

# Değdi mi? Çarpışma ve sekme

Ekranda gezen şekilleri oyuna çeviren soru: iki şey birbirine değdi mi?

Tuval sınırı

Dikdörtgenler: AABB

Daireler: dist

Sekme

## SORU

# Top rakete değdi mi? Bilgisayar bunu nereden bilecek?

Ekranda yalnızca renkli pikseller var. Program “top” ve “raket” diye bir şey görmüyor; elinde birkaç sayı var.



# Çarpışma algılama

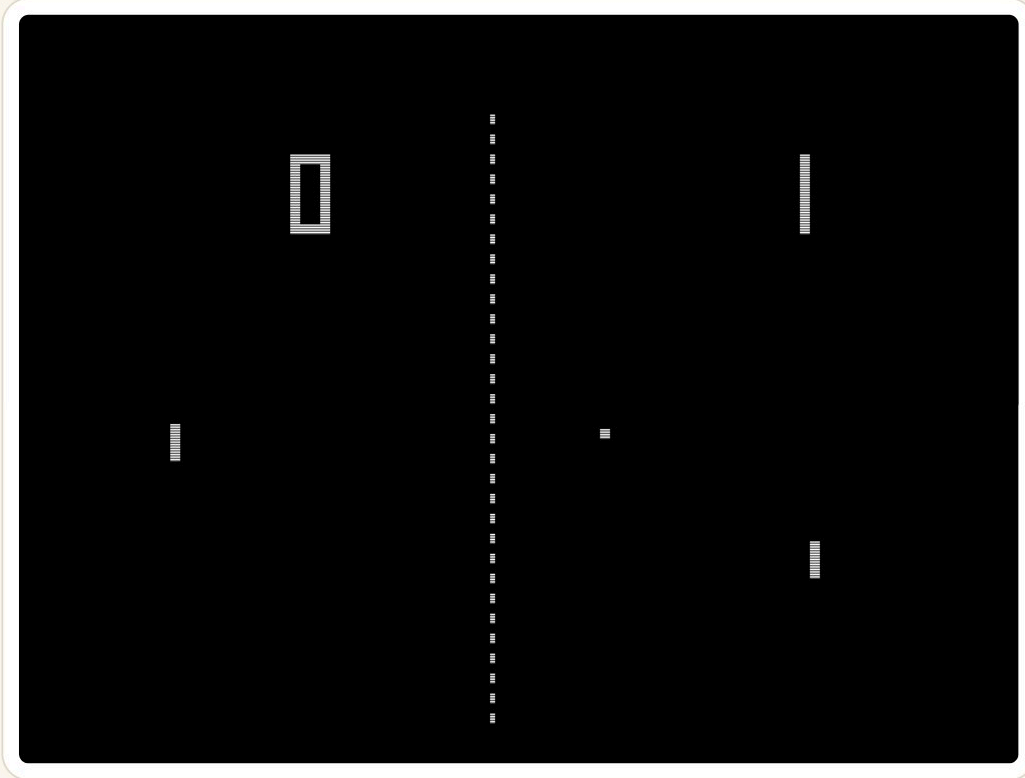
## Çarpışma algılama

İki ya da daha fazla oyun nesnesinin (ya da bir nesne ile sınırın) geometrik olarak kesişip kesişmediğini **her frame** kontrol etme işlemidir.

Görsel değil, **mantıksal** bir testtir: koşul `true` olunca kodumuz tepki verir. Tuvalde kendiliğinden bir efekt oluşmaz.

| Adım | İş                    | Örnek                                    |
|------|-----------------------|------------------------------------------|
| 1    | Girdiyi oku           | <code>keyIsDown</code>                   |
| 2    | Konumu güncelle       | <code>topX += hizX * dt</code>           |
| 3    | Çarpışmayı kontrol et | Kenar, başka blok                        |
| 4    | Tepki uygula          | Sekme, skor, oyun bitişi                 |
| 5    | Çiz                   | <code>rect</code> , <code>ellipse</code> |

# Her şey bununla başladı: Pong, 1972



Kaynak: [Wikimedia Commons, Pong.png](#) (kamu malı)

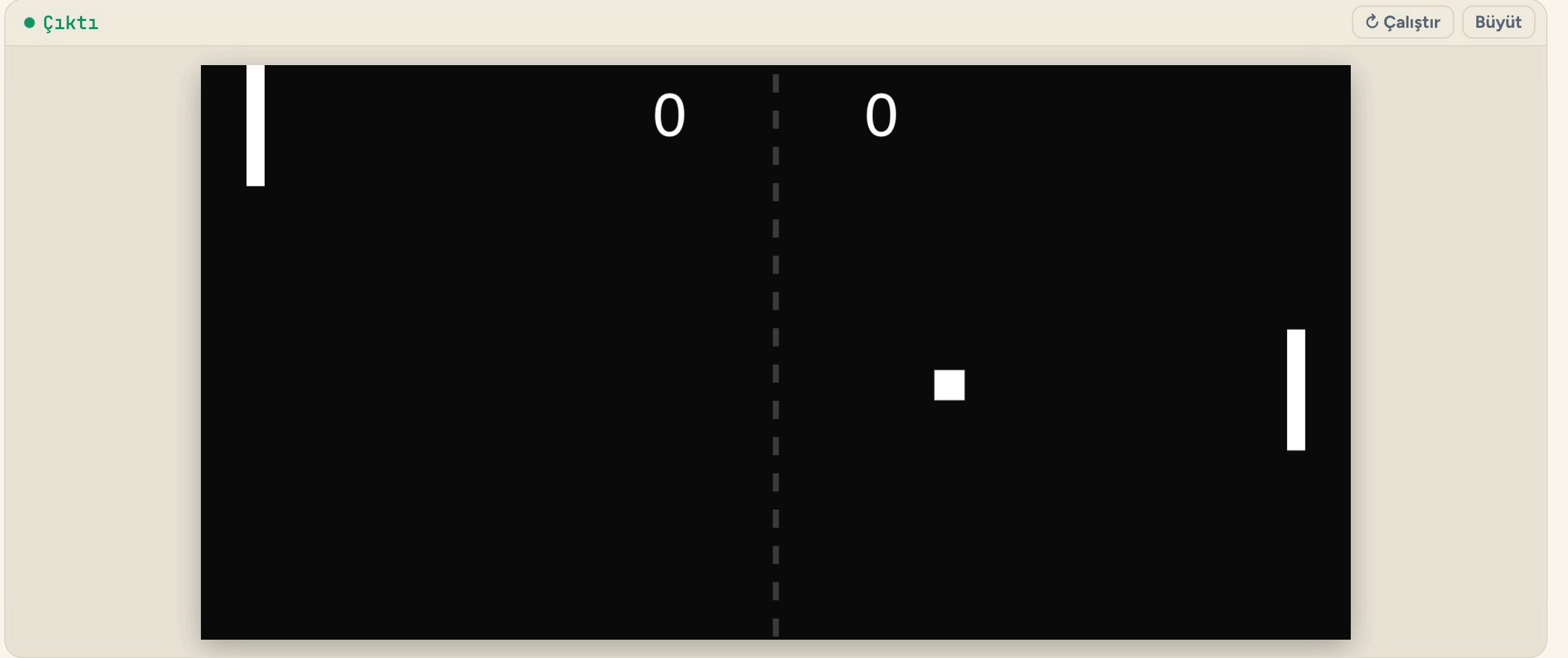
İki raket, bir top, iki kural:

- Top rakete değerse yön değiştirir.
- Top kenardan çıkarsa karşı taraf sayı alır.

## Elli yıldır aynı iskelet

Hareket et → değdi mi? → tepki ver. Bugünün mobil oyunlarında da döngü aynı.

# Mini Pong: fareyle raketi oynatın



Bu oyunun tamamı bugün göreceğimiz iki testten ibaret: tuval kenarı ve dikdörtgen örtüşmesi.

## Çarpışma olunca ne olur? (1/2)

### Pong

Raket topa değer: yön değişir. Kenar: sekme ya da sayı.

### Breakout / Arkanoid

Top tuğlaya değer: tuğla kaybolur, skor artar.

### Pac-Man

Hayaletle değince can gider; duvara değince yol kesilir.

### Super Mario Bros.

Zemine inmek, düşmana üstten ya da yandan değmek, altın almak.

Çarpışma olmasa çoğu oyun “boşlukta süzülen şekiller”e iner: raket topa değmez, mermi hedefi delip geçer, karakter duvardan yürür.

## Çarpışma olunca ne olur? (2/2)

### Flappy Bird

Boruya bir kez değmek:  
oyun biter.

### Space Invaders / Galaga

Mermi uzaylıya değer:  
düşman yok olur.

### Angry Birds

Kuş, blok ve domuz  
kesişince yıkım ve skor.  
Arkada bir fizik motoru  
çalışır.

### Pinball / bilardo

Top engele ya da kenara  
değer: sekme.

Skor artışı, can kaybı, sekme, kapı açma, ölme... Bu kuralların çoğu, tam da çarpışma testi `true` olduğu anda uygulanır.

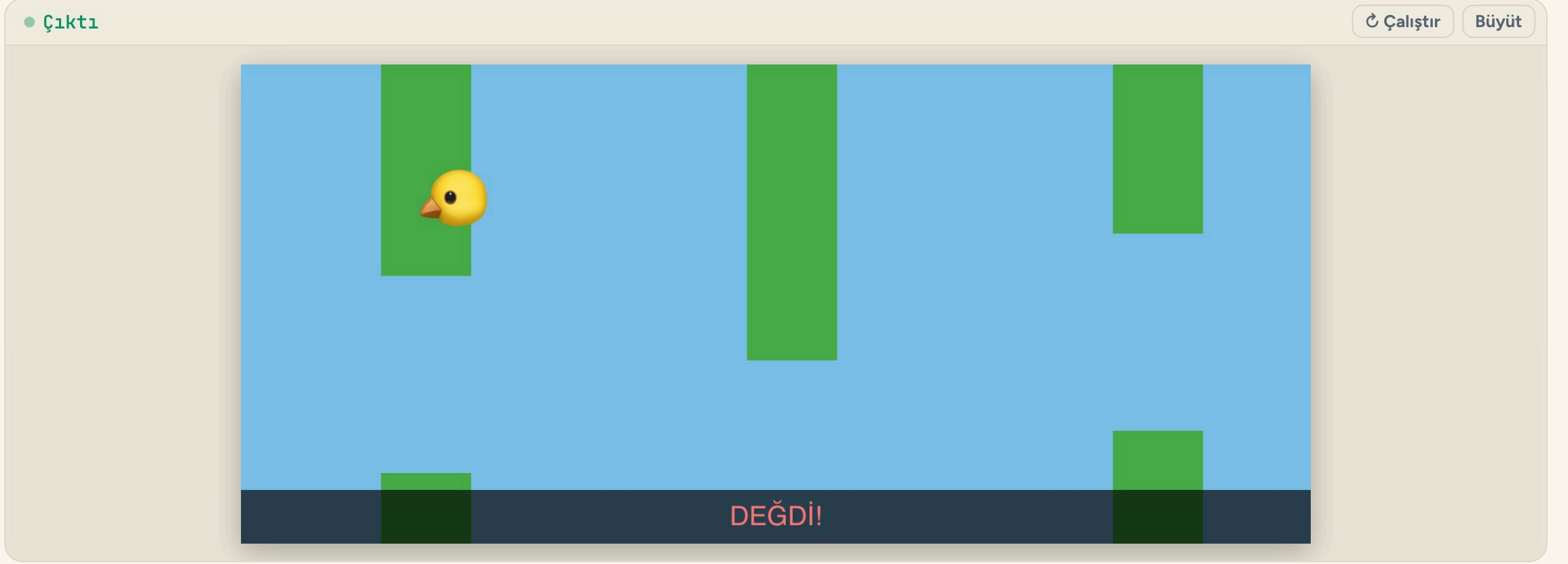
## SORU

# Kuşun kanadı boruya değdi mi? Gagası? Tüyleri?

Karakterin çizimi çok detaylı olabilir. Oyun gerçekten her pikseli tek tek mi kontrol ediyor?



# Hitbox: görünmeyen sade şekil



## Hitbox (çarpışma kutusu)

Çizim ne kadar süslü olursa olsun, "değdi mi?" sorusu çoğu zaman görünmeyen sade bir kutuya ya da daireye sorulur.

# Ticari oyunlarda soru iki adımda sorulur

## 1 • Kaba test

Ucuzdur. Her nesnenin etrafındaki büyük kutuya bakılır. Kutular uzaksa “değmedi” denir ve iş biter.

## 2 • İnce test

Pahalıdır. Yalnızca kutular örtüşünce devreye girer: çoklu hitbox, üçgen-üçgen ya da piksel düzeyinde kontrol.

Amaç, birbirinden uzak çiftler için boşuna ağır hesap yapmamak. Bugün öğreneceğimiz yöntemler (sınır, AABB, `dist`) bu kaba testlerdir ve Pong’dan Flappy Bird’e oyun hissinin omurgasını oluşturur.

### Algılama, fizik simülasyonu değildir

Gerçekçi sekme, sürtünme, dönme ileri konulardır. Burada yönü tersine çevirmek ( `hizX = -hizX` ) yeterli bir tepkidir.

# İlk test: tuval sınırı

## Sınır (boundary) çarpışması

Hareketli nesnenin tuval kenarına (  $x = 0$ ,  $x = \text{width}$ ,  $y = 0$ ,  $y = \text{height}$  ) ya da tanımlı bir oyun alanı sınırına değip değmediğinin kontrolüdür.

| Şekil                                  | Sol kenar | Sağ kenar | Üst kenar | Alt kenar |
|----------------------------------------|-----------|-----------|-----------|-----------|
| Dikdörtgen $(x, y, w, h)$              | $x$       | $x + w$   | $y$       | $y + h$   |
| Daire, merkez $(cx, cy)$ , yarıçap $r$ | $cx - r$  | $cx + r$  | $cy - r$  | $cy + r$  |

Hatırlatma: `rect` sol üst köşeden, `ellipse` merkezden konumlanır. Kenarların formülü bu yüzden farklı.

# Kare tuvalden çıkmasın

sketch.js · satır 11-29

Kopyala

Büyüt

```
11 function draw() {
12   background(35, 40, 55);
13   let dt = deltaTime / 1000;
14
15   if (keyIsDown(RIGHT_ARROW)) {
16     kareX = kareX + HIZ * dt;
17   }
18
19   // Sağ kenar: sol üst x + genişlik tuvali aşmasın
20   if (kareX + BOYUT > width) {
21     kareX = width - BOYUT;
22   }
23   if (kareX < 0) {
24     kareX = 0;
25   }
26
27   fill(100, 200, 255);
28   rect(kareX, kareY, BOYUT, BOYUT);
29 }
```

Çıktı

Çalıştır

Büyüt



Tuvale tıklayıp sağ oka basılı tutun. Karenin **sağ kenarı** `kareX + BOYUT`; tuvali aşarsa kareyi geri çekiyoruz.

## constrain ile aynı iş, ama...

```
constrain(x, 0, width - BOYUT)
```

Geçen bölümde aynı sınırı tek satırda uygulamıştık. Pratik bir kısayol.

### Açık if blokları

Sınırın **ne zaman** devreye girdiğini gösterir. O anda başka bir şey yapma şansı verir.

### Sekme için constrain yetmez

Top duvara çarpınca yön de değişmeli. Bunun için konumu düzelten `if` ile hızı tersine çeviren satır birlikte yazılır.

Daire için test merkez ve yarıçapla yapılır: `topX - r < 0` sol duvara, `topX + r > width` sağ duvara değmedir.

## SORU

# İki dikdörtgen tam olarak ne zaman “çarpmış” sayılır?

Birazdan B kutusunu fareyle gezdireceksiniz. Kenarlardaki renkli çubuklara dikkat edin: kutuların eksenlerdeki gölgeleri.



# İki ekseninde birden örtüşme gerekir

● ÇıktıÇalıştırBüyüt

X ekseninde örtüşme: yok  
Y ekseninde örtüşme: var  
Çarpışma: hayır

B'yi A'nın tam yanına, sonra tam altına götürün: tek ekseninde örtüşme yetmiyor. Bir ekseninde boşluk varsa çarpışma yoktur.

# AABB ve dört koşul

## AABB (Axis-Aligned Bounding Box)

Eksen hizalı sınır kutusu: kenarları tuvalin x ve y eksenlerine paralel dikdörtgen. `rect()` ile çizilen kutular AABB'dir.

Kutu `rotate` ile döndürülürse kenarlar eksene paralel kalmaz; bu dört koşul tek başına yetmez.

Dördü birden `true` ise çarpışma var. İlk ikisi x eksenini, son ikisi y eksenini içerir.

| Koşul          | Anlamı                              |
|----------------|-------------------------------------|
| $ax < bx + bw$ | A'nın solu, B'nin sağından solda    |
| $ax + aw > bx$ | A'nın sağı, B'nin solundan sağda    |
| $ay < by + bh$ | A'nın üstü, B'nin altından yukarıda |
| $ay + ah > by$ | A'nın altı, B'nin üstünden aşağıda  |

# Testi bir fonksiyona koymak

sketch.js · satır 1-6

Kopyala

Büyüt

```
1 function dikdortgenlerCarpisiyor(ax, ay, aw, ah, bx, by, bw, bh) {  
2     return ax < bx + bw &&  
3         ax + aw > bx &&  
4         ay < by + bh &&  
5         ay + ah > by;  
6 }
```

## Simetriktir

Hangisine A, hangisine B dediğiniz fark etmez: (A, B) ile (B, A) aynı sonucu verir.

## Analitik geometri burada işe yarıyor

Koordinatlar, eşitsizlikler, iki nokta arası uzaklık... Formül ezberlemek değil; "gördüğüm şekil hangi geometrik soruya denk geliyor?" diye sormak yeter.

# Oyuncu engele değince engel kızarsın

sketch.js · satır 21-46

Kopyala

Büyüt

```
21 function draw() {
22   background(28, 32, 48);
23   let dt = deltaTime / 1000;
24
25   if (keyIsDown(RIGHT_ARROW)) {
26     oyuncuX = oyuncuX + 160 * dt;
27   }
28   if (keyIsDown(LEFT_ARROW)) {
29     oyuncuX = oyuncuX - 160 * dt;
30   }
31
32   let carpisti = dikdortgenlerCarpisiyor(
33     oyuncuX, oyuncuY, OYUNCU_BOYUT, OYUNCU_BOYUT,
34     engelX, engelY, ENGEL_W, ENGEL_H
35   );
36
37   fill(100, 200, 255);
38   rect(oyuncuX, oyuncuY, OYUNCU_BOYUT, OYUNCU_BOYUT);
39
40   // carpisti true/false → engel rengi (önceki bölümler: if / else)
41   if (carpisti) {
42     fill(255, 90, 90); // çarpışma var: kırmızımsı
43   } else {
44     fill(120, 120, 130); // çarpışma yok: gri
45   }
46   rect(engelX, engelY, ENGEL_W, ENGEL_H);
```

Çıktı

Çalıştır

Büyüt

Ok: hareket | Çarpışma: false



## SORU

**Az önceki örnekte oyuncu engelin içinden geçebiliyordu. Neden durmadı?**

Test `true` döndü, engel kızardı. Peki oyuncuyu durduran bir satır var mıydı?



# Algılama ile çözümleme ayrı adımlardır

## Algılama

Değdi mi?Yalnızca soruyu sorar: `true` ya da `false`. Az önce yaptığımız buydu.

## Çözümleme

Değdiyse ne olacak?Oyuncuyu dışarı itmek, hızı ters çevirmek, hareketi iptal etmek. Ayrıca yazılır.

Önce algılamayı doğru yazın, sonra tepkiyi ekleyin. Birazdan sekmede ikisini birlikte göreceğiz.

Kısa yazım (isteğe bağlı): üçlü operatör

[Kopyala](#)[Büyüt](#)

```
fill(carpisti ? color(255, 90, 90) : color(120, 120, 130));
```

`koşul ? doğruysa : değilse`. Yeni başlarken açık `if / else` daha okunaklıdır.

## SORU

# Peki iki daire? Dört koşul burada da işe yarar mı?

Dairenin “sol üst köşesi” yok. Elimizde merkez ve yarıçap var. Daire Tıklama oyununda benzer bir soruyu nasıl çözmüştük?



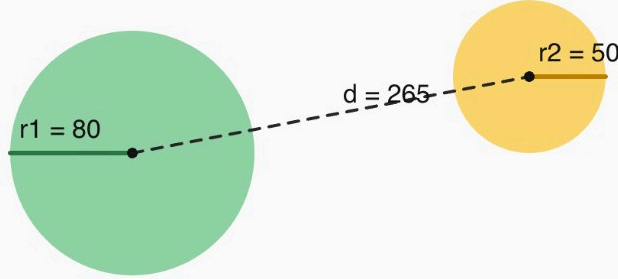
# Merkezler arası mesafe, yarıçapların toplamından küçükse

● Çıktı

Çalıştır

Büyüt

$$d = 265 \geq r1 + r2 = 130 \rightarrow \text{çarpışma yok}$$

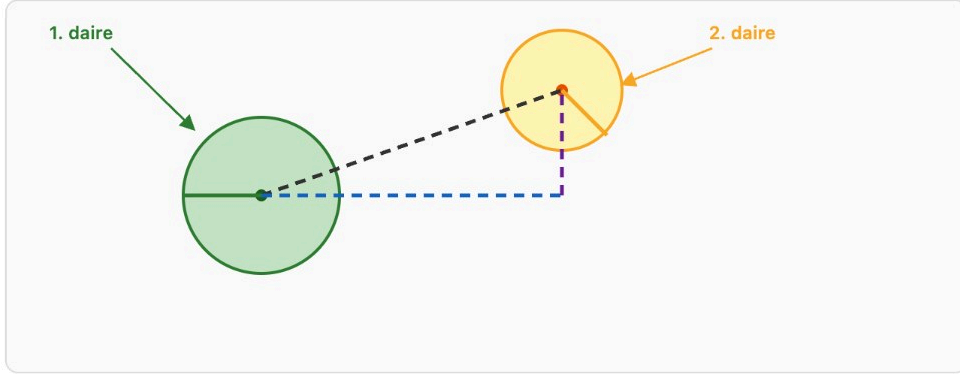


## Daire–daire çarpışması

İki dairenin merkezleri arasındaki mesafe, yarıçapların toplamından küçükse daireler kesişir:  $d < r1 + r2$ .

# Mesafe nereden geliyor? Pisagor

## Daire Çarpışması — Merkez Mesafesi



$$d = \text{sqrt}((x2 - x1) * (x2 - x1) + (y2 - y1) * (y2 - y1))$$

Çarpışma:  $d < r1 + r2$

(mesafe, yarıçaplar toplamından küçükse çarpışma vardır)

### Çizgi Renkleri

- $(x1, y1)$  — 1. dairenin merkezi
- $(x2, y2)$  — 2. dairenin merkezi
- $d$  — merkezler arası mesafe (kesikli)
- $r1$  — 1. dairenin yarıçapı
- $r2$  — 2. dairenin yarıçapı
- $(x2 - x1)$  — yatay fark (kesikli)
- $(y2 - y1)$  — dikey fark (kesikli)

sketch.js

Kopyala

Büyüt

```
d = sqrt((x2 - x1) * (x2 - x1) + (y2 - y1) * (y2 - y1))
```

- Yatay fark  $dx = x2 - x1$ , dikey fark  $dy = y2 - y1$ .
- Dik üçgende hipotenüs:  $d = \text{sqrt}(dx * dx + dy * dy)$ .
- p5.js'te hepsi tek çağrı:  $\text{dist}(x1, y1, x2, y2)$ .

< mi, <= mi?

Tam dışarıdan değmeyi de saymak için `<=` kullanılabilir. Hangisini seçerseniz oyunun her yerinde aynısını kullanın.

# İki daire: ok tuşlarıyla yaklaştırmak

sketch.js · satır 30-41

Kopyala

Büyüt

```
30 let mesafe = dist(daire1X, daire1Y, daire2X, daire2Y);
31 let carpisti = mesafe < R1 + R2; // boolean: dikdörtgen testindeki if ile aynı
32
33 fill(100, 200, 120);
34 ellipse(daire1X, daire1Y, R1 * 2, R1 * 2);
35
36 if (carpisti) {
37   fill(255, 100, 100);
38 } else {
39   fill(255, 220, 80);
40 }
41 ellipse(daire2X, daire2Y, R2 * 2, R2 * 2);
```

Çıktı

Çalıştır

Büyüt

mesafe: 160.3 eşik: 63  
Çarpışma: false  
Ok tuşlarıyla sarı daireyi hareket ettirin



Yeşil daire sabit; sarı daireyi ok tuşlarıyla yaklaştırmak. Bütün test iki satır: mesafeyi ölç, yarıçapların toplamıyla karşılaştırmak.

# Bunu daha önce görmüştük

## Bölüm 1

Daire Tıklama  $\text{dist}(\text{mouseX}, \text{mouseY}, \text{hedefX}, \text{hedefY}) < \text{hedefYaricap}$

Fare bir **nokta**, hedef bir daireydi.

## Bugün

İki daire  $\text{dist}(x1, y1, x2, y2) < r1 + r2$

Aynı sorunun genel hâli. Noktanın yarıçapı sıfır kabul edilirse ikisi aynı.

## SORU

# Top duvara çarptı. Geri dönmesi için hangi sayıyı değiştirmeliyiz?

Top sağa ve aşağı doğru gidiyor. Sağ duvara çarpınca hangi yönü değişmeli, hangisi aynı kalmalı?



# Hız vektörü ve sekme

## Hız vektörü

Nesnenin yatay ( `hizX` ) ve dikey ( `hizY` ) yönlerdeki hızı, piksel/saniye. Güncelleme: `x += hizX * dt`, `y += hizY * dt`.

Sekme: çarpılan duvara **dik** olan bileşenin işaretini ters çevir. Diğerleri aynı kalır.

| Olay                                         | Konum düzeltmesi            | Hız tepkisi               |
|----------------------------------------------|-----------------------------|---------------------------|
| Sol duvar:<br><code>x - r &lt; 0</code>      | <code>x = r</code>          | <code>hizX = -hizX</code> |
| Sağ duvar:<br><code>x + r &gt; width</code>  | <code>x = width - r</code>  | <code>hizX = -hizX</code> |
| Üst duvar:<br><code>y - r &lt; 0</code>      | <code>y = r</code>          | <code>hizY = -hizY</code> |
| Alt duvar:<br><code>y + r &gt; height</code> | <code>y = height - r</code> | <code>hizY = -hizY</code> |

# Duvarlara çarpan top

sketch.js · satır 17-39

Kopyala

Büyüt

```
17 function draw() {  
18   background(22, 28, 42);  
19   let dt = deltaTime / 1000;  
20  
21   topX = topX + hizX * dt;  
22   topY = topY + hizY * dt;  
23  
24   if (topX - YARICAP < 0) {  
25     topX = YARICAP;  
26     hizX = -hizX;  
27   }  
28   if (topX + YARICAP > width) {  
29     topX = width - YARICAP;  
30     hizX = -hizX;  
31   }  
32   if (topY - YARICAP < 0) {  
33     topY = YARICAP;  
34     hizY = -hizY;  
35   }  
36   if (topY + YARICAP > height) {  
37     topY = height - YARICAP;  
38     hizY = -hizY;  
39   }
```

Çıktı

Çalıştır

Büyüt

Duvarlara çarpan top  
hizX: -154.0 hizY: -110.0

Her duvar için iki iş: önce topu içeri it (konum), sonra yönü çevir (hız). Algılama ve çözümleme burada birlikte.

**SORU**

# Konum düzeltmesini silersek ne olur?

sketch.js

Kopyala

Büyüt

```
if (topX + YARICAP > width) {  
  // topX = width - YARICAP; ← silindi  
  hizX = -hizX;  
}
```

Top duvarın biraz içine girmiş hâlde kaldı. Bir sonraki frame’de test ne söyler?

# Top duvara yapışıp titrer



## Önce içeri it, sonra çevir

Top duvarın içinde kalırsa test art arda `true` olur; hız her frame yeniden ters döner.

## Köşede

İki duvara birden değince iki `if` de çalışır; `hizX` ve `hizY` aynı frame'de ters döner. Bu beklenen davranıştır.

# Hangi çift için hangi test?

| Çift                          | Şekiller                       | Test                                     |
|-------------------------------|--------------------------------|------------------------------------------|
| Kare ya da top ↔ tuval kenarı | Dikdörtgen ya da daire ↔ sınır | Kenar <code>if</code> 'leri              |
| Blok ↔ blok                   | İki <code>rect</code> (AABB)   | Dört koşullu örtüşme                     |
| Top ↔ top                     | İki <code>ellipse</code>       | <code>dist</code> < <code>r1 + r2</code> |
| Fare ↔ hedef daire            | Nokta ↔ daire                  | <code>dist</code> < yarıçap              |

Çarpışma hep bir **çift** üzerinden düşünülür: “A ile B değdi mi?” Yeni bir nesne eklediğinizde soru aynı: şekli ne? Sınır mı, AABB mi, `dist` mi?

# Hepsi bir arada: mini tuğla kırma



Breakout'un özü: sınır testi (duvarlar), kutu örtüşmesi (raket ve tuğlalar), sekme ( $vy = -vy$ ) ve tuğlalar için bir dizi ile `splice`.

# Bu konunun üç sınırı

## Algılama ≠ çözümleme

Test yalnızca “değdi mi?” der. Konumu düzeltmek ve hızı çevirmek ayrıca yazılır.

## Test şekle göre seçilir

Dikdörtgen için aralık örtüşmesi, daire için merkezler arası uzaklık. İki modeli karıştırmak yanlış sonuç verir.

## Hızlı nesneler

Tek frame’lik sıçrama, ince bir engelin içinden geçmeye yol açabilir. Küçük adımlar ya da daha büyük hitbox bunu azaltır.

## Processing kaynağı okurken

Jeff Thompson’ın örneklerinde `boolean hit`, `float x`, `void draw()`, `size(600, 400)` görürsünüz. p5.js’te karşılıkları `let hit`, `let x`, `function draw()`, `createCanvas(600, 400)`. Matematik aynı; yalnızca yazım farklı.

# Yapay zekâyla: sekme yönü

🕒 15-20 dk

1. Yatay bir duvara (tavan ya da zemin) çarpan top için  $hizX$  ve  $hizY$  nasıl değişir? Önce kendiniz yazın.
2. Yapay zekâdan aynı senaryoyu doğrulamasını isteyin.
3. Köşeye çarpınca iki bileşenin birden tersine dönmesini bir cümleyle not edin.

# Kod zamanı

 40 dk

1. Kare örneğine yukarı ve aşağı hareketi ekleyin; dört kenarın hepsinde dursun. `constrain` kullanmayın.
2. AABB örneğinde oyuncu engelin **içine giremesin**. İpucu: hareketten önceki konumu saklayın; çarpışma varsa geri dönün.
3. Seken topa yerçekimi ekleyin: her frame `hizY += 600 * dt`. Zeminden her sekişte hızın %15'ini kaybetsin.
4. İki seken top çizin; birbirlerine değdikleri sürece ikisi de kırmızı olsun.
5. **Zorlu:** kendi mini Pong'unuzu yazın. Raket klavyeyle, hareket `deltaTime` ile.

# A = (0, 0, 50, 50), B = (50, 20, 30, 30). Çarpışıyorlar mı?

sketch.js

Kopyala

Büyüt

```
return ax < bx + bw && ax + aw > bx && ay < by + bh && ay + ah > by;
```

☐ A Evet☒ Hayır

`ax + aw > bx` → `50 > 50` yanlış. Kenarlar tam değişiyor ama örtüşmüyor. Kenar kenara durmayı da saymak isteseydik `>=` yazardık.

**Merkezler (100, 100) ve (160, 180); yarıçaplar 40 ve 50.  
Çarpışıyorlar mı?**

**A** Evet

☒ Hayır

$dx = 60, dy = 80 \rightarrow d = 100$ . Yarıçapların toplamı 90.  $100 < 90$  yanlış: aralarında 10 piksel boşluk var.

**Top sağa ve yukarı gidiyor:  $hizX = 200$ ,  $hizY = -150$ . Tavana çarptı. Yeni hızlar?**

**A** -200 ve -150

☒ **200 ve 150**

**C** -200 ve 150

**D** 200 ve -150

Tavan yatay bir duvar: yalnızca dikey bileşen ters döner. Top sağa gitmeye devam eder, artık aşağı doğru.

# Merak edenler için kaynaklar

JEFF THOMPSON

## Collision Detection

Çarpışma testlerini tek tek, etkileşimli örneklerle anlatan ücretsiz kitap. Dikdörtgen, daire ve daire–dikdörtgen bölümlerine bakın.

[jeffreythompson.org/collision-detection](http://jeffreythompson.org/collision-detection)

PATT VİRA

## Collisions Without a Physics Library!

Fizik kütüphanesi kullanmadan çarpışma: bugünkü dersin p5.js ile devamı.

[pattvira.com/coding-tutorials/v/collisions-without-a-physics-library](http://pattvira.com/coding-tutorials/v/collisions-without-a-physics-library)

THE CODING TRAIN

## The Bouncing Ball

Seken top, adım adım.

[thecodingtrain.com](http://thecodingtrain.com) > 3-conditionals > 2-bouncing

THE CODING TRAIN

## 7.6 Object Communication

Birçok dairenin birbirine değip değmediğini kontrol etmek.

[thecodingtrain.com](http://thecodingtrain.com) > 7-arrays > 6-object-communication-1

# Bugün ne öğrendik?

- ✓ Çarpışma algılama her frame sorulan mantıksal bir sorudur: değdi mi?
- ✓ Tuval sınırı: dikdörtgende  $x + w > width$ , dairede  $x + r > width$ .
- ✓ AABB: iki dikdörtgen, hem x hem y ekseninde örtüşüyorsa çarpışır. Dört koşul.
- ✓ İki daire:  $dist < r1 + r2$ .
- ✓ Sekme: önce konumu içeri it, sonra ilgili hız bileşenini ters çevir.
- ✓ Oyunlar süslü çizimi değil, sade bir hitbox'ı test eder.

SIRADAKİ DERS

# Menü, oyun, bitiş

Durum makinesi: oyun şu an hangi ekranda?

Sonraki sunum →

Tüm sunumlar

