

Yapay Zeka

Ders 7

Doç. Dr. Mehmet Dinçer Erbaş
Bolu Abant İzzet Baysal Üniversitesi
Mühendislik Fakültesi
Bilgisayar Mühendisliği Bölümü

Oyunlar

- Çok ajanlı sistemler => ilk hafta bahsetmiştık
 - ~ Karşı tarafın aksiyonları belirsizlik yaratıyor.
- Bu bölümde yarışmacı çevreleri inceleyeceğiz.
 - ~ Mücadeleci arama problemlerini inceleyeceğiz (İng: Adversarial search)
 - Bu problemler oyun olarak adlandırılır.
- YZ'de incelenen oyunlar genellikle kazan/kaybet (ing: zero-sum games) tarzı, tam bilgiye ulaşabileceğimiz oyunlardır.
 - ~ Örneğin: Satranç
 - ~ Deterministik, tam gözlemlenebilir,
- Bu konu YZ konusunda uzun zamandır incelenmektedir.
- Oyunlar genellikle erişilebilir çevrelerdir: Çevre hakkında bütün bilgilere ulaşabiliriz.
- Arama: Oyun oynama mümkün oyun pozisyonları arasında arama işlemidir.
- Tahmin edilemeyen rakip: Karşıda bir rakip olması sistemde belirsizliğe neden olur.

Oyunlar

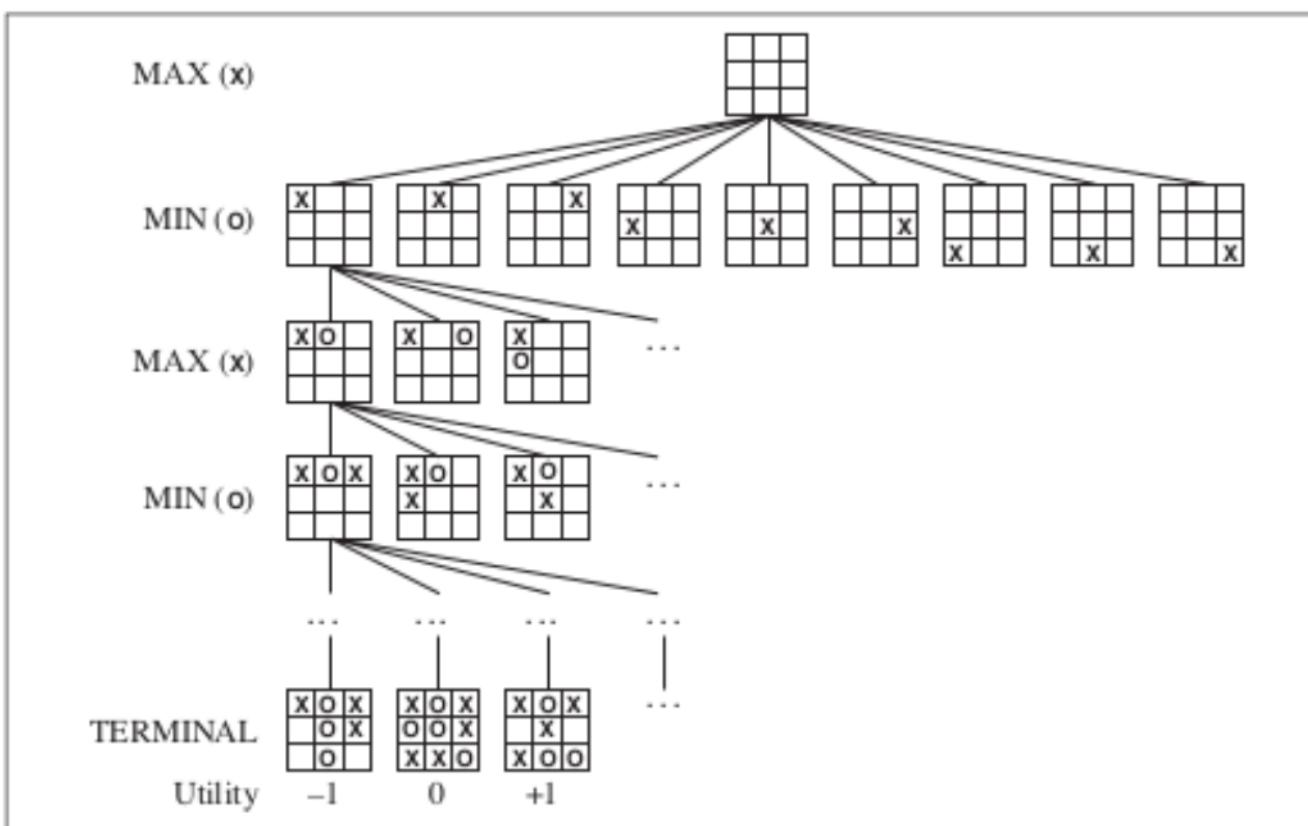
- Bir sonraki hamleyi belirlemek
 - ~ Karmaşıklık
 - Oyunlar genellikle oldukça büyük bir arama uzayına sahiptir.
 - ~ Örnek: Satranç => $b = 35$, $m = 100$, düğüm sayısı 35^{100} veya 10^{154} düğüm oluşturmak gerekecek.
 - Direk optimal çözüme ulaşmak mümkün değil.
 - Öyleyse bir şekilde mümkün olduğunda optimale yakın çözüme ulaşmaya çalışmalıyız.
 - Zaman kısıtlı olduğunda doğru bir karar vermek için kullanılabilecek yöntemleri öğreneceğiz.
 - ~ Örneğin budama (ing: pruning) arama ağacının belli parçalarını görmezden gelebilmemize olanak verecek.
 - ~ Örneğin bulussal değerlendirme fonksiyonları bir durumun yaklaşık yararını tam bir arama yapmadan hesaplamamamıza olanak verecek.

Oyunlar

- Öncelikle iki oyuncu içeren oyunları inceleyeceğiz.
 - ~ Oyunculardan birini MAX diğerini MIN olarak isimlendireceğiz
 - ~ Önce MAX, sonra MIN, sonra tekrar MAX,... şeklinde oynuyorlar
 - ~ Bir oyun resmi olarak şu şekilde tanımlayabiliriz:
 - S_0 : Başlangıç durumu, oyunun ilk olarak nasıl başladığı
 - OYUNCU(s): belli bir durumda oynamaya hakkı olan oyuncu
 - AKSİYON(s): belli bir durumda yapılabilecek aksiyonlar
 - SONUC(s,a): Geçiş modeli, bir aksiyonun sonucunu belirler.
 - BİTİŞ-TESTİ(s): Oyun sonlandığında bu test doğru döner.
 - YARAR(s,p): Yarar fonksiyonu oyunun sonlanması durumunda sonuca bir sayısal değer tanımlar.
 - ~ Örneğin satrançta, +1, 0 veya $\frac{1}{2}$.
 - ~ Sıfır-toplam oyunlar, her oyun oynanmasında toplam karşılığın sabit olduğu oyunlardır.

Oyunlar

- Başlangıç durumu, Aksiyonlar fonksiyonu ve sonuç fonksiyonu bir oyun ağacı tanımlar.
 - ~ Düğümler oyun durumlarını, kenarlar ise hamleleri temsil eder.

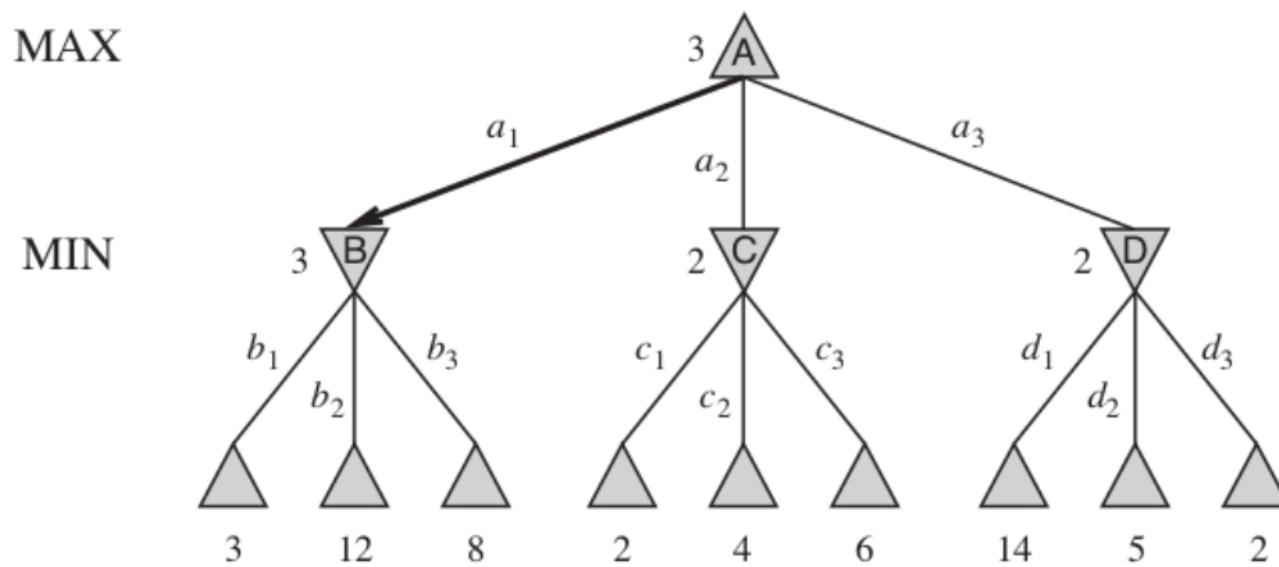


Oyunlar

- Oyunlarda optimal karar verme
 - ~ Normal arama problemlerinde hedef durumuna giden aksiyonlar listesine ulaşmaya çalışırız.
 - ~ Çekişmeli arama (İng: adversarial search) problemlerinde ise karşı tarafına hamleleri hesaplanmalıdır.
 - Bu amaçla MAX oyuncusu koşullu bir strateji geliştirmelidir.
 - ~ Örnek bir basit oyun için oluşturulan arama ağaçları bir sonraki slaytta gösterilmektedir.

Oyunlar

- Oyunlarda optimal karar verme



Oyunlar

- Oyunlarda optimal karar verme
 - ~ MAX oyuncusu MIN oyuncusunun optimal şekilde oynayacağını kabul ederek hareket eder.
 - ~ Bu durumda bir düğüme ait minimax değeri şu şekilde hesaplanır:
- $\text{MINIMAX}(s) = \text{YARAR}(s)$
 - ~ Eğer $\text{BITİŞ_TESTİ}(s)$ ise
 - ~ $\left\{ \begin{array}{ll} \max_{a \in \text{Aksiyonlar}(s)} \text{MINIMAX}(\text{SONUC}(s, a)) & \text{Eğer } \text{OYUNCU}(s) = \text{MAX} \text{ ise} \\ \min_{a \in \text{Aksiyonlar}(s)} \text{MINIMAX}(\text{SONUC}(s, a)) & \text{Eğer } \text{OYUNCU}(s) = \text{MIN} \text{ ise} \end{array} \right.$
 - ~ Bu yöntemi kullanarak kök düğümde minimax karara ulaşabiliriz.
 - Bu karar bizi en yüksek minimax değere sahip duruma götürecekтир.

Oyunlar

fonksiyon MINIMAX-KARAR(durum) **dönüş** bir aksiyon
return argmax_{a ∈ AKSIYONLAR(s)} MIN-DEĞER(SONUC(durum,a))

function MAX-DEĞER(durum) **dönüş** bir yarar değeri
if BITİŞ-TESTİ(durum) **then return** YARAR(durum)
v <= ∞
for each a **in** AKSİYONLAR(durum) **do**
 v <= MAX(v, MIN-DEĞER(SONUC(s,a)))
return v

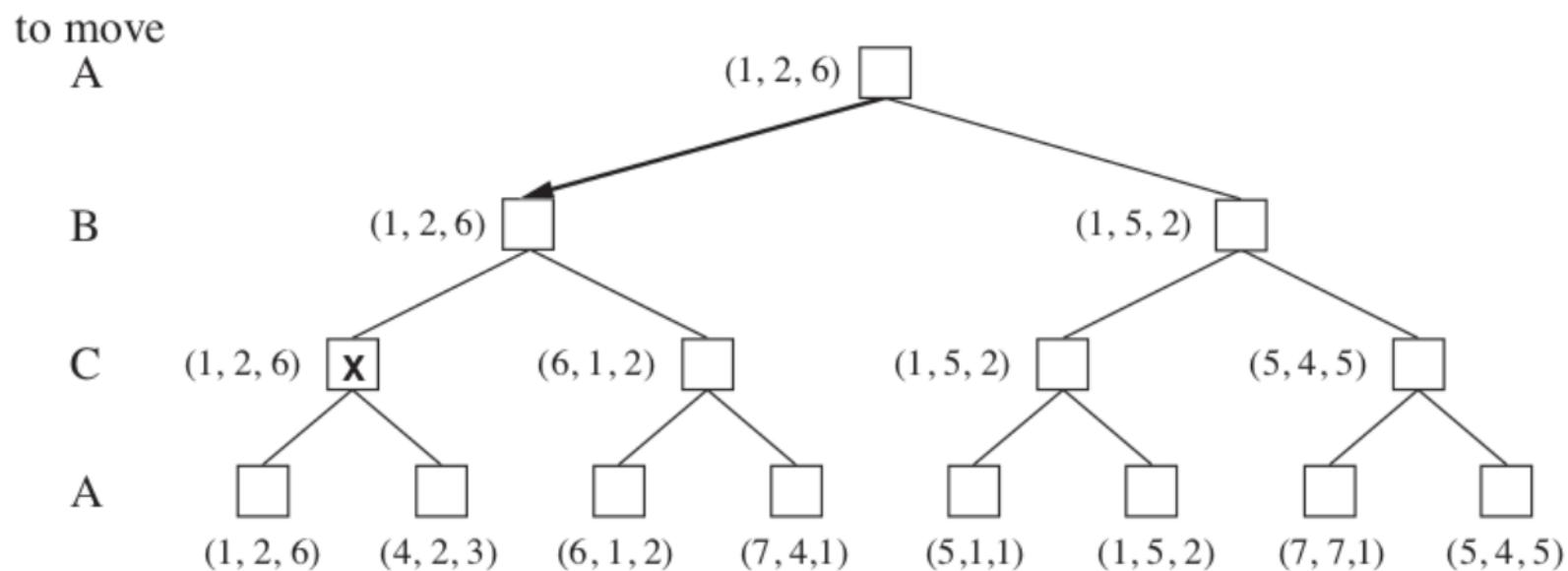
function MIN-DEĞER(durum) **dönüş** bir yarar değeri
if BITİŞ-TESTİ(durum) **then return** YARAR(durum)
v <= ∞
for each a **in** AKSİYONLAR(durum) **do**
 v <= MIN(v, MAX-DEĞER(SONUC(s,a)))
return v

Oyunlar

- Minimax algoritması
 - ~ Minimax algoritması bulunduğu durumdan minimax kararını hesaplar.
 - ~ Bunu yaparken yinelemeli olarak ardıl durumlardan yukarı doğru hesaplar.
 - Önce yapraklara iner, ordan tekrar yukarı çıkar.
 - ~ Minimax algoritması oyun ağacında derinlik-öncelikli keşif yapar.
 - Eğer ağacın maksimum derinliği m ve her düğümde b aksiyon mümkün ise, algoritmanın zaman karmaşıklığı $O(b^m)$.
 - Yer karmaşıklığı
 - ~ her aksiyonu birden üreten yöntem için $O(bm)$
 - ~ Her aksiyonu birer birer üreten algoritma için $O(m)$
 - Gerçek oyunlar için zaman karmaşıklığı oldukça yüksektir.

Oyunlar

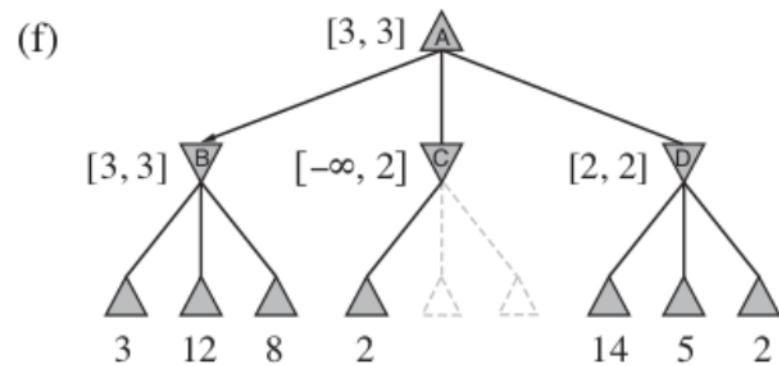
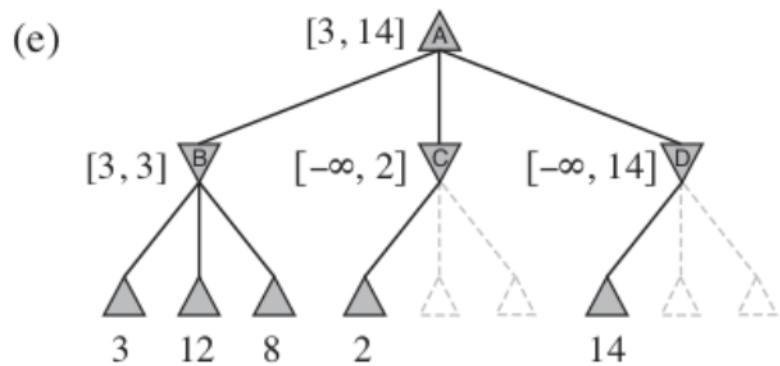
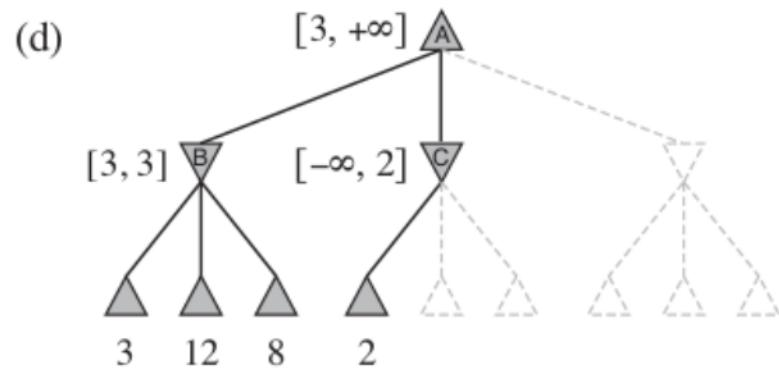
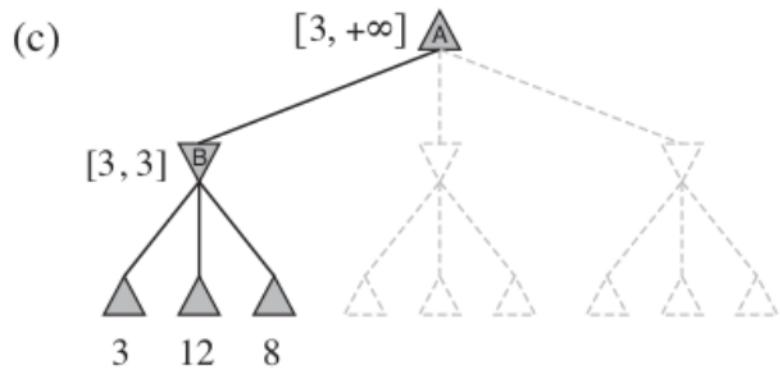
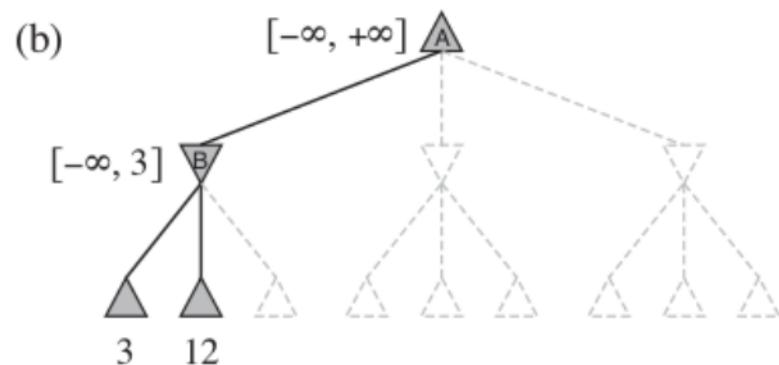
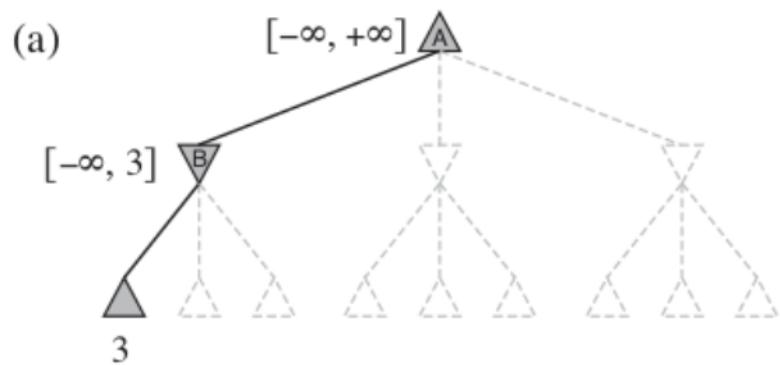
- Çok-oyunculu oyunlarda optimal kararlar
 - ~ Birçok popular oyun ikiden fazla oyuncu içerir.
 - ~ Minimax algoritması çok-oyunculu oyunlar için uyarlanabilir.



Oyunlar

- Alpha-Beta budama
 - ~ Minimax algoritmasının sorunu ağaçın derinliği arttıkça oyuna ait oluşan durumların üstel olarak artabilir.
 - ~ Üstel durumu tamamen ortadan kaldırıramasak da üstel durumu yarıya indirebiliriz.
 - ~ Doğru minimax kararına ulaşmak için her düğüme bakmak zorunda değiliz.
 - Bunun için budama yöntemini kullanacağız.
 - ~ Bu amaçla alpha-beta budama algoritmasını inceleyeceğiz.
 - ~ Önce gördüğümüz örneği düşünürsek
 - ~ MINIMAX (kök) = $\max(\min(3, 12, 8), \min(2, x, y), \min(14, 5, 2))$
 - = $\max(3, \min(2, x, y), 2)$
 - = $\max(3, z, 2)$ eğer $z = \min(2, x, y) \leq 2$
 - = 3.

Oyunlar

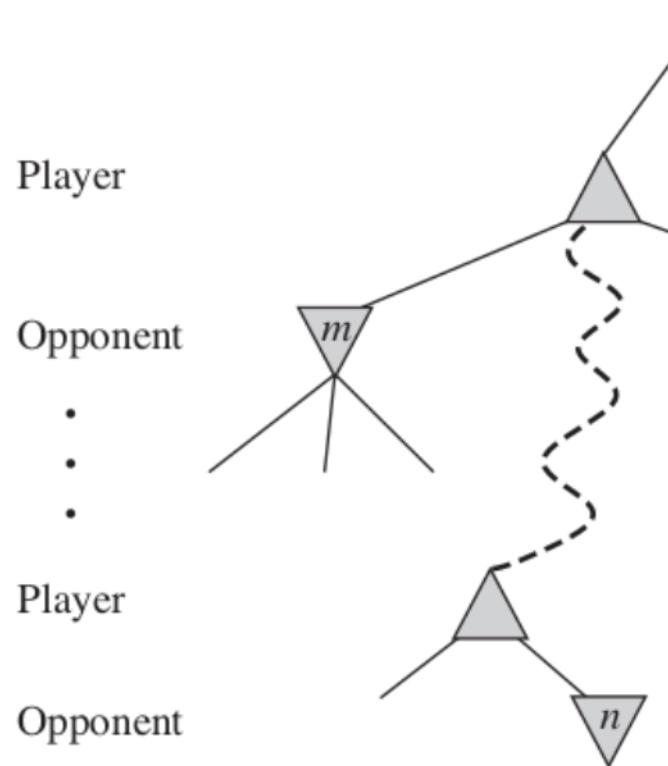


Oyunlar

- Alpha-beta budama
 - ~ Genel çalışma mantığı şudur.
 - Ağacın herhangi bir noktasında bir n düğümü düşünelim
 - Eğer bir oyuncu n düğümünün üst düğümünde veya ağacın daha yüksek bir noktasında daha iyi bir seçim olan m düğümü var ise
 - ~ n düğümüne oyun sırasında asla ulaşılmayacaktır.
 - Öyleyse, n düğümünün bu durumunu yeterli miktarda inceledikten sonra (alt düğümlerinin bazılarını inceledikten sonra) bu düğümü budayabiliriz.
 - Minimax algoritması hatırlayacağınız üzere derinlik-öncelikli olarak çalışmakta
 - ~ Öyleyse belli bir yol üzerinde düğümleri incelemeliyiz.
 - Alpha-beta budama ismini bir yol üzerinde yukarı hareket eden değerlerin üzerinde sınır belirleyen iki parametreden alır.
 - ~ α = MAX için incelediğimiz yol üzerinde herhangi bir seçim noktasına kadar bulunan en iyi (yüksek) değer.
 - ~ β = MIN için incelediğimiz yol üzerinde herhangi bir seçim noktasına kadar bulunan en iyi (düşük) değer

Oyunlar

- Alpha-beta budama
 - ~ Arama yaparken α ve β değerleri yenilenirken bunlara bağlı olarak diğer bazı parçalarda budanır.



Oyunlar

```
function ALPHA-BETA-ARAMA (state) dönüş bir aksiyon  
    v ← MAX-DEĞER (durum,  $-\infty$ ,  $+\infty$ )  
return aksiyon in AKSİYONLAR(durum) v değeri ile
```

```
function MAX-DEĞER(durum,  $\alpha$ ,  $\beta$ ) dönüş bir yarar değeri  
    if BİTİŞ-TESTİ (durum) then return YARAR(durum)  
    v ←  $-\infty$   
    for each  $a$  in AKSİYONLAR(durum) do  
        v ← MAX (v , MIN-DEĞER (SONUÇ(s, $a$ ),  $\alpha$ ,  $\beta$ ))  
        if  $v \geq \beta$  then return v  
         $\alpha$  ← MAX ( $\alpha$ , v )  
return v
```

```
function MIN-DEĞER (durum,  $\alpha$ ,  $\beta$ ) returns bir yarar değeri  
    if BİTİŞ-TESTİ (durum) then return YARAR(durum)  
    v ←  $+\infty$   
    for each  $a$  in AKSİYONLAR(durum) do  
        v ← MIN (v , MAX-DEĞER (SONUÇ(s, $a$ ) ,  $\alpha$ ,  $\beta$ ))  
        if  $v \leq \alpha$  then return v  
         $\beta$  ← MIN ( $\beta$ , v )  
return v
```

Oyunlar

- Alpha-beta budama
 - ~ Alpha-beta algoritmasının etkinliği fazlaıyla alt düğümlerinin hangi sırayla incelendiğine bağlıdır.
 - Örneğimize tekrar bakalım.
 - ~ Bu sebeple en iyi olarak seçilecek düğümleri önce incelemek yararlı olacaktır.
 - ~ Bu durum kısmen mümkün ise:
 - $O(b^m)$ yerine $O(b^{m/2})$ düğümün oluşturulması yeterli olacaktır.
 - Böylece dallanma faktörü b yerine \sqrt{b} olacaktır.
 - ~ Yani satranç özelinde düşünürsek 35 yerine 6.
 - Bu sayede alpha-beta budama minimax algoritmasının aynı sürede incelediğinin iki katı uzaklığı kadar gidebilecektir.
 - Düğümler rastgele seçilse $O(b^{3m/4})$ yaklaşık olarak oluşturulacaktır.
 - ~ Farklı hamle dizileri ile aynı durumlar oluşabilir.
 - ~ Değerlendirmeyi hızlandırmak için bu farklı durumları bir tabloda saklayabiliriz.

Oyunlar

- Mükemmel olmayan gerçek-zamanlı kararlar
 - ~ Minimax ve alpha-beta budama algoritmaları karar verebilmek için bitiş durumuna kadar ilerlemeliler.
 - Bu derinlik genellikle oldukça fazla olabilir.
 - ~ Bunun yerine başka bir yöntem geliştirilebilir.
 - Bitiş durumuna gitmeden bir kesme noktası seçmek
 - Bu seçme noktasında oluşan düğümleri bir değerlendirme fonksiyonuna göndermek
 - Böylece bu düğümlere bitiş noktası gibi davranmaktadır.
 - ~ Böylece daha önceki yarar testi yerine kesme testi uygulayacağız
- H-MINIMAX(s, d) =
 - ~ $\begin{cases} \text{DEĞER}(s) & \text{Eğer} \\ \text{KESME_TESTİ}(s, d) \text{ ise} \\ \max_{a \in \text{Aksiyonlar}(s)} \text{H-MINIMAX}(\text{SONUC}(s, a), d+1) & \text{Eğer OYUNCU}(s) = \text{MAX} \text{ ise} \\ \min_{a \in \text{Aksiyonlar}(s)} \text{H-MINIMAX}(\text{SONUC}(s, a), d+1) & \text{Eğer OYUNCU}(s) = \text{MIN} \text{ ise} \end{cases}$

Oyunlar

- Değerlendirme fonksiyonları (İng: Evaluation functions)
 - ~ Değerlendirme fonksiyonu oyunun belli bir pozisyonunun tahmini yararını döner.
 - Bu yönyle, hedefe ulaşmanın maliyetini tahmin eden bulușsal fonksiyonlara benzer.
 - ~ Birçok değerlendirme fonksiyonu durumun belli özelliklerini hesaplar.
 - Satranç açısından düşünürsek taşların sayısı önemli bir ölçü
 - ~ Eski tecrübelere dayanarak belli durumların kazanma ile bitme ihtimalini hesaplayabiliriz.
 - Örneğin iki piyon tek piyona karşı kalma durumlarında 72% ile kazanma, 8% ile beraberlik, 20% ile kaybetme ile sonlansın.
 - Beklenen değer şu şekilde hesaplanabilir:
 - ~ $(0.72 \times (+1)) + (0.20 \times 0) + (0.08 \times \frac{1}{2}) = 0.76.$
 - ~ Bu tür bir yaklaşım oldukça fazla tecrübe gerektirecektir.
 - ~ Bu sebeple genelde değerlendirme fonksiyonları her farklı özelliğe belli bir değer verebilir.

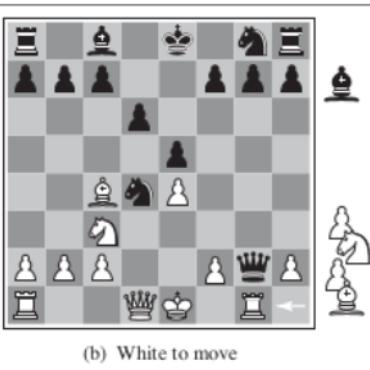
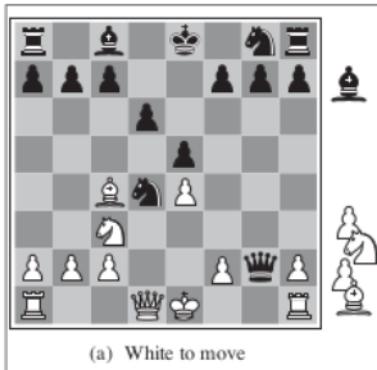
Oyunlar

- Değerlendirme fonksiyonları
 - ~ Satranç özelinde düşünürsek
 - Her piyon 1, at veya fil 3, kale 5, vezir 9 puan.
 - Ayrıca “iyi piyon dağılımı” ve “şah güvenli durumda” 1 puan.
 - Bu şekilde herhangi bir pozisyonu değer biçebiliriz.
 - ~ Bu tür değerlendirme fonksiyonlarına ağırlıklı doğrusal fonksiyon (İng: Weighted linear function) denir.
 - $DEĞER(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s) = \sum w_i f_i(s)$
 - f_i değerleri duruma ait özellik, w_i değerleri ise her bir özelliğin ağırlığını belirler.
 - ~ Doğrusal olarak hesaplama yaptığımızda özelliklerin birbirlerinden bağımsız olduğunu farzediyoruz.
 - Halbuki özellikler birbirini etkileyebilir.
 - Örneğin, oyun sonunda filin etkisi daha fazla olacaktır.
 - ~ Bu sebeple bu tür durumlarda doğrusal olmayan kombinasyonlar kullanılır.

Oyunlar

- Aramayı kesme

- ~ Diğer bir adım da alpha-beta aramayı değiştirmek, uygun olduğunda aramayı keserek DEĞER fonksiyonunu çağrılmaktır.
 - **if** KESME-TESTİ(durum, derinlik) **then return** DEĞER(durum)
 - ~ Sorulması gereken soru: derinlik değerini nasıl belirleyeceğiz?
 - ~ İlk yapılabilecek sabit bir derinlik seçmek
 - Bu derinlik belli bir d derinliğinden fazla seçilir.
 - Ancak bu yöntemin sakıncalı yönleri vardır. Bazen belli bir noktada durmak yerine arama biraz daha devam etmelidir.



(a) White to move

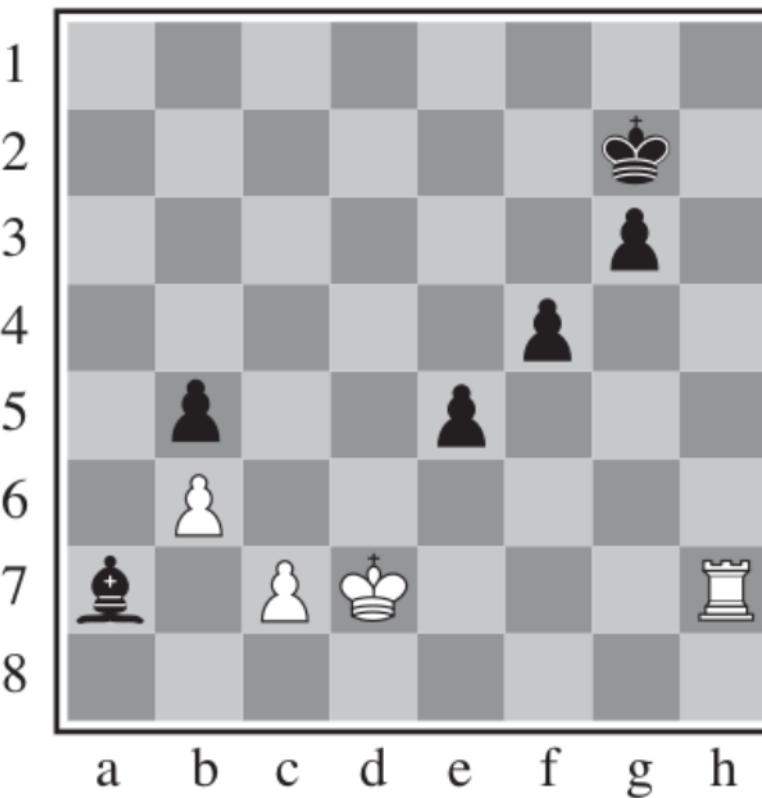
(b) White to move

Oyunlar

- Kesme testi
 - ~ Tabii ki daha zekice bir taktik ile kesme noktası belirlenmelidir.
 - ~ Öncelikle değerlendirme fonksiyonu durağan (İng: quiescent) durumlara uygulanmalıdır.
 - Bu durumlar yakın gelecekte hızlı değişimlere uğramayacaktır.
 - Durağan olmayan durumlar bir miktar daha açılabilir.
 - Bu tip aramaya durağanlık araması adı verilir.
 - ~ Diğer düşünülmlesi gereken bir konu da ufuk etkisi (İng: horizon effect)
 - Ufuk çizgisi yakınsa buraya kadar yapılan geciktirici hamleler ile yanlış bir değerlendirme yapma ihtimali mevcut.
 - Örneği inceleyelim.

Oyunlar

- Kesme testi



Oyunlar

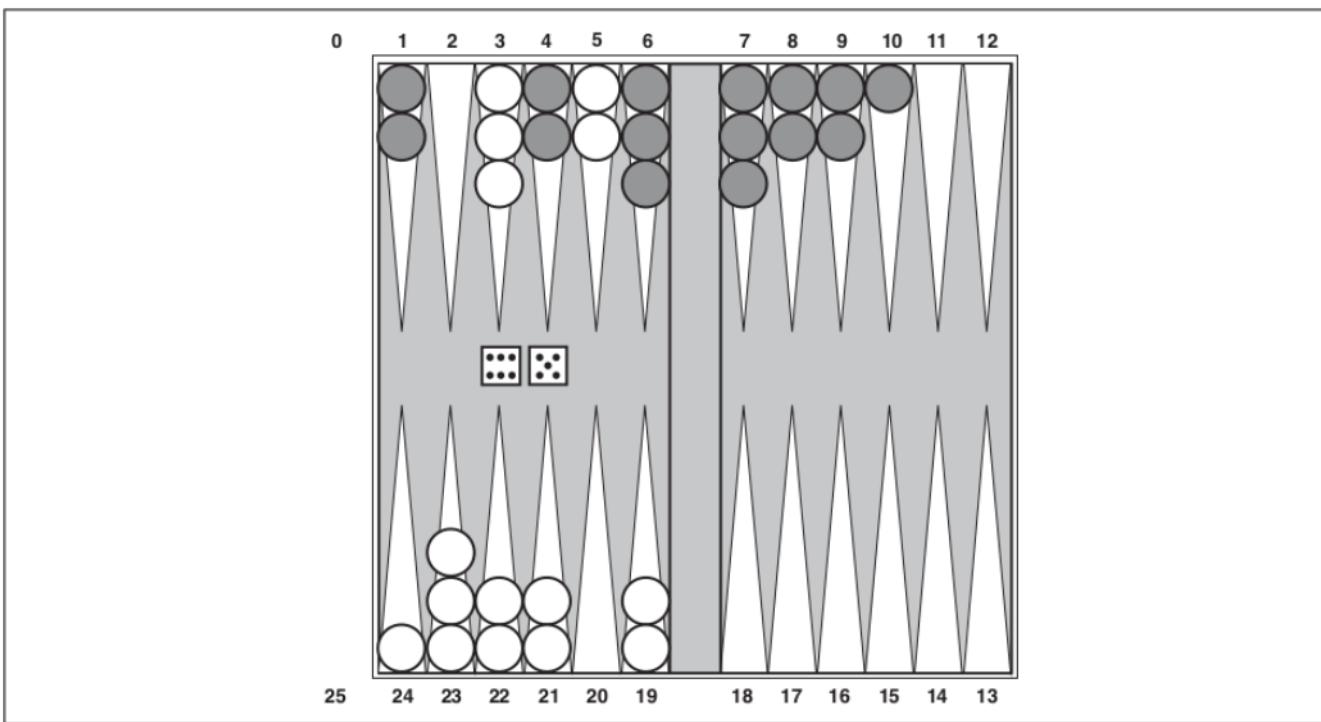
- Kesme testi
 - ~ Ufuk etkisi
 - Ufuk etkisinden kurtulmayı sağlamak için kullanılan yöntemlerden biri tekil genişletme (İng: Singular extension).
 - ~ Diğer hamlelerden bariz olarak iyi bir hamle keşfedilirse bu hamle hatırlanıyor.
 - ~ Arama derinlik limitine ulaştığında, az önce hatırlanan hamle yapılabiliyorsa, arama bir süre daha devam ettiriliyor.
 - ~ Ağaç bir miktar derinleşir ancak bu çok sık olmadığı için ağaçın büyüğünü fazla büyütmez.
 - ~ İleri budama (İng: Forward pruning)
 - Bir diğer hızlandırma metodu ileri budamadır.
 - ~ Belli bir düğümde oluşan düğümler incelenmeden direk budanır.
 - ~ Ancak gerekli düğümleri budama ihtimaliniz var.
 - ~ ProbCut algoritması önceki tecrübelere dayanarak olasılıksal olarak belli düğümlerde ileri budama yapmayı dener.

Oyunlar

- Arama vs Tablodan bakma
 - ~ Belli durumları ve bu durumlarda yapılacak en doğru hamleleri bir tabloda saklayarak karar vermeyi hızlandırabiliriz.
 - Örneğin, satranç için farklı açılış ve oyun sonları kaydedilebilir.
 - ~ Bu durumlar oluştuğunda, arama ağıacı oluşturmak yerine tek yapılması gereken ilgili hamleleri tablodan okumaktır.
 - ~ Satranç için oyun açılışı ve oyun sonlarını tablodan okuyan algoritmalar geliştirilmiştir.

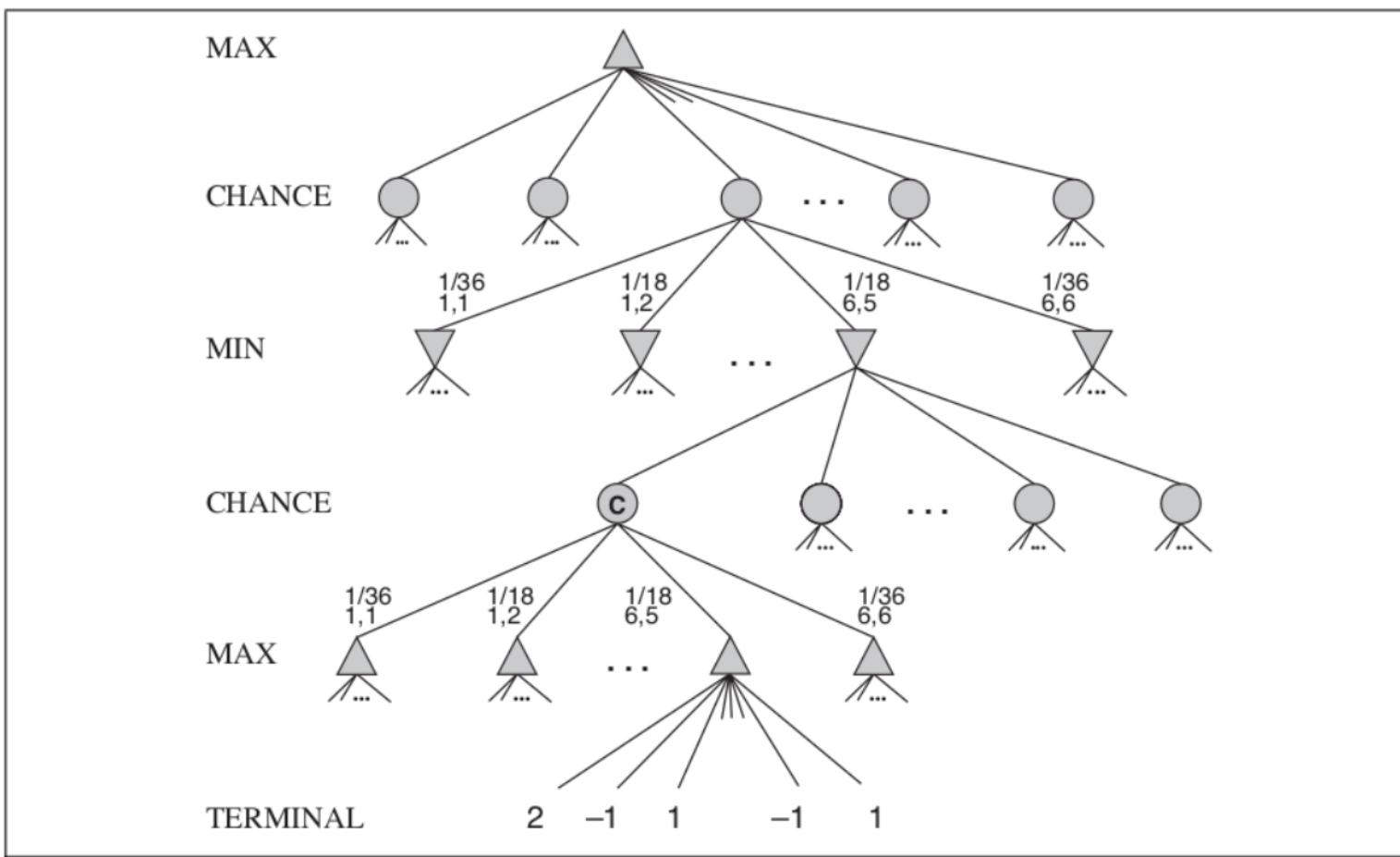
Oyunlar

- Stokastik oyunlar
 - ~ Bu tür oyunlar rastgele belirlenen bir tahmin edilemeyen eleman içerir.
 - Örneğin zar atılması
 - Tavla, bu oyunlara örnektir.



Oyunlar

- Stokastik oyunlar
 - ~ Tavla için oluşturulan oyun ağaçları şans düğümleri içerir.



Oyunlar

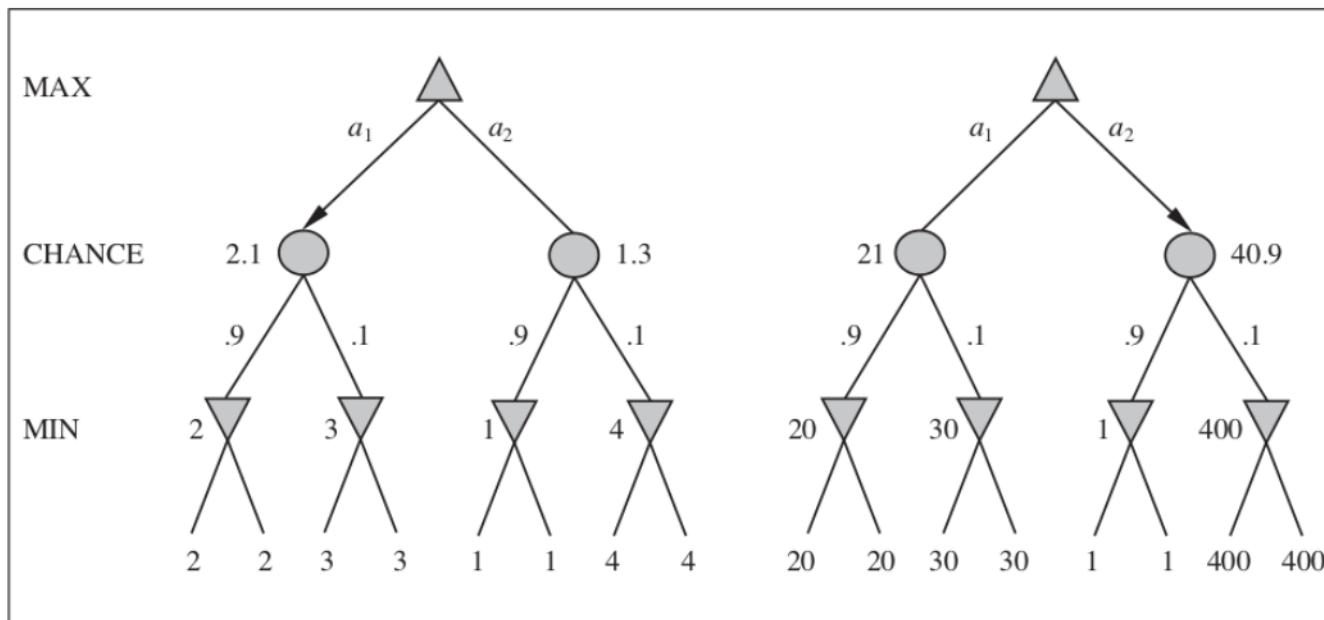
- Stokastik oyunlar
 - ~ Minimax değeri hesabımızın yerini, şans düğümlerini de hesaba katacağımız beklenen-minimax hesabı alır.
- BEKLENEN-MINIMAX(s) =

$$\left\{ \begin{array}{ll} \text{YARAR}(s) & \text{Eğer BİTİŞ_TESTİ}(s) \text{ ise} \\ \max_a \text{BEKLENEN-MINIMAX}(\text{SONUC}(s,a)) & \text{Eğer OYUNCU}(s) = \text{MAX} \text{ ise} \\ \min_a \text{-BEKLENEN-MINIMAX}(\text{SONUC}(s,a)) & \text{Eğer OYUNCU}(s) = \text{MIN} \text{ ise} \\ \sum_r P(r) \text{BEKLENEN - MINIMAX}(\text{SONUC}(s,a)) & \text{Eğer OYUNCU}(s) = \text{ŞANS} \text{ ise} \end{array} \right.$$

- ~ Formülde r olası zar, SONUC(s,r) ise s durumunda r zarının atılmasını temsil etmektedir.

Oyunlar

- Stokastik oyunlar
 - ~ Şans oyunları için değerlendirme fonksiyonları
 - Minimax'da olduğu gibi Beklenen-minimax için de arama bir noktada kesilmelidir.
 - Ancak belirlenen yarar değerleri belli bir pozisyondan galip gelme ihtimalinin pozitif doğrusal transformasyonu ile belirlenmelidir.



Oyunlar

- Stokastik oyunlar
 - ~ Zaman karmaşıklığı
 - Farklı zarları da hesaba kattığımız için gereken zaman $O(b^m n^m)$
 - Tavla için düşünürsek, $n = 21$, b yaklaşık olarak 20 (ancak çift attığımızda 4000'e kadar çıkabilir).
 - ~ Sadece üç hamle sonrasında gidebiliriz gibi görünüyor.
 - Budama, olası ihtimaller üzerinde çalışıyor, ancak zar atımını hesaplamadan bu ihtimaller ortaya çıkmıyor.
 - ~ Bu sebeple olasılık sayısı hızlı artıyor.
 - Çözüm olarak değerlendirme fonksiyonunun vereceği değere sınır koyabiliriz.
 - ~ Bir miktar budama yapmamız mümkün olabilir.
 - Bir diğer alternatif Monte Carlo simülasyonu kullanmak.
 - ~ Bulunan durumdan algoritma farklı rastgele zarlarla binlerce oyun oynayarak en yüksek yararı veren hamle kombinasyonlarını seçebilir.

