10

Tuğla Kırmaca

Bu bölümde dokuzuncu ve son Swift uygulamamızı yazacağız. Bu bölümdeki temel amacımız önceki bölümlerde kullandığımız teknolojileri birleştirerek geniş kapsamlı bir uygulama geliştirmektir.

Geliştireceğimiz uygulama tek kullanıcılı bir oyun olup, çeşitli seviyeleri bulunmaktadır. Oyunun her seviyesinde yer alan tuğlalar dosyadan okunmaktadır. Ekranda yer alan nesneleri çizmek için çeşitli çizim teknolojileri (boyama, dikdörtgen çizme, elips çizme, daire çizme, farklı büyüklüklerde yazı yazma gibi) kullanılmakta, hareket eden nesneler için ise (top ve düşen tuğlalar gibi) zamanlayıcı kullanılarak çizim belirli aralıklarla yenilenmektedir.

Uygulamada oyuncunun kullandığı toplar **Top** sınıfında, topları ekrandan düşürmemek için kullandığı çubuk **Cubuk** sınıfında, zıplayan toplarla vuracağı tuğlalar **Tugla** sınıfında, vurulan tuğlalardan aşağı doğru düşen tuğlalar **DusenTugla** sınıfında, oyunun belirli bir seviyesi **Seviye** sınıfında, oyunun o anki haliyle ilgili bilgiler **Parametre** sınıfında, oyundaki tüm seviyelerle ilgili bilgiler **Oyun** sınıfında, çizilecek ekranla ilgili bilgiler **Ekran** sınıfında ve son olarak ana aktivite de **TuglaKirmaca** sınıfında tutulmaktadır.

10.1 Giriş

Tuğla kırmaca bir oyun uygulaması olup örnek ekran görüntüsü Şekil 10.1'de verilmiştir. Tuğla kırmaca oyununun kuralları aşağıda sıralanmıştır:

- Oyuncu 3 hakla oyuna başlar.
- Oyuncu tüm haklarını kaybettiğinde oyun biter.
- Oyun 10 seviyeden oluşur.

- Oyuncu bir seviyedeki tüm tuğlaları kırdığında bir sonraki seviyeye geçer.
- Oyuncu ekrandaki çubuğa dokunarak sola veya sağa doğru hareket ettirebilir.
- Top, çubuk ile temas ettiğinde seker.
- Top, ekranın sol, sağ ve üst kenarına çarptığında seker.
- Top, ekranın alt kenarında çarptığında yok olur.
- Ekranda hiç top kalmazsa oyuncu bir hak kaybeder.
- Tuğlalardan kahverengi olanı hariç diğer tüm tuğlalar topun tek bir çarpması ile kırılırlar.
- Kahverengi tuğlanın kırılması için topun 2 kere aynı tuğlaya çarpması gerekir.
- Oyuncu kırılan her kahverengi tuğla için 20, diğer renkteki tuğlalar için de 10 puan kazanır.
- Mavi, camgöbeği, turuncu, yeşil, mor, kırmızı ve sarı tuğlalar top çarptığında kırılır ve tuğlanın kırıldığı noktadan aşağıya doğru kırılan tuğla ile aynı renkte eliptik bir top yuvarlanmaya başlar. Eliptik top, çubuk ile temas ettiğinde eliptik topun rengine göre

Mavi: Ekrandaki tüm toplar hızlanır.

Camgöbeği: Ekrandaki tüm toplar yavaşlar.

Turuncu: Çubuk uzar.

Yeşil: Çubuk kısalır.

Mor: Oyuncu bir hak kazanır.

Kırmızı: Oyuncu bir hak kaybeder.

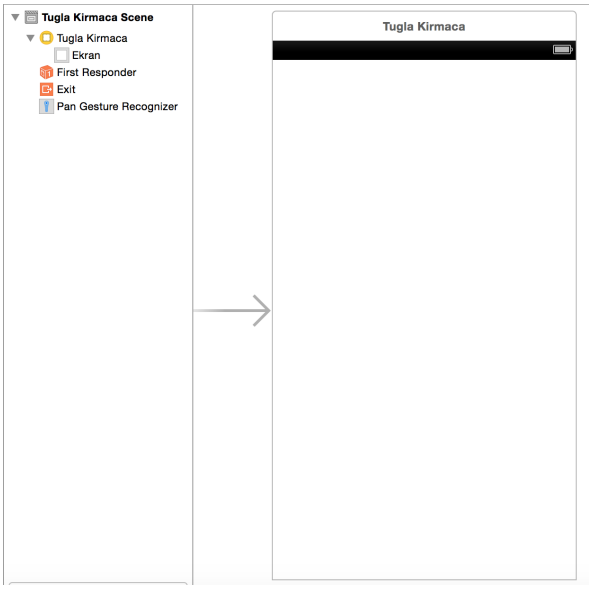
Sarı: Ekstra bir top ekranda görünür ve çubuğun o anda bulunduğu yerden yukarı doğru gider.

Ekranın sol alt köşesinde oyuncunun sahip olduğu her hak için bir top, sağ alt köşesinde oyuncunun toplamış olduğu puan ve ekranın ortasında da oyuncunun bulunduğu seviye gösterilmektedir.

Şekil 10.1: Tuğla Kırmaca uygulamasının ekran görüntüsü

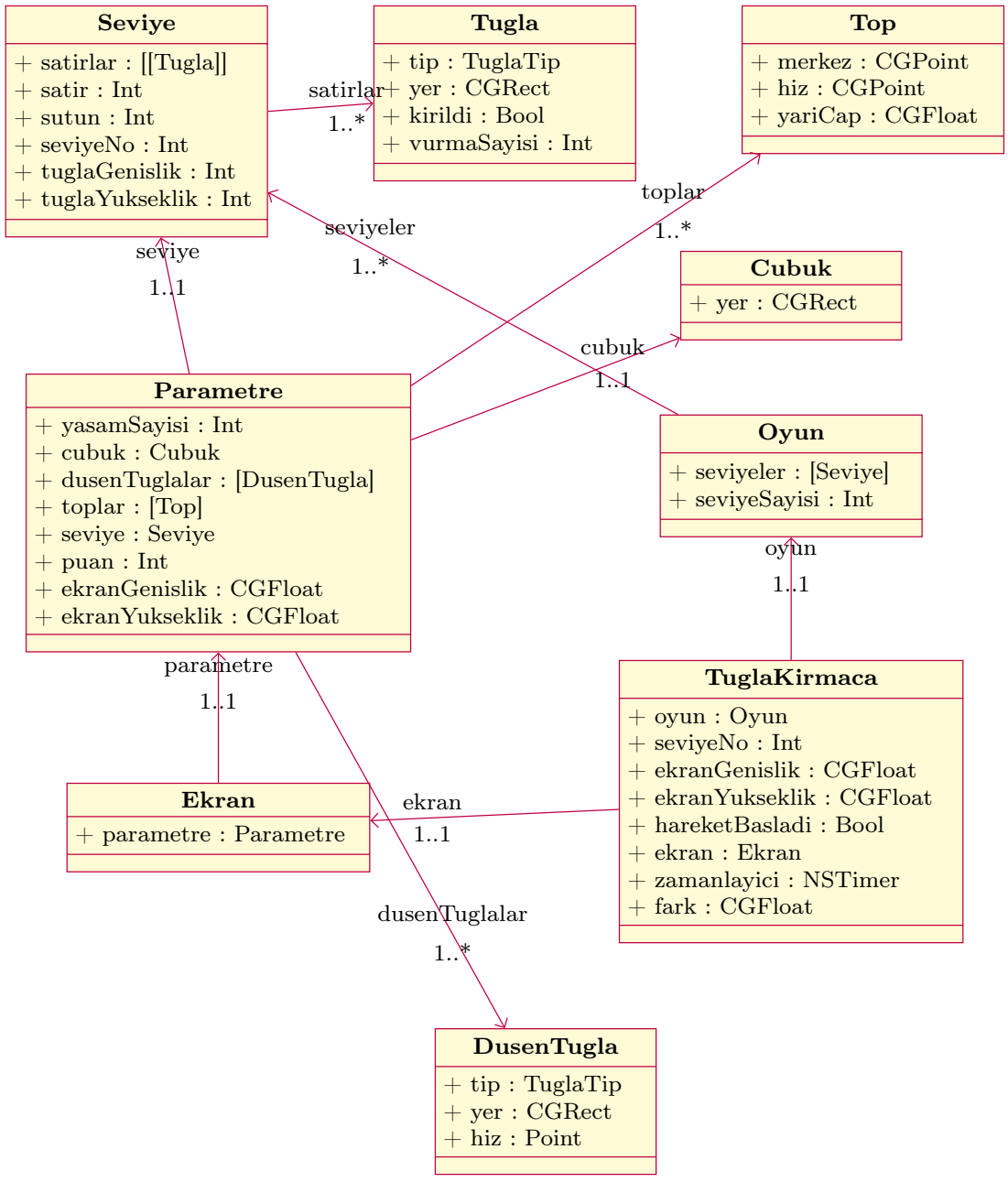
10.2 Arayüz

Satranç Taşları uygulamasının arayüz tasarlama ekranının son hali Şekil 10.2'de verilmiştir. Arayüz program tarafından **Ekran** sınıfı içinde çizileceği için, arayüze aynı adla bir sınıf eklenmiştir. Bunun dışında kullanıcının sürükle bırak komutlarını değerlendirebilmek amacıyla arayüze bir **Pan Gesture Recognizer** eklenmiştir.



Şekil 10.2: Tuğla Kırmaca uygulamasının arayüz tasarlama ekranının son hali

10.3 Sınıf Şeması



Şekil 10.3: Tuğla Kırmaca uygulamasının sınıf şeması

Cubuk, son kullanıcının oyunda sola veya sağa hareket ettirdiği şeritin bilgilerini tutan sınıftır. **yer**, çubuğun koordinatlarını saklar.

Tugla, oyunda son kullanıcının kırarak yeni özellikler kazanabileceği tuğlaları hafızada tutan sınıftır. **tip**, tuğlanın hangi özelliği kazandıracağını; **yer**, tuğlanın koordinatlarını; **kirildi**, tuğlanın kırılıp kırılmadığını, **vurmaSayisi** ise tuğlanın kaç kere vurulduğunu tutmaktadır.

DusenTugla, oyunda aşağıda doğru inen ve son kullanıcının yakalayarak yeni özellikler kazanabileceği tuğlaları hafızada tutan sınıftır. **tip**, tuğlanın hangi özelliği kazandıracağını; **yer**, tuğlanın koordinatlarını; **hiz** ise tuğlanın hangi hızda düştüğünü saklar.

Top, ekranda görülen herhangi bir topun özelliklerini hafızada tutan metottur. Topun merkezi, **merkez** adlı özellikte; hızı, **hiz** adlı özellikte; yarıçapı da **yariCap** adlı özellikte tutulmaktadır.

Seviye, oyunun bir seviyesiyle ilgili bilgileri hafızada tutan sınıftır. Seviyeye ait tuğlalar, **satirlar** adlı iki boyutlu dizide; **satirlar** dizisinin birinci boyutu, **satir** adlı özellikte; ikinci boyutu, **sutun** adlı özellikte; seviyenin numarası, **seviyeNo** adlı özellikte; herhangi bir tuğlanın genişliği ve yüksekliği de **tuglaGenislik** ve **tuglaYukseklik** adlı özelliklerde tutulmaktadır.

Oyun, oyunda var olan seviyeleri dosyadan okuyan ve hafızada tutan sınıftır.

Parametre sınıfında oyuncunun kullanabileceği hak sayısı **yasamSayisi** adlı özellikte; oyuncunun sürükleyeceği çubuk, **cubuk** adlı özellikte; ekranda o anda aşağı düşen tuğlalar, **dusenTuglalar** adlı dizide; ekranda o anda var olan toplar, **toplar** adlı dizide; oyuncunun oynadığı seviye, **seviye** adlı özellikte; oyuncunun o ana kadar topladığı puan, **puan** adlı özellikte; ekranın genişliği ve yüksekliği de **ekranGenislik** ve **ekranYukseklik** adlı özelliklerde tutulmaktadır.

Ekran sınıfı, oynanan oyunun belirli bir anını çizmekle sorumlu olan sınıftır. Oyunla ilgili bilgiler, **parametre** adlı özelikte tutulmaktadır.

TuglaKirmaca, **UIViewController** sınıfını genişleten sınıf olup, uygulamanın ana sınıfıdır. **oyun** adlı özellikte 'seviyeler.txt' dosyasından yüklenen seviyeler tutulmakta, **seviyeNo** o anda oynanan seviyeyi, **ekranGenislik** ve **ekranYukseklik** ise uygulamanın çalıştırıldığı cihazın genişlik ve yüksekliğini göstermektedir. Oyunun başlayıp başlamadığını ise **hareketBasladi** özelliği belirtmektedir.

10.4 Teknolojiler

10.4.1 Çizim metotları

Swift dilinde bir dikdörtgenin içini boyamak için önce dikdörtgensel bölgenin rengi **CGContextSetFillColorWithColor** metoduyla tanımlanır, ardından **CGContextFillRect** metodu ile bu bölge boyanır. **CGContextFillRect** metodunun **rect** parametresi boyanacak dikdörtgensel bölgeyi gösterir.

```
1 func CGContextFillRect(_ c: CGContext?, _ rect: CGRect)
```

Swift dilinde eliptik bir bölgenin içini boyamak için önce eliptik bölgenin rengi **CGContextSetFillColorWithColor** metoduyla tanımlanır, ardından **CGContextFillEllipseInRect** metodu ile bu bölge boyanır. **CGContextSetFillColorWithColor** metodunun **color** parametresi boyanacak rengi gösterir. **CGContextFillEllipseInRect** metodunun **rect** parametresi ise boyanacak eliptik bölgeyi gösterir.

```
1 func CGContextSetFillColorWithColor(_ c: CGContext?, _ color: CGColor?)
2 func  CGContextFillEllipseInRect ( _ c: CGContext?, _ rect: CGRect)
```

10.5 Uygulama Parçaları

10.5.1 Cubuk.swift

Cubuk sınıfının kurucu fonksiyonu Tablo 10.1'de gösterilmiştir. Çubuğun ilk genişliği ve yüksekliği mobil cihazın genişliği ve yüksekliğine bağlı olarak hesaplanır (Satır 2-3).

Tablo 10.1: **Cubuk** *sınıfının özellikleri ve kurucu fonksiyonu*

```
1 init ( ekranGenislik  : CGFloat, ekranYukseklik : CGFloat){
2     yer = CGRectMake(ekranGenislik / 2 − 0.075 ∗ ekranGenislik,
3     ekranYukseklik − 3.5 ∗ 0.05 ∗ ekranGenislik , 0.15 ∗ ekranGenislik , 0.05 ∗ ekranGenislik );
4 }
```

Cubuk sınıfının **buyult**, **kucult** ve **yeniPozisyon** metotları Tablo 10.2'de gösterilmiştir. **buyult** metodunda çubuğun **width** özelliği 1.2 ile çarpılarak çubuğun %20 büyümesi sağlanır (Satır 2). **kucult** metodunda ise çubuğun **width** özelliği 0.8 ile çarpılarak çubuğun %20 küçülmesi sağlanır (Satır 5). **yeniPozisyon** metodunda, dikdörtgenin sol koordinatı, verilen **x** parametresine eşitlenir (Satır 8). Bu şekilde dikdörtgen, **x** koordinatına taşınmış olur.

10.5.2 Tugla.swift

Tugla sınıfının kurucu fonksiyonu Tablo 10.3'te gösterilmiştir.

Tugla sınıfının **vuruldu** metodu Tablo 10.4'te gösterilmiştir. **vuruldu**, herhangi bir top söz konusu tuğlaya vurduğu zaman çağırılan metottur. Önce tuğlanın vurulma sayısı bir artırılır (Satır 2). Eğer tuğlanın tipi ZOR ise (Satır 3), tuğlanın kırılması için 2 kere vurulması gerekir (Satır 5). Eğer tuğlanın tipi ZOR değilse (Satır 7), tuğlanın 1 kere vurulması yeterlidir (Satır 8).

Tablo 10.2: **Cubuk** *sınıfının* **buyult**, **kucult**, **yeniPozisyon** *ve* **yer** *metotları*

```
func buyult(){
    yer.size.width *= 1.2;
}
func kucult(){
    yer.size.width *= 0.8;
}
func yeniPozisyon(x : CGFloat){
    yer.origin.x = x;
}
```

Tablo 10.3: **Tugla** *sınıfının kurucu fonksiyonu*

```
init(tip : TuglaTip, yer : CGRect){
    self.tip = tip;
    self.yer = yer;
     kirildi  = false;
    vurmaSayisi = 0;
}
```

Tablo 10.4: **Tugla** *sınıfının* **vuruldu** *metodu*

```
func vuruldu(){
    vurmaSayisi++;
    if (tip == .ZOR){
        if (vurmaSayisi == 2){
            kirildi  = true;
        }
    } else {
        kirildi  = true;
    }
}
```

10.5.3 DusenTugla.swift

DusenTugla sınıfının kurucu fonksiyonu Tablo 10.5'te gösterilmiştir.

Tablo 10.5: **DusenTugla** *sınıfının kurucu fonksiyonu*

```
init(tip : TuglaTip, yer : CGRect, yukseklik : CGFloat){
    self.tip = tip;
    self.yer = yer;
    hiz = CGPointMake(0, yukseklik / 1000.0);
}
```

DusenTugla sınıfının **hareketEttir** ve **cubuklaTemas** metotları Tablo 10.6'da gösterilmiştir. Tuğlayı hareket ettirmek için **yer**'in x ve y koordinatlarına **hiz**'in x ve y koordinatları eklenir. Bu şekilde düşen tuğla **hiz** yönünde ilerler. **cubuklaTemas**, düşen tuğlanın çubukla temas edip etmediğini kontrol eder (Satır 6-9).

Tablo 10.6: **DusenTugla** *sınıfının* **hareketEttir** *ve* **cubuklaTemas** *metotları*

```
1  func hareketEttir (){
2      yer.origin.x += hiz.x;
3      yer.origin.y += hiz.y;
4  }
5  func cubuklaTemas(cubuk : Cubuk)->Bool{
6      if (yer.origin.x + yer.size.width > cubuk.yer.origin.x &&
7      yer.origin.x < cubuk.yer.origin.x + cubuk.yer.size.width &&
8      yer.origin.y + yer.size.height > cubuk.yer.origin.y &&
9      yer.origin.y < cubuk.yer.origin.y + cubuk.yer.size.height){
10         return true;
11     } else {
12         return false;
13     }
14 }
15 }
```

10.5.4 Top.swift

Top sınıfının kurucu fonksiyonu Tablo 10.7'de gösterilmiştir. Top ilk olarak kuzeybatı yönünde 45 derecelik bir açıyla hareket etmektedir (Satır 2). Topun yarıçapı ekranın %2'si kadardır (Satır 3).

Tablo 10.7: **Top** *sınıfının özellikleri ve kurucu fonksiyonu*

```
1 init(cubukX : CGFloat, cubukY : CGFloat, ekranGenislik : CGFloat){
2     hiz = CGPointMake(-ekranGenislik / 600.0, -ekranGenislik / 600.0);
3     yariCap = ekranGenislik / 50;
4     merkez = CGPointMake(cubukX + yariCap, cubukY - yariCap);
5 }
```

Top sınıfının **sinirlarlaTemas**, **tuglaylaTemas** ve **cubuklaTemas** metotları Tablo 10.8'de gösterilmiştir. **sinirlarlaTemas**, topun ekranın sınırlarına çarpıp çarpmadığı kontrol eden metottur. Eğer top, ekranın sol veya sağ tarafına çarptıysa (Satır 2) seker ve hızının x koordinatı tam yön değiştirir (Satır 3). Eğer top, ekranın üstüne çarptıysa (Satır 5) seker ve hızının y koordinatı tam yön değiştirir (Satır 6). Eğer top, ekranın altına çarptıysa (Satır 8), metot topun ekranın dışına çıktığını saptar ve **true** döndürür (Satır 9). **tuglaylaTemas**, topun verilen bir **tugla**'ya çarpıp çarpmadığını kontrol eden metottur. Eğer top, tuğlaya çarptıysa (Satır 15-18) seker ve hızının y koordinatı tam yön değiştirir (Satır 19).

cubuklaTemas, topun çubuğa çarpıp çarpmadığını kontrol eden metottur. Eğer top çubuğa çarptıysa (Satır 26-29) seker ve hızının y koordinatı tam yön değiştirir (Satır 30).

Tablo 10.8: **Top** *sınıfının* **sinirlarlaTemas**, **tuglaylaTemas** *ve* **cubuklaTemas** *metotları*

```
1  func sinirlarlaTemas ( ekranGenislik : CGFloat, ekranYukseklik : CGFloat)->Bool{
2      if (merkez.x < 0 || merkez.x > ekranGenislik){
3          hiz.x *= -1;
4      }
5      if (merkez.y < 0){
6          hiz.y *= -1;
7      }
8      if (merkez.y > ekranYukseklik){
9          return true;
10     } else {
11         return false;
12     }
13 }
14 func tuglaylaTemas(tugla : Tugla)->Bool{
15     if (merkez.x + yariCap > tugla.yer.origin.x &&
16     merkez.x - yariCap < tugla.yer.origin.x + tugla.yer.size.width &&
17     merkez.y + yariCap > tugla.yer.origin.y &&
18     merkez.y - yariCap < tugla.yer.origin.y + tugla.yer.size.height){
19         hiz.y *= -1;
20         return true;
21     } else {
22         return false;
23     }
24 }
25 func cubuklaTemas(cubuk : Cubuk)->Bool{
26     if (merkez.x + yariCap > cubuk.yer.origin.x &&
27     merkez.x - yariCap < cubuk.yer.origin.x + cubuk.yer.size.width &&
28     merkez.y + yariCap > cubuk.yer.origin.y &&
29     merkez.y - yariCap < cubuk.yer.origin.y + cubuk.yer.size.height){
30         hiz.y *= -1;
31         return true;
32     } else {
33         return false;
34     }
35 }
```

Top sınıfının **hareketEttir**, **hizlandir** ve **yavaslat** metotları Tablo 10.9'da gösterilmiştir. **hareketEttir**, topun **merkez**ine **hiz**ının x ve y koordinatlarını ekler (Satır 2-3). **hizlandir**, topun hızının x ve y koordinatlarını 1.25 katsayısı ile çarparak topu %25 hızlandırır (Satır 6-7). **yavaslat**, topun hızının x ve y koordinatlarını 0.8 katsayısı ile çarparak topu %20 yavaşlatır (Satır 10-11).

Tablo 10.9: **Top** *sınıfının* **hareketEttir**, **hizlandir** *ve* **yavaslat** *metotları*

```
1  func hareketEttir (){
2      merkez.x += hiz.x;
3      merkez.y += hiz.y;
4  }
5  func hizlandir (){
6      hiz.x *= 1.25;
7      hiz.y *= 1.25;
8  }
9  func yavaslat (){
10     hiz.x *= 0.8;
11     hiz.y *= 0.8;
12 }
```

10.5.5 Seviye.swift

Seviye sınıfının kurucu fonksiyonu Tablo 10.10'da gösterilmiştir.

Tablo 10.10: **Seviye** *sınıfının kurucu fonksiyonu*

```
1  init ( satir : Int, sutun : Int, genislik : CGFloat, seviyeNo : Int){
2      self.satir = satir;
3      self.sutun = sutun;
4      self.seviyeNo = seviyeNo;
5      tuglaGenislik = genislik / CGFloat(sutun);
6      tuglaYukseklik = 0.7 * tuglaGenislik;
7  }
```

Seviye sınıfının **bittiMi** ve **tugla** metotları Tablo 10.11'de gösterilmiştir. **bittiMi**, seviyenin bitip bitmediğini kontrol eden metottur. Seviyedeki her tuğla için (Satır 6), o tuğlanın kırılıp kırılmadığı kontrol edilir (Satır 7). Eğer herhangi bir tuğla kırılmadıysa seviye bitmemiştir (Satır 8). Tüm tuğlalar kırıldıysa seviye bitmiştir; metot **true** döndürür (Satır 12). **tugla** verilen **satir** ve **sutun**'daki tuğlayı döndürür (Satır 15).

Seviye sınıfının **satirAyarla** metodu Tablo 10.12'de gösterilmiştir. **satirAyarla**, tuğla dizisinin bir satırını bir katardan ('1111111111' gibi) üreten metottur. Katardaki her i. karakter için (Satır 6), yeni bir dikdörtgen yaratılır (Satır 7-8). Katardaki karakter,

1 NORMAL tuğla yaratılır (Satır 11).

2 Kırılması ZOR tuğla yaratılır (Satır 13).

3 Kırıldığında, topları hızlandırabilecek tuğla yaratılır (Satır 15).

4 Kırıldığında, topları yavaşlatabilecek tuğla yaratılır (Satır 17).

Tablo 10.11: **Seviye** *sınıfının* **bittiMi** *ve* **tugla** *metotları*

```
1  func bittiMi()->Bool{
2      var tugla : Tugla;
3      var i, j : Int;
4      for (i = 0; i < satir; i++){
5          for (j = 0; j < sutun; j++){
6              tugla = satirlar[i][j];
7              if (!tugla.kirildi ){
8                  return false;
9              }
10         }
11     }
12     return true;
13 }
14 func tugla(satir : Int, sutun : Int)->Tugla{
15     return satirlar[satir][sutun];
16 }
```

5 Kırıldığında, çubuğu büyültebilecek tuğla yaratılır (Satır 19).

6 Kırıldığında, çubuğu küçültebilecek tuğla yaratılır (Satır 21).

7 Kırıldığında, oyuncuya yeni bir hak verebilecek tuğla yaratılır (Satır 23).

8 Kırıldığında, oyuncunun bir hakkını azaltabilecek tuğla yaratılır (Satır 25).

9 Kırıldığında, yeni bir top üretebilecek tuğla yaratılır (Satır 27).

10.5.6 Oyun.swift

Oyun sınıfı Tablo 10.13'te gösterilmiştir. Oyundaki seviyeler 'seviyeler.txt' adlı dosyada tutulmaktadır. Her seviye dosyasında önce oyunda kaç seviye olduğu belirtilmekte ardından 1. seviye ile ilgili bilgiler, 2. seviye ile ilgili bilgiler vs. gösterilmektedir. Her seviye için ise, bu seviyedeki tuğlaların kaç satır ve sütundan oluştuğu belirtilmekte, ardından sırasıyla 1. satırdaki, 2. satırdaki, ... tuğlalar kodlanarak gösterilmektedir. Toplam 10 farklı tuğla olduğundan, tuğlalar 0 ile 9 arasındaki rakamlarla kodlanmıştır.

Örnek bir 'seviyeler.txt' dosyası aşağıda verilmiştir. Bu oyunda 3 seviye bulunmakta olup, birinci seviyede 1 satır 10 sütun, ikinci seviyede 2 satır 10 sütun, üçüncü seviyede ise 3 satır 10 sütun tuğla bulunmaktadır. Birinci seviyedeki tüm tuğlalar 9 tipindedir. İkinci seviyenin birinci satırındaki tüm tuğlalar 1 tipinde, ikinci satırdaki tuğlalar ise 9 tipindedir. Üçüncü seviyenin birinci satırındaki tüm tuğlalar 9 tipinde, ikinci satırındaki tüm tuğlalar 2 tipinde, üçüncü satırındaki tuğlalar ise yine 9 tipindedir.

Tablo 10.12: **Seviye** *sınıfının* **satirAyarla** *metodu*

```
1  func satirAyarla(satirNo : Int, satirBilgi : String){
2      var i : Int;
3      var yer: CGRect;
4      var x, y : CGFloat;
5      var satirEklenen : [Tugla] = [];
6      for (i = 0; i < count(satirBilgi.utf16); i++){
7          yer = CGRectMake(CGFloat(i) * tuglaGenislik, CGFloat(satirNo) * tuglaYukseklik,
8           tuglaGenislik, tuglaYukseklik);
9          switch (String(Array(satirBilgi)[i])){
10             case "1":
11                 satirEklenen.append(Tugla(tip: .NORMAL, yer: yer));
12             case "2":
13                 satirEklenen.append(Tugla(tip: .ZOR, yer: yer));
14             case "3":
15                 satirEklenen.append(Tugla(tip: .HIZLI, yer: yer));
16             case "4":
17                 satirEklenen.append(Tugla(tip: .YAVAS, yer: yer));
18             case "5":
19                 satirEklenen.append(Tugla(tip: .BUYUK, yer: yer));
20             case "6":
21                 satirEklenen.append(Tugla(tip: .KUCUK, yer: yer));
22             case "7":
23                 satirEklenen.append(Tugla(tip: .YASAM, yer: yer));
24             case "8":
25                 satirEklenen.append(Tugla(tip: .OLUM, yer: yer));
26             case "9":
27                 satirEklenen.append(Tugla(tip: .COKTOP, yer: yer));
28             default:
29                 break;
30         }
31     }
32     satirlar.append(satirEklenen);
33 }
```

```
3
1 10
9999999999
2 10
1111111111
9999999999
3 10
9999999999
2222222222
9999999999
```

Önce verinin okunacağı dosya 'seviyeler.txt' bir **NSString** yapısına dönüştürülür (Satır 12). Bu aşamada dosya ismi **pathForResource** parametresi, dosya uzantısı ise **ofType** parametresi ile ifade edilir. Sonra, bu **NSString** yapısı ile

Tablo 10.13: **Oyun** *sınıfı*

```
1  init(dosyaAd : String, genislik : CGFloat){
2      var dosyaAdi, dosyaIcerik : String;
3      var ayirici : NSScanner;
4      var i , j : Int;
5      var satir : Int = 0;
6      var sutun : Int = 0;
7      var seviye : Seviye;
8      var satirBilgi : NSString?;
9      dosyaAdi = NSBundle.mainBundle().pathForResource(dosyaAd, ofType: "txt")!;
10     do{
11        dosyaIcerik = try String(contentsOfFile:dosyaAdi, encoding:NSUTF16StringEncoding);
12     }
13    catch{
14        dosyaIcerik = "";
15     }
16      ayirici = NSScanner(string: dosyaIcerik );
17      ayirici.scanInteger(&seviyeSayisi );
18     for (i = 0; i < seviyeSayisi ; i++){
19          ayirici.scanInteger(&satir );
20          ayirici.scanInteger(&sutun);
21          seviye = Seviye( satir : satir , sutun : sutun, genislik : genislik , seviyeNo : i );
22          for (j = 0; j < satir; j++){
23              ayirici .scanUpToString("\n", intoString : &satirBilgi );
24              seviye.satirAyarla (j, satirBilgi : satirBilgi !);
25          }
26          seviyeler .append(seviye);
27     }
28 }
29 func seviye(pozisyon : Int)->Seviye{
30     return seviyeler [pozisyon];
31 }
```

dosya tek bir kataran içine okunur (Satır 13). Bu aşamada dosyanın adı **contentsOfFile** parametresi ile, dosyanın dil kodlaması ise **encoding** parametresi ile belirtilir. Ardından, bu **NSString** yapısı bir ayırıcıya (**NSScanner**) parametre olarak gönderilerek işlenmeye hazırlanır (Satır 14).

Dosyadan veri okunmaya hazırlandıktan sonra, seviye sayısı okunmakta ve **seviyeSayisi** özelliğinde tutulmaktadır (Satır 15). Her seviye için (Satır 16),

- Kaç satır ve sütun tuğla olduğu okunmaktadır (Satır 17-18).
- Yeni bir seviye için hafızada yer açılmakta (Satır 19) ve her satır için,
  - O satırdaki tuğlaların tipleri bir katar halinde okunmaktadır (Satır 21).
  - Okunan tip bilgileri bir satıra çevrilmekte ve oyuna eklenmektedir (Satır 22).

10.5.7 Parametre.swift

Parametre sınıfının kurucu ve alıcı fonksiyonları Tablo 10.14'te gösterilmiştir. Oyunun başında oyuncunun 3 hakkı (Satır 4) ve 0 puanı bulunmaktadır (Satır 5).

Tablo 10.14: **Parametre** *sınıfının özellikleri ve kurucu fonksiyonu*

```
1  init ( genislik  : CGFloat, yukseklik  : CGFloat, seviye  : Seviye){
2      ekranGenislik  = genislik ;
3      ekranYukseklik  = yukseklik;
4      yasamSayisi  = 3;
5      puan = 0;
6      cubuk = Cubuk(ekranGenislik: ekranGenislik , ekranYukseklik  : ekranYukseklik );
7      self . seviye  = seviye;
8      toplar .append(Top(cubukX: cubuk.yer.origin.x, cubukY: cubuk.yer. origin .y,
9      ekranGenislik :  ekranGenislik ));
10 }
```

Parametre sınıfının **yeniSeviye**, **topSayisi**, **top**, **dusenTuglaSayisi** ve **dusenTugla** metotları Tablo 10.15'te gösterilmiştir. **yeniSeviye**, oyunda yeni bir seviyeye başlandığında çağırılan metottur. Çubuğu ekranın ortasına getirir (Satır 3), ekrandaki topları temizleyip (Satır 4), tek bir topu çubuğun sol üst köşesine konuşlandırır (Satır 5). **topSayisi**, ekrandaki topların (Satır 9), **dusenTuglaSayisi** ise düşen tuğlaların (Satır 15) sayısını döndürür. **top**, **toplar** dizisinin belirli bir **pozisyon**'undaki topu (Satır 12), **dusenTugla** ise **dusenTuglalar** dizisinin belirli bir **pozisyon**'undaki düşen tuğlayı (Satır 18) döndürür.

Tablo 10.15: **Parametre** *sınıfının* **yeniSeviye**, **topSayisi**, **top**, **dusenTuglaSayisi** *ve* **dusenTugla** *metotları*

```
1  func  yeniSeviye ( seviye  : Seviye){
2      self . seviye  = seviye;
3      cubuk = Cubuk(ekranGenislik: ekranGenislik , ekranYukseklik  : ekranYukseklik );
4      toplar .removeAll(keepCapacity:  false );
5      toplar .append(Top(cubukX: cubuk.yer.origin.x, cubukY: cubuk.yer. origin .y,
6      ekranGenislik :  ekranGenislik ));
7  }
8  func  topSayisi ()->Int{
9      return  toplar .count;
10 }
11 func top(pozisyon  : Int)->Top{
12     return  toplar [pozisyon ];
13 }
14 func  dusenTuglaSayisi()->Int{
15     return  dusenTuglalar .count;
16 }
17 func dusenTugla(pozisyon  : Int)->DusenTugla{
18     return  dusenTuglalar [pozisyon ];
19 }
```

Parametre sınıfının **topSinirlarlaTemasKontrol** metodu Tablo 10.16'da gösterilmiştir. Önce topun ekranın dışına çıkıp çıkmadığı kontrol edilir (Satır 4). Eğer top ekranın dışına çıktıysa, **toplar** dizisinden silinir (Satır 5) ve ekranda hiç top kalıp kalmadığına bakılır (Satır 6). Eğer ekranda hiç top kalmadıysa, oyuncunun hakları bir azaltılır (Satır 7). Bunun sonucunda eğer oyuncunun başka hakkı kaldıysa (Satır 8), yeni bir top çubuğun sol üst köşesine gelecek şekilde yerleştirilir (Satır 9-10). Oyuncunun başka hakkı kalmadıysa metot **false** döndürür (Satır 13).

Tablo 10.16: **Parametre** *sınıfının* **topSinirlarlaTemasKontrol** *metodu*

```
1  func topSinirlarlaTemasKontrol(top : Top, index : Int)->Bool{
2      var yeniTop : Top;
3      var i : Int;
4      if (top.sinirlarlaTemas(ekranGenislik, ekranYukseklik : ekranYukseklik)){
5          toplar.removeAtIndex(index);
6          if (toplar.count == 0){
7              yasamSayisi--;
8              if (yasamSayisi > 0){
9                  yeniTop = Top(cubukX : cubuk.yer.origin.x, cubukY : cubuk.yer.origin.y,
10                 ekranGenislik : ekranGenislik);
11                 toplar.append(yeniTop);
12             } else {
13                 return false;
14             }
15         }
16     }
17     return true;
18 }
```

Parametre sınıfının **topTuglaylaTemasKontrol** metodu Tablo 10.17'de gösterilmiştir. Oyunun bu seviyesinde yer alan her tuğla için (Satır 7), bu tuğla eğer kırılmadıysa ve top tuğlayla temas ettiyse (Satır 8), söz konusu tuğla vurulmuştur (Satır 9). Eğer tuğla vurulduğunda kırıldıysa (Satır 10), tuğlanın tipine göre 10 veya 20 puan oyuncunun hanesine yazılır. Eğer tuğla ZOR tipinde bir tuğlaysa (Satır 13) 20 puan (Satır 14), başka bir tuğlaysa (Satır 11) 10 puan (Satır 12) oyuncunun hanesine eklenir. Kırılan tuğla özel bir tuğlaysa (ZOR veya NORMAL değilse), yeni bir düşen tuğla yaratılır (Satır 17-18) ve düşen tuğlalar listesine eklenir (Satır 19). Son tuğla da kırıldıysa (Satır 21), seviye biter (Satır 22) ve metot **true** döndürür.

Parametre sınıfının **topCubuklaTemasKontrol**, **elleCubukTemas** ve **cubukYeniX** metotları Tablo 10.18'de gösterilmiştir. **topCubuklaTemasKontrol**, topun çubukla temas edip etmediğini kontrol eden metottur (Satır 2). **elleCubukTemas**, son kullanıcının bastığı yerle çubuğun temas edip etmediğini kontrol eden metottur (Satır 5-6). **cubukYeniX** ise, çubuğun verilen **x** noktasına gidip gitmeyeceğine bakan (Satır 13) ve gidebiliyorsa çubuğu o pozisyona gönderen metottur (Satır 14).

Tablo 10.17: **Parametre** *sınıfının* **topTuglaylaTemasKontrol** *metodu*

```
1  func topTuglaylaTemasKontrol(top : Top)->Bool{
2      var tugla : Tugla;
3      var dusenTugla : DusenTugla;
4      var i, j : Int;
5      for (i = 0; i < seviye.satir; i++){
6          for (j = 0; j < seviye.sutun; j++){
7              tugla = seviye.tugla(i, sutun : j);
8              if (!tugla.kirildi && top.tuglaylaTemas(tugla)){
9                  tugla.vuruldu();
10                 if (tugla.kirildi ){
11                     if (tugla.tip != .ZOR){
12                         puan += 10;
13                     }else{
14                         puan += 20;
15                     }
16                     if (tugla.tip != .NORMAL && tugla.tip != .ZOR){
17                         dusenTugla = DusenTugla(tip: tugla.tip, yer : tugla.yer,
18                         yukseklik : ekranYukseklik);
19                         dusenTuglalar.append(dusenTugla);
20                     }
21                     if (seviye.bittiMi()){
22                         return true;
23                     }
24                 }
25             }
26         }
27     }
28     return false;
29 }
```

Parametre sınıfının **dusenTuglaCubuklaTemasKontrol** metodu Tablo 10.19'da gösterilmiştir. Düşen her tuğlanın (Satır 6) çubukla temas edip etmediği kontrol edilir (Satır 7). Eğer düşen tuğla çubukla temas ettiyse, tuğlanın tipine göre,

- HIZLI: Ekranda var olan tüm toplar (Satır 10) hızlandırılır (Satır 11).
- YAVAS: Ekranda var olan tüm toplar (Satır 14) yavaşlatılır (Satır 15).
- BUYUK: Çubuk büyültülür (Satır 18).
- KUCUK: Çubuk küçültülür (Satır 20).
- YASAM: Oyuncunun hak sayısı bir artırılır (Satır 22).
- OLUM: Oyuncunun hak sayısı bir azaltılır (Satır 24). Oyuncunun hak sayısı 0 kaldıysa, metot **false** döndürür.
- COKTOP: Yeni bir top çubuğun sol üst köşesine gelecek şekilde yaratılır (Satır 29-30) ve top listesine eklenir (Satır 31).

Tablo 10.18: **Parametre** *sınıfının* **topCubuklaTemasKontrol**, **elleCubukTemas** *ve* **cubukYeniX** *metotları*

```
1  func topCubuklaTemasKontrol(top : Top)->Bool{
2      return top.cubuklaTemas(cubuk);
3  }
4  func elleCubukTemas(x : CGFloat, y : CGFloat)->Bool{
5      if (x > cubuk.yer.origin.x && x < cubuk.yer.origin.x + cubuk.yer.size.width &&
6      y > cubuk.yer.origin.y && y < cubuk.yer.origin.y + cubuk.yer.size.height){
7          return true;
8      } else {
9          return false;
10     }
11 }
12 func cubukYeniX(x : CGFloat){
13     if (x > 0 && x < ekranGenislik - cubuk.yer.size.width){
14         cubuk.yeniPozisyon(x);
15     }
16 }
```

10.5.8 Ekran.swift

Ekran sınıfının **tuglaRenk** metodu Tablo 10.20'de gösterilmiştir. **tuglaRenk**, gerek çizilen gerekse düşen tuğlaların renklerini belirtmek için kullanılmaktadır. Normal tuğla siyah (Satır 4), kırılması zor tuğla mor (Satır 6), top(lar)ın hızlanmasını sağlayan tuğla mavi (Satır 8), yavaşlamasını sağlayan tuğla camgöbeği (Satır 10), çubuğun büyümesini sağlayan tuğla turuncu (Satır 12), küçülmesini sağlayan tuğla yeşil (Satır 14), oyuncuya fazladan bir hak veren tuğla mor (Satır 16), oyuncunun bir hakkını alan tuğla kırmızı (Satır 18) ve son olarak oyundaki top sayısını bir artıran tuğla da sarıdır (Satır 20).

Oyunun belirli bir anını çizebilmek için Ekran sınıfı **UIView** sınıfını genişletmiş, ve **drawRect** metodununun da üstüne yazmıştır (Tablo 10.21 ve 10.22).

Ekranda sırasıyla aşağıdaki şekiller çizilmektedir:

- Tuğlalar: Oyunun seviyesine göre, tuğlaların bulunduğu satır ve sütun sayısı belirlenmekte (Satır 16-17), her olası satır ve sütun için, o pozisyonda bulunan tuğlanın (Satır 18) kırılıp kırılmadığına bakılmaktadır (Satır 19). Eğer tuğla kırılmadıysa, çizilecek tuğla **tuglaRenk** metoduyla belirlenmekte (Satır 20), ardından tuğlanın içi o renkte boyanmakta (Satır 21-22), tuğlanın dış kenarı ise gri renkte çizilmektedir (Satır 23-24).

- Çubuk: Çubuk, koyu gri renkte çizilmektedir (Satır 30). Önce tuğlanın içi koyu gri renkle boyanmakta (Satır 31), ardından dış kenarı gri renkle çizilmektedir (Satır 32-33).

- Toplar: Toplar, pembe renkle çizilmektedir (Satır 39). Oyunun içindeki her top için (Satır 35), topu çevreleyen dikdörtgen belirlenmekte (Satır 37-38), ardından bu dikdörtgenin içine daire çizilmektedir (Satır 40).

Tablo 10.19: **Parametre** *sınıfının* **dusenTuglaCubuklaTemasKontrol** *metodu*

```
1  func dusenTuglaCubuklaTemasKontrol()->Bool{
2      var yeniTop : Top;
3      var dusenTugla : DusenTugla;
4      var i : Int;
5      for (i = 0; i < dusenTuglalar.count; i++){
6          dusenTugla = dusenTuglalar[i];
7          if (dusenTugla.cubuklaTemas(cubuk)){
8              switch (dusenTugla.tip){
9                  case .HIZLI:
10                     for top in toplar{
11                         top. hizlandir ();
12                     }
13                 case .YAVAS:
14                     for top in toplar{
15                         top. yavaslat ();
16                     }
17                 case .BUYUK:
18                     cubuk.buyult ();
19                 case .KUCUK:
20                     cubuk.kucult ();
21                 case .YASAM:
22                     yasamSayisi++;
23                 case .OLUM:
24                     yasamSayisi--;
25                     if (yasamSayisi == 0){
26                         return false;
27                     }
28                 case .COKTOP:
29                     yeniTop = Top(cubukX : cubuk.yer.origin.x, cubukY : cubuk.yer. origin .y,
30                     ekranGenislik : ekranGenislik );
31                     toplar .append(yeniTop);
32                 default:
33                     break;
34             }
35             dusenTuglalar.removeAtIndex(i);
36             break;
37         }
38     }
39     return true;
40 }
```

- Düşen Tuğlalar: Her düşen tuğla için (Satır 42), düşen tuğla **tuglaRenk** metoduyla belirlenmekte (Satır 44), ardından tuğla bu renkle eliptik olarak çizilmektedir (Satır 46).

- Haklar: Oyuncunun sahip olduğu haklar küçük daireler biçiminde sol alt köşeye çizilmektedir. Oyuncunun her hakkı için (Satır 49), hakka karşılık gelen daireyi çevreleyen dikdörtgen belirlenmekte (Satır 50-51), ardından bu dikdörtgenin içine pembe renkli daire çizilmektedir (Satır 53).

Tablo 10.20: **Ekran** *sınıfının* **tuglaRenk** *metodu*

```
1  func tuglaRenk(tip : TuglaTip)->UIColor{
2      switch (tip){
3          case .NORMAL:
4              return UIColor.blackColor();
5          case .ZOR:
6              return UIColor.brownColor();
7          case .HIZLI:
8              return UIColor.blueColor();
9          case .YAVAS:
10             return UIColor.cyanColor();
11         case .BUYUK:
12             return UIColor.orangeColor();
13         case .KUCUK:
14             return UIColor.greenColor();
15         case .YASAM:
16             return UIColor.magentaColor();
17         case .OLUM:
18             return UIColor.redColor();
19         case .COKTOP:
20             return UIColor.yellowColor();
21     }
22 }
```

- Puan: Oyuncunun bu ana kadar kazandığı puan ekranın sağ alt köşesine yazılmaktadır. Puan yazısının büyüklüğü ekran genişliğinin %4'ü olacak şekilde ayarlanmaktadır (Satır 56-59). Ardından pembe renkle oyuncunun kazandığı puan ekranın sağ alt köşesine yazılmaktadır (Satır 60-62).

- Seviye: Oyuncunun oynadığı seviyenin yazı büyüklüğü ekran genişliğinin %8'i olacak şekilde ayarlanmaktadır (Satır 63-66). Ardından pembe renkle seviye numarası ekranı ortalayacak biçimde yazılmaktadır (Satır 67-69).

10.5.9 TuglaKirmaca.swift

TuglaKirmaca sınıfının **viewDidLoad** metodu Tablo 10.23'te gösterilmiştir. Ekranın genişliği ve yüksekliği **UIScreen** sınıfının **bounds** özelliğiyle elde edilmektedir (Satır 4-5). Ekranın her 10 milisaniyede bir tekrar çizilebilmesi için, **zamanlayici** adlı bir **NSTimer** tanımlanmış olup (Satır 11), her 10 milisaniyede bu sınıfa ait **onTimer** metodu çağırılmaktadır (Satır 12).

TuglaKirmaca sınıfının **onTimer** metodu Tablo 10.24'te gösterilmiştir. **onTimer**, her 10 milisaniyede bir çağırılan, ekranda hareket eden nesnelerin yerlerini değiştiren ve ekranı yeniden çizen metottur. Ekranda yerleri değişen iki tip nesne vardır. Bunlar,

- Toplar: Her top (Satır 6), hareket ettirilir (Satır 7) ve çubukla teması kontrol edilir (Satır 8). Eğer top bir tuğlayla temas ediyorsa ve temas eden tuğla

Tablo 10.21: **Ekran** *sınıfının* **drawRect** *metodu (1)*

```
1  override func drawRect(rect:CGRect){
2      var i, j: Int;
3      var context: CGContextRef;
4      var alan : CGRect;
5      var seviye : Seviye;
6      var cizilenTugla : Tugla;
7      var cubuk : Cubuk;
8      var top : Top;
9      var dusenTugla : DusenTugla;
10     var renk : UIColor;
11     var fontBuyukluk : CGFloat;
12     var yaziBuyukluk: CGSize;
13     var puan : String;
14     context = UIGraphicsGetCurrentContext();
15     seviye = parametre!.seviye;
16     for (i = 0; i < seviye.satir; i++){
17         for (j = 0; j < seviye.sutun; j++){
18             cizilenTugla = seviye.tugla(i, sutun:j);
19             if (cizilenTugla.kirildi){
20                 renk = tuglaRenk(cizilenTugla.tip);
21                 CGContextSetFillColorWithColor(context, renk.CGColor);
22                 CGContextFillRect(context, cizilenTugla.yer);
23                 CGContextSetStrokeColorWithColor(context, UIColor.grayColor().CGColor);
24                 CGContextAddRect(context, cizilenTugla.yer);
25                 CGContextStrokePath(context);
26             }
27         }
28     }
29     cubuk = parametre!.cubuk;
30     CGContextSetFillColorWithColor(context, UIColor.darkGrayColor().CGColor);
31     CGContextFillRect(context, cubuk.yer);
32     CGContextSetStrokeColorWithColor(context, UIColor.grayColor().CGColor);
33     CGContextAddRect(context, cubuk.yer);
34     CGContextStrokePath(context);
```

son tuğlaysa (Satır 9), oynanan seviye sona ermiştir. Oyunda yeni seviyeye geçilir (Satır 11). Eğer top ekranın dışına çıktıysa ve sınırın dışına çıkan top son topsa (Satır 14), oyun sona erer (Satır 15).

- Düşen Tuğlalar: Her düşen tuğla (Satır 19), hareket ettirilir (Satır 20) ve çubukla teması kontrol edilir (Satır 21). Eğer düşen tuğla çubukla temas ediyorsa ve bu temasın sonucunda oyuncu son hakkını da kaybederse oyun sona erer (Satır 22).

Son kullanıcı ekrana dokunduğunda, ekranda bir nesneyi sürüklediğinde ve bıraktığında **Pan Gesture Recognizer**'a bağladığımız **hareketEttir** metodu çağırılır (Tablo 10.25). Son kullanıcı,

- Ekrana dokunduğunda (Satır 4), dokunduğu yerde çubuk olup olmadığı

Tablo 10.22: **Ekran** *sınıfının* **drawRect** *metodu (2)*

```
35      for (i = 0; i < parametre!.topSayisi(); i++){
36          top = parametre!.top(i);
37          alan = CGRectMake(top.merkez.x - top.yariCap, top.merkez.y - top.yariCap,
38          2 * top.yariCap, 2 * top.yariCap);
39          CGContextSetFillColorWithColor(context, UIColor.purpleColor().CGColor);
40          CGContextFillEllipseInRect(context, alan);
41      }
42      for (i = 0; i < parametre!.dusenTuglaSayisi(); i++){
43          dusenTugla = parametre!.dusenTugla(i);
44          renk = tuglaRenk(dusenTugla.tip);
45          CGContextSetFillColorWithColor(context, renk.CGColor);
46          CGContextFillEllipseInRect(context, dusenTugla.yer);
47      }
48      top = parametre!.top(0);
49      for (i = 0; i < parametre!.yasamSayisi; i++){
50          alan = CGRectMake(5.0 + CGFloat(i * 2) * top.yariCap, parametre!.cubuk.yer.origin.y
51          + 1.4 * parametre!.cubuk.yer.size.height, 1.6 * top.yariCap, 1.6 * top.yariCap);
52          CGContextSetFillColorWithColor(context, UIColor.purpleColor().CGColor);
53          CGContextFillEllipseInRect(context, alan);
54      }
55      CGContextSetTextMatrix(context, CGAffineTransformMakeScale(1.0, -1.0));
56      fontBuyukluk = parametre!.ekranGenislik * 12 / 300;
57      let puanOzellik = [NSFontAttributeName : UIFont.systemFontOfSize(fontBuyukluk)];
58      let puanYazi = NSMutableAttributedString(string:String(format: "%d",
59      parametre!.puan), attributes :puanOzellik);
60      CGContextSetTextPosition(context, parametre!.ekranGenislik - puanYazi.size().width,
61      parametre!.ekranYukseklik - 10 - puanYazi.size().height);
62      CTLineDraw(CTLineCreateWithAttributedString(puanYazi), context);
63      fontBuyukluk = parametre!.ekranGenislik * 12 / 100;
64      let seviyeOzellik = [NSFontAttributeName : UIFont.systemFontOfSize(fontBuyukluk)];
65      let seviyeYazi = NSMutableAttributedString(string:String(format: "Seviye␣%d",
66      parametre!.seviye.seviyeNo + 1), attributes : seviyeOzellik);
67      CGContextSetTextPosition(context, parametre!.ekranGenislik / 2 - seviyeYazi.size().width / 2,
68      parametre!.ekranYukseklik / 2);
69      CTLineDraw(CTLineCreateWithAttributedString(seviyeYazi), context);
70  }
```

kontrol edilir (Satır 5). Eğer dokunduğu yerde çubuk varsa, çubuk hareket ettirilmeye başlamıştır (Satır 7). Eğer dokunduğu yerde çubuk yoksa (Satır 8), çubuk hareket etmiyordur (Satır 9).

- Ekrana dokunmayı bıraktığında (Satır 12), çubuğu artık hareket ettirmiyordur. Çubuğu bıraktığı yer ile ilk buluduğu yer arasındaki fark hesaplanır (Satır 14). Ekran **setNeedsDisplay** metodu kullanılarak yeniden çizilir (Satır 15).

- Ekranda çubuğu sürüklediğinde (Satır 19), çubuğun sürüklendiği yer ile ilk bulunduğu yer arasındaki fark hesaplanır (Satır 21). Ekran **setNeedsDisplay** metodu kullanılarak yeniden çizilir (Satır 22).

Tablo 10.23: **TuglaKirmaca** *sınıfının* **viewDidLoad** *metodu*

```
override func viewDidLoad() {
    var oyunParametre : Parametre;
    super.viewDidLoad();
    ekranGenislik = UIScreen.mainScreen().bounds.size.width;
    ekranYukseklik = UIScreen.mainScreen().bounds.size.height;
    seviyeNo = 0;
    oyun = Oyun(dosyaAd: "seviyeler", genislik : ekranGenislik );
    oyunParametre = Parametre(genislik: ekranGenislik , yukseklik : ekranYukseklik ,
    seviye : oyun!. seviye (seviyeNo ));
    ekran.parametre = oyunParametre;
    zamanlayici = NSTimer.scheduledTimerWithTimeInterval(0.01, target: self ,
    selector : Selector ("onTimer"), userInfo : nil , repeats : true );
}
func yeniSeviye (){
    seviyeNo++;
    ekran.parametre!. yeniSeviye (oyun!. seviye (seviyeNo ));
    zamanlayici = NSTimer.scheduledTimerWithTimeInterval(0.01, target: self ,
    selector : Selector ("onTimer"), userInfo : nil , repeats : true );
}
```

Tablo 10.24: **TuglaKirmaca** *sınıfının* **onTimer** *metodu*

```
func onTimer(){
    var i : Int;
    var top : Top;
    var dusenTugla : DusenTugla;
    for (i = 0; i < ekran.parametre!. topSayisi (); i++){
        top = ekran.parametre!.top(i );
        top. hareketEttir ();
        ekran.parametre!.topCubuklaTemasKontrol(top);
        if (ekran.parametre!.topTuglaylaTemasKontrol(top)){
            zamanlayici ?. invalidate ();
            yeniSeviye ();
            return;
        }
        if (!ekran.parametre!. topSinirlarlaTemasKontrol (top, index: i)){
            zamanlayici ?. invalidate ();
        }
    }
    for (i = 0; i < ekran.parametre!. dusenTuglaSayisi (); i++){
        dusenTugla = ekran.parametre!.dusenTugla(i );
        dusenTugla. hareketEttir ();
        if (!ekran.parametre!. dusenTuglaCubuklaTemasKontrol()){
            zamanlayici ?. invalidate ();
        }
        ekran.setNeedsDisplay ();
    }
}
```

Tablo 10.25: **TuglaKirmaca** *sınıfının* **hareketEttir** *metodu*

```
@IBAction func hareketEttir( hareketAlgilayici : UIPanGestureRecognizer) {
    var p : CGPoint;
    p = hareketAlgilayici.locationInView( hareketAlgilayici.view);
    if ( hareketAlgilayici.state == UIGestureRecognizerState.Began){
        if (ekran.parametre!.elleCubukTemas(p.x, y : p.y)){
            fark = p.x - ekran.parametre!.cubuk.yer.origin.x;
            hareketBasladi = true;
        } else {
            hareketBasladi = false;
        }
    } else {
        if ( hareketAlgilayici.state == UIGestureRecognizerState.Ended){
            if (hareketBasladi){
                ekran.parametre!.cubukYeniX(p.x - fark);
                ekran.setNeedsDisplay();
            }
            hareketBasladi = false;
        } else {
            if ( hareketAlgilayici.state == UIGestureRecognizerState.Changed){
                if (hareketBasladi){
                    ekran.parametre!.cubukYeniX(p.x - fark);
                    ekran.setNeedsDisplay();
                }
            }
        }
    }
}
```

10.6 Alıştırmalar

1. Yılan Oyunu: Yılan oyununu geliştiriniz.