Bölüm 13: Öğrenme ve Adaptasyon

Algoritmik Oyun Kuramı

En iyi tepki dinamiği · Kurgusal oyun · Pişmanlık · Hedge · Korelasyonlu denge · ε-açgözlü ve UCB · Pekiştirmeli öğrenme · Replikatör · Tit-for-Tat

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

© Sercan Külcü. Tüm hakları saklıdır. · Sunumun etkileşimli HTML sürümü: https://sercankulcu.github.io/files/agt/slides/Bolum_13_Ogrenme_ve_Adaptasyon.html

GİRİŞ

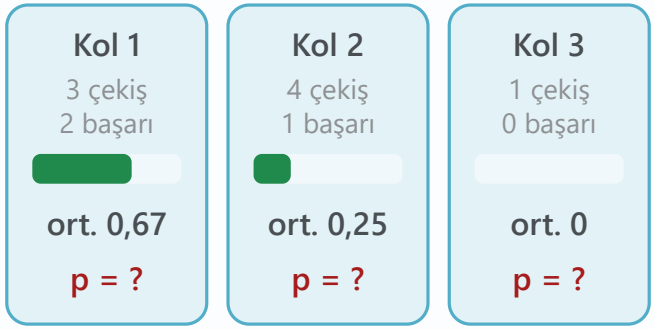
Oyunla Başlangıç: Çok Kollu Haydut

Oyunun kuralı

Her kol gizli bir olasılıkla 1 puan verir; bir kolu çekince yalnızca onun sonucunu görürsünüz ve amaç toplam ödülü en büyük yapmaktır.

Merak sorusu

- Şimdilik ortalaması en yüksek olan Kol 1'e mi bağlanırdınız, yoksa Kol 3'ü yeniden mi denerdiniz?
- Yüz çekişin kaçını denemeye (keşfe) ayırırdınız?



Gerçek olasılıklar gizli; her yeni deneyde yeniden çekilir. Kol 3 yalnızca bir kez denendi.

Çok Kollu Haydut oyunu →

GİRİŞ

Öğrenme Çıktıları

Dinamikler

En iyi tepki dinamiği ve kurgusal oyunu adım adım çalıştırmak; ne zaman dengeye yakınsadıklarını açıklamak.

Pişmanlık

Dışsal pişmanlığı hesaplamak; çarpımsal ağırlıklar (Hedge) algoritmasının pişmanlıksız olduğunu yorumlamak.

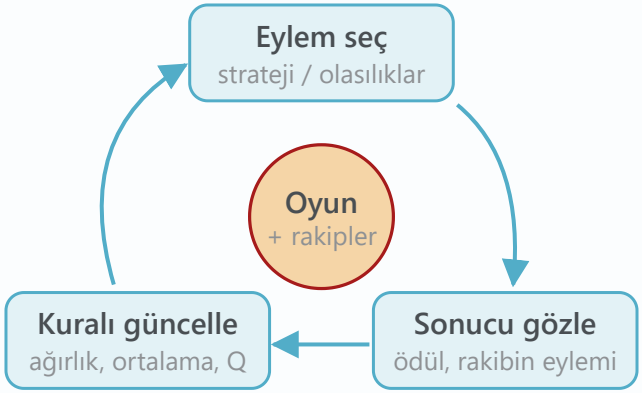
Keşif–sömürü

Çok kollu haydut probleminde ε-açgözlü ve UCB algoritmalarını izlemek ve karşılaştırmak.

Evrim ve tekrar

Replikatör dinamiğini ve tekrarlanan Mahkûmun İkilemi'nde Tit-for-Tat'ın başarısını açıklamak.

Neden Öğrenme ve Adaptasyon?



Öğrenme: geçmiş deneyimden strateji geliştirmek. **Adaptasyon**: rakip ve çevre değişince stratejiyi güncellemek.

- **Denge hesaplamak zor olabilir**: genel oyunlarda Nash dengesi PPAD-tam (büyük oyunlarda hızlı bir çözüm yolu bilinmeyen bir problem sınıfı). Oyuncular dengeyi nasıl "bulur"?
- **Oyunu bilmiyor olabilirler**: kazanç matrisi bilinmez; yalnızca oynayıp sonucu görürler.
- Soru: basit, yerel güncelleme kuralları uzun vadede **dengeye** ya da en azından **iyi bir performansa** götürür mü?

Örnekler: satranç ve poker yapay zekâları kendine karşı oynayarak öğrenir; reklam açık artırmalarında teklif algoritmaları her gün uyum sağlar; navigasyon uygulamaları trafiğe göre rota değiştirir.

GİRİŞ

Öğrenme Modelleri: Genel Bakış

Model	Oyuncu neyi gözler?	Nasıl günceller?
En iyi tepki dinamiği	Rakibin son eylemi	Ona en iyi tepkiyi verir
Kurgusal oyun (fictitious play)	Rakibin tüm geçmişi	Frekanslara en iyi tepki
Pişmanlığı en aza indirme (Hedge)	Her eylemin kazancı	İyi eylemlerin ağırlığını çarpımsal artırır
Çok kollu haydut (ε-açgözlü, UCB)	Yalnızca seçtiği eylemin ödülü	Ortalama tahmini + keşif
Pekiştirmeli öğrenme (Q-öğrenme)	Ödül ve yeni durum	Q(s, a) değerlerini günceller
Replikatör dinamiği	Nüfustaki ortalama başarı	Başarılı strateji yayılır

Yukarıdan aşağı: oyuncu rakibi değil kendi ödülünü izlemeye başlar; son satırda bakış bireyden nüfusa kayar.

En İyi Tepki Dinamiği (Adım Adım)

Satır oyuncusu \ Sütun oyuncusu	X	Y	Z
A	(4, 3)	(0, 2)	(1, 1)
B	(2, 1)	(3, 3)	(0, 2)
C	(1, 4)	(2, 0)	(3, 2)

Başlangıç profili **(B, Z)**. Oyuncular sırayla, rakibin *şu anki* eylemine en iyi tepki verir.

Adım 1 / 5

(C, Y)'den başlansaydı (B, Y) dengesine gidilirdi: varılan denge başlangıca bağlı.

En İyi Tepki Dinamiği (Adım Adım)

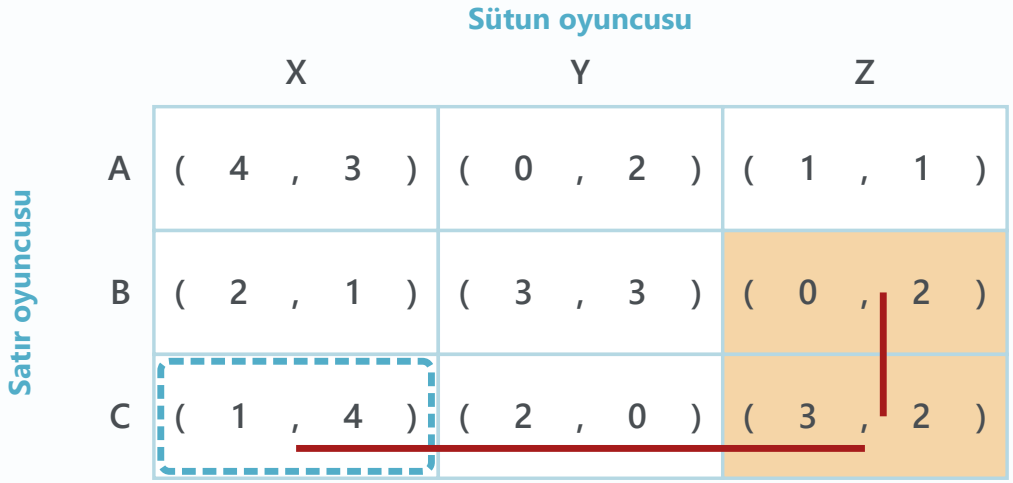


Satır oyuncusu Z'ye karşı en iyi tepkiyi arar: $u_1 = 1, 0, 3 \rightarrow$ **C**. Profil (C, Z).

Adım 2 / 5

(C, Y)'den başlansaydı (B, Y) dengesine gidilirdi: varılan denge başlangıca bağlı.

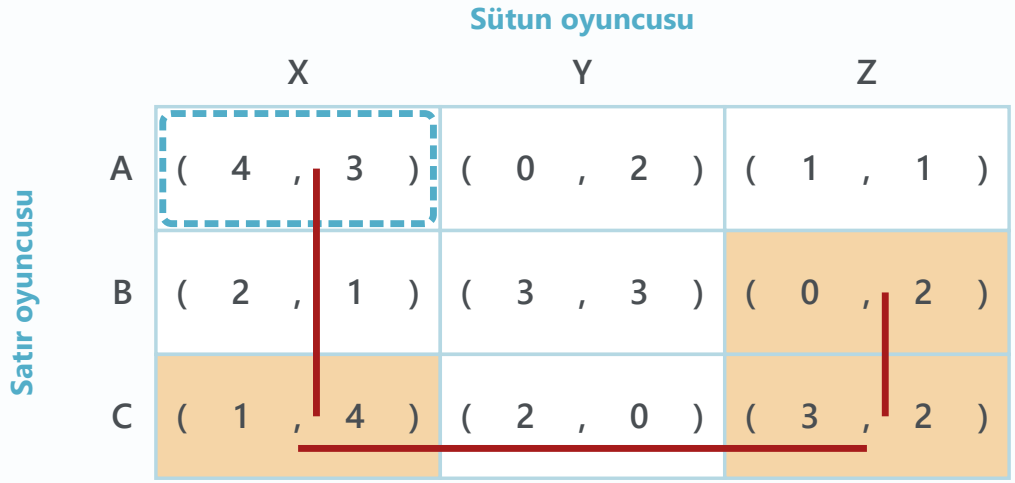
En İyi Tepki Dinamiği (Adım Adım)



Sütun oyuncusu C'ye karşı: $u_2 = 4, 0, 2 \rightarrow$ **X**. Profil (C, X).

(C, Y)'den başlansaydı (B, Y) dengesine gidilirdi: varılan denge başlangıca bağlı.

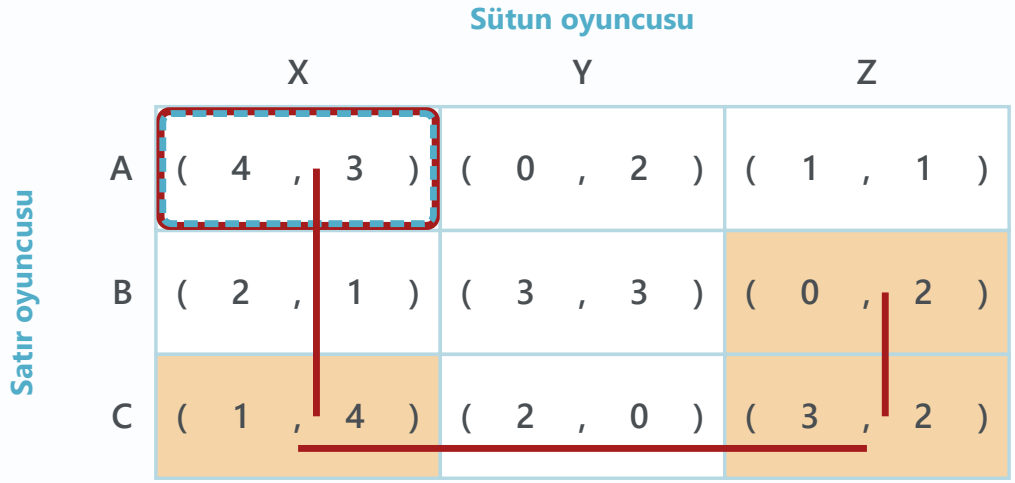
En İyi Tepki Dinamiği (Adım Adım)



Satır oyuncusu X'e karşı: $u_1 = 4, 2, 1 \rightarrow$ **A**. Profil (A, X).

(C, Y)'den başlansaydı (B, Y) dengesine gidilirdi: varılan denge başlangıca bağlı.

En İyi Tepki Dinamiği (Adım Adım)

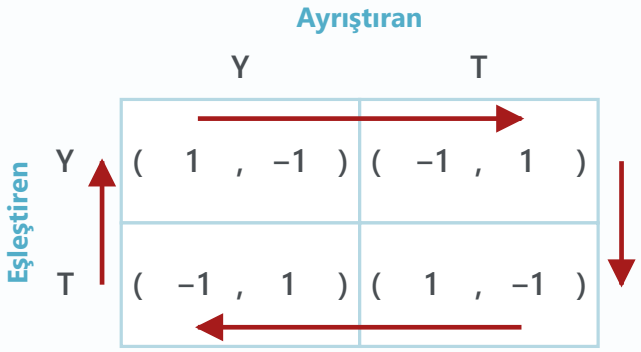


Sütun oyuncusu A'ya karşı zaten X'i seçiyor (3 > 2, 1); satır da A'dan ayrılmaz → dinamik **(A, X) Nash dengesinde** durur.

Adım 5 / 5

(C, Y)'den başlansaydı (B, Y) dengesine gidilirdi: varılan denge başlangıca bağlı.

En İyi Tepki Dinamiği Ne Zaman Yakınsar?



Yazı-Tura Eşleştirme: (Y,Y) → (Y,T) → (T,T) → (T,Y) → (Y,Y) … sonsuz döngü

Potansiyel oyunlar: yakınsar

Her kârlı sapma ortak bir potansiyel fonksiyonu artırır; sonlu oyunda sonsuza kadar artamaz (Monderer–Shapley 1996; tıkanıklık oyunları).

Saf dengesi olmayan oyunlar: döner

Yazı-Tura'da her adım bir oyuncuyu mutlu, diğerini mutsuz eder.

Çare: geçmişi hatırla

Yalnızca son eyleme değil, tüm geçmişe tepki ver → kurgusal oyun.

Kurgusal Oyun: Yazı-Tura Eşleştirme

t	Ayr. geçmişi Y/T (önceki turlar)	Eşleştiren	Eşl. geçmişi Y/T (önceki turlar)	Ayrıştıran	Eşl. Y oranı (bu tur dahil)	Ayr. Y oranı (bu tur dahil)
1	0 / 0	**Y**	0 / 0	**Y**	1,00	1,00
2						
3						
4						
5						
6						
7						
8						
9						
10						

1. tur keyfî: ikisi de **Y** oynar (henüz geçmiş yok).

Kurgusal Oyun: Yazı-Tura Eşleştirme

t	Ayr. geçmişi Y/T (önceki turlar)	Eşleştiren	Eşl. geçmişi Y/T (önceki turlar)	Ayrıştıran	Eşl. Y oranı (bu tur dahil)	Ayr. Y oranı (bu tur dahil)
1	0 / 0	**Y**	0 / 0	**Y**	1,00	1,00
2	1 / 0	**Y**	1 / 0	**T**	1,00	0,50
3						
4						
5						
6						
7						
8						
9						
10						

Eşleştiren rakip geçmişinde 1 / 0 görür → **Y** (çoğunluğu eşler, eşitlikte Y). Ayrıştıran 1 / 0 görür → **T** (çoğunluğun tersi, eşitlikte Y).

Kurgusal Oyun: Yazı-Tura Eşleştirme

t	Ayr. geçmişi Y/T (önceki turlar)	Eşleştiren	Eşl. geçmişi Y/T (önceki turlar)	Ayrıştıran	Eşl. Y oranı (bu tur dahil)	Ayr. Y oranı (bu tur dahil)
1	0 / 0	**Y**	0 / 0	**Y**	1,00	1,00
2	1 / 0	**Y**	1 / 0	**T**	1,00	0,50
3	1 / 1	**Y**	2 / 0	**T**	1,00	0,33
4						
5						
6						
7						
8						
9						
10						

Eşleştiren rakip geçmişinde 1 / 1 görür → **Y** (çoğunluğu eşler, eşitlikte Y). Ayrıştıran 2 / 0 görür → **T** (çoğunluğun tersi, eşitlikte Y).

Kurgusal Oyun: Yazı-Tura Eşleştirme

t	Ayr. geçmişi Y/T (önceki turlar)	Eşleştiren	Eşl. geçmişi Y/T (önceki turlar)	Ayrıştıran	Eşl. Y oranı (bu tur dahil)	Ayr. Y oranı (bu tur dahil)
1	0 / 0	**Y**	0 / 0	**Y**	1,00	1,00
2	1 / 0	**Y**	1 / 0	**T**	1,00	0,50
3	1 / 1	**Y**	2 / 0	**T**	1,00	0,33
4	1 / 2	**T**	3 / 0	**T**	0,75	0,25
5						
6						
7						
8						
9						
10						

Eşleştiren rakip geçmişinde 1 / 2 görür → **T** (çoğunluğu eşler, eşitlikte Y). Ayrıştıran 3 / 0 görür → **T** (çoğunluğun tersi, eşitlikte Y).

Kurgusal Oyun: Yazı-Tura Eşleştirme

t	Ayr. geçmişi Y/T (önceki turlar)	Eşleştiren	Eşl. geçmişi Y/T (önceki turlar)	Ayrıştıran	Eşl. Y oranı (bu tur dahil)	Ayr. Y oranı (bu tur dahil)
1	0 / 0	**Y**	0 / 0	**Y**	1,00	1,00
2	1 / 0	**Y**	1 / 0	**T**	1,00	0,50
3	1 / 1	**Y**	2 / 0	**T**	1,00	0,33
4	1 / 2	**T**	3 / 0	**T**	0,75	0,25
5	1 / 3	**T**	3 / 1	**T**	0,60	0,20
6						
7						
8						
9						
10						

Eşleştiren rakip geçmişinde 1 / 3 görür → **T** (çoğunluğu eşler, eşitlikte Y). Ayrıştıran 3 / 1 görür → **T** (çoğunluğun tersi, eşitlikte Y).

Kurgusal Oyun: Yazı-Tura Eşleştirme

t	Ayr. geçmişi Y/T (önceki turlar)	Eşleştiren	Eşl. geçmişi Y/T (önceki turlar)	Ayrıştıran	Eşl. Y oranı (bu tur dahil)	Ayr. Y oranı (bu tur dahil)
1	0 / 0	**Y**	0 / 0	**Y**	1,00	1,00
2	1 / 0	**Y**	1 / 0	**T**	1,00	0,50
3	1 / 1	**Y**	2 / 0	**T**	1,00	0,33
4	1 / 2	**T**	3 / 0	**T**	0,75	0,25
5	1 / 3	**T**	3 / 1	**T**	0,60	0,20
6	1 / 4	**T**	3 / 2	**T**	0,50	0,17
7						
8						
9						
10						

Eşleştiren rakip geçmişinde 1 / 4 görür → **T** (çoğunluğu eşler, eşitlikte Y). Ayrıştıran 3 / 2 görür → **T** (çoğunluğun tersi, eşitlikte Y).

Kurgusal Oyun: Yazı-Tura Eşleştirme

t	Ayr. geçmişi Y/T (önceki turlar)	Eşleştiren	Eşl. geçmişi Y/T (önceki turlar)	Ayrıştıran	Eşl. Y oranı (bu tur dahil)	Ayr. Y oranı (bu tur dahil)
1	0 / 0	**Y**	0 / 0	**Y**	1,00	1,00
2	1 / 0	**Y**	1 / 0	**T**	1,00	0,50
3	1 / 1	**Y**	2 / 0	**T**	1,00	0,33
4	1 / 2	**T**	3 / 0	**T**	0,75	0,25
5	1 / 3	**T**	3 / 1	**T**	0,60	0,20
6	1 / 4	**T**	3 / 2	**T**	0,50	0,17
7	1 / 5	**T**	3 / 3	**Y**	0,43	0,29
8						
9						
10						

Eşleştiren rakip geçmişinde 1 / 5 görür → **T** (çoğunluğu eşler, eşitlikte Y). Ayrıştıran 3 / 3 görür → **Y** (çoğunluğun tersi, eşitlikte Y).

Kurgusal Oyun: Yazı-Tura Eşleştirme

t	Ayr. geçmişi Y/T (önceki turlar)	Eşleştiren	Eşl. geçmişi Y/T (önceki turlar)	Ayrıştıran	Eşl. Y oranı (bu tur dahil)	Ayr. Y oranı (bu tur dahil)
1	0 / 0	**Y**	0 / 0	**Y**	1,00	1,00
2	1 / 0	**Y**	1 / 0	**T**	1,00	0,50
3	1 / 1	**Y**	2 / 0	**T**	1,00	0,33
4	1 / 2	**T**	3 / 0	**T**	0,75	0,25
5	1 / 3	**T**	3 / 1	**T**	0,60	0,20
6	1 / 4	**T**	3 / 2	**T**	0,50	0,17
7	1 / 5	**T**	3 / 3	**Y**	0,43	0,29
8	2 / 5	**T**	3 / 4	**Y**	0,38	0,38
9						
10						

Eşleştiren rakip geçmişinde 2 / 5 görür → **T** (çoğunluğu eşler, eşitlikte Y). Ayrıştıran 3 / 4 görür → **Y** (çoğunluğun tersi, eşitlikte Y).

Kurgusal Oyun: Yazı-Tura Eşleştirme

t	Ayr. geçmişi Y/T (önceki turlar)	Eşleştiren	Eşl. geçmişi Y/T (önceki turlar)	Ayrıştıran	Eşl. Y oranı (bu tur dahil)	Ayr. Y oranı (bu tur dahil)
1	0 / 0	**Y**	0 / 0	**Y**	1,00	1,00
2	1 / 0	**Y**	1 / 0	**T**	1,00	0,50
3	1 / 1	**Y**	2 / 0	**T**	1,00	0,33
4	1 / 2	**T**	3 / 0	**T**	0,75	0,25
5	1 / 3	**T**	3 / 1	**T**	0,60	0,20
6	1 / 4	**T**	3 / 2	**T**	0,50	0,17
7	1 / 5	**T**	3 / 3	**Y**	0,43	0,29
8	2 / 5	**T**	3 / 4	**Y**	0,38	0,38
9	3 / 5	**T**	3 / 5	**Y**	0,33	0,44
10						

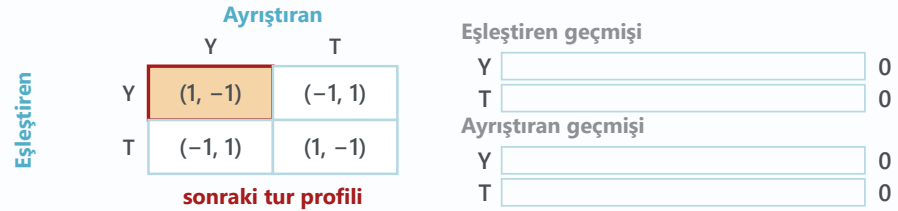
Eşleştiren rakip geçmişinde 3 / 5 görür → **T** (çoğunluğu eşler, eşitlikte Y). Ayrıştıran 3 / 5 görür → **Y** (çoğunluğun tersi, eşitlikte Y).

Kurgusal Oyun: Yazı-Tura Eşleştirme

t	Ayr. geçmişi Y/T (önceki turlar)	Eşleştiren	Eşl. geçmişi Y/T (önceki turlar)	Ayrıştıran	Eşl. Y oranı (bu tur dahil)	Ayr. Y oranı (bu tur dahil)
1	0 / 0	**Y**	0 / 0	**Y**	1,00	1,00
2	1 / 0	**Y**	1 / 0	**T**	1,00	0,50
3	1 / 1	**Y**	2 / 0	**T**	1,00	0,33
4	1 / 2	**T**	3 / 0	**T**	0,75	0,25
5	1 / 3	**T**	3 / 1	**T**	0,60	0,20
6	1 / 4	**T**	3 / 2	**T**	0,50	0,17
7	1 / 5	**T**	3 / 3	**Y**	0,43	0,29
8	2 / 5	**T**	3 / 4	**Y**	0,38	0,38
9	3 / 5	**T**	3 / 5	**Y**	0,33	0,44
10	4 / 5	**T**	3 / 6	**Y**	0,30	0,50

Eşleştiren rakip geçmişinde 4 / 5 görür → **T** (çoğunluğu eşler, eşitlikte Y). Ayrıştıran 3 / 6 görür → **Y** (çoğunluğun tersi, eşitlikte Y).

Kod İzleme: Kurgusal Oyun (Yazı-Tura, 6 tur)



Çağrı `fictitious(6)`. Sayaçlar sıfır; geçmiş olmadığı için 1. turda ikisi de **Y** oynar.

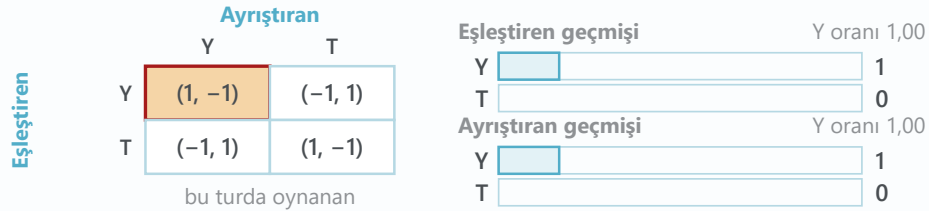
Adım 1 / 13

```
1  def fictitious(T):
▶       ne, na = [0, 0], [0, 0]   # Y/T sayaçları
▶       a, b = 0, 0               # 1. tur: ikisi de Y
4       for t in range(1, T + 1):
5           ne[a] += 1            # eşleştirenin geçmişi
6           na[b] += 1            # ayrıştıranın geçmişi
7           # eşleştiren: ayrıştıranın çoğunluğunu eşle
8           a = 0 if na[0] >= na[1] else 1
9           # ayrıştıran: eşleştirenin çoğunluğundan ayrış
10          b = 1 if ne[0] > ne[1] else 0
11      return ne, na
```

DEĞİŞKENLER

`T = 6` `ne = [0, 0]` `na = [0, 0]` `a = 0 (Y)` `b = 0 (Y)`

Kod İzleme: Kurgusal Oyun (Yazı-Tura, 6 tur)



Tur 1: (Y, Y) oynanır, eşleştiren kazanır. Sayaçlar: eşleştiren Y/T = 1/0, ayrıştıran Y/T = 1/0.

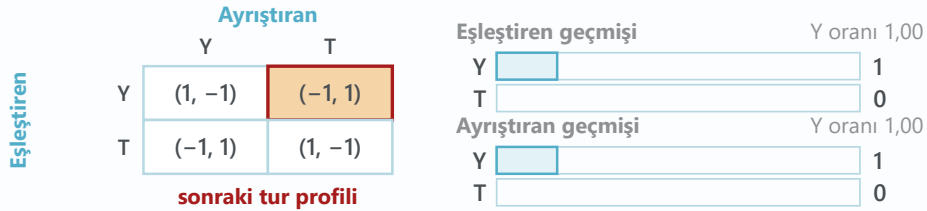
Adım 2 / 13

```
1  def fictitious(T):
2      ne, na = [0, 0], [0, 0]   # Y/T sayaçları
3      a, b = 0, 0               # 1. tur: ikisi de Y
▶      for t in range(1, T + 1):
▶          ne[a] += 1            # eşleştirenin geçmişi
▶          na[b] += 1            # ayrıştıranın geçmişi
7          # eşleştiren: ayrıştıranın çoğunluğunu eşle
8          a = 0 if na[0] >= na[1] else 1
9          # ayrıştıran: eşleştirenin çoğunluğundan ayrış
10         b = 1 if ne[0] > ne[1] else 0
11     return ne, na
```

DEĞİŞKENLER

`t = 1` `a = 0 (Y)` `b = 0 (Y)` `ne = [1, 0]` `na = [1, 0]`

Kod İzleme: Kurgusal Oyun (Yazı-Tura, 6 tur)



Sonraki tur: ayrıştıran çoğunlukla Y oynadı → eşleştiren **Y**; eşleştiren çoğunlukla Y oynadı → ayrıştıran **T**.

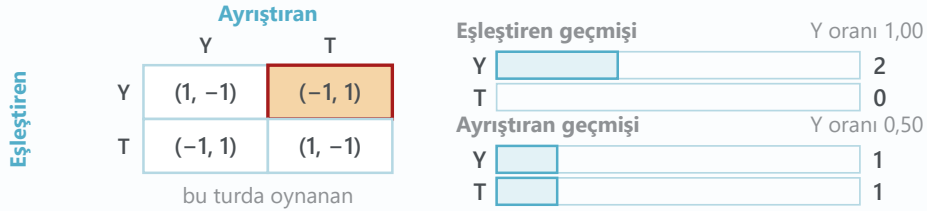
Adım 3 / 13

```
1  def fictitious(T):
2      ne, na = [0, 0], [0, 0]   # Y/T sayaçları
3      a, b = 0, 0               # 1. tur: ikisi de Y
4      for t in range(1, T + 1):
5          ne[a] += 1            # eşleştirenin geçmişi
6          na[b] += 1            # ayrıştıranın geçmişi
7          # eşleştiren: ayrıştıranın çoğunluğunu eşle
▶          a = 0 if na[0] >= na[1] else 1
9          # ayrıştıran: eşleştirenin çoğunluğundan ayrış
▶          b = 1 if ne[0] > ne[1] else 0
11     return ne, na
```

DEĞİŞKENLER

`t = 1` `a = 0 (Y)` `b = 1 (T)` `ne = [1, 0]` `na = [1, 0]`

Kod İzleme: Kurgusal Oyun (Yazı-Tura, 6 tur)



Tur 2: (Y, T) oynanır, ayrıştıran kazanır. Sayaçlar: eşleştiren Y/T = 2/0, ayrıştıran Y/T = 1/1.

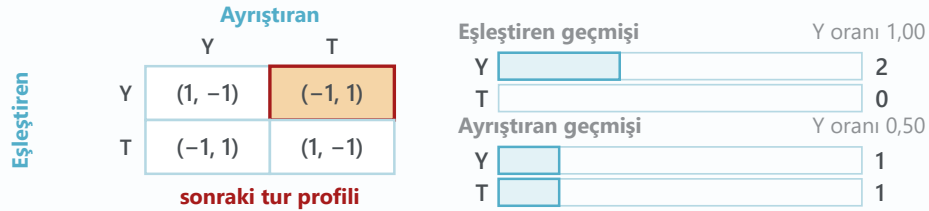
Adım 4 / 13

```
1  def fictitious(T):
2      ne, na = [0, 0], [0, 0]   # Y/T sayaçları
3      a, b = 0, 0               # 1. tur: ikisi de Y
▶      for t in range(1, T + 1):
▶          ne[a] += 1            # eşleştirenin geçmişi
▶          na[b] += 1            # ayrıştıranın geçmişi
7          # eşleştiren: ayrıştıranın çoğunluğunu eşle
8          a = 0 if na[0] >= na[1] else 1
9          # ayrıştıran: eşleştirenin çoğunluğundan ayrış
10         b = 1 if ne[0] > ne[1] else 0
11     return ne, na
```

DEĞİŞKENLER

`t = 2` `a = 0 (Y)` `b = 1 (T)` `ne = [2, 0]` `na = [1, 1]`

Kod İzleme: Kurgusal Oyun (Yazı-Tura, 6 tur)



Sonraki tur: ayrıştıranın sayaçları eşit (eşitlikte Y) → eşleştiren **Y**; eşleştiren çoğunlukla Y oynadı → ayrıştıran **T**.

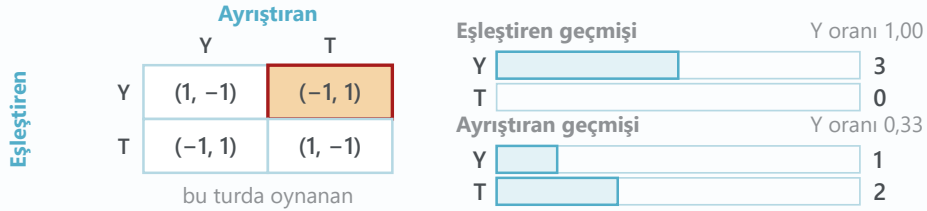
Adım 5 / 13

```
1  def fictitious(T):
2      ne, na = [0, 0], [0, 0]   # Y/T sayaçları
3      a, b = 0, 0               # 1. tur: ikisi de Y
4      for t in range(1, T + 1):
5          ne[a] += 1            # eşleştirenin geçmişi
6          na[b] += 1            # ayrıştıranın geçmişi
7          # eşleştiren: ayrıştıranın çoğunluğunu eşle
▶          a = 0 if na[0] >= na[1] else 1
9          # ayrıştıran: eşleştirenin çoğunluğundan ayrış
▶          b = 1 if ne[0] > ne[1] else 0
11     return ne, na
```

DEĞİŞKENLER

`t = 2` `a = 0 (Y)` `b = 1 (T)` `ne = [2, 0]` `na = [1, 1]`

Kod İzleme: Kurgusal Oyun (Yazı-Tura, 6 tur)



Tur 3: (Y, T) oynanır, ayrıştıran kazanır. Sayaçlar: eşleştiren Y/T = 3/0, ayrıştıran Y/T = 1/2.

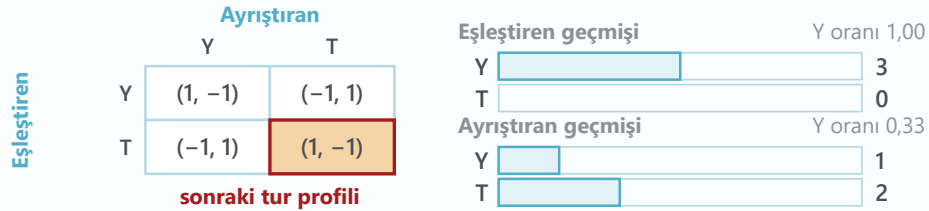
Adım 6 / 13

```
1  def fictitious(T):
2      ne, na = [0, 0], [0, 0]   # Y/T sayaçları
3      a, b = 0, 0               # 1. tur: ikisi de Y
▶      for t in range(1, T + 1):
▶          ne[a] += 1            # eşleştirenin geçmişi
▶          na[b] += 1            # ayrıştıranın geçmişi
7          # eşleştiren: ayrıştıranın çoğunluğunu eşle
8          a = 0 if na[0] >= na[1] else 1
9          # ayrıştıran: eşleştirenin çoğunluğundan ayrış
10         b = 1 if ne[0] > ne[1] else 0
11     return ne, na
```

DEĞİŞKENLER

`t = 3` `a = 0 (Y)` `b = 1 (T)` `ne = [3, 0]` `na = [1, 2]`

Kod İzleme: Kurgusal Oyun (Yazı-Tura, 6 tur)



Sonraki tur: ayrıştıran çoğunlukla T oynadı → eşleştiren **T**; eşleştiren çoğunlukla Y oynadı → ayrıştıran **T**.

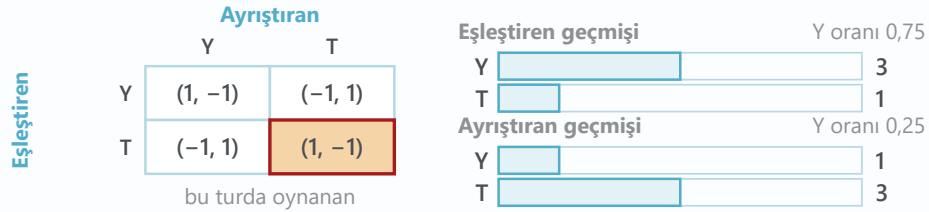
Adım 7 / 13

```
1  def fictitious(T):
2      ne, na = [0, 0], [0, 0]   # Y/T sayaçları
3      a, b = 0, 0               # 1. tur: ikisi de Y
4      for t in range(1, T + 1):
5          ne[a] += 1            # eşleştirenin geçmişi
6          na[b] += 1            # ayrıştıranın geçmişi
7          # eşleştiren: ayrıştıranın çoğunluğunu eşle
▶          a = 0 if na[0] >= na[1] else 1
9          # ayrıştıran: eşleştirenin çoğunluğundan ayrış
▶          b = 1 if ne[0] > ne[1] else 0
11     return ne, na
```

DEĞİŞKENLER

`t = 3` `a = 1 (T)` `b = 1 (T)` `ne = [3, 0]` `na = [1, 2]`

Kod İzleme: Kurgusal Oyun (Yazı-Tura, 6 tur)



Tur 4: (T, T) oynanır, eşleştiren kazanır. Sayaçlar: eşleştiren Y/T = 3/1, ayrıştıran Y/T = 1/3.

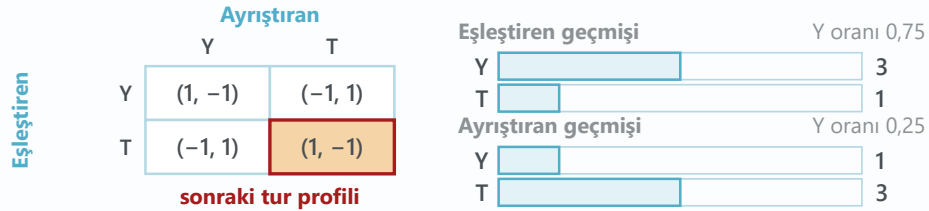
Adım 8 / 13

```
1  def fictitious(T):
2      ne, na = [0, 0], [0, 0]   # Y/T sayaçları
3      a, b = 0, 0               # 1. tur: ikisi de Y
▶      for t in range(1, T + 1):
▶          ne[a] += 1            # eşleştirenin geçmişi
▶          na[b] += 1            # ayrıştıranın geçmişi
7          # eşleştiren: ayrıştıranın çoğunluğunu eşle
8          a = 0 if na[0] >= na[1] else 1
9          # ayrıştıran: eşleştirenin çoğunluğundan ayrış
10         b = 1 if ne[0] > ne[1] else 0
11     return ne, na
```

DEĞİŞKENLER

`t = 4` `a = 1 (T)` `b = 1 (T)` `ne = [3, 1]` `na = [1, 3]`

Kod İzleme: Kurgusal Oyun (Yazı-Tura, 6 tur)



Sonraki tur: ayrıştıran çoğunlukla T oynadı → eşleştiren **T**; eşleştiren çoğunlukla Y oynadı → ayrıştıran **T**.

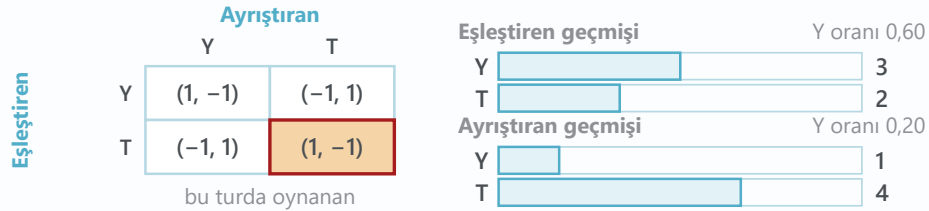
Adım 9 / 13

```
1  def fictitious(T):
2      ne, na = [0, 0], [0, 0]   # Y/T sayaçları
3      a, b = 0, 0               # 1. tur: ikisi de Y
4      for t in range(1, T + 1):
5          ne[a] += 1            # eşleştirenin geçmişi
6          na[b] += 1            # ayrıştıranın geçmişi
7          # eşleştiren: ayrıştıranın çoğunluğunu eşle
▶          a = 0 if na[0] >= na[1] else 1
9          # ayrıştıran: eşleştirenin çoğunluğundan ayrış
▶          b = 1 if ne[0] > ne[1] else 0
11     return ne, na
```

DEĞİŞKENLER

`t = 4` `a = 1 (T)` `b = 1 (T)` `ne = [3, 1]` `na = [1, 3]`

Kod İzleme: Kurgusal Oyun (Yazı-Tura, 6 tur)



Tur 5: (T, T) oynanır, eşleştiren kazanır. Sayaçlar: eşleştiren Y/T = 3/2, ayrıştıran Y/T = 1/4.

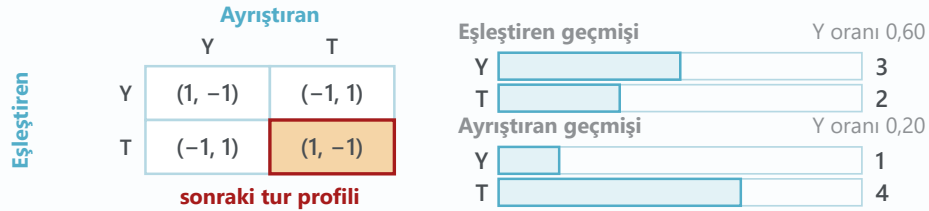
Adım 10 / 13

```
1  def fictitious(T):
2      ne, na = [0, 0], [0, 0]   # Y/T sayaçları
3      a, b = 0, 0               # 1. tur: ikisi de Y
▶      for t in range(1, T + 1):
▶          ne[a] += 1            # eşleştirenin geçmişi
▶          na[b] += 1            # ayrıştıranın geçmişi
7          # eşleştiren: ayrıştıranın çoğunluğunu eşle
8          a = 0 if na[0] >= na[1] else 1
9          # ayrıştıran: eşleştirenin çoğunluğundan ayrış
10         b = 1 if ne[0] > ne[1] else 0
11     return ne, na
```

DEĞİŞKENLER

`t = 5` `a = 1 (T)` `b = 1 (T)` `ne = [3, 2]` `na = [1, 4]`

Kod İzleme: Kurgusal Oyun (Yazı-Tura, 6 tur)



Sonraki tur: ayrıştıran çoğunlukla T oynadı → eşleştiren **T**; eşleştiren çoğunlukla Y oynadı → ayrıştıran **T**.

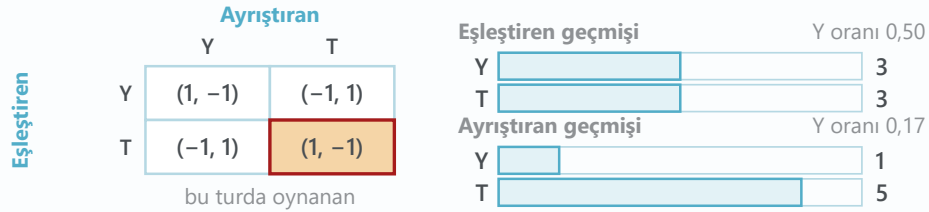
Adım 11 / 13

```
1  def fictitious(T):
2      ne, na = [0, 0], [0, 0]   # Y/T sayaçları
3      a, b = 0, 0               # 1. tur: ikisi de Y
4      for t in range(1, T + 1):
5          ne[a] += 1            # eşleştirenin geçmişi
6          na[b] += 1            # ayrıştıranın geçmişi
7          # eşleştiren: ayrıştıranın çoğunluğunu eşle
▶          a = 0 if na[0] >= na[1] else 1
9          # ayrıştıran: eşleştirenin çoğunluğundan ayrış
▶          b = 1 if ne[0] > ne[1] else 0
11     return ne, na
```

DEĞİŞKENLER

`t = 5` `a = 1 (T)` `b = 1 (T)` `ne = [3, 2]` `na = [1, 4]`

Kod İzleme: Kurgusal Oyun (Yazı-Tura, 6 tur)



Tur 6: (T, T) oynanır, eşleştiren kazanır. Sayaçlar: eşleştiren Y/T = 3/3, ayrıştıran Y/T = 1/5.

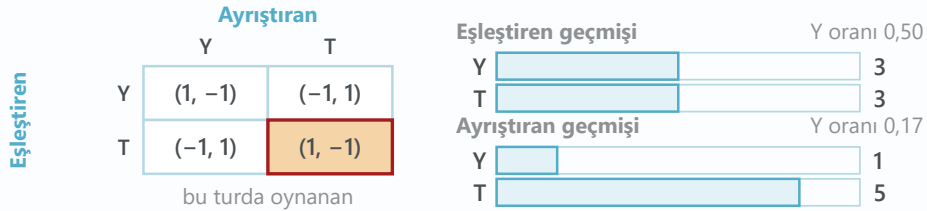
Adım 12 / 13

```
1  def fictitious(T):
2      ne, na = [0, 0], [0, 0]   # Y/T sayaçları
3      a, b = 0, 0               # 1. tur: ikisi de Y
▶      for t in range(1, T + 1):
▶          ne[a] += 1            # eşleştirenin geçmişi
▶          na[b] += 1            # ayrıştıranın geçmişi
7          # eşleştiren: ayrıştıranın çoğunluğunu eşle
8          a = 0 if na[0] >= na[1] else 1
9          # ayrıştıran: eşleştirenin çoğunluğundan ayrış
10         b = 1 if ne[0] > ne[1] else 0
11     return ne, na
```

DEĞİŞKENLER

`t = 6` `a = 1 (T)` `b = 1 (T)` `ne = [3, 3]` `na = [1, 5]`

Kod İzleme: Kurgusal Oyun (Yazı-Tura, 6 tur)



return ne = [3, 3], na = [1, 5]. Eylemler döngüye girer (YY → YT → TT → TY …); döngüler uzadıkça frekanslar ½'ye yaklaşır (6 turda 0,50 ve 0,17; 400 turda 0,472 ve 0,492).

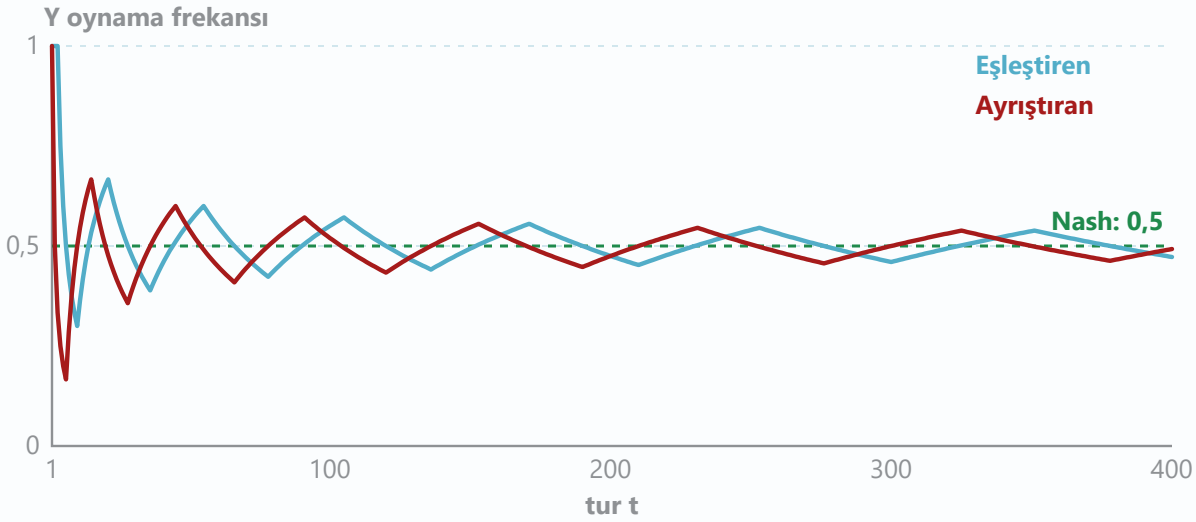
Adım 13 / 13

```
1  def fictitious(T):
2      ne, na = [0, 0], [0, 0]   # Y/T sayaçları
3      a, b = 0, 0               # 1. tur: ikisi de Y
4      for t in range(1, T + 1):
5          ne[a] += 1            # eşleştirenin geçmişi
6          na[b] += 1            # ayrıştıranın geçmişi
7          # eşleştiren: ayrıştıranın çoğunluğunu eşle
8          a = 0 if na[0] >= na[1] else 1
9          # ayrıştıran: eşleştirenin çoğunluğundan ayrış
10         b = 1 if ne[0] > ne[1] else 0
▶      return ne, na
```

DEĞİŞKENLER

`ne = [3, 3]` `na = [1, 5]`

Kurgusal Oyun: Frekanslar Dengeye Yakınsar



- Eylemler döngüler hâlinde değişir (döngüler giderek uzar), ama **frekanslar** karışık dengeye (½, ½) yaklaşır: 400 turda 0,472 ve 0,492.
- Yakınsama garantisi: iki kişilik sıfır toplamlı oyunlar (Robinson 1951), 2×2 oyunlar, potansiyel oyunlar. Shapley (1964): 3×3 bir oyunda frekanslar döner, yakınsamaz.

Dışsal Pişmanlık: “Lideri İzle” Neden Başarısız?

Pişmanlık = (geriye bakınca en iyi tek eylemi hep oynasaydın alacağın toplam) − (gerçekte aldığın toplam). **Dışsal pişmanlık** $R_T = \max_i \Sigma_t\, g_t(i) - \Sigma_t\, g_t(a_t)$: geriye dönük en iyi *sabit* eylemle fark. $R_T/T \to 0$ ise algoritma **pişmanlıksızdır**.

t	Önceki toplam A / B	g(A)	g(B)	Lideri izle	Kazanç
1	0,0 / 0,0	0,5	0,0	**A**	0,5
2					
3					
4					
5					
6					

Lider A (toplam 0,0 / 0,0) → **A** seçilir; tur kazancı 0,5. Algoritmanın toplamı: 0,5.

Dışsal Pişmanlık: “Lideri İzle” Neden Başarısız?

Pişmanlık = (geriye bakınca en iyi tek eylemi hep oynasaydın alacağın toplam) − (gerçekte aldığın toplam). **Dışsal pişmanlık** $R_T = \max_i \Sigma_t\, g_t(i) - \Sigma_t\, g_t(a_t)$: geriye dönük en iyi *sabit* eylemle fark. $R_T/T \to 0$ ise algoritma **pişmanlıksızdır**.

t	Önceki toplam A / B	g(A)	g(B)	Lideri izle	Kazanç
1	0,0 / 0,0	0,5	0,0	**A**	0,5
2	0,5 / 0,0	0,0	1,0	**A**	0,0
3					
4					
5					
6					

Lider A (toplam 0,5 / 0,0) → **A** seçilir; tur kazancı 0,0. Algoritmanın toplamı: 0,5.

Dışsal Pişmanlık: “Lideri İzle” Neden Başarısız?

Pişmanlık = (geriye bakınca en iyi tek eylemi hep oynasaydın alacağın toplam) − (gerçekte aldığın toplam). **Dışsal pişmanlık** $R_T = \max_i \Sigma_t\, g_t(i) - \Sigma_t\, g_t(a_t)$: geriye dönük en iyi *sabit* eylemle fark. $R_T/T \to 0$ ise algoritma **pişmanlıksızdır**.

t	Önceki toplam A / B	g(A)	g(B)	Lideri izle	Kazanç
1	0,0 / 0,0	0,5	0,0	**A**	0,5
2	0,5 / 0,0	0,0	1,0	**A**	0,0
3	0,5 / 1,0	1,0	0,0	**B**	0,0
4					
5					
6					

Lider B (toplam 0,5 / 1,0) → **B** seçilir; tur kazancı 0,0. Algoritmanın toplamı: 0,5.

Dışsal Pişmanlık: “Lideri İzle” Neden Başarısız?

Pişmanlık = (geriye bakınca en iyi tek eylemi hep oynasaydın alacağın toplam) − (gerçekte aldığın toplam). **Dışsal pişmanlık** $R_T = \max_i \Sigma_t\, g_t(i) - \Sigma_t\, g_t(a_t)$: geriye dönük en iyi *sabit* eylemle fark. $R_T/T \to 0$ ise algoritma **pişmanlıksızdır**.

t	Önceki toplam A / B	g(A)	g(B)	Lideri izle	Kazanç
1	0,0 / 0,0	0,5	0,0	**A**	0,5
2	0,5 / 0,0	0,0	1,0	**A**	0,0
3	0,5 / 1,0	1,0	0,0	**B**	0,0
4	1,5 / 1,0	0,0	1,0	**A**	0,0
5					
6					

Lider A (toplam 1,5 / 1,0) → **A** seçilir; tur kazancı 0,0. Algoritmanın toplamı: 0,5.

Dışsal Pişmanlık: “Lideri İzle” Neden Başarısız?

Pişmanlık = (geriye bakınca en iyi tek eylemi hep oynasaydın alacağın toplam) − (gerçekte aldığın toplam). **Dışsal pişmanlık** $R_T = \max_i \Sigma_t\, g_t(i) - \Sigma_t\, g_t(a_t)$: geriye dönük en iyi *sabit* eylemle fark. $R_T/T \to 0$ ise algoritma **pişmanlıksızdır**.

t	Önceki toplam A / B	g(A)	g(B)	Lideri izle	Kazanç
1	0,0 / 0,0	0,5	0,0	**A**	0,5
2	0,5 / 0,0	0,0	1,0	**A**	0,0
3	0,5 / 1,0	1,0	0,0	**B**	0,0
4	1,5 / 1,0	0,0	1,0	**A**	0,0
5	1,5 / 2,0	1,0	0,0	**B**	0,0
6					

Lider B (toplam 1,5 / 2,0) → **B** seçilir; tur kazancı 0,0. Algoritmanın toplamı: 0,5.

Dışsal Pişmanlık: “Lideri İzle” Neden Başarısız?

Pişmanlık = (geriye bakınca en iyi tek eylemi hep oynasaydın alacağın toplam) − (gerçekte aldığın toplam). **Dışsal pişmanlık** $R_T = \max_i \Sigma_t\, g_t(i) - \Sigma_t\, g_t(a_t)$: geriye dönük en iyi *sabit* eylemle fark. $R_T/T \to 0$ ise algoritma **pişmanlıksızdır**.

t	Önceki toplam A / B	g(A)	g(B)	Lideri izle	Kazanç
1	0,0 / 0,0	0,5	0,0	**A**	0,5
2	0,5 / 0,0	0,0	1,0	**A**	0,0
3	0,5 / 1,0	1,0	0,0	**B**	0,0
4	1,5 / 1,0	0,0	1,0	**A**	0,0
5	1,5 / 2,0	1,0	0,0	**B**	0,0
6	2,5 / 2,0	0,0	1,0	**A**	0,0

Lider A (toplam 2,5 / 2,0) → **A** seçilir; tur kazancı 0,0. Algoritmanın toplamı: 0,5. En iyi sabit eylem B: 3. **Pişmanlık = 3 − 0,5 = 2,5 ≈ T/2**.

Çarpımsal Ağırlıklar / Hedge

1. Her eyleme ağırlık ver: $w_i = 1$.
2. Her turda olasılıkla oyna: $p_i = w_i / \Sigma_j w_j$.
3. Tüm eylemlerin kazancını gör, güncelle:

$$w_i \leftarrow w_i \cdot e^{\eta \cdot g_t(i)}$$

Kazancı 1 olan eylemin ağırlığı e^{η} katına çıkar ($\eta = \ln 2$ ise tam 2 katına).

Teorem. Kazançlar [0, 1] aralığında ve n eylem varsa $R_T \le (\ln n)/\eta + \eta T$ ($\ln n$: n'nin doğal logaritması; $n = 3$ için $\approx 1{,}1$). $\eta = \sqrt{(\ln n / T)}$ seçilirse $R_T \le 2\sqrt{(T \ln n)}$, yani $\mathbf{R_T/T \to 0}$.

Rastgelelik şart

Lideri izle deterministik olduğu için rakip onu sömürür; Hedge karışık oynar.

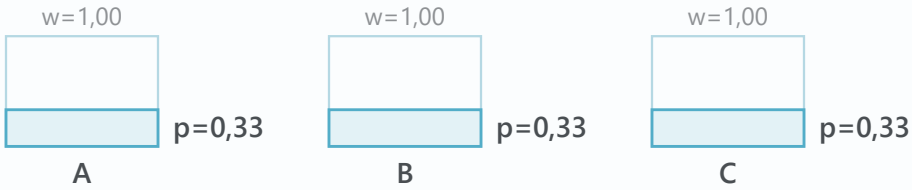
η: öğrenme oranı

Büyük η hızlı tepki ama gürültüye duyarlı; küçük η yavaş ama kararlı.

Oyunlarda

Herkes pişmanlıksız oynarsa zaman ortalaması *kaba korelasyonlu dengeye*, sıfır toplamlıda minimaksa yaklaşır.

Kod İzleme: Çarpımsal Ağırlıklar



Üç uzman (eylem) A, B, C; η = 0,5. Başlangıçta ağırlıklar eşit: **w = [1, 1, 1]**.

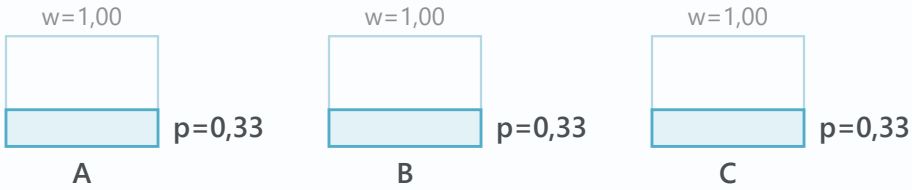
Adım 1 / 17

```
1  import math
2  def hedge(G, eta):
▶      w = [1.0] * len(G[0])
▶      top = 0.0
5      for g in G:                          # her tur
6          W = sum(w)
7          p = [x / W for x in w]
8          top += sum(pi * gi for pi, gi in zip(p, g))
9          w = [x*math.exp(eta*gi) for x, gi in zip(w, g)]
10     return top
```

DEĞİŞKENLER

`eta = 0.5` `w = [1.00, 1.00, 1.00]` `top = 0.00`

Kod İzleme: Çarpımsal Ağırlıklar



Tur 1: W = 3,00, olasılıklar **p = [0,33, 0,33, 0,33]**.

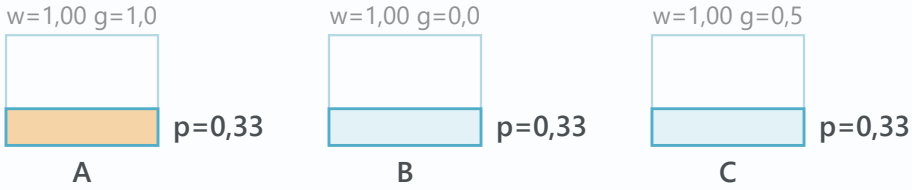
Adım 2 / 17

```
1  import math
2  def hedge(G, eta):
3      w = [1.0] * len(G[0])
4      top = 0.0
5      for g in G:                          # her tur
▶          W = sum(w)
▶          p = [x / W for x in w]
8          top += sum(pi * gi for pi, gi in zip(p, g))
9          w = [x*math.exp(eta*gi) for x, gi in zip(w, g)]
10     return top
```

DEĞİŞKENLER

- t = 1
- W = 3.00
- p = [0.33, 0.33, 0.33]
- w = [1.00, 1.00, 1.00]
- top = 0.00

Kod İzleme: Çarpımsal Ağırlıklar



Kazançlar açıklanır g = [1, 0, 0,5]. Beklenen kazanç Σ p·g = 0,50 → **top = 0,50**.

Adım 3 / 17

```
1  import math
2  def hedge(G, eta):
3      w = [1.0] * len(G[0])
4      top = 0.0
5      for g in G:                          # her tur
6          W = sum(w)
7          p = [x / W for x in w]
▶          top += sum(pi * gi for pi, gi in zip(p, g))
9          w = [x*math.exp(eta*gi) for x, gi in zip(w, g)]
10     return top
```

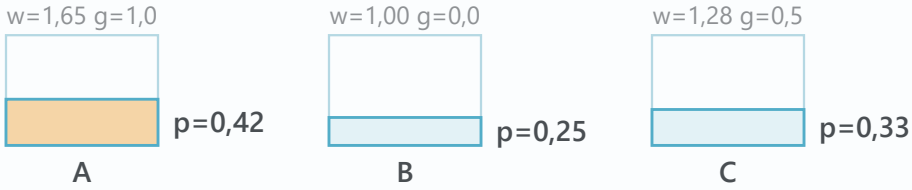
DEĞİŞKENLER

`t = 1` `g = [1, 0, 0.5]`

`p = [0.33, 0.33, 0.33]`

`top = 0.50`

Kod İzleme: Çarpımsal Ağırlıklar



Güncelleme $w \leftarrow w \cdot e^{0,5 \cdot g}$: **w = [1,65, 1,00, 1,28]**. Bu turda kazancı yüksek olan uzmanın ağırlığı büyür.

Adım 4 / 17

```
1  import math
2  def hedge(G, eta):
3      w = [1.0] * len(G[0])
4      top = 0.0
5      for g in G:                          # her tur
6          W = sum(w)
7          p = [x / W for x in w]
8          top += sum(pi * gi for pi, gi in zip(p, g))
▶          w = [x*math.exp(eta*gi) for x, gi in zip(w, g)]
10     return top
```

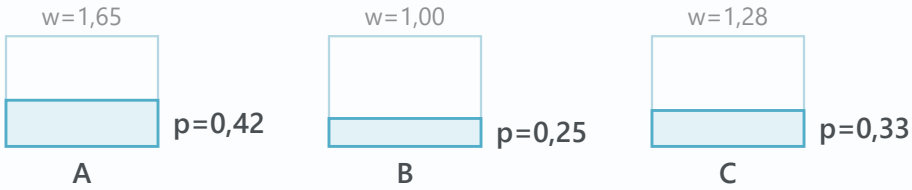
DEĞİŞKENLER

`t = 1` `g = [1, 0, 0.5]`

`w = [1.65, 1.00, 1.28]`

`top = 0.50`

Kod İzleme: Çarpımsal Ağırlıklar



Tur 2: W = 3,93, olasılıklar **p = [0,42, 0,25, 0,33]**.

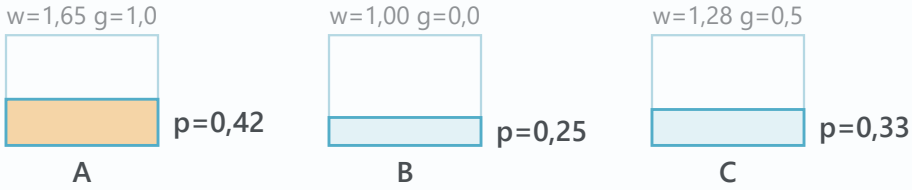
Adım 5 / 17

```
1  import math
2  def hedge(G, eta):
3      w = [1.0] * len(G[0])
4      top = 0.0
5      for g in G:                          # her tur
▶          W = sum(w)
▶          p = [x / W for x in w]
8          top += sum(pi * gi for pi, gi in zip(p, g))
9          w = [x*math.exp(eta*gi) for x, gi in zip(w, g)]
10     return top
```

DEĞİŞKENLER

- `t = 2`
- `W = 3.93`
- `p = [0.42, 0.25, 0.33]`
- `w = [1.65, 1.00, 1.28]`
- `top = 0.50`

Kod İzleme: Çarpımsal Ağırlıklar



Kazançlar açıklanır g = [1, 0, 0,5]. Beklenen kazanç Σ p·g = 0,58 → **top = 1,08**.

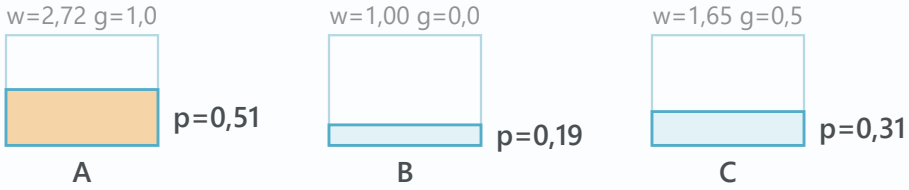
Adım 6 / 17

```
1  import math
2  def hedge(G, eta):
3      w = [1.0] * len(G[0])
4      top = 0.0
5      for g in G:                          # her tur
6          W = sum(w)
7          p = [x / W for x in w]
▶          top += sum(pi * gi for pi, gi in zip(p, g))
9          w = [x*math.exp(eta*gi) for x, gi in zip(w, g)]
10     return top
```

DEĞİŞKENLER

`t = 2` `g = [1, 0, 0.5]` `p = [0.42, 0.25, 0.33]` `top = 1.08`

Kod İzleme: Çarpımsal Ağırlıklar



Güncelleme $w \leftarrow w \cdot e^{0,5 \cdot g}$: **w = [2,72, 1,00, 1,65]**. Bu turda kazancı yüksek olan uzmanın ağırlığı büyür.

Adım 7 / 17

```
1  import math
2  def hedge(G, eta):
3      w = [1.0] * len(G[0])
4      top = 0.0
5      for g in G:                          # her tur
6          W = sum(w)
7          p = [x / W for x in w]
8          top += sum(pi * gi for pi, gi in zip(p, g))
▶          w = [x*math.exp(eta*gi) for x, gi in zip(w, g)]
10     return top
```

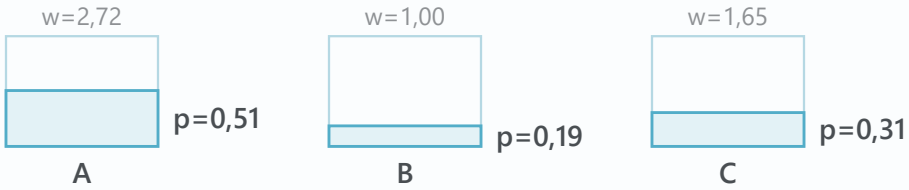
DEĞİŞKENLER

`t = 2` `g = [1, 0, 0.5]`

`w = [2.72, 1.00, 1.65]`

`top = 1.08`

Kod İzleme: Çarpımsal Ağırlıklar



Tur 3: W = 5,37, olasılıklar **p = [0,51, 0,19, 0,31]**.

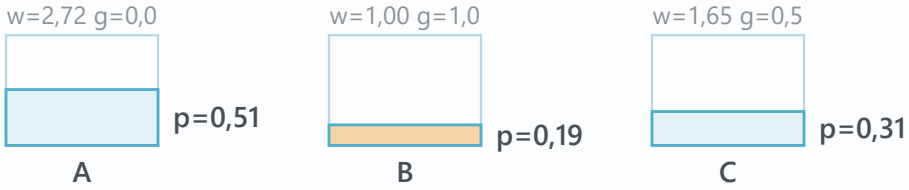
Adım 8 / 17

```
1  import math
2  def hedge(G, eta):
3      w = [1.0] * len(G[0])
4      top = 0.0
5      for g in G:                          # her tur
▶          W = sum(w)
▶          p = [x / W for x in w]
8          top += sum(pi * gi for pi, gi in zip(p, g))
9          w = [x*math.exp(eta*gi) for x, gi in zip(w, g)]
10     return top
```

DEĞİŞKENLER

- `t = 3`
- `W = 5.37`
- `p = [0.51, 0.19, 0.31]`
- `w = [2.72, 1.00, 1.65]`
- `top = 1.08`

Kod İzleme: Çarpımsal Ağırlıklar



Kazançlar açıklanır g = [0, 1, 0,5]. Beklenen kazanç Σ p·g = 0,34 → **top = 1,42**.

Adım 9 / 17

```
1  import math
2  def hedge(G, eta):
3      w = [1.0] * len(G[0])
4      top = 0.0
5      for g in G:                          # her tur
6          W = sum(w)
7          p = [x / W for x in w]
▶          top += sum(pi * gi for pi, gi in zip(p, g))
9          w = [x*math.exp(eta*gi) for x, gi in zip(w, g)]
10     return top
```

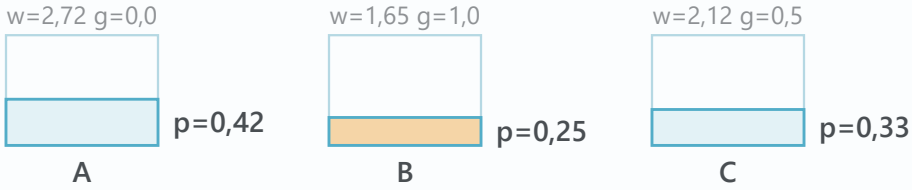
DEĞİŞKENLER

`t = 3` `g = [0, 1, 0.5]`

`p = [0.51, 0.19, 0.31]`

`top = 1.42`

Kod İzleme: Çarpımsal Ağırlıklar



Güncelleme $w \leftarrow w \cdot e^{0,5 \cdot g}$: **w = [2,72, 1,65, 2,12]**. Bu turda kazancı yüksek olan uzmanın ağırlığı büyür.

Adım 10 / 17

```
1  import math
2  def hedge(G, eta):
3      w = [1.0] * len(G[0])
4      top = 0.0
5      for g in G:                          # her tur
6          W = sum(w)
7          p = [x / W for x in w]
8          top += sum(pi * gi for pi, gi in zip(p, g))
▶          w = [x*math.exp(eta*gi) for x, gi in zip(w, g)]
10     return top
```

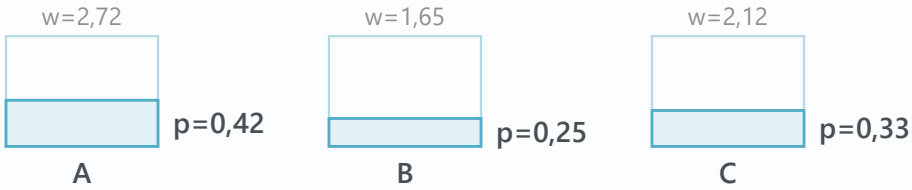
DEĞİŞKENLER

`t = 3` `g = [0, 1, 0.5]`

`w = [2.72, 1.65, 2.12]`

`top = 1.42`

Kod İzleme: Çarpımsal Ağırlıklar



Tur 4: W = 6,48, olasılıklar **p = [0,42, 0,25, 0,33]**.

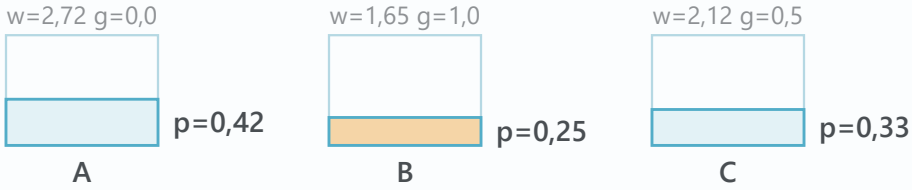
Adım 11 / 17

```
1  import math
2  def hedge(G, eta):
3      w = [1.0] * len(G[0])
4      top = 0.0
5      for g in G:                          # her tur
▶          W = sum(w)
▶          p = [x / W for x in w]
8          top += sum(pi * gi for pi, gi in zip(p, g))
9          w = [x*math.exp(eta*gi) for x, gi in zip(w, g)]
10     return top
```

DEĞİŞKENLER

- `t = 4`
- `W = 6.48`
- `p = [0.42, 0.25, 0.33]`
- `w = [2.72, 1.65, 2.12]`
- `top = 1.42`

Kod İzleme: Çarpımsal Ağırlıklar



Kazançlar açıklanır g = [0, 1, 0,5]. Beklenen kazanç Σ p·g = 0,42 → **top = 1,84**.

Adım 12 / 17

```
1  import math
2  def hedge(G, eta):
3      w = [1.0] * len(G[0])
4      top = 0.0
5      for g in G:                          # her tur
6          W = sum(w)
7          p = [x / W for x in w]
▶          top += sum(pi * gi for pi, gi in zip(p, g))
9          w = [x*math.exp(eta*gi) for x, gi in zip(w, g)]
10     return top
```

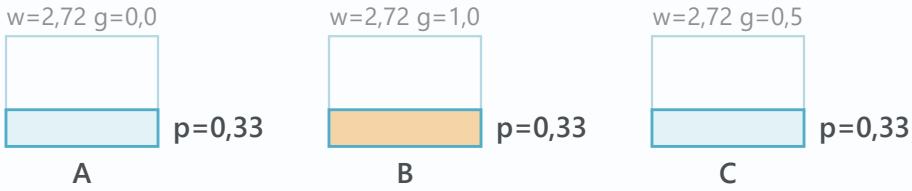
DEĞİŞKENLER

`t = 4` `g = [0, 1, 0.5]`

`p = [0.42, 0.25, 0.33]`

`top = 1.84`

Kod İzleme: Çarpımsal Ağırlıklar



Güncelleme $w \leftarrow w \cdot e^{0,5 \cdot g}$: **w = [2,72, 2,72, 2,72]**. Bu turda kazancı yüksek olan uzmanın ağırlığı büyür.

Adım 13 / 17

```
1  import math
2  def hedge(G, eta):
3      w = [1.0] * len(G[0])
4      top = 0.0
5      for g in G:                          # her tur
6          W = sum(w)
7          p = [x / W for x in w]
8          top += sum(pi * gi for pi, gi in zip(p, g))
▶          w = [x*math.exp(eta*gi) for x, gi in zip(w, g)]
10     return top
```

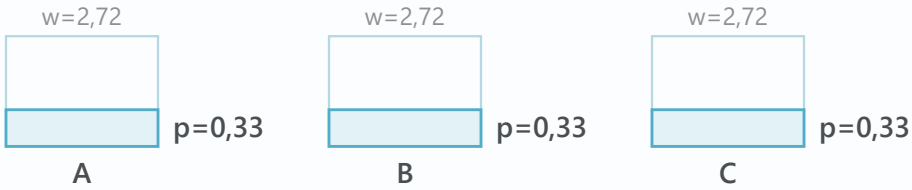
DEĞİŞKENLER

`t = 4` `g = [0, 1, 0.5]`

`w = [2.72, 2.72, 2.72]`

`top = 1.84`

Kod İzleme: Çarpımsal Ağırlıklar



Tur 5: W = 8,15, olasılıklar **p = [0,33, 0,33, 0,33]**.

Adım 14 / 17

```
1  import math
2  def hedge(G, eta):
3      w = [1.0] * len(G[0])
4      top = 0.0
5      for g in G:                          # her tur
▶          W = sum(w)
▶          p = [x / W for x in w]
8          top += sum(pi * gi for pi, gi in zip(p, g))
9          w = [x*math.exp(eta*gi) for x, gi in zip(w, g)]
10     return top
```

DEĞİŞKENLER

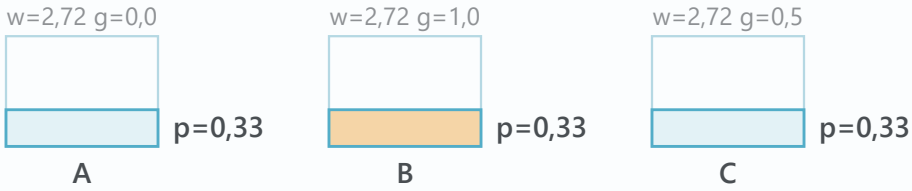
`t = 5` `W = 8.15`

`p = [0.33, 0.33, 0.33]`

`w = [2.72, 2.72, 2.72]`

`top = 1.84`

Kod İzleme: Çarpımsal Ağırlıklar



Kazançlar açıklanır g = [0, 1, 0,5]. Beklenen kazanç Σ p·g = 0,50 → **top = 2,34**.

Adım 15 / 17

```
1  import math
2  def hedge(G, eta):
3      w = [1.0] * len(G[0])
4      top = 0.0
5      for g in G:                          # her tur
6          W = sum(w)
7          p = [x / W for x in w]
▶          top += sum(pi * gi for pi, gi in zip(p, g))
9          w = [x*math.exp(eta*gi) for x, gi in zip(w, g)]
10     return top
```

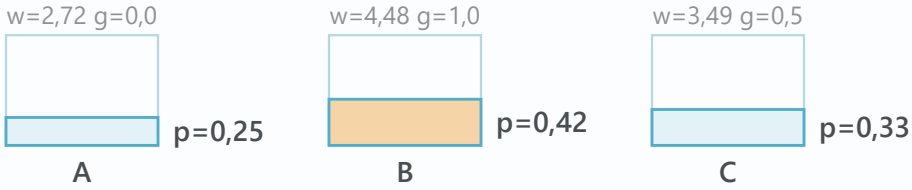
DEĞİŞKENLER

`t = 5` `g = [0, 1, 0.5]`

`p = [0.33, 0.33, 0.33]`

`top = 2.34`

Kod İzleme: Çarpımsal Ağırlıklar



Güncelleme $w \leftarrow w \cdot e^{0,5 \cdot g}$: **w = [2,72, 4,48, 3,49]**. Bu turda kazancı yüksek olan uzmanın ağırlığı büyür.

Adım 16 / 17

```
1  import math
2  def hedge(G, eta):
3      w = [1.0] * len(G[0])
4      top = 0.0
5      for g in G:                          # her tur
6          W = sum(w)
7          p = [x / W for x in w]
8          top += sum(pi * gi for pi, gi in zip(p, g))
▶          w = [x*math.exp(eta*gi) for x, gi in zip(w, g)]
10     return top
```

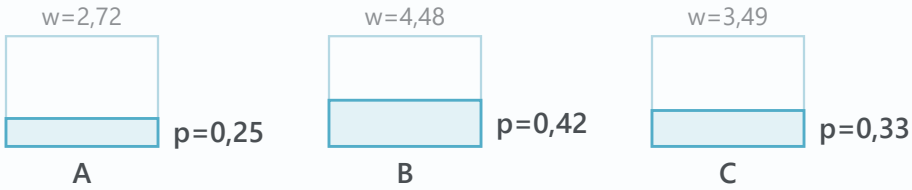
DEĞİŞKENLER

`t = 5` `g = [0, 1, 0.5]`

`w = [2.72, 4.48, 3.49]`

`top = 2.34`

Kod İzleme: Çarpımsal Ağırlıklar



return 2,34. En iyi sabit uzman B: 3. Pişmanlık = 3 − 2,34 = **0,66** (lideri izle 5 turda bundan kötü olabilir).

Adım 17 / 17

```
1 import math
2 def hedge(G, eta):
3     w = [1.0] * len(G[0])
4     top = 0.0
5     for g in G:                          # her tur
6         W = sum(w)
7         p = [x / W for x in w]
8         top += sum(pi * gi for pi, gi in zip(p, g))
9         w = [x*math.exp(eta*gi) for x, gi in zip(w, g)]
▶     return top
```

DEĞİŞKENLER

`w = [2.72, 4.48, 3.49]`

`top = 2.34`

Varsayımı Değiştir: Ya Rakip de Öğrenirse?

Taş-Kağıt-Makas: kazanan 1, berabere kalan ½, kaybeden 0 alır. Oyuncu 1 Hedge ile öğrenir ($\eta = 0{,}1$). Rakip hep (Taş 0,5; Kağıt 0,3; Makas 0,2) oynarsa Hedge nereye gider? **Rakip de Hedge ile öğrenirse ne değişir?**

Varsayımı Değiştir: Ya Rakip de Öğrenirse?

Taş-Kağıt-Makas: kazanan 1, berabere kalan ½, kaybeden 0 alır. Oyuncu 1 Hedge ile öğrenir ($\eta = 0{,}1$). Rakip hep (Taş 0,5; Kağıt 0,3; Makas 0,2) oynarsa Hedge nereye gider? **Rakip de Hedge ile öğrenirse ne değişir?**

Sabit rakip: en iyi tepkiye gider. Beklenen kazanç: Taş 0,45, **Kağıt 0,65**, Makas 0,40. Hedge'in Kağıt olasılığı 100. turda 0,82, 300. turda 0,997.

Varsayımı Değiştir: Ya Rakip de Öğrenirse?

Taş-Kağıt-Makas: kazanan 1, berabere kalan ½, kaybeden 0 alır. Oyuncu 1 Hedge ile öğrenir ($\eta = 0{,}1$). Rakip hep (Taş 0,5; Kağıt 0,3; Makas 0,2) oynarsa Hedge nereye gider? **Rakip de Hedge ile öğrenirse ne değişir?**

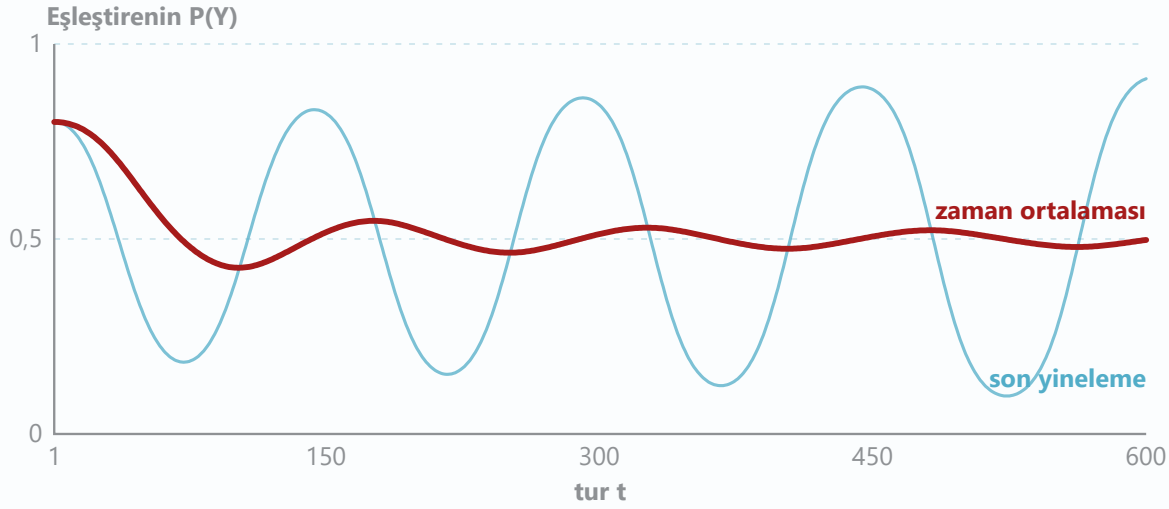
Sabit rakip: en iyi tepkiye gider. Beklenen kazanç: Taş 0,45, **Kağıt 0,65**, Makas 0,40. Hedge'in Kağıt olasılığı 100. turda 0,82, 300. turda 0,997.

Öğrenen rakip: hedef kayar. İkisi aynı anda güncellenir; oyuncu 1 (0,5; 0,3; 0,2), oyuncu 2 eşit olasılıklarla başlar.

tur	100	200	800	1000
o turdaki P(Taş)	0,19	0,46	0,14	0,19
ortalama P(Taş)	0,35	0,31	0,33	0,34

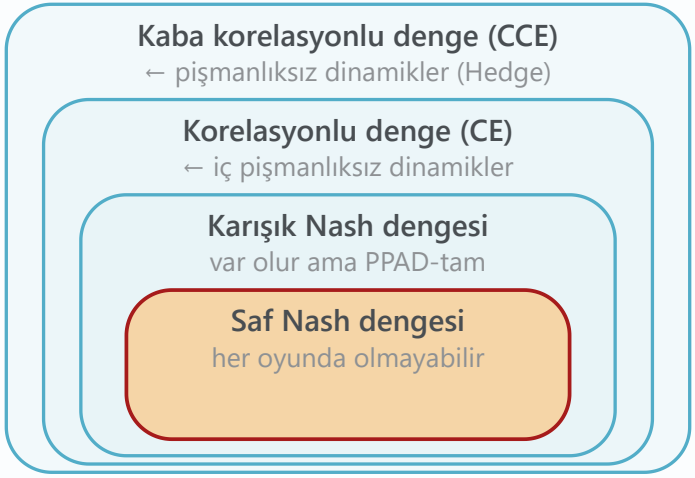
Son tur stratejisi dönmeye devam eder; zaman ortalaması (⅓, ⅓, ⅓) dengesine yaklaşır.

Hedge Kendine Karşı: Yazı-Tura Eşleştirme



- İki oyuncu da Hedge ($\eta = 0{,}1$) kullanıyor. Başlangıç ağırlıkları: eşleştiren (Y, T) = (0,8; 0,2), yani P(Y) = 0,8; ayrıştıran (0,5; 0,5). Kazanan 1, kaybeden 0 alır; her turda iki oyuncu da eylemlerinin beklenen kazancıyla aynı anda günceller.
- Her turdaki olasılık (**son yineleme**) dengeye oturmaz, etrafında büyük salınımlar yapar; **zaman ortalaması** minimaks stratejisine (½, ½) yaklaşır (600 turda 0,498). Minimaks teoreminin öğrenme yoluyla ispatı buna dayanır.

Pişmanlıksız Öğrenme Hangi Dengeye Gider?



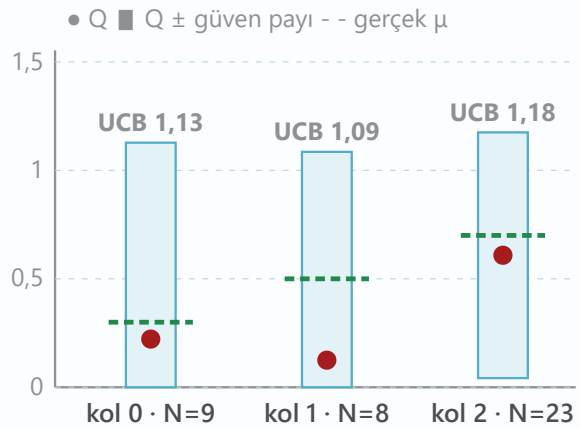
- Bir **öneri makinesi** herkese ne oynayacağını söyler (trafik ışığı gibi).
- **Korelasyonlu denge** (correlated equilibrium, CE): önerini *gördükten sonra* da uymak en iyisidir. Daha güçlü bir pişmanlık türünü (iç pişmanlık) küçülten algoritmalar CE'ye götürür.
- **Kaba korelasyonlu denge** (CCE) daha zayıftır: öneriyi görmeden "hep şunu oynarım" demek kazandırmaz. Herkes pişmanlıksız oynarsa *zaman ortalaması* CCE'ye yaklaşır.
- CE ve CCE hızlı hesaplanır; karışık Nash dengesi ise PPAD-tam.

Kavşaktaki trafik ışığı Tavuk oyununda bir **korelasyonlu denge** önerir: ışığa uymamak kimseye kazanç sağlamaz. Pişmanlıksız öğrenme bu dengeleri merkezî bir hesaplayıcı olmadan bulur.

Çok Kollu Haydut (Multi-Armed Bandit)

- K kollu kumar makinesi: kol i, ortalaması μ_i olan (bilinmeyen) rastgele ödül verir.
- Her turda **bir** kol çekilir ve yalnızca onun ödülü görülür (Hedge'den farkı: kısmi geri bildirim).
- Pişmanlık: $R_T = T \cdot \mu^* - \Sigma_t \mu_{a_t}$, μ^* = en iyi kolun ortalaması.

Keşif ve sömürü ikilemi: keşif yeni kollar hakkında bilgi toplar; sömürü o ana kadar en iyi görüneni kullanır. Yalnız sömürü yanlış kola takılır, yalnız keşif ödülü boşa harcar.



UCB1, μ = (0,3; 0,5; 0,7), 40 tur: az denenen kolun aralığı geniştir, bu yüzden yeniden denenir.

Uygulamalar: reklam ve haber önerisi, A/B testi, klinik deneyler, fiyat deneme.

Kod İzleme: ε-Açgözlü (ε = 0,2)



ε-açgözlü her tur ε olasılıkla rastgele kol çeker (keşif), yoksa en büyük Q'lu kolu (sömürü). Çağrı: `egreedy([0.3, 0.5, 0.7], 8, 0.2, random.Random(11))`. Tahminler Q = 0, sayaçlar N = 0. Kodda kollar 0'dan numaralanır: kol 0, 1, 2 = Kol 1, 2, 3.

Adım 1 / 18

```
1  def egreedy(mu, T, eps, rng):
▶      N, Q = [0] * len(mu), [0.0] * len(mu)
3      for t in range(1, T + 1):
4          if rng.random() < eps:           # keşif
5              a = rng.randrange(len(mu))
6          else:                            # sömürü
7              a = Q.index(max(Q))
8          r = 1 if rng.random() < mu[a] else 0
9          N[a] += 1
10         Q[a] += (r - Q[a]) / N[a]
11     return N, Q
```

DEĞİŞKENLER

`eps = 0.2` `N = [0, 0, 0]` `Q = [0.00, 0.00, 0.00]`

Kod İzleme: ε-Açgözlü (ε = 0,2)



t = 1: 0,452 ≥ 0,2 → **sömürü**: en büyük Q'lu kol **0** (eşitlikte ilk indis).

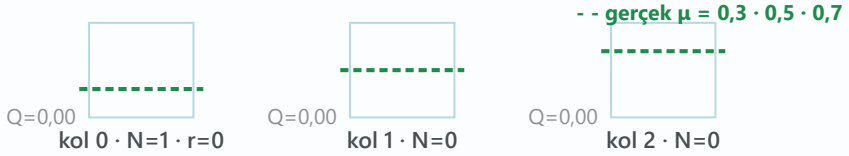

```
1  def egreedy(mu, T, eps, rng):
2      N, Q = [0] * len(mu), [0.0] * len(mu)
3      for t in range(1, T + 1):
▶          if rng.random() < eps:          # keşif
5              a = rng.randrange(len(mu))
6          else:                           # sömürü
▶              a = Q.index(max(Q))
8          r = 1 if rng.random() < mu[a] else 0
9          N[a] += 1
10         Q[a] += (r - Q[a]) / N[a]
11     return N, Q
```

DEĞİŞKENLER

`t = 1` `u = 0.452` `a = 0`

`N = [0, 0, 0]` `Q = [0.00, 0.00, 0.00]`

Kod İzleme: ε-Açgözlü (ε = 0,2)



Ödül **r = 0**; N[0] = 1, Q[0] = 0,00 (artımlı ortalama).

Adım 3 / 18

```
1  def egreedy(mu, T, eps, rng):
2      N, Q = [0] * len(mu), [0.0] * len(mu)
3      for t in range(1, T + 1):
4          if rng.random() < eps:           # keşif
5              a = rng.randrange(len(mu))
6          else:                            # sömürü
7              a = Q.index(max(Q))
▶          r = 1 if rng.random() < mu[a] else 0
▶          N[a] += 1
▶          Q[a] += (r - Q[a]) / N[a]
11     return N, Q
```

DEĞİŞKENLER

`t = 1` `a = 0` `r = 0` `N = [1, 0, 0]`

`Q = [0.00, 0.00, 0.00]`

Kod İzleme: ε-Açgözlü (ε = 0,2)



t = 2: 0,924 ≥ 0,2 → **sömürü**: en büyük Q'lu kol **0** (eşitlikte ilk indis).

Adım 4 / 18

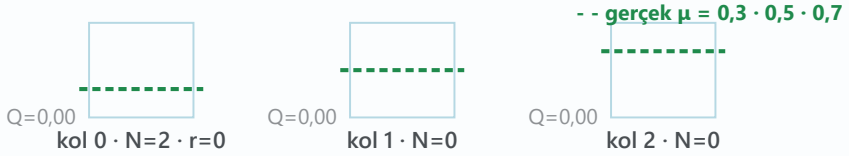
```
1  def egreedy(mu, T, eps, rng):
2      N, Q = [0] * len(mu), [0.0] * len(mu)
3      for t in range(1, T + 1):
▶          if rng.random() < eps:          # keşif
5              a = rng.randrange(len(mu))
6          else:                           # sömürü
▶              a = Q.index(max(Q))
8          r = 1 if rng.random() < mu[a] else 0
9          N[a] += 1
10         Q[a] += (r - Q[a]) / N[a]
11     return N, Q
```

DEĞİŞKENLER

`t = 2` `u = 0.924` `a = 0`

`N = [1, 0, 0]` `Q = [0.00, 0.00, 0.00]`

Kod İzleme: ε-Açgözlü (ε = 0,2)



Ödül **r = 0**; N[0] = 2, Q[0] = 0,00 (artımlı ortalama).


```
1  def egreedy(mu, T, eps, rng):
2      N, Q = [0] * len(mu), [0.0] * len(mu)
3      for t in range(1, T + 1):
4          if rng.random() < eps:           # keşif
5              a = rng.randrange(len(mu))
6          else:                            # sömürü
7              a = Q.index(max(Q))
           r = 1 if rng.random() < mu[a] else 0
           N[a] += 1
           Q[a] += (r - Q[a]) / N[a]
11     return N, Q
```

DEĞİŞKENLER

`t = 2` `a = 0` `r = 0` `N = [2, 0, 0]`

`Q = [0.00, 0.00, 0.00]`

Kod İzleme: ε-Açgözlü (ε = 0,2)



t = 3: 0,508 ≥ 0,2 → **sömürü**: en büyük Q'lu kol **0** (eşitlikte ilk indis).


```
1  def egreedy(mu, T, eps, rng):
2      N, Q = [0] * len(mu), [0.0] * len(mu)
3      for t in range(1, T + 1):
▶          if rng.random() < eps:          # keşif
5              a = rng.randrange(len(mu))
6          else:                           # sömürü
▶              a = Q.index(max(Q))
8          r = 1 if rng.random() < mu[a] else 0
9          N[a] += 1
10         Q[a] += (r - Q[a]) / N[a]
11     return N, Q
```

DEĞİŞKENLER

`t = 3` `u = 0.508` `a = 0`

`N = [2, 0, 0]` `Q = [0.00, 0.00, 0.00]`

Kod İzleme: ε-Açgözlü (ε = 0,2)



Ödül **r = 0**; N[0] = 3, Q[0] = 0,00 (artımlı ortalama).

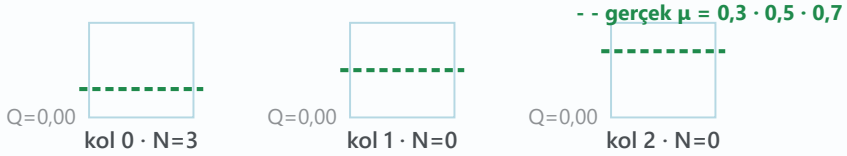

```
1  def egreedy(mu, T, eps, rng):
2      N, Q = [0] * len(mu), [0.0] * len(mu)
3      for t in range(1, T + 1):
4          if rng.random() < eps:           # keşif
5              a = rng.randrange(len(mu))
6          else:                            # sömürü
7              a = Q.index(max(Q))
▶          r = 1 if rng.random() < mu[a] else 0
▶          N[a] += 1
▶          Q[a] += (r - Q[a]) / N[a]
11     return N, Q
```

DEĞİŞKENLER

`t = 3` `a = 0` `r = 0` `N = [3, 0, 0]`

`Q = [0.00, 0.00, 0.00]`

Kod İzleme: ε-Açgözlü (ε = 0,2)



t = 4: rastgele sayı 0,185 < 0,2 → **keşif**, rastgele kol **2**.

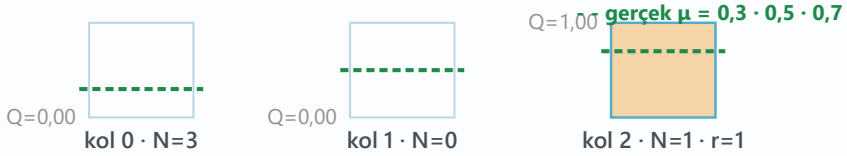

```
1  def egreedy(mu, T, eps, rng):
2      N, Q = [0] * len(mu), [0.0] * len(mu)
3      for t in range(1, T + 1):
▶          if rng.random() < eps:          # keşif
▶              a = rng.randrange(len(mu))
6          else:                           # sömürü
7              a = Q.index(max(Q))
8          r = 1 if rng.random() < mu[a] else 0
9          N[a] += 1
10         Q[a] += (r - Q[a]) / N[a]
11     return N, Q
```

DEĞİŞKENLER

`t = 4` `u = 0.185` `a = 2`

`N = [3, 0, 0]` `Q = [0.00, 0.00, 0.00]`

Kod İzleme: ε-Açgözlü (ε = 0,2)



Ödül **r = 1**; N[2] = 1, Q[2] = 1,00 (artımlı ortalama).

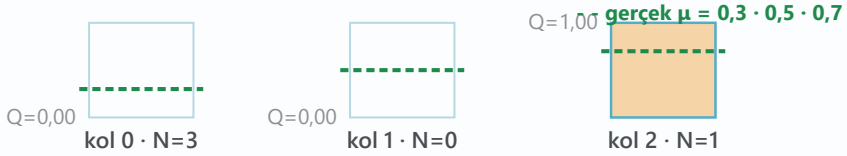

```
1  def egreedy(mu, T, eps, rng):
2      N, Q = [0] * len(mu), [0.0] * len(mu)
3      for t in range(1, T + 1):
4          if rng.random() < eps:           # keşif
5              a = rng.randrange(len(mu))
6          else:                            # sömürü
7              a = Q.index(max(Q))
▶          r = 1 if rng.random() < mu[a] else 0
▶          N[a] += 1
▶          Q[a] += (r - Q[a]) / N[a]
11     return N, Q
```

DEĞİŞKENLER

`t = 4` `a = 2` `r = 1` `N = [3, 0, 1]`

`Q = [0.00, 0.00, 1.00]`

Kod İzleme: ε-Açgözlü (ε = 0,2)



t = 5: 0,614 ≥ 0,2 → **sömürü**: en büyük Q'lu kol **2**.

Adım 10 / 18

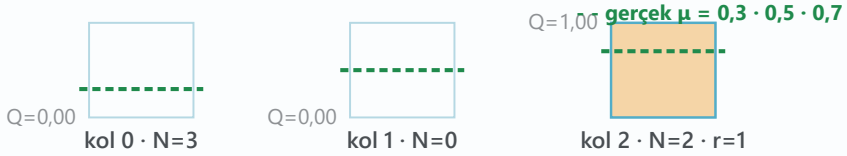
```
1  def egreedy(mu, T, eps, rng):
2      N, Q = [0] * len(mu), [0.0] * len(mu)
3      for t in range(1, T + 1):
▶          if rng.random() < eps:           # keşif
5              a = rng.randrange(len(mu))
6          else:                            # sömürü
▶              a = Q.index(max(Q))
8          r = 1 if rng.random() < mu[a] else 0
9          N[a] += 1
10         Q[a] += (r - Q[a]) / N[a]
11     return N, Q
```

DEĞİŞKENLER

`t = 5` `u = 0.614` `a = 2`

`N = [3, 0, 1]` `Q = [0.00, 0.00, 1.00]`

Kod İzleme: ε-Açgözlü (ε = 0,2)



Ödül **r = 1**; N[2] = 2, Q[2] = 1,00 (artımlı ortalama).

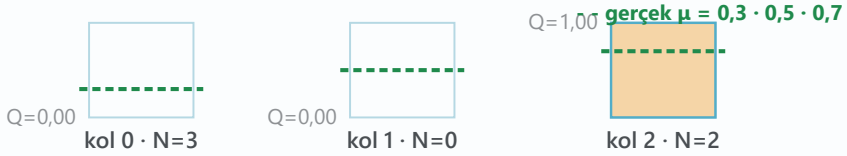

```
1  def egreedy(mu, T, eps, rng):
2      N, Q = [0] * len(mu), [0.0] * len(mu)
3      for t in range(1, T + 1):
4          if rng.random() < eps:           # keşif
5              a = rng.randrange(len(mu))
6          else:                            # sömürü
7              a = Q.index(max(Q))
▶          r = 1 if rng.random() < mu[a] else 0
▶          N[a] += 1
▶          Q[a] += (r - Q[a]) / N[a]
11     return N, Q
```

DEĞİŞKENLER

`t = 5` `a = 2` `r = 1` `N = [3, 0, 2]`

`Q = [0.00, 0.00, 1.00]`

Kod İzleme: ε-Açgözlü (ε = 0,2)



t = 6: 0,447 ≥ 0,2 → **sömürü**: en büyük Q'lu kol **2**.

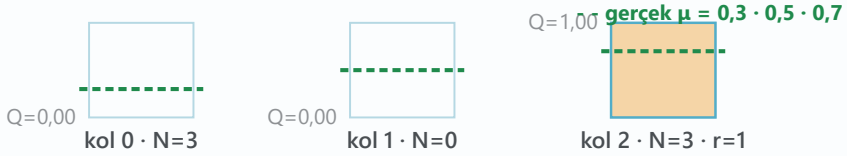

```
1  def egreedy(mu, T, eps, rng):
2      N, Q = [0] * len(mu), [0.0] * len(mu)
3      for t in range(1, T + 1):
▶          if rng.random() < eps:          # keşif
5              a = rng.randrange(len(mu))
6          else:                           # sömürü
▶              a = Q.index(max(Q))
8          r = 1 if rng.random() < mu[a] else 0
9          N[a] += 1
10         Q[a] += (r - Q[a]) / N[a]
11     return N, Q
```

DEĞİŞKENLER

`t = 6` `u = 0.447` `a = 2`

`N = [3, 0, 2]` `Q = [0.00, 0.00, 1.00]`

Kod İzleme: ε-Açgözlü (ε = 0,2)



Ödül **r = 1**; N[2] = 3, Q[2] = 1,00 (artımlı ortalama).


```
1  def egreedy(mu, T, eps, rng):
2      N, Q = [0] * len(mu), [0.0] * len(mu)
3      for t in range(1, T + 1):
4          if rng.random() < eps:           # keşif
5              a = rng.randrange(len(mu))
6          else:                            # sömürü
7              a = Q.index(max(Q))
           r = 1 if rng.random() < mu[a] else 0
           N[a] += 1
           Q[a] += (r - Q[a]) / N[a]
11     return N, Q
```

DEĞİŞKENLER

`t = 6` `a = 2` `r = 1` `N = [3, 0, 3]`

`Q = [0.00, 0.00, 1.00]`

Kod İzleme: ε-Açgözlü (ε = 0,2)



t = 7: 0,539 ≥ 0,2 → **sömürü**: en büyük Q'lu kol **2**.

Adım 14 / 18

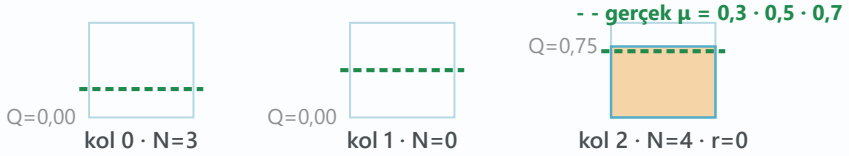
```
1  def egreedy(mu, T, eps, rng):
2      N, Q = [0] * len(mu), [0.0] * len(mu)
3      for t in range(1, T + 1):
▶          if rng.random() < eps:          # keşif
5              a = rng.randrange(len(mu))
6          else:                           # sömürü
▶              a = Q.index(max(Q))
8          r = 1 if rng.random() < mu[a] else 0
9          N[a] += 1
10         Q[a] += (r - Q[a]) / N[a]
11     return N, Q
```

DEĞİŞKENLER

`t = 7` `u = 0.539` `a = 2`

`N = [3, 0, 3]` `Q = [0.00, 0.00, 1.00]`

Kod İzleme: ε-Açgözlü (ε = 0,2)



Ödül **r = 0**; N[2] = 4, Q[2] = 0,75 (artımlı ortalama).

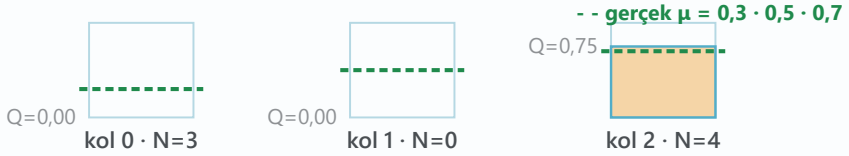

```
1  def egreedy(mu, T, eps, rng):
2      N, Q = [0] * len(mu), [0.0] * len(mu)
3      for t in range(1, T + 1):
4          if rng.random() < eps:           # keşif
5              a = rng.randrange(len(mu))
6          else:                            # sömürü
7              a = Q.index(max(Q))
           r = 1 if rng.random() < mu[a] else 0
           N[a] += 1
           Q[a] += (r - Q[a]) / N[a]
11     return N, Q
```

DEĞİŞKENLER

`t = 7` `a = 2` `r = 0` `N = [3, 0, 4]`

`Q = [0.00, 0.00, 0.75]`

Kod İzleme: ε-Açgözlü (ε = 0,2)



t = 8: 0,634 ≥ 0,2 → **sömürü**: en büyük Q'lu kol **2**.

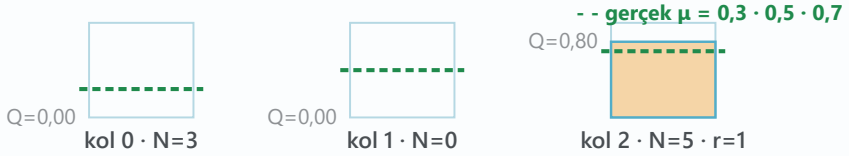

```
1  def egreedy(mu, T, eps, rng):
2      N, Q = [0] * len(mu), [0.0] * len(mu)
3      for t in range(1, T + 1):
▶          if rng.random() < eps:          # keşif
5              a = rng.randrange(len(mu))
6          else:                           # sömürü
▶              a = Q.index(max(Q))
8          r = 1 if rng.random() < mu[a] else 0
9          N[a] += 1
10         Q[a] += (r - Q[a]) / N[a]
11     return N, Q
```

DEĞİŞKENLER

`t = 8` `u = 0.634` `a = 2`

`N = [3, 0, 4]` `Q = [0.00, 0.00, 0.75]`

Kod İzleme: ε-Açgözlü (ε = 0,2)



Ödül **r = 1**; N[2] = 5, Q[2] = 0,80 (artımlı ortalama).


```
1  def egreedy(mu, T, eps, rng):
2      N, Q = [0] * len(mu), [0.0] * len(mu)
3      for t in range(1, T + 1):
4          if rng.random() < eps:           # keşif
5              a = rng.randrange(len(mu))
6          else:                            # sömürü
7              a = Q.index(max(Q))
▶          r = 1 if rng.random() < mu[a] else 0
▶          N[a] += 1
▶          Q[a] += (r - Q[a]) / N[a]
11     return N, Q
```

DEĞİŞKENLER

`t = 8` `a = 2` `r = 1` `N = [3, 0, 5]` `Q = [0.00, 0.00, 0.80]`

Kod İzleme: ε-Açgözlü (ε = 0,2)



return N = [3, 0, 5]. İlk 3 tur kol 0'a takıldı (hepsi 0 ödül, eşitlikte ilk kol); tek bir keşif en iyi kolu (2) buldu.

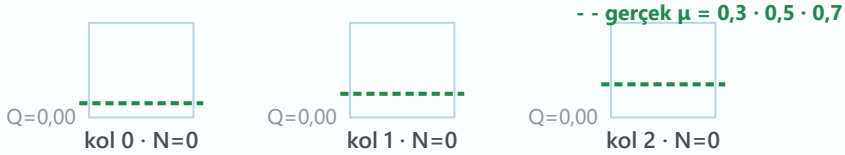
Adım 18 / 18

```
1  def egreedy(mu, T, eps, rng):
2      N, Q = [0] * len(mu), [0.0] * len(mu)
3      for t in range(1, T + 1):
4          if rng.random() < eps:           # keşif
5              a = rng.randrange(len(mu))
6          else:                            # sömürü
7              a = Q.index(max(Q))
8          r = 1 if rng.random() < mu[a] else 0
9          N[a] += 1
10         Q[a] += (r - Q[a]) / N[a]
▶      return N, Q
```

DEĞİŞKENLER

`N = [3, 0, 5]` `Q = [0.00, 0.00, 0.80]`

Kod İzleme: UCB1 (Üst Güven Sınırı)



Çağrı: `ucb([0.3, 0.5, 0.7], 8, random.Random(3))`. Tahminler ve sayaçlar sıfır. Puan Q + √(2 ln t / N): t şu anki tur numarası, N kolun çekilme sayısı. Az çekilen kolun payı büyüktür.

Adım 1 / 18

```
1  import math
2  def ucb(mu, T, rng):
▶      K = len(mu)
▶      N, Q = [0] * K, [0.0] * K
5      for t in range(1, T + 1):
6          b = [math.inf if n == 0 else
7               math.sqrt(2*math.log(t)/n) for n in N]
8          a = max(range(K), key=lambda i: Q[i] + b[i])
9          r = 1 if rng.random() < mu[a] else 0
10         N[a] += 1
11         Q[a] += (r - Q[a]) / N[a]
12     return N, Q
```

DEĞİŞKENLER

`K = 3` `N = [0, 0, 0]`

`Q = [0.00, 0.00, 0.00]`

Kod İzleme: UCB1 (Üst Güven Sınırı)



t = 1: hiç denenmemiş kolun payı ∞ → ilk denenmemiş kol **0** seçilir (her kol bir kez).


```
1  import math
2  def ucb(mu, T, rng):
3      K = len(mu)
4      N, Q = [0] * K, [0.0] * K
5      for t in range(1, T + 1):
▶          b = [math.inf if n == 0 else
▶               math.sqrt(2*math.log(t)/n) for n in N]
▶          a = max(range(K), key=lambda i: Q[i] + b[i])
9          r = 1 if rng.random() < mu[a] else 0
10         N[a] += 1
11         Q[a] += (r - Q[a]) / N[a]
12     return N, Q
```

DEĞİŞKENLER

`t = 1` `b = [inf, inf, inf]` `a = 0`

`N = [0, 0, 0]` `Q = [0.00, 0.00, 0.00]`

Kod İzleme: UCB1 (Üst Güven Sınırı)



Ödül **r = 1**; Q[0] = 1,00, N[0] = 1.


```
1  import math
2  def ucb(mu, T, rng):
3      K = len(mu)
4      N, Q = [0] * K, [0.0] * K
5      for t in range(1, T + 1):
6          b = [math.inf if n == 0 else
7               math.sqrt(2*math.log(t)/n) for n in N]
8          a = max(range(K), key=lambda i: Q[i] + b[i])
▶          r = 1 if rng.random() < mu[a] else 0
▶          N[a] += 1
▶          Q[a] += (r - Q[a]) / N[a]
12     return N, Q
```

DEĞİŞKENLER

`t = 1` `a = 0` `r = 1` `N = [1, 0, 0]` `Q = [1.00, 0.00, 0.00]`

Kod İzleme: UCB1 (Üst Güven Sınırı)



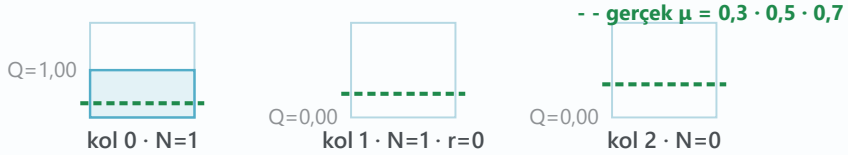
t = 2: hiç denenmemiş kolun payı ∞ → ilk denenmemiş kol **1** seçilir (her kol bir kez).


```
1  import math
2  def ucb(mu, T, rng):
3      K = len(mu)
4      N, Q = [0] * K, [0.0] * K
5      for t in range(1, T + 1):
▶          b = [math.inf if n == 0 else
▶               math.sqrt(2*math.log(t)/n) for n in N]
▶          a = max(range(K), key=lambda i: Q[i] + b[i])
9          r = 1 if rng.random() < mu[a] else 0
10         N[a] += 1
11         Q[a] += (r - Q[a]) / N[a]
12     return N, Q
```

DEĞİŞKENLER

`t = 2` `b = [1.18, inf, inf]` `a = 1` `N = [1, 0, 0]` `Q = [1.00, 0.00, 0.00]`

Kod İzleme: UCB1 (Üst Güven Sınırı)



Ödül **r = 0**; Q[1] = 0,00, N[1] = 1.

Adım 5 / 18

```
1  import math
2  def ucb(mu, T, rng):
3      K = len(mu)
4      N, Q = [0] * K, [0.0] * K
5      for t in range(1, T + 1):
6          b = [math.inf if n == 0 else
7               math.sqrt(2*math.log(t)/n) for n in N]
8          a = max(range(K), key=lambda i: Q[i] + b[i])
▶          r = 1 if rng.random() < mu[a] else 0
▶          N[a] += 1
▶          Q[a] += (r - Q[a]) / N[a]
12     return N, Q
```

DEĞİŞKENLER

`t = 2` `a = 1` `r = 0` `N = [1, 1, 0]`

`Q = [1.00, 0.00, 0.00]`

Kod İzleme: UCB1 (Üst Güven Sınırı)



t = 3: hiç denenmemiş kolun payı ∞ → ilk denenmemiş kol **2** seçilir (her kol bir kez).


```
1  import math
2  def ucb(mu, T, rng):
3      K = len(mu)
4      N, Q = [0] * K, [0.0] * K
5      for t in range(1, T + 1):
▶          b = [math.inf if n == 0 else
▶               math.sqrt(2*math.log(t)/n) for n in N]
▶          a = max(range(K), key=lambda i: Q[i] + b[i])
9          r = 1 if rng.random() < mu[a] else 0
10         N[a] += 1
11         Q[a] += (r - Q[a]) / N[a]
12     return N, Q
```

DEĞİŞKENLER

`t = 3` `b = [1.48, 1.48, inf]` `a = 2`

`N = [1, 1, 0]` `Q = [1.00, 0.00, 0.00]`

Kod İzleme: UCB1 (Üst Güven Sınırı)



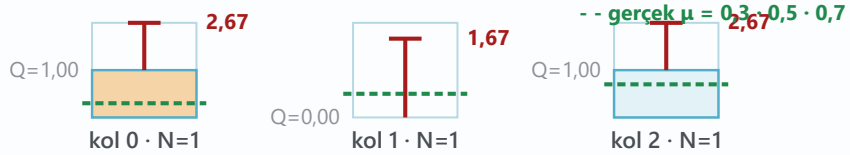
Ödül **r = 1**; Q[2] = 1,00, N[2] = 1.


```
1  import math
2  def ucb(mu, T, rng):
3      K = len(mu)
4      N, Q = [0] * K, [0.0] * K
5      for t in range(1, T + 1):
6          b = [math.inf if n == 0 else
7               math.sqrt(2*math.log(t)/n) for n in N]
8          a = max(range(K), key=lambda i: Q[i] + b[i])
▶          r = 1 if rng.random() < mu[a] else 0
▶          N[a] += 1
▶          Q[a] += (r - Q[a]) / N[a]
12     return N, Q
```

DEĞİŞKENLER

`t = 3` `a = 2` `r = 1` `N = [1, 1, 1]` `Q = [1.00, 0.00, 1.00]`

Kod İzleme: UCB1 (Üst Güven Sınırı)



t = 4: güven payı √(2 ln t / N) = [1,67, 1,67, 1,67]; UCB = Q + b = [2,67, 1,67, 2,67] → en büyük: kol **0**.


```
1  import math
2  def ucb(mu, T, rng):
3      K = len(mu)
4      N, Q = [0] * K, [0.0] * K
5      for t in range(1, T + 1):
▶          b = [math.inf if n == 0 else
▶               math.sqrt(2*math.log(t)/n) for n in N]
▶          a = max(range(K), key=lambda i: Q[i] + b[i])
9          r = 1 if rng.random() < mu[a] else 0
10         N[a] += 1
11         Q[a] += (r - Q[a]) / N[a]
12     return N, Q
```

DEĞİŞKENLER

`t = 4` `b = [1.67, 1.67, 1.67]` `a = 0`

`N = [1, 1, 1]` `Q = [1.00, 0.00, 1.00]`

Kod İzleme: UCB1 (Üst Güven Sınırı)



Ödül **r = 0**; Q[0] = 0,50, N[0] = 2.

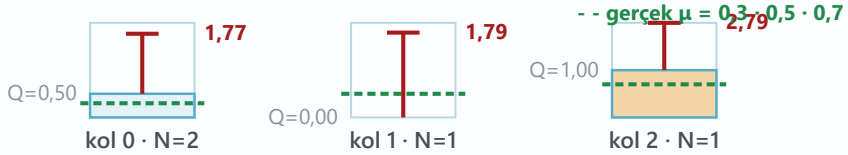
Adım 9 / 18

```
1  import math
2  def ucb(mu, T, rng):
3      K = len(mu)
4      N, Q = [0] * K, [0.0] * K
5      for t in range(1, T + 1):
6          b = [math.inf if n == 0 else
7               math.sqrt(2*math.log(t)/n) for n in N]
8          a = max(range(K), key=lambda i: Q[i] + b[i])
▶          r = 1 if rng.random() < mu[a] else 0
▶          N[a] += 1
▶          Q[a] += (r - Q[a]) / N[a]
12     return N, Q
```

DEĞİŞKENLER

`t = 4` `a = 0` `r = 0` `N = [2, 1, 1]` `Q = [0.50, 0.00, 1.00]`

Kod İzleme: UCB1 (Üst Güven Sınırı)



t = 5: güven payı √(2 ln t / N) = [1,27, 1,79, 1,79]; UCB = Q + b = [1,77, 1,79, 2,79] → en büyük: kol **2**.


```
1  import math
2  def ucb(mu, T, rng):
3      K = len(mu)
4      N, Q = [0] * K, [0.0] * K
5      for t in range(1, T + 1):
▶          b = [math.inf if n == 0 else
▶               math.sqrt(2*math.log(t)/n) for n in N]
▶          a = max(range(K), key=lambda i: Q[i] + b[i])
9          r = 1 if rng.random() < mu[a] else 0
10         N[a] += 1
11         Q[a] += (r - Q[a]) / N[a]
12     return N, Q
```

DEĞİŞKENLER

`t = 5` `b = [1.27, 1.79, 1.79]` `a = 2`

`N = [2, 1, 1]` `Q = [0.50, 0.00, 1.00]`

Kod İzleme: UCB1 (Üst Güven Sınırı)



Ödül **r = 1**; Q[2] = 1,00, N[2] = 2.

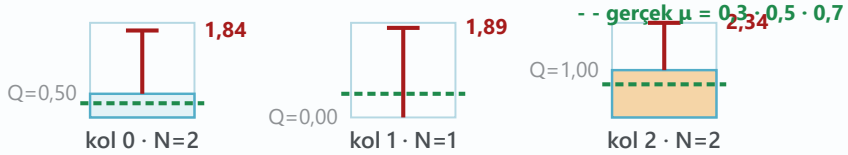

```
1  import math
2  def ucb(mu, T, rng):
3      K = len(mu)
4      N, Q = [0] * K, [0.0] * K
5      for t in range(1, T + 1):
6          b = [math.inf if n == 0 else
7               math.sqrt(2*math.log(t)/n) for n in N]
8          a = max(range(K), key=lambda i: Q[i] + b[i])
▶          r = 1 if rng.random() < mu[a] else 0
▶          N[a] += 1
▶          Q[a] += (r - Q[a]) / N[a]
12     return N, Q
```

DEĞİŞKENLER

`t = 5` `a = 2` `r = 1` `N = [2, 1, 2]`

`Q = [0.50, 0.00, 1.00]`

Kod İzleme: UCB1 (Üst Güven Sınırı)



t = 6: güven payı √(2 ln t / N) = [1,34, 1,89, 1,34]; UCB = Q + b = [1,84, 1,89, 2,34] → en büyük: kol **2**.


```
1  import math
2  def ucb(mu, T, rng):
3      K = len(mu)
4      N, Q = [0] * K, [0.0] * K
5      for t in range(1, T + 1):
▶          b = [math.inf if n == 0 else
▶               math.sqrt(2*math.log(t)/n) for n in N]
▶          a = max(range(K), key=lambda i: Q[i] + b[i])
9          r = 1 if rng.random() < mu[a] else 0
10         N[a] += 1
11         Q[a] += (r - Q[a]) / N[a]
12     return N, Q
```

DEĞİŞKENLER

`t = 6` `b = [1.34, 1.89, 1.34]` `a = 2`

`N = [2, 1, 2]` `Q = [0.50, 0.00, 1.00]`

Kod İzleme: UCB1 (Üst Güven Sınırı)



Ödül **r = 1**; Q[2] = 1,00, N[2] = 3.

Adım 13 / 18

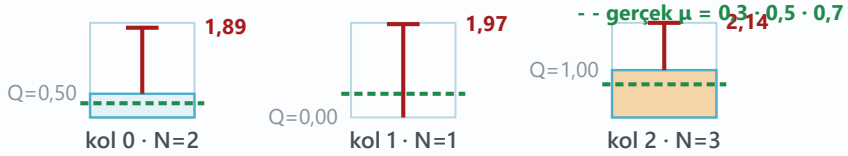
```
1  import math
2  def ucb(mu, T, rng):
3      K = len(mu)
4      N, Q = [0] * K, [0.0] * K
5      for t in range(1, T + 1):
6          b = [math.inf if n == 0 else
7               math.sqrt(2*math.log(t)/n) for n in N]
8          a = max(range(K), key=lambda i: Q[i] + b[i])
▶          r = 1 if rng.random() < mu[a] else 0
▶          N[a] += 1
▶          Q[a] += (r - Q[a]) / N[a]
12     return N, Q
```

DEĞİŞKENLER

`t = 6` `a = 2` `r = 1` `N = [2, 1, 3]`

`Q = [0.50, 0.00, 1.00]`

Kod İzleme: UCB1 (Üst Güven Sınırı)



t = 7: güven payı √(2 ln t / N) = [1,39, 1,97, 1,14]; UCB = Q + b = [1,89, 1,97, 2,14] → en büyük: kol **2**.


```
1  import math
2  def ucb(mu, T, rng):
3      K = len(mu)
4      N, Q = [0] * K, [0.0] * K
5      for t in range(1, T + 1):
▶          b = [math.inf if n == 0 else
▶               math.sqrt(2*math.log(t)/n) for n in N]
▶          a = max(range(K), key=lambda i: Q[i] + b[i])
9          r = 1 if rng.random() < mu[a] else 0
10         N[a] += 1
11         Q[a] += (r - Q[a]) / N[a]
12     return N, Q
```

DEĞİŞKENLER

`t = 7` `b = [1.39, 1.97, 1.14]` `a = 2`

`N = [2, 1, 3]` `Q = [0.50, 0.00, 1.00]`

Kod İzleme: UCB1 (Üst Güven Sınırı)



Ödül **r = 1**; Q[2] = 1,00, N[2] = 4.

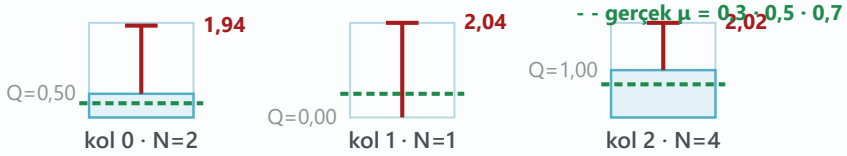

```
1  import math
2  def ucb(mu, T, rng):
3      K = len(mu)
4      N, Q = [0] * K, [0.0] * K
5      for t in range(1, T + 1):
6          b = [math.inf if n == 0 else
7               math.sqrt(2*math.log(t)/n) for n in N]
8          a = max(range(K), key=lambda i: Q[i] + b[i])
▶          r = 1 if rng.random() < mu[a] else 0
▶          N[a] += 1
▶          Q[a] += (r - Q[a]) / N[a]
12     return N, Q
```

DEĞİŞKENLER

`t = 7` `a = 2` `r = 1` `N = [2, 1, 4]`

`Q = [0.50, 0.00, 1.00]`

Kod İzleme: UCB1 (Üst Güven Sınırı)



t = 8: güven payı √(2 ln t / N) = [1,44, 2,04, 1,02]; UCB = Q + b = [1,94, 2,04, 2,02] → en büyük: kol **1**.

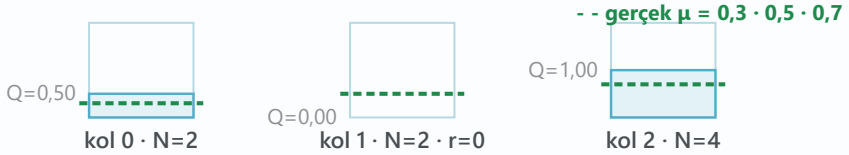

```
1  import math
2  def ucb(mu, T, rng):
3      K = len(mu)
4      N, Q = [0] * K, [0.0] * K
5      for t in range(1, T + 1):
▶          b = [math.inf if n == 0 else
▶               math.sqrt(2*math.log(t)/n) for n in N]
▶          a = max(range(K), key=lambda i: Q[i] + b[i])
9          r = 1 if rng.random() < mu[a] else 0
10         N[a] += 1
11         Q[a] += (r - Q[a]) / N[a]
12     return N, Q
```

DEĞİŞKENLER

`t = 8` `b = [1.44, 2.04, 1.02]` `a = 1`

`N = [2, 1, 4]` `Q = [0.50, 0.00, 1.00]`

Kod İzleme: UCB1 (Üst Güven Sınırı)



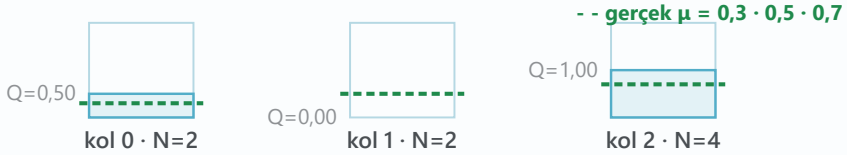
Ödül **r = 0**; Q[1] = 0,00, N[1] = 2.


```
1  import math
2  def ucb(mu, T, rng):
3      K = len(mu)
4      N, Q = [0] * K, [0.0] * K
5      for t in range(1, T + 1):
6          b = [math.inf if n == 0 else
7               math.sqrt(2*math.log(t)/n) for n in N]
8          a = max(range(K), key=lambda i: Q[i] + b[i])
▶          r = 1 if rng.random() < mu[a] else 0
▶          N[a] += 1
▶          Q[a] += (r - Q[a]) / N[a]
12     return N, Q
```

DEĞİŞKENLER

`t = 8` `a = 1` `r = 0` `N = [2, 2, 4]` `Q = [0.50, 0.00, 1.00]`

Kod İzleme: UCB1 (Üst Güven Sınırı)



return N = [2, 2, 4]. Az denenen kolun payı büyük kalır: t = 8'de kol 1 (tek deneme, Q = 0) yeniden denendi — keşif kendiliğinden.

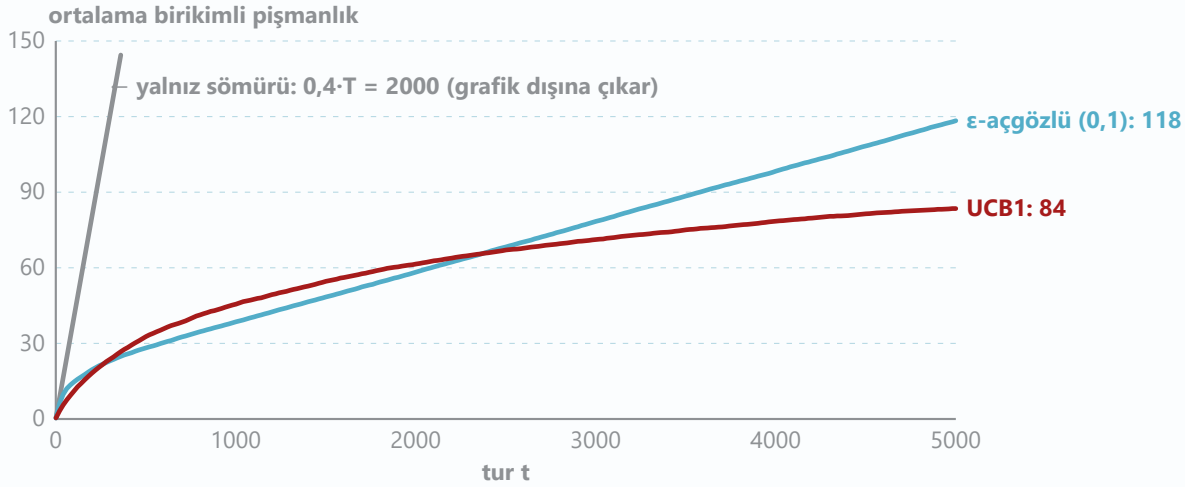
Adım 18 / 18

```
1  import math
2  def ucb(mu, T, rng):
3      K = len(mu)
4      N, Q = [0] * K, [0.0] * K
5      for t in range(1, T + 1):
6          b = [math.inf if n == 0 else
7               math.sqrt(2*math.log(t)/n) for n in N]
8          a = max(range(K), key=lambda i: Q[i] + b[i])
9          r = 1 if rng.random() < mu[a] else 0
10         N[a] += 1
11         Q[a] += (r - Q[a]) / N[a]
▶      return N, Q
```

DEĞİŞKENLER

`N = [2, 2, 4]` `Q = [0.50, 0.00, 1.00]`

Karşılaştırma: 5000 Tur, 100 Tekrar



- Kollar μ = (0,3; 0,5; 0,7), tohumlar 0–99. Yalnız sömürü her seferinde kol 0'a takılır: pişmanlık 0,4·T, **doğrusal**.
- ε-açgözlü hep %10 keşif yaptığı için doğrusal büyür. UCB1'in pişmanlığı yaklaşık 270. ile 2.370. tur arasında ε-açgözlüden yüksektir; sonra daha düşük kalır, çünkü UCB1'in pişmanlığı **logaritmik**, yani çok yavaş büyür (Auer vd., 2002).
- ε bir ayar parametresidir: kısa izlemede keşif görünsün diye 0,2, uzun karşılaştırmada 0,1, Çok Kollu Haydut oyununda 0,15 seçildi. Küçük ε daha az keşif, daha geç düzeltme demektir.

Oyuna Dönüş: Çok Kollu Haydut

Kuramın cevabı: keşfin bir bedeli var

Kollar μ = (0,3; 0,5; 0,7) iken 100 turda beklenen ödül ve pişmanlık:

Strateji	Ödül	Pişmanlık
hep rastgele (yalnız keşif)	50	20
0,5'lik kola bağlanmak	50	20
ε-açgözlü, ε = 0,15 (en iyiyi bulduktan sonra)	67	3
hep en iyi kol	70	0

ε-açgözlü her tur 0,15 olasılıkla keşfe devam eder; UCB1 keşfi az denenen kollara yöneltir, pişmanlığı logaritmik büyür.

Açılıştaki soru

Kol 3'ün tek başarısız çekişi çok az bilgi verir. UCB1 burada Kol 3'ü seçer (sıradaki tur t = 9): puanı $0 + \sqrt{(2 \ln 9 / 1)} \approx 2{,}10$, Kol 1'inki $0{,}67 + \sqrt{(2 \ln 9 / 3)} \approx 1{,}88$.

Gerçekte: Dört kollu bir haydut deneyinde insan deneklerin seçimleri, belirsizliğe bonus veren (UCB benzeri) keşiften çok, değer farkına göre olasılıklı seçimle (softmax seçimi) açıklandı (Daw vd., 2006).

Tek bir deneyle algoritmaları sıralamak yanıltıcıdır: ödüller rastgeledir, karşılaştırma birçok deneyin ortalamasıyla yapılır.

Hayduttan Pekiştirmeli Öğrenmeye

- Haydut = tek durumlu problem. **Pekiştirmeli öğrenmede** eylem, ödülün yanında bir sonraki **durumu** (ajanın bulunduğu konum, ör. tahtadaki dizilim) da belirler.
- **Q-öğrenme** (Watkins, 1989): durum-eylem değerini gözlenen geçişle günceller

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \cdot \max_{a'} Q(s', a') - Q(s, a) \right]$$

α: öğrenme oranı (yeni bilgiye ne kadar güvenilir); γ: gelecekteki ödülün bugünkü değeri (iskonto).

Sayısal örnek

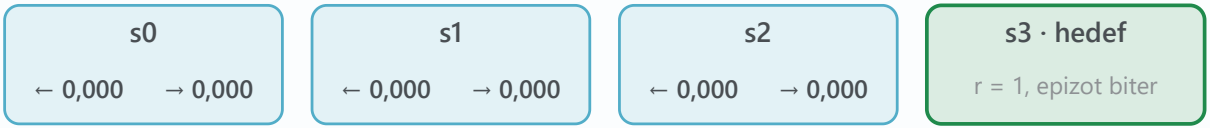
Q(s, a) = 2, r = 1, γ = 0,9, max Q(s′, ·) = 4, α = 0,5:

`Q ← 2 + 0,5·(1 + 3,6 − 2) = 3,3`

Çoklu ajan ortamında

Diğer ajanlar da öğrendiği için oyunun kuralları ajanın gözünde sürekli değişir (ortam durağan değildir); tek ajanlı yakınsama garantileri geçerliliğini yitirir. Kendine karşı oyun (AlphaZero) bir çözüm yoludur.

Kod İzleme: Q-Öğrenme (α = 0,5, γ = 0,9)



Çağrı `q_learn(ep, 0.5, 0.9)` : s0'dan başlayan 3 epizot. Tüm Q değerleri 0; s3 hedef (+1).

Adım 1 / 13

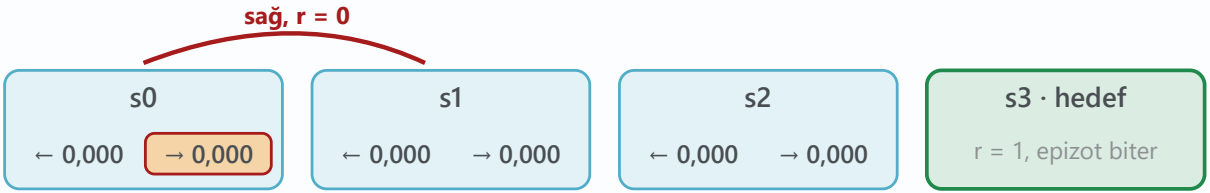
```
1 def q_learn(episodes, alpha, gamma):
▶       Q = [[0.0, 0.0] for s in range(4)]  # [sol, sağ]
3       for ep in episodes:
4           for s, a, r, s2 in ep:
5               best = 0 if s2 == 3 else max(Q[s2])
6               hedef = r + gamma * best
7               Q[s][a] += alpha * (hedef - Q[s][a])
8       return Q
```

DEĞİŞKENLER

`alpha = 0.5` `gamma = 0.9`

`Q = hepsi 0`

Kod İzleme: Q-Öğrenme (α = 0,5, γ = 0,9)



Epizot 1: s0 → sağ → s1, r = 0. best = max Q(s1, ·) = 0,000; hedef = 0,000; Q(s0,sağ) = 0,000 + 0,5·(0,000 − 0,000) = **0,000**.

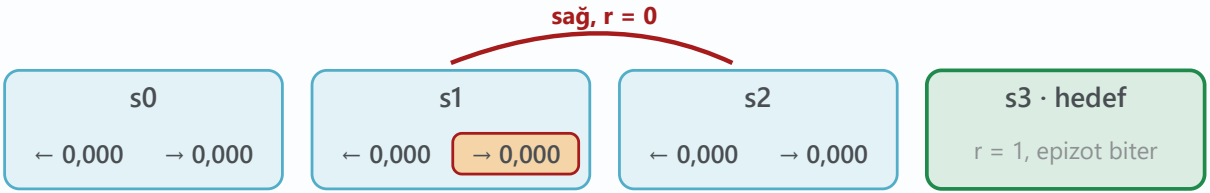
Adım 2 / 13

```
1 def q_learn(episodes, alpha, gamma):
2     Q = [[0.0, 0.0] for s in range(4)]  # [sol, sağ]
3     for ep in episodes:
▶         for s, a, r, s2 in ep:
▶             best = 0 if s2 == 3 else max(Q[s2])
▶             hedef = r + gamma * best
▶             Q[s][a] += alpha * (hedef - Q[s][a])
8     return Q
```

DEĞİŞKENLER

`ep = 1` `s = 0` `a = 1 (sağ)`
`r = 0` `s2 = 1` `best = 0.0`
`hedef = 0.0` `Q[s][a] = 0.0`

Kod İzleme: Q-Öğrenme (α = 0,5, γ = 0,9)



Epizot 1: s1 → sağ → s2, r = 0. best = max Q(s2, ·) = 0,000; hedef = 0,000; Q(s1,sağ) = 0,000 + 0,5·(0,000 − 0,000) = **0,000**.

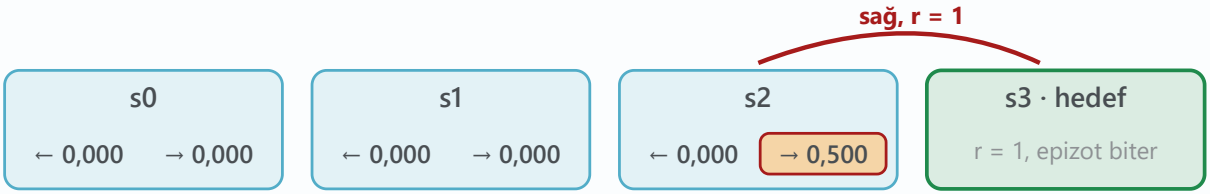
Adım 3 / 13

```
1 def q_learn(episodes, alpha, gamma):
2     Q = [[0.0, 0.0] for s in range(4)]  # [sol, sağ]
3     for ep in episodes:
▶         for s, a, r, s2 in ep:
▶             best = 0 if s2 == 3 else max(Q[s2])
▶             hedef = r + gamma * best
▶             Q[s][a] += alpha * (hedef - Q[s][a])
8     return Q
```

DEĞİŞKENLER

`ep = 1` `s = 1` `a = 1 (sağ)`
`r = 0` `s2 = 2` `best = 0.0`
`hedef = 0.0` `Q[s][a] = 0.0`

Kod İzleme: Q-Öğrenme (α = 0,5, γ = 0,9)



Epizot 1: s2 → sağ → s3, r = 1. s3 hedef → best = 0; hedef = 1,000; Q(s2,sağ) = 0,000 + 0,5·(1,000 − 0,000) = **0,500**.

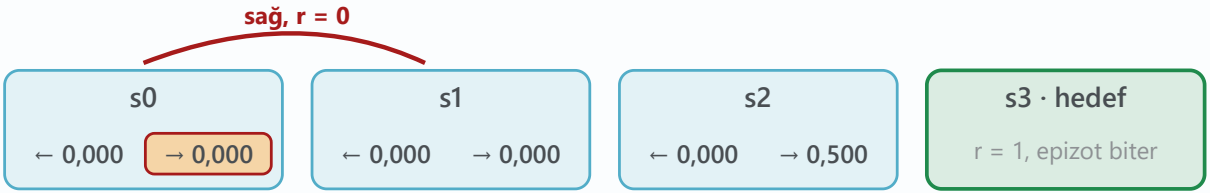
Adım 4 / 13

```
1 def q_learn(episodes, alpha, gamma):
2     Q = [[0.0, 0.0] for s in range(4)]  # [sol, sağ]
3     for ep in episodes:
▶         for s, a, r, s2 in ep:
▶             best = 0 if s2 == 3 else max(Q[s2])
▶             hedef = r + gamma * best
▶             Q[s][a] += alpha * (hedef - Q[s][a])
8     return Q
```

DEĞİŞKENLER

`ep = 1` `s = 2` `a = 1 (sağ)`
`r = 1` `s2 = 3` `best = 0`
`hedef = 1.0` `Q[s][a] = 0.5`

Kod İzleme: Q-Öğrenme (α = 0,5, γ = 0,9)



Epizot 2: s0 → sağ → s1, r = 0. best = max Q(s1, ·) = 0,000; hedef = 0,000; Q(s0,sağ) = 0,000 + 0,5·(0,000 − 0,000) = **0,000**.

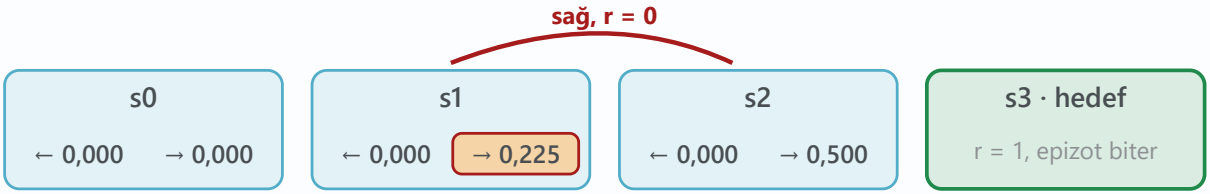
Adım 5 / 13

```
1 def q_learn(episodes, alpha, gamma):
2     Q = [[0.0, 0.0] for s in range(4)]  # [sol, sağ]
3     for ep in episodes:
          for s, a, r, s2 in ep:
              best = 0 if s2 == 3 else max(Q[s2])
              hedef = r + gamma * best
              Q[s][a] += alpha * (hedef - Q[s][a])
8     return Q
```

DEĞİŞKENLER

`ep = 2` `s = 0` `a = 1 (sağ)` `r = 0` `s2 = 1` `best = 0.0` `hedef = 0.0` `Q[s][a] = 0.0`

Kod İzleme: Q-Öğrenme (α = 0,5, γ = 0,9)



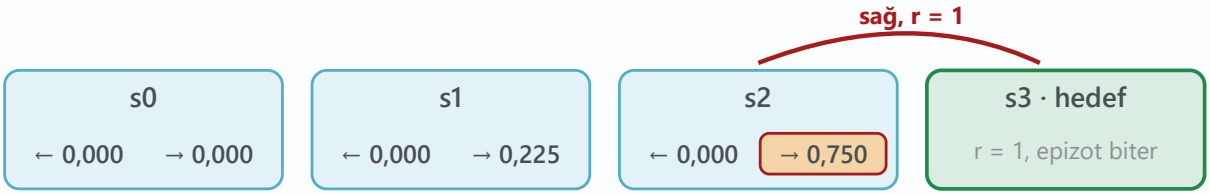
Epizot 2: s1 → sağ → s2, r = 0. best = max Q(s2, ·) = 0,500; hedef = 0,450; Q(s1,sağ) = 0,000 + 0,5·(0,450 − 0,000) = **0,225**.


```
1 def q_learn(episodes, alpha, gamma):
2     Q = [[0.0, 0.0] for s in range(4)]  # [sol, sağ]
3     for ep in episodes:
          for s, a, r, s2 in ep:
              best = 0 if s2 == 3 else max(Q[s2])
              hedef = r + gamma * best
              Q[s][a] += alpha * (hedef - Q[s][a])
8     return Q
```

DEĞİŞKENLER

`ep = 2` `s = 1` `a = 1 (sağ)`
`r = 0` `s2 = 2` `best = 0.5`
`hedef = 0.45` `Q[s][a] = 0.225`

Kod İzleme: Q-Öğrenme (α = 0,5, γ = 0,9)



Epizot 2: s2 → sağ → s3, r = 1. s3 hedef → best = 0; hedef = 1,000; Q(s2,sağ) = 0,500 + 0,5·(1,000 − 0,500) = **0,750**.

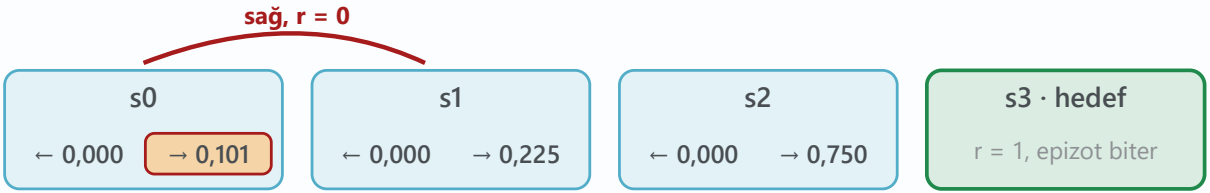
Adım 7 / 13

```
1 def q_learn(episodes, alpha, gamma):
2     Q = [[0.0, 0.0] for s in range(4)]  # [sol, sağ]
3     for ep in episodes:
          for s, a, r, s2 in ep:
              best = 0 if s2 == 3 else max(Q[s2])
              hedef = r + gamma * best
              Q[s][a] += alpha * (hedef - Q[s][a])
8     return Q
```

DEĞİŞKENLER

`ep = 2` `s = 2` `a = 1 (sağ)`
`r = 1` `s2 = 3` `best = 0`
`hedef = 1.0` `Q[s][a] = 0.75`

Kod İzleme: Q-Öğrenme (α = 0,5, γ = 0,9)



Epizot 3: s0 → sağ → s1, r = 0. best = max Q(s1, ·) = 0,225; hedef = 0,203; Q(s0,sağ) = 0,000 + 0,5·(0,203 − 0,000) = **0,101**.

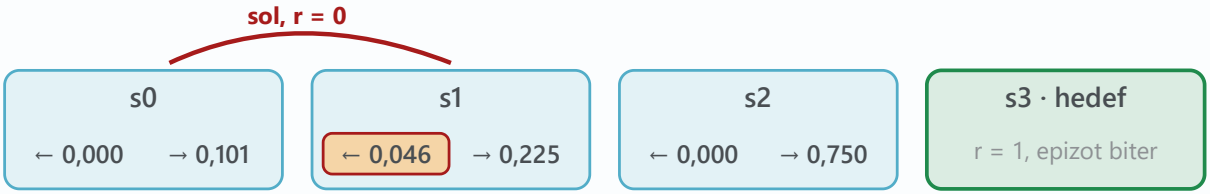
Adım 8 / 13

```
1 def q_learn(episodes, alpha, gamma):
2     Q = [[0.0, 0.0] for s in range(4)]  # [sol, sağ]
3     for ep in episodes:
          for s, a, r, s2 in ep:
              best = 0 if s2 == 3 else max(Q[s2])
              hedef = r + gamma * best
              Q[s][a] += alpha * (hedef - Q[s][a])
8     return Q
```

DEĞİŞKENLER

`ep = 3` `s = 0` `a = 1 (sağ)` `r = 0` `s2 = 1` `best = 0.225` `hedef = 0.2025` `Q[s][a] = 0.1013`

Kod İzleme: Q-Öğrenme (α = 0,5, γ = 0,9)



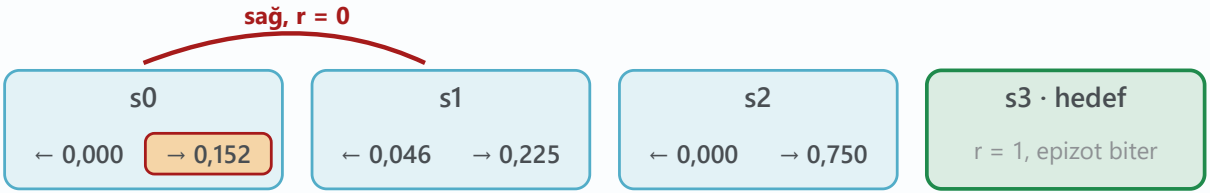
Epizot 3: s1 → sol → s0, r = 0. best = max Q(s0, ·) = 0,101; hedef = 0,091; Q(s1,sol) = 0,000 + 0,5·(0,091 − 0,000) = **0,046**.


```
1 def q_learn(episodes, alpha, gamma):
2     Q = [[0.0, 0.0] for s in range(4)]  # [sol, sağ]
3     for ep in episodes:
▶         for s, a, r, s2 in ep:
▶             best = 0 if s2 == 3 else max(Q[s2])
▶             hedef = r + gamma * best
▶             Q[s][a] += alpha * (hedef - Q[s][a])
8     return Q
```

DEĞİŞKENLER

`ep = 3` `s = 1` `a = 0 (sol)`
`r = 0` `s2 = 0` `best = 0.1013`
`hedef = 0.0911`
`Q[s][a] = 0.0456`

Kod İzleme: Q-Öğrenme (α = 0,5, γ = 0,9)



Epizot 3: s0 → sağ → s1, r = 0. best = max Q(s1, ·) = 0,225; hedef = 0,203; Q(s0,sağ) = 0,101 + 0,5·(0,203 − 0,101) = **0,152**.

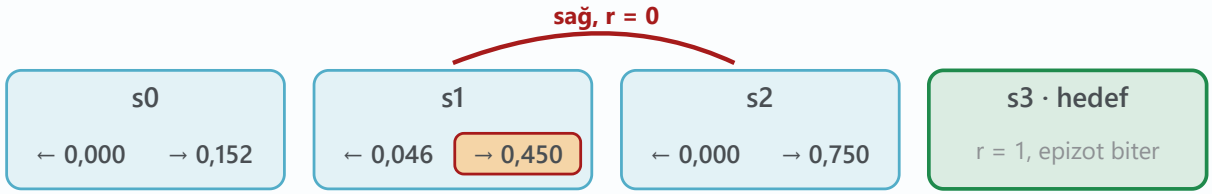
Adım 10 / 13

```
1 def q_learn(episodes, alpha, gamma):
2     Q = [[0.0, 0.0] for s in range(4)]  # [sol, sağ]
3     for ep in episodes:
▶           for s, a, r, s2 in ep:
▶               best = 0 if s2 == 3 else max(Q[s2])
▶               hedef = r + gamma * best
▶               Q[s][a] += alpha * (hedef - Q[s][a])
8     return Q
```

DEĞİŞKENLER

`ep = 3` `s = 0` `a = 1 (sağ)`
`r = 0` `s2 = 1` `best = 0.225`
`hedef = 0.2025`
`Q[s][a] = 0.1519`

Kod İzleme: Q-Öğrenme (α = 0,5, γ = 0,9)



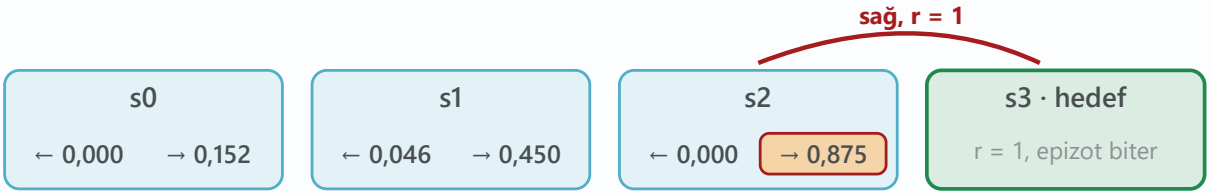
Epizot 3: s1 → sağ → s2, r = 0. best = max Q(s2, ·) = 0,750; hedef = 0,675; Q(s1,sağ) = 0,225 + 0,5·(0,675 − 0,225) = **0,450**.


```
1 def q_learn(episodes, alpha, gamma):
2     Q = [[0.0, 0.0] for s in range(4)]  # [sol, sağ]
3     for ep in episodes:
          for s, a, r, s2 in ep:
              best = 0 if s2 == 3 else max(Q[s2])
              hedef = r + gamma * best
              Q[s][a] += alpha * (hedef - Q[s][a])
8     return Q
```

DEĞİŞKENLER

`ep = 3` `s = 1` `a = 1 (sağ)`
`r = 0` `s2 = 2` `best = 0.75`
`hedef = 0.675` `Q[s][a] = 0.45`

Kod İzleme: Q-Öğrenme (α = 0,5, γ = 0,9)



Epizot 3: s2 → sağ → s3, r = 1. s3 hedef → best = 0; hedef = 1,000; Q(s2,sağ) = 0,750 + 0,5·(1,000 − 0,750) = **0,875**.

Adım 12 / 13

```
1 def q_learn(episodes, alpha, gamma):
2     Q = [[0.0, 0.0] for s in range(4)]  # [sol, sağ]
3     for ep in episodes:
▶         for s, a, r, s2 in ep:
▶             best = 0 if s2 == 3 else max(Q[s2])
▶             hedef = r + gamma * best
▶             Q[s][a] += alpha * (hedef - Q[s][a])
8     return Q
```

DEĞİŞKENLER

`ep = 3` `s = 2` `a = 1 (sağ)`
`r = 1` `s2 = 3` `best = 0`
`hedef = 1.0` `Q[s][a] = 0.875`

Kod İzleme: Q-Öğrenme (α = 0,5, γ = 0,9)

s0	s1	s2	s3 · hedef
← 0,000 → 0,152	← 0,046 → 0,450	← 0,000 → 0,875	r = 1, epizot biter

return Q. Ödül hedeften geriye yayıldı: Q(s2,sağ) = 0,875 → Q(s1,sağ) = 0,450 → Q(s0,sağ) = 0,152. Her hücrede Q(sağ) > Q(sol): açgözlü politika "hep sağa".

Adım 13 / 13

```
1 def q_learn(episodes, alpha, gamma):
2     Q = [[0.0, 0.0] for s in range(4)]  # [sol, sağ]
3     for ep in episodes:
4         for s, a, r, s2 in ep:
5             best = 0 if s2 == 3 else max(Q[s2])
6             hedef = r + gamma * best
7             Q[s][a] += alpha * (hedef - Q[s][a])
▶     return Q
```

DEĞİŞKENLER

- `Q[0] = [0.0, 0.152]`
- `Q[1] = [0.046, 0.45]`
- `Q[2] = [0.0, 0.875]`

Q-öğrenmenin Ötesi: Yöntemler ve Kilometre Taşları

- **Thompson örneklemesi** (1933): her kol için "ortalaması şu olabilir" dağılımı (son olasılık dağılımı) tutar, her tur bu dağılımdan bir tahmin çeker ve en büyük tahminin kolunu çeker.
- **Kazan-kal, kaybet-değiş** (Win-Stay Lose-Shift, WSLS; Nowak–Sigmund, 1993): 3 ya da 5 aldıysan aynı hamle, 0 ya da 1 aldıysan değiştir; Mahkûmun İkileminde tek hatadan iki turda işbirliğine döner.
- **Diğerleri:** EXP3 (yalnız çekilen kolun ödülünü gören Hedge; ödülleri bir rakip seçse bile pişmanlığı T'den yavaş büyür), SARSA ve DQN (Q tablosu yerine sinir ağı), CFR (pişmanlığa dayalı poker yöntemi).

Yıl	Sistem	Yöntem
1992	TD-Gammon (tavla)	kendine karşı oyun
2017–19	Libratus, Pluribus (poker)	CFR (pişmanlık)
2018	AlphaZero (satranç, Go)	kendine karşı oyun

Oyunlar RL için iyi bir laboratuvardır: kurallar açık, ödül net, oyun milyonlarca kez tekrarlanabilir; pokerde pişmanlık tabanlı yöntemler öne çıktı.

EVRİMSEL DİNAMİK

Replikatör Dinamiği

- Büyük bir nüfusta bireyler rastgele eşleşir; her birey tek bir strateji oynar ve onu yavrularına aktarır. Rasyonellik yok, **seçilim** var.
- f_i: i stratejisinin ortalama kazancı (**uygunluk**); $\bar{f}$: nüfusun ortalama kazancı. Ortalamanın üstünde kazanan strateji yayılır.
- Bu derste kesikli biçim yeterli: **yeni pay = eski pay × f_i / $\bar{f}$** (uygunluklar pozitif olsun diye her kazanca taban 3 eklenir).
- **Evrimsel kararlı strateji (ESS)**: küçük bir mutant grubu istila edemez (Maynard Smith–Price, 1973).

Birey \ Rakip	Şahin	Güvercin
Şahin	(−1 , −1)	(4 , 0)
Güvercin	(0 , 4)	(2 , 2)

Şahin–Güvercin: kaynak V = 4, kavga maliyeti C = 6. İki şahin kavga eder: (V − C)/2 = −1; şahin güvercine karşı V = 4, güvercin 0 alır; iki güvercin paylaşır: V/2 = 2.

Replikatör Dinamiği Adım Adım: Şahin–Güvercin

nesil	x (şahin payı)	$f_Ş$	f_G	$\bar{f}$	yeni x = $x \cdot f_Ş / \bar{f}$
0	0,100	6,50	4,80	4,97	0,131
1					
2					
3					
4					
5					
6					
7					

x = 0,100: $f_Ş = 3 + (-1)\cdot x + 4\cdot(1 - x) = 6{,}50$, $f_G = 3 + 0\cdot x + 2\cdot(1 - x) = 4{,}80$. $f_Ş > \bar{f}$ → şahinler artar.

Replikatör Dinamiği Adım Adım: Şahin–Güvercin

nesil	x (şahin payı)	$f_Ş$	f_G	$\bar{f}$	yeni x = $x \cdot f_Ş / \bar{f}$
0	0,100	6,50	4,80	4,97	0,131
1	0,131	6,35	4,74	4,95	0,168
2					
3					
4					
5					
6					
7					

$x = 0{,}131$: $f_Ş = 3 + (-1)\cdot x + 4\cdot(1 - x) = 6{,}35$, $f_G = 3 + 0\cdot x + 2\cdot(1 - x) = 4{,}74$. $f_Ş > \bar{f}$ → şahinler artar.

Replikatör Dinamiği Adım Adım: Şahin–Güvercin

nesil	x (şahin payı)	$f_Ş$	f_G	$\bar{f}$	yeni x = $x \cdot f_Ş / \bar{f}$
0	0,100	6,50	4,80	4,97	0,131
1	0,131	6,35	4,74	4,95	0,168
2	0,168	6,16	4,66	4,92	0,210
3					
4					
5					
6					
7					

x = 0,168: $f_Ş = 3 + (-1)\cdot x + 4\cdot(1 - x) = 6{,}16$, $f_G = 3 + 0\cdot x + 2\cdot(1 - x) = 4{,}66$. $f_Ş > \bar{f}$ → şahinler artar.

Replikatör Dinamiği Adım Adım: Şahin–Güvercin

nesil	x (şahin payı)	$f_Ş$	f_G	$\bar{f}$	yeni x = $x \cdot f_Ş / \bar{f}$
0	0,100	6,50	4,80	4,97	0,131
1	0,131	6,35	4,74	4,95	0,168
2	0,168	6,16	4,66	4,92	0,210
3	0,210	5,95	4,58	4,87	0,257
4					
5					
6					
7					

$x = 0{,}210$: $f_Ş = 3 + (-1)\cdot x + 4\cdot(1 - x) = 5{,}95$, $f_G = 3 + 0\cdot x + 2\cdot(1 - x) = 4{,}58$. $f_Ş > \bar{f}$ → şahinler artar.

Replikatör Dinamiği Adım Adım: Şahin–Güvercin

nesil	x (şahin payı)	$f_Ş$	f_G	$\bar{f}$	yeni x = $x \cdot f_Ş / \bar{f}$
0	0,100	6,50	4,80	4,97	0,131
1	0,131	6,35	4,74	4,95	0,168
2	0,168	6,16	4,66	4,92	0,210
3	0,210	5,95	4,58	4,87	0,257
4	0,257	5,72	4,49	4,80	0,306
5					
6					
7					

x = 0,257: $f_Ş = 3 + (-1)\cdot x + 4\cdot(1 - x) = 5{,}72$, $f_G = 3 + 0\cdot x + 2\cdot(1 - x) = 4{,}49$. $f_Ş > \bar{f}$ → şahinler artar.

Replikatör Dinamiği Adım Adım: Şahin–Güvercin

nesil	x (şahin payı)	$f_Ş$	f_G	$\bar{f}$	yeni x = $x \cdot f_Ş / \bar{f}$
0	0,100	6,50	4,80	4,97	0,131
1	0,131	6,35	4,74	4,95	0,168
2	0,168	6,16	4,66	4,92	0,210
3	0,210	5,95	4,58	4,87	0,257
4	0,257	5,72	4,49	4,80	0,306
5	0,306	5,47	4,39	4,72	0,354
6					
7					

x = 0,306: $f_Ş = 3 + (-1)\cdot x + 4\cdot(1 - x) = 5{,}47$, $f_G = 3 + 0\cdot x + 2\cdot(1 - x) = 4{,}39$. $f_Ş > \bar{f}$ → şahinler artar.

Replikatör Dinamiği Adım Adım: Şahin–Güvercin

nesil	x (şahin payı)	$f_Ş$	f_G	$\bar{f}$	yeni x = $x \cdot f_Ş / \bar{f}$
0	0,100	6,50	4,80	4,97	0,131
1	0,131	6,35	4,74	4,95	0,168
2	0,168	6,16	4,66	4,92	0,210
3	0,210	5,95	4,58	4,87	0,257
4	0,257	5,72	4,49	4,80	0,306
5	0,306	5,47	4,39	4,72	0,354
6	0,354	5,23	4,29	4,62	0,401
7					

$x = 0{,}354$: $f_Ş = 3 + (-1)\cdot x + 4\cdot(1 - x) = 5{,}23$, $f_G = 3 + 0\cdot x + 2\cdot(1 - x) = 4{,}29$. $f_Ş > \bar{f}$ → şahinler artar.

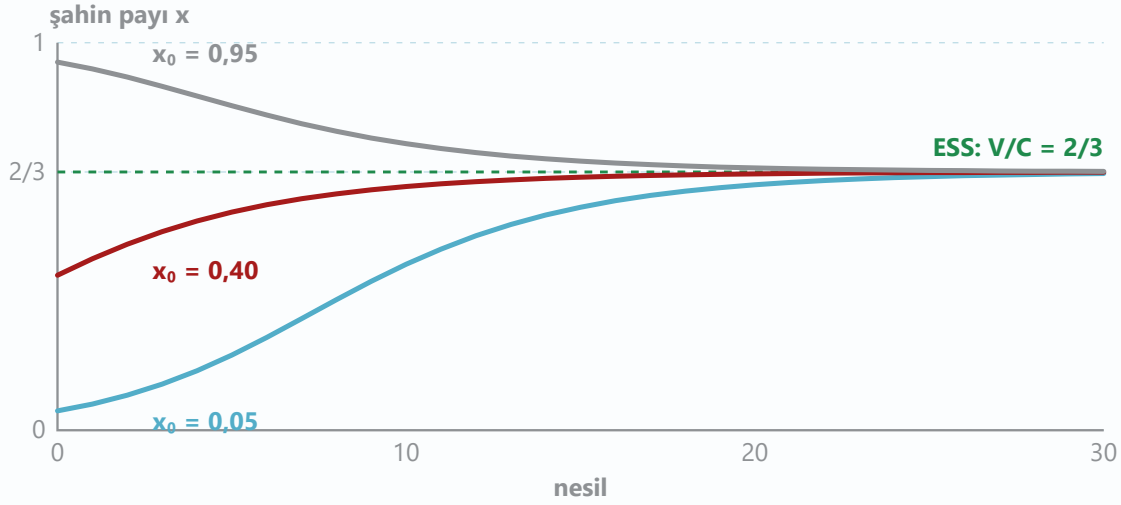
Replikatör Dinamiği Adım Adım: Şahin–Güvercin

nesil	x (şahin payı)	$f_Ş$	f_G	$\bar{f}$	yeni x = $x \cdot f_Ş / \bar{f}$
0	0,100	6,50	4,80	4,97	0,131
1	0,131	6,35	4,74	4,95	0,168
2	0,168	6,16	4,66	4,92	0,210
3	0,210	5,95	4,58	4,87	0,257
4	0,257	5,72	4,49	4,80	0,306
5	0,306	5,47	4,39	4,72	0,354
6	0,354	5,23	4,29	4,62	0,401
7	0,401	5,00	4,20	4,52	0,443

$x = 0{,}401$: $f_Ş = 3 + (-1)\cdot x + 4\cdot(1 - x) = 5{,}00$, $f_G = 3 + 0\cdot x + 2\cdot(1 - x) = 4{,}20$. $f_Ş > \bar{f}$ → şahinler artar.

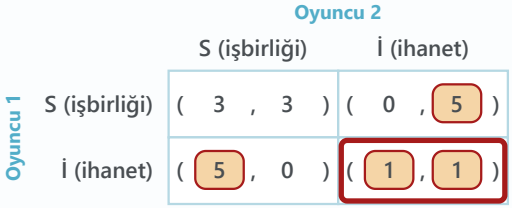
EVRİMSEL DİNAMİK

Her Başlangıçtan ESS'ye: x* = V/C = 2/3



- Dinlenme noktası $f_Ş = f_G$: $7 - 5x = 5 - 2x \Rightarrow$ **x* = 2/3** — oyunun simetrik karışık Nash dengesiyle aynı.
- Taş-Kağıt-Makas'ta replikatör dinamiği dengenin etrafında **döner**; Geyik Avı'nda başlangıca göre iki saf dengeden birine gider.

Tekrarlanan Mahkûmun İkilemi



Puan (yüksek iyi): cazibe 5 > karşılıklı işbirliği 3 > karşılıklı ihanet 1 > enayi 0. S: sessiz kalmak.

- Tek seferlik oyunda ve sonlu, bilinen sayıda tekrarda (geriye doğru çıkarımla) her turda ihanet.
- Oyun her turdan sonra δ olasılıkla sürüyorsa (**geleceğin gölgesi**) kinci tetik stratejisi (Grim) işbirliğini koruyabilir:

Uymak: her tur karşılıklı işbirliği, $3 \Rightarrow 3 + 3\delta + \ldots = \mathbf{3/(1 - \delta)}$.

Tekrarlanan Mahkûmun İkilemi

Oyuncu 1 \ Oyuncu 2	S (işbirliği)	İ (ihanet)
S (işbirliği)	(3, 3)	(0, **5**)
İ (ihanet)	(**5**, 0)	(**1**, **1**)

Puan (yüksek iyi): cazibe 5 > karşılıklı işbirliği 3 > karşılıklı ihanet 1 > enayi 0. S: sessiz kalmak.

- Tek seferlik oyunda ve sonlu, bilinen sayıda tekrarda (geriye doğru çıkarımla) her turda ihanet.
- Oyun her turdan sonra δ olasılıkla sürüyorsa (**geleceğin gölgesi**) kinci tetik stratejisi (Grim) işbirliğini koruyabilir:

Uymak: her tur karşılıklı işbirliği, $3 \Rightarrow 3 + 3\delta + \ldots = \mathbf{3/(1-\delta)}$.

Sapmak: bu tur cazibe 5, sonra hep karşılıklı ihanet $1 \Rightarrow \mathbf{5 + \delta/(1-\delta)}$.

Tekrarlanan Mahkûmun İkilemi

Oyuncu 1 \ Oyuncu 2	S (işbirliği)	İ (ihanet)
S (işbirliği)	(3, 3)	(0, **5**)
İ (ihanet)	(**5**, 0)	(**1**, **1**)

Puan (yüksek iyi): cazibe 5 > karşılıklı işbirliği 3 > karşılıklı ihanet 1 > enayi 0. S: sessiz kalmak.

- Tek seferlik oyunda ve sonlu, bilinen sayıda tekrarda (geriye doğru çıkarımla) her turda ihanet.
- Oyun her turdan sonra δ olasılıkla sürüyorsa (**geleceğin gölgesi**) kinci tetik stratejisi (Grim) işbirliğini koruyabilir:

Uymak: her tur karşılıklı işbirliği, $3 \Rightarrow 3 + 3\delta + \ldots = \mathbf{3/(1-\delta)}$.

Sapmak: bu tur cazibe 5, sonra hep karşılıklı ihanet 1 $\Rightarrow$ $\mathbf{5 + \delta/(1-\delta)}$.

Uymak ≥ sapmak; iki taraf $(1-\delta)$ ile çarpılınca: $3 \geq 5(1-\delta) + \delta \Leftrightarrow \mathbf{\delta \geq (5-3)/(5-1) = 1/2}$.

Tekrarlanan Mahkûmun İkilemi

Oyuncu 1 \ Oyuncu 2	S (işbirliği)	İ (ihanet)
S (işbirliği)	(3, 3)	(0, 5)
İ (ihanet)	(5, 0)	(1, 1)

Puan (yüksek iyi): cazibe 5 > karşılıklı işbirliği 3 > karşılıklı ihanet 1 > enayi 0. S: sessiz kalmak.

- Tek seferlik oyunda ve sonlu, bilinen sayıda tekrarda (geriye doğru çıkarımla) her turda ihanet.
- Oyun her turdan sonra δ olasılıkla sürüyorsa (**geleceğin gölgesi**) kinci tetik stratejisi (Grim) işbirliğini koruyabilir:

Uymak: her tur karşılıklı işbirliği, $3 \Rightarrow 3 + 3\delta + \ldots = \mathbf{3/(1-\delta)}$.

Sapmak: bu tur cazibe 5, sonra hep karşılıklı ihanet 1 ⇒ $\mathbf{5 + \delta/(1-\delta)}$.

Uymak ≥ sapmak; iki taraf $(1-\delta)$ ile çarpılınca: $3 \geq 5(1-\delta) + \delta \Leftrightarrow \mathbf{\delta \geq (5-3)/(5-1) = 1/2}$.

$\delta = 0{,}9$: uymak 30, sapmak 14 ⇒ işbirliği. $\delta = 0{,}3$: uymak ≈ 4,29, sapmak ≈ 5,43 ⇒ ihanet.

Tekrarlanan Mahkûmun İkilemi

Oyuncu 1 \ Oyuncu 2	S (işbirliği)	İ (ihanet)
S (işbirliği)	(3, 3)	(0, 5)
İ (ihanet)	(5, 0)	(1, 1)

Puan (yüksek iyi): cazibe 5 > karşılıklı işbirliği 3 > karşılıklı ihanet 1 > enayi 0. S: sessiz kalmak.

- Tek seferlik oyunda ve sonlu, bilinen sayıda tekrarda (geriye doğru çıkarımla) her turda ihanet.
- Oyun her turdan sonra δ olasılıkla sürüyorsa (**geleceğin gölgesi**) kinci tetik stratejisi (Grim) işbirliğini koruyabilir:

Uymak: her tur karşılıklı işbirliği, $3 \Rightarrow 3 + 3\delta + \ldots = \mathbf{3/(1-\delta)}$.

Sapmak: bu tur cazibe 5, sonra hep karşılıklı ihanet $1 \Rightarrow \mathbf{5 + \delta/(1-\delta)}$.

Uymak ≥ sapmak; iki taraf $(1-\delta)$ ile çarpılınca: $3 \geq 5(1-\delta) + \delta \Leftrightarrow \mathbf{\delta \geq (5-3)/(5-1) = 1/2}$.

$\delta = 0{,}9$: uymak 30, sapmak 14 ⇒ işbirliği. $\delta = 0{,}3$: uymak ≈ 4,29, sapmak ≈ 5,43 ⇒ ihanet.

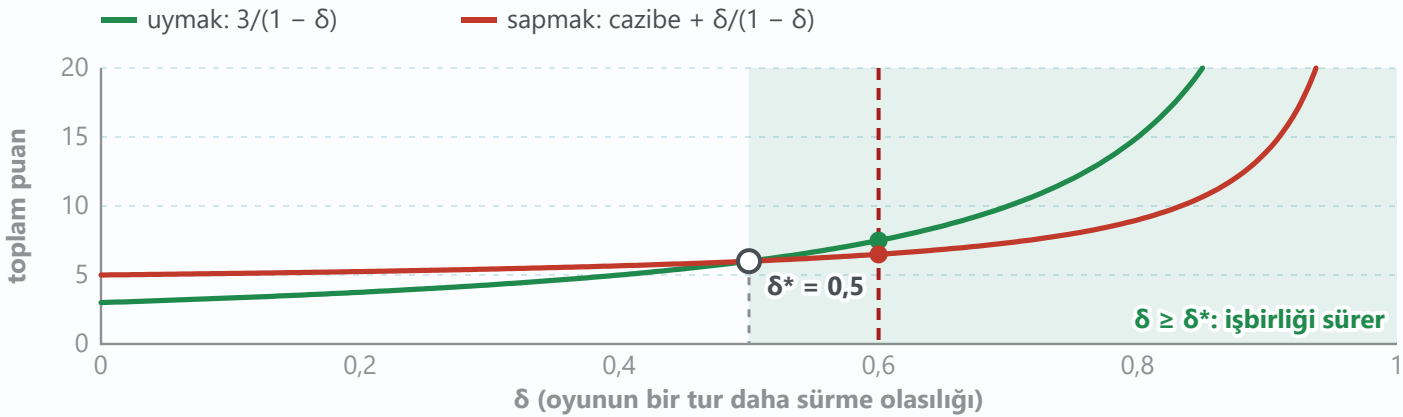
Ceza inandırıcıdır: (İ, İ) tek turun dengesidir. **Halk teoremi** (folk theorem): δ büyükse, herkesin ihanetle garantilediğinden fazlasını veren pek çok ortalama kazanç dengede sürer.

Kinci Tetik: İşbirliği Hangi δ'da Sürer?

Uymak: **3/(1 − δ)** · sapmak: **cazibe + δ/(1 − δ)** · uymak ≥ sapmak ⇔ **δ ≥ δ*** · δ* = (cazibe − 3)/(cazibe − 1)

δ = 0,6 **cazibe =** 5

uymak = **7,5** sapmak = 5 + δ/(1 − δ) = **6,5** eşik δ* = **1/2 = 0,5** **İşbirliği sürdürülebilir**



Gelecek ne kadar değerliyse (δ büyükse) kinci tetik tehdidi o kadar caydırıcıdır; cazibe arttıkça işbirliği için gereken δ* eşiği yükselir.

TEKRARLANAN OYUNLAR

10 Turluk Maçlar: Kim Ne Oynar?

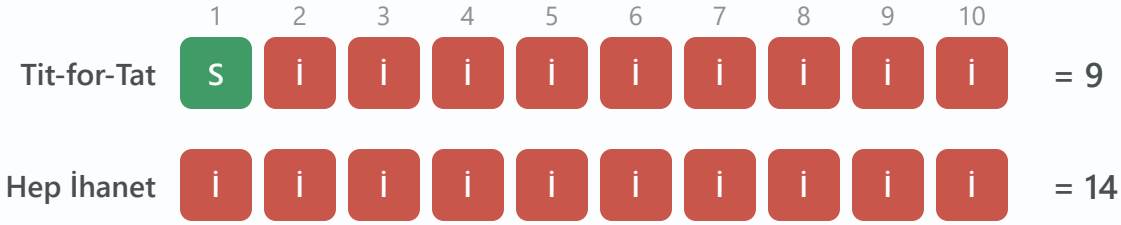


TFT işbirliği yapana işbirliğiyle karşılık verir: iki taraf da 30 puan. (Toplam: 30 – 30)

Adım 1 / 5

S: sessiz kalmak, yani işbirliği (yeşil), İ: ihanet (kırmızı). TFT: ilk turda işbirliği, sonra rakibin son hamlesini tekrarla.

10 Turluk Maçlar: Kim Ne Oynar?

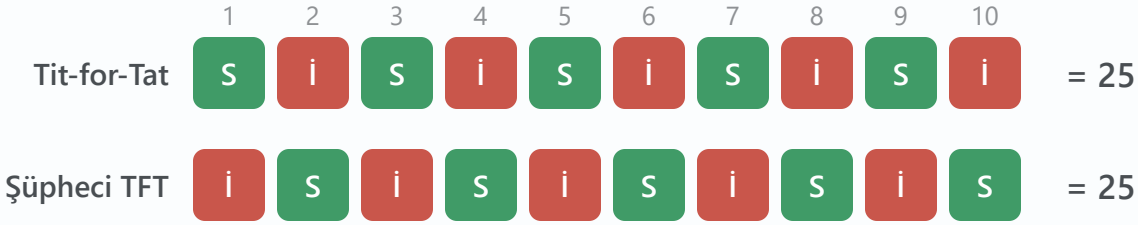


TFT ilk turda sömürülür, sonra hep ihanet eder: 9'a 14. TFT bir maçı **asla kazanamaz** (en fazla berabere). (Toplam: 9 – 14)

S: sessiz kalmak, yani işbirliği (yeşil), İ: ihanet (kırmızı). TFT: ilk turda işbirliği, sonra rakibin son hamlesini tekrarla.

TEKRARLANAN OYUNLAR

10 Turluk Maçlar: Kim Ne Oynar?

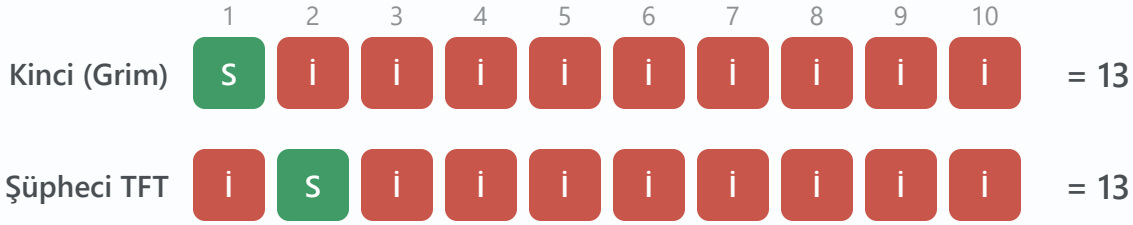


Şüpheci TFT ihanetle başlar; iki TFT birbirini "yankılar" → S/İ dönüşümlü, ikisi de 25. (Toplam: 25 – 25)

Adım 3 / 5

S: sessiz kalmak, yani işbirliği (yeşil), İ: ihanet (kırmızı). TFT: ilk turda işbirliği, sonra rakibin son hamlesini tekrarla.

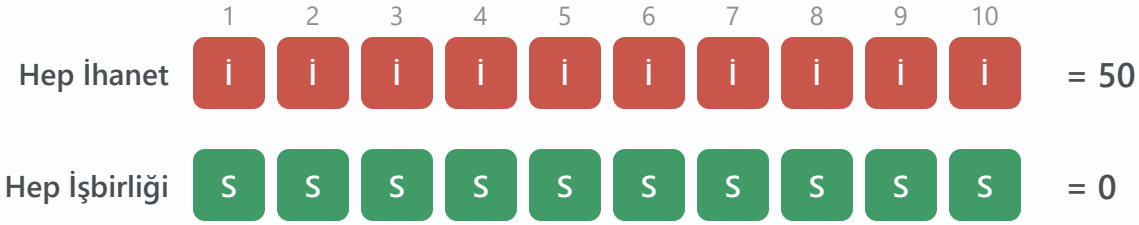
10 Turluk Maçlar: Kim Ne Oynar?



Kinci, tek ihaneti hiç affetmez: ikinci turdan sonra karşılıklı ihanet sürer. (Toplam: 13 – 13)

S: sessiz kalmak, yani işbirliği (yeşil), İ: ihanet (kırmızı). TFT: ilk turda işbirliği, sonra rakibin son hamlesini tekrarla.

10 Turluk Maçlar: Kim Ne Oynar?



Hep İhanet, saf işbirlikçiyi tamamen sömürür: 50'ye 0. (Toplam: 50 – 0)

S: sessiz kalmak, yani işbirliği (yeşil), İ: ihanet (kırmızı). TFT: ilk turda işbirliği, sonra rakibin son hamlesini tekrarla.

Kod İzleme: `match(tft, stft, 6)`



Çağrı `match(tft, stft, 6)` : geçmişler boş, puanlar 0.

Adım 1 / 14

```
1  PAY = {"SS": (3, 3), "Sİ": (0, 5),
2         "İS": (5, 0), "İİ": (1, 1)}
3  tft = lambda ben, rakip: rakip[-1] if rakip else "S"
4  stft = lambda ben, rakip: rakip[-1] if rakip else "İ"
5  def match(s1, s2, T):
▶      h1, h2, p1, p2 = [], [], 0, 0
7      for t in range(T):
8          a, b = s1(h1, h2), s2(h2, h1)
9          u, v = PAY[a + b]
10         h1.append(a); h2.append(b)
11         p1, p2 = p1 + u, p2 + v
12     return p1, p2
```

DEĞİŞKENLER

`T = 6` `h1 = []` `h2 = []` `p1 = 0` `p2 = 0`

Kod İzleme: `match(tft, stft, 6)`



t = 0: Geçmiş boş: TFT **S**, Şüpheci TFT **İ** ile açar.


```
1  PAY = {"SS": (3, 3), "Sİ": (0, 5),
2         "İS": (5, 0), "İİ": (1, 1)}
▶  tft = lambda ben, rakip: rakip[-1] if rakip else "S"
▶  stft = lambda ben, rakip: rakip[-1] if rakip else "İ"
5  def match(s1, s2, T):
6      h1, h2, p1, p2 = [], [], 0, 0
▶      for t in range(T):
▶          a, b = s1(h1, h2), s2(h2, h1)
9          u, v = PAY[a + b]
10         h1.append(a); h2.append(b)
11         p1, p2 = p1 + u, p2 + v
12     return p1, p2
```

DEĞİŞKENLER

`t = 0` `a = S` `b = İ`

`h1 = []` `h2 = []` `p1 = 0`

`p2 = 0`

Kod İzleme: `match(tft, stft, 6)`



PAY["Sİ"] = (0, 5) → puanlar **0 – 5**.

Adım 3 / 14

```
1  PAY = {"SS": (3, 3), "Sİ": (0, 5),
2         "İS": (5, 0), "İİ": (1, 1)}
3  tft = lambda ben, rakip: rakip[-1] if rakip else "S"
4  stft = lambda ben, rakip: rakip[-1] if rakip else "İ"
5  def match(s1, s2, T):
6      h1, h2, p1, p2 = [], [], 0, 0
7      for t in range(T):
8          a, b = s1(h1, h2), s2(h2, h1)
▶          u, v = PAY[a + b]
▶          h1.append(a); h2.append(b)
▶          p1, p2 = p1 + u, p2 + v
12     return p1, p2
```

DEĞİŞKENLER

`t = 0` `u = 0` `v = 5`
`h1 = [S]` `h2 = [İ]` `p1 = 0`
`p2 = 5`

Kod İzleme: `match(tft, stft, 6)`



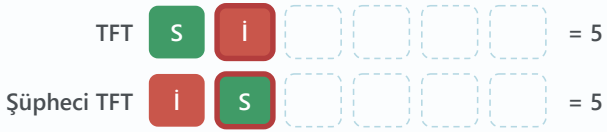
t = 1: İkisi de rakibin son hamlesini kopyalar: TFT → **İ**, ŞTFT → **S**.


```
1  PAY = {"SS": (3, 3), "Sİ": (0, 5),
2         "İS": (5, 0), "İİ": (1, 1)}
▶ tft = lambda ben, rakip: rakip[-1] if rakip else "S"
▶ stft = lambda ben, rakip: rakip[-1] if rakip else "İ"
5  def match(s1, s2, T):
6      h1, h2, p1, p2 = [], [], 0, 0
▶      for t in range(T):
▶          a, b = s1(h1, h2), s2(h2, h1)
9          u, v = PAY[a + b]
10         h1.append(a); h2.append(b)
11         p1, p2 = p1 + u, p2 + v
12     return p1, p2
```

DEĞİŞKENLER

`t = 1` `a = İ` `b = S` `h1 = [S]` `h2 = [İ]` `p1 = 0` `p2 = 5`

Kod İzleme: `match(tft, stft, 6)`



PAY["İS"] = (5, 0) → puanlar **5 – 5**.

Adım 5 / 14

```
1  PAY = {"SS": (3, 3), "Sİ": (0, 5),
2         "İS": (5, 0), "İİ": (1, 1)}
3  tft = lambda ben, rakip: rakip[-1] if rakip else "S"
4  stft = lambda ben, rakip: rakip[-1] if rakip else "İ"
5  def match(s1, s2, T):
6      h1, h2, p1, p2 = [], [], 0, 0
7      for t in range(T):
8          a, b = s1(h1, h2), s2(h2, h1)
▶          u, v = PAY[a + b]
▶          h1.append(a); h2.append(b)
▶          p1, p2 = p1 + u, p2 + v
12     return p1, p2
```

DEĞİŞKENLER

`t = 1` `u = 5` `v = 0`

`h1 = [S, İ]` `h2 = [İ, S]`

`p1 = 5` `p2 = 5`

Kod İzleme: `match(tft, stft, 6)`



t = 2: İkisi de rakibin son hamlesini kopyalar: TFT → **S**, ŞTFT → **İ**.

Adım 6 / 14

```
1  PAY = {"SS": (3, 3), "Sİ": (0, 5),
2         "İS": (5, 0), "İİ": (1, 1)}
▶ tft = lambda ben, rakip: rakip[-1] if rakip else "S"
▶ stft = lambda ben, rakip: rakip[-1] if rakip else "İ"
5  def match(s1, s2, T):
6      h1, h2, p1, p2 = [], [], 0, 0
▶      for t in range(T):
▶          a, b = s1(h1, h2), s2(h2, h1)
9          u, v = PAY[a + b]
10         h1.append(a); h2.append(b)
11         p1, p2 = p1 + u, p2 + v
12     return p1, p2
```

DEĞİŞKENLER

`t = 2` `a = S` `b = İ`

`h1 = [S, İ]` `h2 = [İ, S]`

`p1 = 5` `p2 = 5`

Kod İzleme: `match(tft, stft, 6)`



PAY["Sİ"] = (0, 5) → puanlar **5 – 10**.

Adım 7 / 14

```
1  PAY = {"SS": (3, 3), "Sİ": (0, 5),
2         "İS": (5, 0), "İİ": (1, 1)}
3  tft = lambda ben, rakip: rakip[-1] if rakip else "S"
4  stft = lambda ben, rakip: rakip[-1] if rakip else "İ"
5  def match(s1, s2, T):
6      h1, h2, p1, p2 = [], [], 0, 0
7      for t in range(T):
8          a, b = s1(h1, h2), s2(h2, h1)
▶          u, v = PAY[a + b]
▶          h1.append(a); h2.append(b)
▶          p1, p2 = p1 + u, p2 + v
12     return p1, p2
```

DEĞİŞKENLER

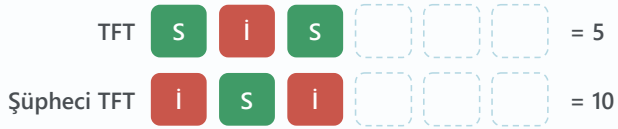
`t = 2` `u = 0` `v = 5`

`h1 = [S, İ, S]`

`h2 = [İ, S, İ]` `p1 = 5`

`p2 = 10`

Kod İzleme: `match(tft, stft, 6)`



t = 3: İkisi de rakibin son hamlesini kopyalar: TFT → **İ**, ŞTFT → **S**.


```
1  PAY = {"SS": (3, 3), "Sİ": (0, 5),
2         "İS": (5, 0), "İİ": (1, 1)}
▶  tft = lambda ben, rakip: rakip[-1] if rakip else "S"
▶  stft = lambda ben, rakip: rakip[-1] if rakip else "İ"
5  def match(s1, s2, T):
6      h1, h2, p1, p2 = [], [], 0, 0
▶      for t in range(T):
▶          a, b = s1(h1, h2), s2(h2, h1)
9          u, v = PAY[a + b]
10         h1.append(a); h2.append(b)
11         p1, p2 = p1 + u, p2 + v
12     return p1, p2
```

DEĞİŞKENLER

`t = 3` `a = İ` `b = S`

`h1 = [S, İ, S]`

`h2 = [İ, S, İ]` `p1 = 5`

`p2 = 10`

Kod İzleme: `match(tft, stft, 6)`



PAY["İS"] = (5, 0) → puanlar **10 – 10**.

Adım 9 / 14

```
1  PAY = {"SS": (3, 3), "Sİ": (0, 5),
2         "İS": (5, 0), "İİ": (1, 1)}
3  tft = lambda ben, rakip: rakip[-1] if rakip else "S"
4  stft = lambda ben, rakip: rakip[-1] if rakip else "İ"
5  def match(s1, s2, T):
6      h1, h2, p1, p2 = [], [], 0, 0
7      for t in range(T):
8          a, b = s1(h1, h2), s2(h2, h1)
▶          u, v = PAY[a + b]
▶          h1.append(a); h2.append(b)
▶          p1, p2 = p1 + u, p2 + v
12     return p1, p2
```

DEĞİŞKENLER

`t = 3` `u = 5` `v = 0`

`h1 = [S, İ, S, İ]`

`h2 = [İ, S, İ, S]` `p1 = 10`

`p2 = 10`

Kod İzleme: `match(tft, stft, 6)`



t = 4: İkisi de rakibin son hamlesini kopyalar: TFT → **S**, ŞTFT → **İ**.

Adım 10 / 14

```
1  PAY = {"SS": (3, 3), "Sİ": (0, 5),
2         "İS": (5, 0), "İİ": (1, 1)}
▶  tft = lambda ben, rakip: rakip[-1] if rakip else "S"
▶  stft = lambda ben, rakip: rakip[-1] if rakip else "İ"
5  def match(s1, s2, T):
6      h1, h2, p1, p2 = [], [], 0, 0
▶      for t in range(T):
▶          a, b = s1(h1, h2), s2(h2, h1)
9          u, v = PAY[a + b]
10         h1.append(a); h2.append(b)
11         p1, p2 = p1 + u, p2 + v
12     return p1, p2
```

DEĞİŞKENLER

`t = 4` `a = S` `b = İ`

`h1 = [S, İ, S, İ]`

`h2 = [İ, S, İ, S]` `p1 = 10`

`p2 = 10`

Kod İzleme: `match(tft, stft, 6)`



PAY["Sİ"] = (0, 5) → puanlar **10 – 15**.

Adım 11 / 14

```
1  PAY = {"SS": (3, 3), "Sİ": (0, 5),
2         "İS": (5, 0), "İİ": (1, 1)}
3  tft = lambda ben, rakip: rakip[-1] if rakip else "S"
4  stft = lambda ben, rakip: rakip[-1] if rakip else "İ"
5  def match(s1, s2, T):
6      h1, h2, p1, p2 = [], [], 0, 0
7      for t in range(T):
8          a, b = s1(h1, h2), s2(h2, h1)
▶          u, v = PAY[a + b]
▶          h1.append(a); h2.append(b)
▶          p1, p2 = p1 + u, p2 + v
12     return p1, p2
```

DEĞİŞKENLER

`t = 4` `u = 0` `v = 5`

`h1 = [S, İ, S, İ, S]`

`h2 = [İ, S, İ, S, İ]` `p1 = 10`

`p2 = 15`

Kod İzleme: `match(tft, stft, 6)`



t = 5: İkisi de rakibin son hamlesini kopyalar: TFT → **İ**, ŞTFT → **S**.

Adım 12 / 14

```
1  PAY = {"SS": (3, 3), "Sİ": (0, 5),
2         "İS": (5, 0), "İİ": (1, 1)}
▶  tft = lambda ben, rakip: rakip[-1] if rakip else "S"
▶  stft = lambda ben, rakip: rakip[-1] if rakip else "İ"
5  def match(s1, s2, T):
6      h1, h2, p1, p2 = [], [], 0, 0
▶      for t in range(T):
▶          a, b = s1(h1, h2), s2(h2, h1)
9          u, v = PAY[a + b]
10         h1.append(a); h2.append(b)
11         p1, p2 = p1 + u, p2 + v
12     return p1, p2
```

DEĞİŞKENLER

`t = 5` `a = İ` `b = S`

`h1 = [S, İ, S, İ, S]`

`h2 = [İ, S, İ, S, İ]` `p1 = 10`

`p2 = 15`

Kod İzleme: `match(tft, stft, 6)`



PAY["İS"] = (5, 0) → puanlar **15 – 15**.

Adım 13 / 14

```
1  PAY = {"SS": (3, 3), "Sİ": (0, 5),
2         "İS": (5, 0), "İİ": (1, 1)}
3  tft = lambda ben, rakip: rakip[-1] if rakip else "S"
4  stft = lambda ben, rakip: rakip[-1] if rakip else "İ"
5  def match(s1, s2, T):
6      h1, h2, p1, p2 = [], [], 0, 0
7      for t in range(T):
8          a, b = s1(h1, h2), s2(h2, h1)
▶          u, v = PAY[a + b]
▶          h1.append(a); h2.append(b)
▶          p1, p2 = p1 + u, p2 + v
12     return p1, p2
```

DEĞİŞKENLER

`t = 5` `u = 5` `v = 0`

`h1 = [S, İ, S, İ, S, İ]`

`h2 = [İ, S, İ, S, İ, S]`

`p1 = 15` `p2 = 15`

Kod İzleme: `match(tft, stft, 6)`



return (15, 15). Tek bir ilk ihanet sonsuz bir "yankı" başlatır: Sİ/İS dönüşümlü, tur başına ortalama 2,5 puan (karşılıklı işbirliğinde 3 olurdu).

Adım 14 / 14

```
1  PAY = {"SS": (3, 3), "Sİ": (0, 5),
2         "İS": (5, 0), "İİ": (1, 1)}
3  tft = lambda ben, rakip: rakip[-1] if rakip else "S"
4  stft = lambda ben, rakip: rakip[-1] if rakip else "İ"
5  def match(s1, s2, T):
6      h1, h2, p1, p2 = [], [], 0, 0
7      for t in range(T):
8          a, b = s1(h1, h2), s2(h2, h1)
9          u, v = PAY[a + b]
10         h1.append(a); h2.append(b)
11         p1, p2 = p1 + u, p2 + v
▶      return p1, p2
```

DEĞİŞKENLER

- h1 = **[S, İ, S, İ, S, İ]**
- h2 = **[İ, S, İ, S, İ, S]**
- p1 = **15**
- p2 = **15**

TEKRARLANAN OYUNLAR

Mini Axelrod Turnuvası: Adım Adım

Strateji \ rakip	Hep-S	Hep-İ	TFT	Kinci	ŞTFT	Toplam	Sıra
Hep İşbirliği	30	0	30	30	27	**117**	
Hep İhanet							
Tit-for-Tat							
Kinci (Grim)							
Şüpheci TFT							

Hep İşbirliği herkesle (kendi ikiziyle de) 10 tur oynar: toplam **117**; en çok Hep İşbirliği karşısında kazanır (30).

Adım 1 / 6

Satırdaki stratejinin, sütundaki rakibe karşı 10 turda topladığı puan. Hep-S: Hep İşbirliği, Hep-İ: Hep İhanet, ŞTFT: Şüpheci TFT.

TEKRARLANAN OYUNLAR

Mini Axelrod Turnuvası: Adım Adım

Strateji \ rakip	Hep-S	Hep-İ	TFT	Kinci	ŞTFT	Toplam	Sıra
Hep İşbirliği	30	0	30	30	27	**117**	
Hep İhanet	50	10	14	14	10	**98**	
Tit-for-Tat							
Kinci (Grim)							
Şüpheci TFT							

Hep İhanet herkesle (kendi ikiziyle de) 10 tur oynar: toplam **98**; en çok Hep İşbirliği karşısında kazanır (50).

Adım 2 / 6

Satırdaki stratejinin, sütundaki rakibe karşı 10 turda topladığı puan. Hep-S: Hep İşbirliği, Hep-İ: Hep İhanet, ŞTFT: Şüpheci TFT.

TEKRARLANAN OYUNLAR

Mini Axelrod Turnuvası: Adım Adım

Strateji \ rakip	Hep-S	Hep-İ	TFT	Kinci	ŞTFT	Toplam	Sıra
Hep İşbirliği	30	0	30	30	27	**117**	
Hep İhanet	50	10	14	14	10	**98**	
Tit-for-Tat	30	9	30	30	25	**124**	
Kinci (Grim)							
Şüpheci TFT							

Tit-for-Tat herkesle (kendi ikiziyle de) 10 tur oynar: toplam **124**; en çok Hep İşbirliği karşısında kazanır (30).

Adım 3 / 6

Satırdaki stratejinin, sütundaki rakibe karşı 10 turda topladığı puan. Hep-S: Hep İşbirliği, Hep-İ: Hep İhanet, ŞTFT: Şüpheci TFT.

TEKRARLANAN OYUNLAR

Mini Axelrod Turnuvası: Adım Adım

Strateji \ rakip	Hep-S	Hep-İ	TFT	Kinci	ŞTFT	Toplam	Sıra
Hep İşbirliği	30	0	30	30	27	**117**	
Hep İhanet	50	10	14	14	10	**98**	
Tit-for-Tat	30	9	30	30	25	**124**	
Kinci (Grim)	30	9	30	30	13	**112**	
Şüpheci TFT							

Kinci (Grim) herkesle (kendi ikiziyle de) 10 tur oynar: toplam **112**; en çok Hep İşbirliği karşısında kazanır (30).

Adım 4 / 6

Satırdaki stratejinin, sütundaki rakibe karşı 10 turda topladığı puan. Hep-S: Hep İşbirliği, Hep-İ: Hep İhanet, ŞTFT: Şüpheci TFT.

TEKRARLANAN OYUNLAR

Mini Axelrod Turnuvası: Adım Adım

Strateji \ rakip	Hep-S	Hep-İ	TFT	Kinci	ŞTFT	Toplam	Sıra
Hep İşbirliği	30	0	30	30	27	**117**	
Hep İhanet	50	10	14	14	10	**98**	
Tit-for-Tat	30	9	30	30	25	**124**	
Kinci (Grim)	30	9	30	30	13	**112**	
Şüpheci TFT	32	10	25	13	10	**90**	

Şüpheci TFT herkesle (kendi ikiziyle de) 10 tur oynar: toplam **90**; en çok Hep İşbirliği karşısında kazanır (32).

Adım 5 / 6

Satırdaki stratejinin, sütundaki rakibe karşı 10 turda topladığı puan. Hep-S: Hep İşbirliği, Hep-İ: Hep İhanet, ŞTFT: Şüpheci TFT.

TEKRARLANAN OYUNLAR

Mini Axelrod Turnuvası: Adım Adım

Strateji \ rakip	Hep-S	Hep-İ	TFT	Kinci	ŞTFT	Toplam	Sıra
Hep İşbirliği	30	0	30	30	27	**117**	2
Hep İhanet	50	10	14	14	10	**98**	4
Tit-for-Tat	30	9	30	30	25	**124**	1
Kinci (Grim)	30	9	30	30	13	**112**	3
Şüpheci TFT	32	10	25	13	10	**90**	5

Sıralama: Tit-for-Tat (124), Hep İşbirliği (117), Kinci (Grim) (112), Hep İhanet (98), Şüpheci TFT (90). **Tit-for-Tat birinci** — hiçbir maçı kazanmadan! Hep İhanet ise sondan ikinci.

Adım 6 / 6

Satırdaki stratejinin, sütundaki rakibe karşı 10 turda topladığı puan. Hep-S: Hep İşbirliği, Hep-İ: Hep İhanet, ŞTFT: Şüpheci TFT.

TEKRARLANAN OYUNLAR

Axelrod Turnuvaları (1980) ve Dersleri

Robert Axelrod iki bilgisayar turnuvası düzenledi (14 ve 62 program). İkisini de Anatol Rapoport'un 4 satırlık **Tit-for-Tat**'ı kazandı.

İyi niyetli
İlk ihaneti yapan olma.

Misilleme yapan
İhanet cezasız kalmasın.

Affedici
Rakip dönerse sen de dön.

Anlaşılır
Rakip kuralını kolayca öğrenebilsin.

- **Gürültü** (yanlış hamle olasılığı) varsa iki TFT sonsuz misilleme döngüsüne girer; *cömert TFT* ihaneti bazen affeder ve daha iyi sonuç verir.
- Başarı ortama bağlıdır: rakip havuzu değişirse sıralama değişir — evrimsel turnuvada (replikatör) başarılı stratejiler çoğalır.



İki TFT, 4. turda A'nın tek bir hatası: sonrası sonsuz S/İ yankısı. 10 turda 29 – 24 (hatasız 30 – 30).

ÖZET

Tartışma: Öğrenme, İnsan ve Etik

Öğrenme her zaman optimuma götürür mü?

Hayır. Bazı dinamikler döner (Yazı-Tura, Shapley örneği); çevre ve rakipler değişir; yanlış model kötü yakınsama doğurur. Güvenilir hedef çoğu zaman *pişmanlıksızlık*tır.

İnsanlar algoritmalar gibi mi öğrenir?

Benzer: deneme-yanılma, geçmişten öğrenme. Farklı: insanlar risk algısı, duygular ve sosyal normlarla hareket eder; algoritmalar tanımlı bir ödülü izler.

Uyarlanabilir sistemlerde etik riskler

Veri önyargısı, kullanıcıyı manipüle eden öneriler (tıklama tuzağı), mahremiyet, algoritmik fiyat koordinasyonu (öğrenen fiyat botlarının örtük anlaşması).

Özet

Yöntem	Açıklama	Sonuç
En iyi tepki dinamiği	Son eyleme en iyi tepki	Potansiyel oyunda dengeye; aksi hâlde döngü
Kurgusal oyun	Geçmiş frekanslara en iyi tepki	Sıfır toplamlıda frekanslar dengeye
Hedge	$w \leftarrow w \cdot e^{\eta g}$	$R_T = O(\sqrt{(T \ln n)})$; zaman ortalaması → CCE
ε-açgözlü / UCB	Keşif–sömürü dengesi	UCB: logaritmik pişmanlık
Q-öğrenme	$Q \leftarrow Q + \alpha(r + \gamma \cdot \max Q' - Q)$	Ödül hedeften geriye yayılır
Replikatör	Başarılı strateji yayılır	Şahin–Güvercin: $x^* = V/C$
Tekrarlanan MI	Geleceğin gölgesi	$\delta \geq 1/2$ ise işbirliği; TFT güçlü

ÖZET

Örnek Problemler

1. Hedge'de $\eta = \ln 2$, iki eylem, ilk iki turda kazançlar (1, 0) ve (1, 0). Üçüncü turda p_1 kaçtır?
2. UCB1, $t = 10$'da: kol 1 için $Q = 0{,}6$, $N = 5$; kol 2 için $Q = 0{,}2$, $N = 1$. Hangi kol seçilir?
3. Şahin–Güvercin ($V = 4$, $C = 6$, taban 3) oyununda $x = 0{,}5$ iken bir sonraki nesilde şahin payı artar mı?
4. Cazibe 5, karşılıklı işbirliği 3, karşılıklı ihanet 1 iken Grim ile işbirliğini sürdürmek için gereken en küçük δ nedir? Cazibe 7 olsaydı?
5. Kurgusal oyun ile en iyi tepki dinamiği arasındaki fark nedir? Yazı-Tura'da hangisi dengeye "yaklaşır"?
6. Q-öğrenmede $Q(s, a) = 1$, $r = 0$, $\gamma = 0{,}9$, $\max Q(s', \cdot) = 2$, $\alpha = 0{,}1$. Güncellemeden sonra $Q(s, a)$ kaç olur?

▶ **Cevapları göster**

ÖZET

Etkileşimli Oyunlar ve Laboratuvarlar

Çok Kollu Haydut

ε-açgözlü, UCB ve elle seçilen stratejiyle kol çekme.

Nashpy (Python kütüphanesi)

Herhangi bir kazanç matrisinde kurgusal oyun (`fictitious_play`) ve replikatör dinamiği (`replicator_dynamics`).

Tekrarlanan MI Turnuvası

Stratejilerin turnuvada yarışı: TFT yenilebilir mi?

Replikatör Dinamiği

Şahin–Güvercin, Taş-Kağıt-Makas ve evrimsel laboratuvar.

Kaynaklar

- D. Fudenberg, D. Levine, *The Theory of Learning in Games*, MIT Press, 1998.
- N. Cesa-Bianchi, G. Lugosi, *Prediction, Learning, and Games*, Cambridge, 2006.
- T. Lattimore, C. Szepesvári, *Bandit Algorithms*, Cambridge, 2020.
- P. Auer, N. Cesa-Bianchi, P. Fischer, "Finite-time analysis of the multiarmed bandit problem", *Machine Learning* 47, 2002.
- C. Watkins, P. Dayan, "Q-learning", *Machine Learning* 8, 1992; A. Blum, Y. Mansour, "From external to internal regret", *JMLR* 8, 2007.
- R. Axelrod, *The Evolution of Cooperation*, Basic Books, 1984.
- J. Hofbauer, K. Sigmund, *Evolutionary Games and Population Dynamics*, Cambridge, 1998.
- T. Roughgarden, *Twenty Lectures on Algorithmic Game Theory*, 2016 (Ders 16–18); ders sayfası: CS364A (Stanford, 2013).
- N. Nisan, T. Roughgarden, É. Tardos, V. Vazirani (ed.), *Algorithmic Game Theory*, Cambridge, 2007 (Bölüm 4: öğrenme, pişmanlık ve denge).
- A. R. Karlin, Y. Peres, *Game Theory, Alive*, AMS, 2017 (Bölüm 7: evrimsel denge; Bölüm 18: uyarlanabilir karar verme); D. Easley, J. Kleinberg, *Networks, Crowds, and Markets*, 2010 (Bölüm 7: evrimsel oyun kuramı).
- Araç: Nashpy (kurgusal oyun, replikatör dinamiği).

Kaynaklar (2/2)

- J. Robinson, “An iterative method of solving a game”, *Annals of Mathematics* 54(2), 1951; K. Miyasawa, “On the convergence of the learning process in a 2×2 non-zero-sum two-person game”, Princeton Research Memo 33, 1961; L. S. Shapley, “Some topics in two-person games”, *Advances in Game Theory*, 1964.
- D. Monderer, L. S. Shapley, “Potential games”, *Games and Economic Behavior* 14(1), 1996.
- D. Foster, R. Vohra, “Calibrated learning and correlated equilibrium”, *Games and Economic Behavior* 21, 1997; S. Hart, A. Mas-Colell, “A simple adaptive procedure leading to correlated equilibrium”, *Econometrica* 68(5), 2000.
- J. Maynard Smith, G. R. Price, “The logic of animal conflict”, *Nature* 246, 1973.
- W. R. Thompson, “On the likelihood that one unknown probability exceeds another in view of the evidence of two samples”, *Biometrika* 25, 1933.
- P. Auer, N. Cesa-Bianchi, Y. Freund, R. E. Schapire, “The nonstochastic multiarmed bandit problem”, *SIAM Journal on Computing* 32(1), 2002.
- N. D. Daw, J. P. O’Doherty, P. Dayan, B. Seymour, R. J. Dolan, “Cortical substrates for exploratory decisions in humans”, *Nature* 441, 2006.
- C. Watkins, *Learning from Delayed Rewards*, doktora tezi, Cambridge Üniversitesi, 1989; V. Mnih vd., “Human-level control through deep reinforcement learning”, *Nature* 518, 2015.
- M. Zinkevich, M. Johanson, M. Bowling, C. Piccione, “Regret minimization in games with incomplete information”, *NeurIPS*, 2007.
- M. Nowak, K. Sigmund, “A strategy of win-stay, lose-shift that outperforms tit-for-tat in the Prisoner’s Dilemma game”, *Nature* 364, 1993.

Son

Bölüm 13: Öğrenme ve Adaptasyon

Sonraki bölüm: Bölüm 14: Gelecek Yönelimler

Çalışma soruları · Terim sözlüğü