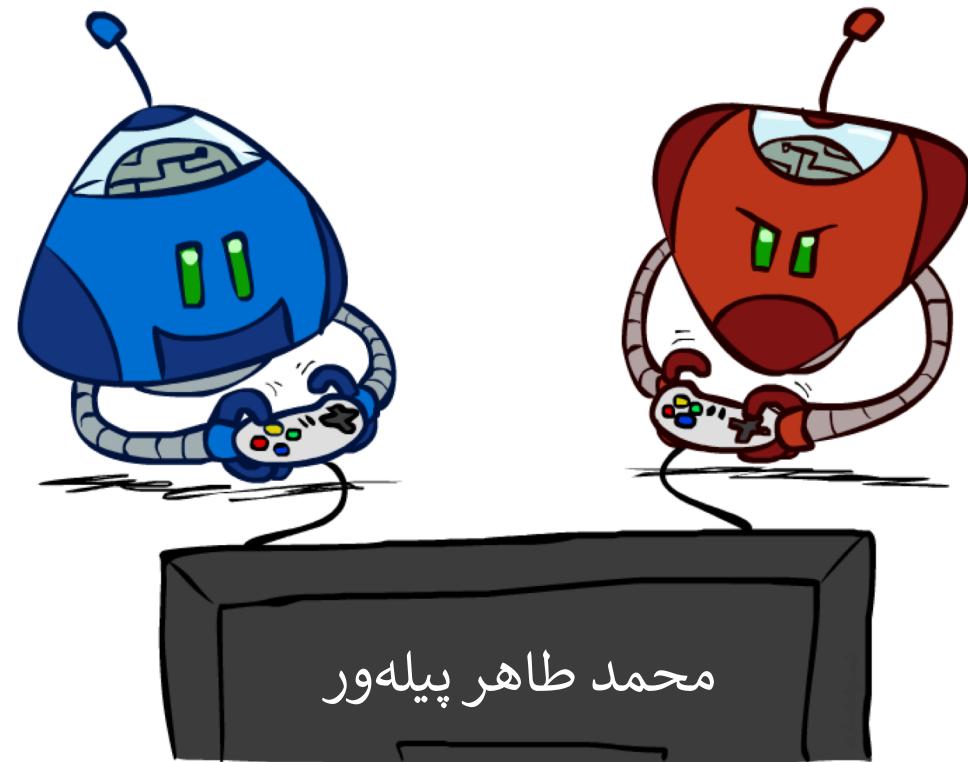


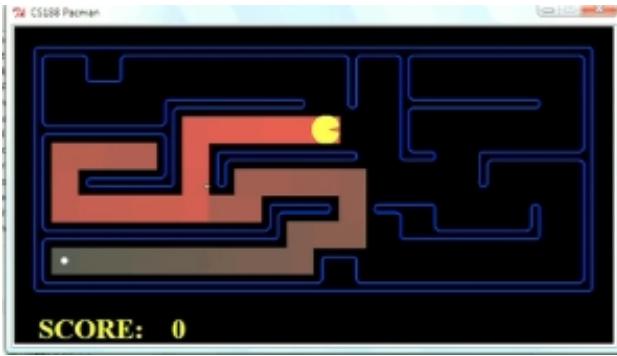
# هوش مصنوعی

## جستجوی رقابتی - بازی

### Game - Adversarial Search



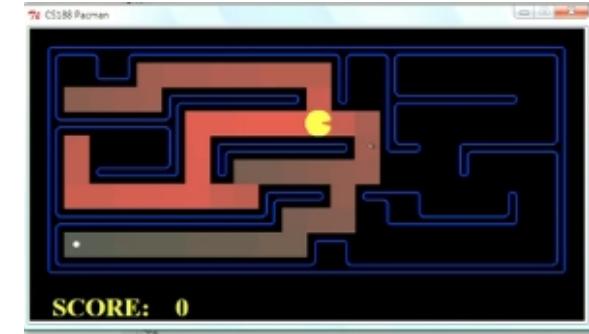
# تا به اینجا



Greedy

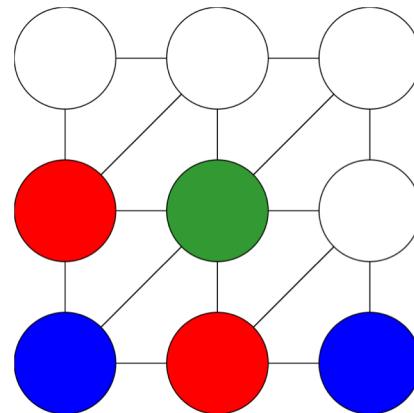


Uniform Cost



A\*

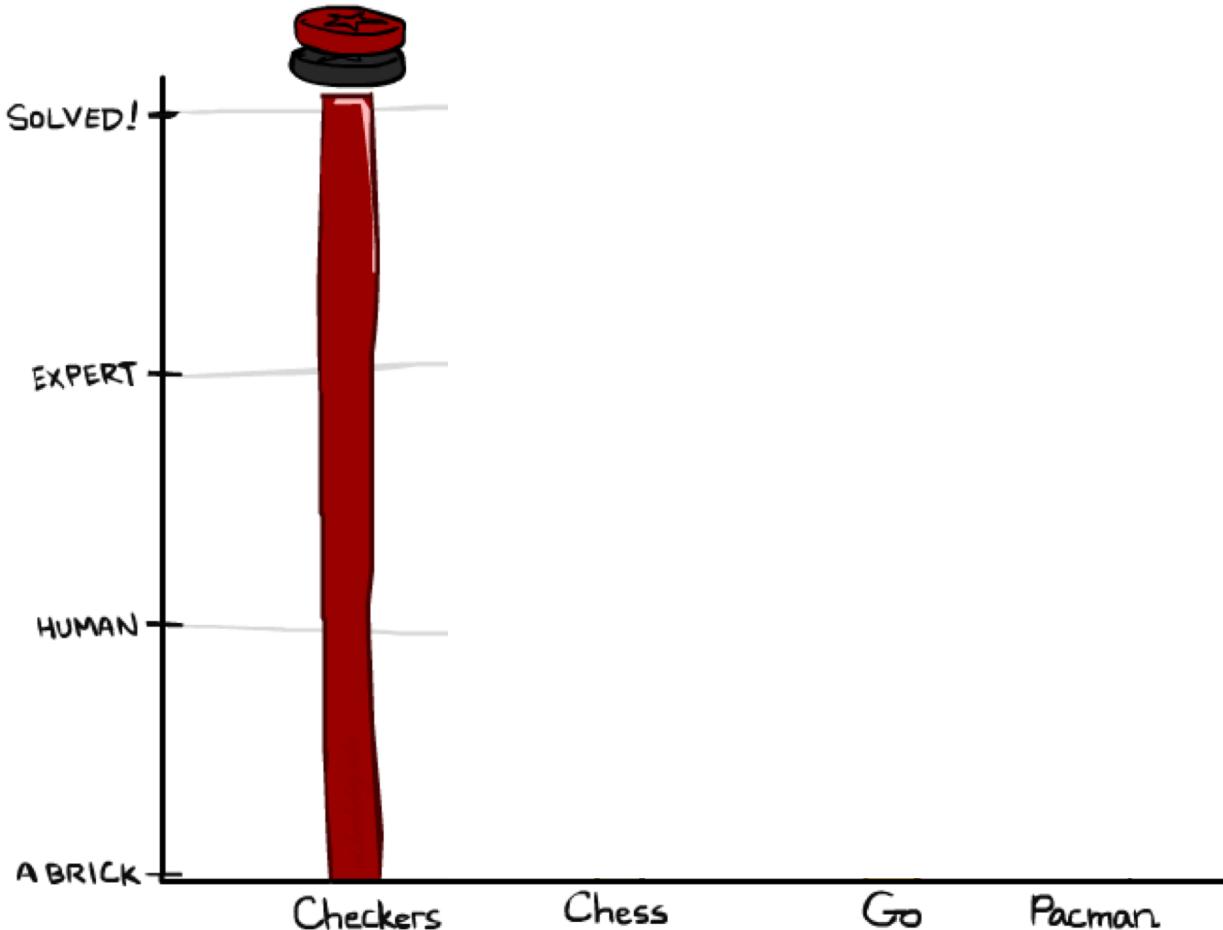
CSP



Backtracking  
Arc consistency  
Ordering  
Cutset conditioning  
...

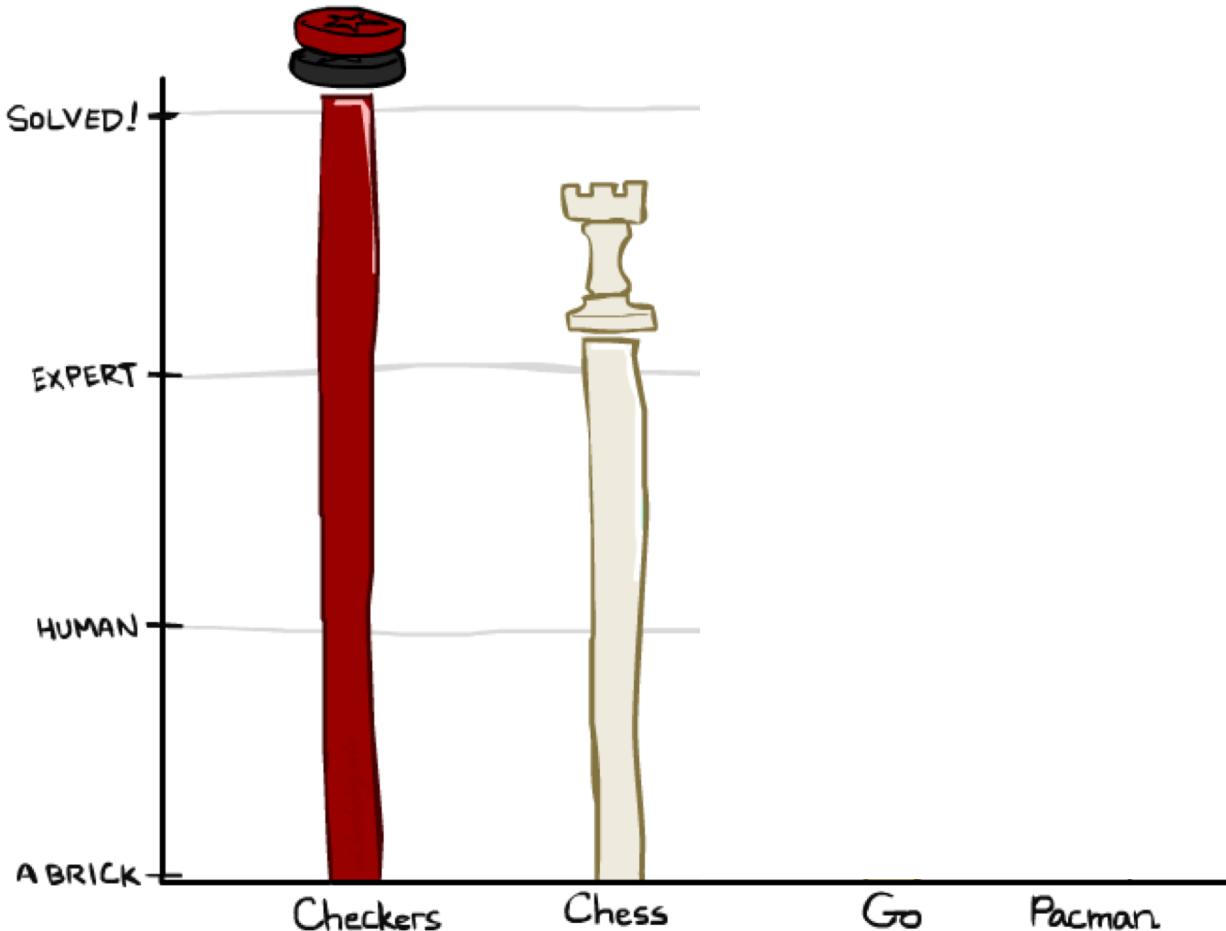
# در بازی State-of-the-Art

- **Checkers:** 1950: First computer player. 1994: First computer champion: Chinook ended 40-year-reign of human champion Marion Tinsley using complete 8-piece endgame. 2007: Checkers solved!



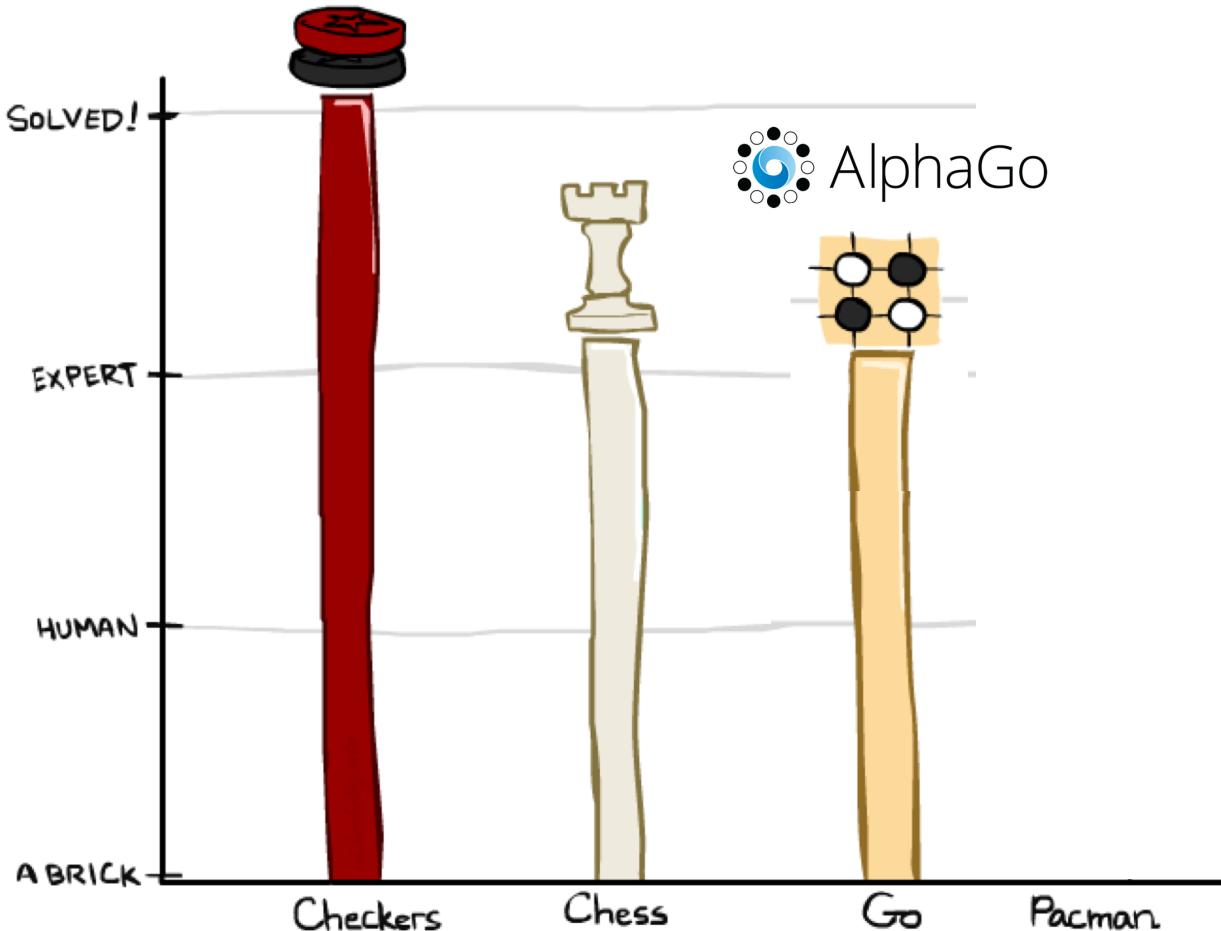
# در بازی State-of-the-Art

- **Checkers:** 1950: First computer player. 1994: First computer champion: Chinook ended 40-year-reign of human champion Marion Tinsley using complete 8-piece endgame. 2007: Checkers solved!
- **Chess:** 1997: Deep Blue defeats human champion Gary Kasparov in a six-game match. Deep Blue examined 200M positions per second, used very sophisticated evaluation and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic.

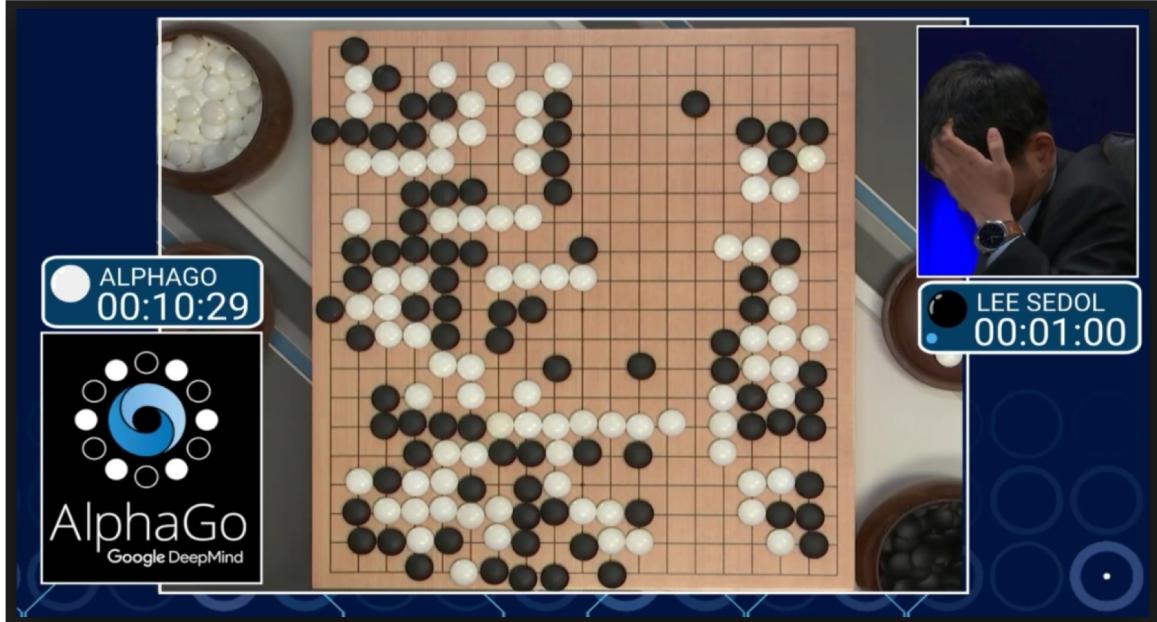


# در بازی State-of-the-Art

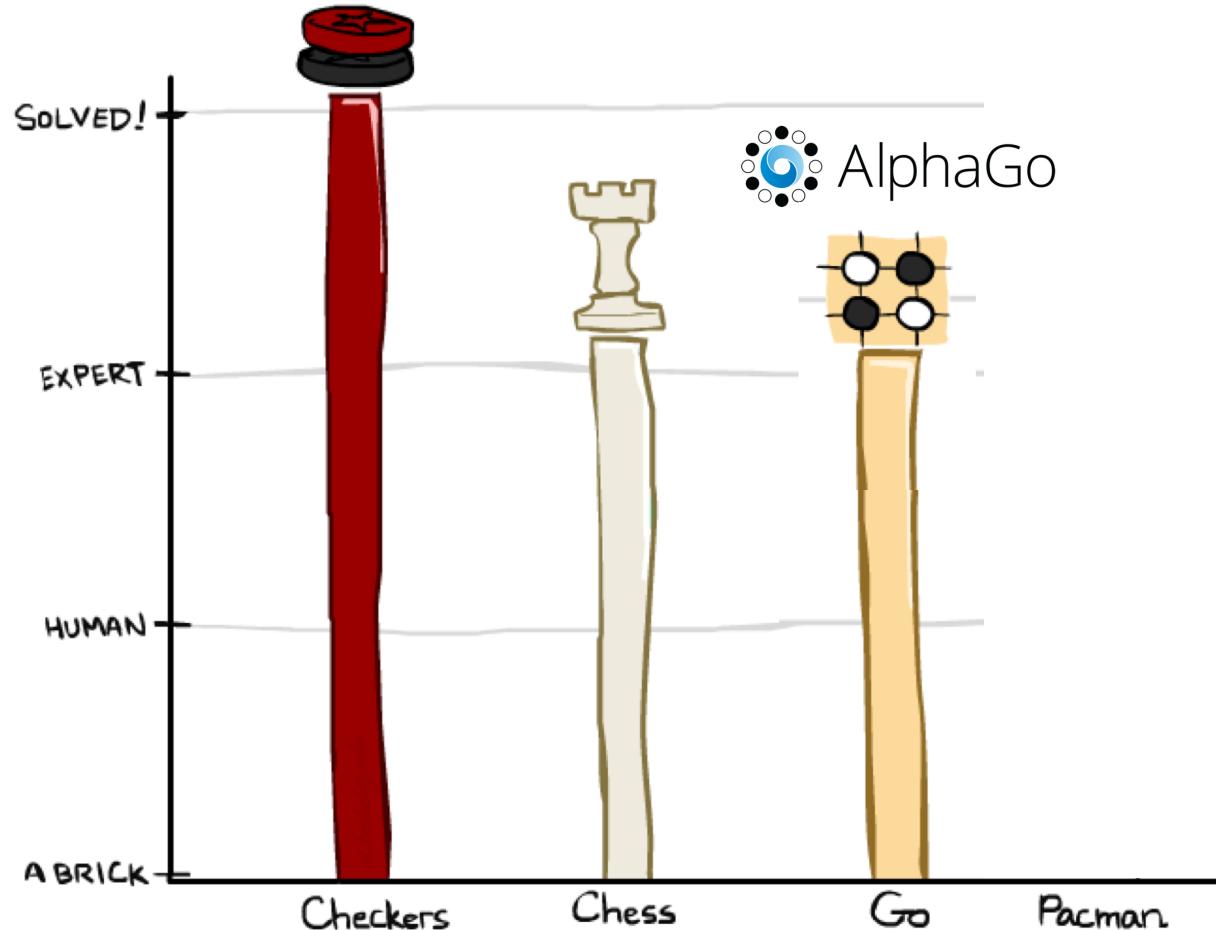
- **Checkers:** 1950: First computer player. 1994: First computer champion: Chinook ended 40-year-reign of human champion Marion Tinsley using complete 8-piece endgame. 2007: Checkers solved!
- **Chess:** 1997: Deep Blue defeats human champion Gary Kasparov in a six-game match. Deep Blue examined 200M positions per second, used very sophisticated evaluation and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic.
- **Go:** Using reinforcement learning, AlphaGo (Google DeepMind). In go,  $b > 300$ ! In October 2015, AlphaGo became the first computer Go program to beat a human professional Go player on a full-sized  $19 \times 19$  board. In March 2016, it beat Lee Sedol in a five-game match, the first time a computer Go program has beaten a 9-dan professional.



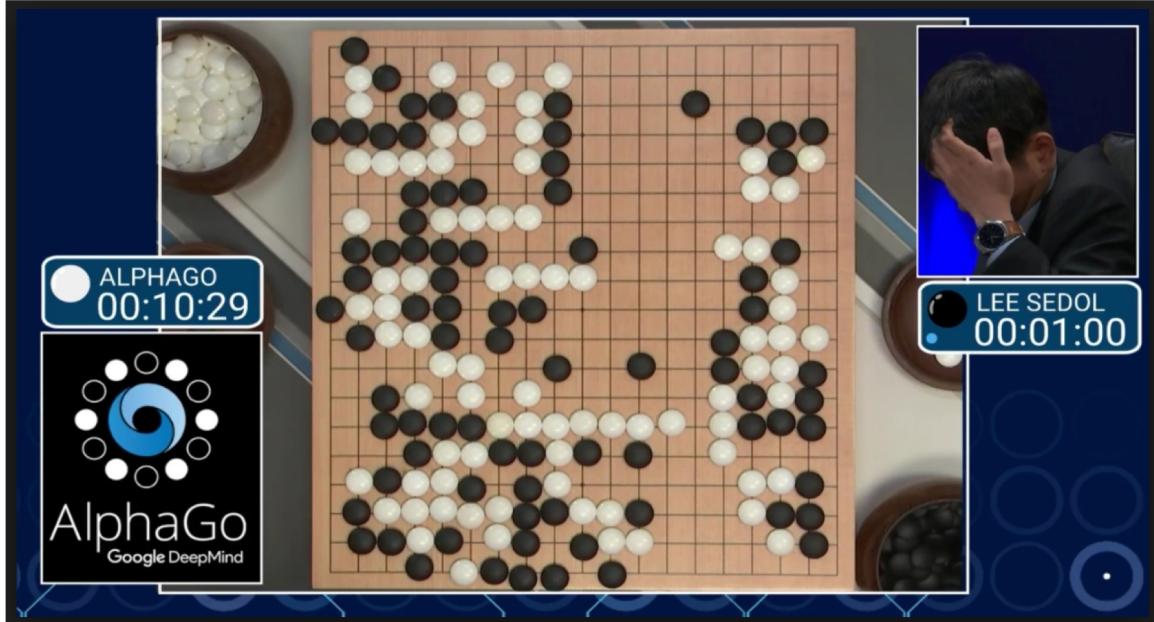
# در بازی State-of-the-Art



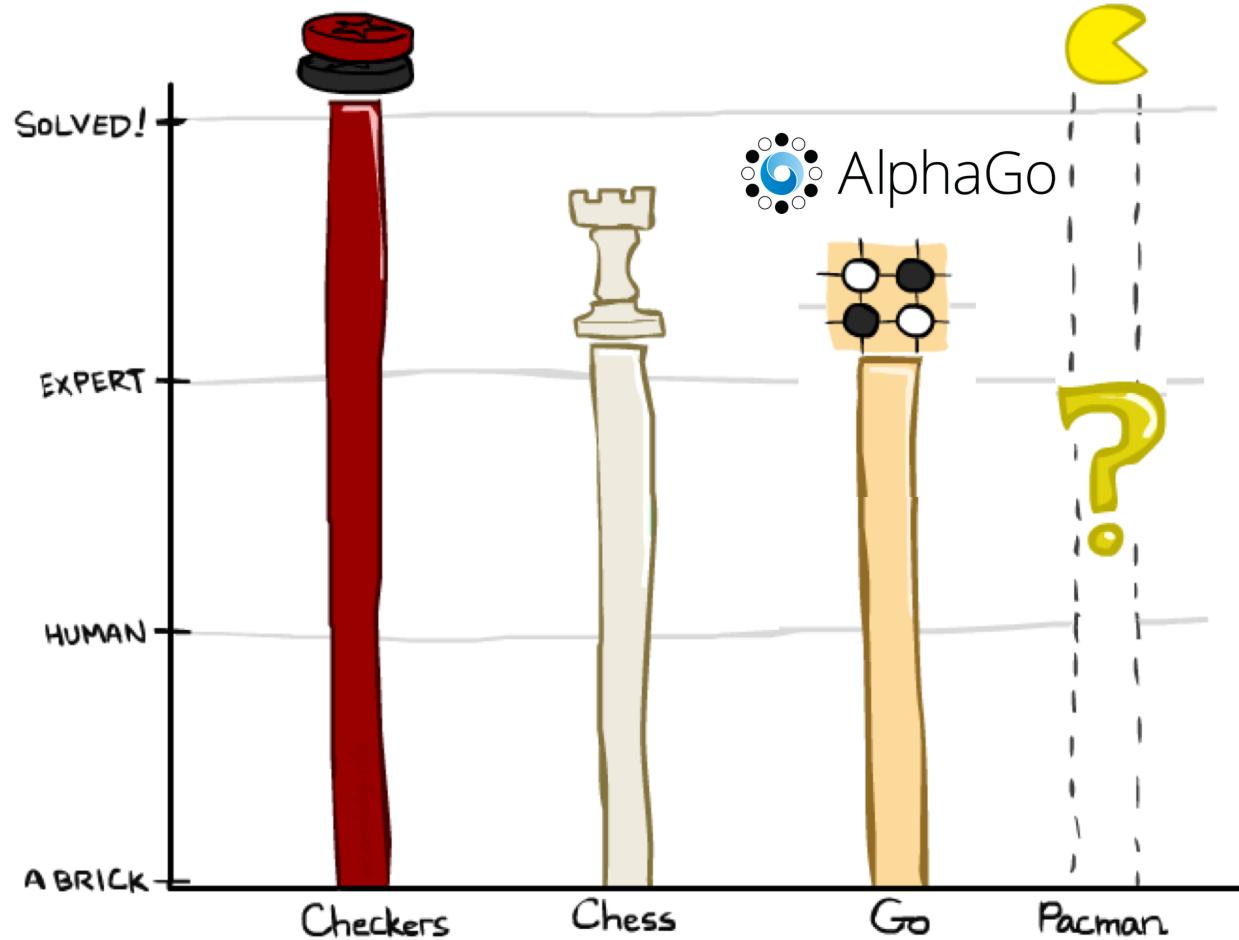
- **Go:** Using reinforcement learning, AlphaGo (Google DeepMind). In go,  $b > 300$ ! In October 2015, AlphaGo became the first computer Go program to beat a human professional Go player on a full-sized  $19 \times 19$  board. In March 2016, it beat Lee Sedol in a five-game match, the first time a computer Go program has beaten a 9-dan professional.



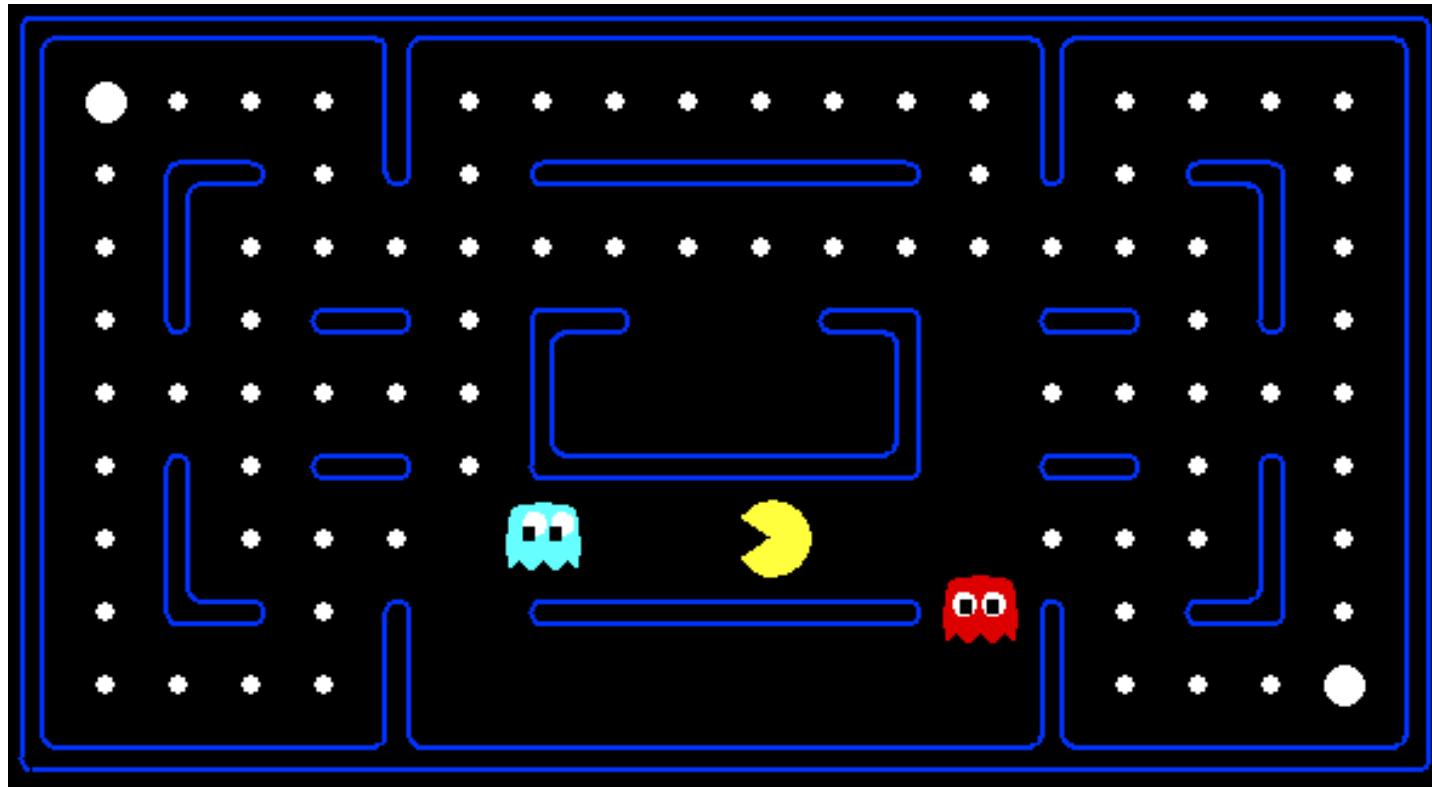
# در بازی State-of-the-Art



- **Go:** Using reinforcement learning, AlphaGo (Google DeepMind). In go,  $b > 300$ ! In October 2015, AlphaGo became the first computer Go program to beat a human professional Go player on a full-sized 19×19 board. In March 2016, it beat Lee Sedol in a five-game match, the first time a computer Go program has beaten a 9-dan professional.
- **Pacman**



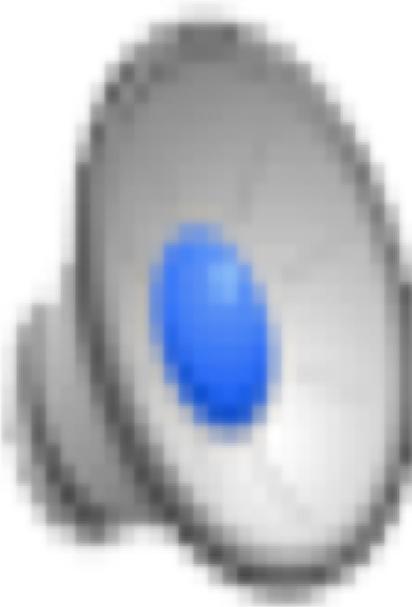
# رفتار هوشمندانه



[Demo: mystery pacman (L6D1)]

# رفتار هوشمندانه

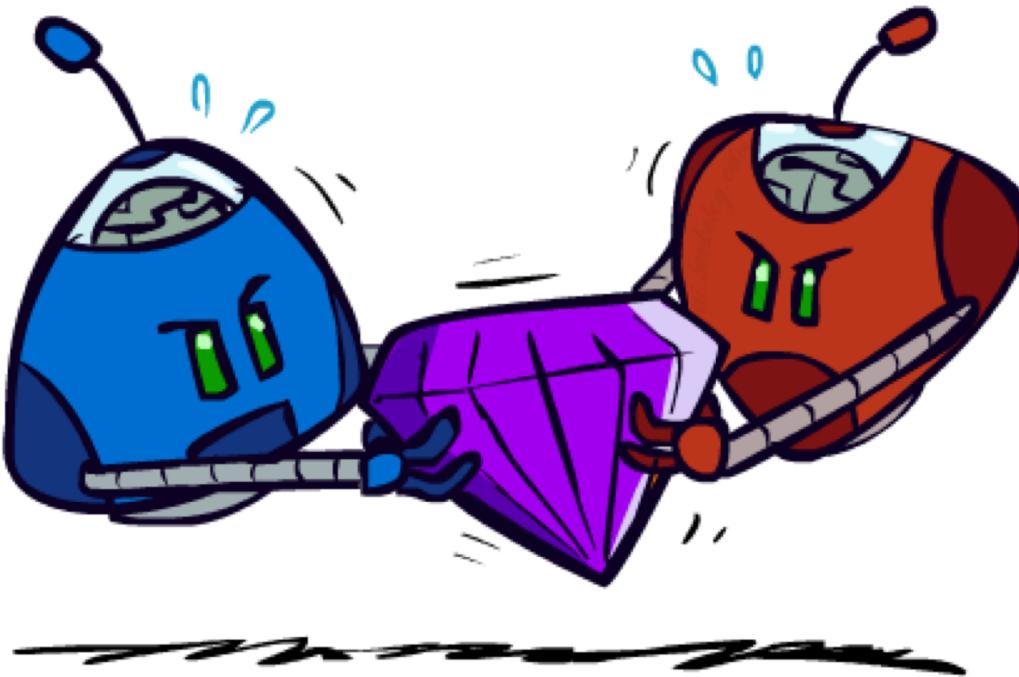
---



pacman-mystery.mp4  
ویدئو

# بازی‌های رقابتی

---



# انواع بازی



- Zero sum?

- قطعی یا تصادفی؟

- چند بازیکن؟ یک، دو، سه یا...؟

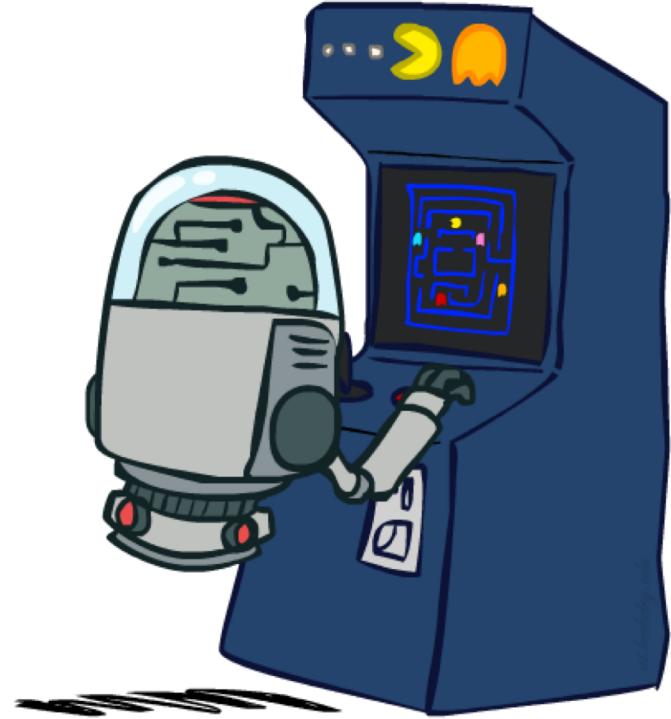
- کاملاً قابل مشاهده؟

دنبال الگوریتمی هستیم که استراتژی بازی را برای ما محاسبه کند:  
بهترین حرکت چیست؟

# بازی‌های قطعی

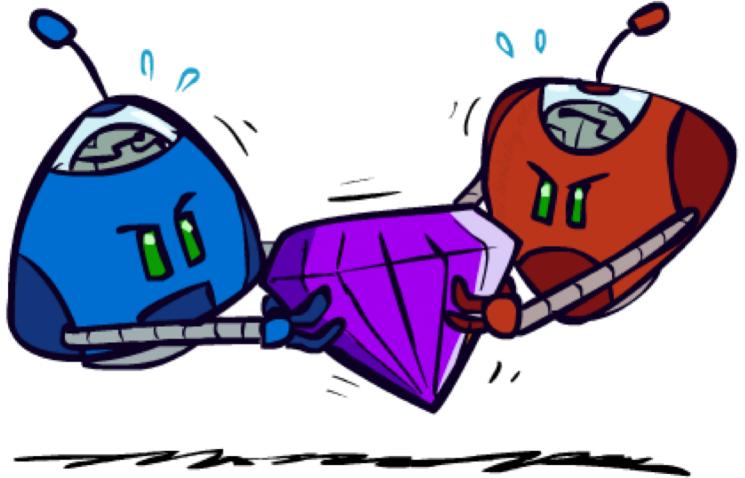
- Many possible formalizations, one is:

- حالات:  $S$  (start at  $s_0$ )
- بازیکن‌ها:  $P=\{1\dots N\}$  (usually take turns)
- Actions:  $A$  (may depend on player / state)
- Transition Function:  $S \times A \rightarrow S$
- Terminal Test:  $S \rightarrow \{t, f\}$
- Terminal Utilities:  $S \times P \rightarrow R$



راه حل یک **policy** است که به بازیکن بگوید در  $S$  کدام  $A$  را بکار گیرد

# بازی‌های Zero-Sum



## بازی‌های Zero-Sum

عامل‌ها منافع متناظر دارند  
یک مقدار کلی وجود دارد که یکی قصد دارد آن را زیاد کند و  
دیگری کم  
کاملاً رقابتی

## بازی‌های کلی

عامل‌ها منافع مستقل از هم‌دیگر دارند  
حالات ممکن: همکاری، رقابت، بازی جدآگانه و...

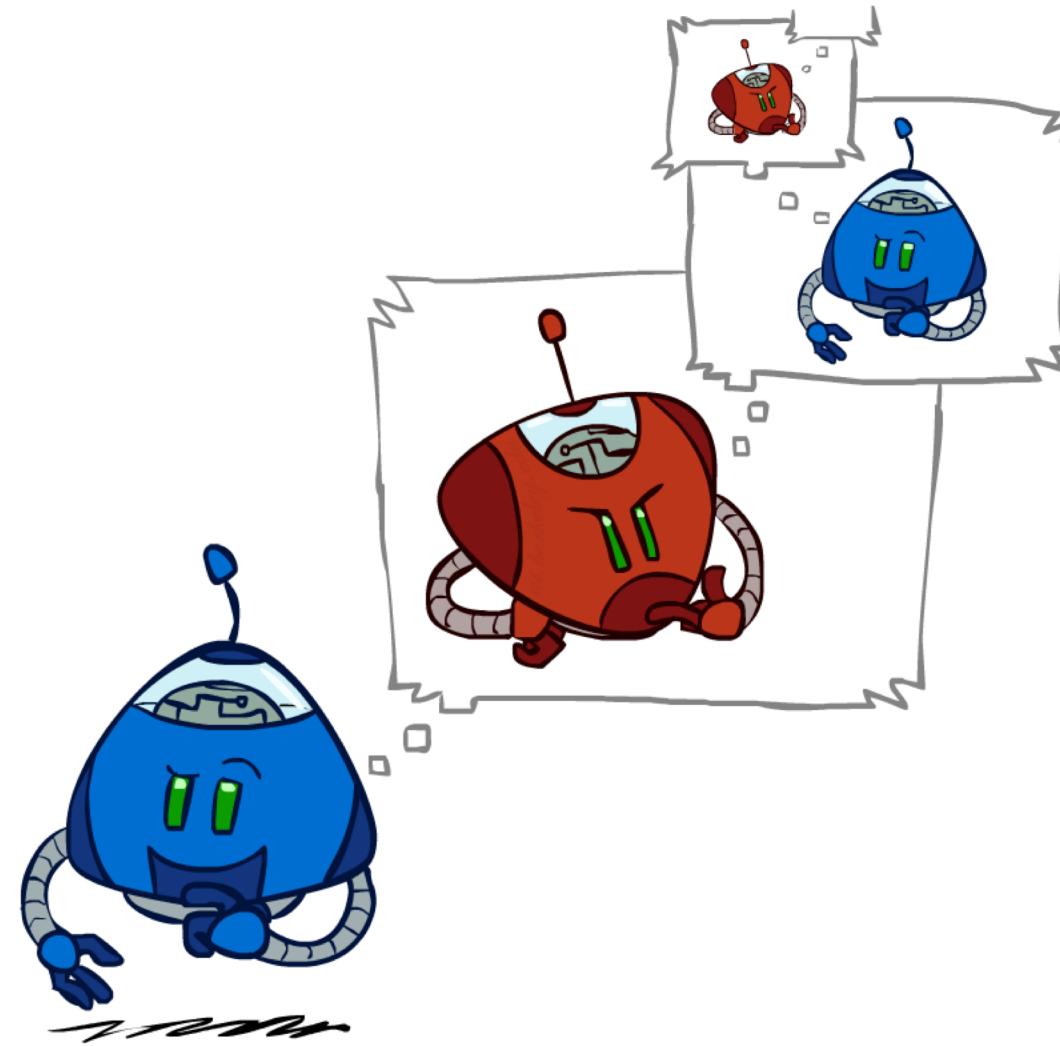
# مثال

		Blue	A	B	C
		Red	-30	10	-20
		1	30	-10	20
2	1	-10	10	-20	20
	2	20	-20	-20	20

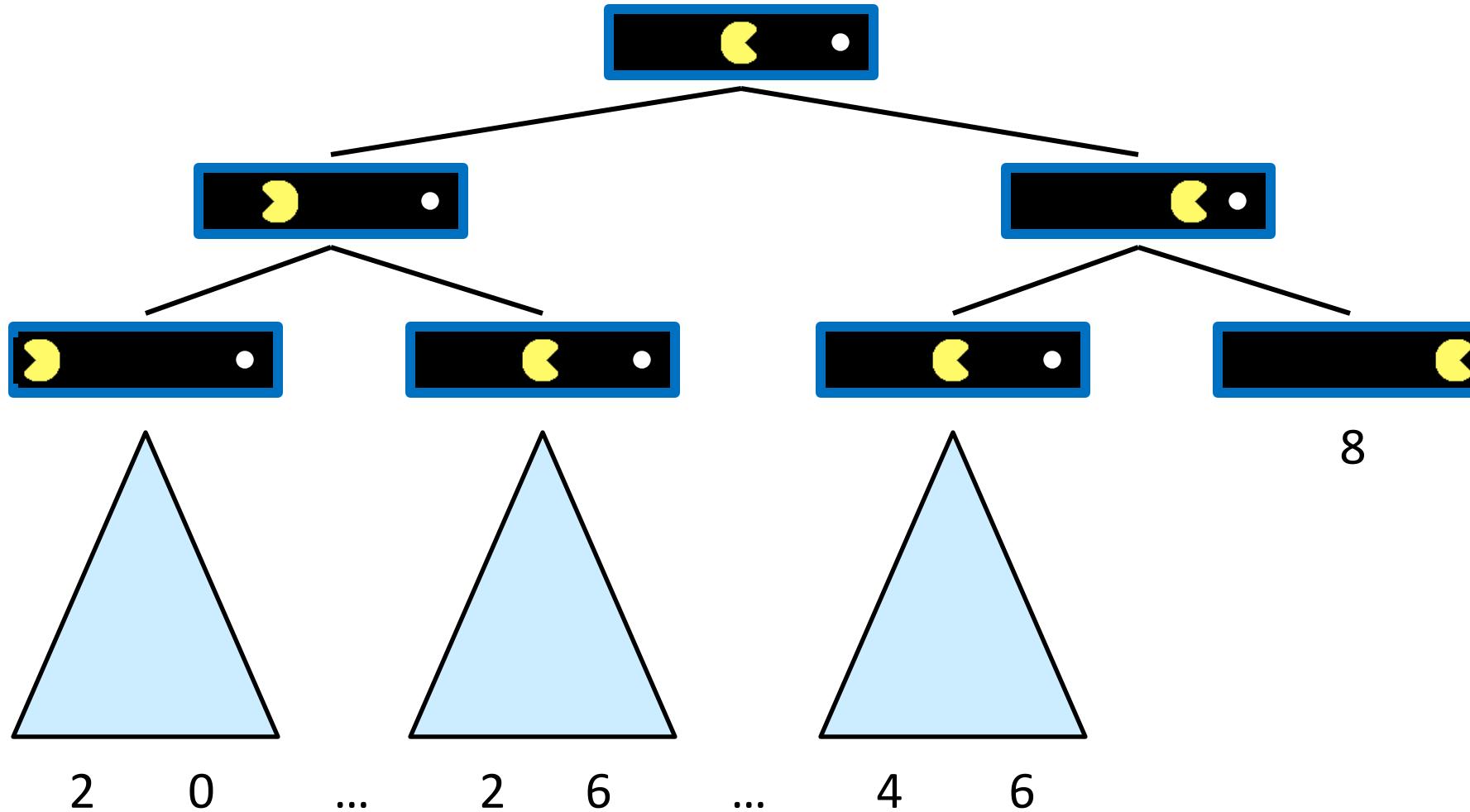
*A zero-sum game*

# جستجوی رقابتی

---

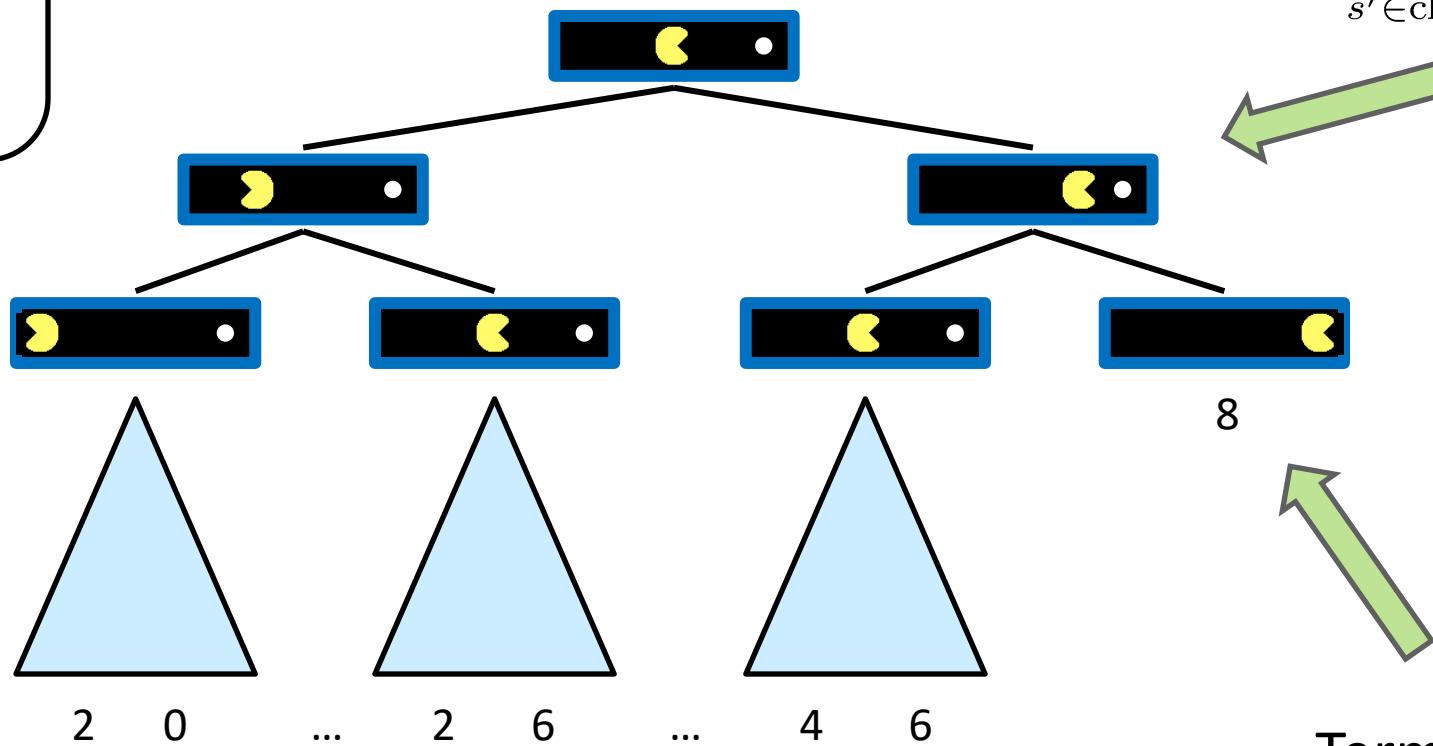


# درخت تک عاملی



# ارزش هر حالت

ارزش هر حالت:  
بهترین نتیجه‌ای که  
ممکن است از این  
حالت به دست آید



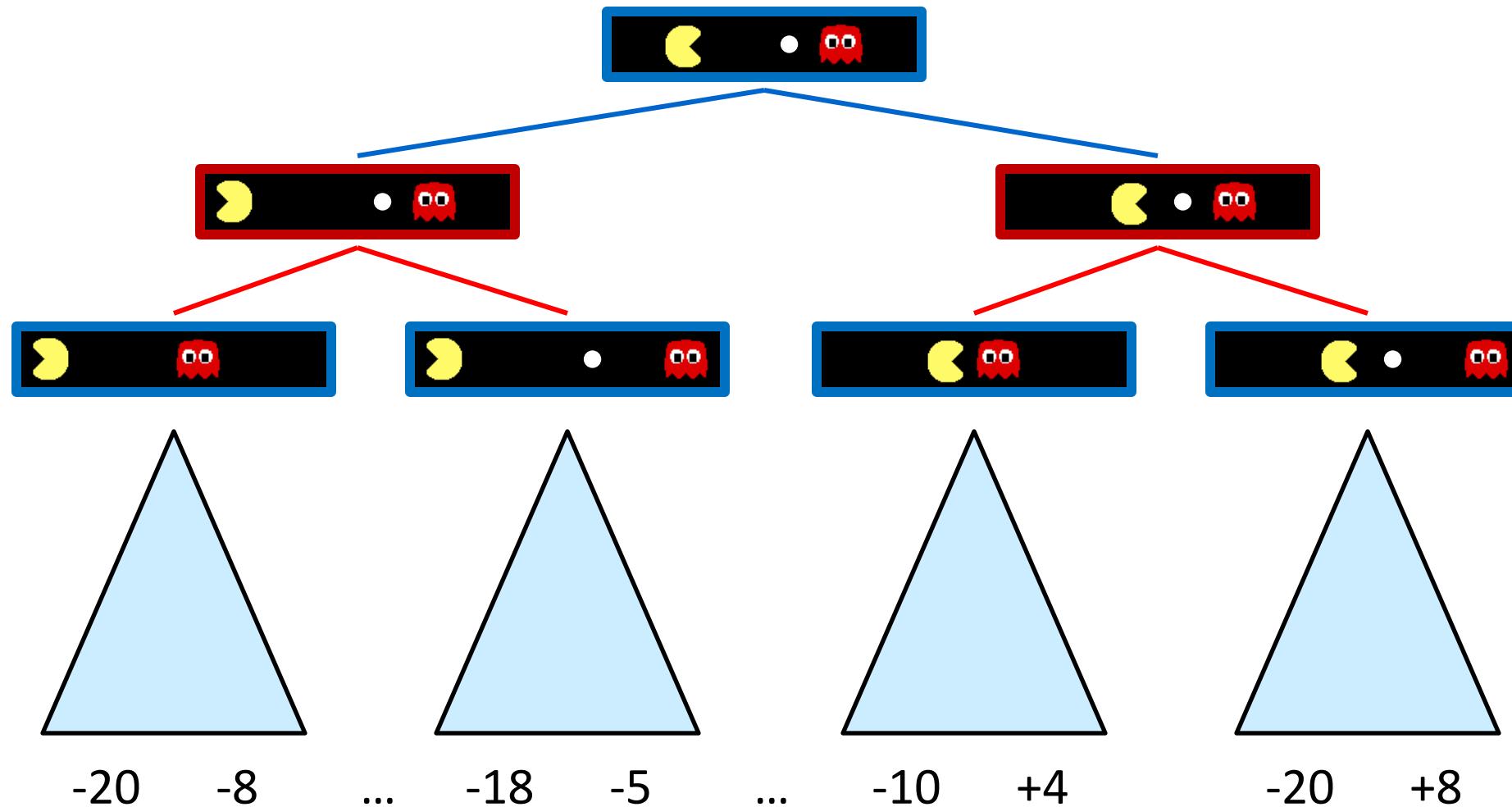
Non-Terminal States:

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

Terminal States:

$$V(s) = \text{known}$$

# درخت بازی رقابتی



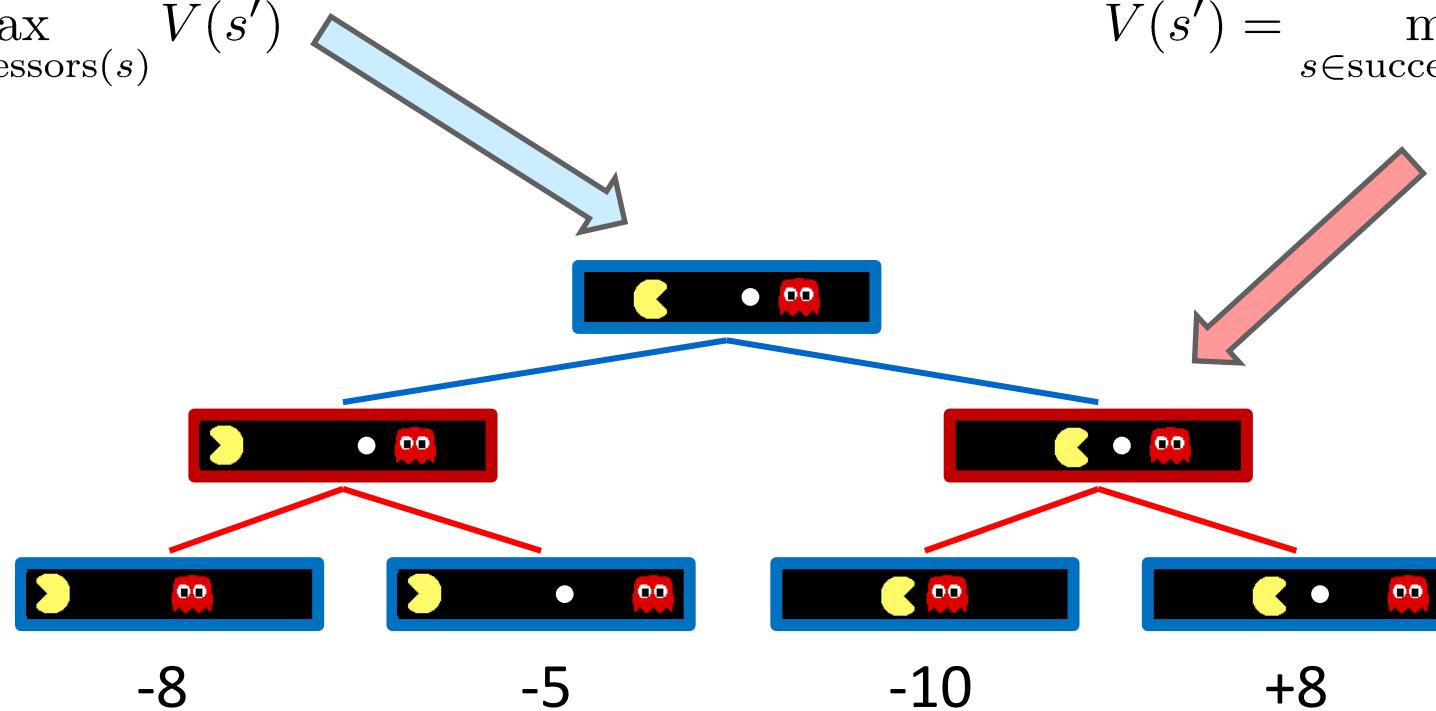
# مقدیر Minimax

حالت‌های تحت کنترل عامل

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

حالت‌های تحت کنترل حریف

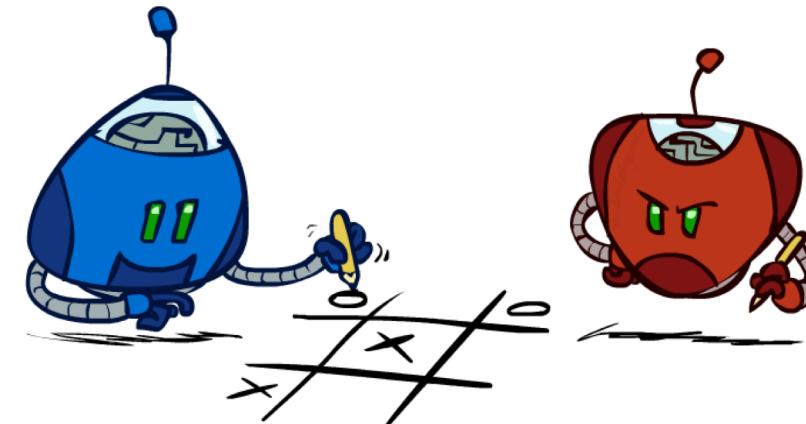
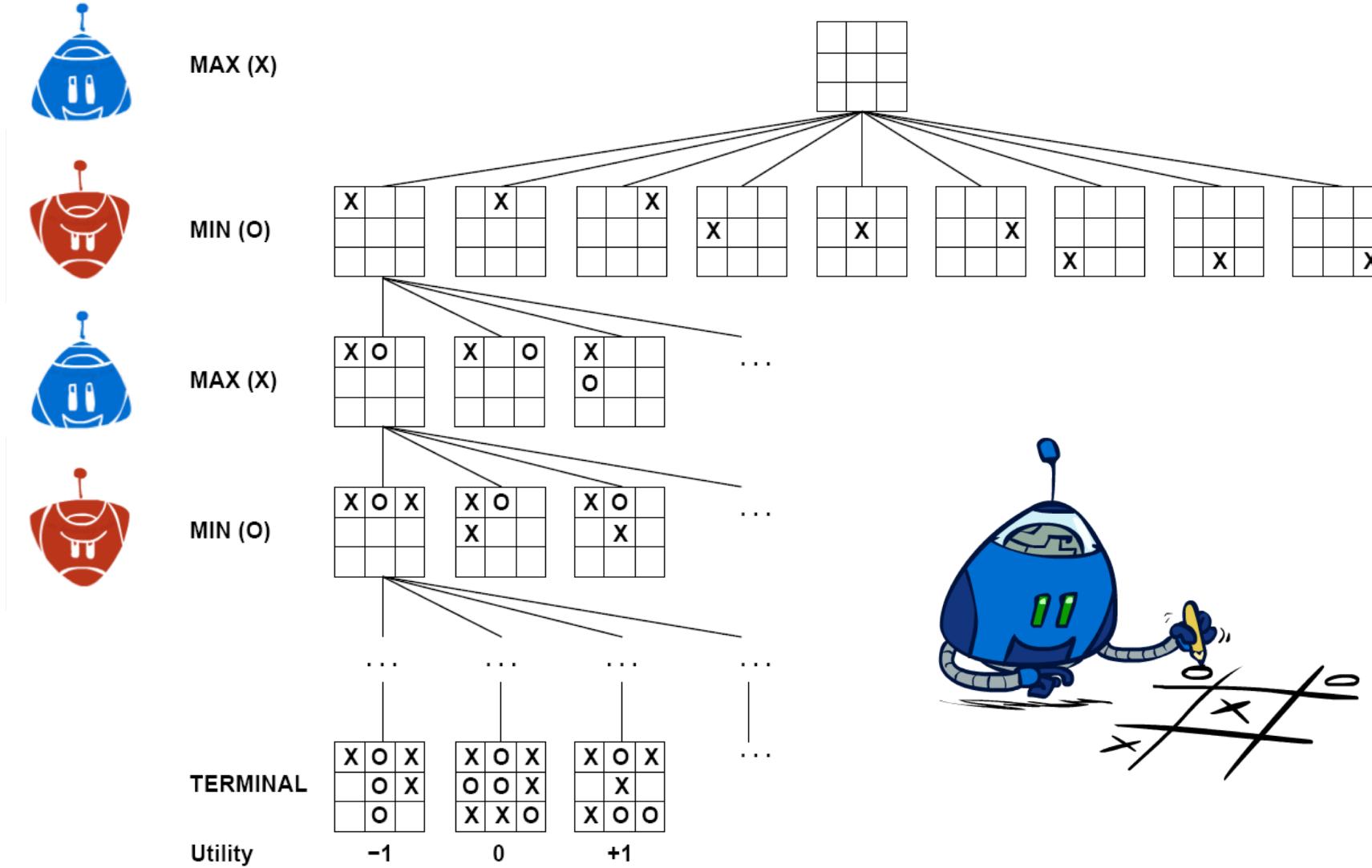
$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$



Terminal States:

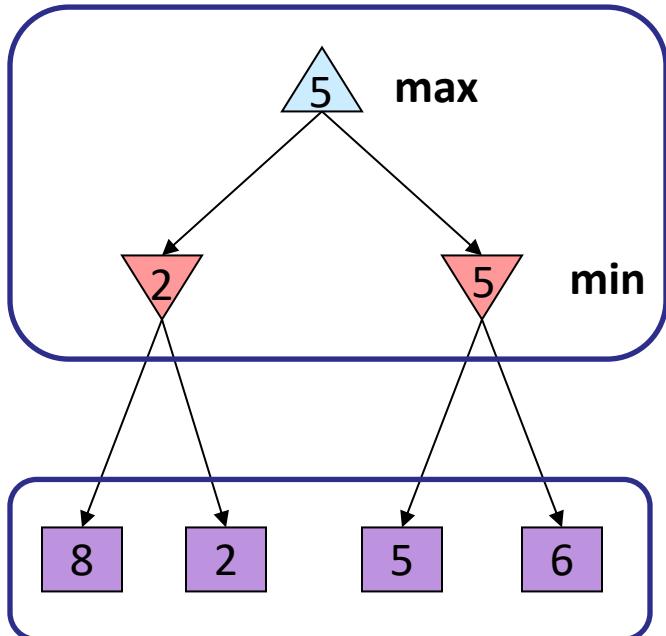
$$V(s) = \text{known}$$

# درخت بازی Tic-Tac-Toe



# جستجوی رقابتی - Minimax

Minimax values:  
computed recursively



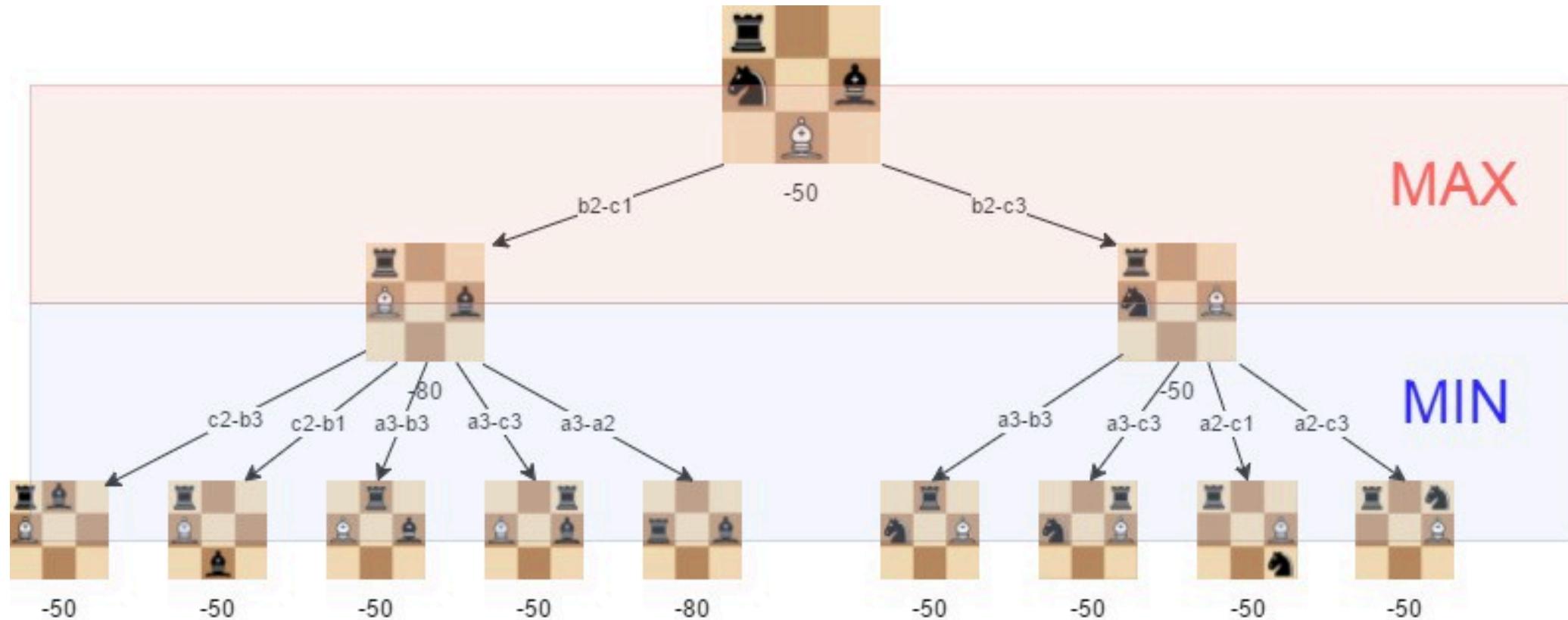
Terminal values:  
part of the game

بازی‌های zero-sum قطعی  
Tic-tac-toe, chess, checkers

یک عامل سعی در ماقسیمایز کردن نتیجه دارد  
دیگری سعی در مینیمایز کردن

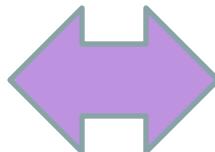
جستجوی Minimax  
عامل‌ها نوبتی بازی می‌کنند  
ارزش Minimax هر گره:  
بهترین نتیجه قابل حصول از این گره، در برابر یک  
رقیب هوشمند (بهینه)

# مثال Minimax



# پیاده‌سازی Minimax

```
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
```

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

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

# پیاده‌سازی Minimax

```
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)

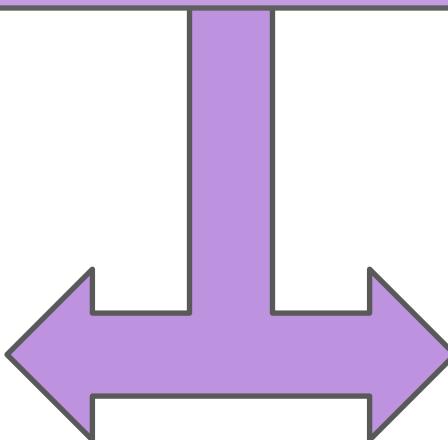
```
def max-value(state):
```

initialize  $v = -\infty$

for each successor of state:

$v = \max(v, \text{value}(\text{successor}))$

return  $v$



```
def min-value(state):
```

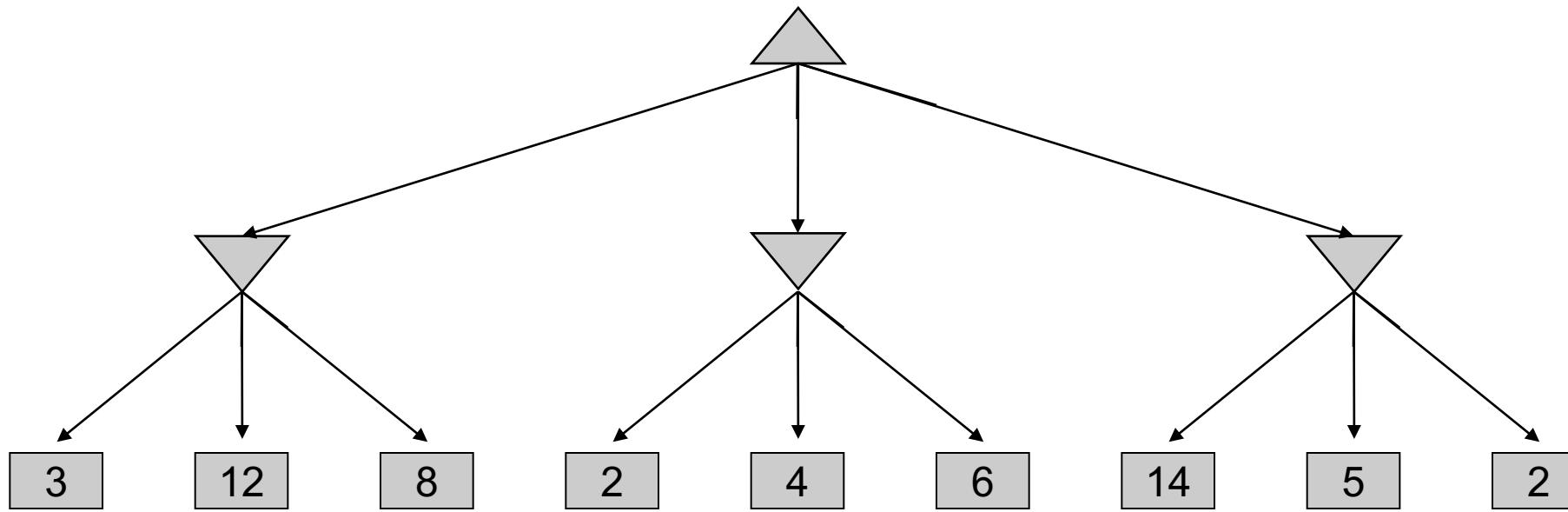
initialize  $v = +\infty$

for each successor of state:

$v = \min(v, \text{value}(\text{successor}))$

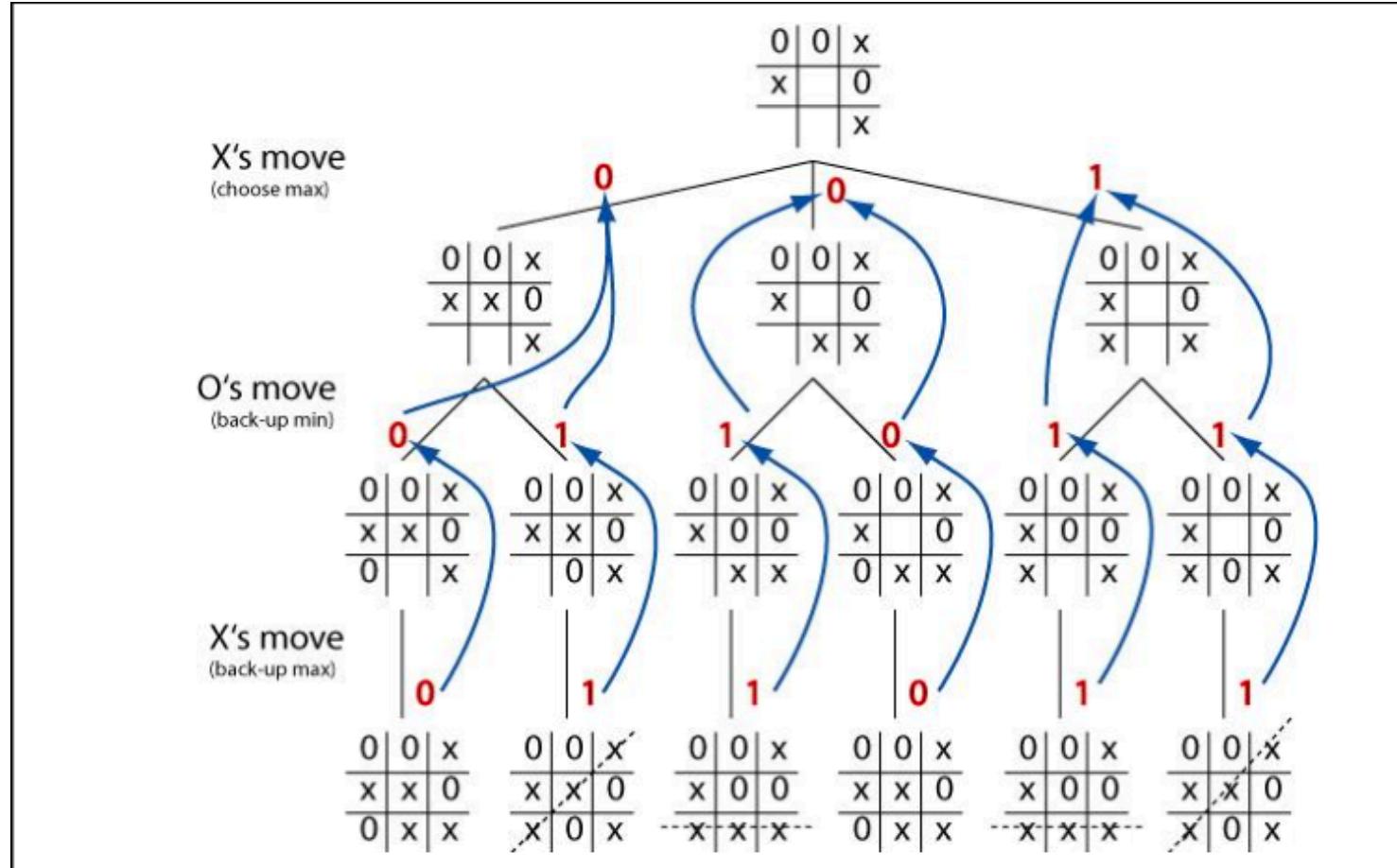
return  $v$

# مثال Minimax



# تمرین Minimax

▪ بهترین استراتژی را بیابید! (نوبت شماست که بازی کنید، شما X هستید)



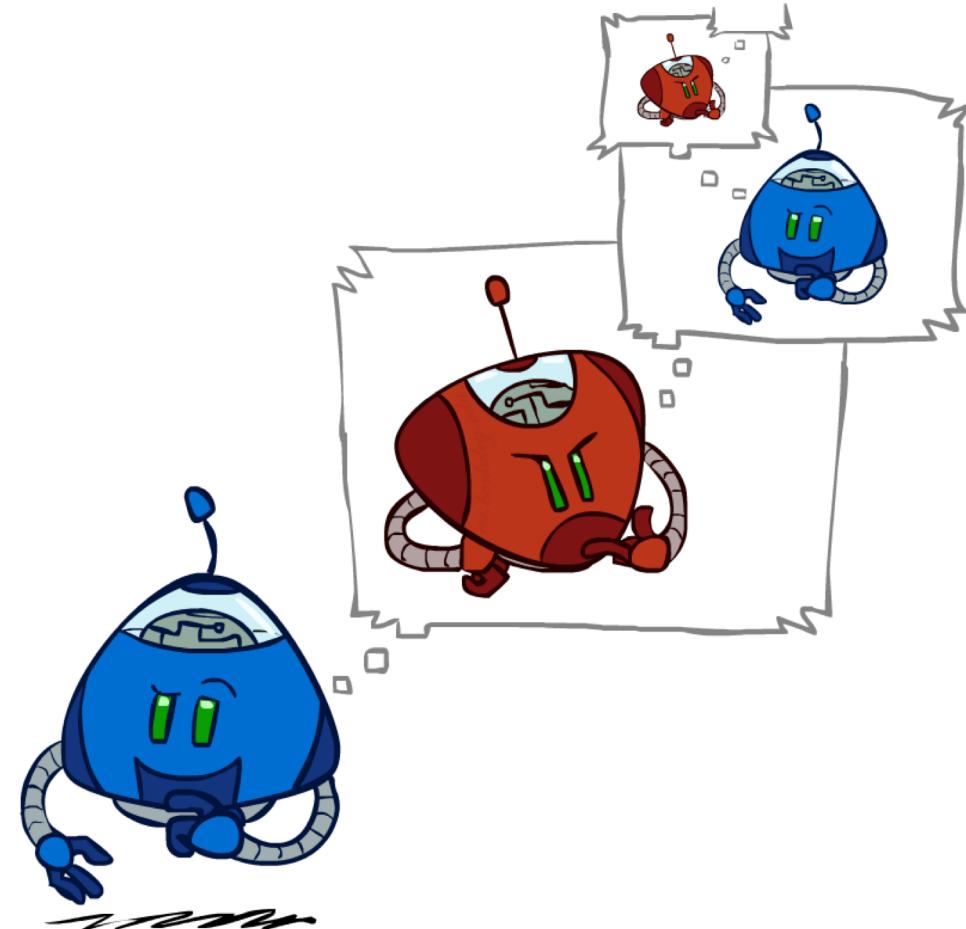
# عملکرد Minimax

- How efficient is minimax?

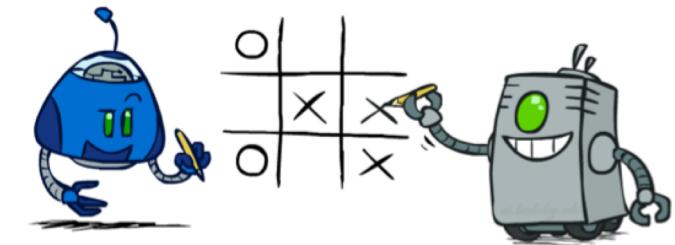
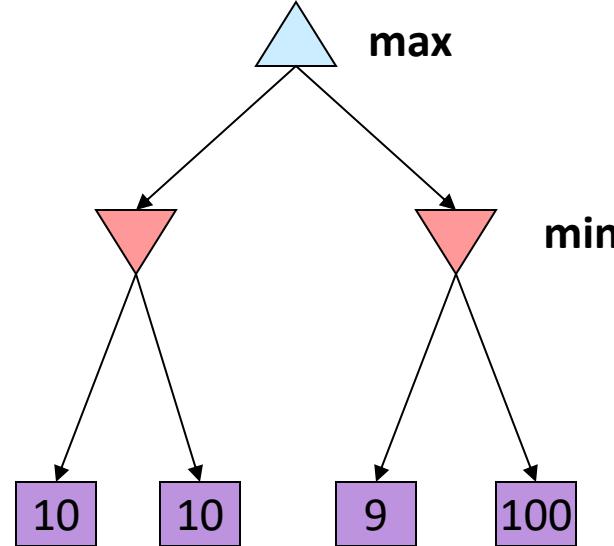
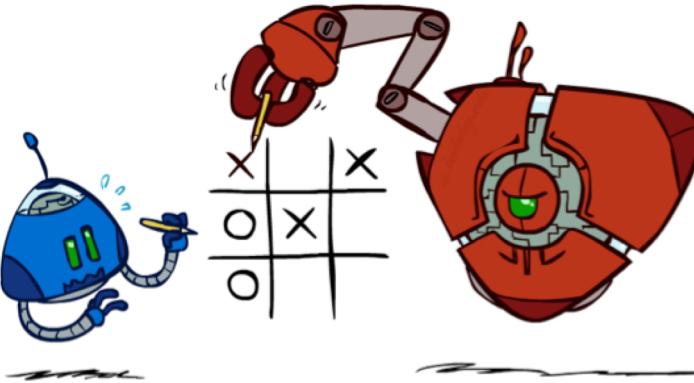
- Just like (exhaustive) DFS
- Time:  $O(b^m)$
- Space:  $O(bm)$

مثال: برای شطرنج  $b \approx 35, m \approx 100$

دستیابی به جواب ناممکن است  
ولی، آیا باید تمام درخت را جستجو کنیم؟



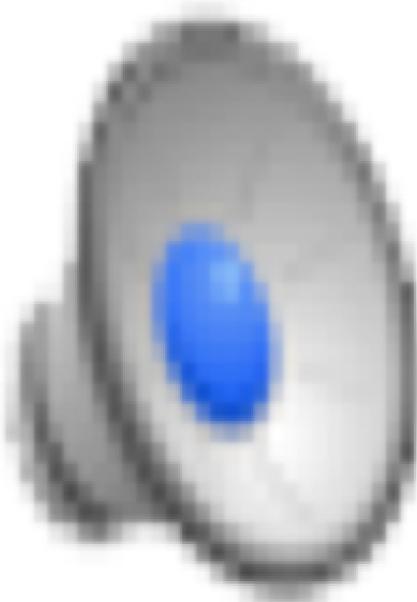
# ویژگی‌های Minimax



بهینه در صورتی که حریف بی اشتباه باشد!  
اگر حریف ضعیفی داشتیم چطور؟

# مثال: حریف بی اشتباه در برابر حریف خطدار

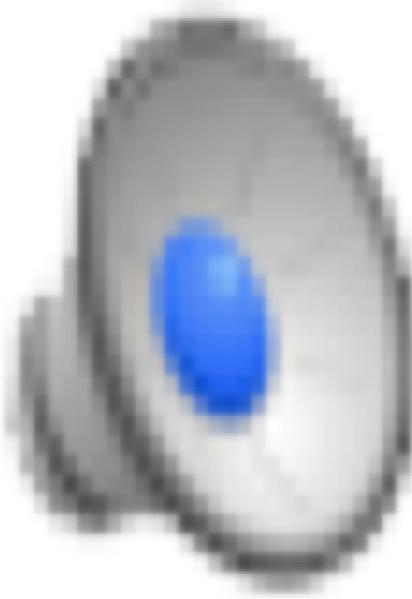
---



min-vs-exp-(min).mp4 ویدئو

# مثال: حریف بی اشتباه در برابر حریف خطدار

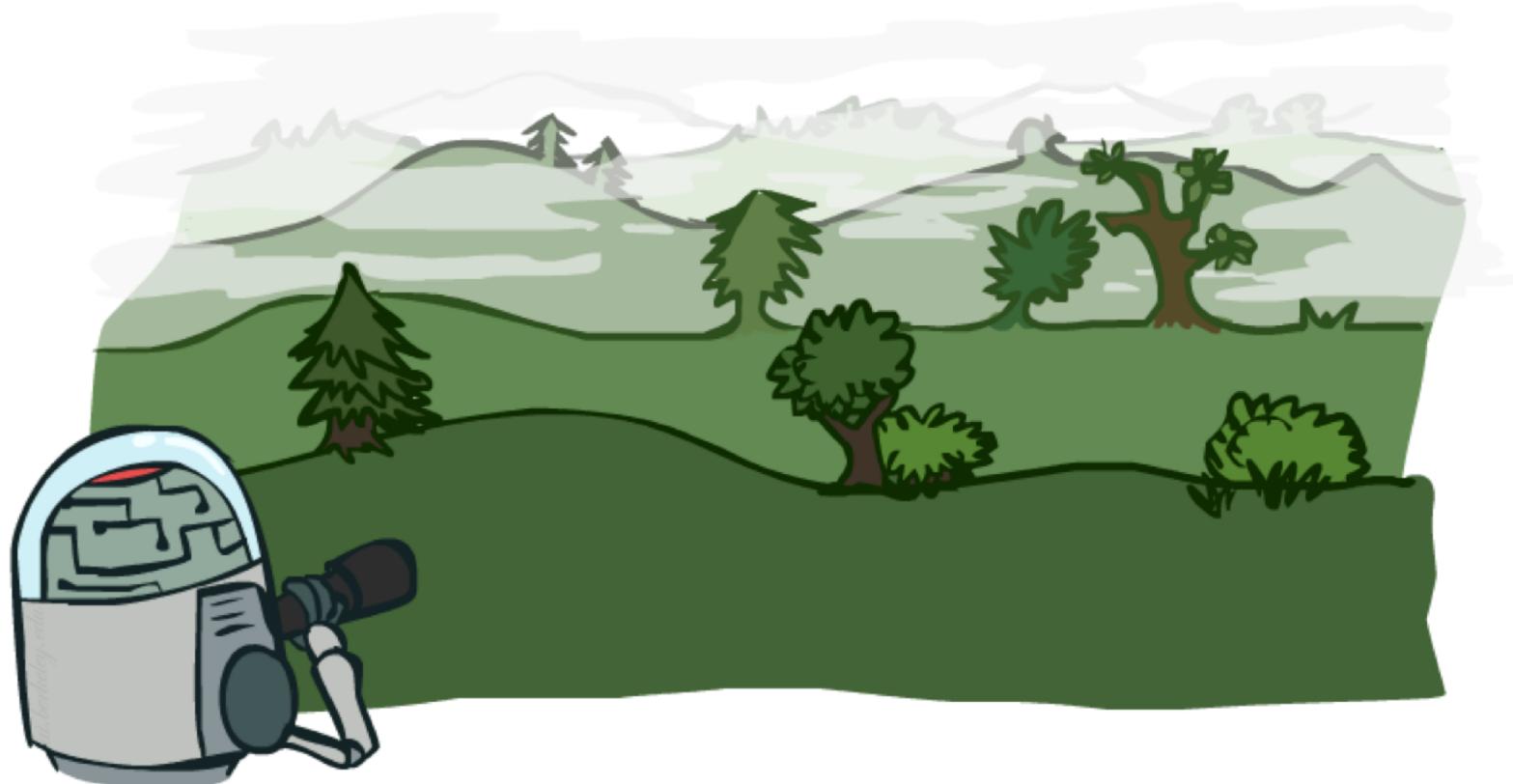
---



ویدئو 4  
min-vs-exp-(exp).mp4

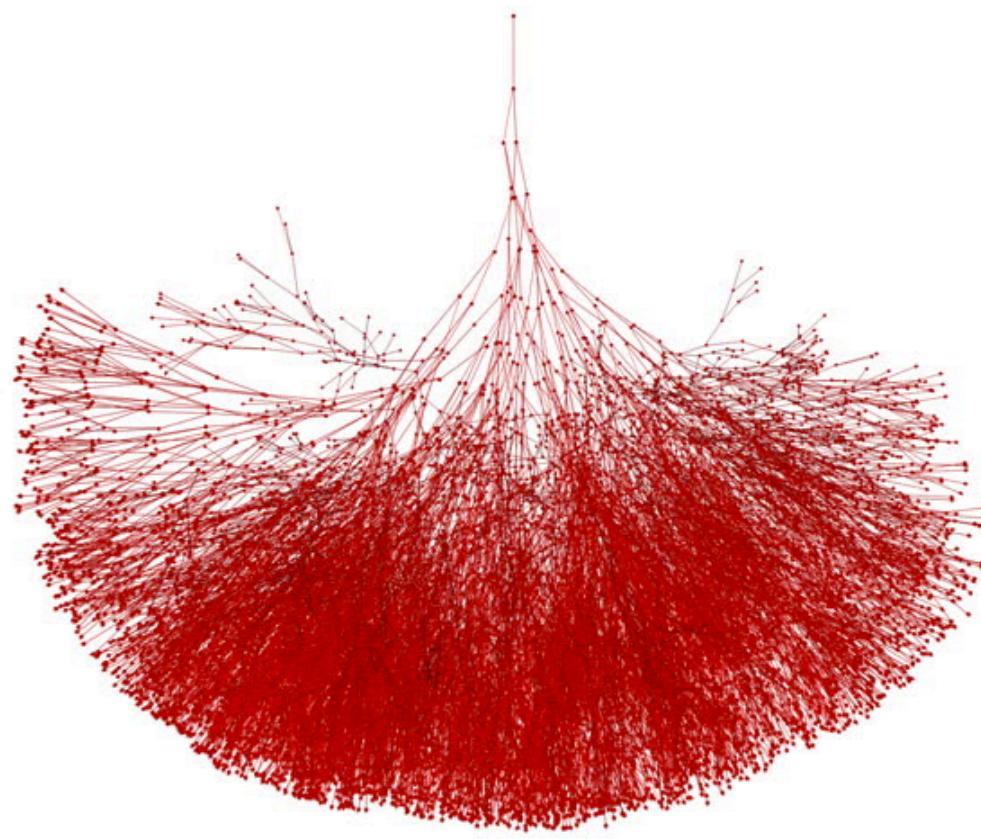
# محدودیت‌های منابع

---

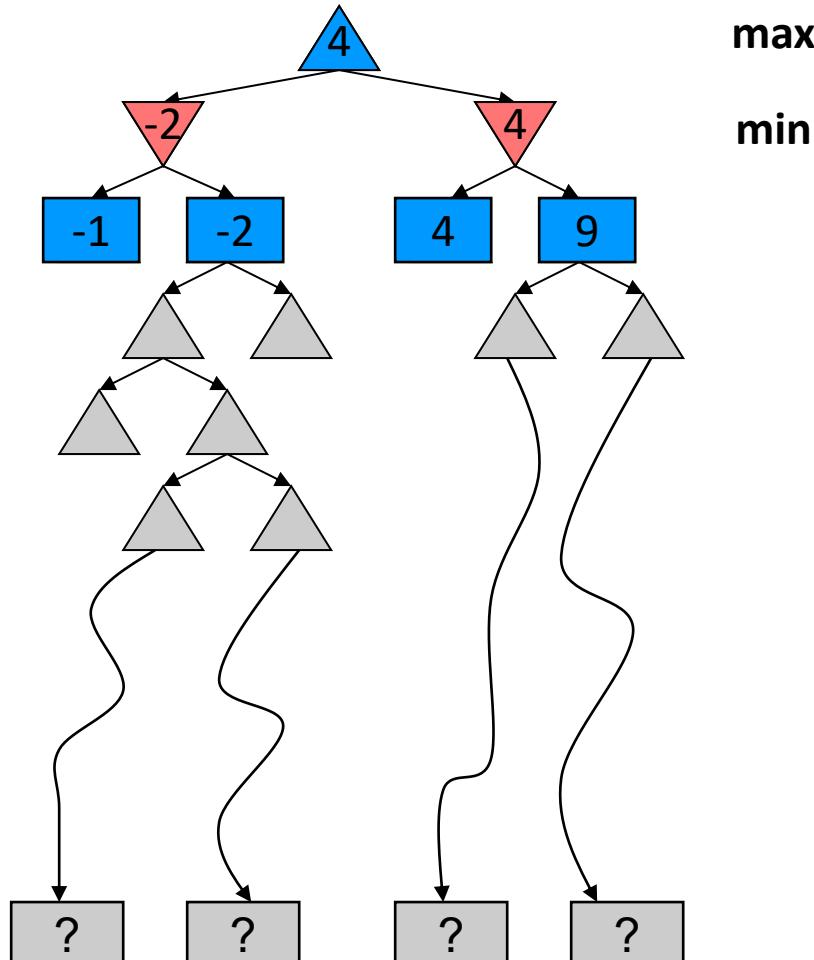


# محدودیت‌های منابع

مشکل: در بازی‌های پیچیده‌تر (مثلاً شطرنج)، نمی‌توان تا برگ پیش رفت!



# محدودیت‌های منابع



مشکل: در بازی‌های پیچیده‌تر (مثلا شطرنج)، نمی‌توان تا برگ پیش رفت!

راه حل: عمق محدود

مقدار کمتری در عمق گراف حرکت کن  
مقادیر واقعی را با یک تابع ارزیابی (که تخمینی از خوبی گره می‌زند) جانشین کن

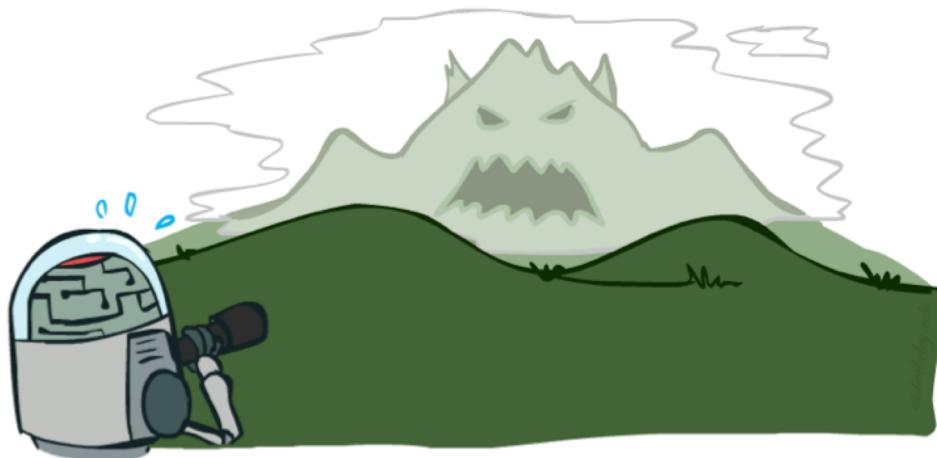
مثال:

- Suppose we have 100 seconds, can explore 10K nodes / sec
- So can check 1M nodes per move
- at about depth 8 – decent chess program

دیگر تضمینی برای بهینگی نداریم!

عمقی که بررسی می‌کنیم اهمیت خیلی زیادی دارد

# اهمیت عمق



تابع ارزیابی بی خطا نیستند

هر چقدر به هدف نزدیکتر باشیم (عمق بیشتر باشد)، اهمیت تابع ارزیابی کمتر می شود

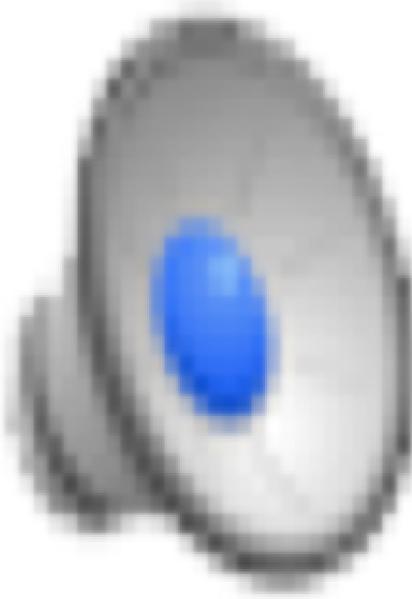


Tradeoff:

تابع ارزیابی بهتر، محاسبه بیشتر

# مثال: اهمیت تابع ارزیابی - پکمن عادی

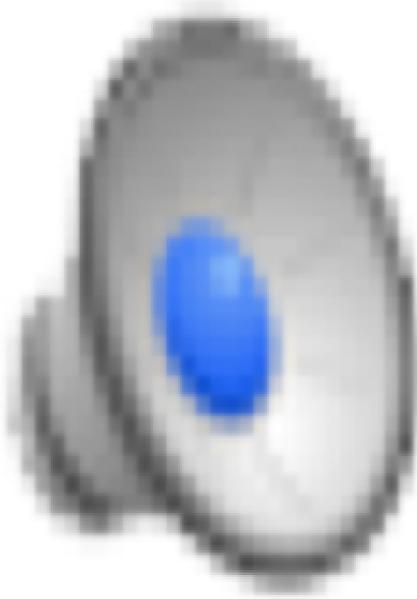
---



ویدئو 4.mp4

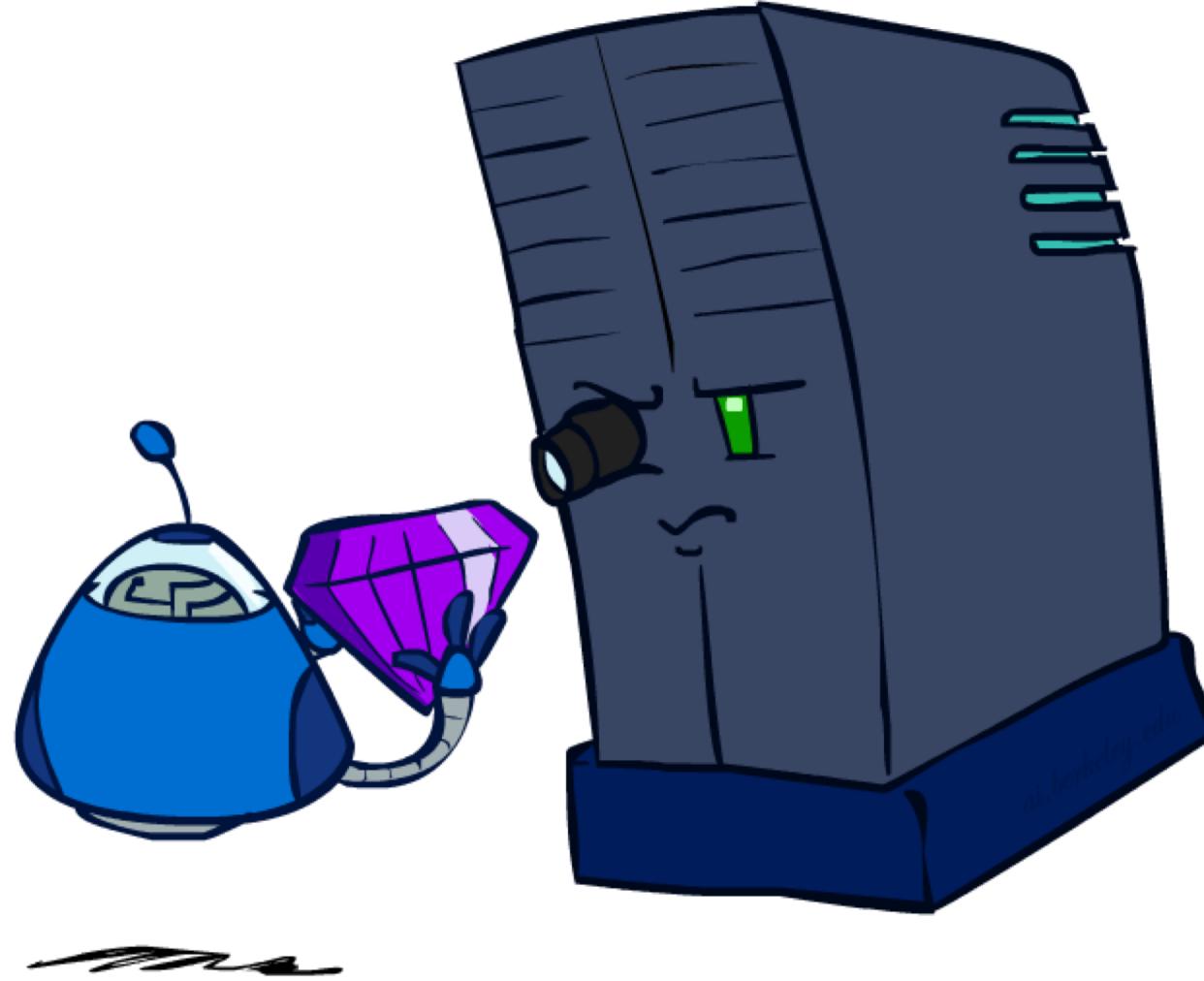
# مثال: اهمیت تابع ارزیابی - پکمن ناچه

---



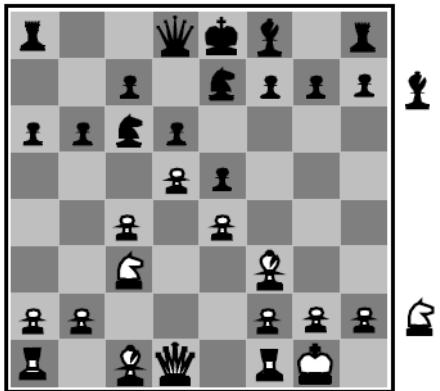
Limited-depth-2.mp4  
ویدئو

# تابع ارزیابی - Evaluation Function



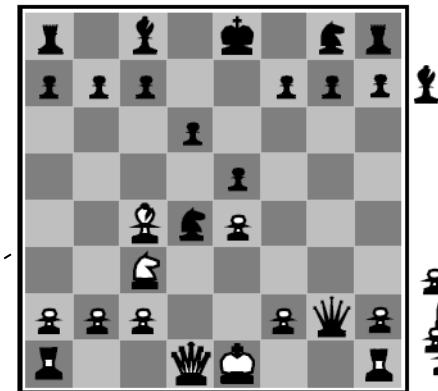
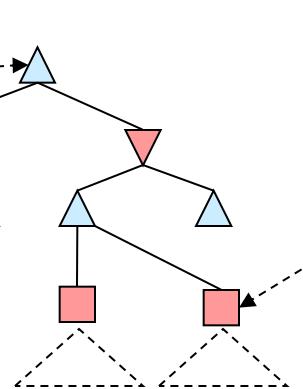
# تابع ارزیابی

به گرههای میانی در درخت امتیاز می‌دهد



Black to move

White slightly better



White to move

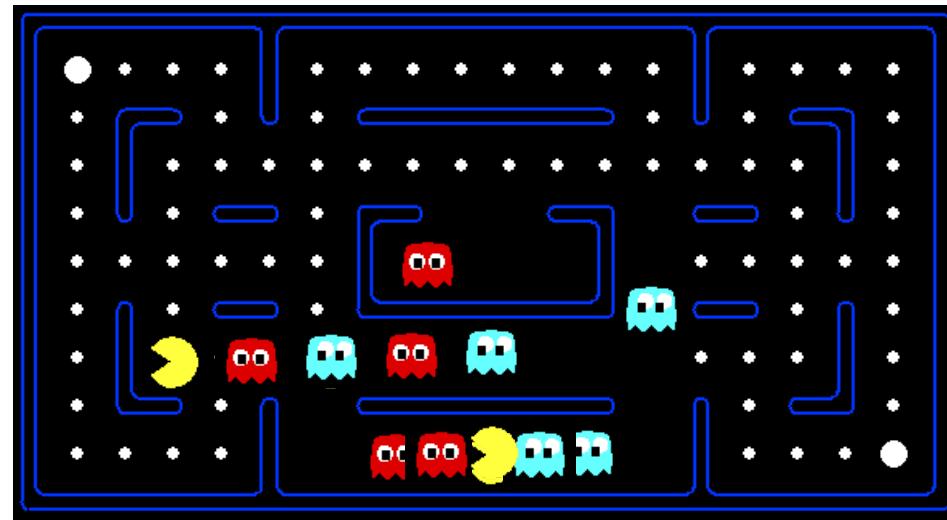
Black winning

تابع ایدهآل: مقدار واقعی minimax هر گره را بدهد  
در عمل: جمع وزنداری از feature های مختلف

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

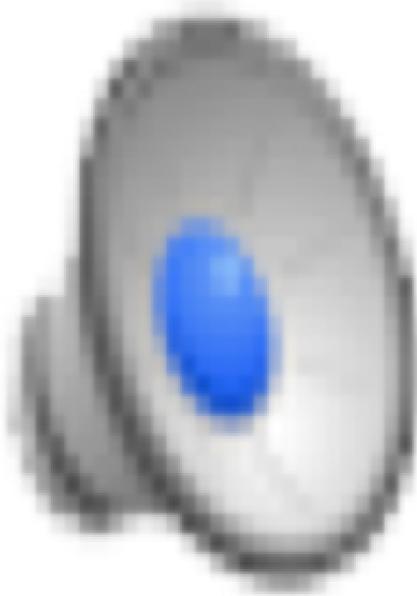
e.g.  $f_1(s) = (\text{num white queens} - \text{num black queens})$ , etc.

# تابع ارزیابی برای پکمن



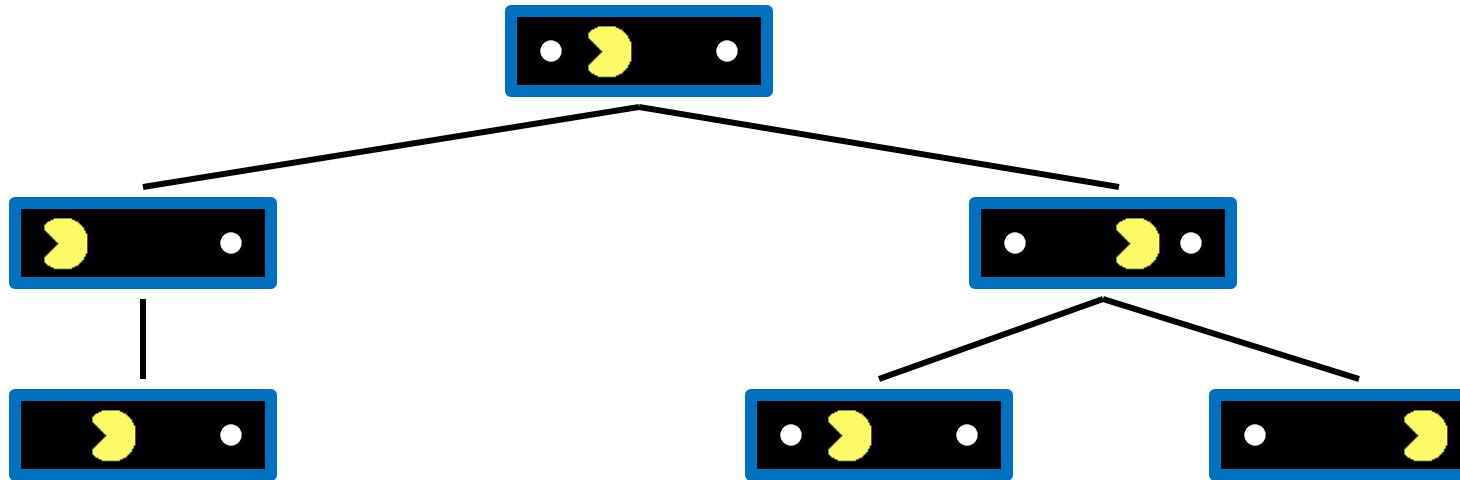
# مثال: تابع ارزیابی ضعیف

---



Thrashing.mp4  
ویدئو

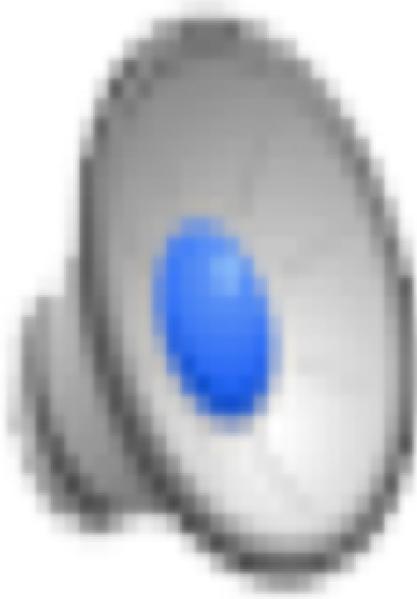
# مشکل پکمن کجا بود؟



- A danger of replanning agents!
  - He knows his score will go up by eating the dot now (west, east)
  - He knows his score will go up just as much by eating the dot later (east, west)
  - There are no point-scoring opportunities after eating the dot (within the horizon, two here)
  - Therefore, waiting seems just as good as eating: he may go east, then back west in the next round of replanning!

# مثال: تابع ارزیابی بهبود داده شده

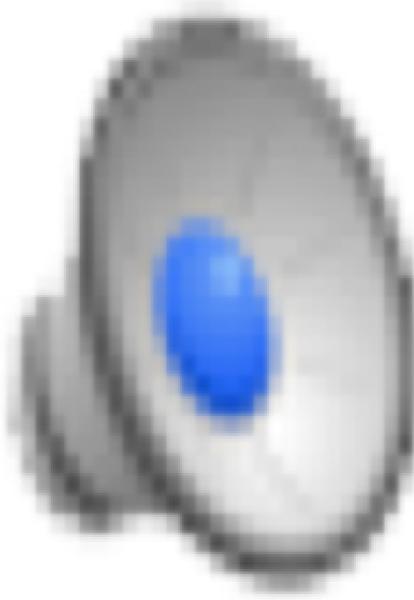
---



Thrashing-fixed.mp4  
ویدئو4

# مثال: غولهای باهوش - همکاری

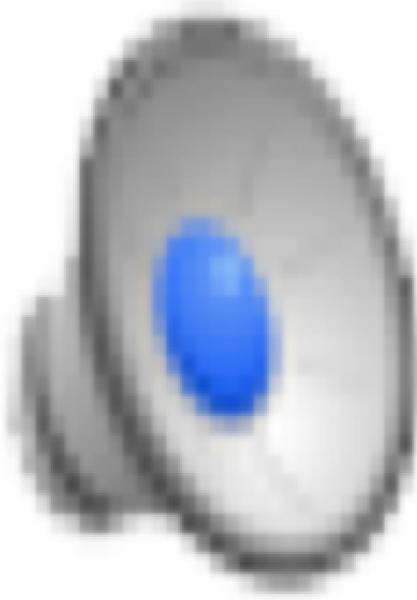
---



cooperation-1.mp4  
ویدئو

# مثال: غولهای باهوش - همکاری

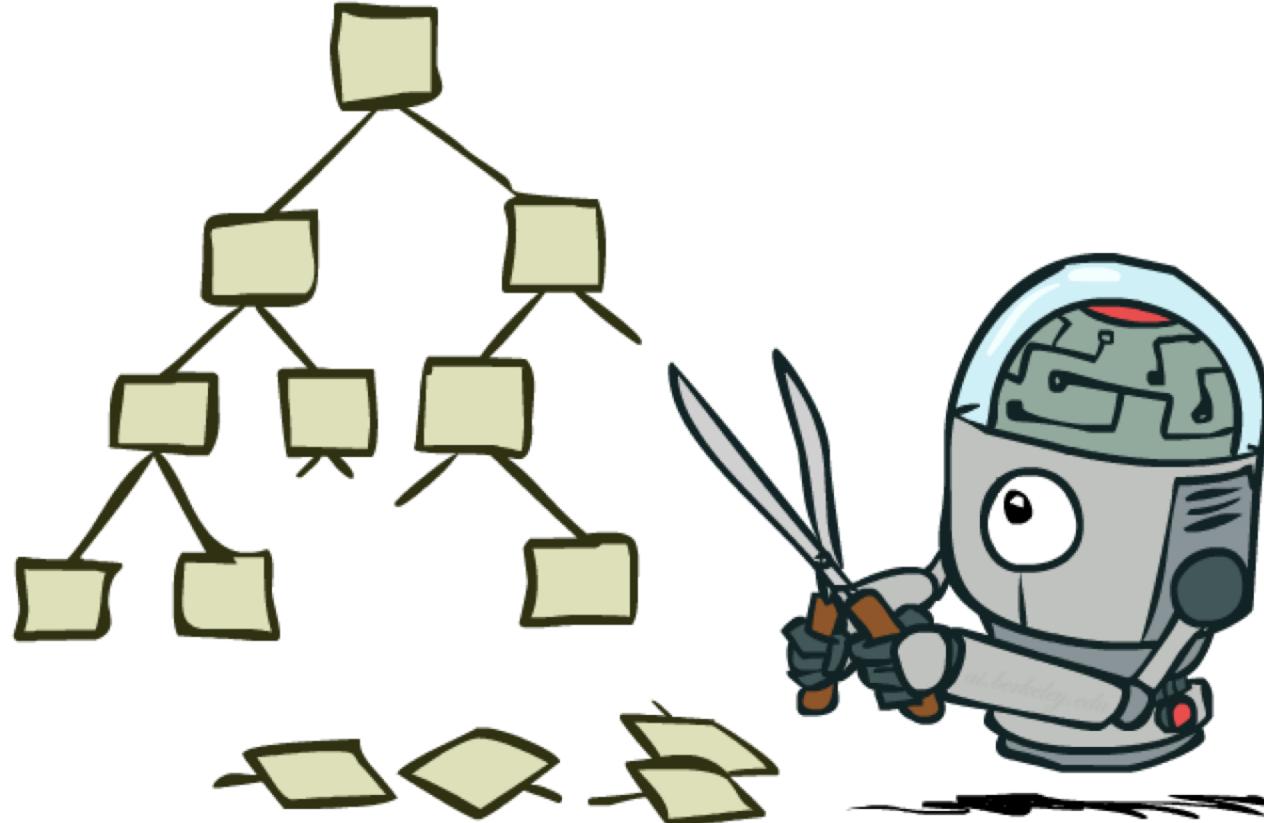
---



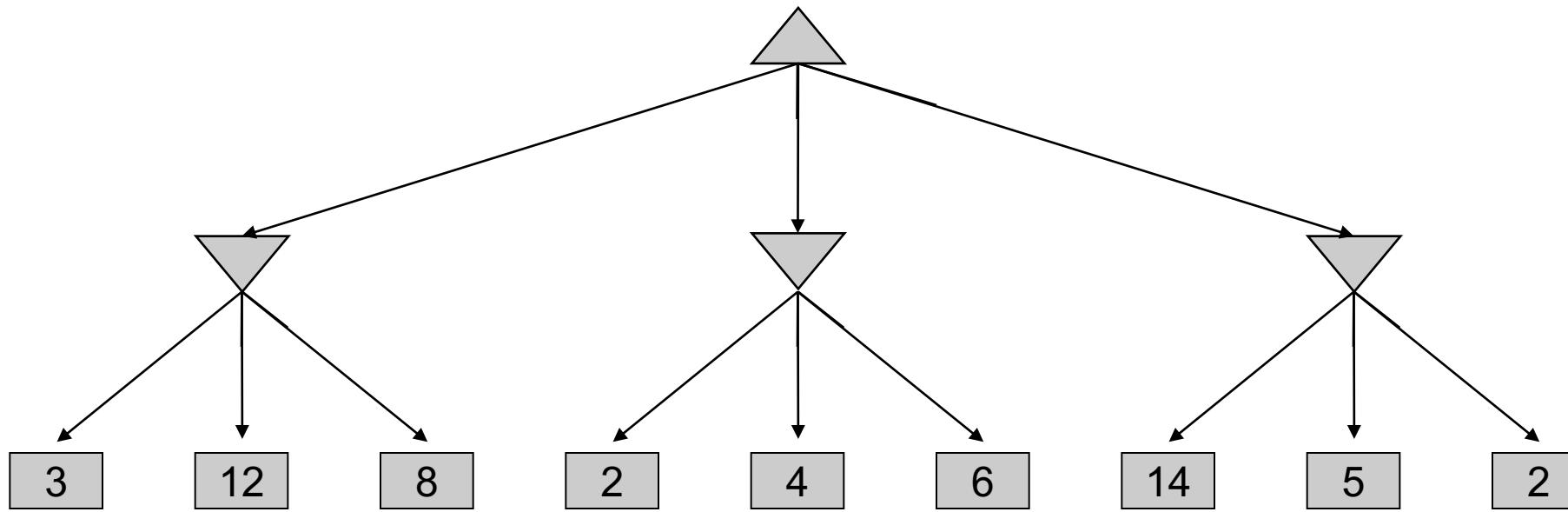
coordination-2.mp4

# هرس کردن درخت - Pruning

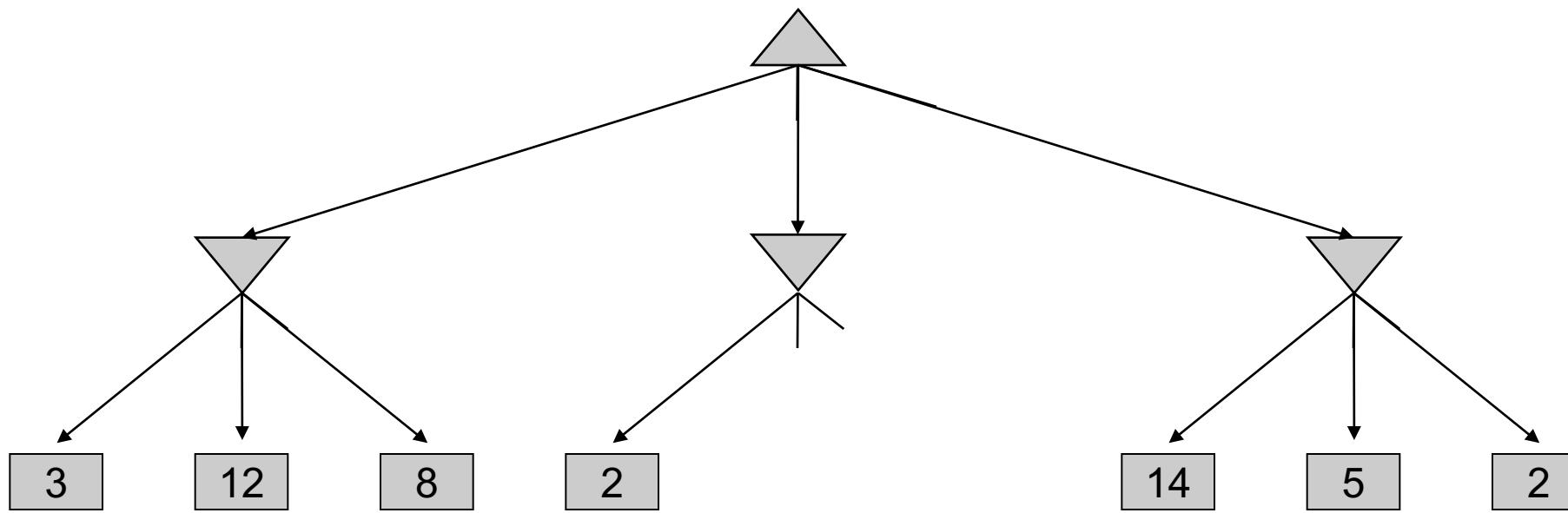
---



# مثال Minimax



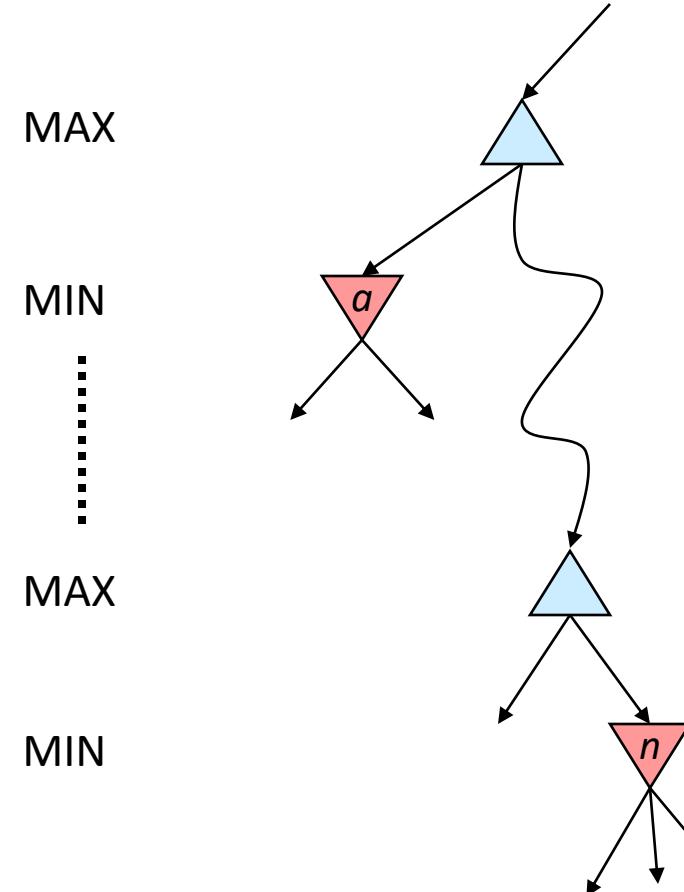
# هرس کردن Minimax



# روش هرس کردن Alpha-Beta

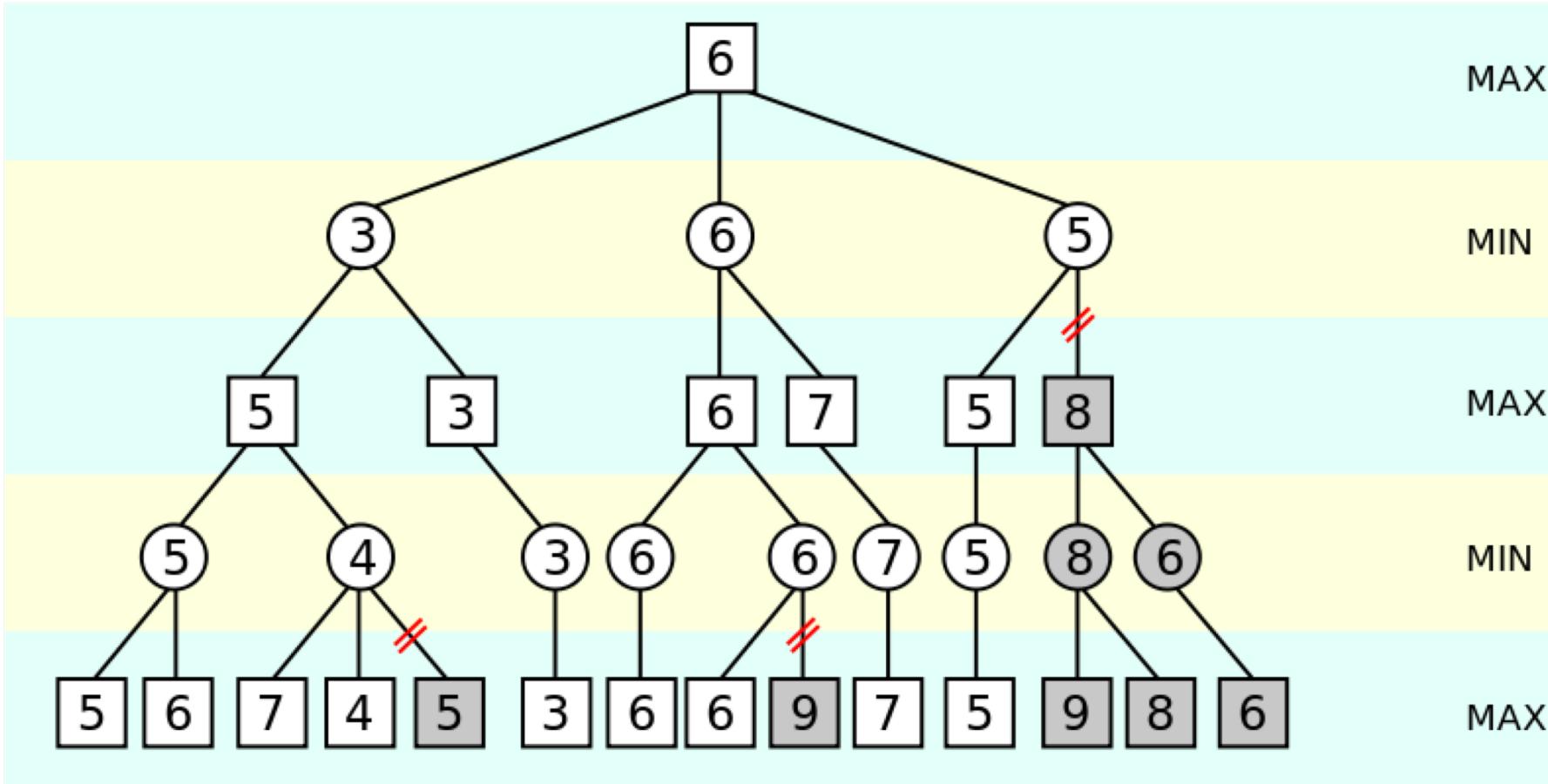
- General configuration (MIN version)

- 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  $a$  be the best value that MAX can get at any choice point along the current path from the root
- If  $n$  becomes worse than  $a$ , MAX will avoid it, so we can stop considering  $n$ 's other children (it's already bad enough that it won't be played)



- MAX version is symmetric

# روش هرس کردن



# پیاده‌سازی Alpha-Beta

```
function alphabeta(node, depth, α, β, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := -∞
        for each child of node do
            value := max(value, alphabeta(child, depth - 1, α, β, FALSE))
            α := max(α, value)
            if α ≥ β then
                break (* β cut-off *)
        return value
    else
        value := +∞
        for each child of node do
            value := min(value, alphabeta(child, depth - 1, α, β, TRUE))
            β := min(β, value)
            if α ≥ β then
                break (* α cut-off *)
        return value
```

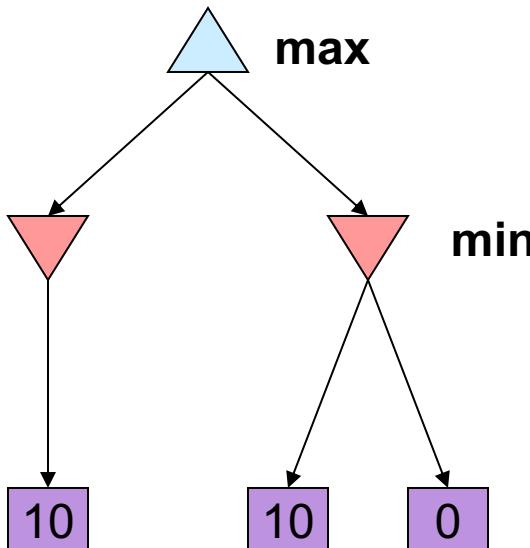
```
alphabeta(origin, depth, -∞, +∞, TRUE)
```

# ویژگی‌های الگوریتم هرس کردن Alpha-Beta

بر روی مقدار minimax گرهی روت تاثیری ندارد، چرا؟

مقادیر گره‌های میانی ممکن است دقیق نباشند

بنابراین، از مقادیر محاسبه شده در حالت ساده‌ی الگوریتم نمی‌توان برای انتخاب استراتژی استفاده کرد!



# ویژگی‌های الگوریتم هرس کردن Alpha-Beta

---

بر روی مقدار minimax گرهی روت تاثیری ندارد، چرا؟

مقادیر گره‌های میانی ممکن است دقیق نباشند

بنابراین، از مقادیر محاسبه شده در حالت ساده‌ی الگوریتم نمی‌توان برای انتخاب استراتژی استفاده کرد!

- Good child ordering improves effectiveness of pruning

# ویژگی‌های الگوریتم هرس کردن Alpha-Beta

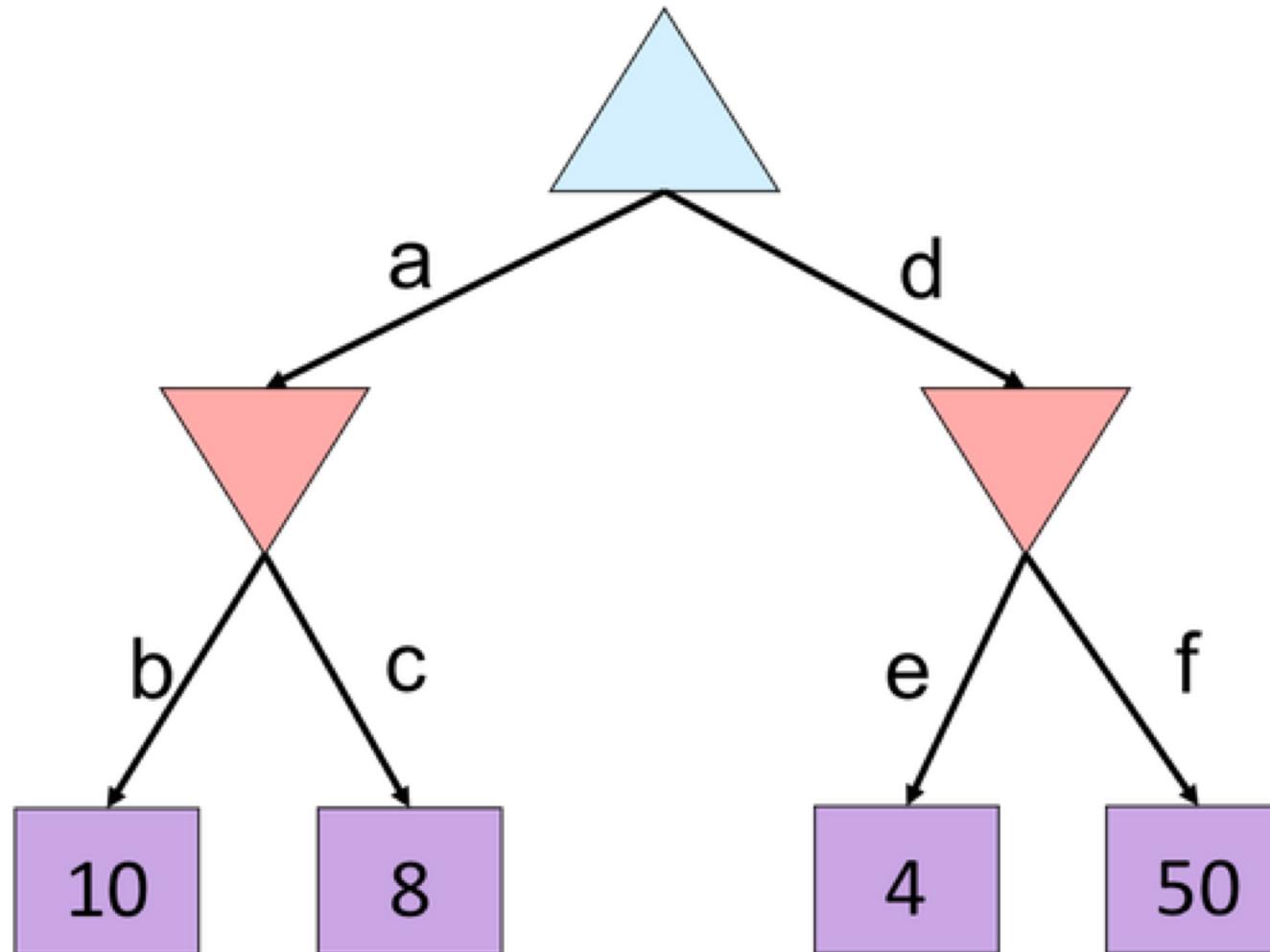
بر روی مقدار  $\text{minimax}$  گرهی روت تاثیری ندارد، چرا؟

مقادیر گره‌های میانی ممکن است دقیق نباشند

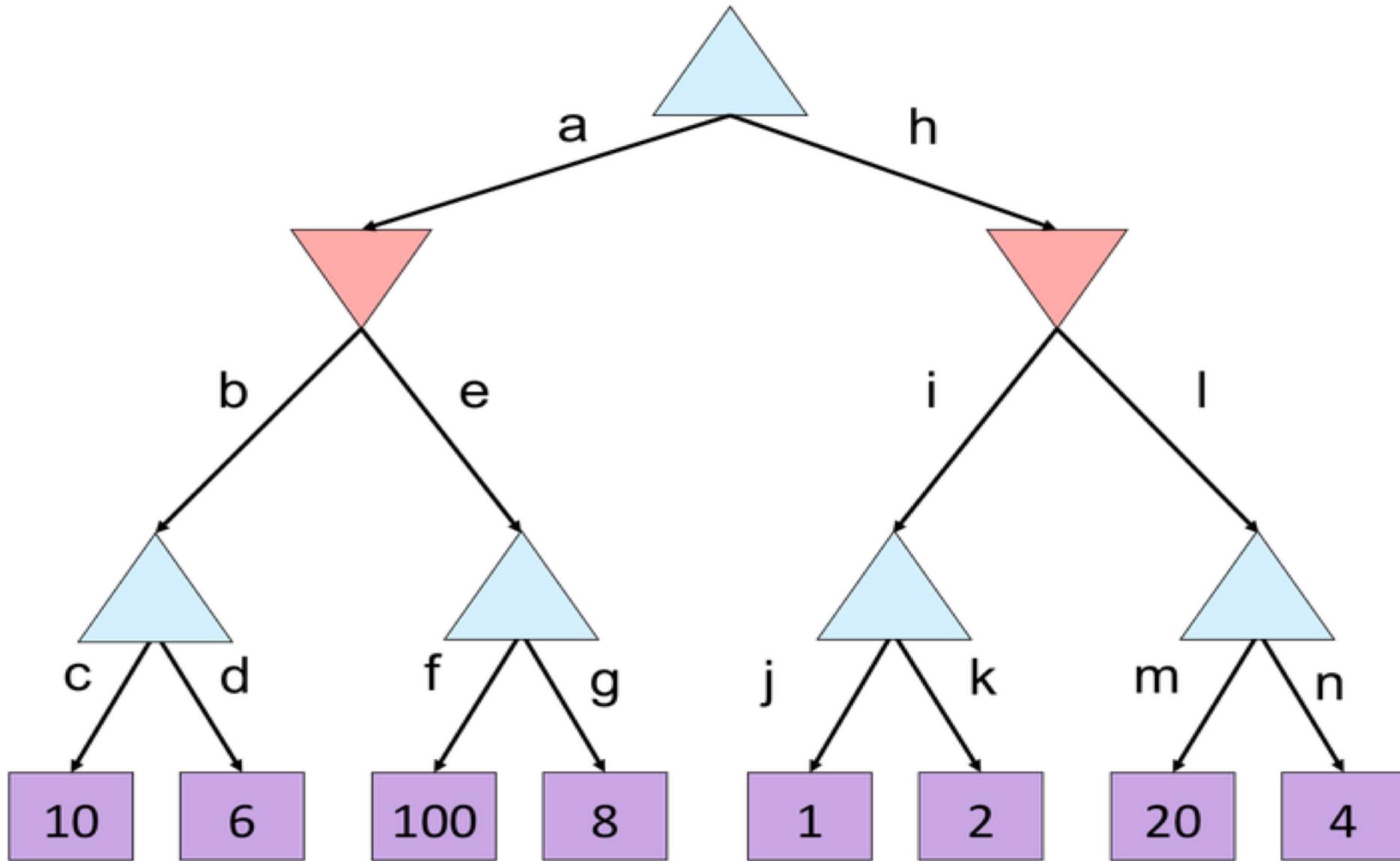
بنابراین، از مقادیر محاسبه شده در حالت ساده‌ی الگوریتم نمی‌توان برای انتخاب استراتژی استفاده کرد!

- 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)

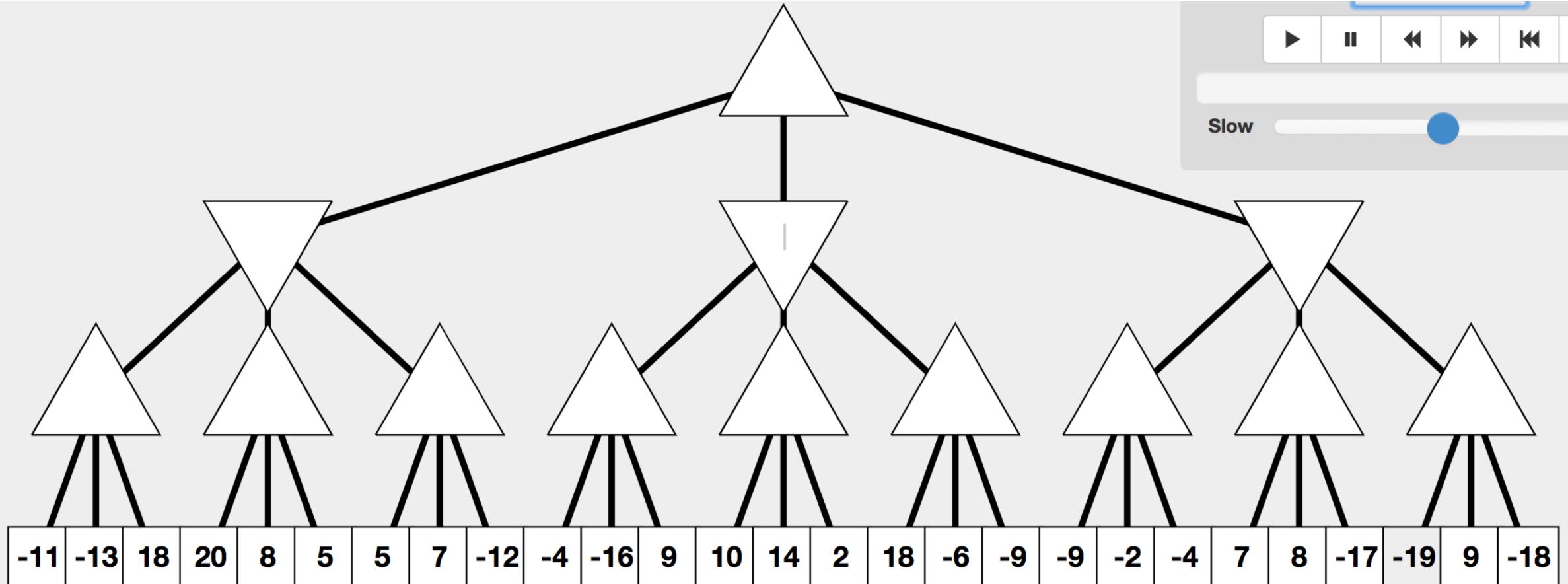
# کوییز : Alpha-Beta



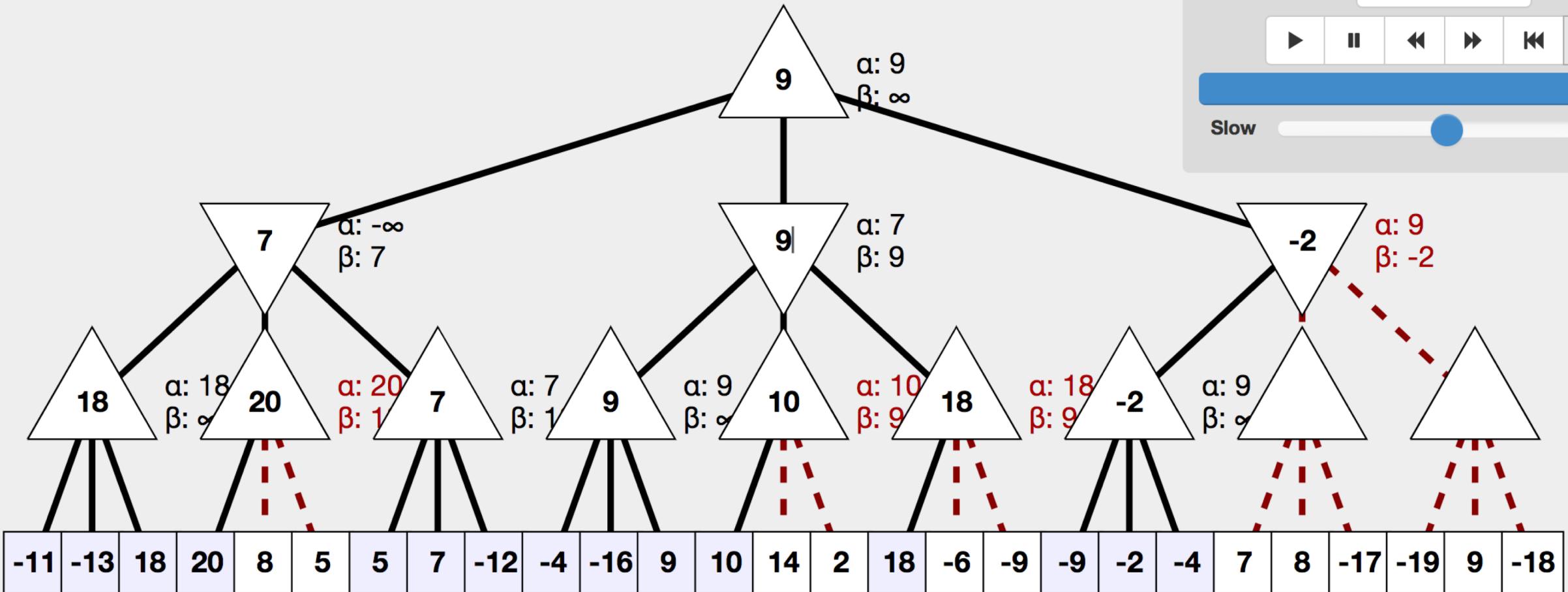
# کوییز : Alpha-Beta



# کوییز : Alpha-Beta



# کوییز : Alpha-Beta



# سؤال؟

---

