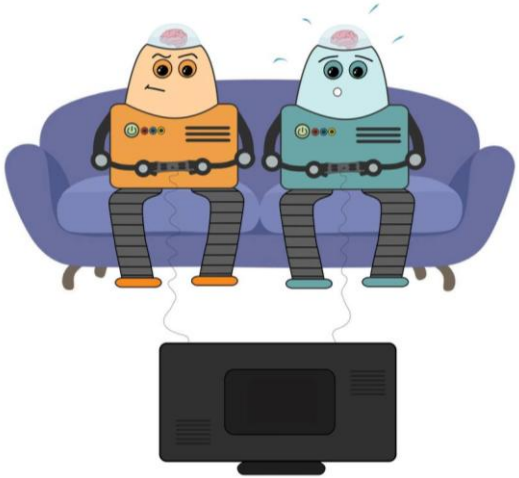
مبانی و کاربردهای هوش مصنوعی

جستجوی خصمانه - 1 (فصل 5.2 الی 5.5)

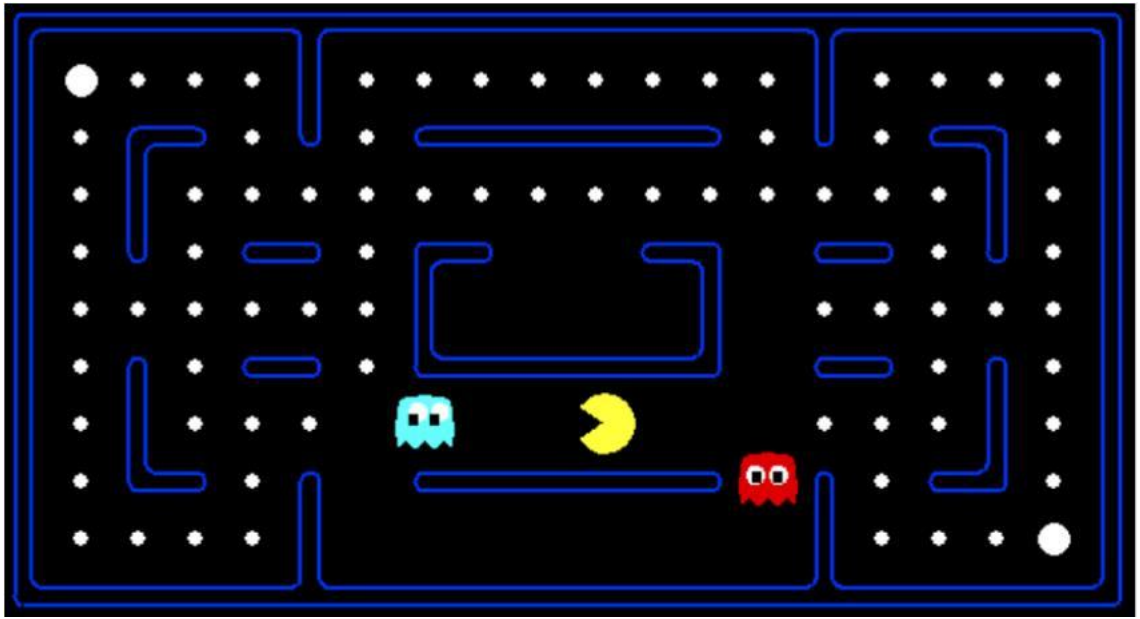


مدرس: مهدی جوانمردی

دانشکده مهندسی کامپیوتر دانشگاه صنعتی امیرکبیر

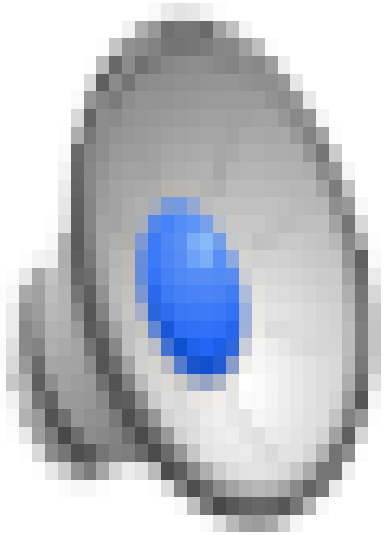
(الهام گرفته از محتوای درس هوش مصنوعی دانشگاه برکلی)

رفتار براساس محاسبات

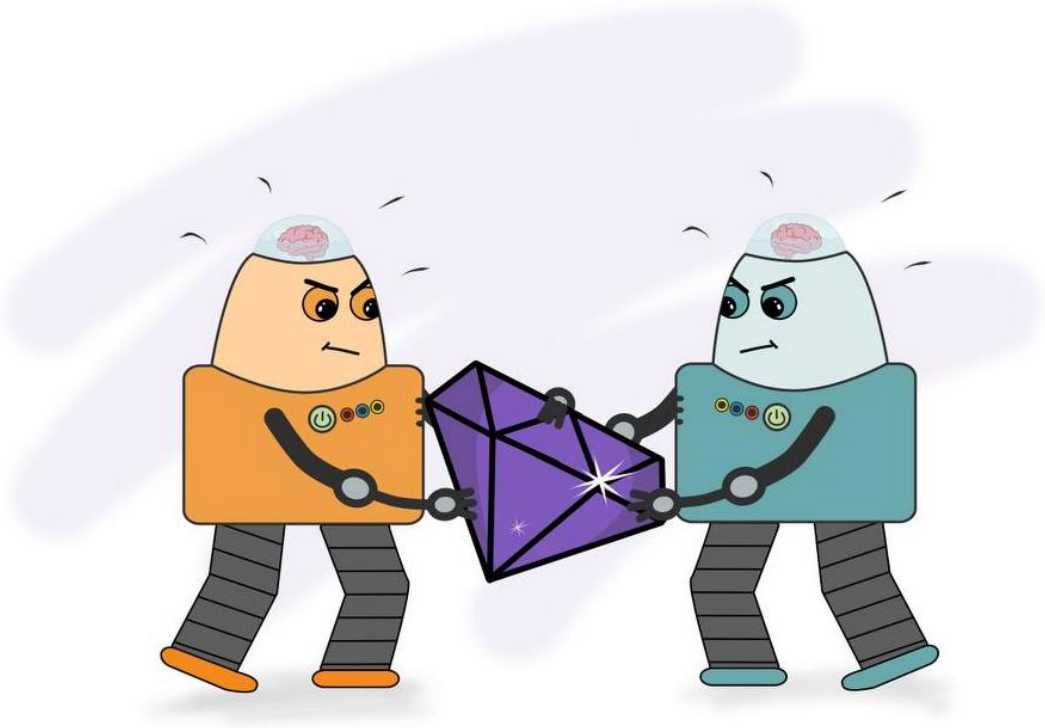


[Demo: mystery pacman (L6D1)]

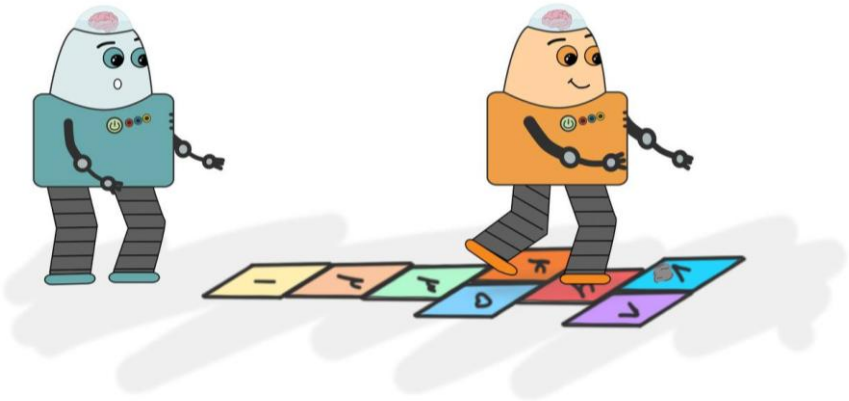
ویدیوی دموی راز پکمن



بازی‌های خصمانه (Adversarial Games)

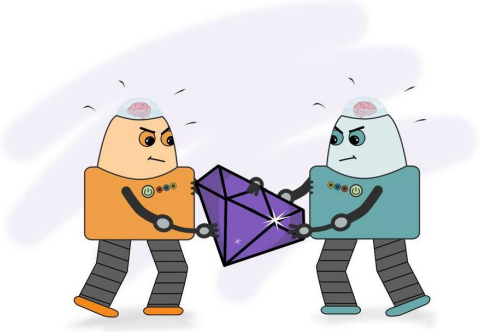


انواع بازی‌ها



- انواع زیادی از بازی‌ها وجود دارد!
- محورها:
  - قطعیت: قطعی یا تصادفی؟
  - تعداد عامل: یک بازیکن، دوتا، یا بیشتر؟
  - جمع 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 جمع صفر (Zero-Sum)



- بازی‌های جمع صفر (Zero-Sum):
  - عامل‌ها سودمندی متضاد دارند (ارزش خروجی‌ها)
  - می‌توانیم فرض کنیم که ارزشی (سومندی) وجود دارد که یکی آن را بیشینه و دیگری آن را کمینه می‌کند
  - خصمانه، رقابت خالص

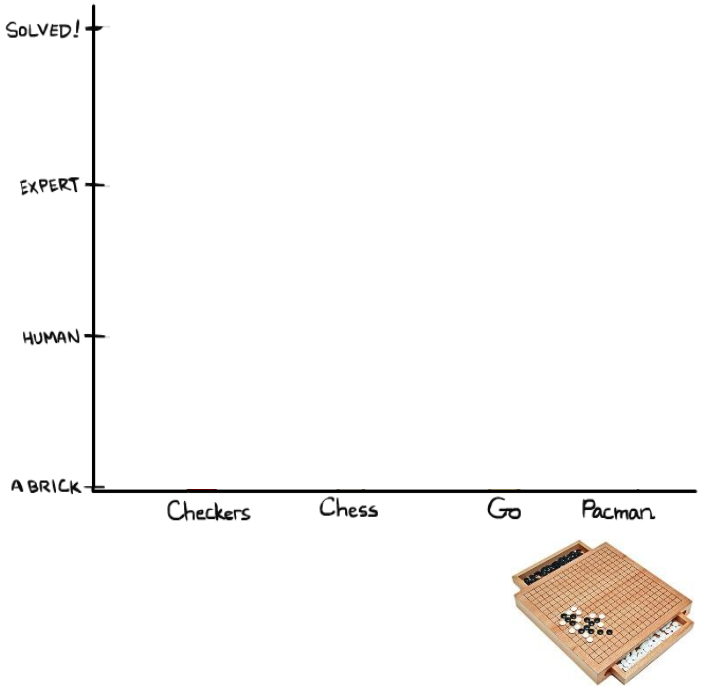
- بازی‌های عمومی (General Games):
  - عامل‌ها سودمندی مستقل دارند (ارزش خروجی‌ها)
  - همکاری، بی‌توجهی، رقابت و دیگر شرایط همگی ممکن است
  - اطلاعات بیشتر در بخش بازی‌های جمع غیر صفر

پیشرفته‌ترین روش بازی کردن

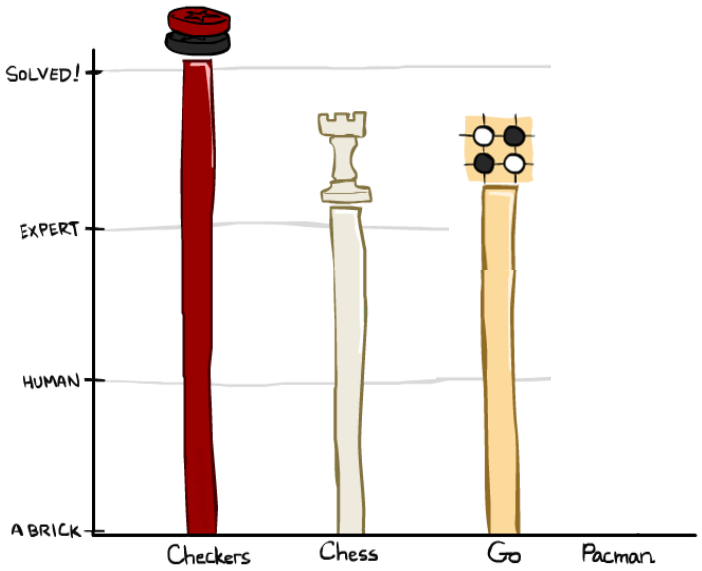
- **چکرز (Checkers)**: در سال 1950، اولین بازیکن کامپیوتری. در سال 1994، اولین قهرمان کامپیوتری: چینوک دوره قهرمانی 40 ساله قهرمان انسانی "ماریون تینسلی" را پایان داد. در سال 2007، چکرز حل شد!

- **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 پیچیده 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.



پیشرفته‌ترین روش بازی کردن



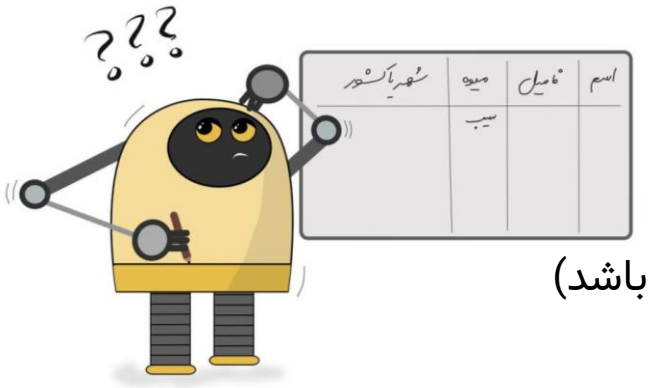
- **چکرز (Checkers)**: در سال 1950، اولین بازیکن کامپیوتری. در سال 1994، اولین قهرمان کامپیوتری: چینوک دوره قهرمانی 40 ساله قهرمان انسانی "ماریون تینسلی" را پایان داد. در سال 2007، چکرز حل شد!

- **شطرنج (Chess)**: در سال 1997، دیپ بلو (Deep Blue by IBM) قهرمان انسانی "گری کاسپاروف" را در یک مسابقه شش مرحله‌ای شکست داد. دیپ بلو 200 میلیون موقعیت در ثانیه را آزمایش کرد، از ارزیابی پیچیده 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 روش‌های اخیر از تکنیک‌های مونت کارلو (تصادفی) استفاده می‌کنند (AlphaGo by DeepMind).

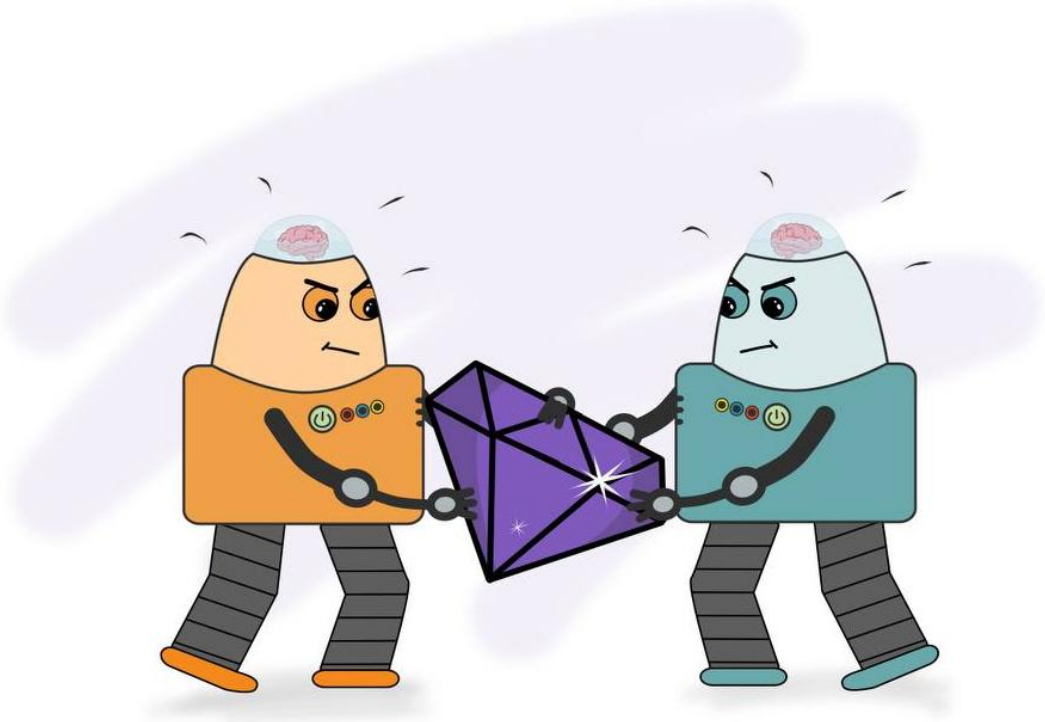
- **پکمن (Pacman)**

بازی‌های قطعی با سودمندی نهایی

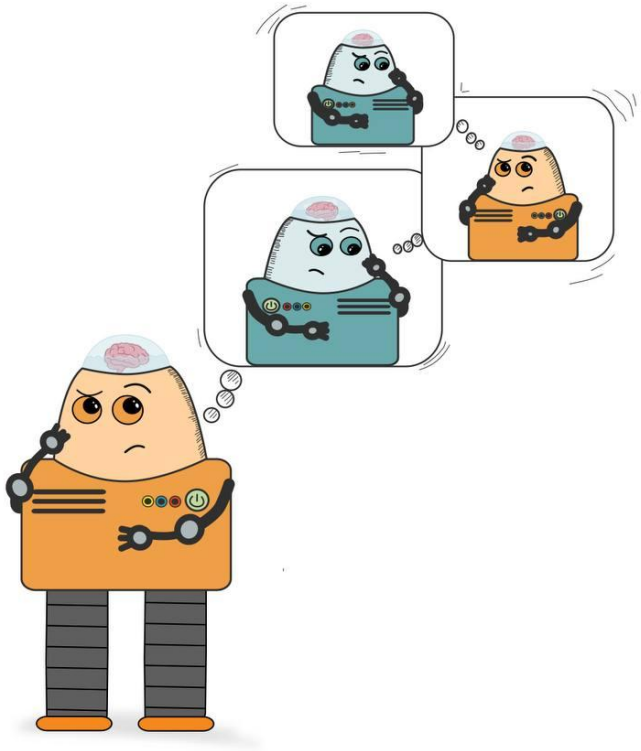


- فرمول‌بندی‌های زیادی ممکن است، به طور مثال:
  - حالت‌ها: $S(\text{start at } s_0)$
  - بازیکنان: $P = \{1 \dots N\}$ (معمولا به صورت نوبتی)
  - اعمال: A (ممکن است به بازیکن / حالت بستگی داشته باشد)
  - تابع انتقال (Transition Function): $S \times A \rightarrow S$
  - آزمون پایانی (Terminal Test): $S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

بازی‌های خصمانه (Adversarial Games)



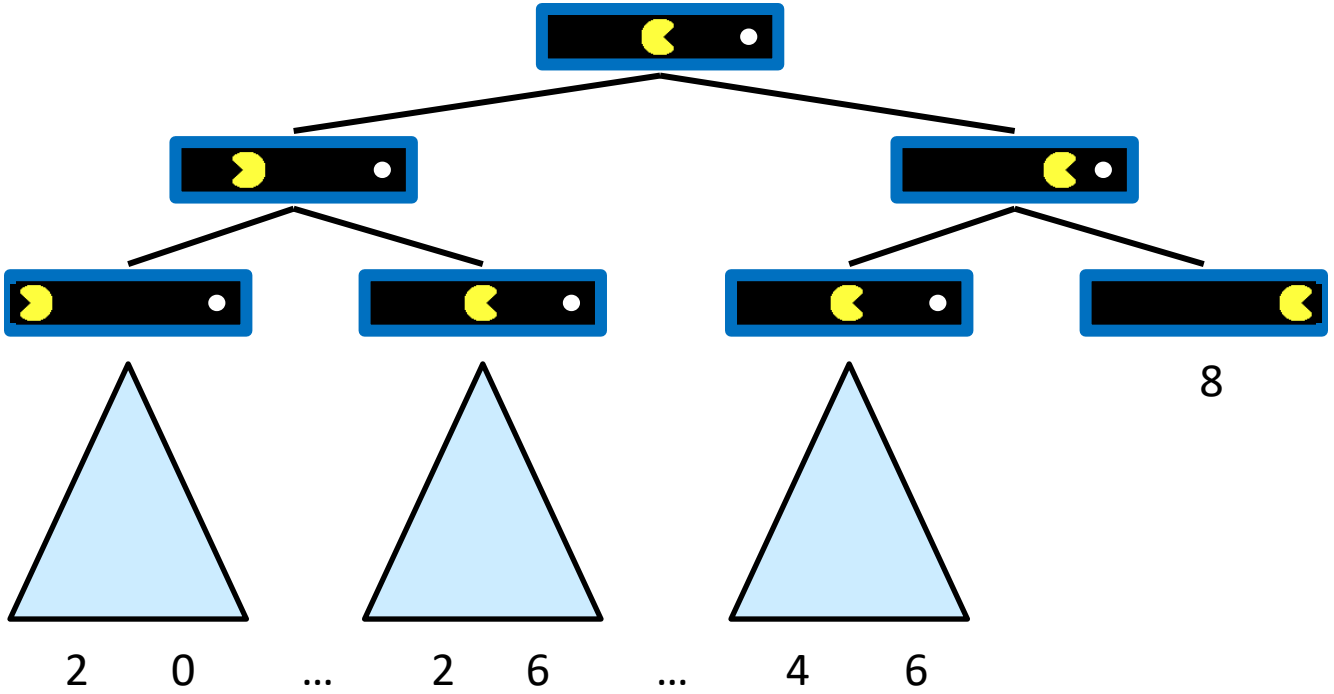
جستجوی خصمانه (Adversarial Search)



خبر جدید: هزینه -> سودمندی

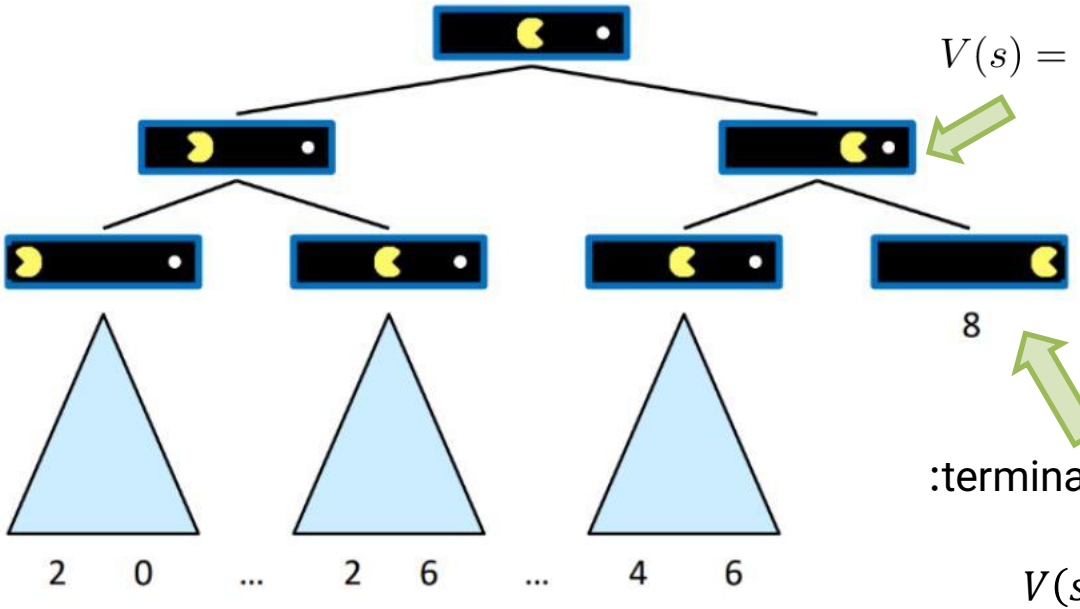
- دیگر به دنبال کمینه کردن هزینه نیستیم
- در اینجا عامل می‌خواهد امتیاز / سودمندی خود را بیشینه کند!

درخت‌های تک عاملی (Single-Agent Trees)



ارزش یک حالت (Value of a state)

ارزش یک حالت:
بهترین نتیجه (سود) قابل رسیدن از این حالت



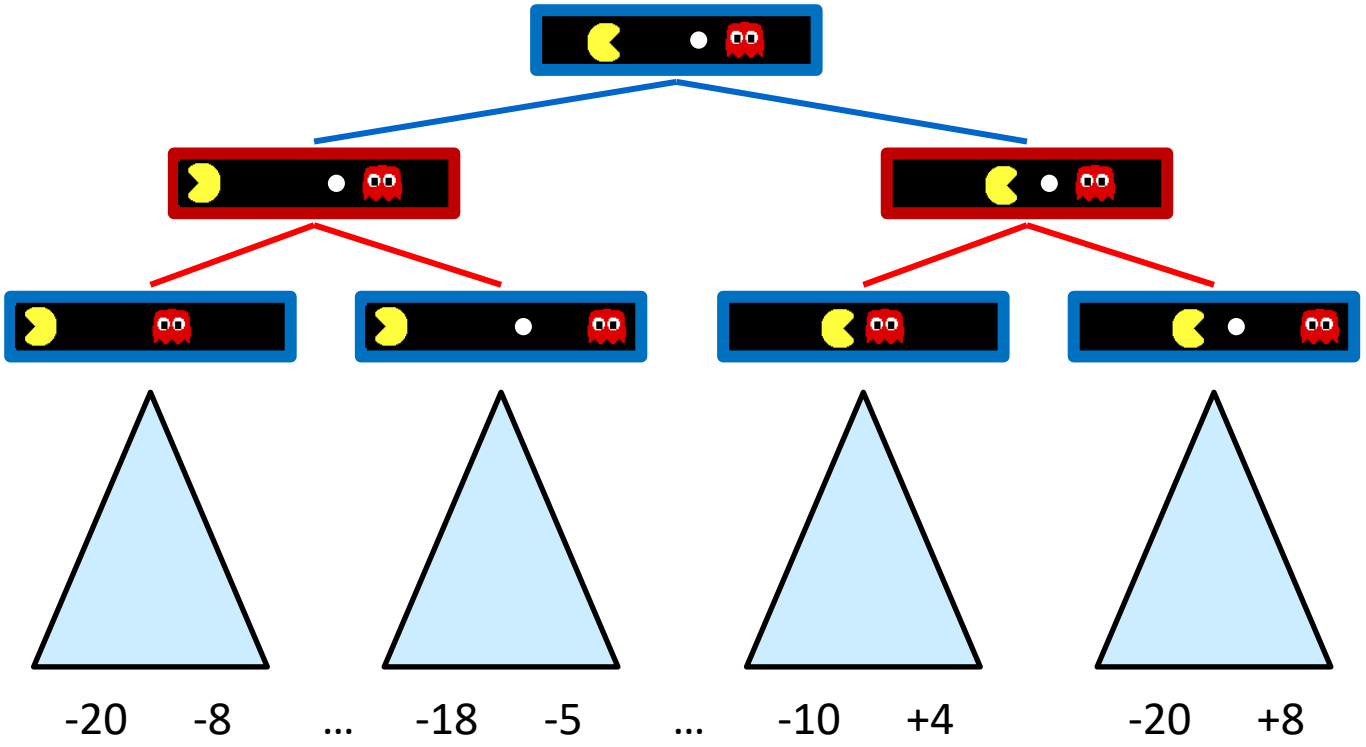
حالت‌های غیرپایانی:

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

حالت‌های پایانی terminal:

$V(s) =$ معلوم

درخت‌های بازی خصمانه



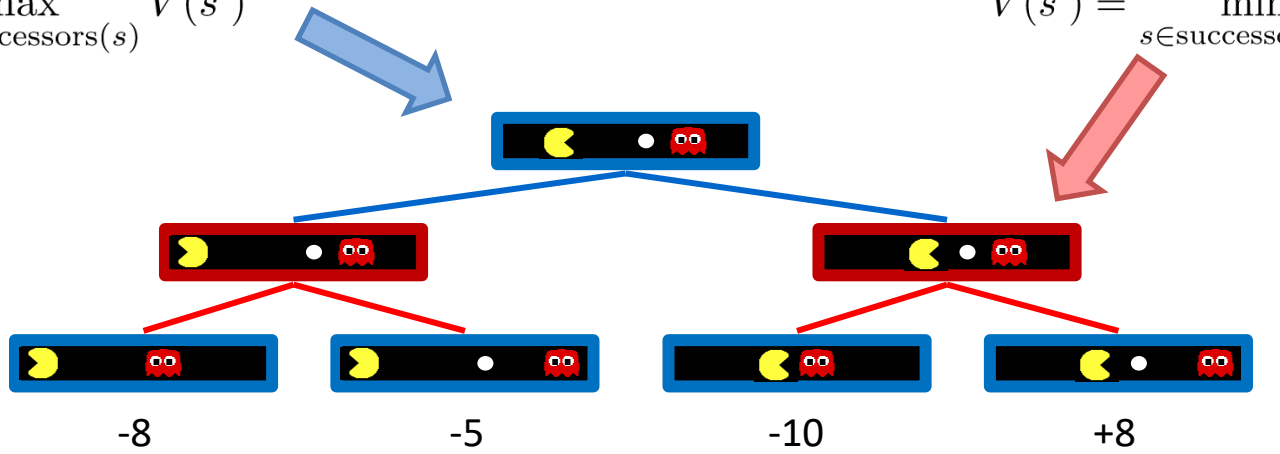
ارزش‌های کمینه-بیشینه (Minimax Values)

حالت‌های تحت کنترل عامل:

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

حالت‌های تحت کنترل حریف:

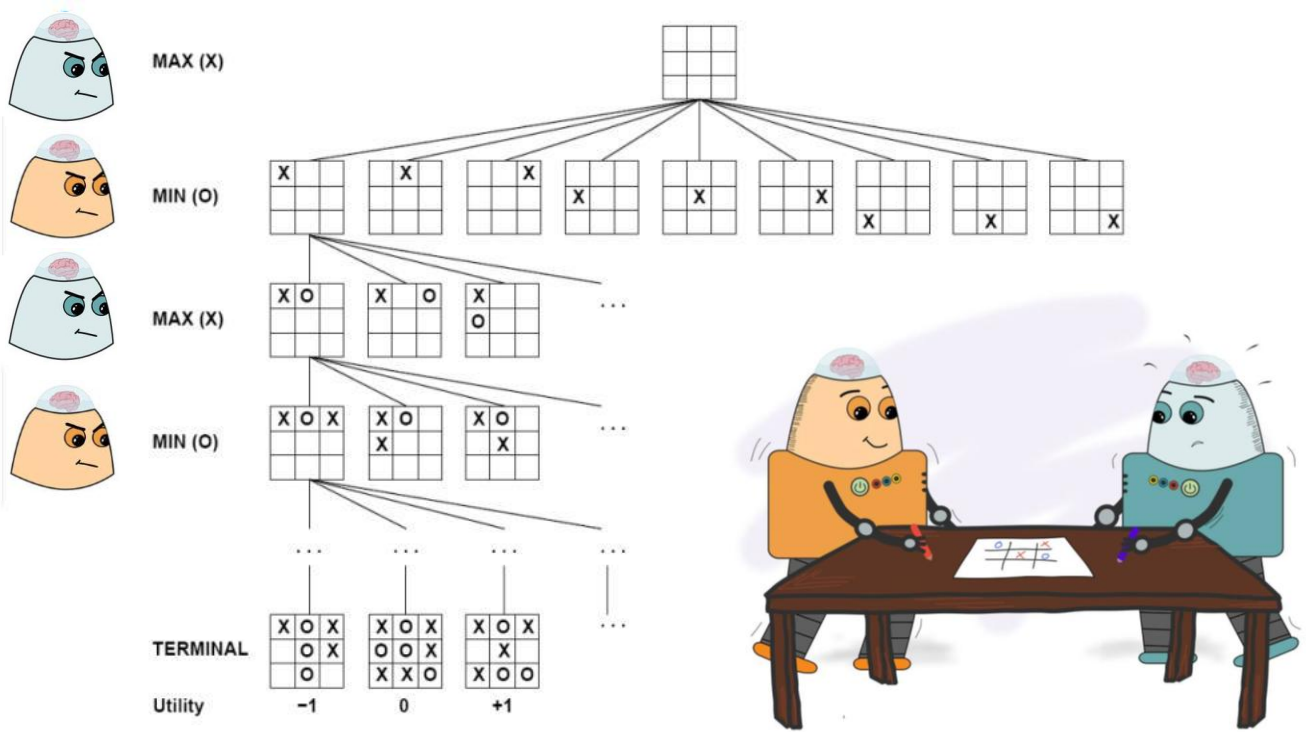
$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$



حالت‌های پایانی terminal:

$V(s) =$ معلوم

درخت بازی دوز



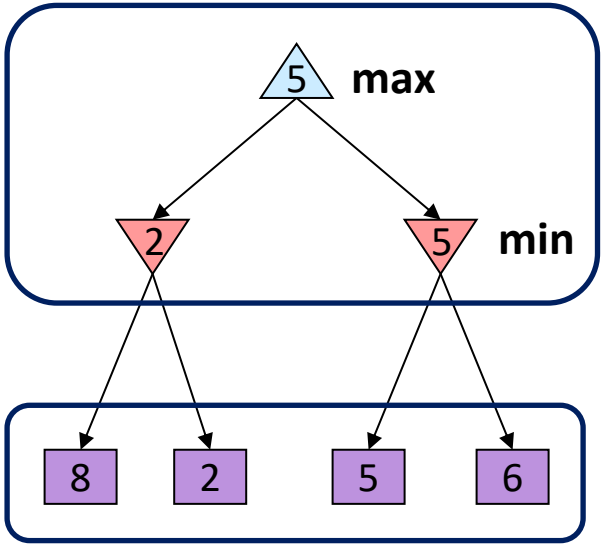
جستجوی تخاصمی (کمینه-بیشینه) Minimax

- بازی‌های جمع 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)

مقادیر کمینه-بیشینه:

به صورت بازگشتی محاسبه می‌شوند

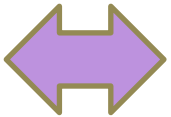


مقادیر پایانی:

بخشی از بازی

پیاده‌سازی کمینه-بیشینه

```
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
```

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

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

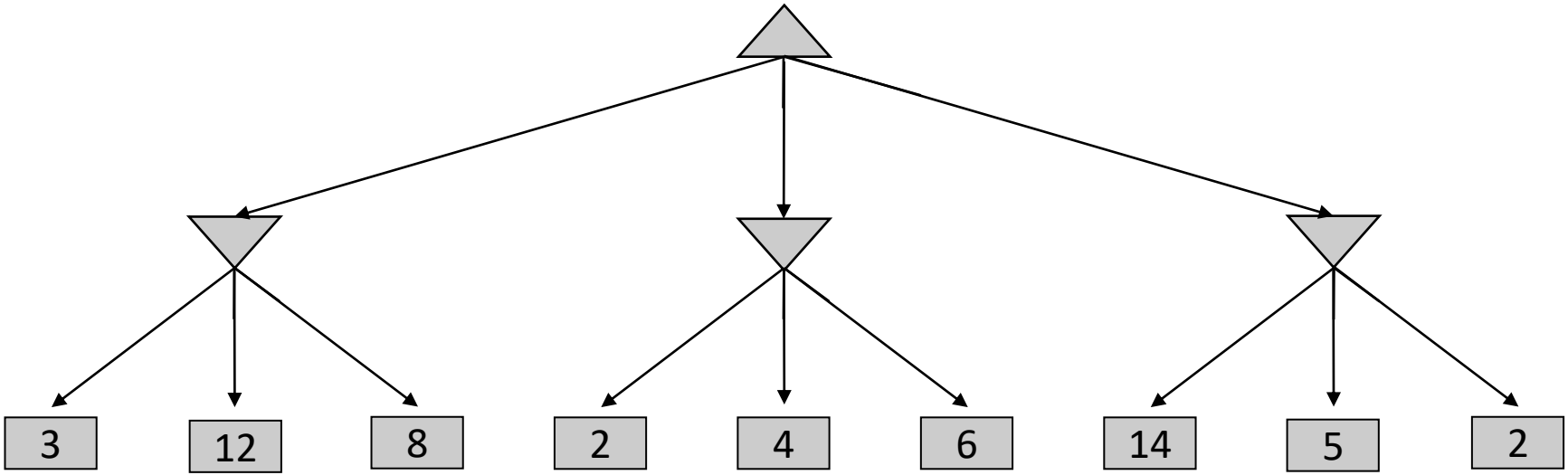
پیاده‌سازی کمینه-بیشینه (پخش)

```
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)
```

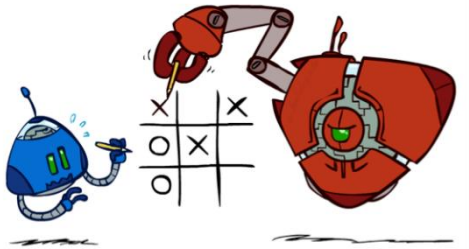
```
def max-value(state):
        initialize v = -∞
        for each successor of state:
                v = max(v, value(successor))
        return v
```

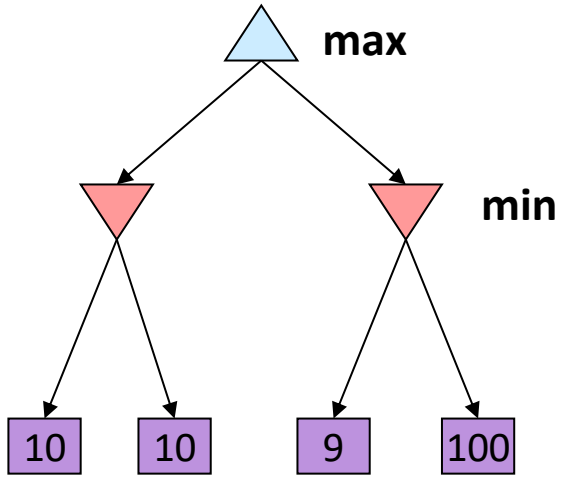
```
def min-value(state):
        initialize v = +∞
        for each successor of state:
                v = min(v, value(successor))
        return v
```

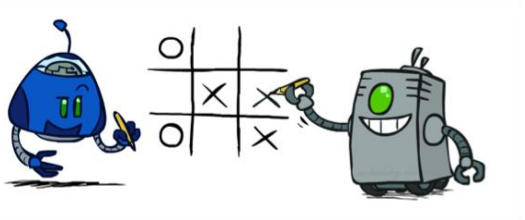
مثال کمینه-بیشینه



ویژگی‌های کمینه-بیشینه

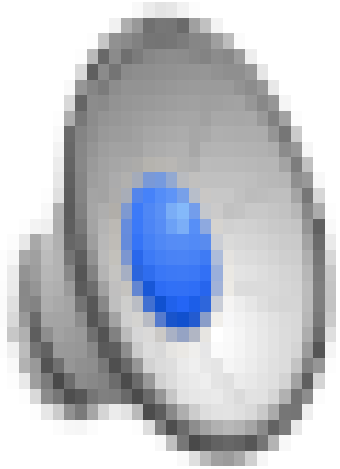




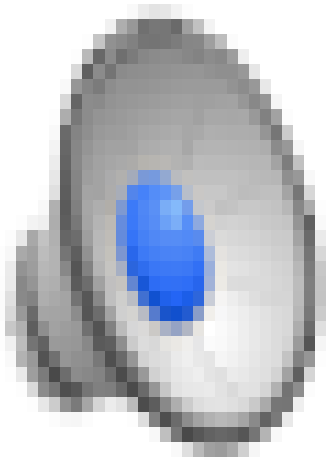


بهینه در برابر یک حریف بی‌نقص.در غیر اینصورت؟

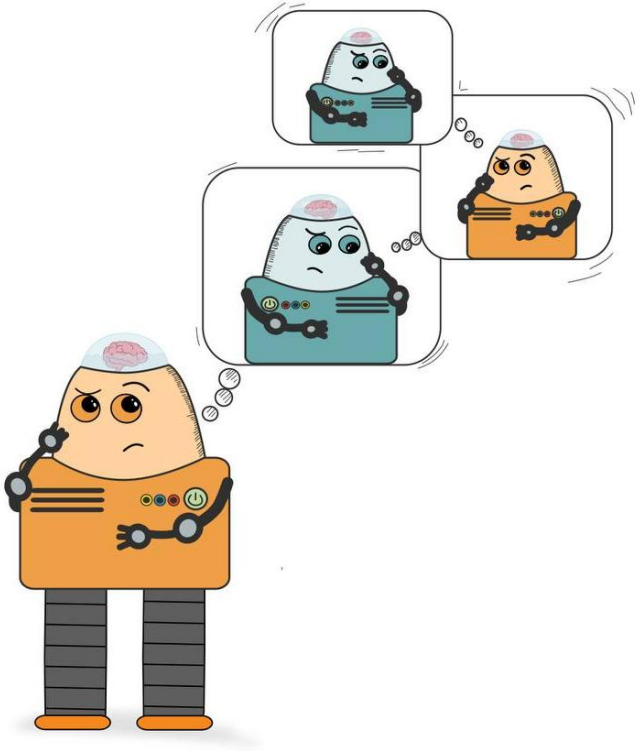
ویدیوی نمایشی Min vs. Exp (Min)



ویدیوی نمایشی Min vs. Exp (Exp)

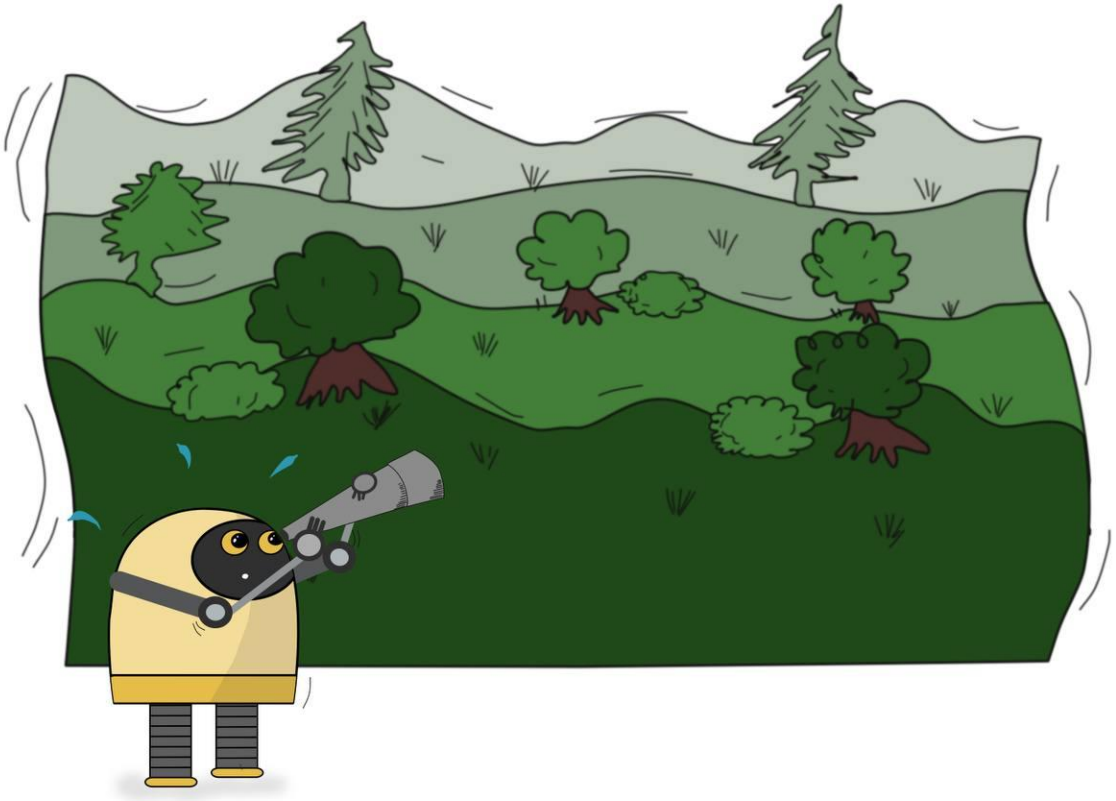


بهینگی کمینه-بیشینه

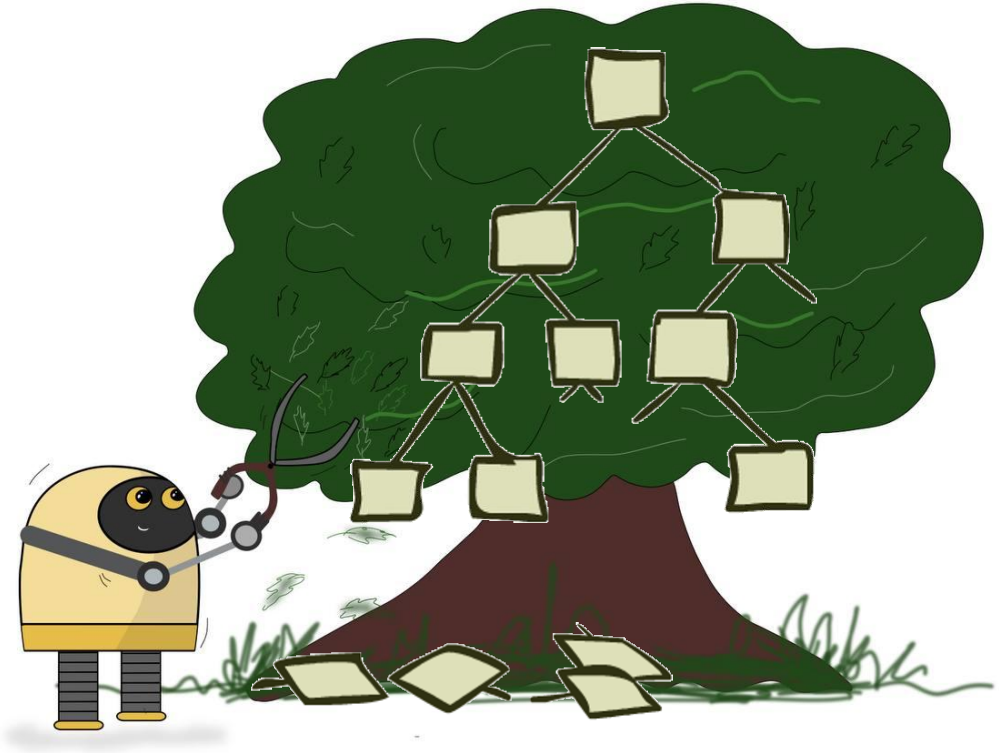


- کمینه-بیشینه چقدر بهینه است؟
  - مثل DFS (کامل)
  - پیچیدگی زمانی: $O(b^m)$
  - پیچیدگی مکانی: $O(bm)$
- مثال: برای شطرنج، $b \approx 35$، $m \approx 100$
  - راه‌حل دقیق غیر دستیافتنی است
  - ولی، آیا باید کل درخت را بگردیم؟

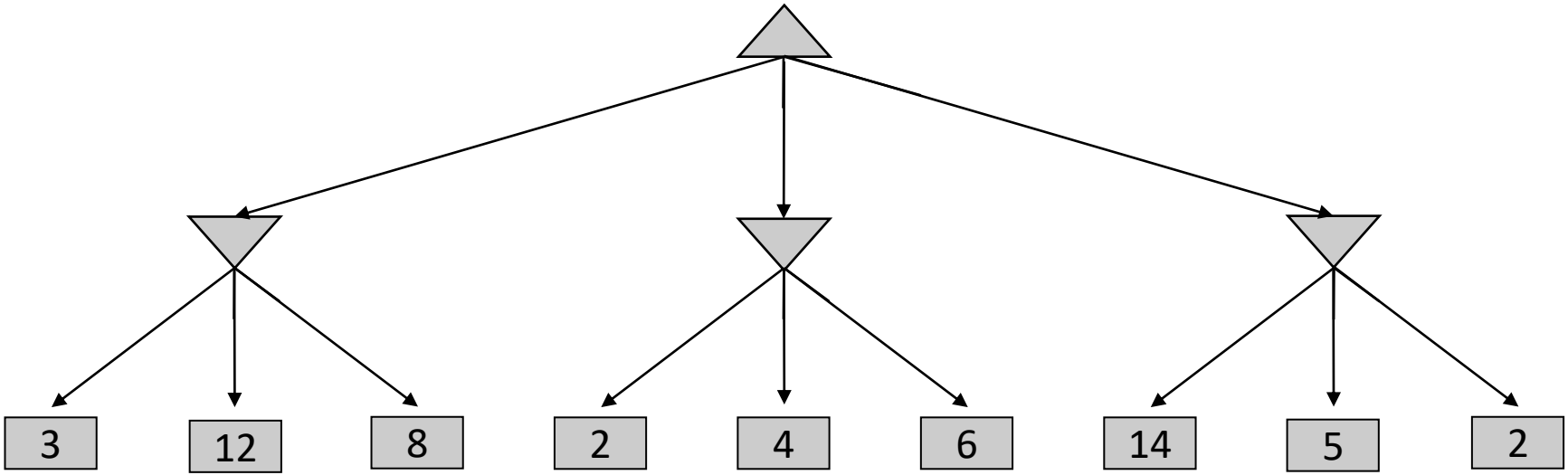
محدودیت منابع



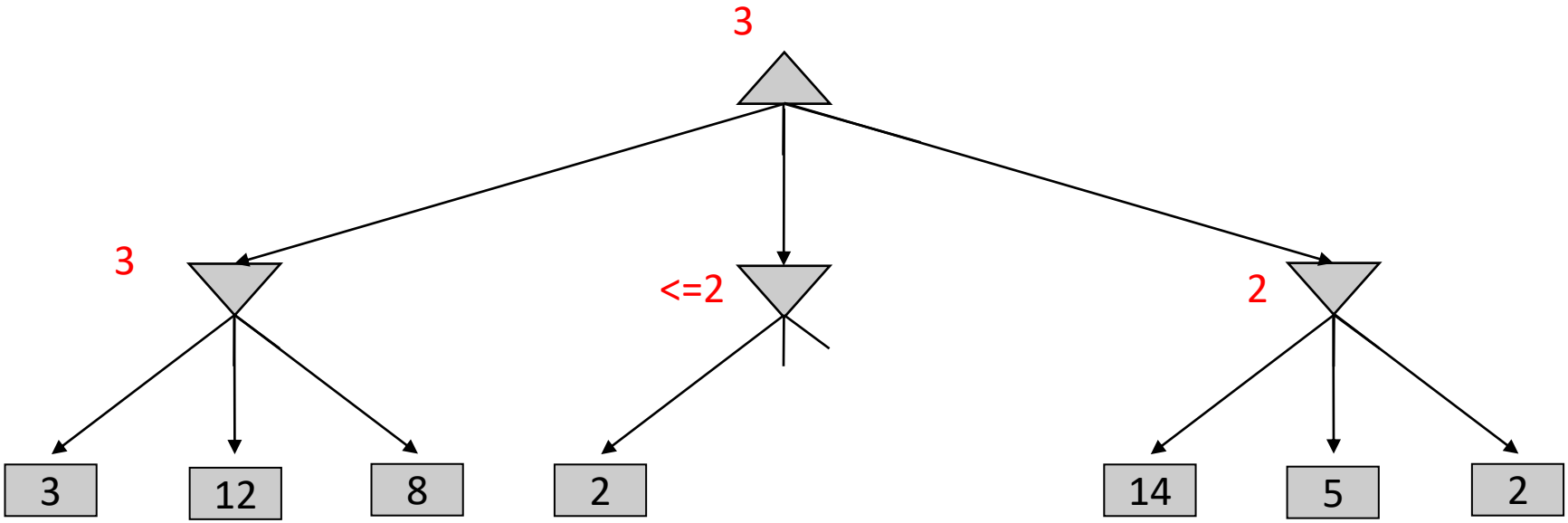
هرس درخت بازی (Game Tree Pruning)



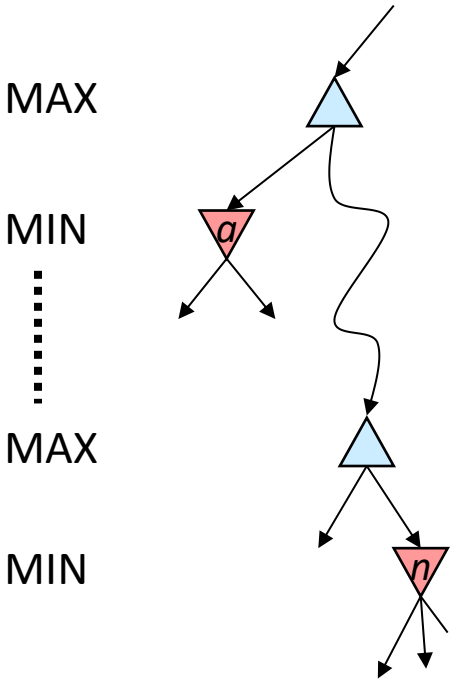
مثال کمینه-بیشینه



هرس کمینه-بیشینه (Minimax Pruning)



هرس آلفا-بتا (Alpha-Beta Pruning)



- ساختار کلی (نسخه کمینه)
  - در حال محاسبه مقدار کمینه در یک گره دلخواه n هستیم
  - و در این راستا، در حال بررسی و پیمایش فرزندان گره n هستیم
  - در طول بررسی، 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 مقدار 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

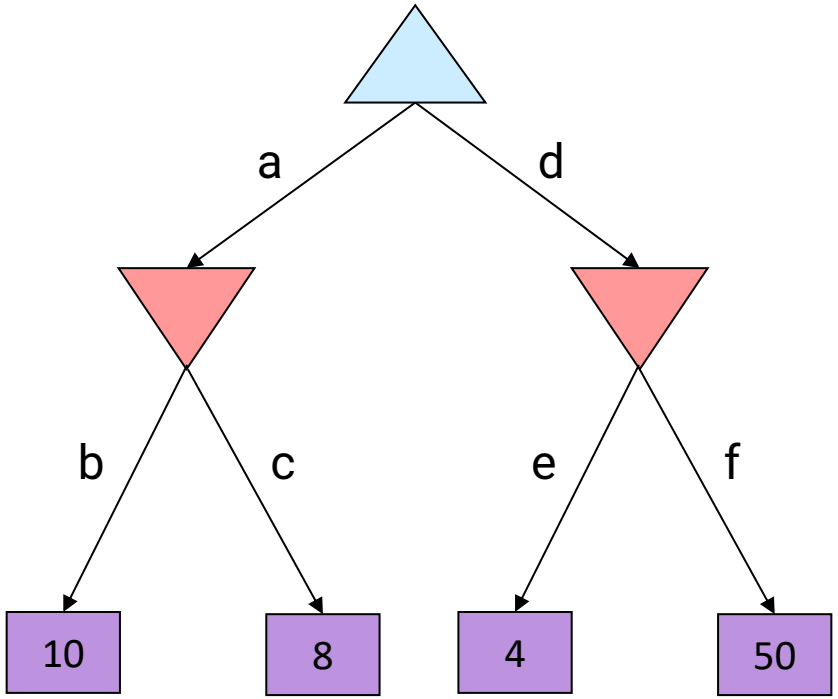
پیاده‌سازی آلفا-بتا

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 = -∞
    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
```

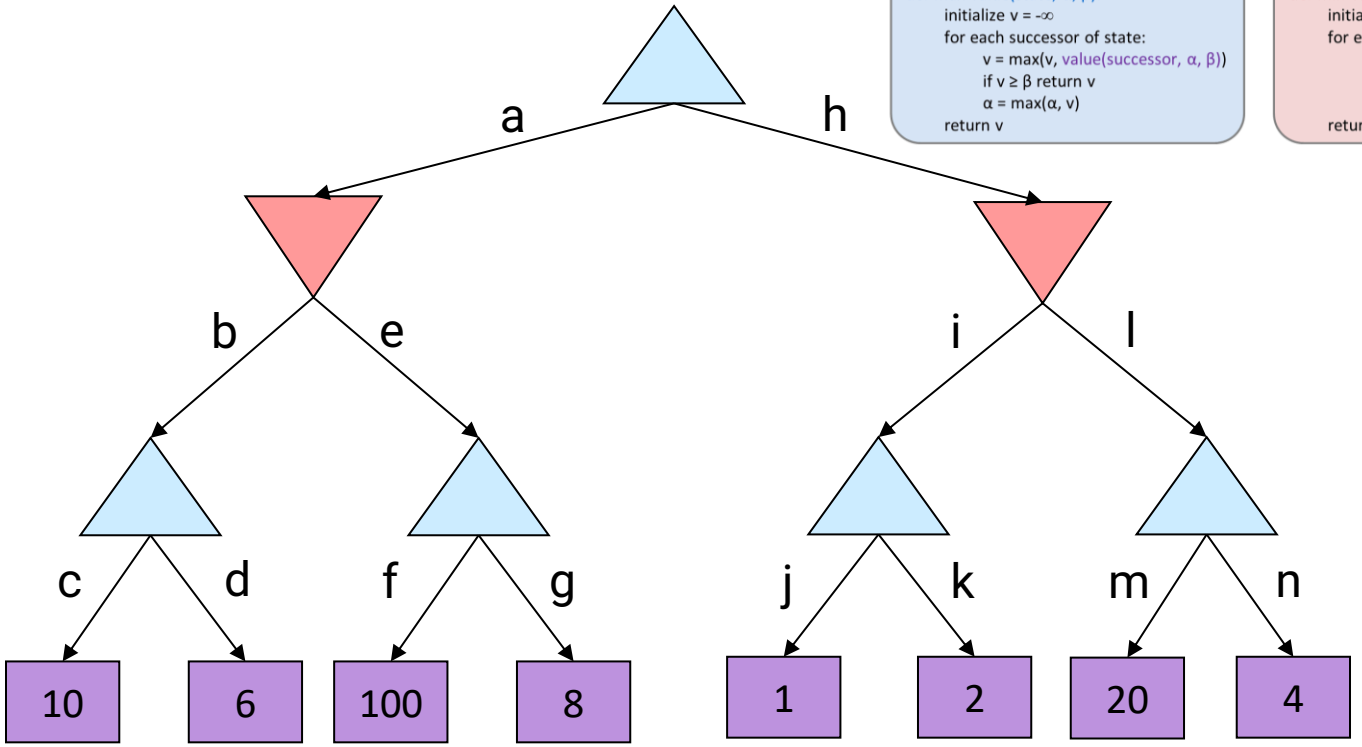
آزمونک آلفا-بتا



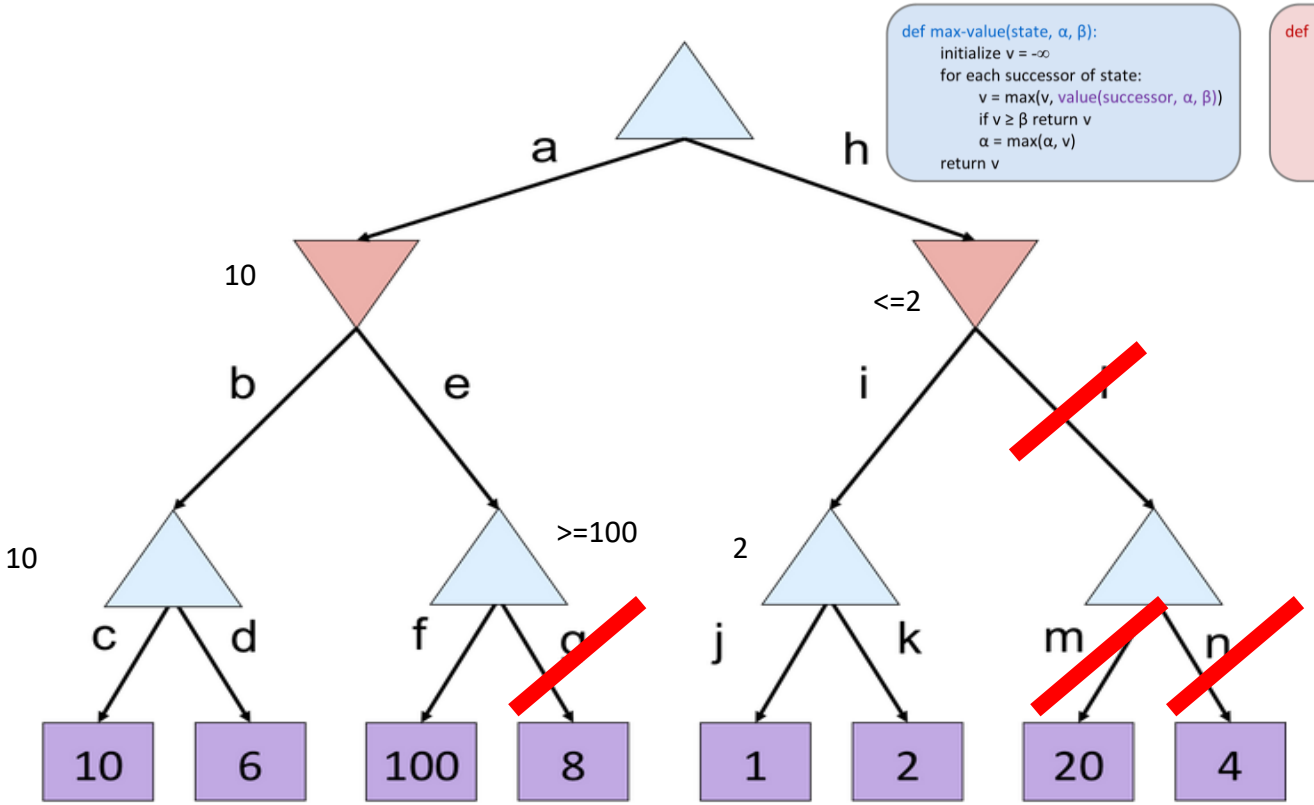
آزمونک آلفا-بتا 2

```
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
```

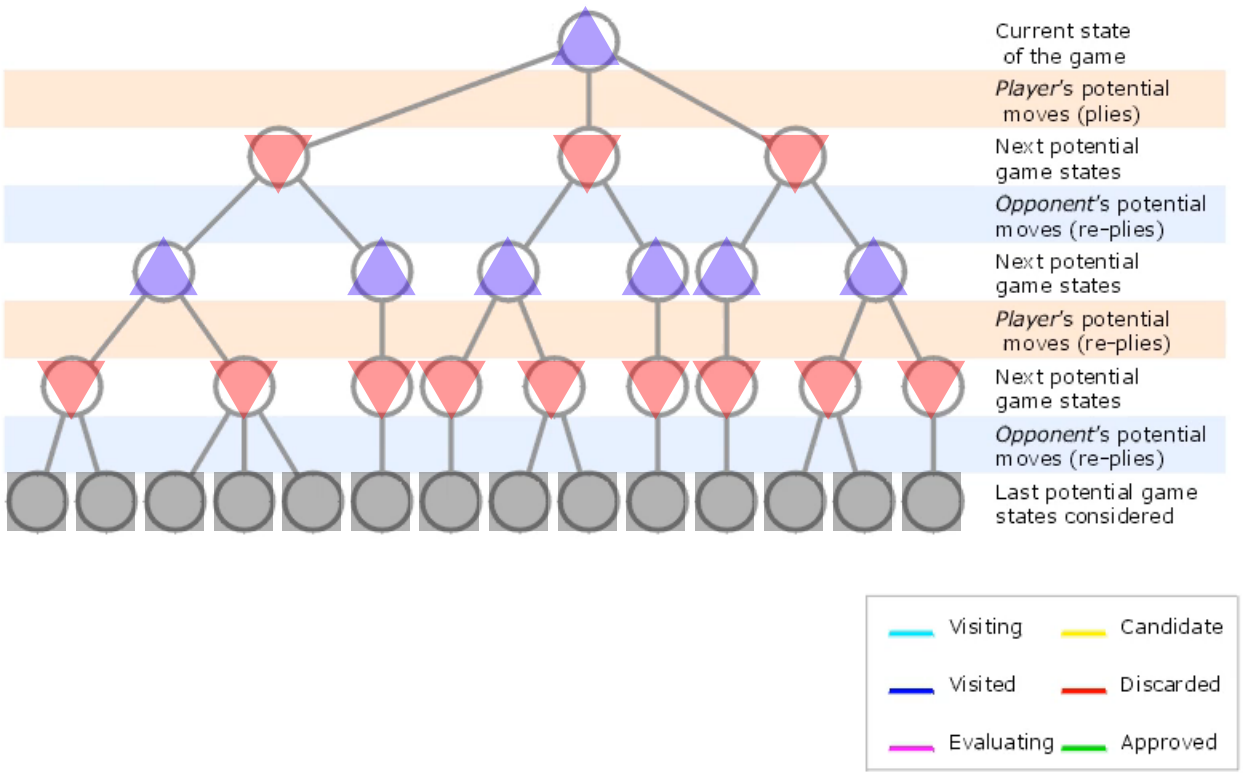
```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```



آزمونک آلفا-بتا 2



دموی آلفا-بتا

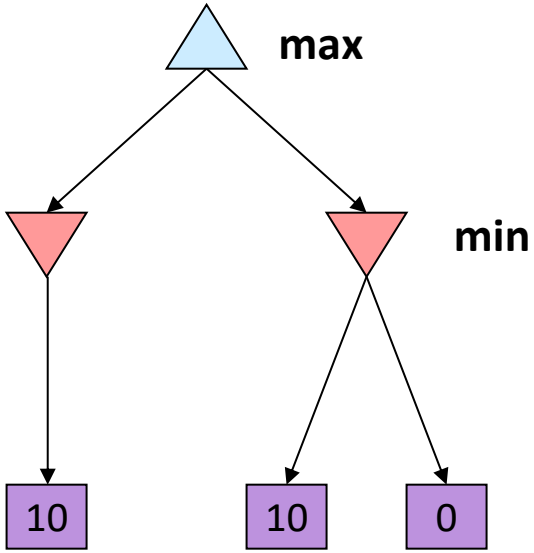


α: MAX's best option on path to root
β: MIN's best option on path to root

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

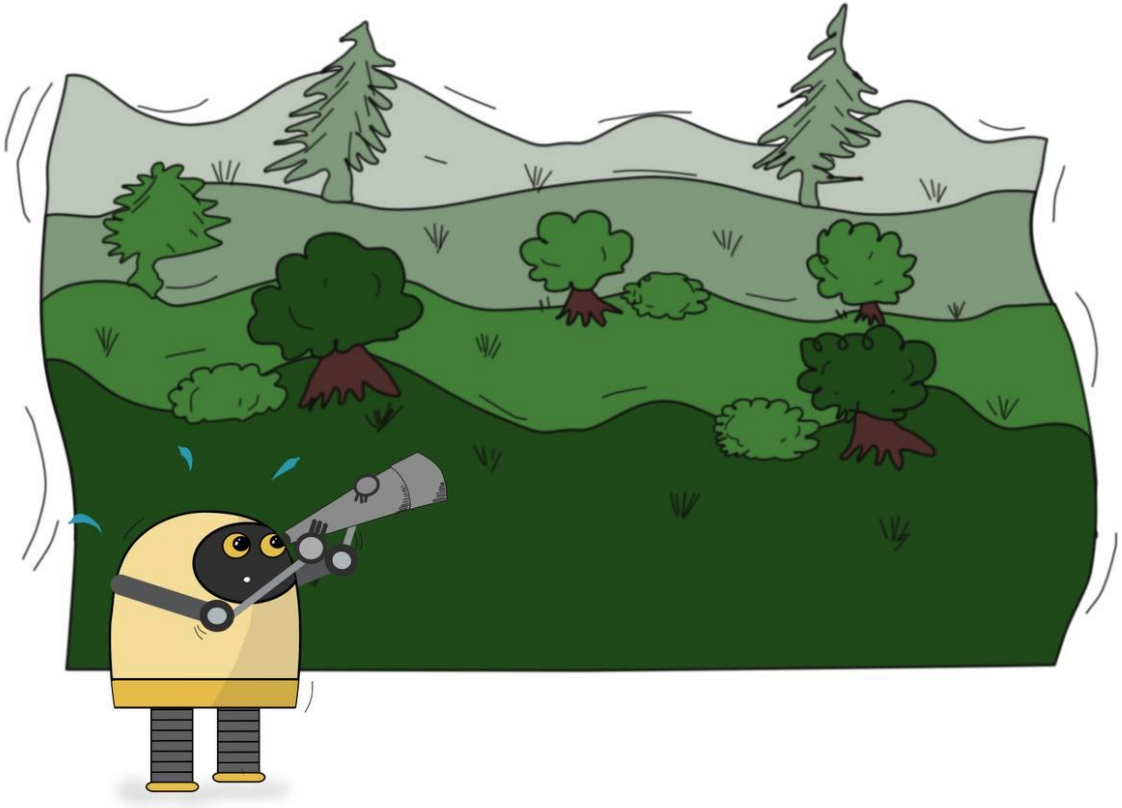
```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

ویژگی‌های هرس آلفا-بتا

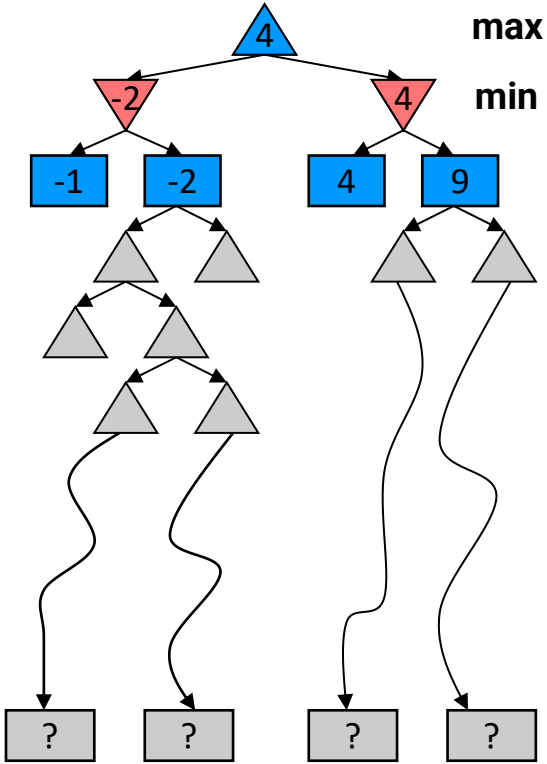


- این هرس هیچ اثری بر مقدار کمینه-بیشینه محاسبه شده برای ریشه ندارد!
- مقادیر گره‌های میانی ممکن است اشتباه باشد
  - مهم: فرزندان ریشه ممکن است مقادیر اشتباهی داشته باشند
  - بنابراین ساده‌ترین نسخه به شما اجازه انتخاب عمل را نمی‌دهد
- مرتب سازی خوب فرزندان، اثرگذاری هرس را افزایش می‌دهد
- با "مرتب سازی 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، مثلا در شطرنج، همچنان ممکن نیست...
- این یک مثال ساده از metareasoning است. (محاسبه اینکه چه چیزهایی محاسبه شوند)

محدودیت منابع

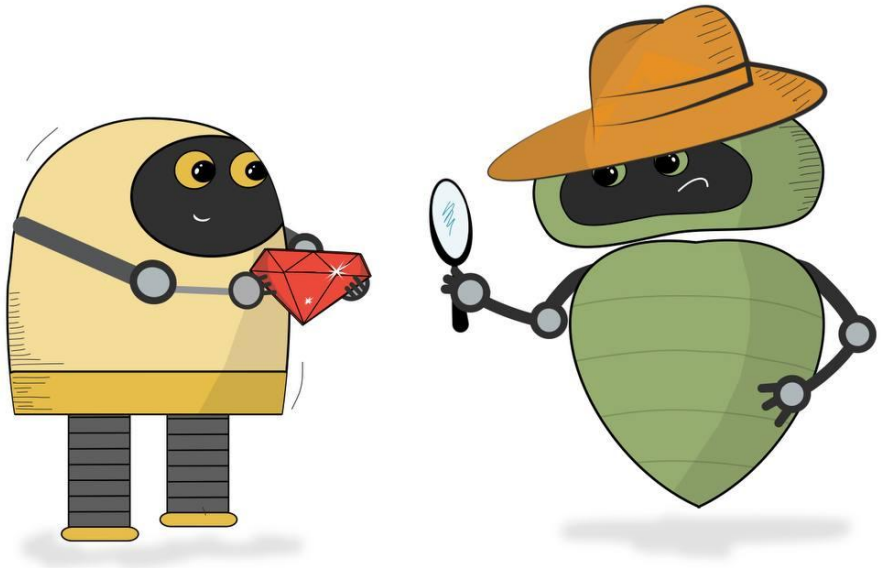


محدودیت منابع



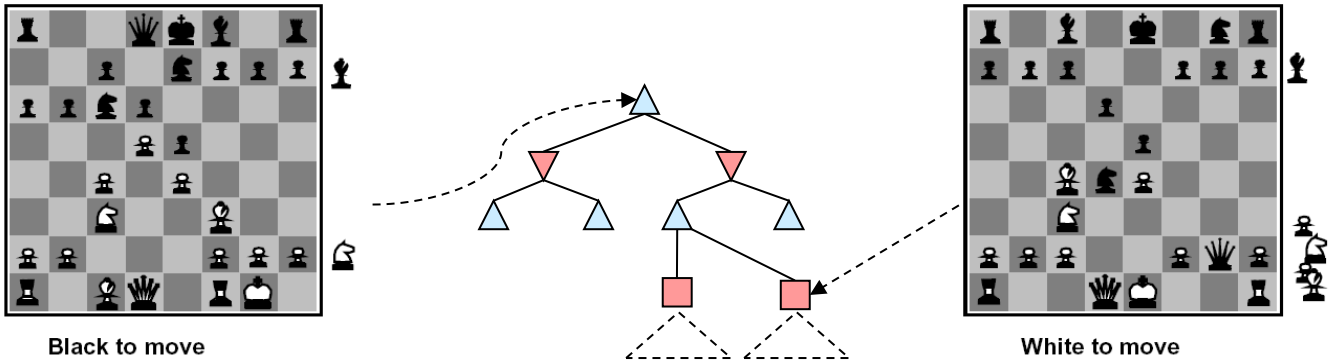
- مشکل: در بازی‌های واقعی، نمی‌توان تا برگ‌ها جستجو کرد!
- راه‌حل: جستجو با عمق محدود شده (Depth-limited Search)
  - به جای اینکار، تا یک عمق محدودی از درخت را جستجو می‌کنیم
  - جایگزینی ارزش پایانی با تابع ارزیابی برای حالت‌های غیرپایانی
- مثال:
  - فرض: 100 ثانیه وقت داریم، جستجو با سرعت 10k گره بر ثانیه
  - بنابراین می‌توانیم 1M گره را در هر حرکت بررسی کنیم
  - در بازی شطرنج α-β تقریبا به عمق 8 می‌رسد
- بجای planning از این به بعد replanning
- ضمانت بازی بهینه از بین می‌رود
- بررسی لایه‌های بیشتر باعث ایجاد تفاوت **بزرگ** می‌شود
- استفاده از افزایش مرحله‌ای عمق، برای هر الگوریتم‌های هرزمانی (Anytime Algorithm)

توابع ارزیابی (Evaluation Functions)



توابع ارزیابی

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

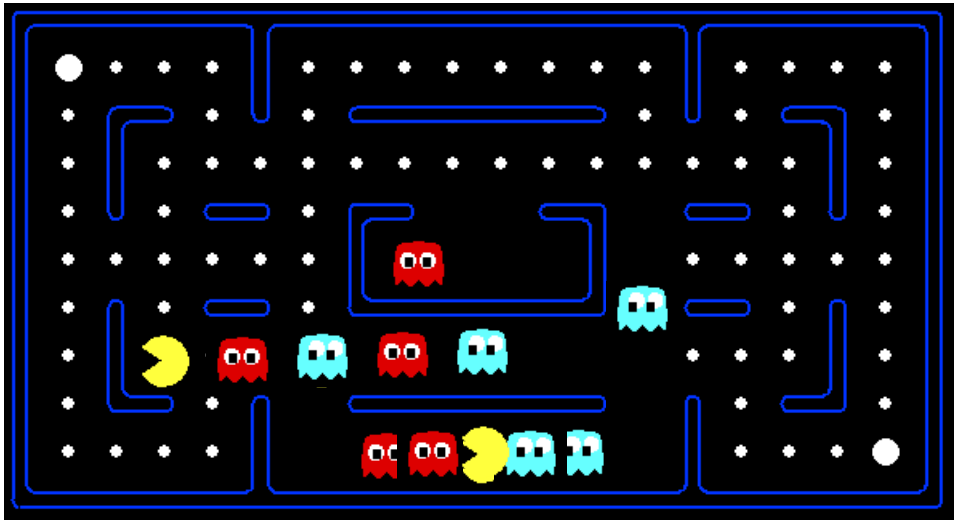


- تابع ایده‌آل: مقدار واقعی کمینه-بیشینه حالت را بر می‌گرداند
- درعمل: جمع جبری وزن دار ویژگی‌ها:

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

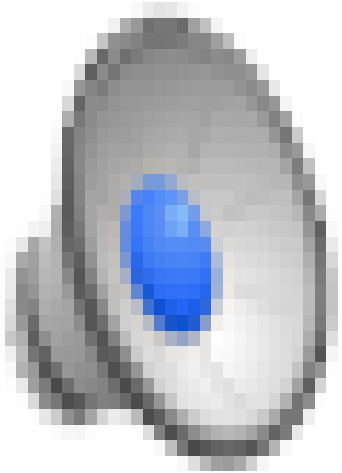
- مثال: $f_1(s)$ = (num white queens – num black queens) ، و غیره

ارزیابی برای پکمن

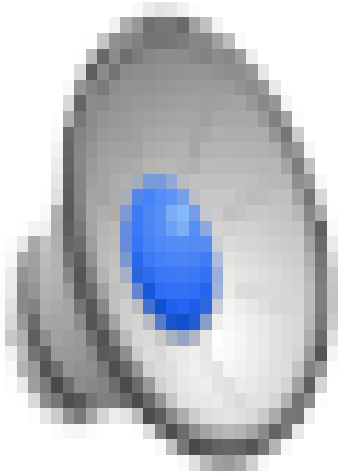


[Demo: thrashing d=2, thrashing d=2 (fixed evaluation function), smart ghosts coordinate (L6D6,7,8,10)]

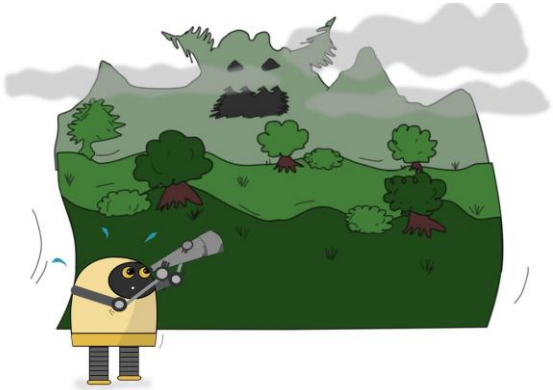
ویدیوی نمایشی Smart Ghosts (Coordination)

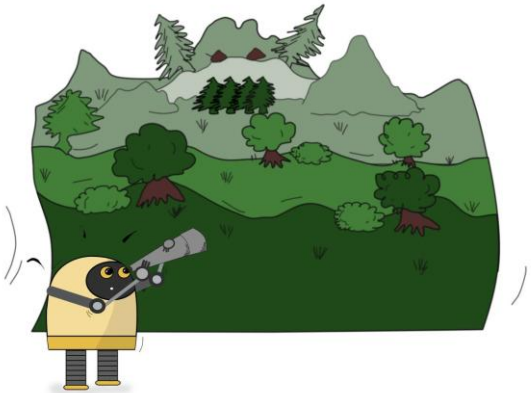


Smart Ghosts (Coordination) – Zoomed In ویدیوی دموی



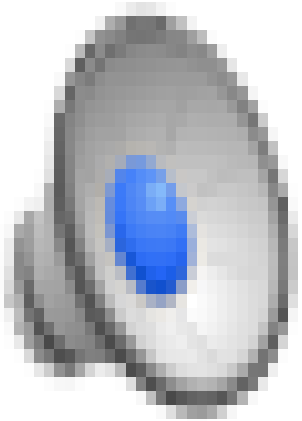
عمق اهمیت دارد





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

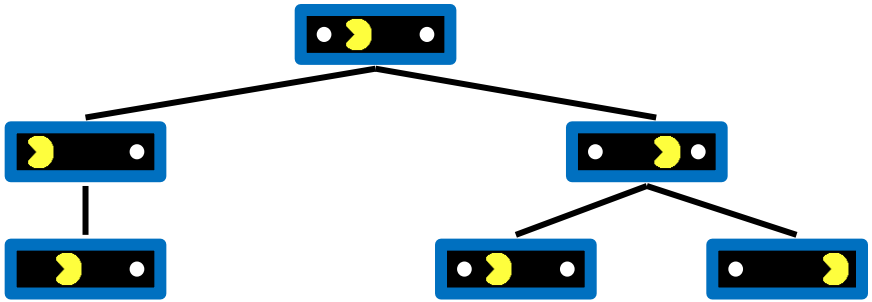
ویدیوی نمایشی Thrashing (d=3)



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

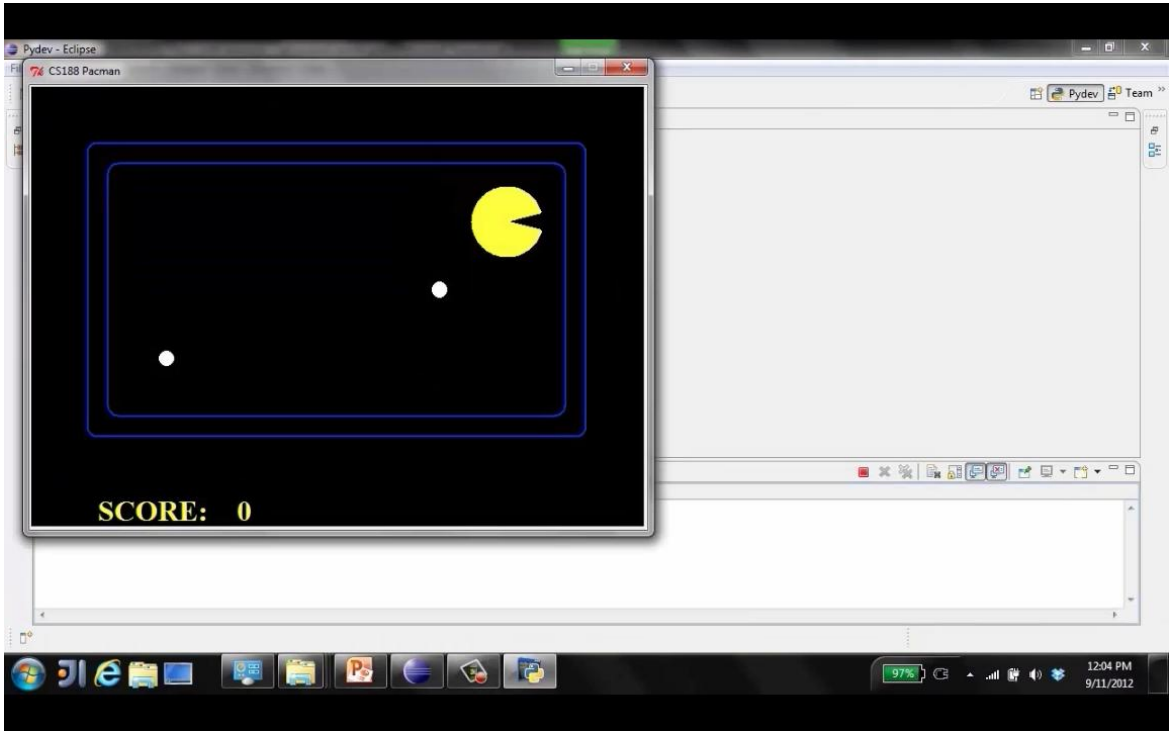
[Demo: thrashing d=3, thrashing d=3 (fixed evaluation function) (L6D6)]

چرا پکمن گرسنه می‌ماند



- 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 دوتاست)
  - 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!

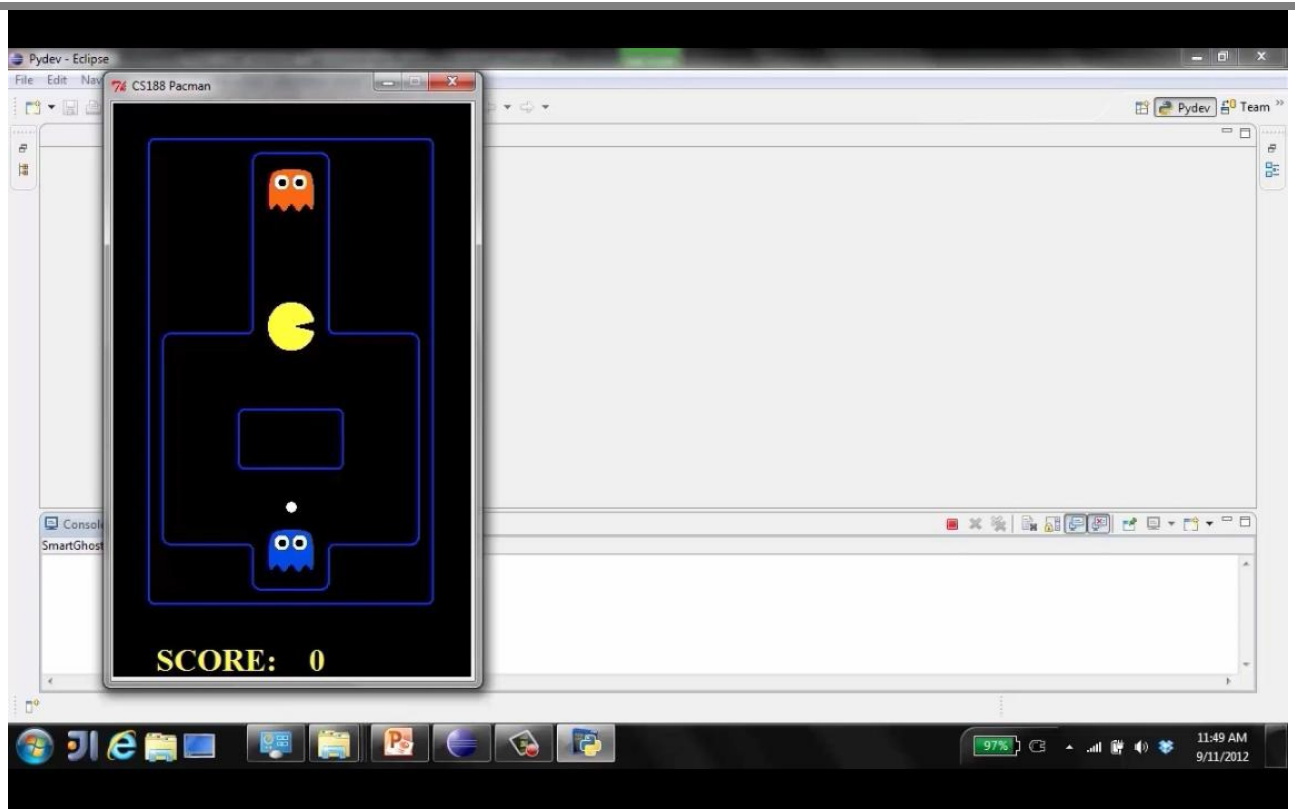
Thrashing -- Fixed (d=3) ویدیوی نمایشی



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

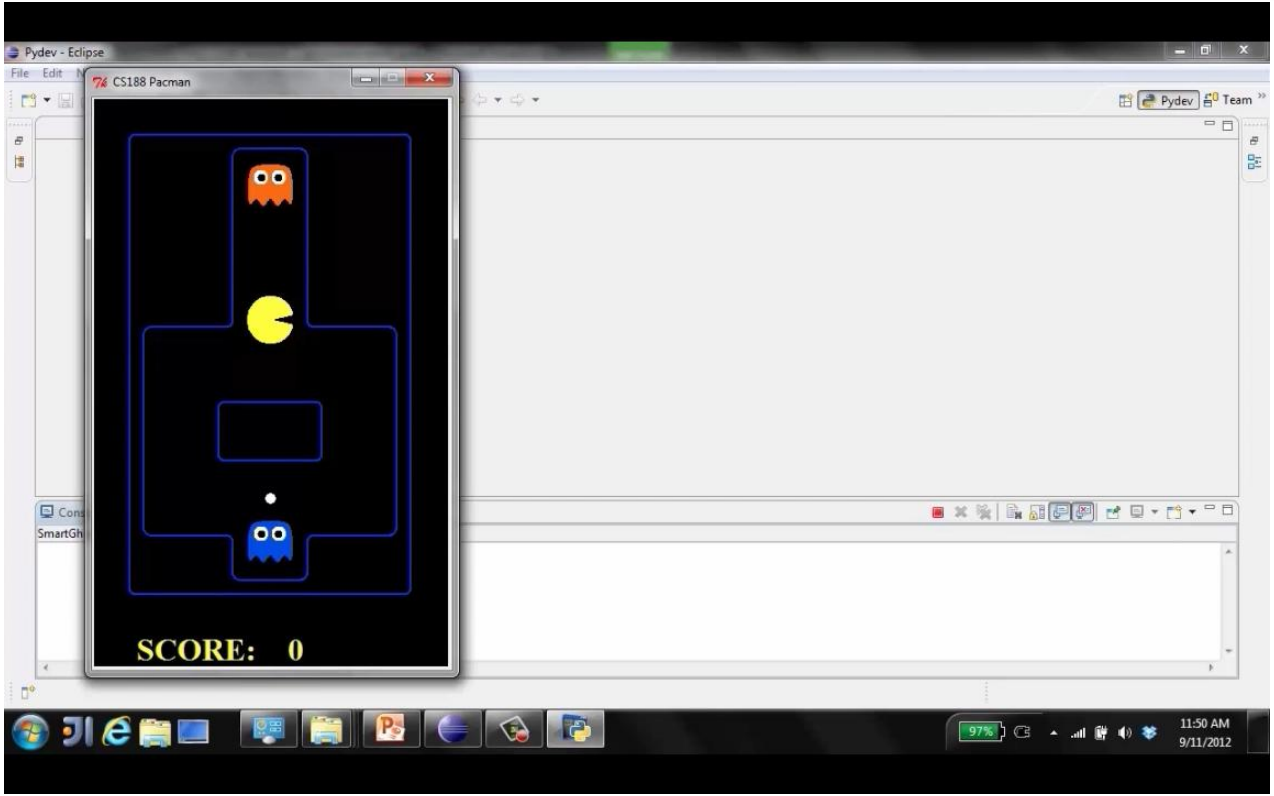
[Demo: thrashing d=3 thrashing d=3 (fixed evaluation function) (L6D7)]

دموی عمق محدود 3



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

دموی عمق محدود 10



Eval(s)=score for depth 10 + credit for being closer to dot + credit for being far from ghosts

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 بسط دهیم
    - (تقریبا شبیه نقش هیوریستیک در A^*، ترتیب در 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!