simple uniform distribution (roll a die)
  - Model could be sophisticated and require a great deal of computation
  - We have a chance node for any outcome out of our control: opponent or environment
  - The model might say that adversarial actions are likely!

- **For now, assume each chance node magically comes along with probabilities that specify the distribution over its outcomes**



Having a probabilistic belief about another agent's action does not mean that the agent is flipping any coins!

What are Probabilities?

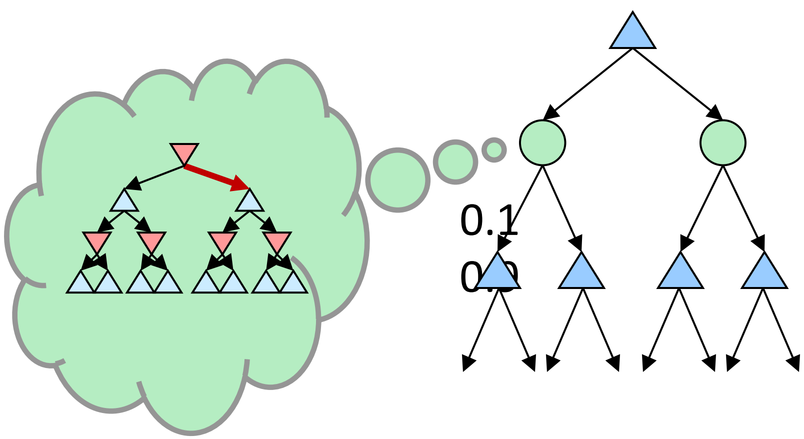
- **Objectivist / frequentist answer:**
  - Averages over repeated *experiments*
  - E.g. empirically estimating P(rain) from historical observation
  - Assertion about how future experiments will go (in the limit)
  - New evidence changes the *reference class*
  - Makes one think of *inherently random* events, like rolling c

- **Subjectivist / Bayesian answer:**
  - Degrees of belief about unobserved variables
  - E.g. an agent's belief that it's raining, given the temperatur
  - E.g. agent's belief how an opponent will behave, given the state
  - Often *learn* probabilities from past experiences (more later
  - New evidence *updates beliefs* (more later)

Penn
University *of* Pennsylvania

Quiz: Informed Probabilities

- **Let's say you know that your opponent is actually running a depth 2 minimax, using the result 80% of the time, and moving randomly otherwise**
- **Question: What tree search should you use?**



- Answer: Expectimax!
  - To figure out EACH chance node's probabilities, you have to run a simulation of your opponent
  - This kind of thing gets very slow very quickly
  - Even worse if you have to simulate your opponent simulating you…
  - … except for minimax, which has the nice property that it all collapses into one game tree

Example: Backgammon

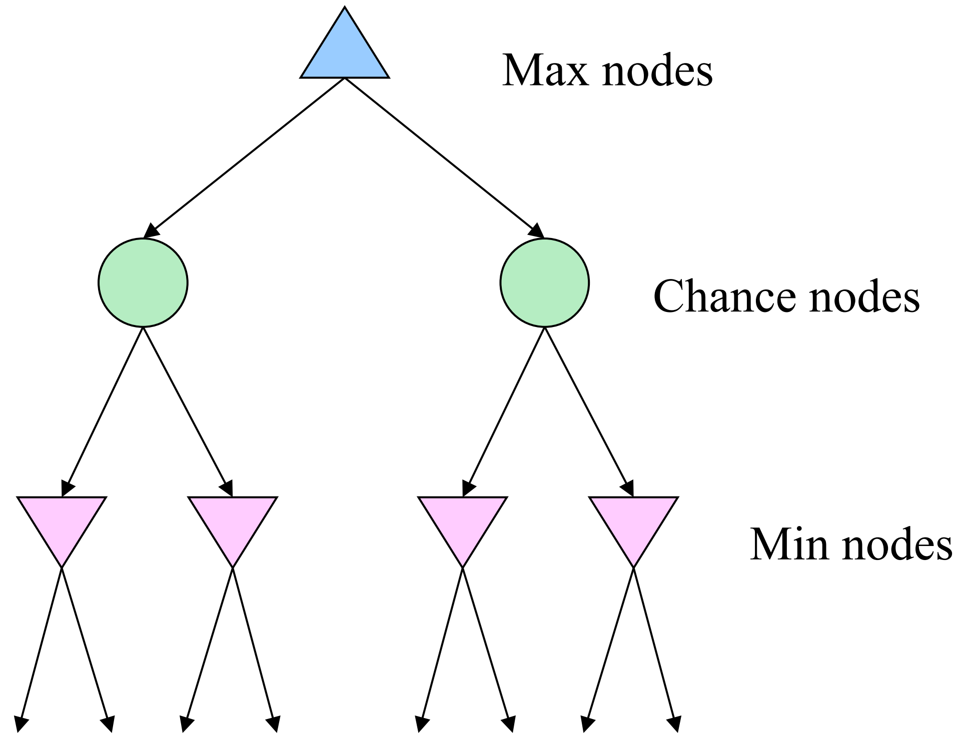
- **Dice rolls increase *b*: 21 possible rolls with 2 dice**
  - Backgammon ≈ 20 legal moves
  - Depth 2 = $20 \times (21 \times 20)^3 = 1.2 \times 10^9$
- **As depth increases, probability of reaching a given search node shrinks**
  - So usefulness of search is diminished
  - So limiting depth is less damaging
  - But pruning is trickier…
- **Historic AI: TDGammon uses depth-2 search + very good evaluation function + reinforcement learning: world-champion level play**
- **1st AI world champion in any game!**

Image: Wikipedia

Penn
UNIVERSITY of PENNSYLVANIA

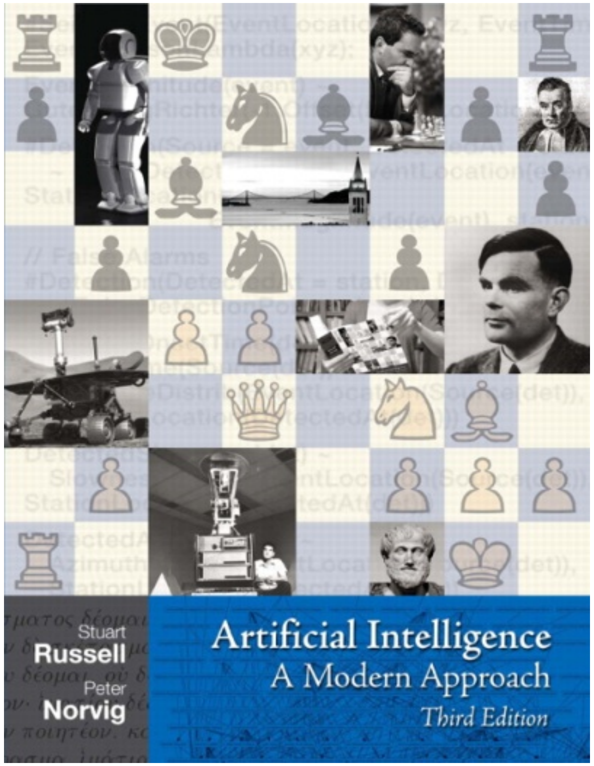
Mixed Layer Types

- **E.g. Backgammon**
- **Expectiminimax**
  - Environment is an extra "random agent" player that moves after each min/max agent
  - Each node computes the appropriate combination of its children



Utilities

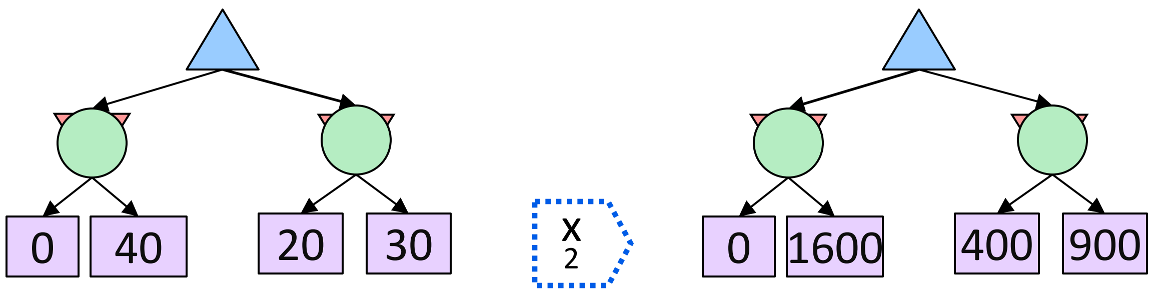
AIMA Chapter 16.1-16.3



Maximum Expected Utility

- **Why should we average utilities? Why not minimax?**
- **Principle of maximum expected utility:**
  - A rational agent should chose the action that maximizes its expected utility, given its knowledge
- **Questions:**
  - Where do utilities come from?
  - How do we know such utilities even exist?
  - How do we know that averaging even makes sense?
  - What if our behavior (preferences) can't be described by utilities?

What Utilities to Use?



- **For worst-case minimax reasoning, terminal function scale doesn't matter**
  - We just want better states to have higher evaluations (get the ordering right)
  - We call this insensitivity to monotonic transformations
- **For average-case expectimax reasoning, we need *magnitudes* to be meaningful**

Penn
UNIVERSITY of PENNSYLVANIA

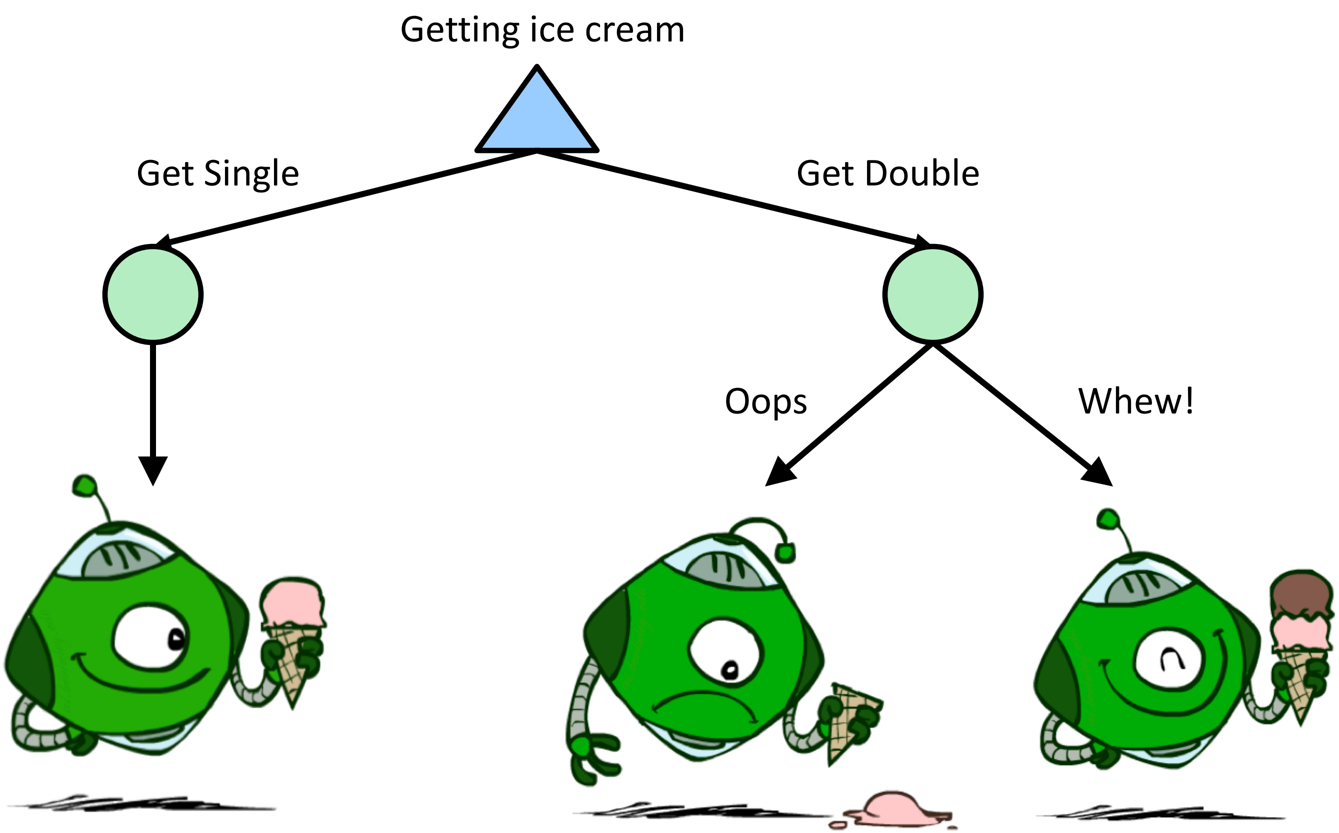
Utilities

- **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 "rational" preferences can be summarized as a utility function

- **We hard-wire utilities and let behaviors emerge**
  - Why don't we let agents pick utilities?
  - Why don't we prescribe behaviors?

Utilities: Uncertain Outcomes

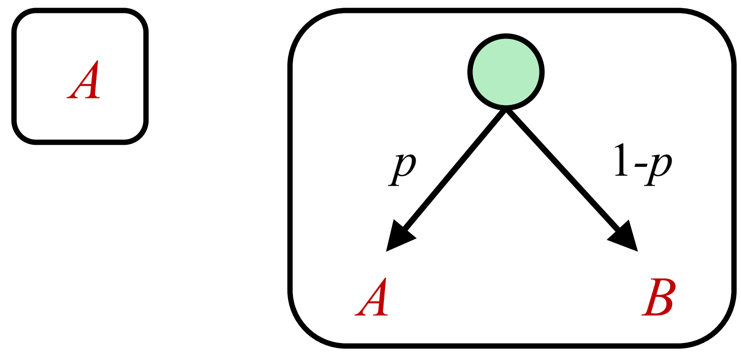


Preferences

- **An agent must have preferences among:**
  - Prizes: A, B, etc.
  - Lotteries: situations with uncertain prizes

$$L = [p, A; \ (1-p), B]$$

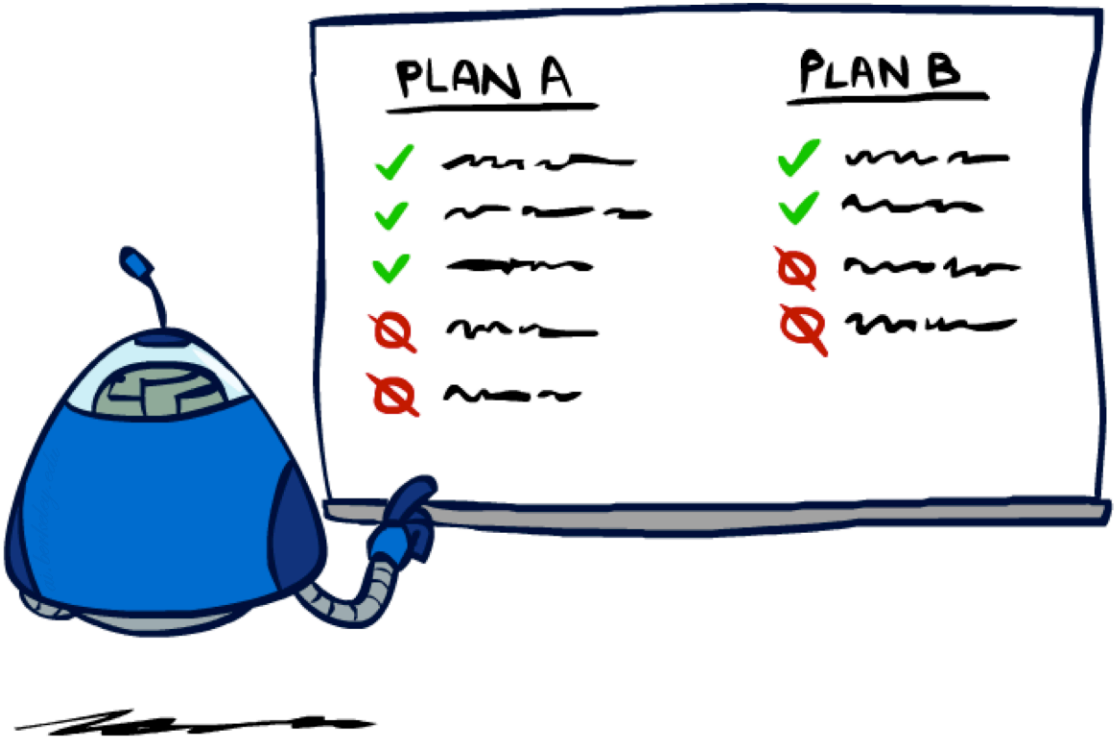
A Prize A Lottery



- **Notation:**
  - Preference: $A \succ B$
  - Indifference: $A \sim B$

Penn
UNIVERSITY of PENNSYLVANIA

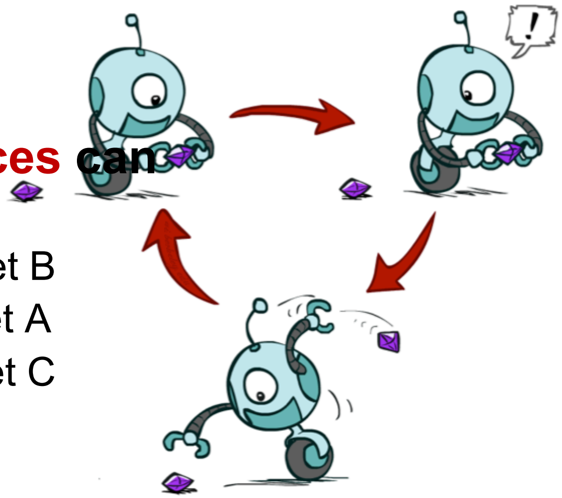
Rationality



Rational Preferences

- **We want some constraints on preferences before we call them rational, such as:**

$$\text{Axiom of Transitivity: } (A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$$



- **For example: an agent with intransitive preferences can be induced to give away all of its money**
  - If B > C, then an agent with C would pay (say) 1 cent to get B
  - If A > B, then an agent with B would pay (say) 1 cent to get A
  - If C > A, then an agent with A would pay (say) 1 cent to get C

Rational Preferences

The Axioms of Rationality

Orderability

$$(A \succ B) \vee (B \succ A) \vee (A \sim B)$$

Transitivity

$$(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$$

Continuity

$$A \succ B \succ C \Rightarrow \exists p \; [p, A; \; 1-p, C] \sim B$$

Substitutability

$$A \sim B \Rightarrow [p, A; \; 1-p, C] \sim [p, B; 1-p, C]$$

Monotonicity

$$A \succ B \Rightarrow (p \geq q \Leftrightarrow [p, A; \; 1-p, B] \succsim [q, A; \; 1-q, B])$$

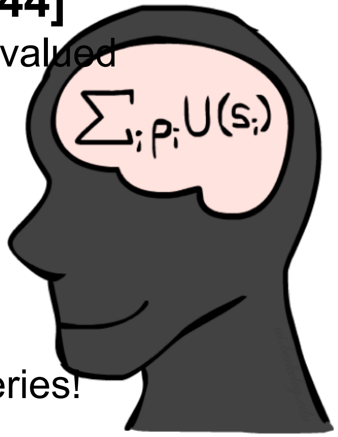
Theorem: Rational preferences imply behavior describable as maximization of expected utility

MEU Principle

- **Theorem [Ramsey, 1931; von Neumann & Morgenstern, 1944]**
  - Given any preferences satisfying these constraints, there exists a real-valued function U such that:

$$U(A) \geq U(B) \;\Leftrightarrow\; A \succeq B$$

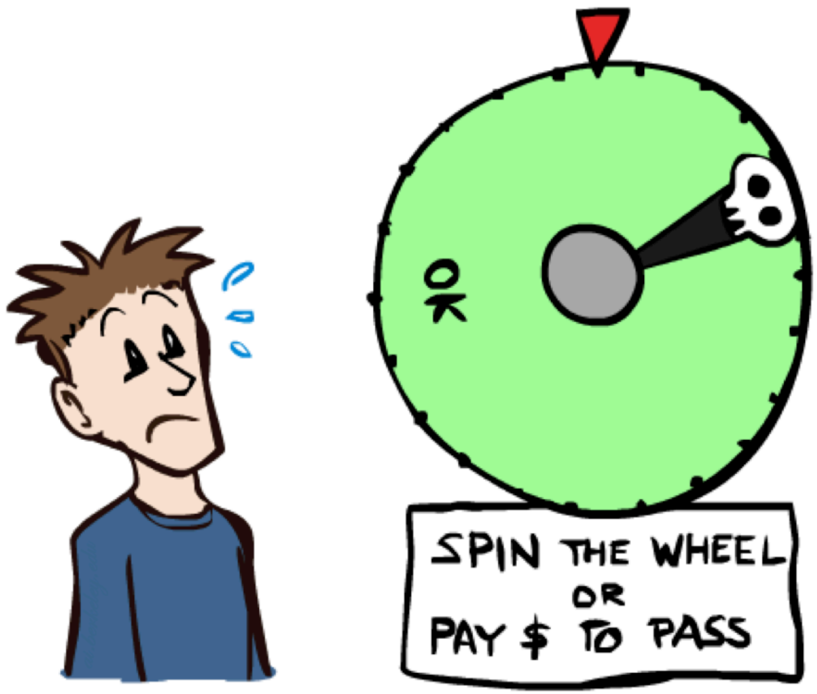
$$U([p_1, S_1;\ \ldots\ ;\ p_n, S_n]) = \sum_i p_i U(S_i)$$



  - I.e. values assigned by U preserve preferences of both prizes and lotteries!

- **Maximum expected utility (MEU) principle:**
  - Choose the action that maximizes expected utility
  - Note: an agent can be entirely rational (consistent with MEU) without ever representing or manipulating utilities and probabilities
  - E.g., a lookup table for perfect tic-tac-toe, a reflex vacuum cleaner

Human Utilities



Utility Scales

- **Normalized utilities**: $u_+ = 1.0$, $u_- = 0.0$
- **Micromorts**: one-millionth chance of death, useful for paying to reduce product risks, etc.
- **QALYs**: quality-adjusted life years, useful for medical decisions involving substantial risk
- Note: behavior is invariant under positive linear transformation

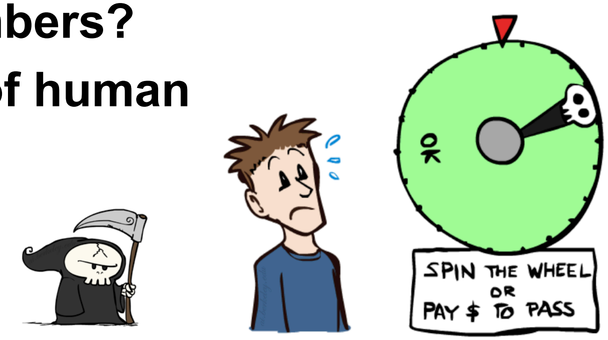
$$U'(x) = k_1 U(x) + k_2 \quad \text{where } k_1 > 0$$

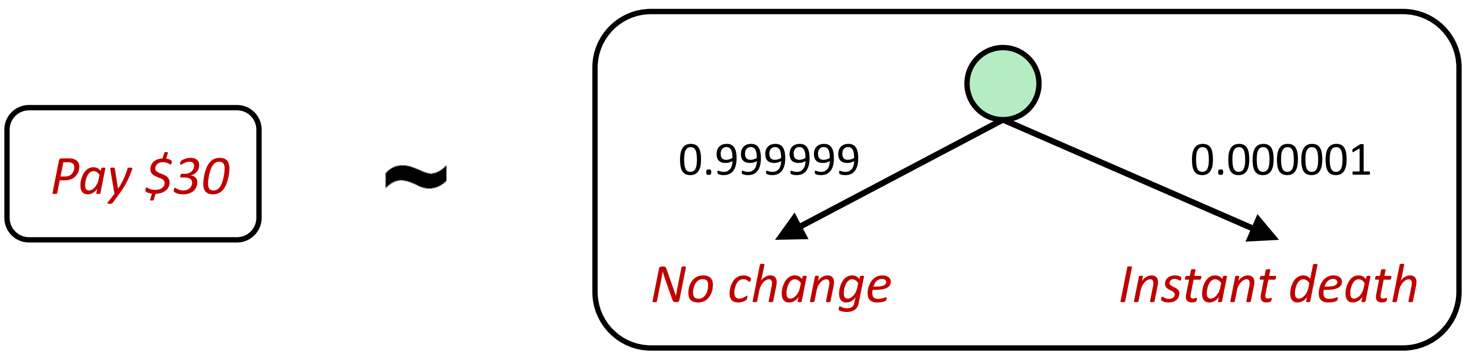
- With deterministic prizes only (no lottery choices), only **ordinal utility** can be determined, i.e., total order on prizes

Penn
UNIVERSITY of PENNSYLVANIA

Human Utilities

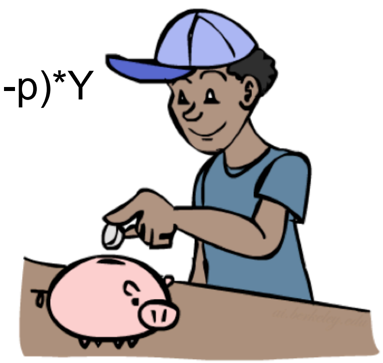
- **Utilities map states to real numbers. Which numbers?**
- **Standard approach to assessment (elicitation) of human**
  - Compare a prize A to a standard lottery L_p between
    - "best possible prize" u_+ with probability p
    - "worst possible catastrophe" u_- with probability 1-p
  - Adjust lottery probability p until indifference: A ~ L_p
  - Resulting p is a utility in [0,1]

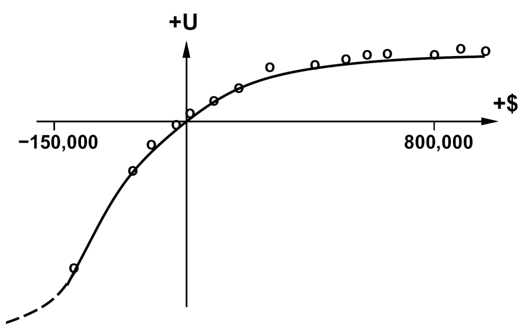




Money

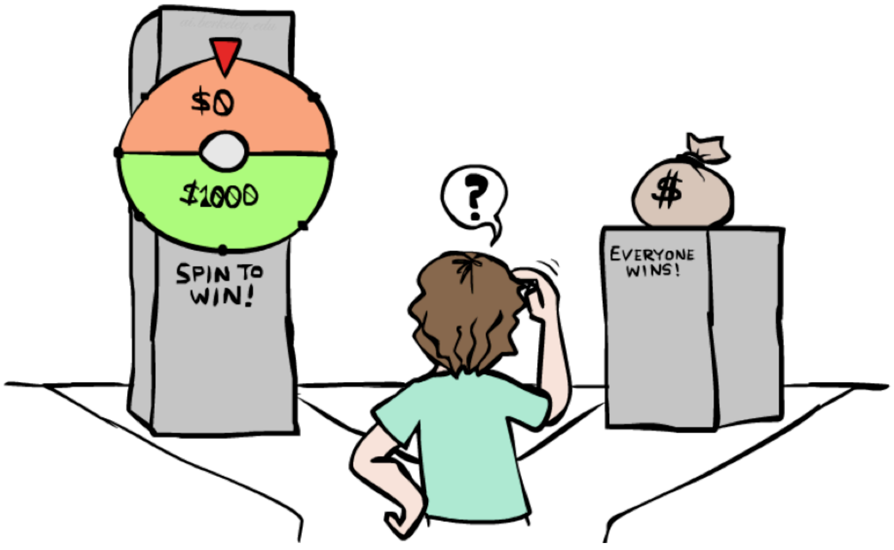
- **Money <u>does</u> <u>not</u> behave as a utility function, but we can talk about the utility of having money (or being in debt)**
- **Given a lottery L = [p, $X; (1-p), $Y]**
  - The expected monetary value EMV(L) is p*X + (1-p)*Y
  - U(L) = p*U($X) + (1-p)*U($Y)
  - Typically, U(L) < U(EMV(L))
  - In this sense, people are risk-averse
  - When deep in debt, people are risk-prone





Example: Insurance

- **Consider the lottery [0.5, $1000; 0.5, $0]**
  - What is its expected monetary value? ($500)
  - What is its certainty equivalent?
    - — Monetary value acceptable in lieu of lottery
    - — $400 for most people
  - Difference of $100 is the insurance premium
    - — There's an insurance industry because people will pay to reduce their risk
    - — If everyone were risk-neutral, no insurance needed!
  - It's win-win: you'd rather have the $400 and the insurance company would rather have the lottery (their utility curve is linear and they have many lotteries)



Example: Human Rationality?

- **Famous example of Allais (1953)**
  - A: [0.8, $4k;   0.2, $0]
  - B: [1.0, $3k;   0.0, $0]

  - C: [0.2, $4k;   0.8, $0]
  - D: [0.25, $3k;   0.75, $0]

- **Most people prefer B > A, C > D**

- **But if U($0) = 0, then**
  - B > A $\Rightarrow$ U($3k) > 0.8 U($4k)
  - C > D $\Rightarrow$ 0.8 U($4k) > U($3k)