دانشکده ریاضی و آمار

# هوش مصنوعی

UnderGrad.Q5/7: Artificial Intelligence


فریا نصیری مفخّم

Faria Nassiri-Mofakham

https://eng.ui.ac.ir/**fnasiri**


4016282-01

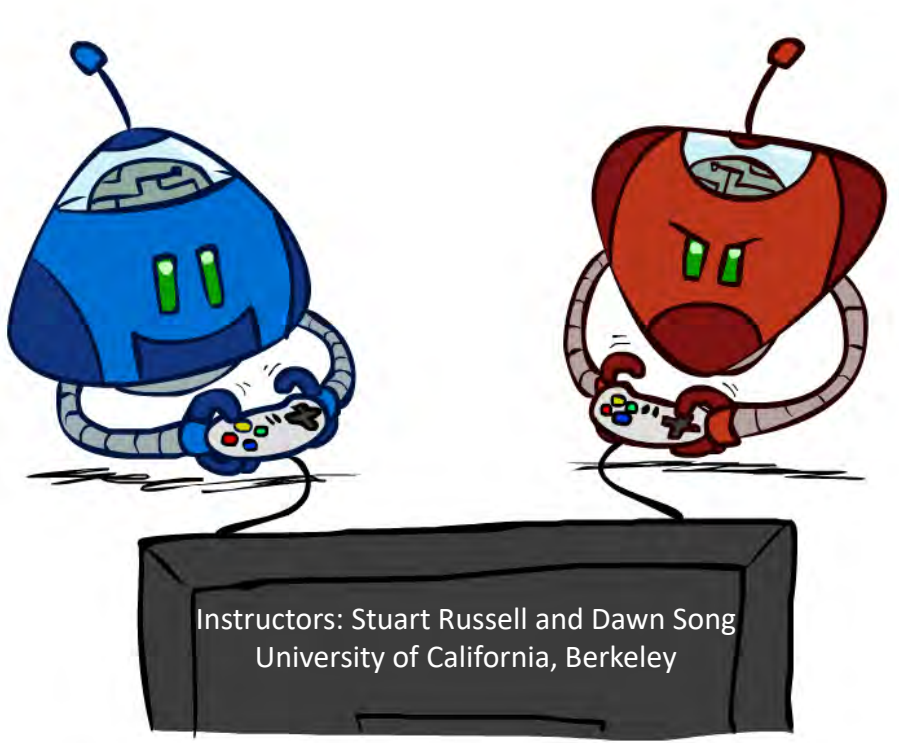
یکشنبه‌ها ۱۸–۱۷–۱۶، سه‌شنبه‌ها ۱۰–۸

lms.ui.ac.ir, ۳۶۰–۷

نیمسال ۴۰۴۲

# CS 188: Artificial Intelligence

## Adversarial Search



Instructors: Stuart Russell and Dawn Song
University of California, Berkeley

# Game Theory

- تقسیم‌بندی بازی‌ها
  - ایستایی یا پویایی
  - تعارض منافع یا امکان تشریک مساعی و همکاری
  - تعداد دفعات انجام بازی
  - اطلاعات

- **Complete** vs **asymmetric information** vs **incomplete information**
- **Perfect** vs **imperfect information**

  - ثابت یا متغیر بودن قواعد بازی
  - همکارانه یا غیرهمکارانه بودن بازی

# نمایش بازی

- بازی ایستا با اطلاعات کامل: بازی که در آن بازیکنان به طور همزمان انتخاب خود را انجام داده و پیامد (outcome) بازی برای هر ترکیب انتخاب آن‌ها برای همه بازیکنان به صورت اطلاعات (آگاهی) عمومی شده باشد (common knowledge)

- Complete vs incomplete information
- Perfect vs imperfect information

- نمایش بازی:
  - فرم استراتژیک (نرمال و عادی)
  - فرم بسط‌یافته (نمایش درختی)

# تعریف رسمی بازی

- مجموعه بازیکنان N={1,2,…n}
- استراتژی $S_i = \{s_1^i, s_2^i, \dots, s_m^i\}, i \in N$
- $S = S_1 \times S_2 \times \cdots \times S_n$
- G=<S,u>
- $u_i : S \to R$ سودمندی یا مطلوبیت برای بازیکن i

# مثال

- حوزه نفتی با 4 میلیون بشکه نفت
- شرکت A این حوزه را برای دو سال اجاره کرده است.
- قیمت جهانی هر بشکه نفت طی دو سال ثابت و 20 دلار است.
- دو استراتژی برای استخراج نفت: حفر چاه باریک (Narrow) و حفر چاه عریض (Wide)
- چاه باریک: استخراج 2 میلیون بشکه در سال و هزینه ثابت 16 میلیون دلار
- چاه عریض: استخراج 6 میلیون بشکه در سال و هزینه ثابت 29 میلیون دلار
- هزینه پمپاژ: 5 دلار برای هر بشکه

- سود شرکت A (میلیون دلار):

| سود/هزینه | چاه باریک (N) | چاه عریض (W) |
|---|---|---|
| هزینه زدن چاه | 16 | 29 |
| هزینه کل پمپاژ | 20 | 20 |
| هزینه کل | 36 | 49 |
| درآمد | 80 | 80 |
| سود | 44 | 31 |

- شرکت A و B همزمان از دو دشت مجاور، این حوزه نفتی را برای دو سال اجاره کرده‌اند (مثلا حوزه نفتی بین ایران و کشورهای عربی)
- اگر مقدار استخراج شرکت A را با $x_A$ و برای شرکت B با $x_B$ نشان دهیم

1) اگر هر دو حفر چاه باریک را انتخاب کنند،

- $\begin{cases} x_A + x_B = 4 \\ x_A = x_B \end{cases}$ -> $x_A = \frac{4}{2} = 2, x_B = 2$ میلیون بشکه

به همین ترتیب برای دیگر استراتژی‌ها...

2) $x_A = x_B = 2$

3) $x_A = 3, x_B = 1$

1) سود شرکت A و B در حالت چاه باریک

| سود/هزینه | شرکت (N) A | شرکت (N) B |
|---|---|---|
| هزینه زدن چاه | 16 | 16 |
| هزینه کل پمپاژ | 10 | 10 |
| هزینه کل | 26 | 26 |
| درآمد | 40 | 40 |
| سود | 14 | 14 |

2) سود شرکت A و B در حالت چاه عریض

| سود/هزینه | شرکت (W) A | شرکت (W) B |
|---|---|---|
| هزینه زدن چاه | 29 | 29 |
| هزینه کل پمپاژ | 10 | 10 |
| هزینه کل | 39 | 39 |
| درآمد | 40 | 40 |
| سود | 1 | 1 |

3) سود شرکت A و B در حالت انتخاب چاه‌های مختلف

| سود/هزینه | شرکت (W) A | شرکت (N) B |
|---|---|---|
| هزینه زدن چاه | 29 | 16 |
| هزینه کل پمپاژ | 15 | 5 |
| هزینه کل | 44 | 21 |
| درآمد | 60 | 20 |
| سود | 16 | -1 |

- سود و پیامد هر شرکت به استراتژی خود و شرکت دیگر بستگی دارد
- **سود** هر شرکت در تمام حالات برای یکدیگر **معلوم** است
- **استراتژی** که یک شرکت انتخاب خواهد کرد برای شرکت دیگر (حریف) **نامعلوم** است

- فرم استراتژیک (ماتریسی) بازی شرکت A و B

| | | شرکت B | |
|---|---|---|---|
| | | N | W |
| شرکت A | N | 14,14 | -1,16 |
| | W | 16,-1 | 1,1 |

- بازی با **جمع غیرصفر** (Non-zero-sum game): برد یک نفر به منزله باخت طرف دیگر نیست. فقط استراتژی‌های آنها روی سود همدیگر اثر می‌گذارد

# بازی معمای دو زندانی
# Prisoner's Dilemma

در اقتصاد، سیاست، روابط بین‌الملل، تجارت، مسائل نظامی، ...

- M: ساکت باشد (یعنی گویا به جرم خودش اعتراف کرده)
- F: گردن مظنون دیگر بیندازد (به جرم نفر دیگر اعتراف کند)
  - -1,-1: هر کسی جرم خود را اعتراف کند و هر دو 1 سال به زندان بروند
  - -6,-6: هر کسی به جرم دیگری اعتراف کند و هر دو 6 سال به زندان بروند
  - ...

| مظنون 1 | | مظنون 2: M | مظنون 2: F |
|---|---|---|---|
| | M | -1,-1 | -9,0 |
| | F | 0,-9 | -6,-6 |

- عملا بازیکنان "باهوش" به -6,-6 می رسند!! (نتیجه‌ی رفتار محافظه‌کارانه ناشی از تعریف حریصانه و رقابت‌جویانه!)
  - نقطه تعادل نش (**Nash** Equilibrium)
  - John Forbes Nash 1950s, (A Beautiful Mind) Nobel Prize 1998
- در حالی‌که نقطه‌ی بهینه پارتو، (M,M) بود که نتیجه -1,-1 می‌داد: در نقطه **Pareto** جمع سودها بیشینه است
- فرم کلی بازی‌های به شکل معمای زندانی

$$\alpha > \beta > \gamma > \theta, \qquad \frac{\alpha + \theta}{2} < \beta$$

| بازیکن 1 | | بازیکن 2: $S_1$ | بازیکن 2: $S_2$ |
|---|---|---|---|
| | $S_1$ | $\beta, \beta$ | $\theta, \alpha$ |
| | $S_2$ | $\alpha, \theta$ | $\gamma, \gamma$ |

مرجع: https://plato.stanford.edu/entries/game-evolutionary/

- Game of Fiscal & Monetary Policies
- ...
- Battle of the Sexes
- ...
- بازی دو جوان (Chicken Game): بر سر چیزی دوئل می‌کنند James 1950s Dean ... زنده ماندن در این بازی هم سود منفی دارد
- Investment Game
- ...

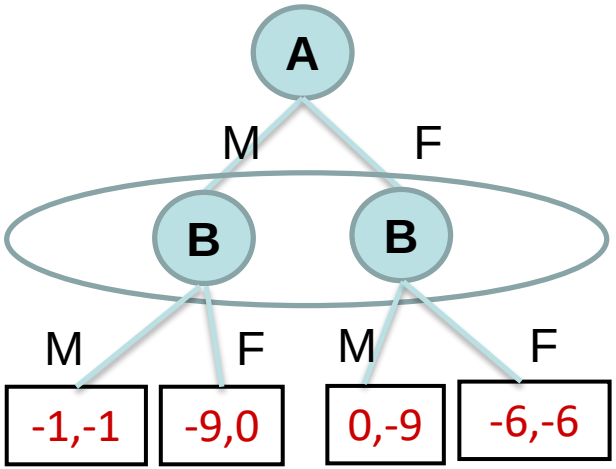
- **One Step Beyond Nash Equilibrium**
  - F Nassiri-Mofakham, MA Nematbakhsh, A Baraani-Dastjerdi, and Nasser Ghasem-Aghaee
  - WORLDCOMP'08: The 2008 International Conference on Scientific Computations (CSC'08), CSREA CSC, pp 186-192, Las Vegas - USA (2008)

# استراتژی غالب و فرم بسط یافته (درختی) بازی

| مظنون 1 \ مظنون 2 | M | F |
|---|---|---|
| M | -1,-1 | -9,0 |
| F | 0,-9 | -6,-6 |

- استراتژی غالب (Dominant Strategy):
  آیا سطری (یا ستونی) داریم که همیشه به همه سطرها (یا همه ستونها) غالب باشد؟

- فرم درختی بازی معمای دو زندانی



# بازی با جمع صفر (Zero-Sum Game)

- بازی‌های کاملا رقابتی و روش Max-Min: در این بازی هر رقیبی می‌خواهد سود خود را بیشینه (Max) و سود دیگری را کمینه (Min) کند.
  - James Waldegrave, 1713

بازیکن 2 (Min)

| بازیکن 1 (Max) | E | F | G | |
|---|---|---|---|---|
| A | 2 | 5 | 13 | Min=2 using E |
| B | 6 | 5.6 | 10.5 | Min=5.6 using F |
| C | 6 | 4.5 | 1 | Min=1 using G |
| D | 10 | 3 | -2 | Min=-2 using G |
| | Max=10 using D | Max=5.6 using B | Max=13 using A | |

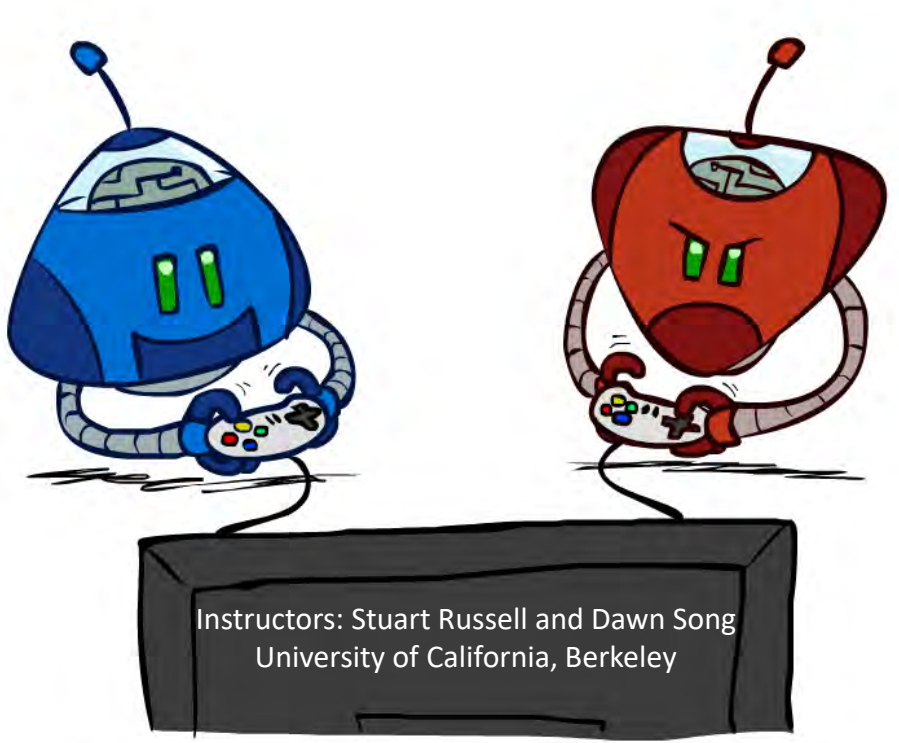
Max(min)=5.6 using F

Min(max)=5.6 using B

- تعادل نش این بازی رقابتی توسط پروفایل استراتژی (B,F) و سود 5.6 برای بازیکن 1 است.
  - John Von Neumann, Oskar Morgenstern, …
- **Evolutionary Game Theory**

# CS 188: Artificial Intelligence

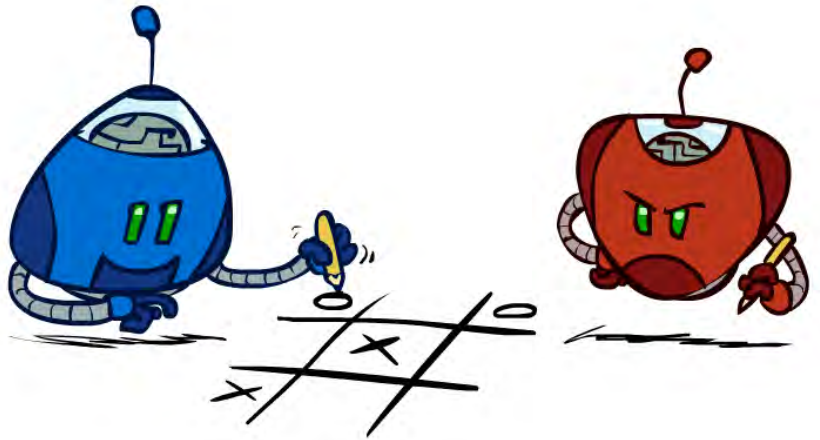
## Adversarial Search



Instructors: Stuart Russell and Dawn Song
University of California, Berkeley

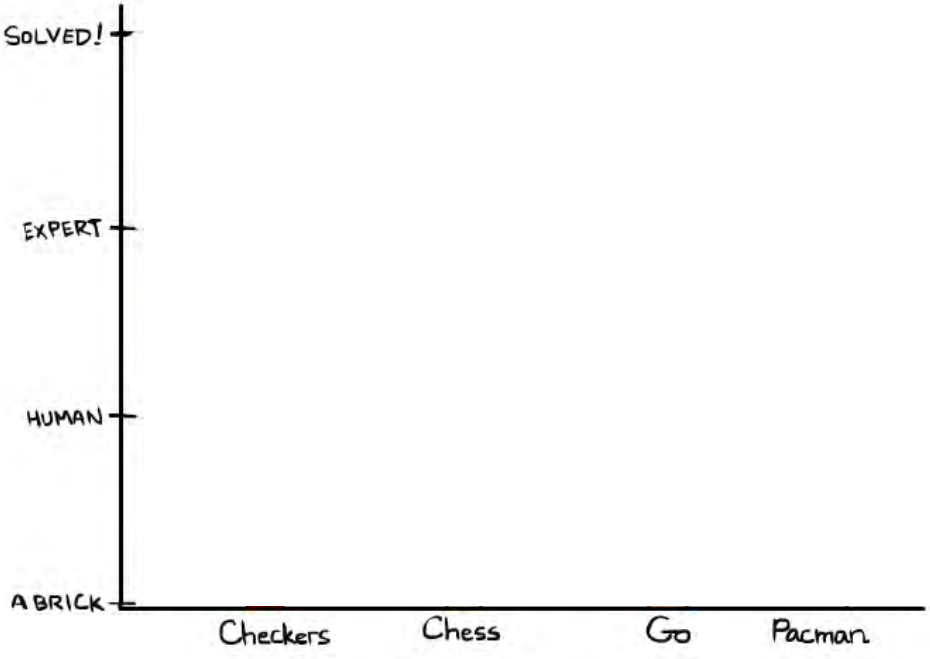
# Outline

- History / Overview
- Minimax for Zero-Sum Games
- α-β Pruning
- Finite lookahead and evaluation



# A brief history

- **Checkers:**
  - 1950: First computer player.
  - 1959: Samuel's self-taught program.
  - 1994: First computer world champion: Chinook defeats Tinsley
  - 2007: Checkers solved! Endgame database of 39 trillion states
- **Chess:**
  - 1945-1960: Zuse, Wiener, Shannon, Turing, Newell & Simon, McCarthy.
  - 1960s onward: gradual improvement under "standard model"
  - 1997: Deep Blue defeats human champion Gary Kasparov
  - 2021: Stockfish rating 3551 (vs 2870 for Magnus Carlsen).
- **Go:**
  - 1968: Zobrist's program plays legal Go, barely (b>300!)
  - 1968-2005: various ad hoc approaches tried, novice level
  - 2005-2014: Monte Carlo tree search -> strong amateur
  - 2016-2017: AlphaGo defeats human world champions
- **Pacman**



# Types of Games

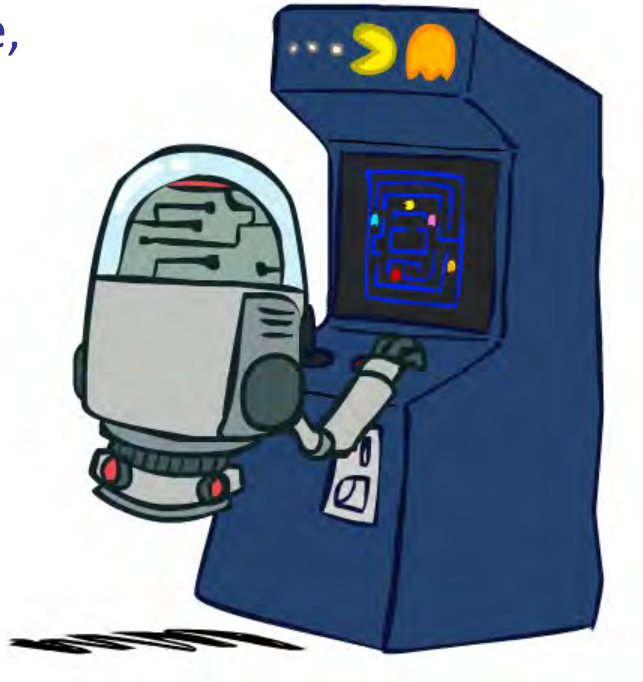
- Game = task environment with > 1 agent

- Axes:
  - Deterministic or stochastic?
  - Perfect information (fully observable)?
  - One, two, or more players?
  - Turn-taking or simultaneous?
  - Zero sum?

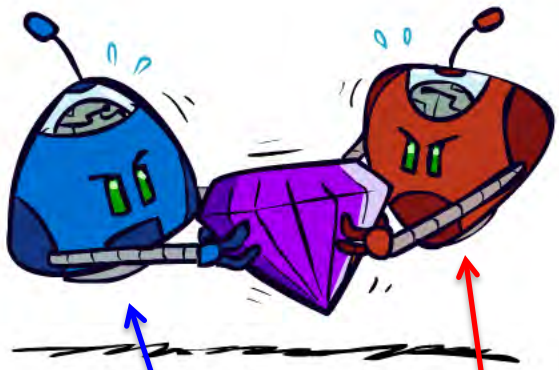
- Want algorithms for calculating a ***contingent plan*** (a.k.a. strategy or policy) which recommends a move for every possible eventuality

# “Standard” Games

- Standard games are deterministic, observable, two-player, turn-taking, zero-sum
- Game formulation:
  - Initial state: $s_0$
  - Players: Player(s) indicates whose move it is
  - Actions: Actions(s) for player on move
  - Transition model: Result(s,a)
  - Terminal test: Terminal-Test(s)
  - Terminal values: Utility(s,p) for player p
    - Or just Utility(s) for player making the decision at root



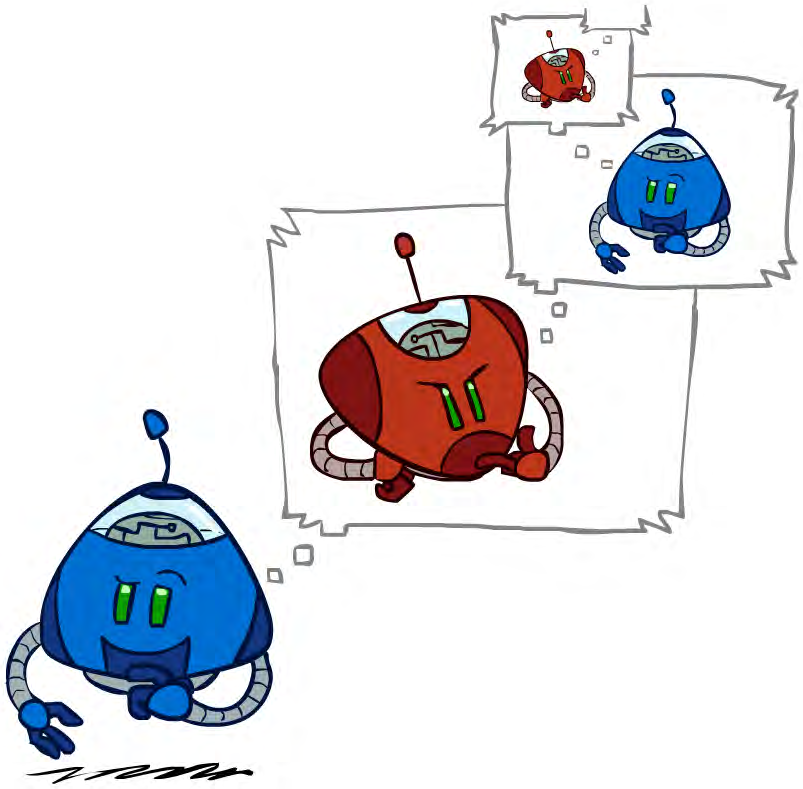
# Zero-Sum Games



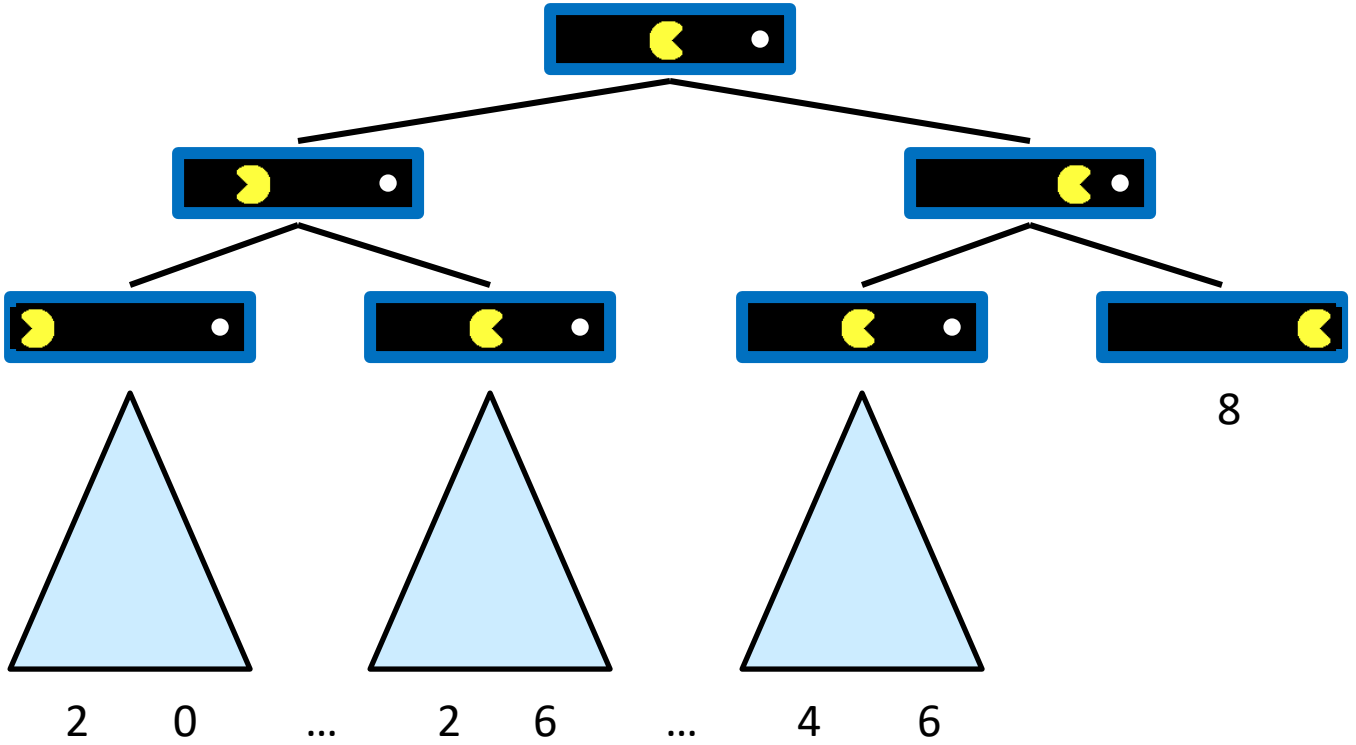
- Zero-Sum Games
  - Agents have ***opposite*** utilities
  - Pure competition:
    - One ***maximizes***, the other ***minimizes***

- General Games
  - Agents have ***independent*** utilities
  - Cooperation, indifference, competition, shifting alliances, and more are all possible

# Adversarial Search



# Single-Agent Trees

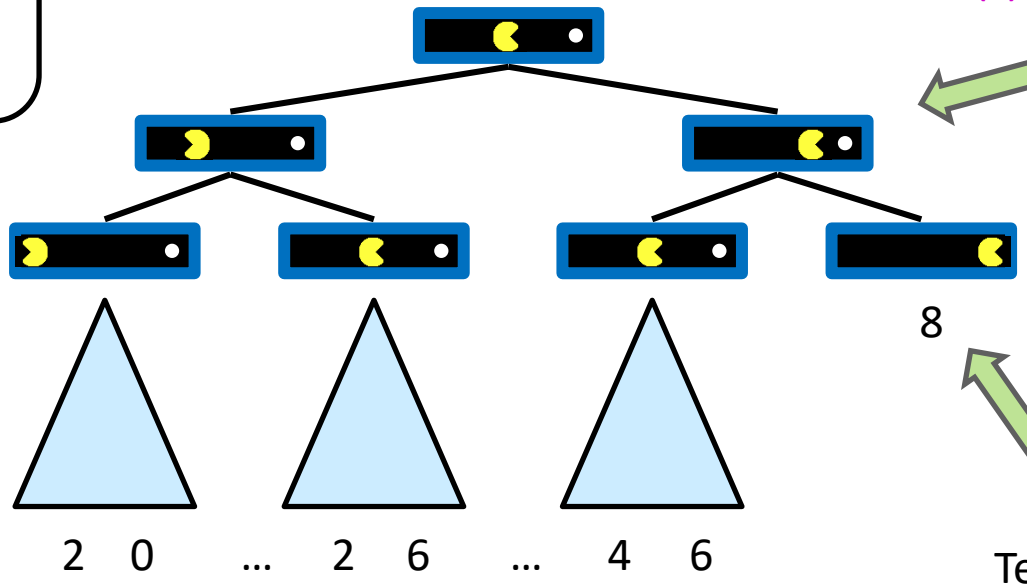


# Value of a State

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

Non-Terminal States:

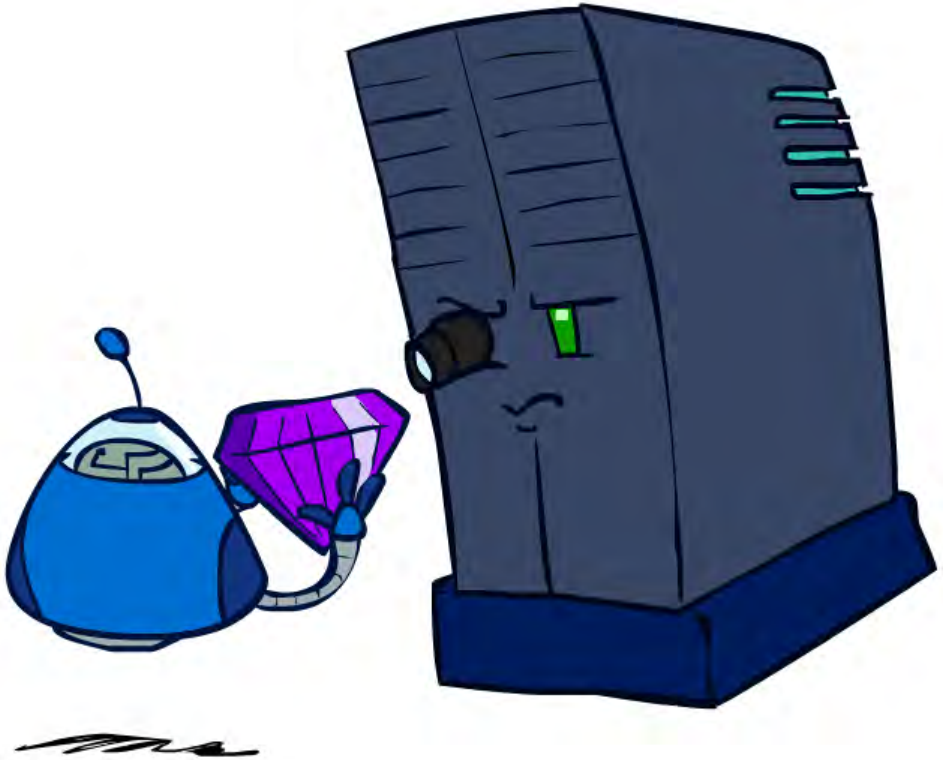
$V(s) = \max_{s' \in \text{successors}(s)} V(s')$




Terminal States:

$V(s)$ = known

# Evaluation Functions



# Evaluation Functions

- Typically weighted linear sum of features:
  - EVAL(s) = $w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$
  - E.g., $w_1 = 9$, $f_1(s)$ = (num white queens – num black queens), etc.

| رنگ چشم | ارزش f1 |
|---|---|
| آبی | 0.3 |
| سبز | 0.3 |
| قهوه‌ای | 0.4 |

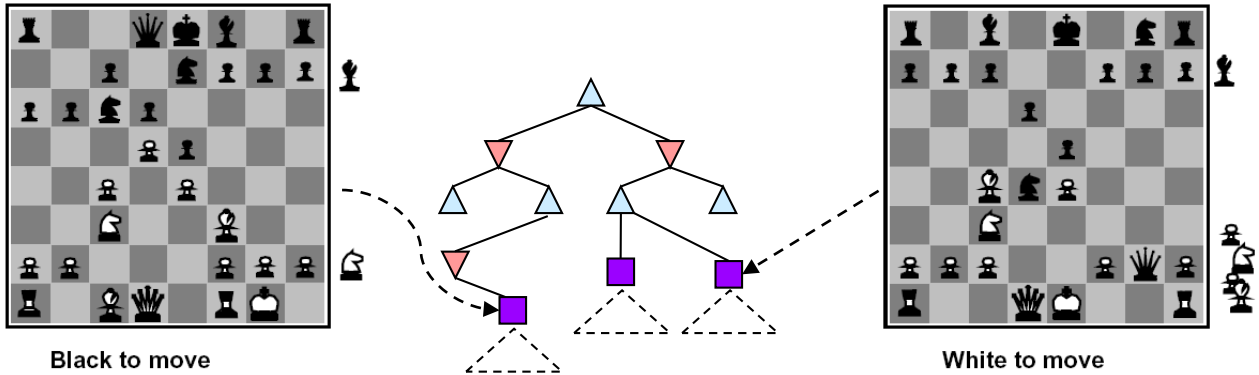
| وزن | ارزش f2 |
|---|---|
| < 100 | 0.3 |
| 100< < 150 | 0.5 |
| >150 | 0.2 |
| | |

| قد | ارزش f3 |
|---|---|
| < 100 | 0.2 |
| 100< < 150 | 0.6 |
| >200 | 0.2 |

| معیار | اهمیت | |
|---|---|---|
| رنگ چشم | 0.3 | w1 |
| وزن | 0.4 | w2 |
| قد | 0.3 | w3 |

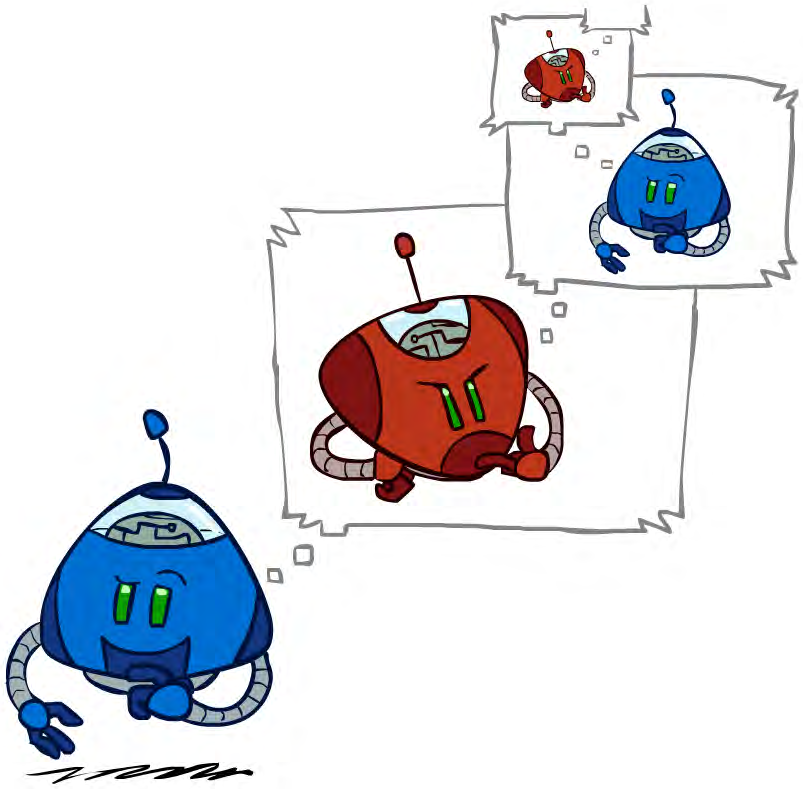
# Evaluation Functions

- Evaluation functions score non-terminals in depth-limited search

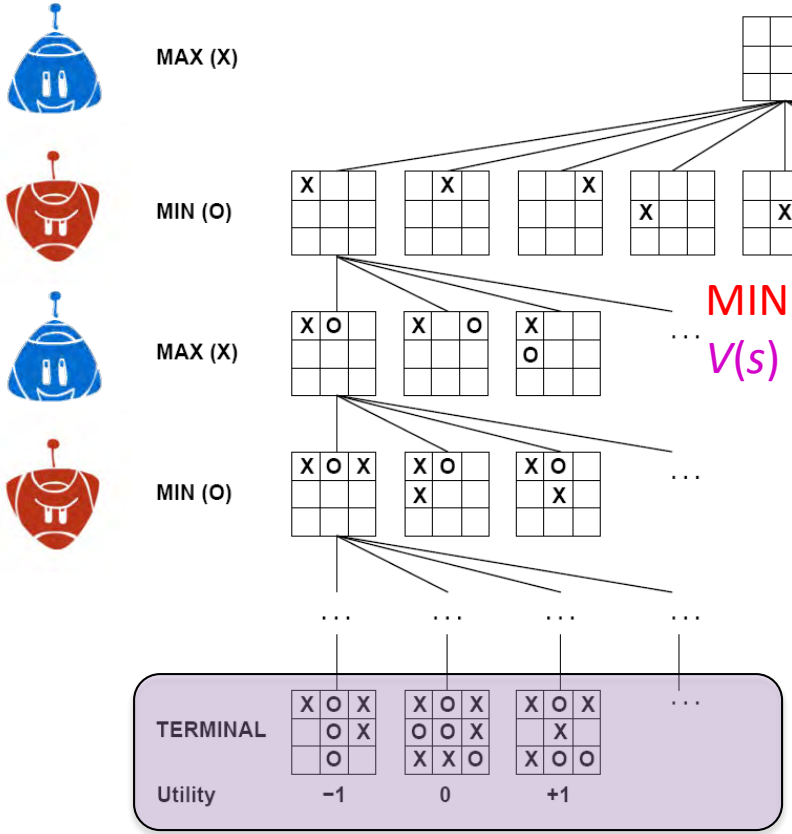



- Typically weighted linear sum of features:
  - EVAL(s) = $w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$
  - E.g., $w_1 = 9$, $f_1(s)$ = (num white queens – num black queens), etc.
- Or a more complex nonlinear function (e.g., NN) trained by self-play RL
- Terminate search only in ***quiescent*** positions, i.e., no major changes expected in feature values

# Adversarial Search



# Tic-Tac-Toe Game Tree: Minimax Values

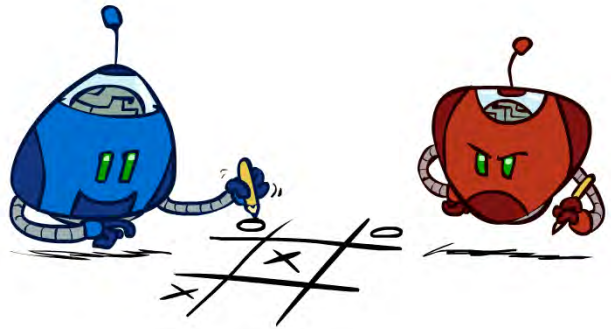



MAX nodes: under Agent's control

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

MIN nodes: under Opponent's control

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



Terminal States:

$V(s)$ = known

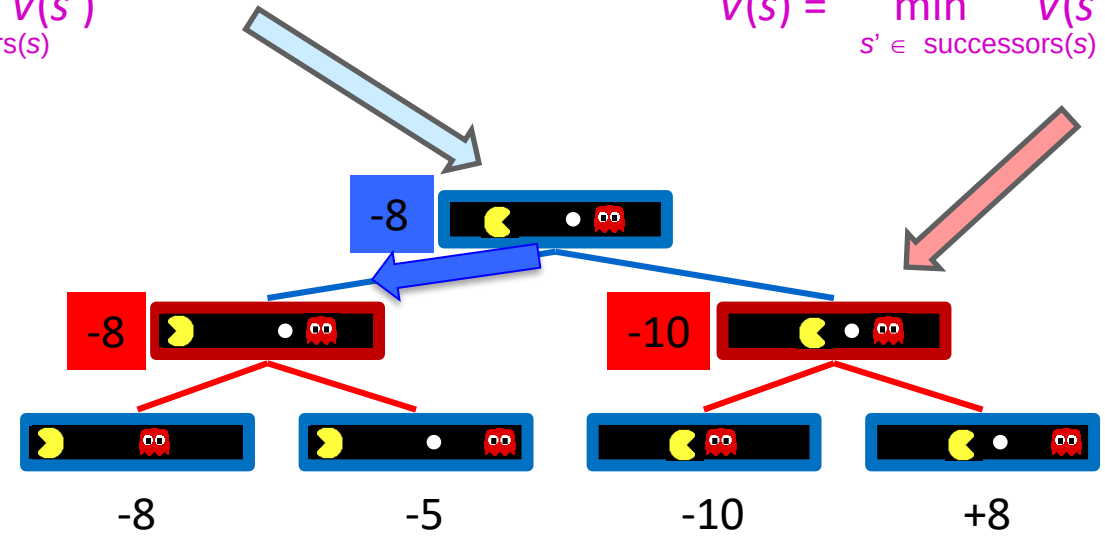
# Minimax Values

MAX nodes: under Agent's control

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

MIN nodes: under Opponent's control

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




Terminal States:

$V(s)$ = known

# Minimax algorithm

- Choose action leading to state with best ***minimax value***
- Assumes all future moves will be optimal
- => rational against a rational player

# Implementation

function minimax-decision(s) returns an action

return the action a in Actions(s) with the highest minimax_value(Result(s,a))

function minimax_value(s) returns a value

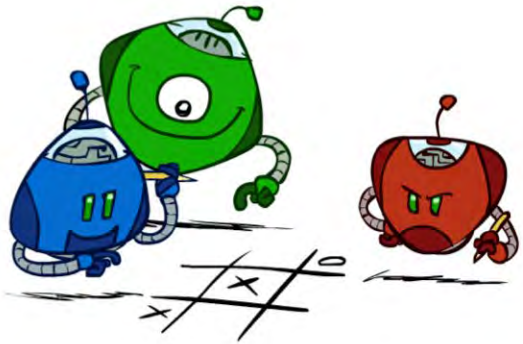
if Terminal-Test(s) then return Utility(s)

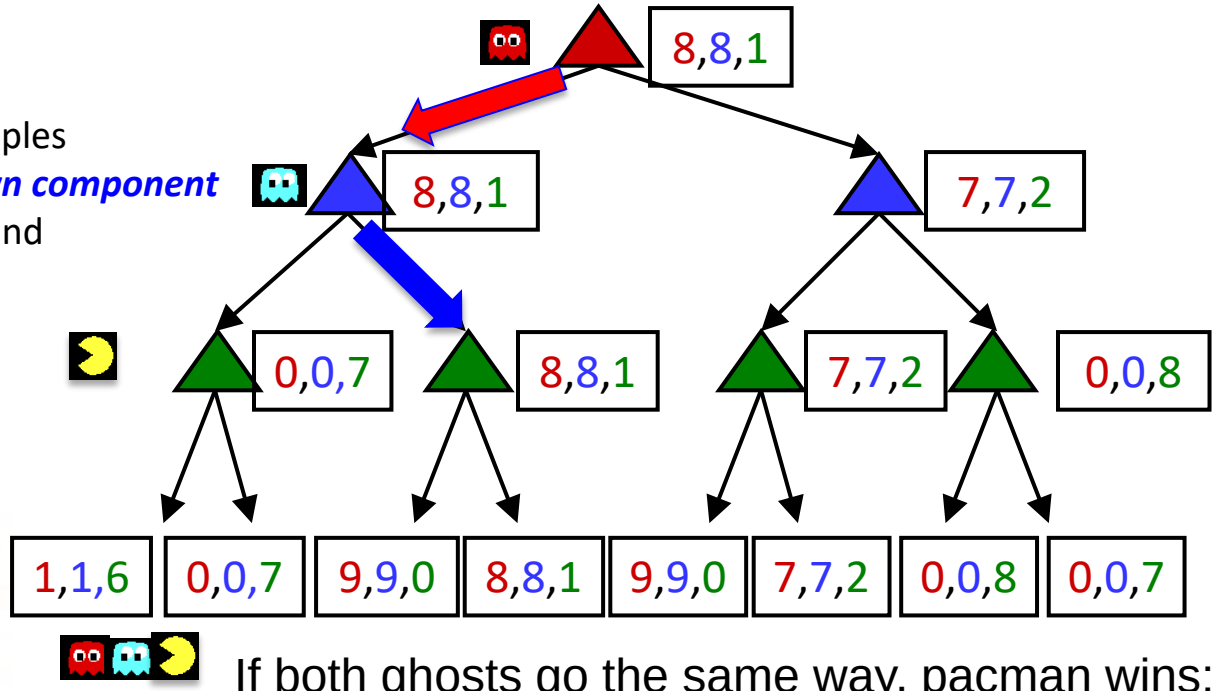
if Player(s) = MAX then return $\max_{a \text{ in Actions(s)}}$ minimax_value(Result(s,a))

if Player(s) = MIN then return $\min_{a \text{ in Actions(s)}}$ minimax_value(Result(s,a))

# Generalized minimax

- What if the game is not zero-sum, or has multiple players?
- Generalization of minimax:
  - Terminals have ***utility tuples***
  - Node values are also utility tuples
  - ***Each player maximizes its own component***
  - Can give rise to cooperation and competition dynamically…



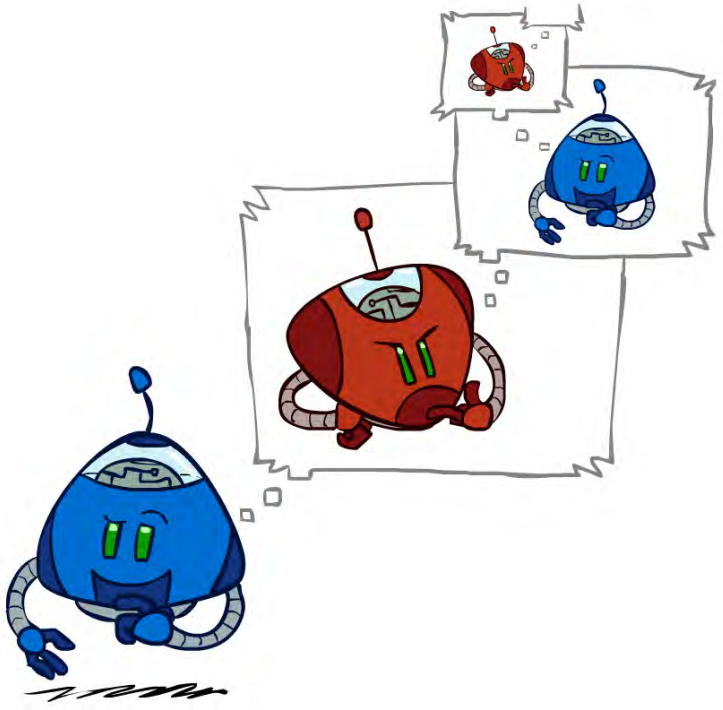



If both ghosts go the same way, pacman wins; if they go different ways, pacman loses

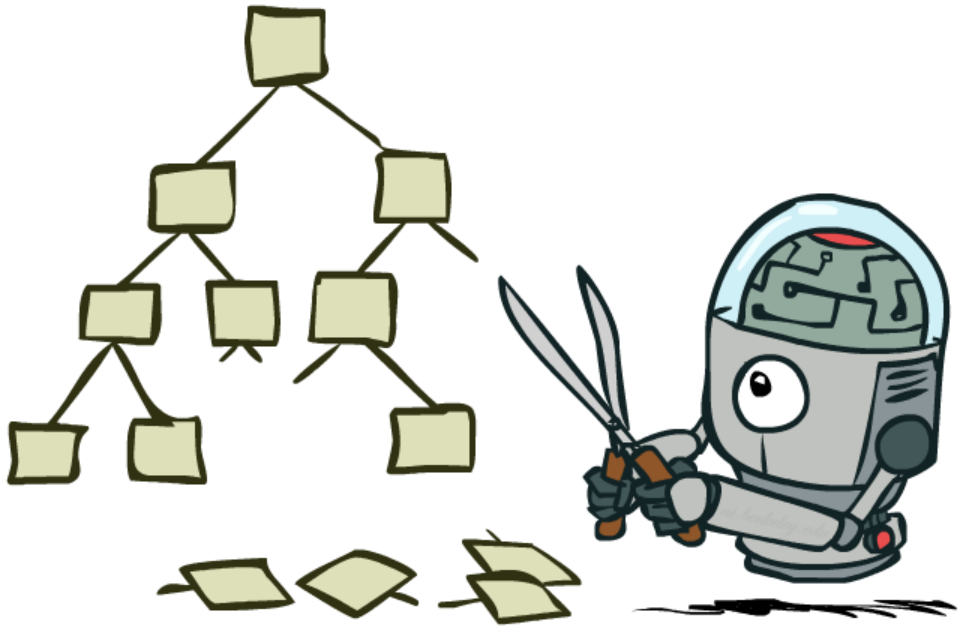
# Minimax Efficiency

- How efficient is minimax?
  - Just like (exhaustive) DFS
  - Time: $O(b^m)$
  - Space: O(bm)

- Example: For chess, $b \approx 35$, $m \approx 100$
  - Exact solution is completely infeasible
  - Humans can't do this either, so how do we play chess?

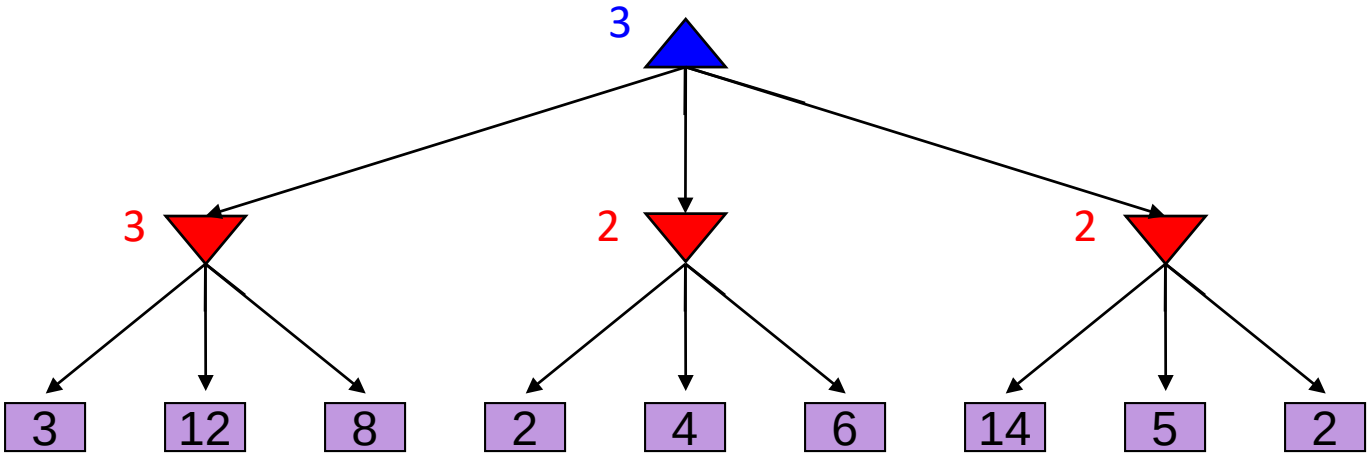


# Game Tree Pruning



- امکان محاسبه تصمیم minimax صحیح بدون دیدن همه گره‌های درخت بازی
- حذف انشعاب‌هایی که در تصمیم نهایی تأثیر ندارند

# Minimax Example



MAX

MIN

MAX

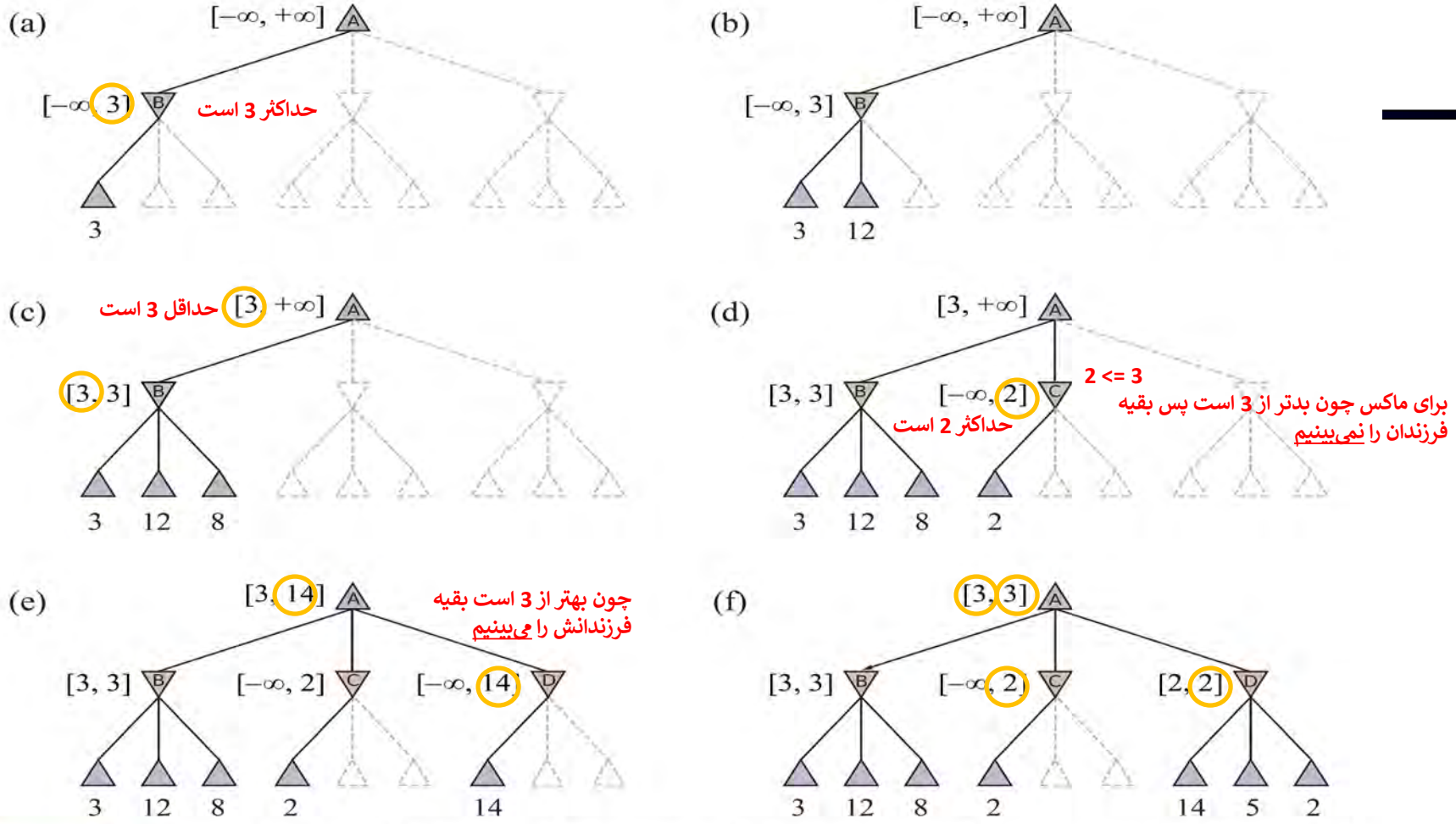



**Figure 6.5** Stages in the calculation of the optimal decision for the game tree in Figure 6.2.

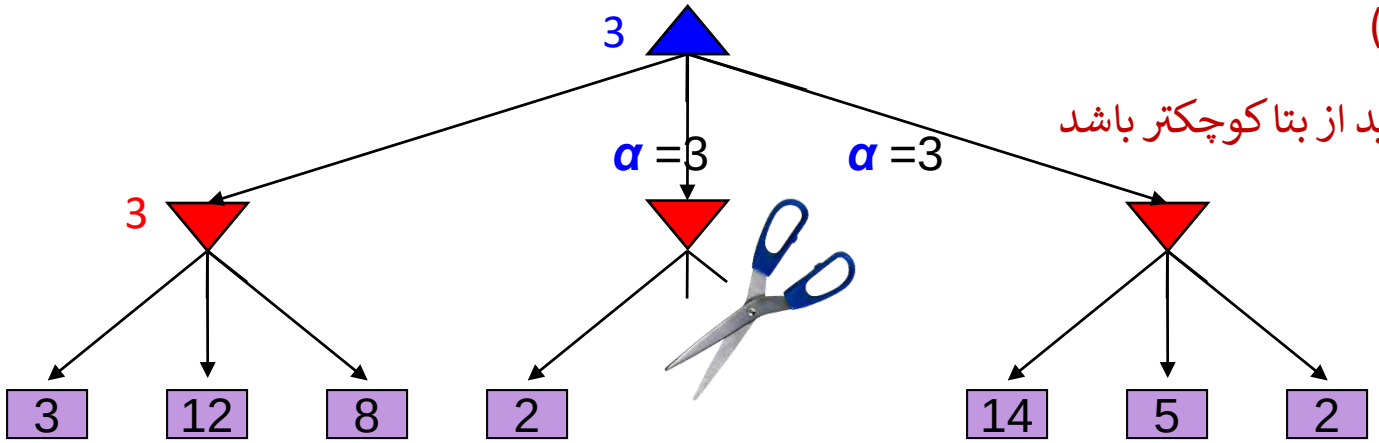
# Alpha-Beta Example

***α*** = best option so far from any MAX node on this path

آلفا افزایش داده شود: متعلق به ماکس (یعنی بالاترین مقدار که تاکنون پیدا شده است)

بتا کاهش داده شود: متعلق به مین (یعنی پایین‌ترین مقدار که تاکنون پیدا شده است)

تجربی: آلفا نباید از بتا کوچکتر باشد

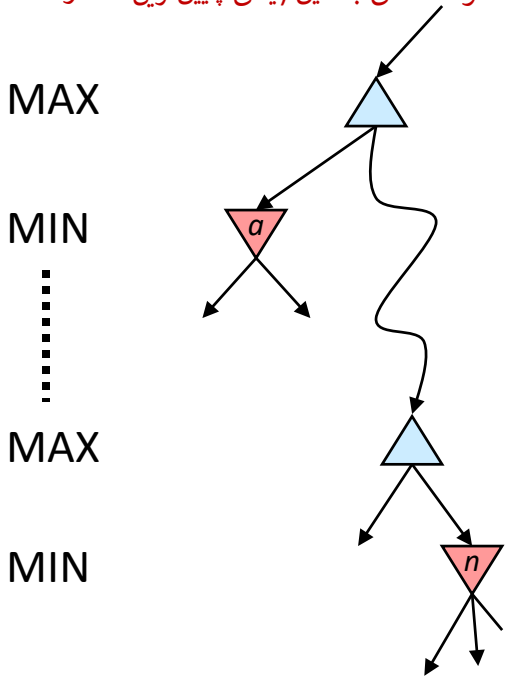



***The order of generation matters***: more pruning is possible if good moves come first

# Alpha-Beta Pruning

آلفا افزایش داده شود: متعلق به ماکس (یعنی بالاترین مقدار که تاکنون پیدا شده است)
بتا کاهش داده شود: متعلق به مین (یعنی پایین‌ترین مقدار که تاکنون پیدا شده است)

- General case (**pruning children of MIN node**)
  - We're computing the MIN-VALUE at some node *n*
  - We're looping over *n*'s children
  - *n*'s estimate of the childrens' min is dropping
  - Who cares about *n*'s value? MAX
  - Let **α** be the best value that MAX can get so far at any choice point along the current path from the root
  - **If *n* becomes worse than α, MAX will avoid it**, so we can **prune *n*'s other children** (it's already bad enough that it won't be played)
- Pruning children of MAX node is symmetric
  - Let **β** be the best value that MIN can get so far at any choice point along the current path from the root




تجربی: آلفا نباید از بتا کوچکتر باشد

# Alpha-Beta Implementation

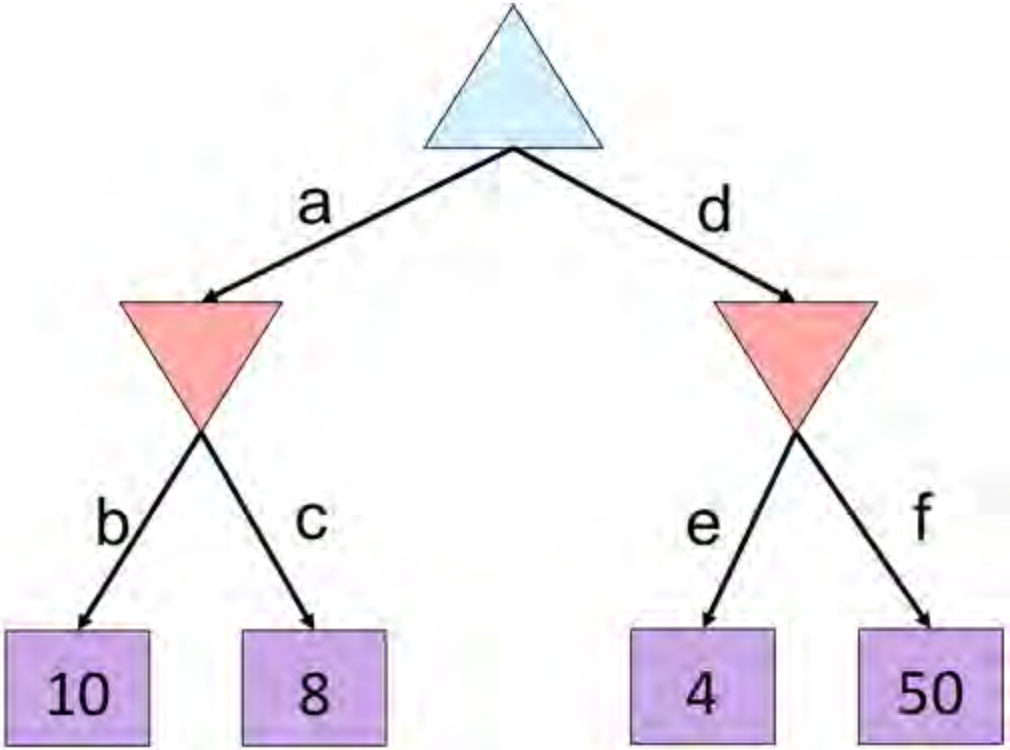
α: MAX's best option on path to root
β: MIN's best option on path to root

Pick an order for two reasons: sequential processor and pruning

```
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
```

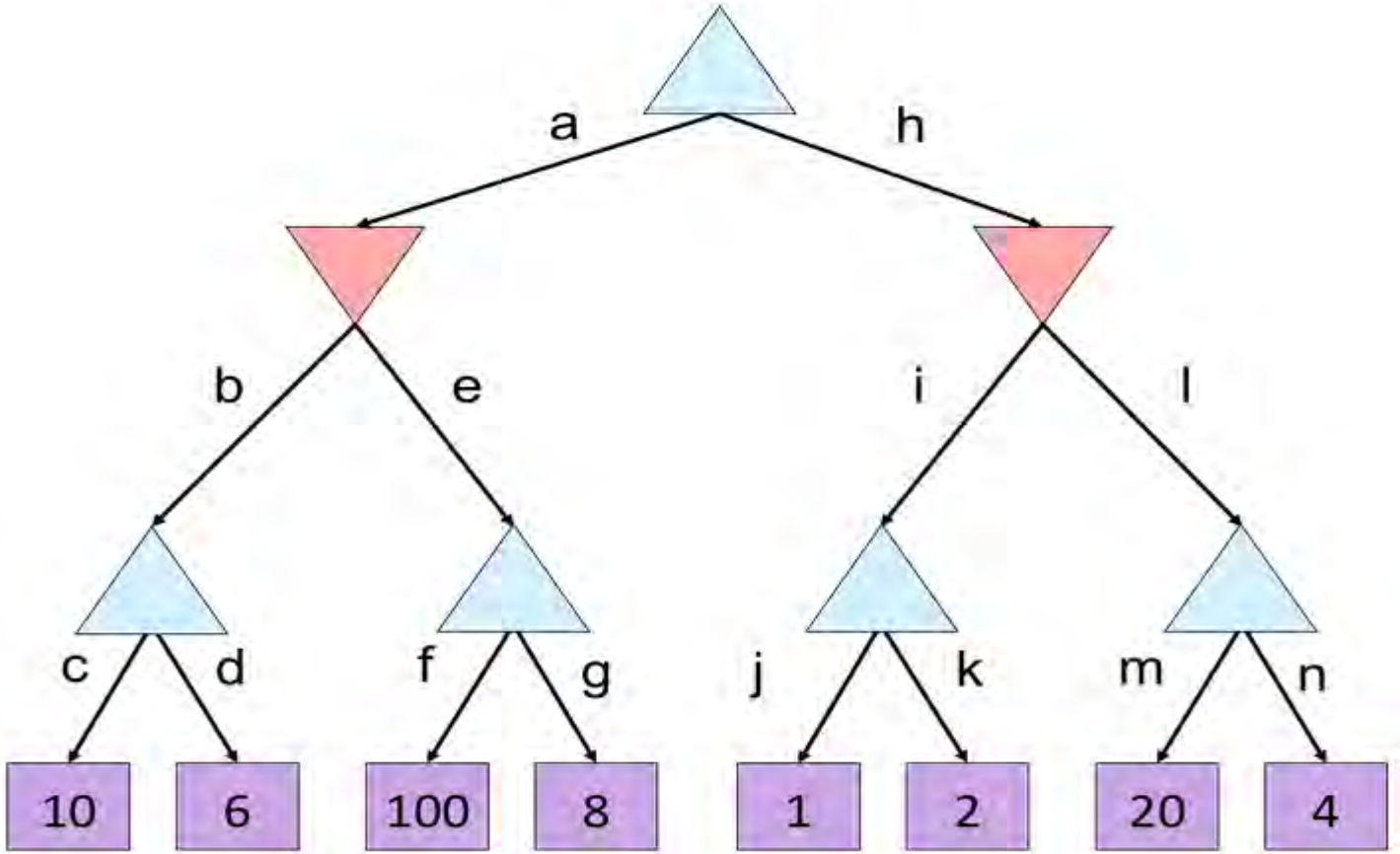
```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α
            return v
        β = min(β, v)
    return v
```

# Alpha-Beta Example Quiz




f is not evaluated

# Alpha-Beta Quiz 2



# Alpha-Beta Quiz 2

```
function ALPHA-BETA-SEARCH(game, state) returns an action
  player ← game.TO-MOVE(state)
  value, move ← MAX-VALUE(game, state, −∞, +∞)
  return move

function MAX-VALUE(game, state, α, β) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← −∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MIN-VALUE(game, game.RESULT(state, a), α, β)
    if v2 > v then
      v, move ← v2, a
      α ← MAX(α, v)
    if v ≥ β then return v, move
  return v, move

function MIN-VALUE(game, state, α, β) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← +∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MAX-VALUE(game, game.RESULT(state, a), α, β)
    if v2 < v then
      v, move ← v2, a
      β ← MIN(β, v)
    if v ≤ α then return v, move
  return v, move
```

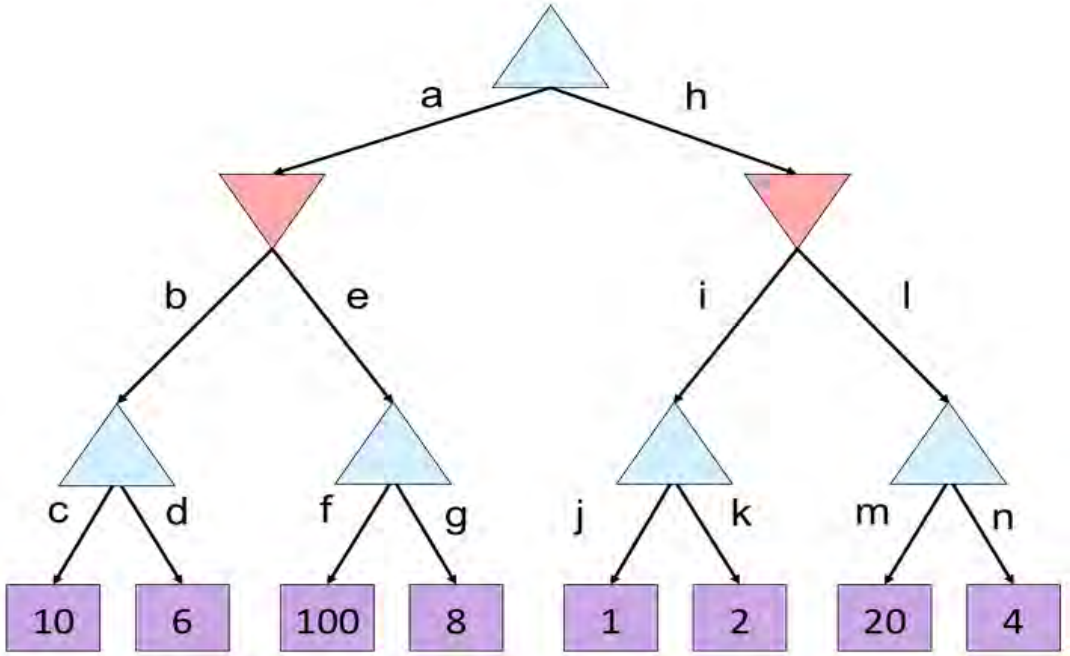



**Figure 6.7** The alpha–beta search algorithm. Notice that these functions are the same as the MINIMAX-SEARCH functions in Figure 6.3, except that we maintain bounds in the variables $\alpha$ and $\beta$, and use them to cut off search when a value is outside the bounds.

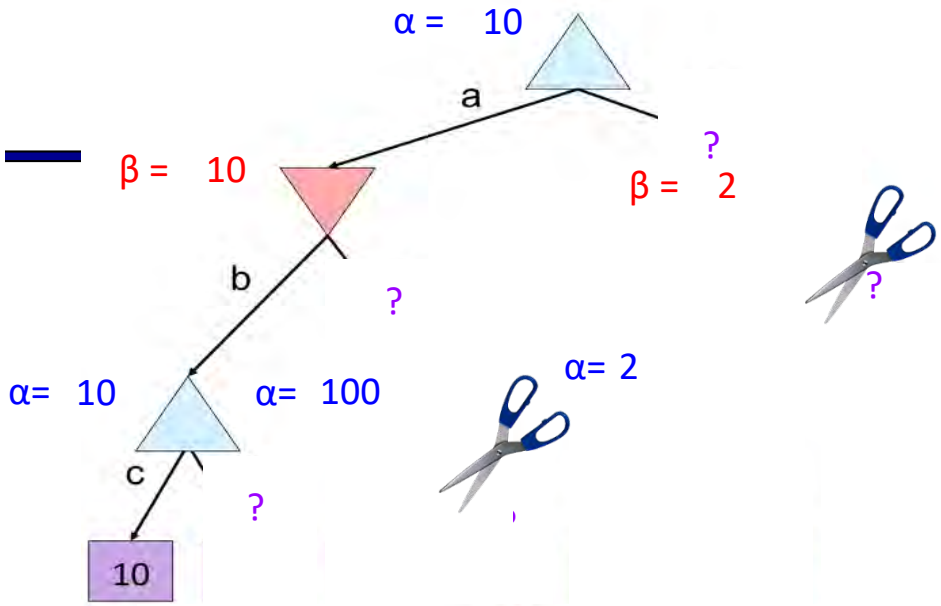
# Alpha-Beta Quiz 2

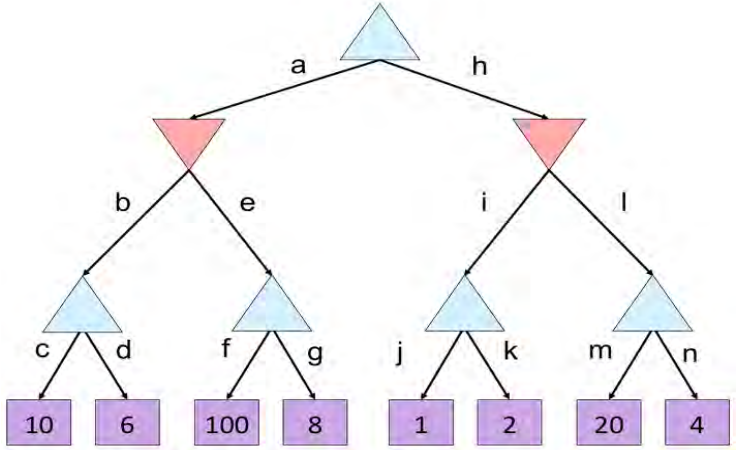
```
function ALPHA-BETA-SEARCH(game, state) returns an action
  player ← game.TO-MOVE(state)
  value, move ← MAX-VALUE(game, state, −∞, +∞)
  return move

function MAX-VALUE(game, state, α, β) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← −∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MIN-VALUE(game, game.RESULT(state, a), α, β)
    if v2 > v then
      v, move ← v2, a
      α ← MAX(α, v)
    if v ≥ β then return v, move
  return v, move

function MIN-VALUE(game, state, α, β) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← +∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MAX-VALUE(game, game.RESULT(state, a), α, β)
    if v2 < v then
      v, move ← v2, a
      β ← MIN(β, v)
    if v ≤ α then return v, move
  return v, move
```

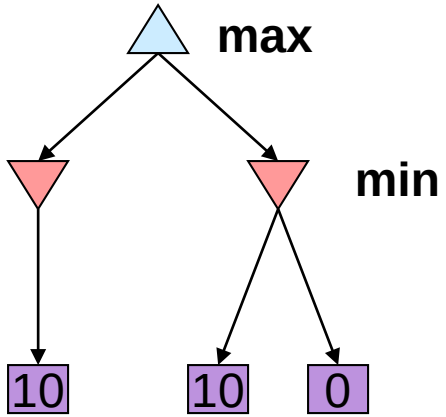
**Figure 6.7** The alpha–beta search algorithm. Notice that these functions are the same as the MINIMAX-SEARCH functions in Figure 6.3, except that we maintain bounds in the variables $\alpha$ and $\beta$, and use them to cut off search when a value is outside the bounds.



# Alpha-Beta Pruning Properties

- Theorem: This pruning has ***no effect*** on minimax value computed for the root!
- Good child ordering improves effectiveness of pruning
  - Iterative deepening helps with this
- With "perfect ordering":
  - Time complexity drops to $O(b^{m/2})$
  - Doubles solvable depth!

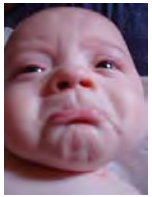



- This is a simple example of metareasoning (reasoning about reasoning)
- For chess: only $35^{50}$ instead of $35^{100}$!! Yaaay!!!!!

- ابتدا پسین‌هایی بررسی شوند که ممکن است بهتر باشند!
- پسین‌ها به طور تصادفی بررسی شوند
- جستجوی تعمیق تکراری

# The story so far…

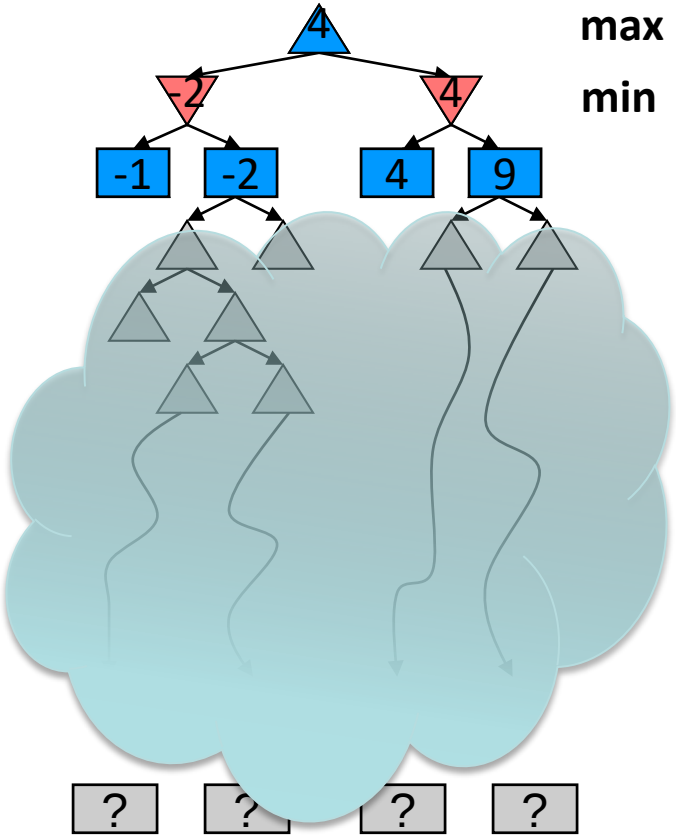
- Focus on two-player, zero-sum, deterministic, observable, turn-taking games
- Minimax defines rational behavior
- Recursive DFS implementation: space complexity $O(bm)$, time complexity $O(b^m)$
- Alpha-beta pruning with good node ordering reduces time complexity to $O(b^{m/2})$
- Still nowhere close to solving chess, let alone Go or StarCraft



# Resource Limits

# Resource Limits

- Problem: In realistic games, cannot search to leaves!
- Solution 1 (Shannon, 1950): : Bounded lookahead
  - Search only to a preset ***depth limit*** or ***horizon***
  - Use an ***evaluation function*** for non-terminal positions
- Guarantee of optimal play is gone
- More plies make a BIG difference
- Example:
  - Suppose we have 100 seconds, can explore 10K nodes / sec
  - So can check 1M nodes per move
  - Chess with alpha-beta, $35^{(8/2)}$ =~ 1M; depth 8 is good

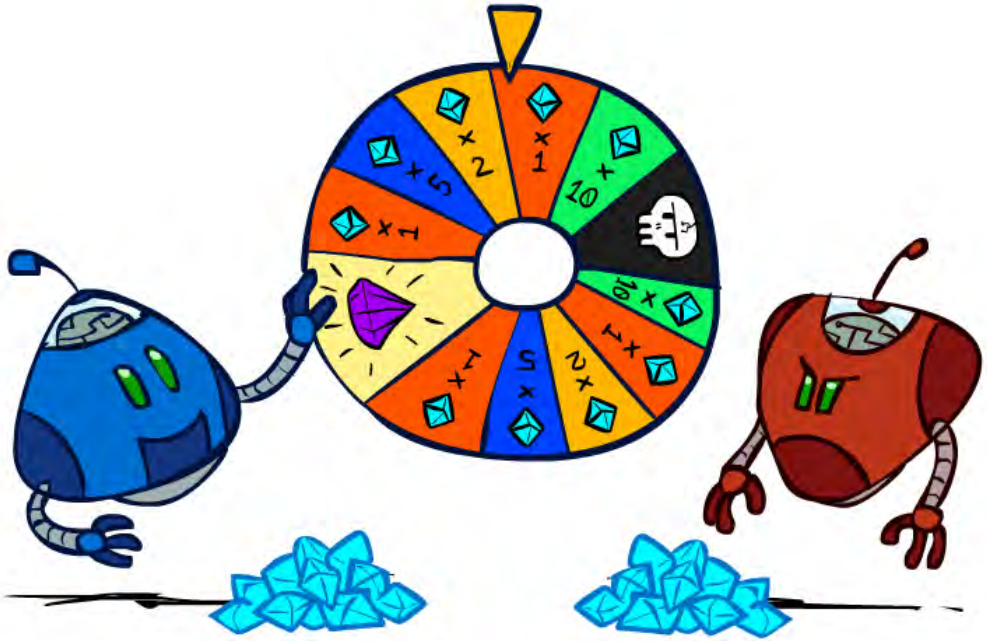


# Depth Matters

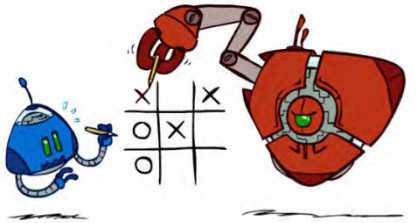
- Evaluation functions are always imperfect
- Deeper search => better play (usually)
- Or, deeper search gives same quality of play with a less accurate evaluation function
- An important example of the tradeoff between complexity of features and complexity of computation

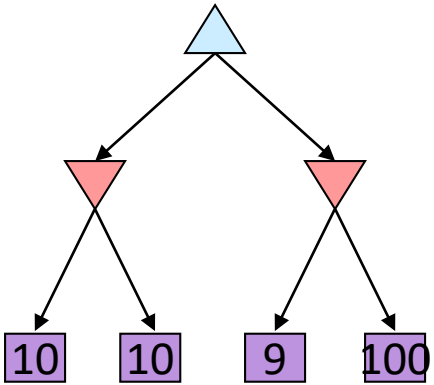
[Demo: depth limited (L6D4, L6D5)]

# Games with uncertain outcomes



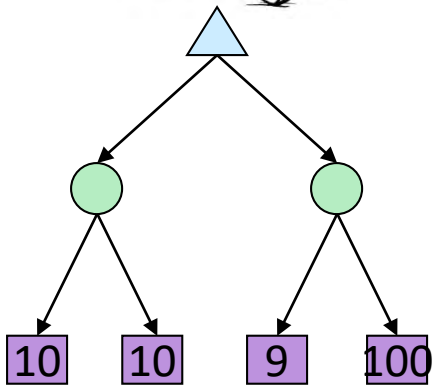
# Chance outcomes in trees





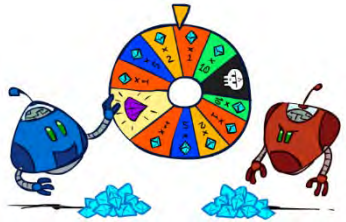

Tictactoe, chess

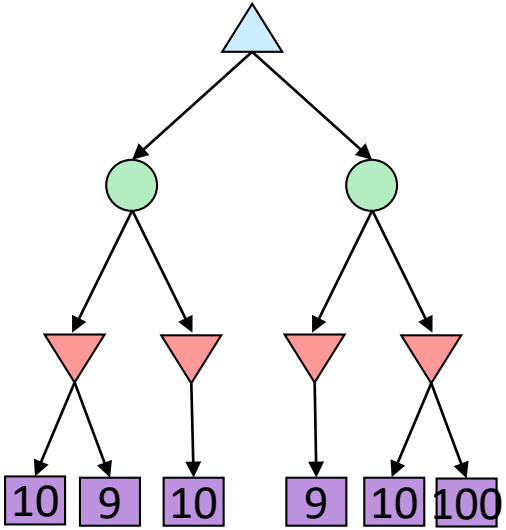
***Minimax***




Tetris, investing

***Expectimax***






Backgammon, Monopoly

***Expectiminimax***

At least 10, or otherwise just rather dead; or just 100, or otherwise rather dead; that's yet something different than minimax or expectimax

# Minimax

```
function decision(s) returns an action
    return the action a in Actions(s) with the highest
     minimax_value(Result(s,a))
```

```
function minimax_value(s) returns a value
    if Terminal-Test(s) then return Utility(s)
    if Player(s) = MAX      then return max_{a in Actions(s)} minimax_value(Result(s,a))
    if Player(s) = MIN      then return min_{a in Actions(s)} minimax_value(Result(s,a))
```

Note that “minimax-decision” has become just “decision”

# Expectiminimax

function decision(s) returns an action

return the action a in Actions(s) with the highest value(Result(s,a))

function value(s) returns a value

if Terminal-Test(s) then return Utility(s)

if Player(s) = MAX then return $\max_{a \text{ in Actions}(s)}$ value(Result(s,a))

if Player(s) = MIN then return $\min_{a \text{ in Actions}(s)}$ value(Result(s,a))

if Player(s) = CHANCE then return $\text{sum}_{a \text{ in Actions}(s)}$ Pr(a) * value(Result(s,a))

# Summary

- Games require decisions when optimality is impossible
  - Bounded-depth search and approximate evaluation functions
- Games force efficient use of computation
  - Alpha-beta pruning, MCTS
- Game playing has produced important research ideas
  - Reinforcement learning (checkers)
  - Iterative deepening (chess)
  - Rational metareasoning (Othello)
  - Monte Carlo tree search (chess, Go)
  - Solution methods for partial-information games in economics (poker)
- Video games present much greater challenges – lots to do!
  - $b = 10^{500}$, $|S| = 10^{4000}$, $m = 10{,}000$, partially observable, often > 2 players

# با سپاس از توجه شما
# دانشجوی گرامی