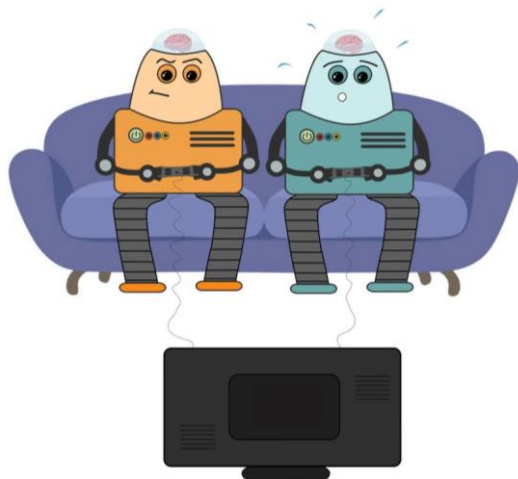


مبانی و کاربردهای هوش مصنوعی

جستجوی خصمانه (فصل 5.2 الی 5.5)



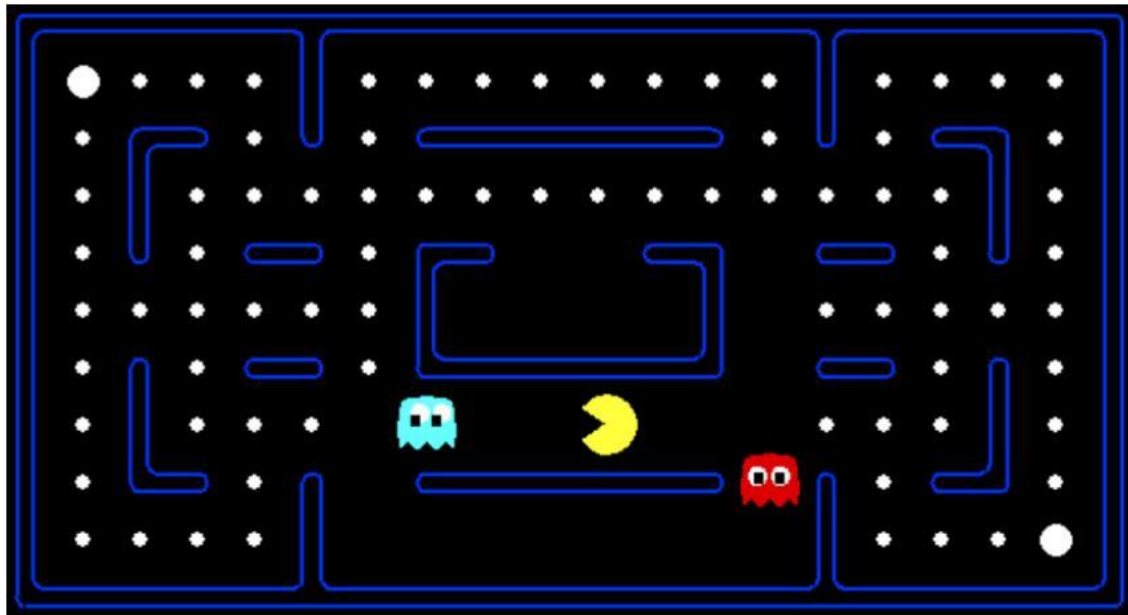
مدرس: مهدی جوانمردی

دانشکده مهندسی کامپیوتر دانشگاه صنعتی امیرکبیر



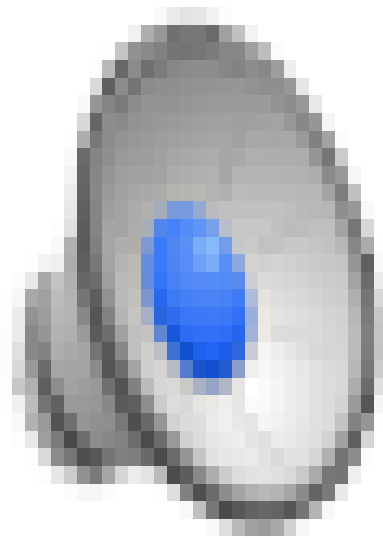
(الهام گرفته از محتوای درس هوش مصنوعی دانشگاه برکلی)

رفتار براساس محاسبات

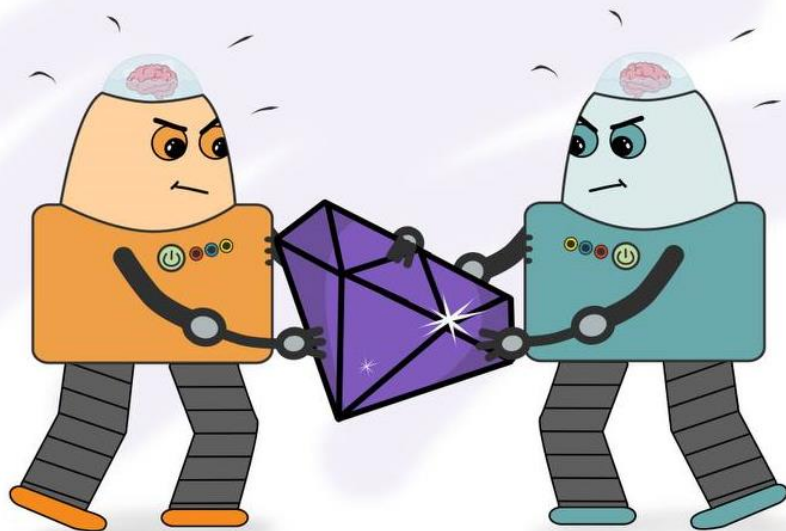


[Demo: mystery pacman (L6D1)]

ویدیوی دموی راز پکمن



بازی‌های خصمانه (Adversarial Games)



انواع بازی‌ها

- انواع زیادی از بازی‌ها وجود دارد!

- محورها:

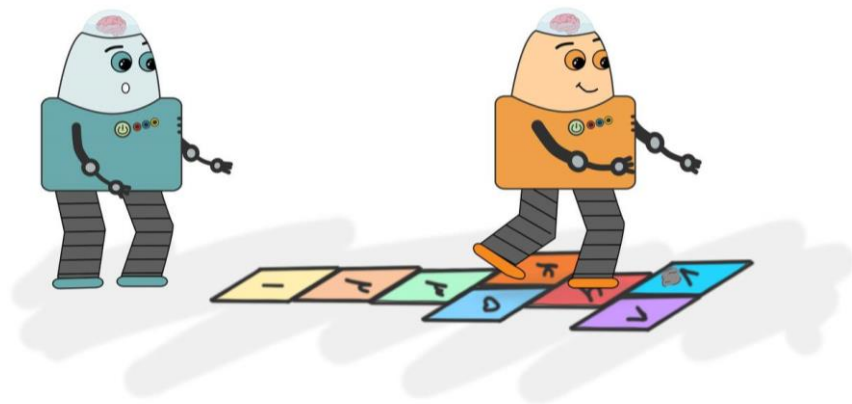
- قطعیت: قطعی یا تصادفی؟

- تعداد عامل: یک بازیکن، دوتا، یا بیشتر؟

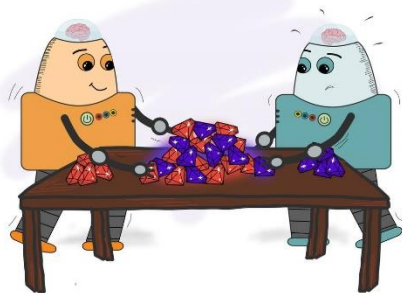
- مجموع صفر؟ Zero-sum

- اطلاعات کامل (آیا می‌توان حالت را دید)؟

- الگوریتمی برای محاسبه **سیاست** می‌خواهیم که بهترین حرکت را برای هر حالت از بازی پیشنهاد دهد

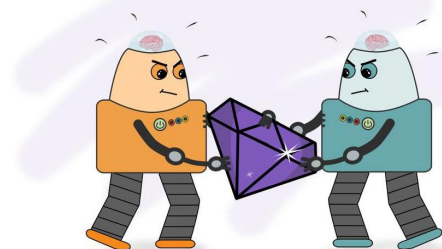


بازی‌های مجموع صفر (Zero-Sum)



بازی‌های عمومی (General Games):

- عامل‌ها سودمندی مستقل دارند (مقدار خروجی‌ها)
- همکاری، بی‌توجهی، رقابت و دیگر شرایط همگی ممکن است
- اطلاعات بیشتر در بخش بازی‌های مجموع غیر صفر



بازی‌های مجموع صفر (Zero-Sum):

- عامل‌ها سودمندی متضاد دارند (مقدار خروجی‌ها)
- می‌توانیم فرض کنیم که ارزشی (سودمندی) وجود دارد که یکی آن را بیشینه و دیگری آن را کمینه می‌کند
- خصمانه، رقابت خالص

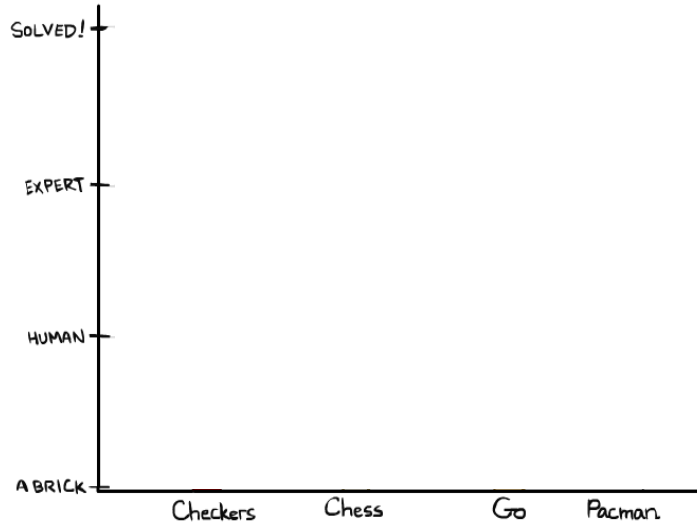
پیشرفته ترین روش بازی کردن



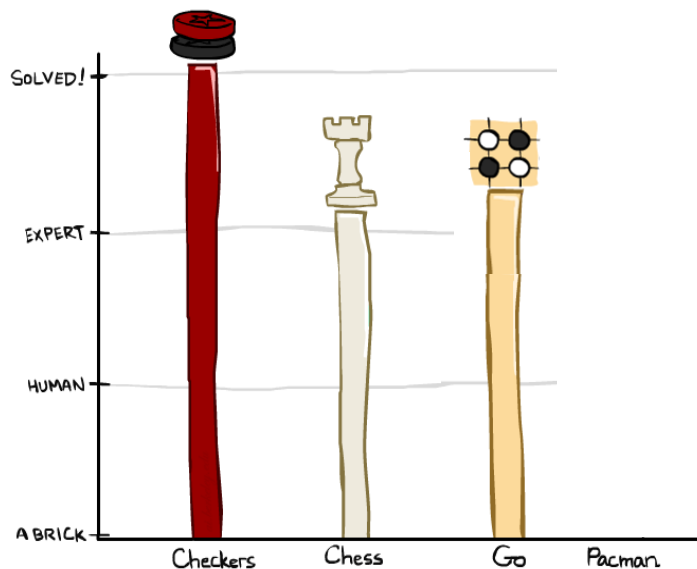
- چکرز (Checkers): در سال 1950، اولین بازیکن کامپیوتری. در سال 1994، اولین قهرمان کامپیوتری: چینوک دوره قهرمانی 40 ساله قهرمان انسانی "ماریون تینسلی" را به وسیله پایان کامل 8 قطعه‌ای پایان داد. در سال 2007، چکرز حل شد!

- شطرنج (Chess): در سال 1997، دیپ بلو (Deep Blue) قهرمان انسانی "گری کاسپاروف" را در یک مسابقه شش مرحله‌ای شکست داد. دیپ بلو 200 میلیون موقعیت در ثانیه را آزمایش کرد، از ارزیابی خیلی سطح بالا و روش‌های افشا نشده‌ای برای گسترش برخی خطوط جستجو تا 40 حرکت استفاده کرد. برنامه‌های فعلی حتی بهتر هستند، اگرچه به اندازه دیپ بلو تاریخ ساز نیستند.

- گو (Go): در گو، $b < 300$! برنامه‌های کلاسیک از پایگاه دانش‌های الگو استفاده می‌کنند، ولی پیشرفت‌های بزرگ اخیر از روش‌های گسترش مونت کارلو (تصادفی) استفاده می‌کنند.



پیشرفته ترین روش بازی کردن



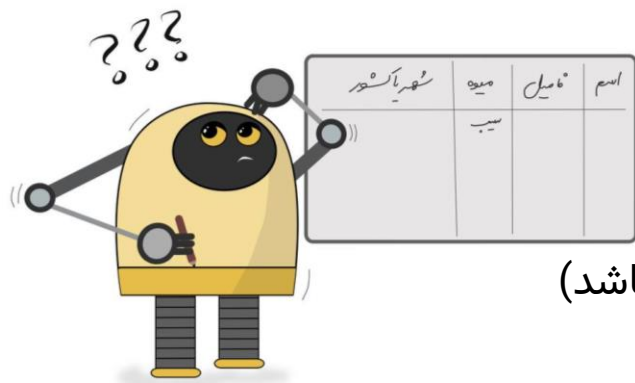
- **چکرز:** در سال 1950، اولین بازیکن کامپیوتری. در سال 1994، اولین قهرمان کامپیوتری: چینوک دوره قهرمانی 40 ساله قهرمان انسانی "ماریون تینسلی" را به وسیله پایان کامل 8 قطعه‌ای پایان داد. در سال 2007، چکرز حل شد!

- **شطرنج:** در سال 1997، دیپ بلو (Deep Blue) قهرمان انسانی "گری کاسپاروف" را در یک مسابقه شش مرحله‌ای شکست داد. دیپ بلو 200 میلیون موقعیت در ثانیه را آزمایش کرد، از ارزیابی خیلی سطح بالا و روش‌های افشا نشده‌ای برای گسترش برخی خطوط جستجو تا 40 حرکت استفاده کرد. برنامه‌های فعلی حتی بهتر هستند، اگرچه به اندازه دیپ بلو تاریخ ساز نیستند.

- **گو:** در سال 2016، آلفا گو قهرمان انسانی را شکست می‌دهد. از جستجوی درختی مونت کارلو (Monte Carlo) استفاده می‌کند، تابع ارزیابی را یاد گرفت.

- **پکمن (Pacman)**

بازی‌های قطعی با سودمندی نهایی

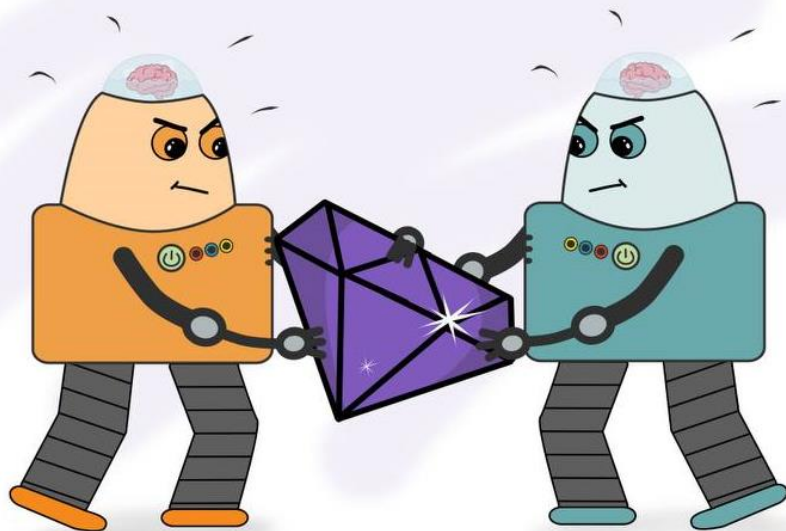


■ فرمول‌بندی‌های زیادی ممکن است، به طور مثال:

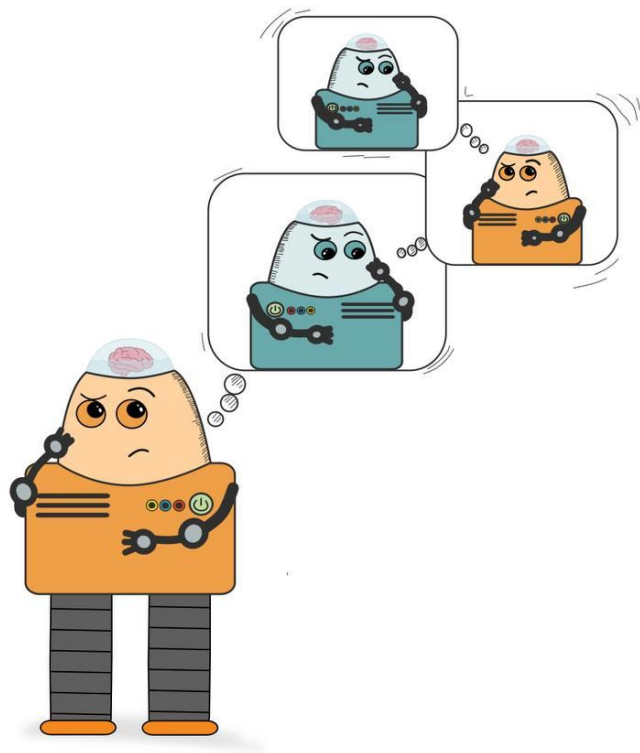
- حالت‌ها: S (start at s_0)
- بازیکنان: $P = \{1 \dots N\}$ (معمولا به صورت نوبتی)
- اعمال: A (ممکن است به بازیکن / حالت بستگی داشته باشد)
- تابع انتقال (Transition Function): $S \times A \rightarrow S$
- آزمون پایانی (Terminal Test): $S \rightarrow \{t, f\}$
- سود پایانی (Terminal Utilities): $S \times P \rightarrow R$

■ راه حل برای یک بازیکن یک سیاست است: $S \rightarrow A$

بازی‌های خصمانه (Adversarial Games)



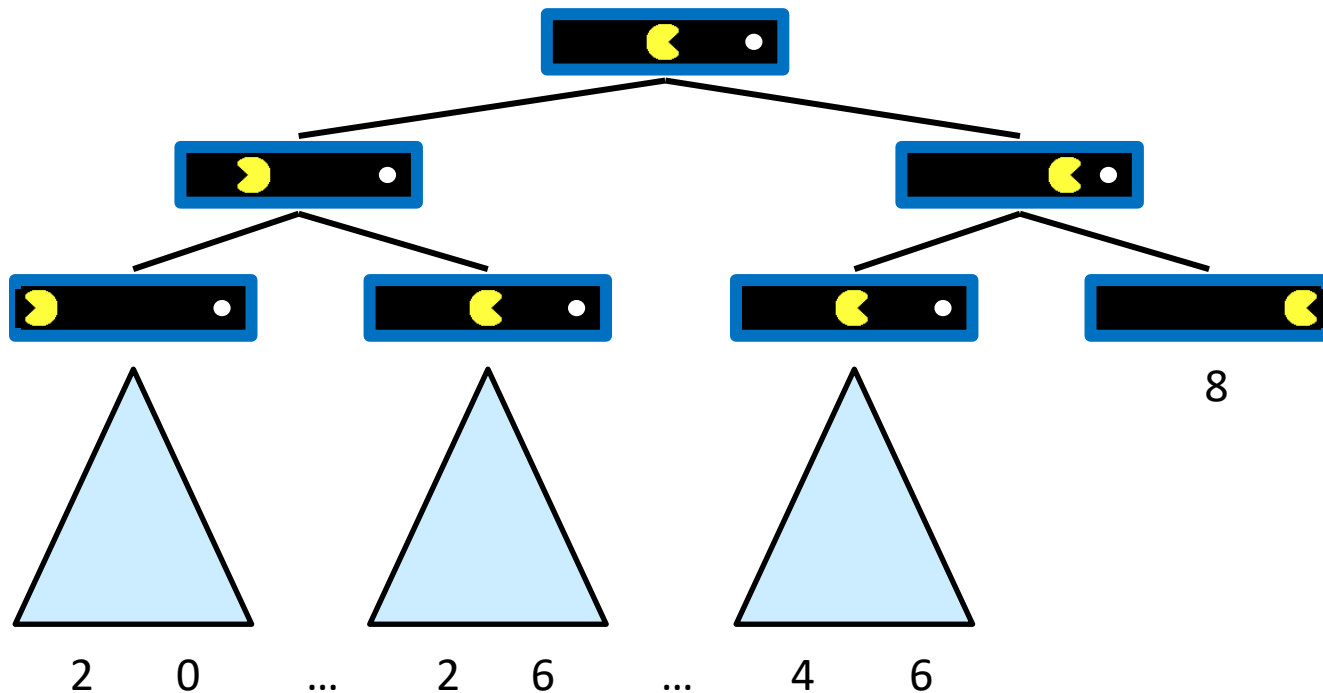
جستجوی خصمانه (Adversarial Search)



خبر جدید: هزینه - سودمندی

- دیگر به دنبال کمینه کردن هزینه نیستیم
- در اینجا عامل می‌خواهد امتیاز / سودمندی خود را بیشینه کند!

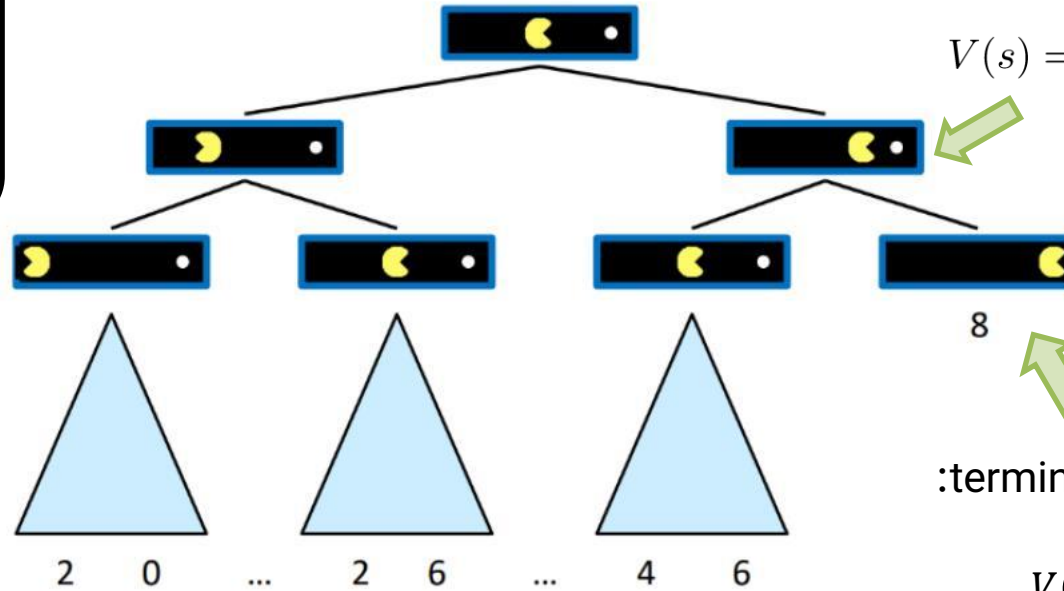
درختهای تک عاملی (Single-Agent Trees)



ارزش یک حالت (Value of a state)

ارزش یک حالت:

بهترین نتیجه (سود)
قابل رسیدن از این
حالت



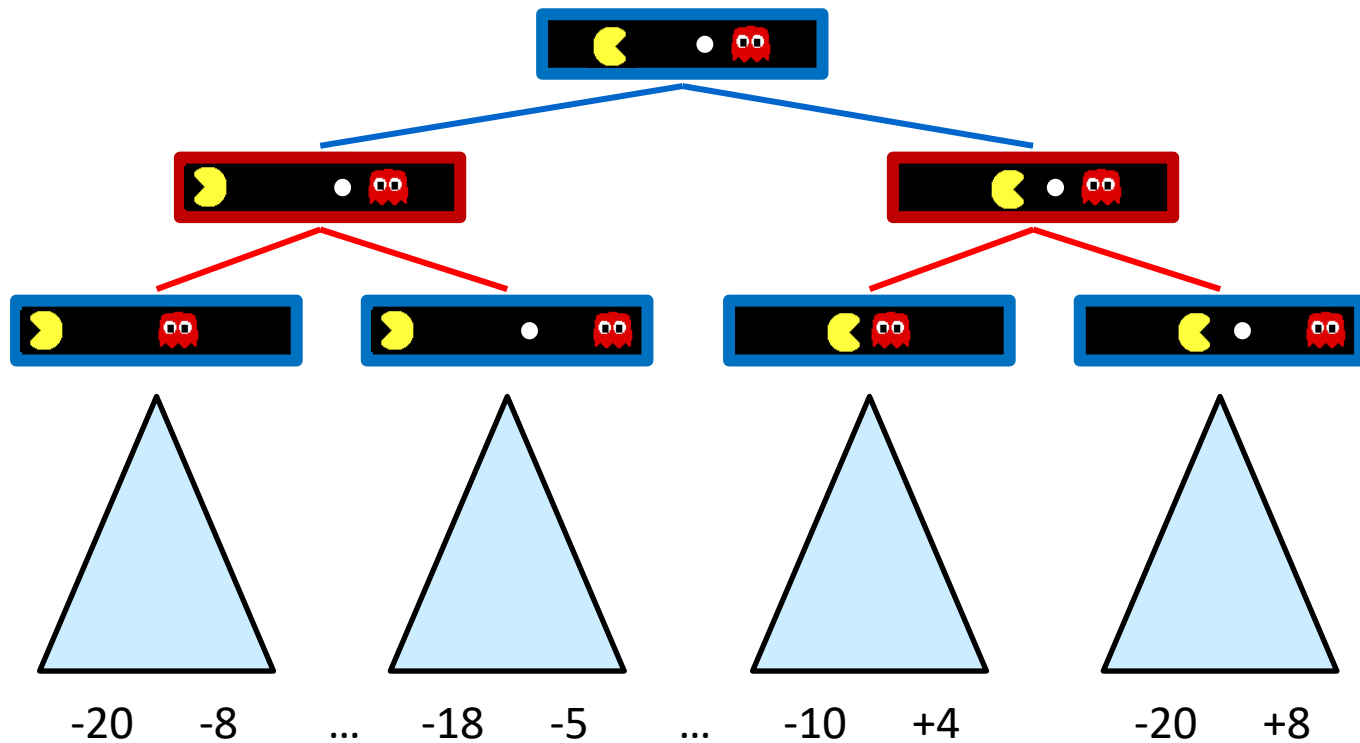
حالت‌های غیرپایانی:

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

حالت‌های پایانی terminal:

$$V(s) = \text{معلوم}$$

درخت‌های بازی خصمانه



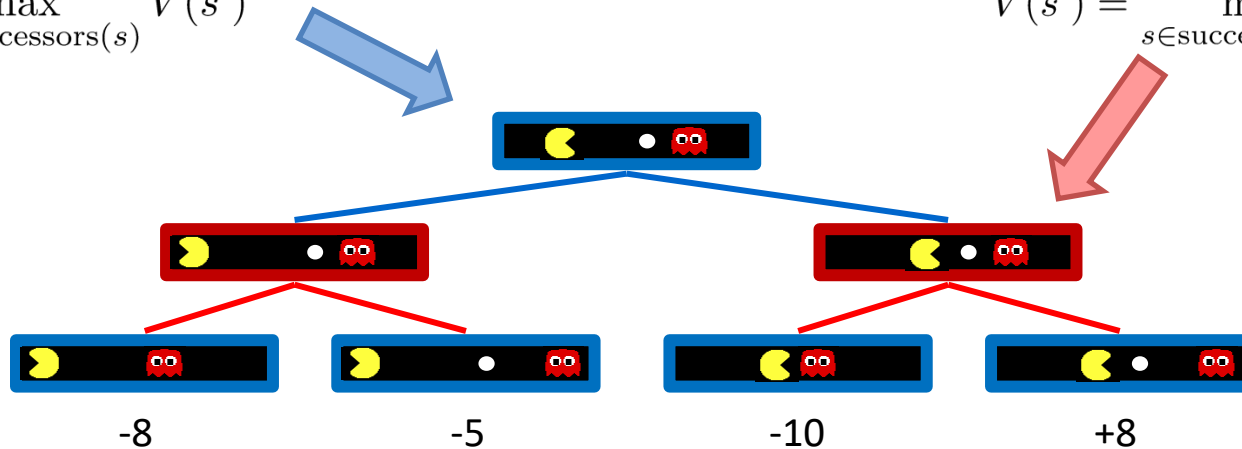
ارزش‌های کمینه-بیشینه (Minimax Values)

حالت‌های تحت کنترل عامل:

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

حالت‌های تحت کنترل حریف:

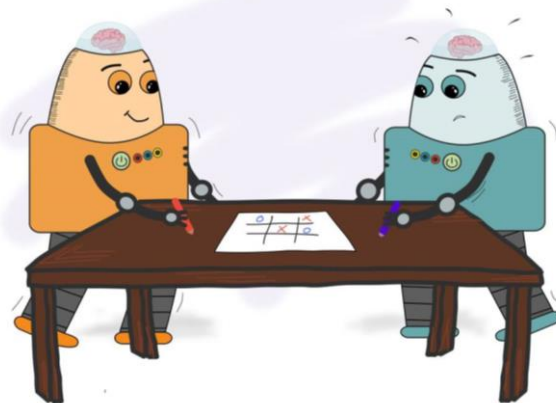
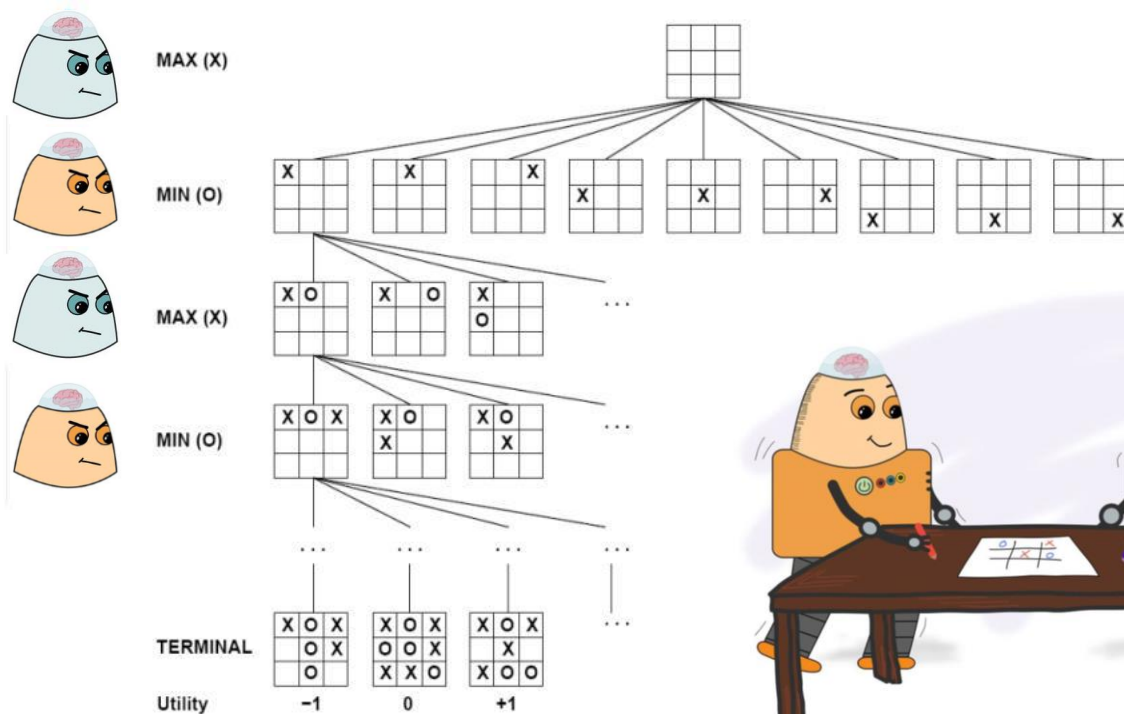
$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$



حالت‌های پایانی terminal:

$$V(s) = \text{معلوم}$$

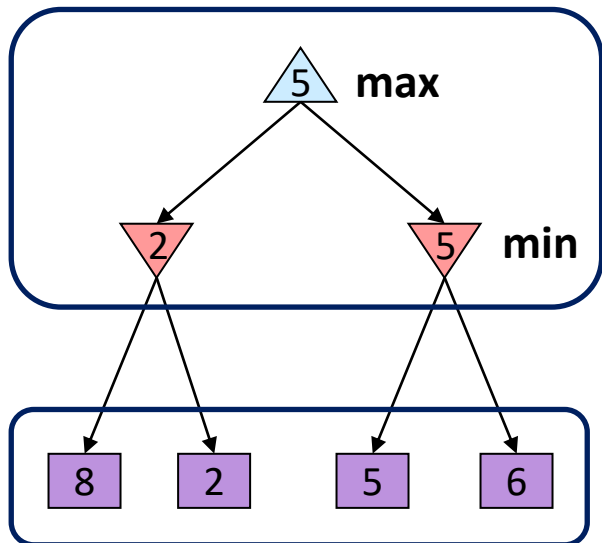
درخت بازی دوز



جستجوی تخصی (کمینه-بیشینه) Minimax

مقادیر کمینه-بیشینه:

به صورت بازگشتی محاسبه می‌شوند



مقادیر پایانی:

بخشی از بازی

■ بازی‌های مجموع صفر قطعی

- دوز، شطرنج، چکرز
- یک بازیکن نتیجه را بیشینه می‌کند
- بازیکن دیگر نتیجه را کمینه می‌کند

■ جستجوی کمینه-بیشینه:

- یک درخت جستجوی فضای حالت
- بازیکن‌ها به نوبت بازی می‌کنند
- مقدار کمینه-بیشینه هر گره محاسبه می‌شود: بهترین سود قابل حاصل در مقابل یک حریف منطقی (بهینه)

پیاده‌سازی کمینه-بیشینه

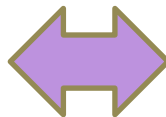
```
def max-value(state):
```

```
    initialize v =  $-\infty$ 
```

```
    for each successor of state:
```

```
        v = max(v, min-value(successor))
```

```
    return v
```



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

```
    initialize v =  $+\infty$ 
```

```
    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 کمینه-بیشینه (پخش)

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):

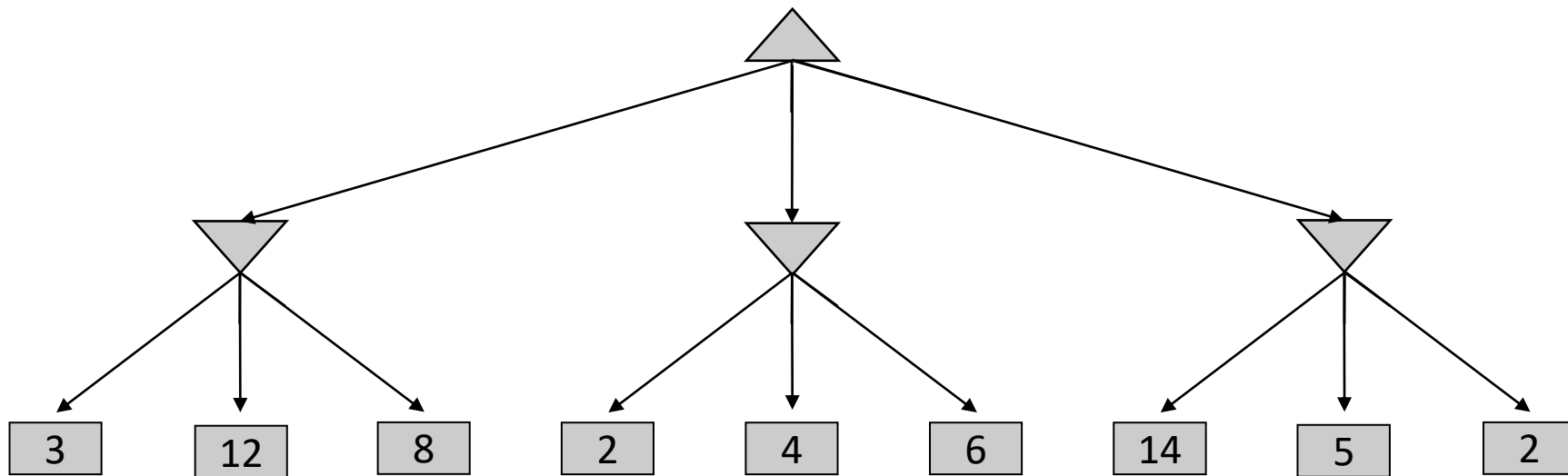
initialize $v = +\infty$

for each successor of state:

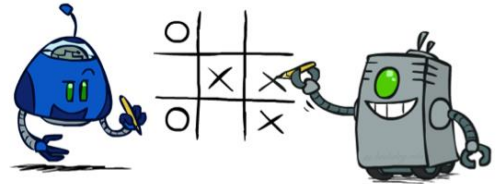
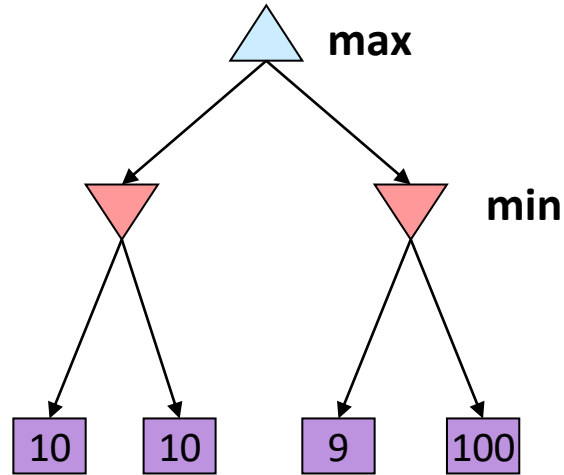
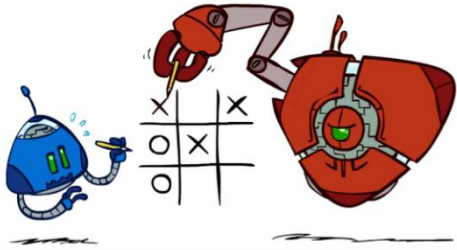
$v = \min(v, \text{value}(\text{successor}))$

return v

مثال کمینه-بیشینه

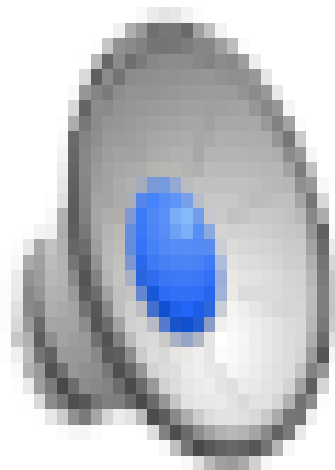


ویژگی‌های کمینه-بیشینه

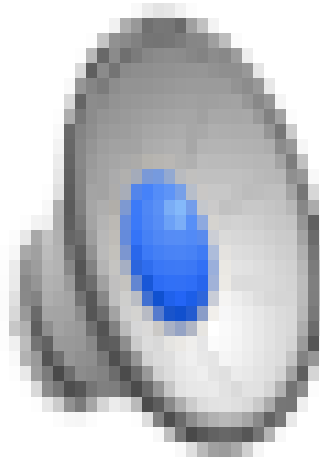


بهینه در برابر یک حریف بی نقص. در غیر این صورت؟

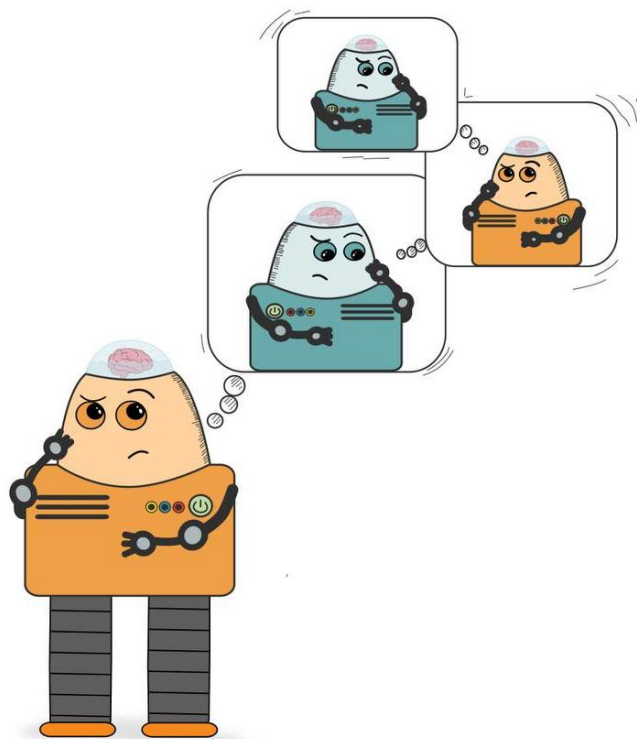
ویدیوی نمایشی Min vs. Exp (Min)



ویدیوی نمایشی (Exp) Min vs. Exp



بهینگی کمینه-پیشینه



■ کمینه-پیشینه چقدر بهینه است؟

■ مثل DFS (کامل)

■ پیچیدگی زمانی: $O(b^m)$

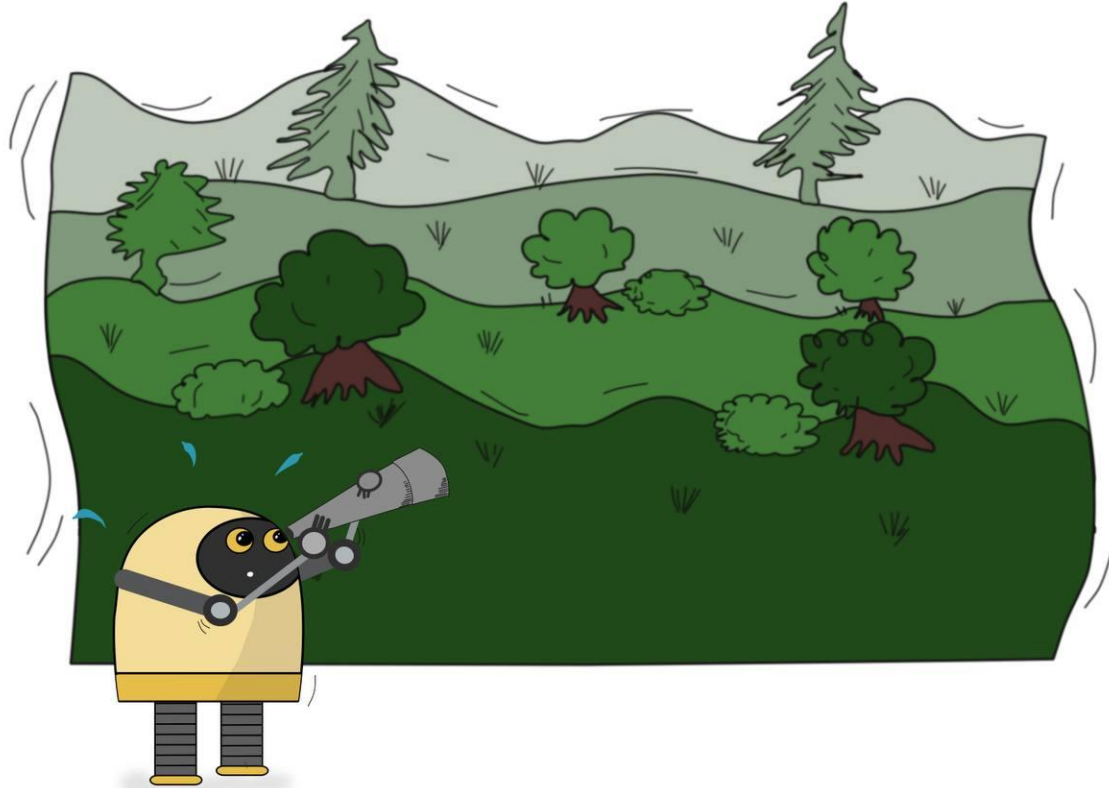
■ پیچیدگی مکانی: $O(bm)$

■ مثال: برای شطرنج، $b \approx 35$ ، $m \approx 100$

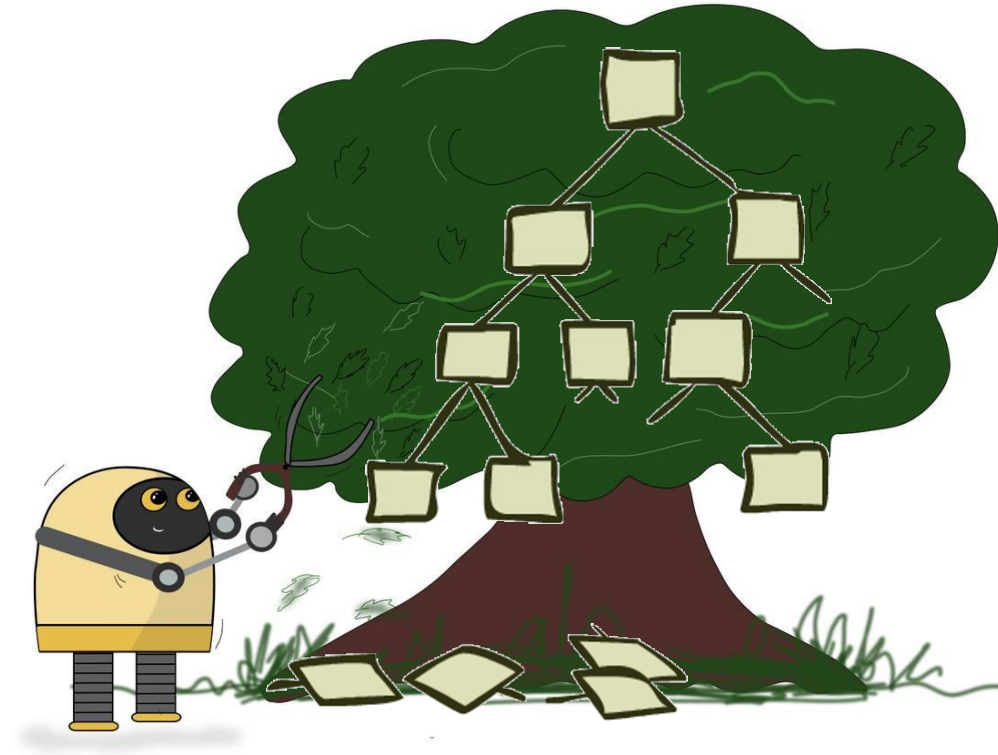
■ راه حل دقیق غیر دستیافتنی است

■ ولی، آیا باید کل درخت را بگردیم؟

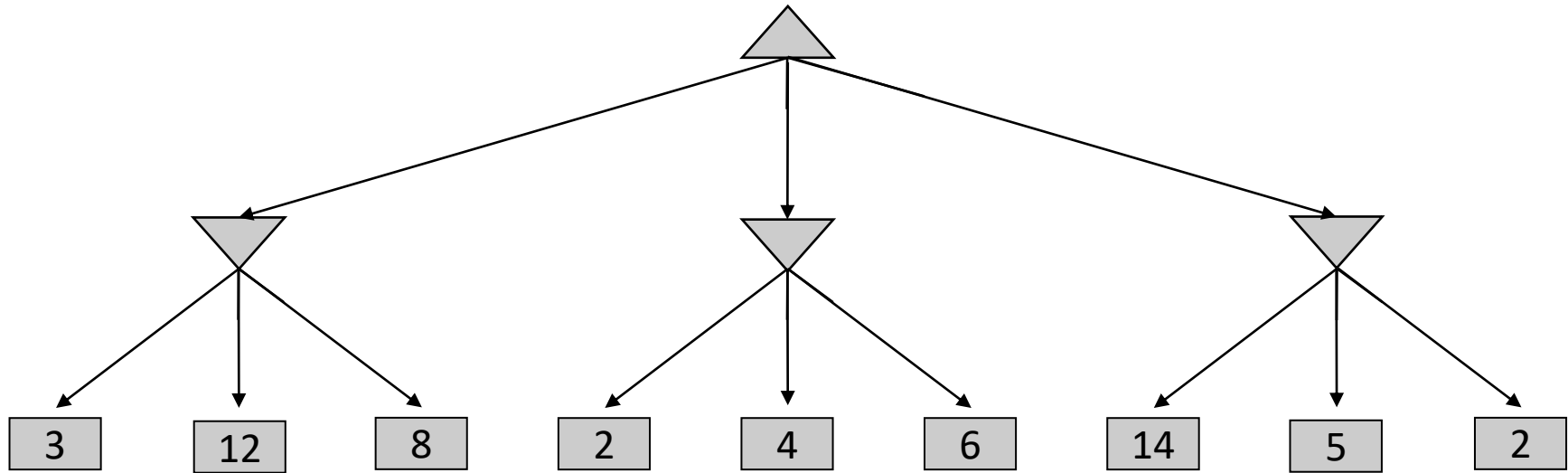
محدودیت منابع



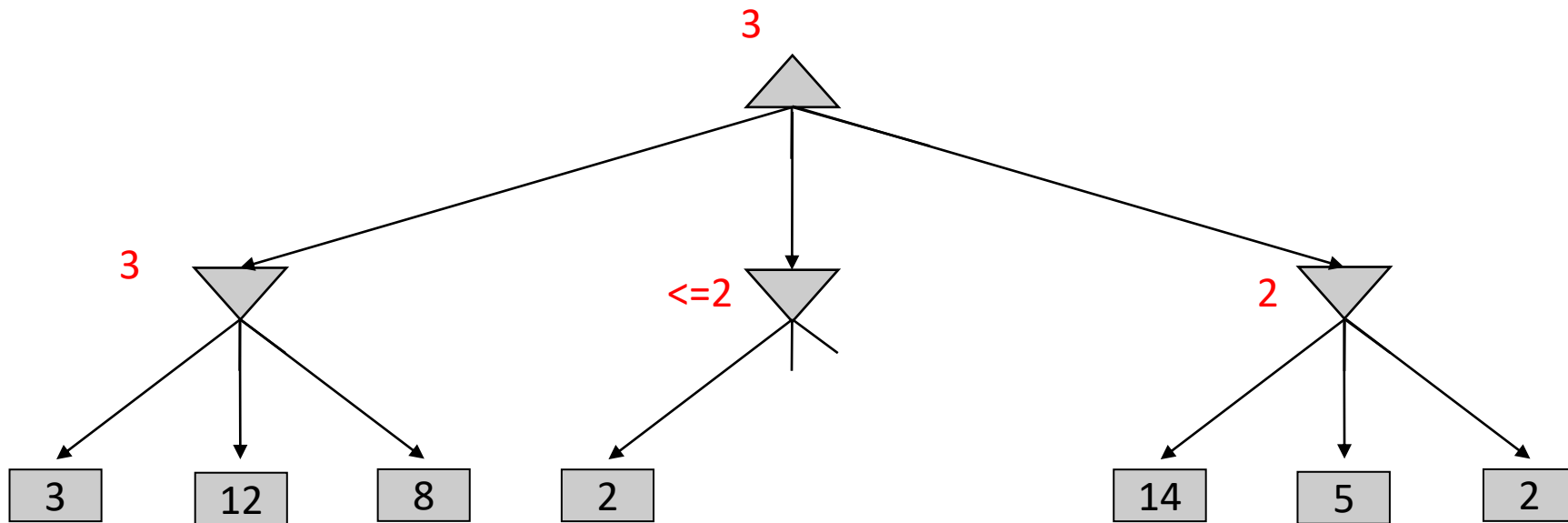
هرس درخت بازی (Game Tree Pruning)



مثال کمینه-بیشینه

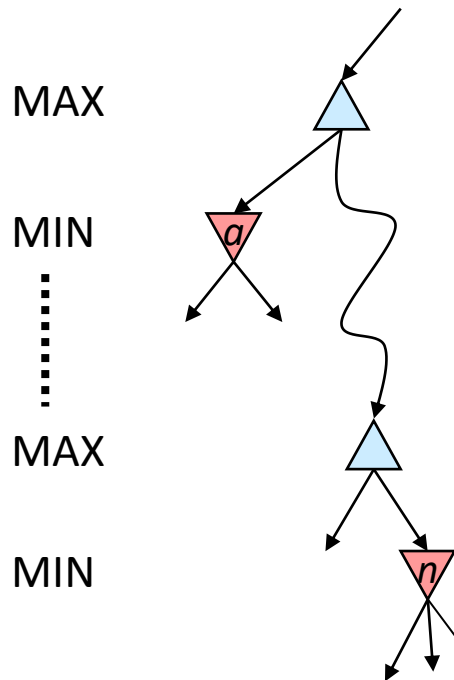


هرس کمینه-بیشینه (Minimax Pruning)



هرس آلفا-بتا (Alpha-Beta Pruning)

■ ساختار کلی (نسخه کمینه)



- در حال محاسبه مقدار کمینه در یک دلخواه گره n هستیم
- بر روی فرزندان گره n ، پیمایش می‌کنی.
- برآورد گره n از کمینه فرزندان در حال کاهش است
- کدام گره به مقدار n اهمیت می‌دهد؟ بیشینه (MAX)
- فرض کنید a بهترین مقداری باشد که بیشینه می‌تواند در هر نقطه انتخابی در مسیر فعلی از ریشه انتخاب کند
- اگر n بدتر از a شود، بیشینه از آن پرهیز می‌کند، بنابراین می‌توانیم بقیه فرزندان n را در نظر نگیریم (همین الان به قدری بد است که دیگر بازی نخواهد شد).

■ نسخه بیشینه متقارن کمینه است

پیاده‌سازی آلفا-بتا

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 $v = -\infty$

 for each successor of state:

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

 if $v \geq \beta$ return v

$\alpha = \max(\alpha, v)$

 return v

def min-value(state, α , β):

 initialize $v = +\infty$

 for each successor of state:

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

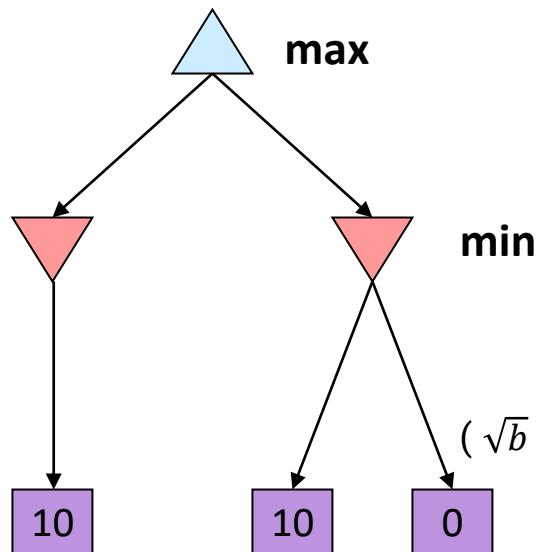
 if $v \leq \alpha$ return v

$\beta = \min(\beta, v)$

 return v

ویژگی‌های هرس آلفا-بتا

- این هرس هیچ اثری بر مقدار کمینه-بیشینه محاسبه شده برای ریشه ندارد!



- مقادیر گره‌های میانی ممکن است اشتباه باشد

- مهم: فرزندان ریشه ممکن است مقادیر اشتباهی داشته باشند

- بنابراین ساده‌ترین نسخه به شما اجازه انتخاب عمل را نمی‌دهد

- مرتب سازی خوب فرزندان، اثرگذاری هرس را افزایش می‌دهد

- با "مرتب سازی کامل":

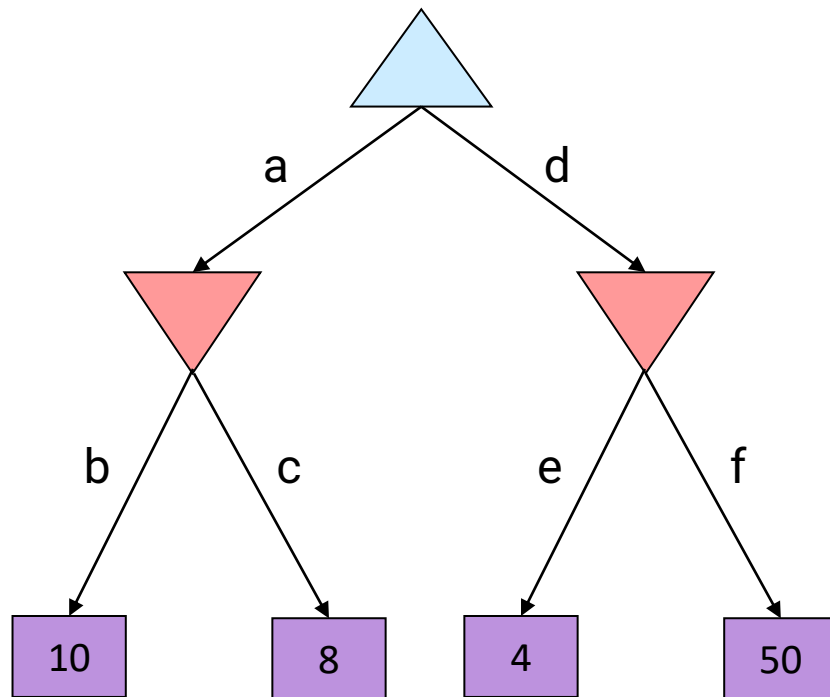
- پیچیدگی زمانی به $O(b^{m/2})$ کاهش پیدا می‌کند (کاهش ضریب انشعاب به \sqrt{b})

- عمق قابل حل را دو برابر می‌کند!

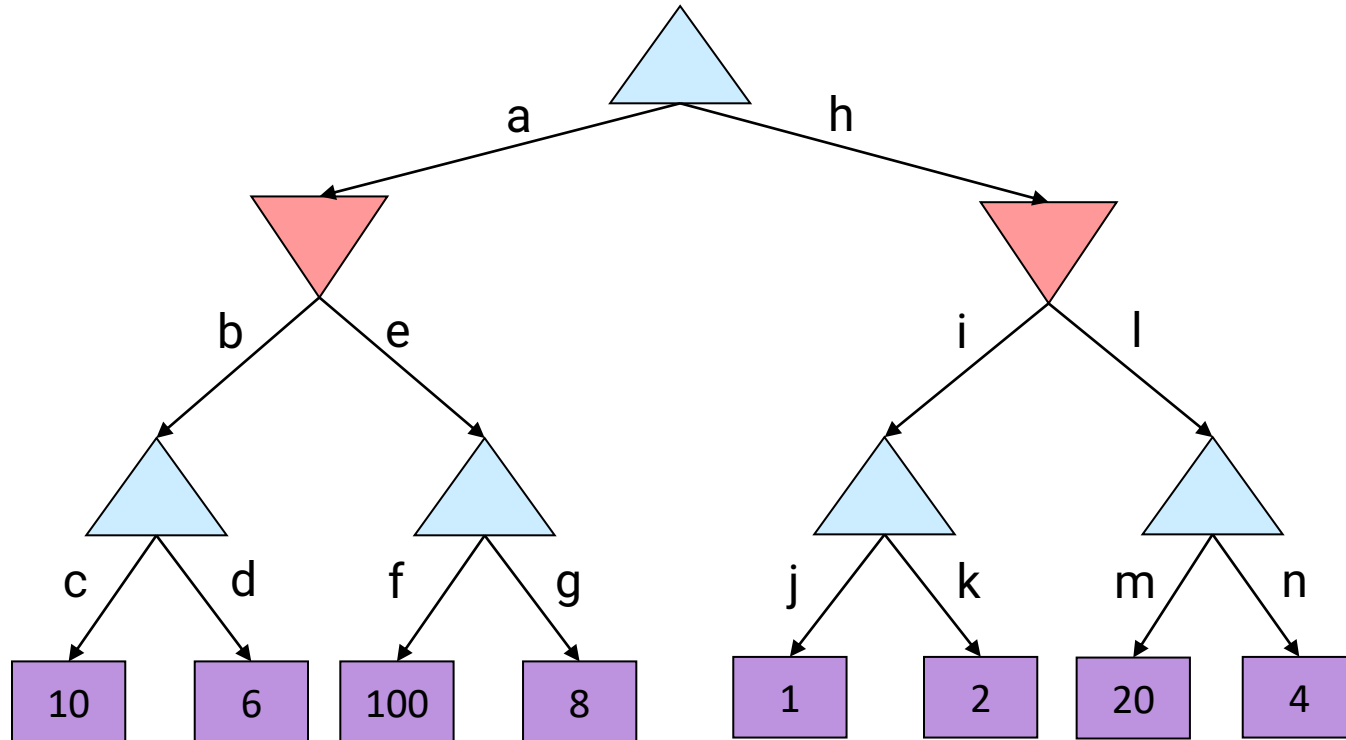
- جستجوی کامل، مثلاً در شطرنج، همچنان ممکن نیست...

- این یک مثال ساده از **metareasoning** است. (محاسبه اینکه چه چیزهایی محاسبه شوند)

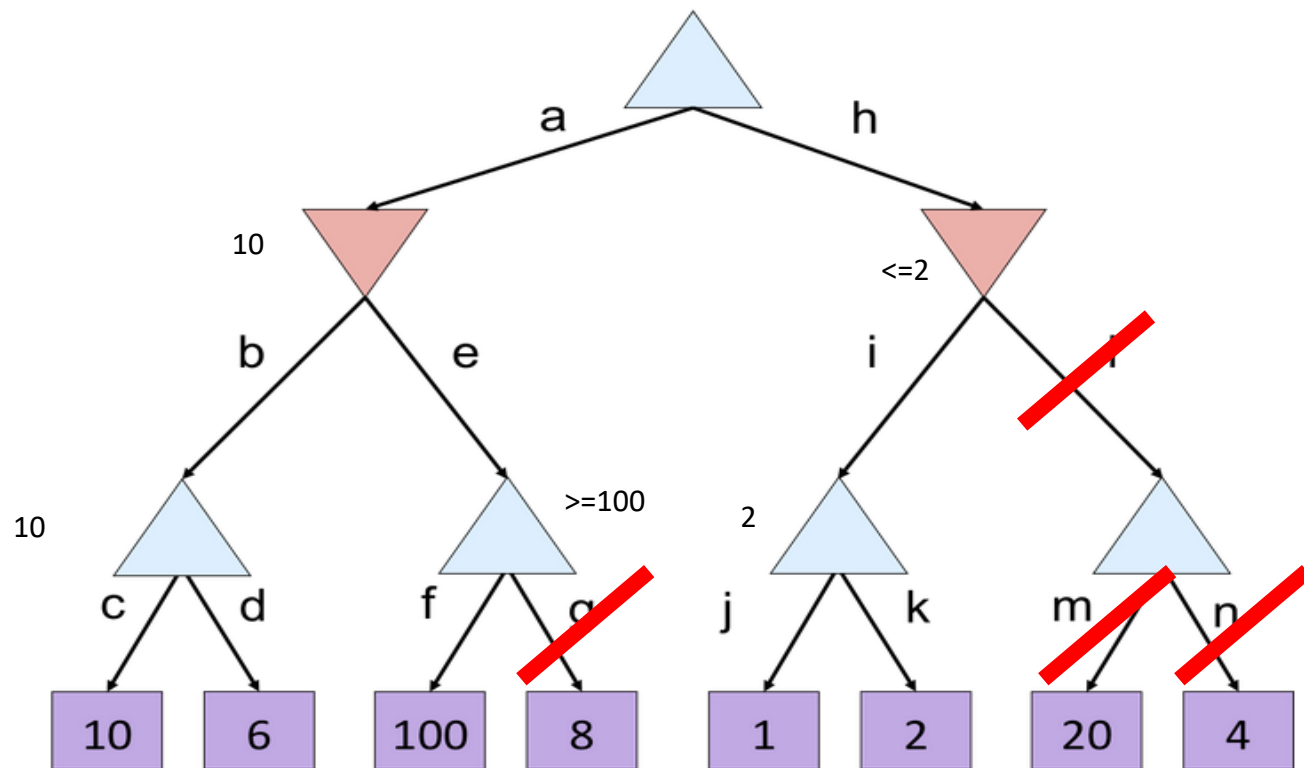
آزمونک آلفا-بتا



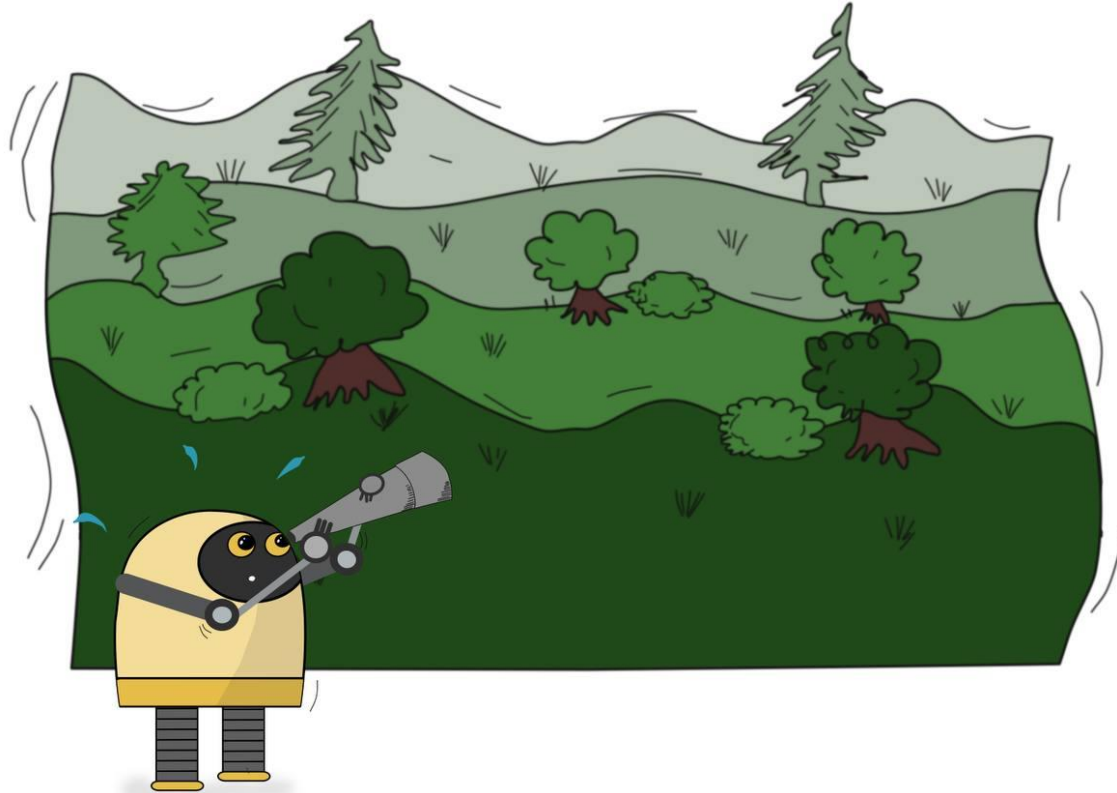
آزمونک آلفا-بتا 2



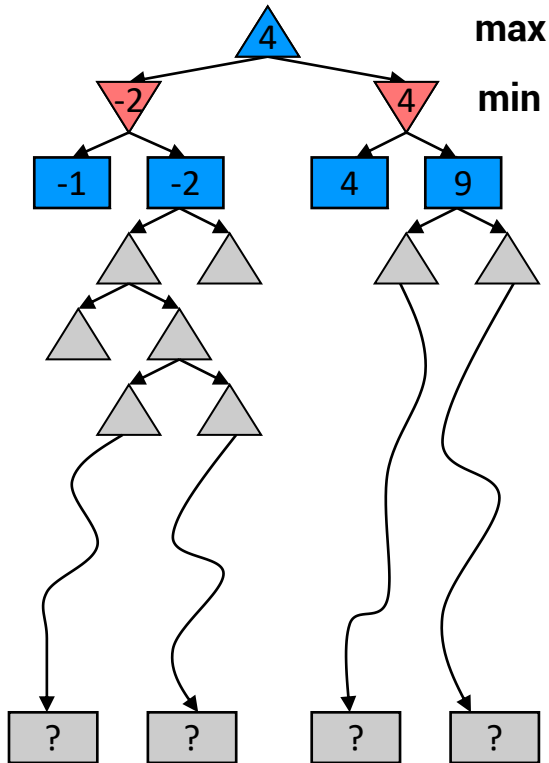
آزمونک آلفا-بتا 2



محدودیت منابع

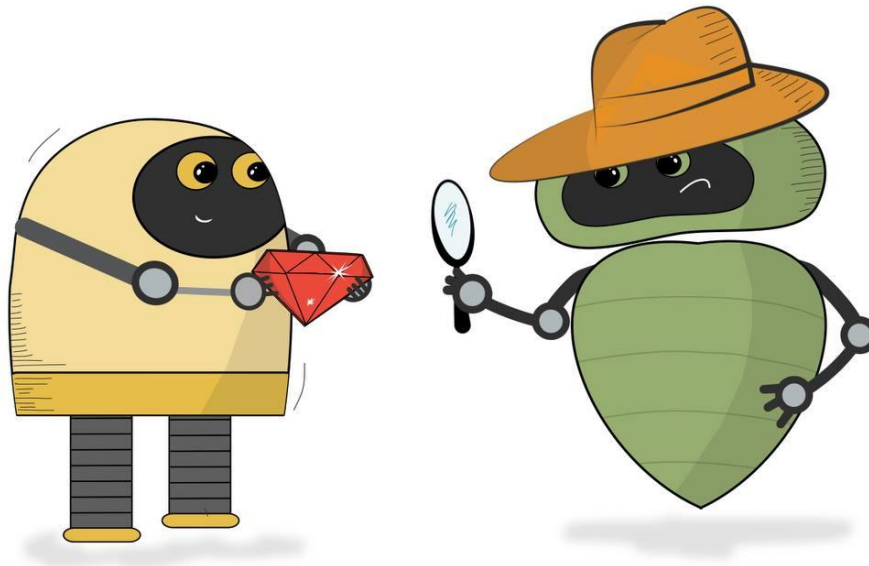


محدودیت منابع



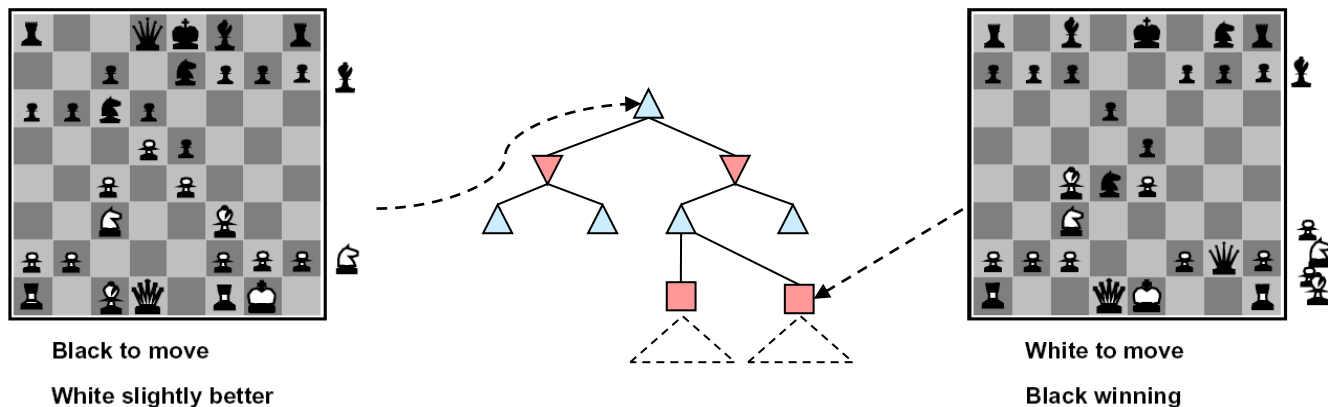
- مشکل: در بازی‌های واقعی، نمی‌توان تا برگ‌ها جستجو کرد!
- راه حل: جستجو با عمق محدود شده (Depth-limited Search)
 - به جای اینکار، تا یک عمق محدودی از درخت را جستجو می‌کنیم
 - سودهای پایانی را با یک تابع ارزیابی برای حالت‌های غیرپایانی، جایگزین می‌کنیم
- مثال:
 - فرض کنید ما 100 ثانیه وقت داریم و می‌توانیم با سرعت 10k گره بر ثانیه جستجو کنیم.
 - بنابراین می‌توانیم 1M گره را در هر حرکت بررسی کنیم.
 - در بازی شطرنج α - β تقریباً به عمق 8 می‌رسد
- ضمانت بازی بهینه از بین می‌رود.
- لایه‌های بیشتر تفاوت بزرگی ایجاد می‌کند.
- از افزایش عمق مرحله‌ای، برای هر الگوریتم در هر زمان (Anytime Algorithm) استفاده می‌کنیم.

توابع ارزیابی (Evaluation Functions)



توابع ارزیابی

توابع ارزیابی به موقعیت‌های غیرپایانی در جستجو با عمق محدود امتیاز می‌دهند.



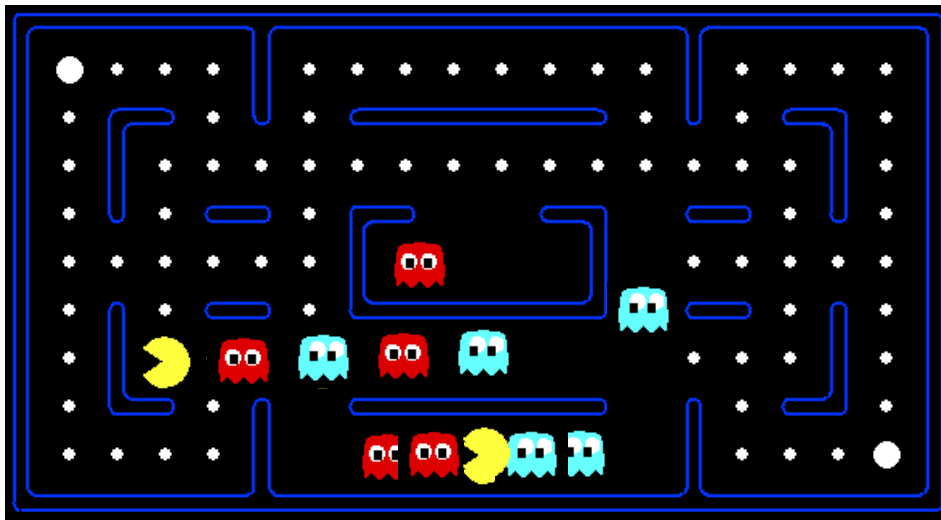
تابع ایده آل: مقدار واقعی کمینه-بیشینه موقعیت را بر می‌گرداند.

درعمل: جمع جبری وزن دار ویژگی‌ها:

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

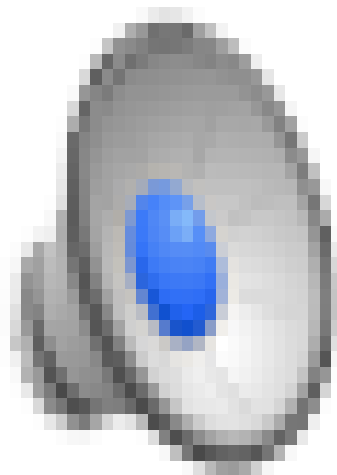
مثال: $f_1(s) = (\text{num white queens} - \text{num black queens})$ ، و غیره

ارزیابی برای پکن

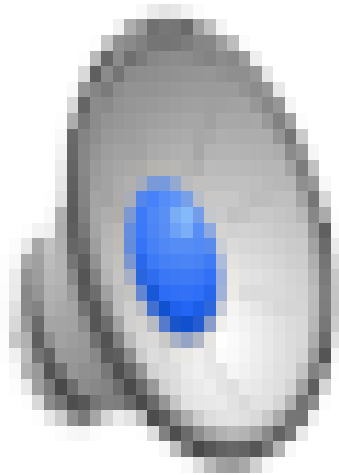


[Demo: thrashing d=2, thrashing d=2 (fixed evaluation function), smart ghosts coordinate (L6D6,7,8,10)]

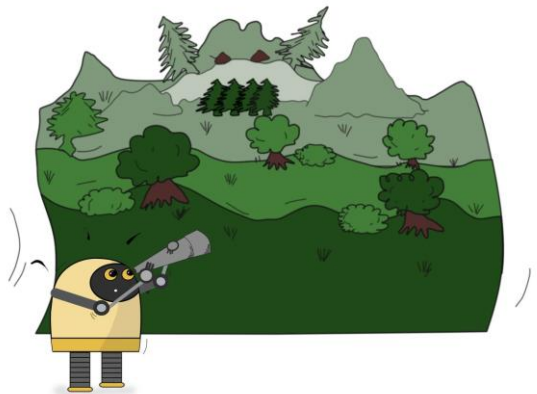
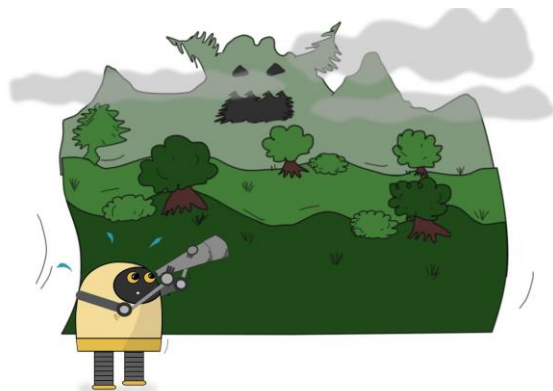
ویدیوی نمایشی Smart Ghosts (Coordination)



Smart Ghosts (Coordination) – Zoomed In ویدیوی دمو

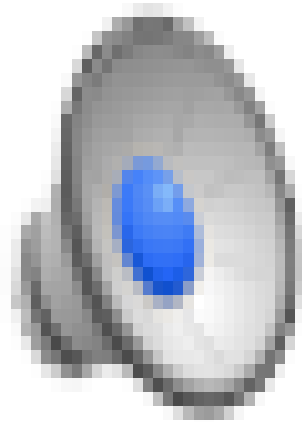


عمق اهمیت دارد



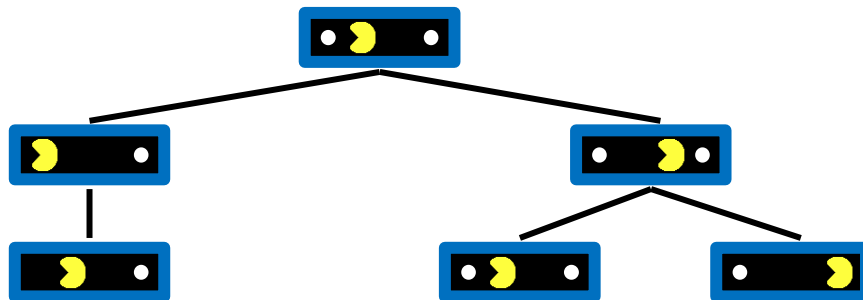
- توابع ارزیابی همیشه ناکاملند
- هرچقدر تابع ارزیابی عمیق‌تر در داخل درخت استفاده شود، اهمیت کیفیت ارزیابی آن کمتر می‌شود
- یک مثال مهم مبادله (Tradeoff) پیچیدگی ویژگی‌ها و پیچیدگی محاسبات است

ویدیوی نمایشی Thrashing (d=3)



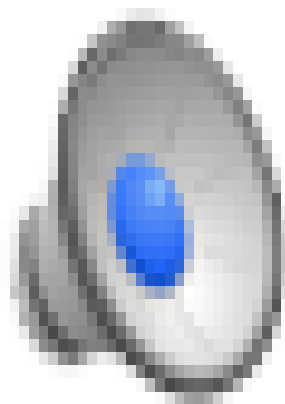
Eval(s)=score for depth 3... is that good?

چرا پکمن گرسنه می ماند



- یک خطر عامل‌های با برنامه‌ریزی مجدد
 - او می‌داند امتیازش با الان خوردن نقطه بالا می‌رود. (غرب، شرق)
 - او می‌داند امتیازش با خوردن بعدا نیز همانقدر بالا می‌رود. (غرب، شرق)
 - هیچ فرصت امتیازگیری دیگری پس از خوردن نقطه وجود ندارد. (در افق دید، اینجا دوتا است)
 - بنابراین، صبر کردن دقیقا به اندازه خوردن خوب به نظر می‌آید: او می‌تواند به شرق برود، سپس در دور بعدی برنامه‌ریزی مجدد به غرب برگردد!

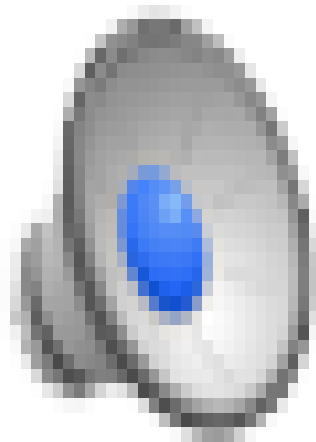
ویدیوی نمایشی Thrashing -- Fixed ($d=3$)



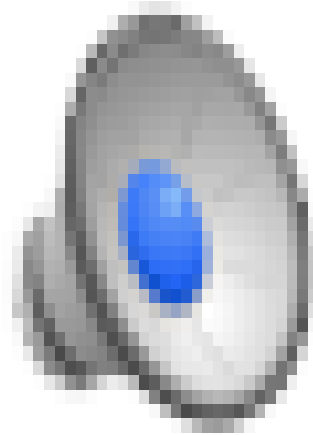
Eval(s)=score for depth 3 + credit for being closer to dot

[Demo: thrashing $d=2$, thrashing $d=2$ (fixed evaluation function) (L6D7)]

دموی عمق محدود 2



دموی عمق محدود 10



هم افزایی تابع ارزیابی و آلفا-بتا؟

- آلفا-بتا: مقدار هرس به ضریب انشعاب و ترتیب گره‌ها بستگی دارد
- تابع ارزیابی می‌تواند به ما کمک کند که گره‌های بهتر را اول گسترش دهیم
- (تقریباً شبیه نقش هیوریستیک A^* ، فیلترینگ CSP)
- مقدار یک گره Min همواره رو به کاهش است
- وقتی که مقدار گره Min کمتر از بهترین گزینه موجود برای Max‌های در راه ریشه باشد، می‌توان هرس کرد
- بنابراین: اگر تابع ارزیابی برای مقدار گره Min حد بالا ارائه دهد و این حد بالا کمتر از بهترین گزینه موجود برای Max‌های در راه ریشه باشد، آنگاه می‌توان هرس کرد

جلسه بعد: عدم قطعیت!
