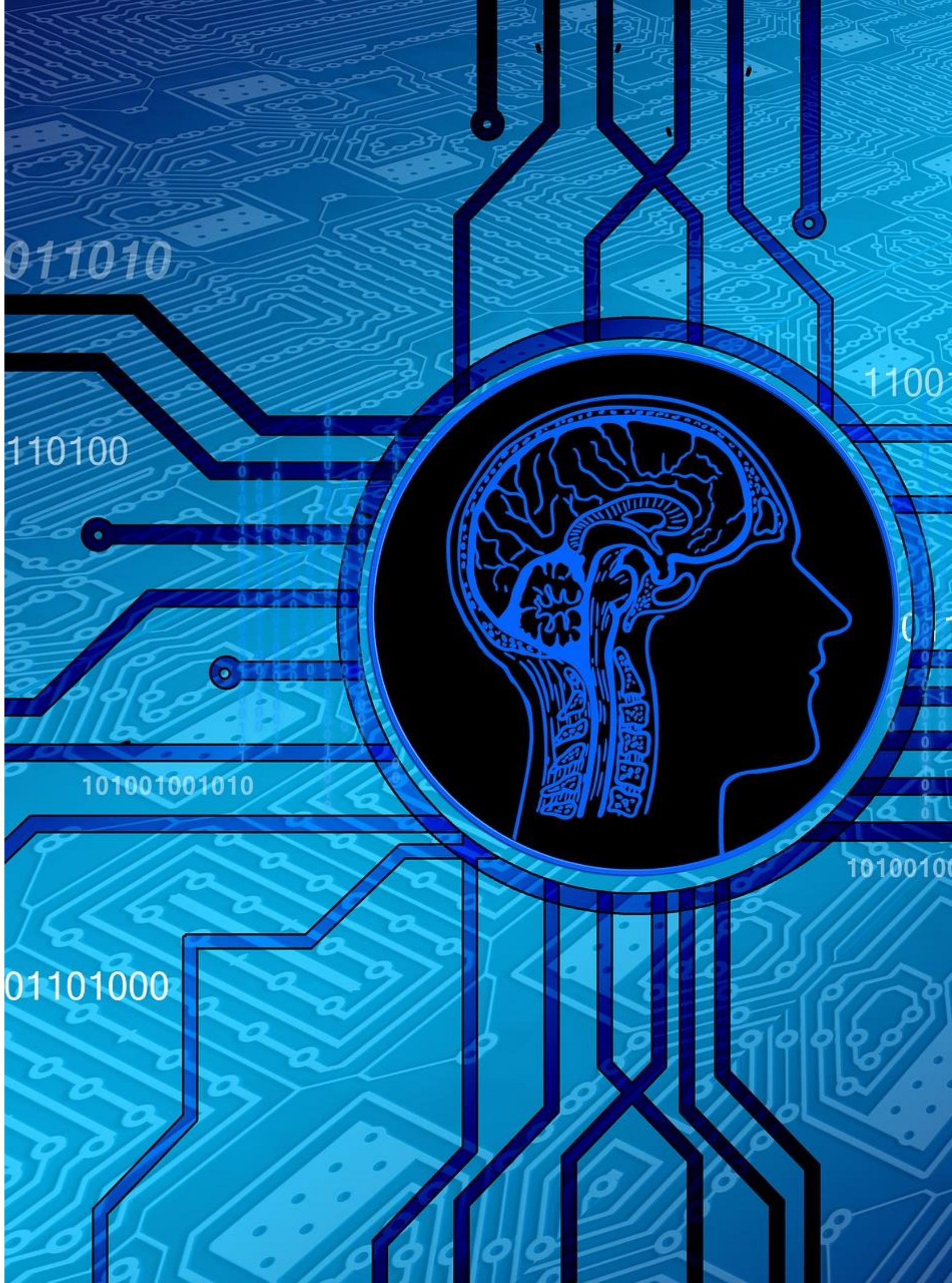
Adversarial Search

Informatics 2D: Reasoning and Agents
Lecture 7

Adapted from slides provided by Dr Petros Papapanagiotou



Games vs. Search Problems

"Unpredictable" opponent → solution is a **strategy** / **policy**

- Specify a move for *every possible* opponent reply

Time limits → unlikely to find goal, must approximate

Discrete!

TYPES OF GAMES	**deterministic**	**chance**
perfect information	Chess, Checkers	Backgammon, Monopoly
imperfect information	Battleship	Card games, Scrabble

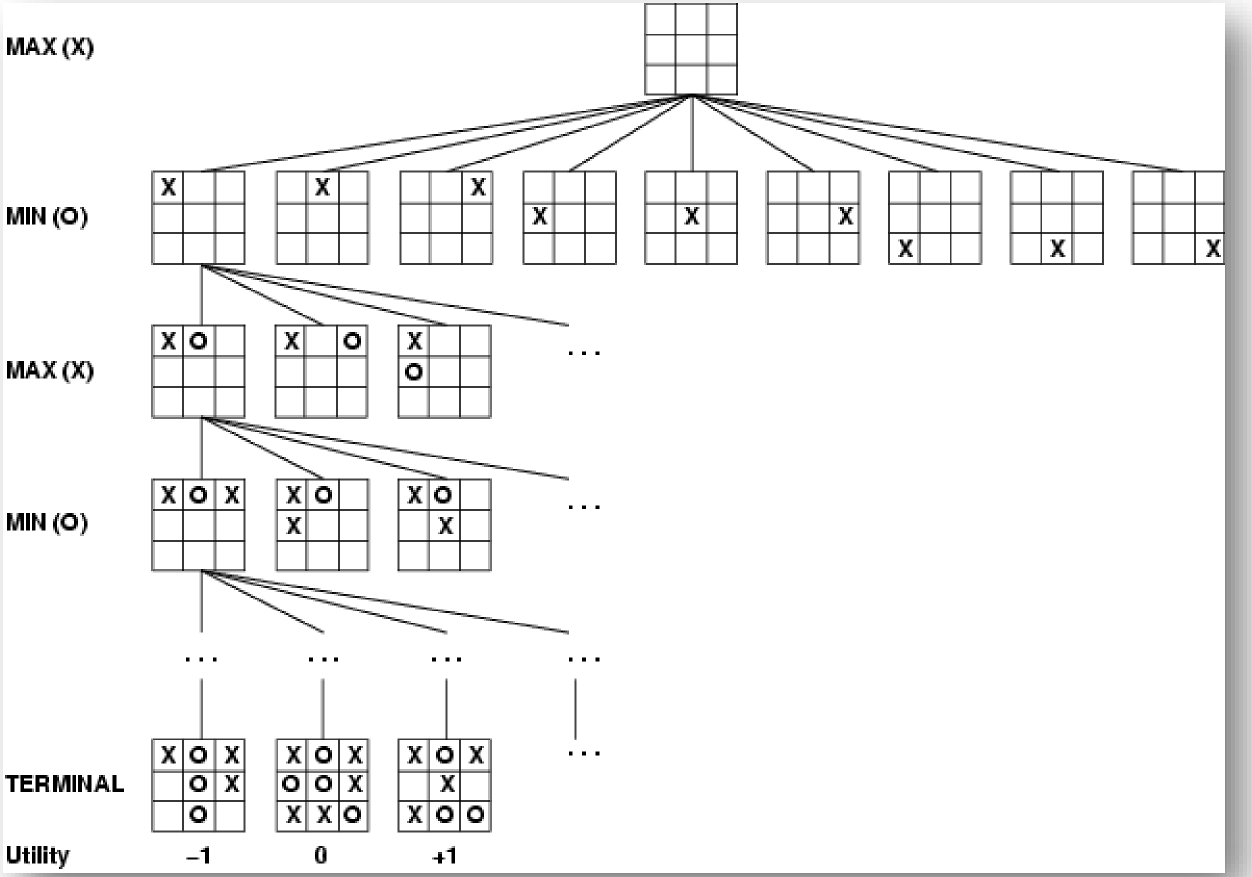
Games vs. Search Problems

We are interested in zero-sum games:

- Deterministic, perfect information
- Agents act alternately
- Utilities at end of game are equal and opposite (adding up to 0)
- This opposition between the agents' utility functions makes the situation is adversarial

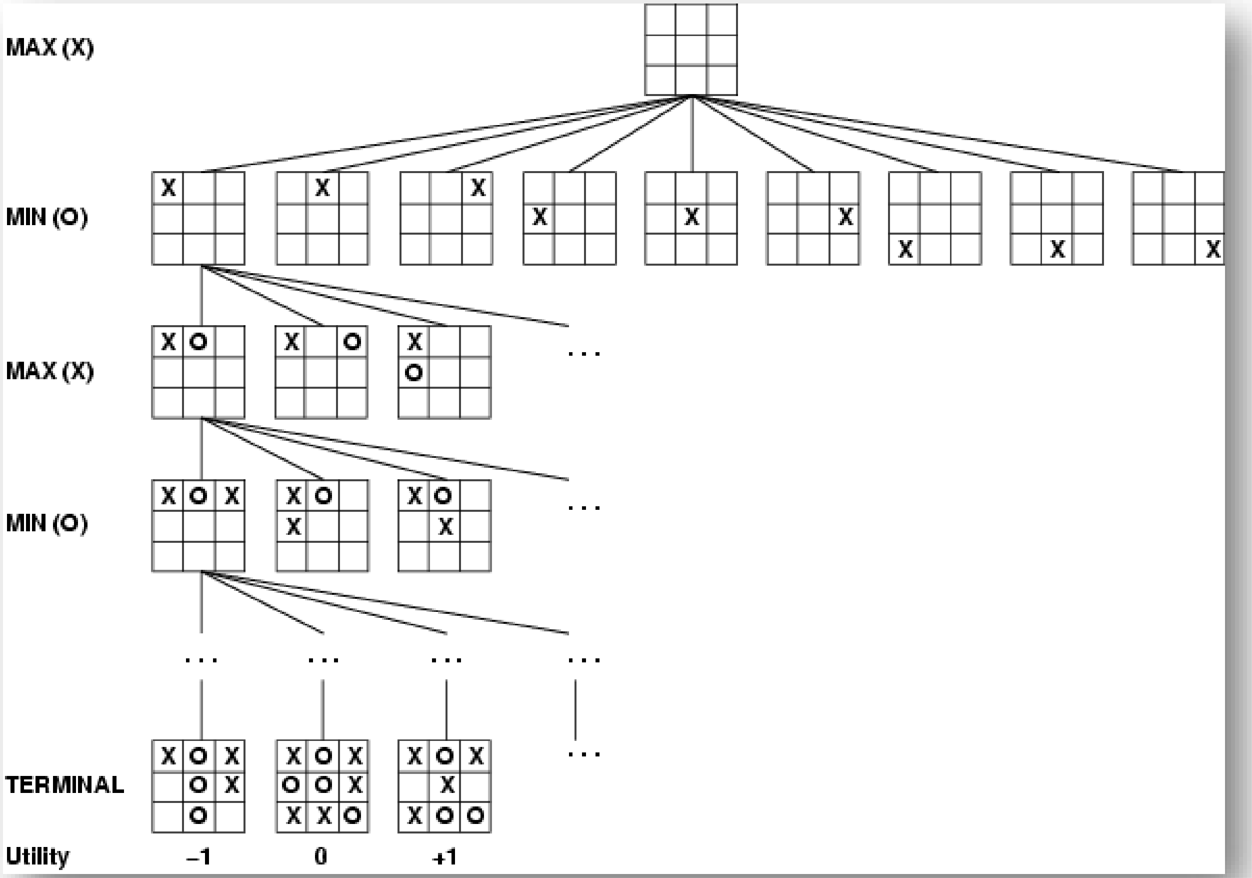
Game Tree for Tic-Tac-Toe (2-player, deterministic, turns)

- 2 players: MAX and MIN
- MAX moves first
- Game tree built from MAX's point of view



Game Tree for Tic-Tac-Toe (2-player, deterministic, turns)

- S_0: the initial state
- *Player(s)*
- *Actions(s)*
- *Result(s,a)*: the transition model
- *Terminal-Test(s)*
- *Utility(s,p)*: a *utility function*



Optimal Decisions

Normal search:

- optimal decision is a *sequence of actions* leading to a goal state (i.e., a solution that satisfies the goal test)

Adversarial search:

MIN has a say in game

- MAX needs to find a contingent strategy which specifies:
  - ➢ MAX's move in initial state then…
  - ➢ MAX's moves in states resulting from every response by MIN to the move then…
  - ➢ MAX's moves in states resulting from every response by MIN to those moves, etc…

Minimax value

minimax value of a node = utility for MAX of being in corresponding state:

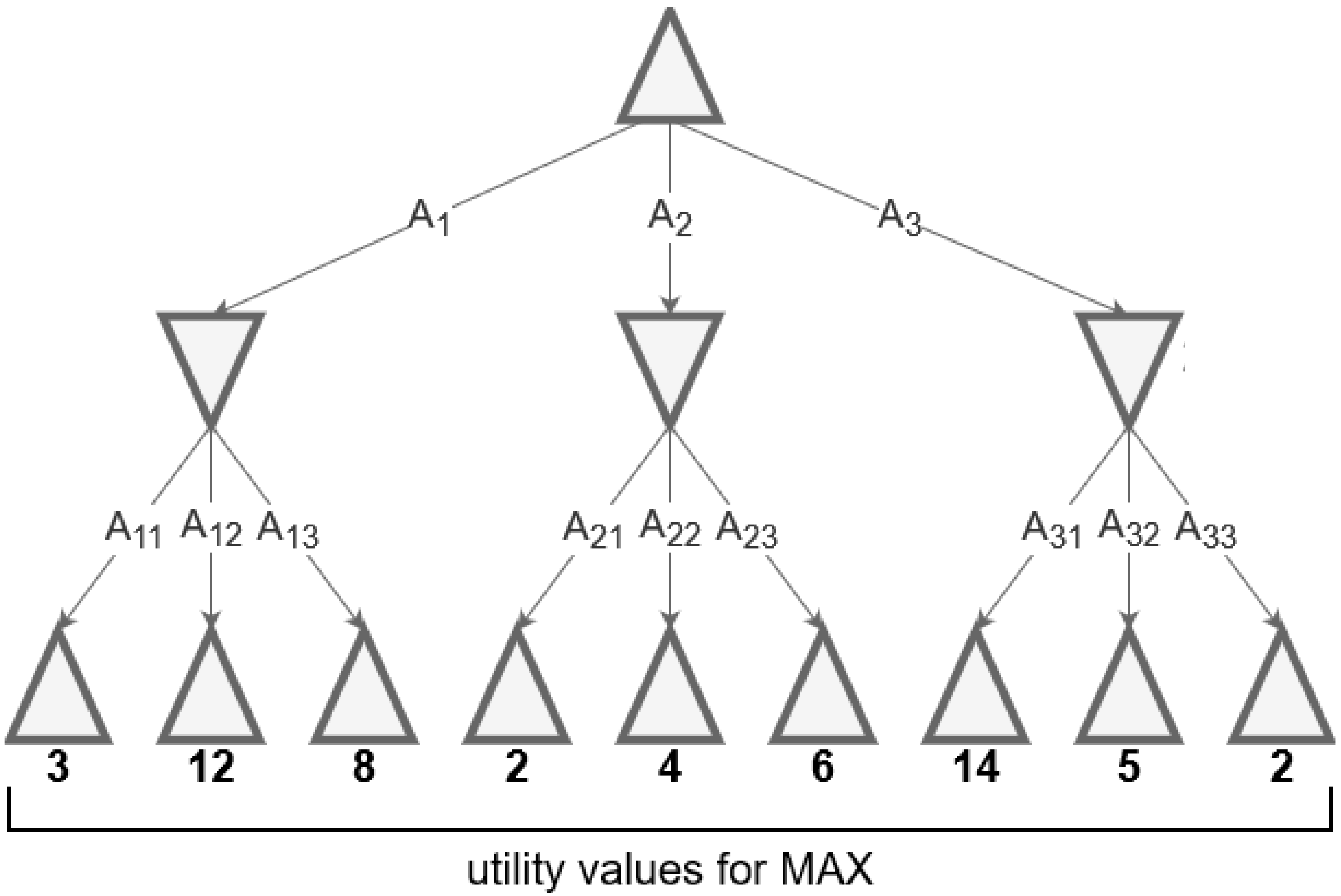
$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$

Minimax

Perfect play for deterministic, perfect-information games

Idea: choose move to position with highest minimax value

= best achievable payoff against best play

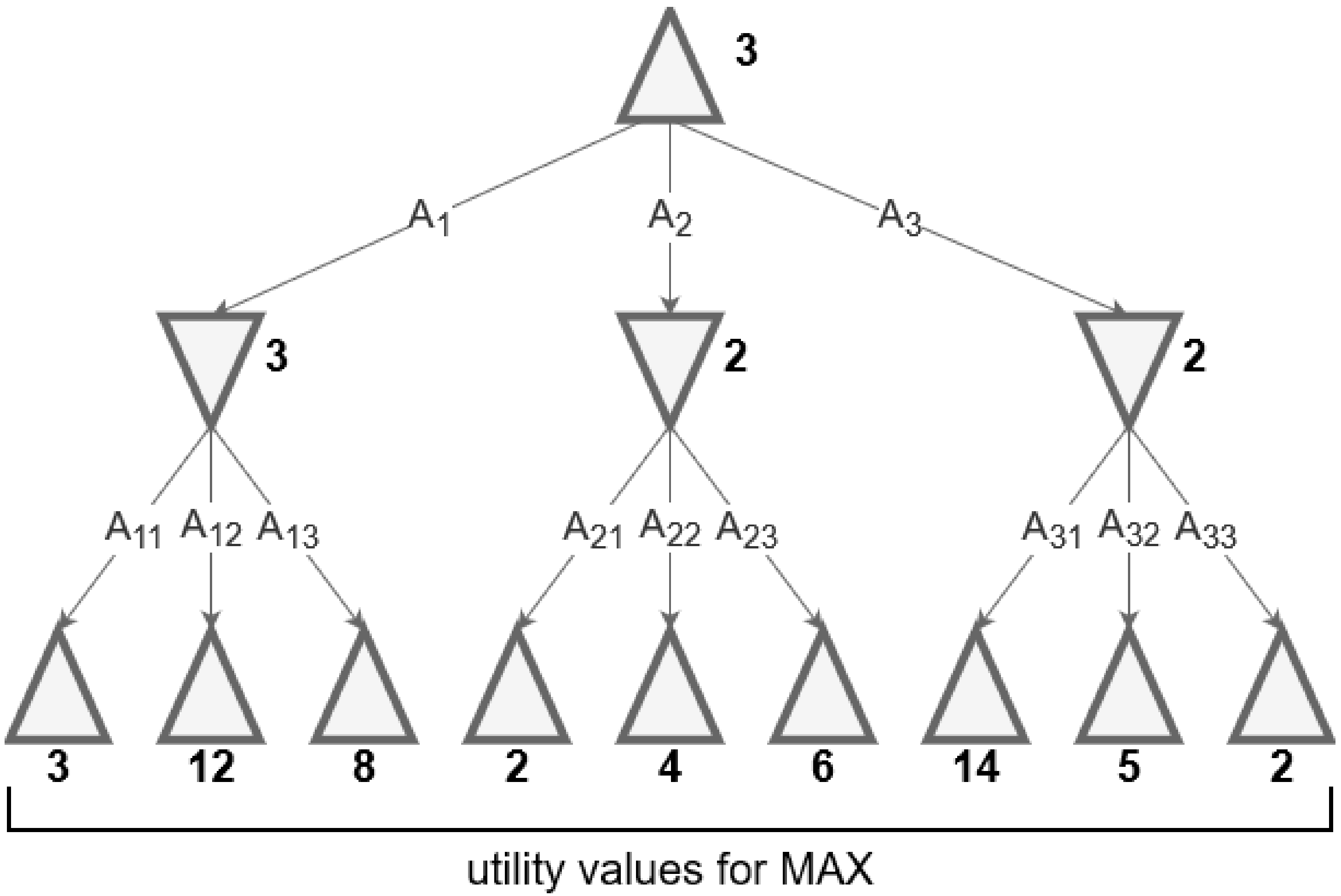


Minimax

Perfect play for deterministic, perfect-information games

Idea: choose move to position with highest minimax value

= best achievable payoff against best play



```
function MINIMAX-DECISION(state) returns an action
  return arg max_{a ∈ ACTIONS(s)} MIN-VALUE(RESULT(state, a))

function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s, a)))
  return v

function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← ∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s, a)))
  return v
```

Minimax algorithm

Idea:

- Proceed all the way down to the leaves of the tree
- then minimax values are backed up through tree

Properties of Minimax

Complete?

Time complexity?

Space complexity?

Optimal?

Properties of Minimax

Complete?
Yes (if tree is finite)

Time complexity?

Space complexity?

Optimal?

Properties of Minimax

Complete?
Yes (if tree is finite)

Time complexity?
$O(b^m)$

Space complexity?

Optimal?

Properties of Minimax

Complete?
Yes (if tree is finite)

Time complexity?
$O(b^m)$

Space complexity?
$O(bm)$

Optimal?

Properties of Minimax

Complete?
Yes (if tree is finite)

Time complexity?
$O(b^m)$

Space complexity?
$O(bm)$

Optimal?
Yes (against an optimal opponent)

Time Complexity

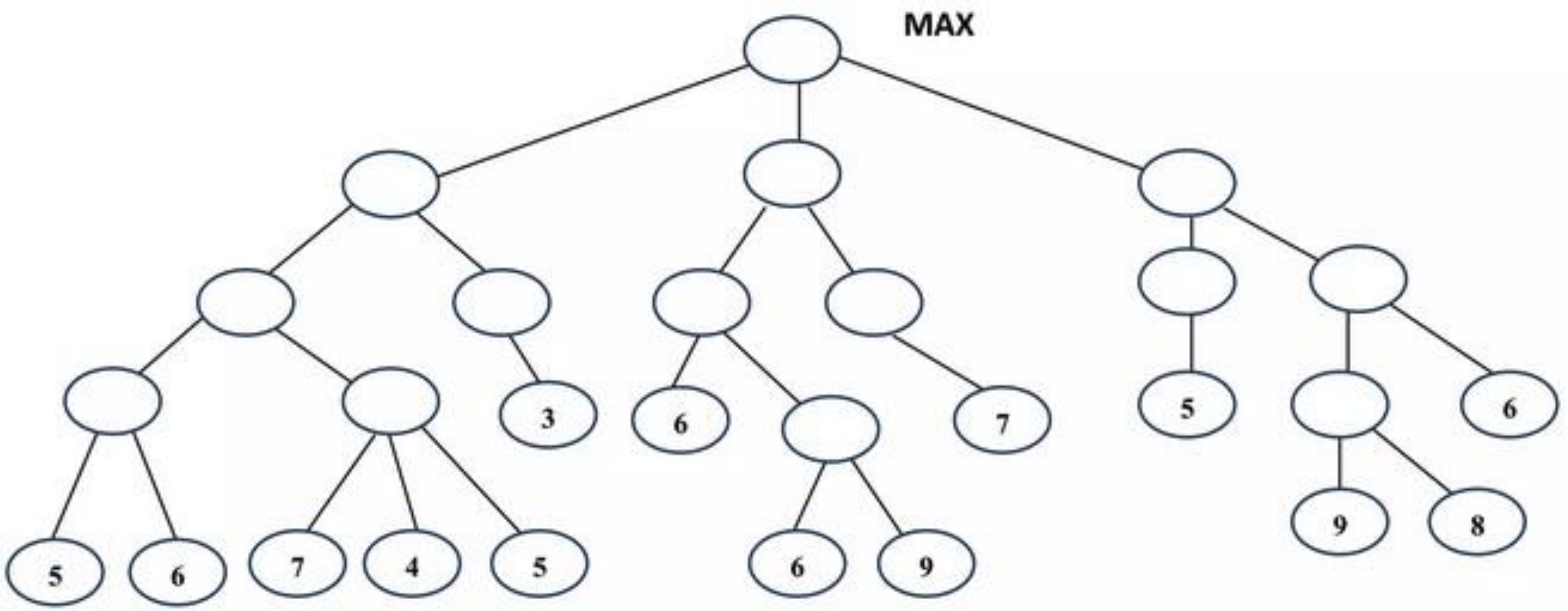
For chess, $b \approx 35$, $m \approx 100$ (average ≈ 40) for "reasonable" games

- exact solution completely infeasible!
- would like to eliminate (large) parts of game tree

$$35^{40} = 5.791 \times 10^{61}$$

$$35^{100} = 2.552 \times 10^{154}$$

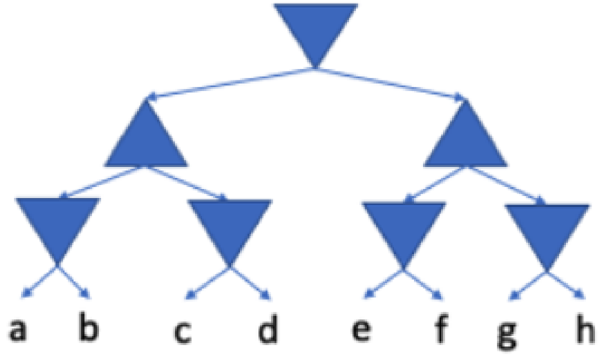
Exercise (Minimax)



https://www.slideshare.net/nishanthysubramaniam90/answer-quiz-minimax

Exercise (Minimax) -- Your turn!

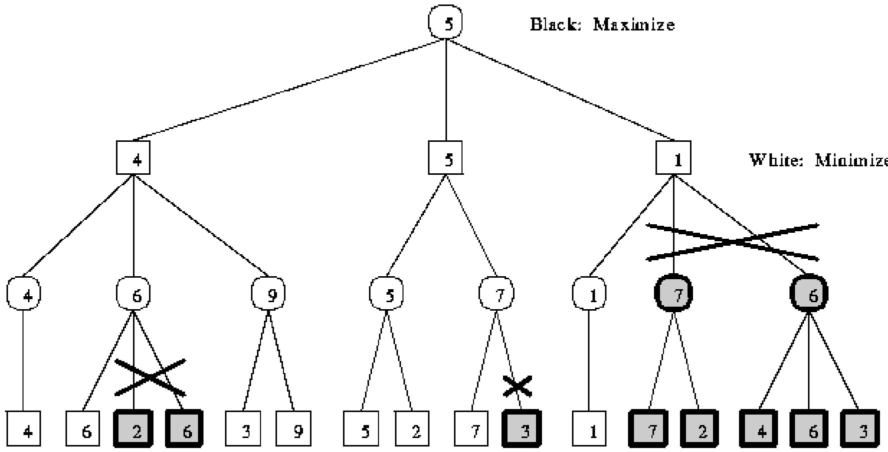
Consider the minimax game tree shown below. Decisions by MAX are represented as upward-pointing triangles; decisions by MIN are represented as downward-pointing triangles; small letters denote outcomes of the game:



The values of each of the outcomes, to the MAX player, are as shown in the following table:

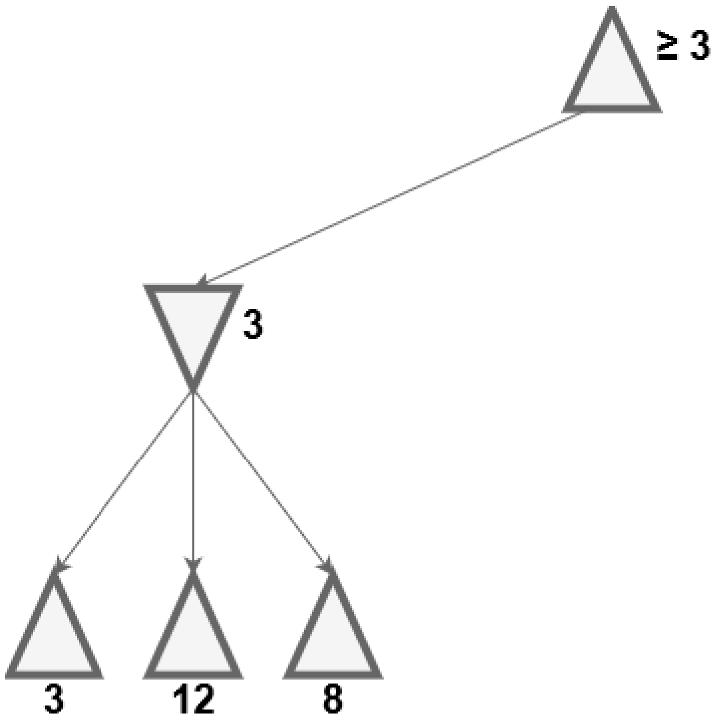
	Outcome							
	a	b	c	d	e	f	g	h
Value to the MAX player:	8	3	1	7	2	5	6	4

http://www.isle.illinois.edu/speech_web_lg/coursematerials/ece448/sp2021/exam3_review.pdf

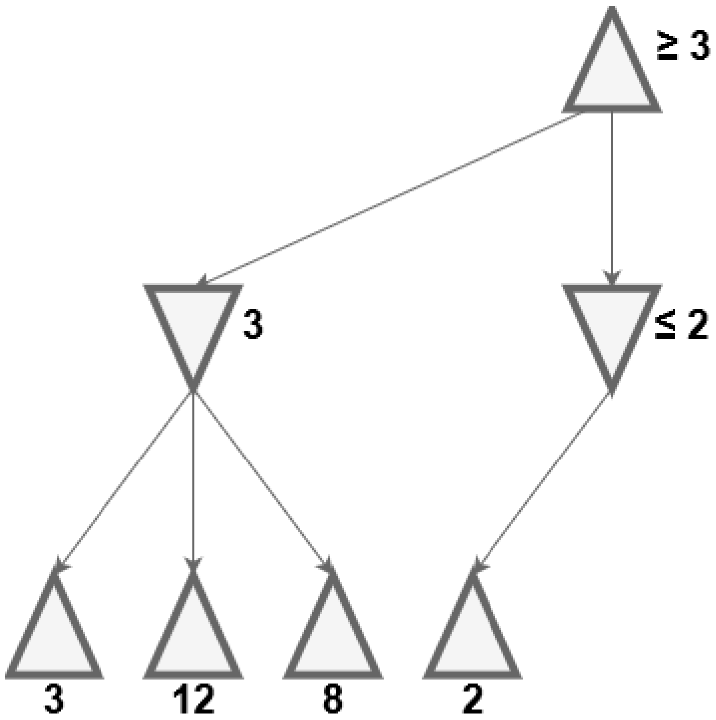


α-β Pruning

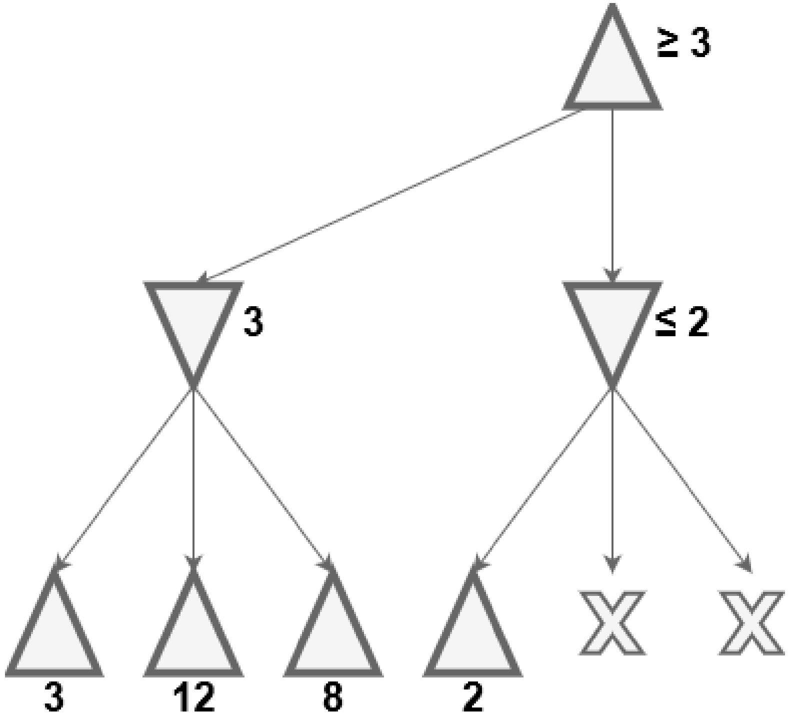
α-β pruning example



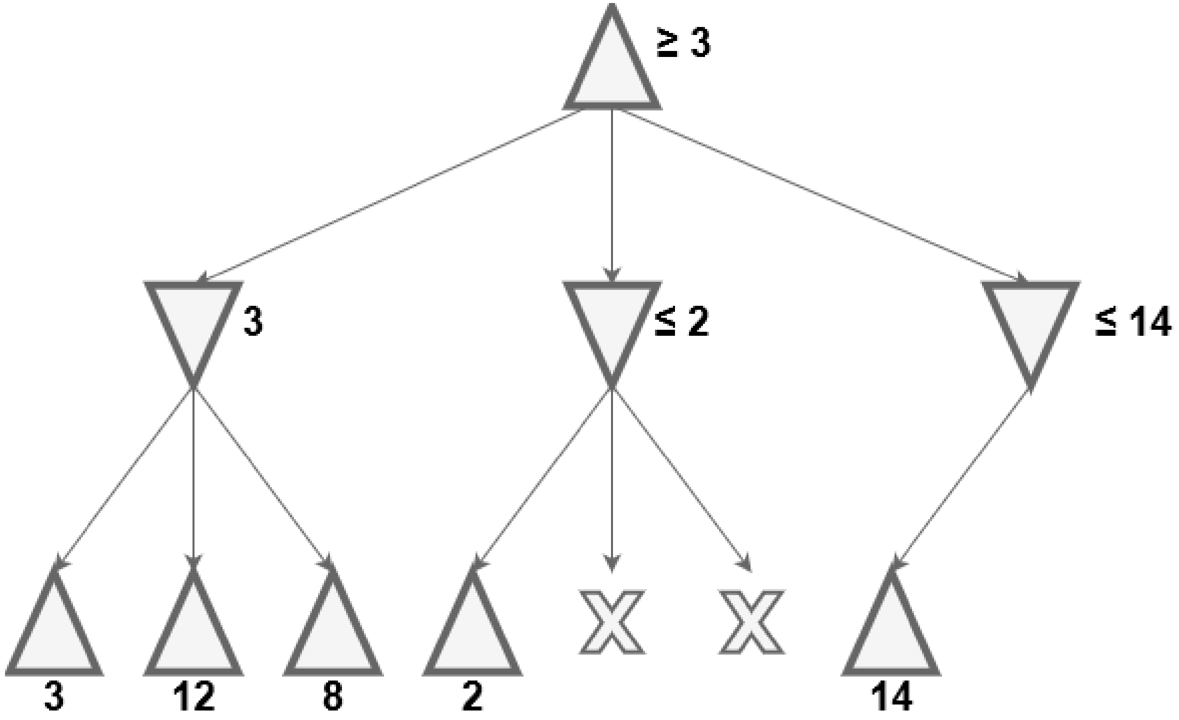
α-β pruning example



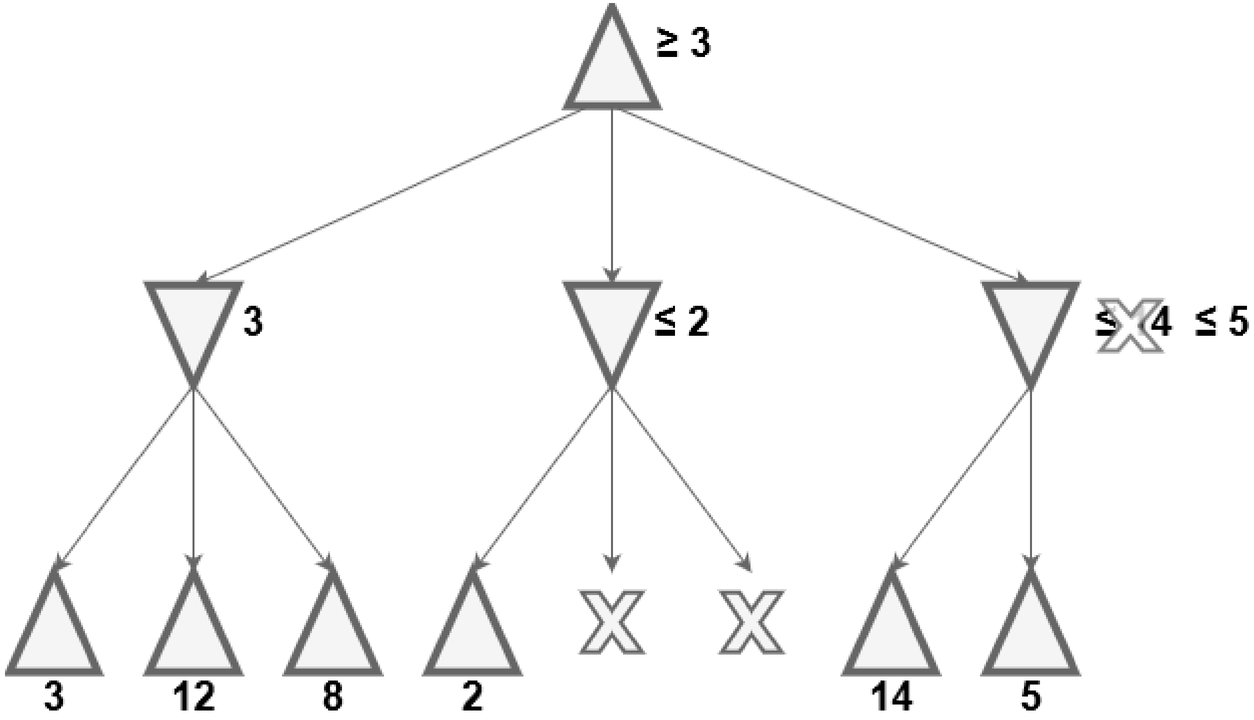
α-β pruning example



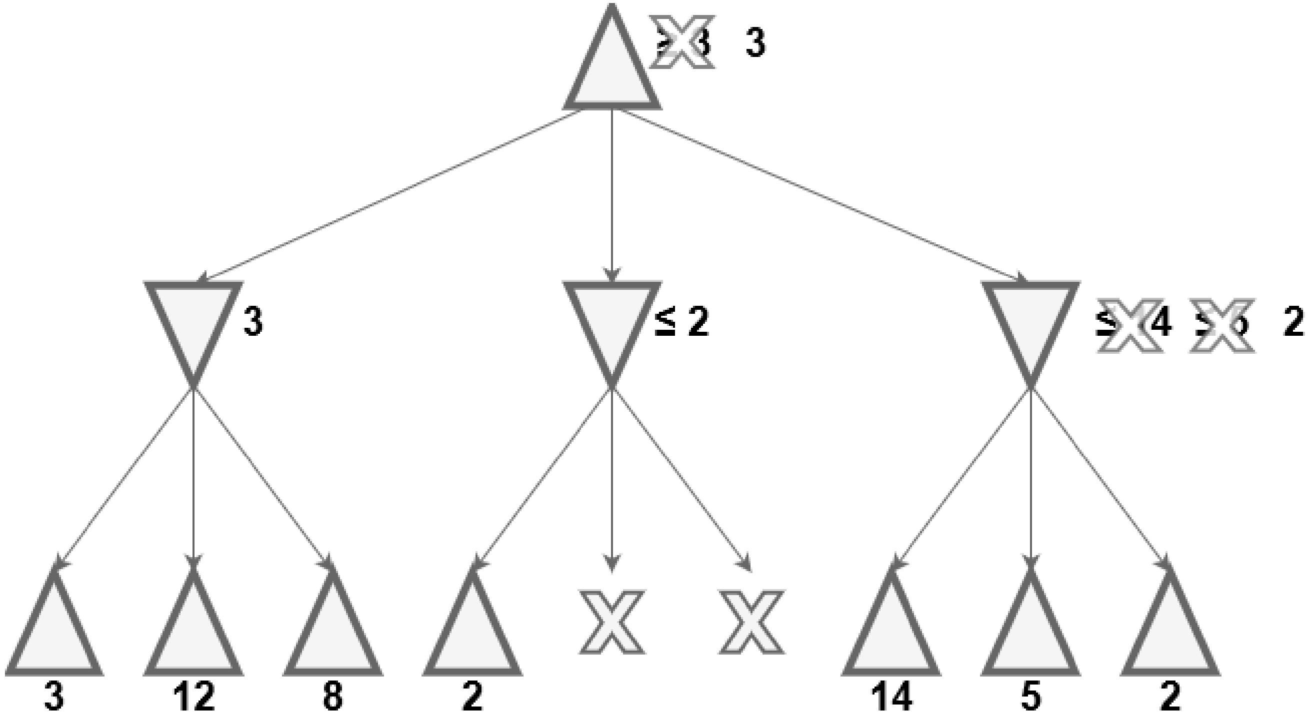
α-β pruning example



α-β pruning example



α-β pruning example



α-β pruning example

Are minimax value of root and, hence, minimax decision independent of pruned leaves?

Let pruned leaves have values u and v,

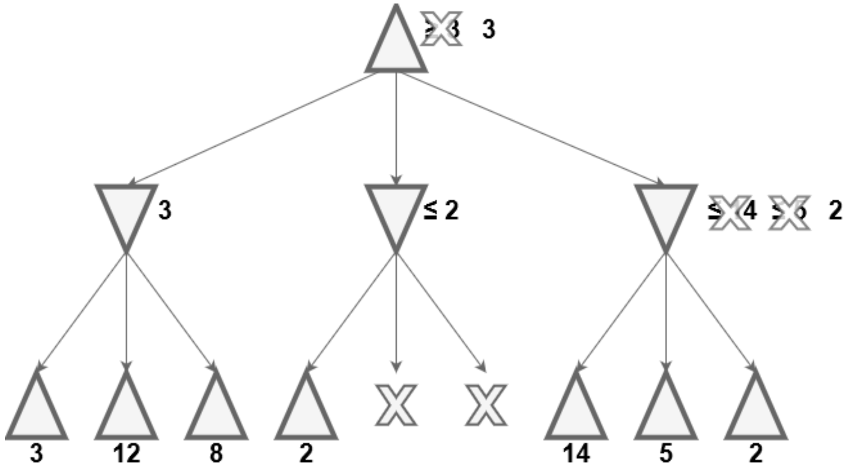
MINIMAX(root)

$= \max(\min(3,12,8), \min(2,u,v), \min(14,5,2))$

$= \max(3, \min(2,u,v), 2)$

$= \max(3, z, 2)$ where $z \leq 2$

$= 3$



α-β pruning example

Are minimax value of root and, hence, minimax decision independent of pruned leaves?

YES!

Let pruned leaves have values u and v,

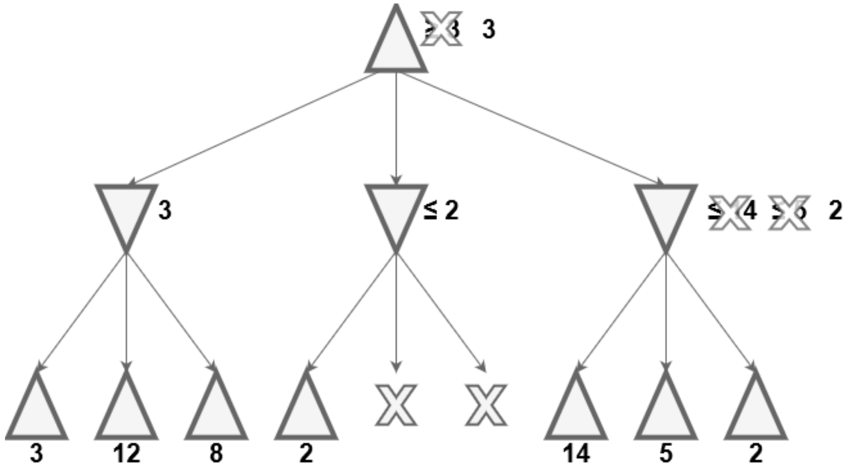
MINIMAX(root)

= max(min(3,12,8), min(2,u,v), min(14,5,2))

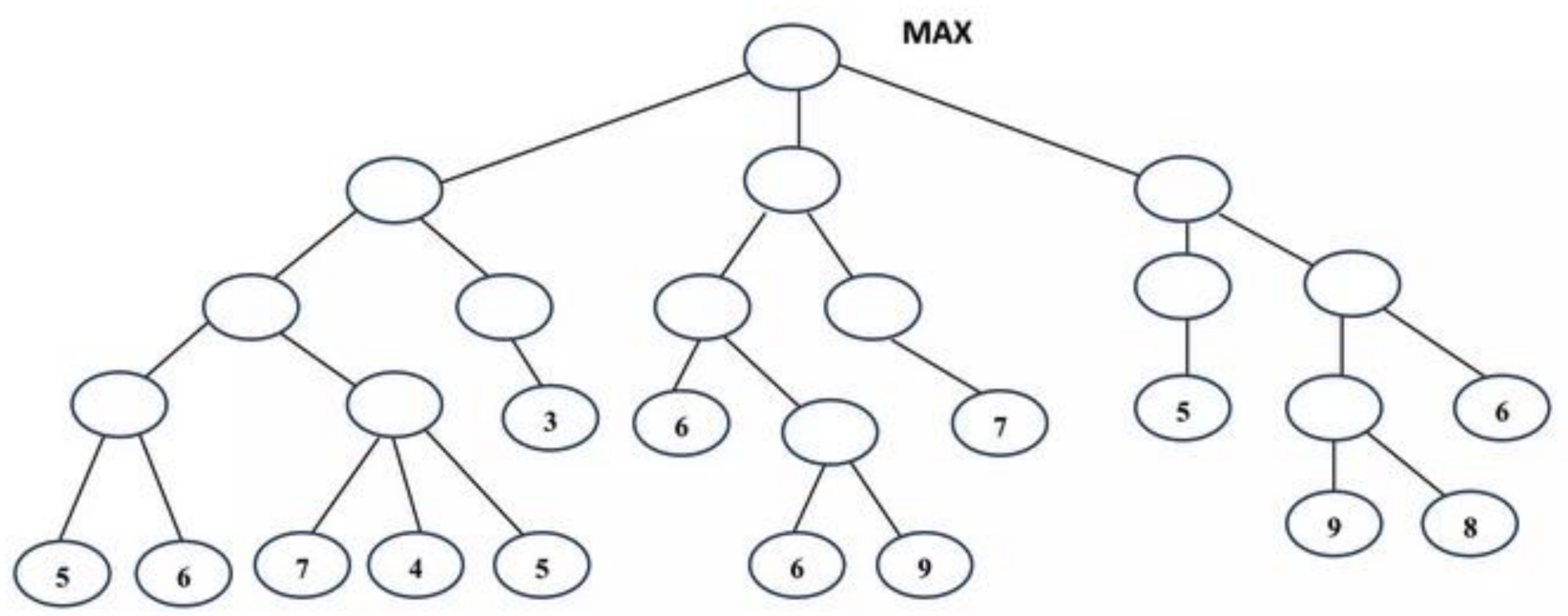
= max(3, min(2,u,v), 2)

= max(3, z, 2) where z ≤ 2

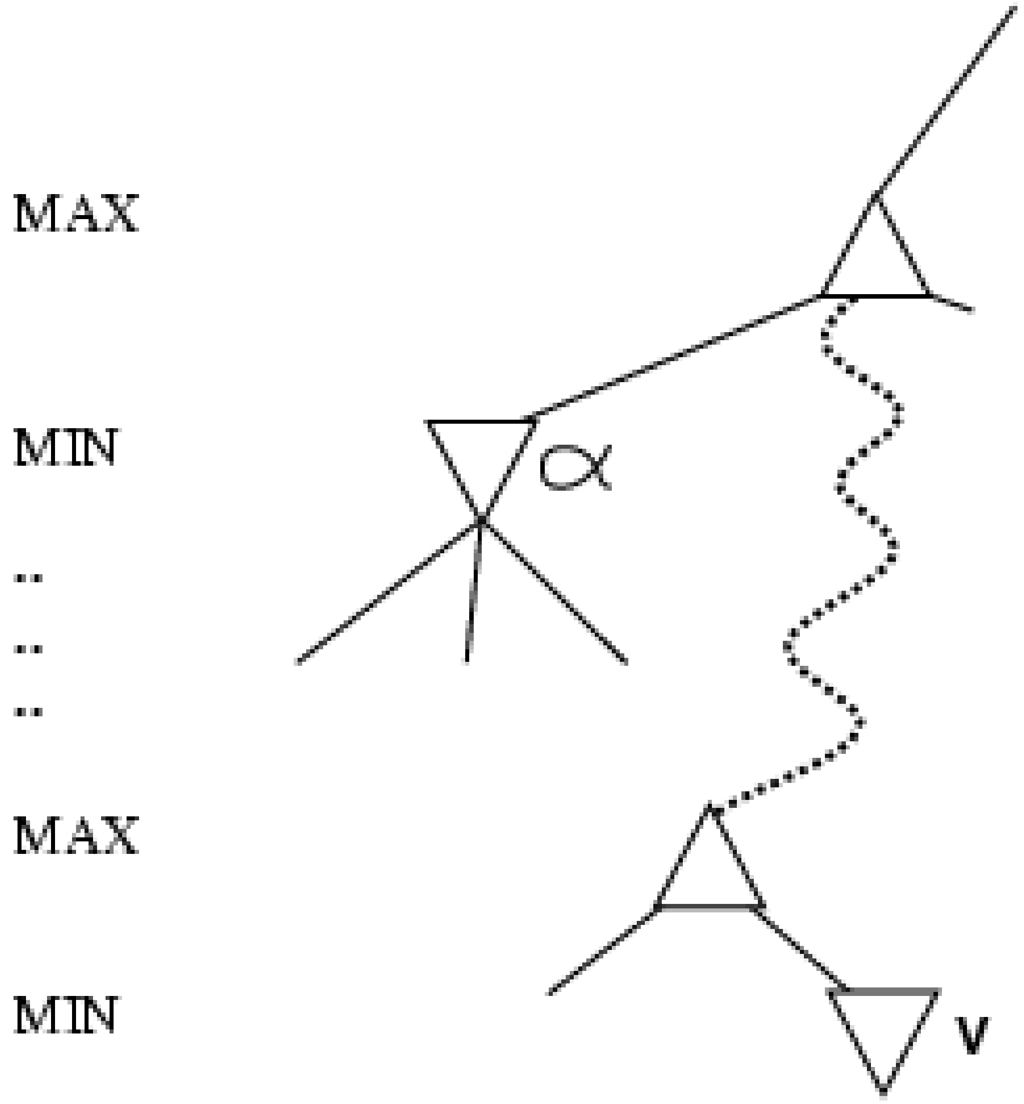
= 3



HW: Exercise
(alpha-beta pruning, left-to-right evaluation)



https://www.slideshare.net/nishanthysubramaniam90/answer-quiz-minimax



Why is it called α-β?

- α is the value of the best (i.e., highest-value) choice found so far at any choice point along the path for MAX
- If v is worse than α, MAX will avoid it
  → prune that branch
- β is defined symmetrically for MIN

The α-β algorithm

- α is value of the best i.e., **highest**-value choice found so far at any choice point along the path for MAX

- β is value of the best i.e., **lowest**-value choice found so far at any choice point along the path for MIN

```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state, −∞, +∞)
  return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v

function MIN-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

Complexity of α-β

Pruning does not affect final result (as we saw for example)

Good move ordering improves effectiveness of pruning

With "perfect ordering", time complexity = $O(b^{m/2})$

- branching factor goes from b to $\sqrt{b}$
- **doubles solvable depth** of search compared to minimax

A simple example of the value of reasoning about which computations are relevant (a form of meta-reasoning)

Resource limits

Suppose we have 100 secs and can explore 10^4 nodes/sec

- 10^6 nodes per move
- $b^m = 10^6$
- For $b = 35 \rightarrow 35^4 = 1.5\text{x}10^6 \rightarrow$ so $m \approx 4$

4-ply lookahead is a hopeless chess player!

- 4-ply ≈ human novice
- 8-ply ≈ typical PC, human master
- 12-ply ≈ Deep Blue, Kasparov

Altering Minimax or Alpha-Beta

- We cannot generate the entire game search space, not practical!
- Cutoff test

  e.g., depth limit (perhaps add quiescence search, which tries to search interesting positions to a greater depth than quiet ones)

- Evaluation function

  = estimated desirability of a position (like what we did for A*)

The *α-β* algorithm

- *α* is value of the best i.e., **highest**-value choice found so far at any choice point along the path for MAX

- *β* is value of the best i.e., **lowest**-value choice found so far at any choice point along the path for MIN

```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state, −∞, +∞)
  return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v

function MIN-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

The *α*-*β* algorithm

Let's cut off the search!

function ALPHA-BETA-SEARCH(*state*) **returns** an action
  $v \leftarrow$ MAX-VALUE(*state*, $-\infty, +\infty$)
  return the *action* in ACTIONS(*state*) with value v

---

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

---

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

The *α-β* algorithm

Let's cut off the search!

- Cutoff-Test returns *true* for:
  - all depth greater than d
  - all terminal states just as Terminal-Test

```
function ALPHA-BETA-SEARCH(state) returns an action
   v ← MAX-VALUE(state, −∞, +∞)
   return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
   if CUTOFF-TEST(state, depth) then return EVAL(state)
   v ← −∞
   for each a in ACTIONS(state) do
      v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
      if v ≥ β then return v
      α ← MAX(α, v)
   return v

function MIN-VALUE(state, α, β) returns a utility value
   if CUTOFF-TEST(state, depth) then return EVAL(state)
   v ← +∞
   for each a in ACTIONS(state) do
      v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
      if v ≤ α then return v
      β ← MIN(β, v)
   return v
```

Evaluation functions

Often a linear weighted sum of features

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

where each w_i is a weight and each f_i is a feature of state s

Chess example

- queen = 1, king = 2, etc.
- f_i = number of pieces of type *i* on board
- w_i = value of the piece of type *i*

Deterministic games in practice

Checkers

Playing checkers on the 701
On February 24, 1956, Arthur Samuel's Checkers program, which was developed for play on the IBM 701, was demonstrated to the public on television. In 1962, self-proclaimed checkers master Robert Nealey played the game on an IBM 7094 computer. The computer won. Other games resulted in losses for the Samuel Checkers program, but it is still considered a milestone for artificial intelligence, and offered the public in the early 1960s an example of the capabilities of an electronic computer.

01/02

https://www.ibm.com/ibm/history/ibm100/us/en/icons/ibm700series/impacts/

Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used a precomputed endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 444 billion positions.

http://jonathanschaeffer.blogspot.com/2012/08/chinook-twenty-years-later.html

Chess

Deep Blue defeated human world champion Garry 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.

https://en.wikipedia.org/wiki/Deep_Blue_(chess_computer)

Modern Chess

Stockfish

- Uses and advanced version of α-β pruning among other algorithms.
- Recently added a simple neural network in its evaluation.
  - Improved by 100+ Elo points since.
- Analyses 10^8 positions per second (half when using the neural network).

AlphaZero (successor of AlphaGo Zero)

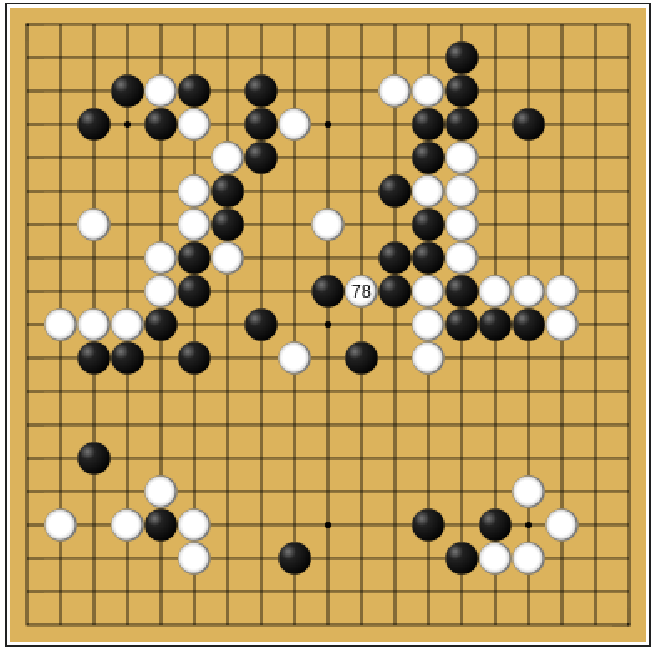
- Based on Monte Carlo tree search, deep neural networks and self-play.
- Analyses 80,000 positions per second.
- Defeated Stockfish with 28W-72D-0L in 2016.

Leela Zero

- Released 2017 with ideas from AlphaGo Zero's paper.
- Believed to have surpassed AlphaZero.
- Neck to neck with modern Stockfish, losing narrowly to it in the last 3 TCEC (Top Chess Engine Championship) super finals.

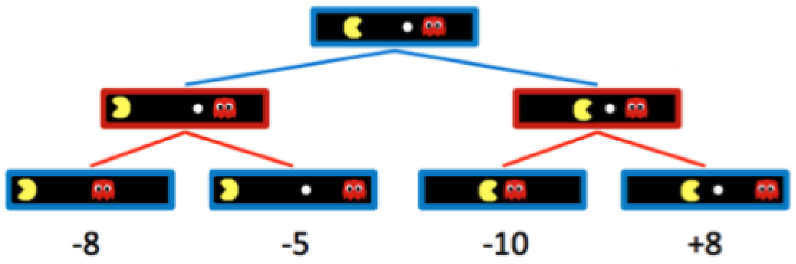
Go

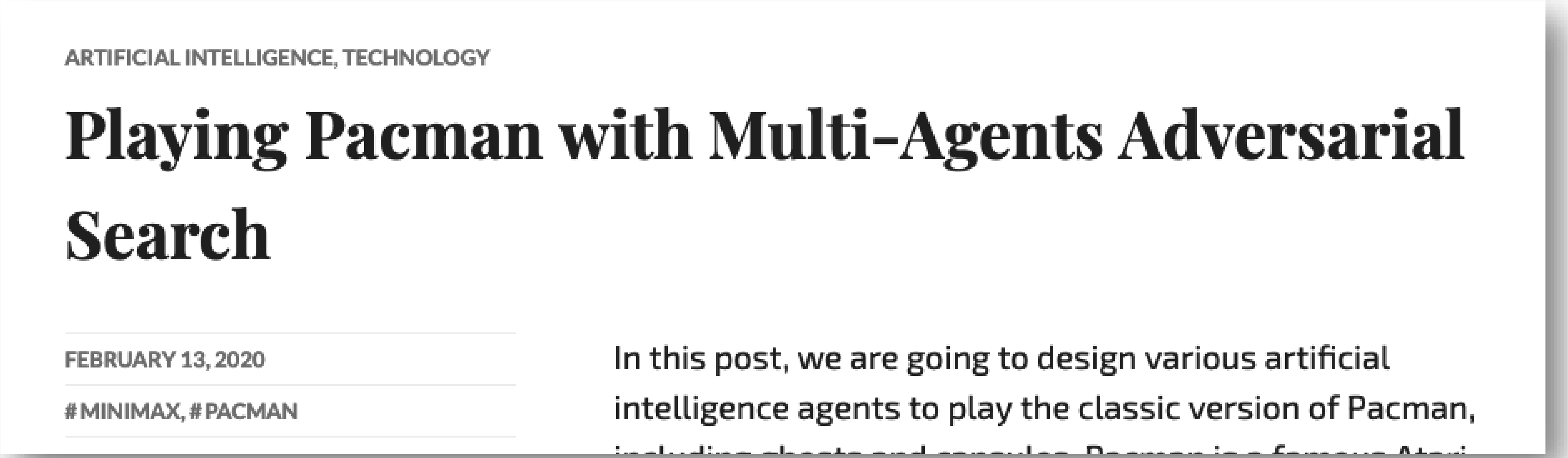
- In Go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.
- In 2015 AlphaGo became the first computer program to beat a human professional Go player (Fan Hui) without handicap.
- In 2016 AlphaGo beat world's #2 Lee Sedol 4-1.
- Evolved into AlphaGo Zero (without human datasets), then AlphaZero, and more recently MuZero (model-free) .



Game 4, Lee Sedol (white) v. AlphaGo (black).
First 78 moves

https://en.wikipedia.org/wiki/Lee_Sedol




ARTIFICIAL INTELLIGENCE, TECHNOLOGY

Playing Pacman with Multi-Agents Adversarial Search

FEBRUARY 13, 2020

#MINIMAX, #PACMAN

In this post, we are going to design various artificial intelligence agents to play the classic version of Pacman,

https://davideliu.com/2020/02/13/playing-pacman-with-multi-agents-adversarial-search/

Summary

- Games are fun to work on!
- They illustrate several important points about AI.
- Perfection is unattainable → must approximate!
- Good idea to think about what to think about (meta-reasoning)
- Modern AI demonstrating superhuman performance.