10

Tuğla Kırmaca

Bu bölümde dokuzuncu ve son Objective C 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 periyotlarla 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 durumuyla ilgili bilgiler **Parametre** sınıfında, oyundaki tüm seviyelerle ilgili bilgiler **Oyun** sınıfında, çizilecek ekranla ilgili bilgiler **Ekran** sınıfında ve son olarak da ana aktivite, **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.

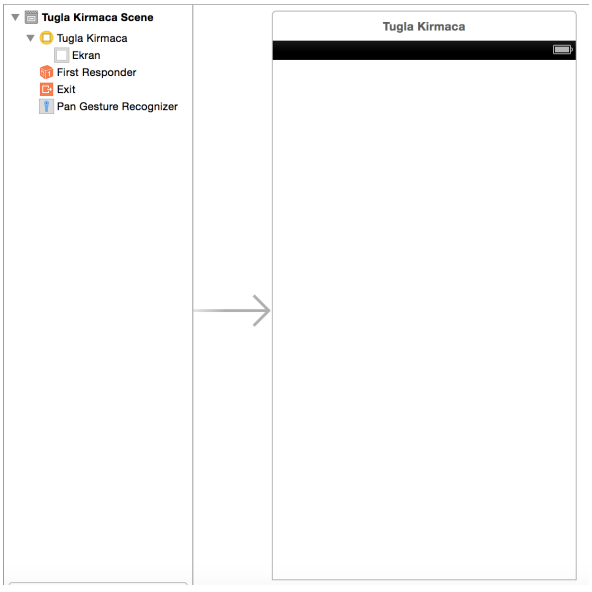
- Oyuncu bir seviyedeki tüm tuğlaları kırdığında bir sonraki seviyeye geçer.
- Oyuncu ekrandaki çubuğa dokunarak sola veya sağa doğru hareket ettirebilir.
- Top, çubuk ile temas ettiğinde seker.
- Top, ekranın sol, sağ ve üst kenarına çarptığında seker.
- Top, ekranın alt kenarında çarptığında yok olur.
- Ekranda hiç top kalmazsa oyuncu bir hak kaybeder.
- Tuğlalardan kahverengi olanı hariç diğer tüm tuğlalar topun bir kere çarpması ile kırılırlar.
- Kahverengi tuğlanın kırılması için topun iki 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 kırılırlar 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 Teknolojiler

10.3.1 Çizim metotları

Objective C 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 void CGContextFillRect ( CGContextRef c, CGRect rect );
```

Objective C 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 void CGContextSetFillColorWithColor(CGContextRef c, CGColorRef color);
2 void  CGContextFillEllipseInRect  (CGContextRef c, CGRect rect);
```

10.4 Uygulama Parçaları

10.4.1 Cubuk

Cubuk, son kullanıcının oyunda sola veya sağa hareket ettirdiği şeritin bilgilerini tutan sınıftır (Tablo 10.1). **yer**, çubuğun koordinatlarını saklar (Satır 2). **Cubuk** sınıfının kurucu fonksiyonu Tablo 10.2'de gösterilmiştir. Çubuğun ilk genişliği ve yüksekliği mobil cihazın genişliği ve yüksekliğine bağlı olarak hesaplanır (Satır 4-7).

Tablo 10.1: Cubuk.h dosyası

```
1 @interface Cubuk : NSObject
2 @property CGRect yer;
3 @end
```

Cubuk sınıfının **buyult**, **kucult** ve **yeniPozisyon** metotları Tablo 10.3'te gösterilmiştir. **buyult** metodunda çubuğun **width** özelliği 1.2 ile çarpılarak çubuğun %20 büyümesi sağlanır (Satır 2). **kucult** metodunda ise çubuğun **width**

Tablo 10.2: **Cubuk** *sınıfının kurucu fonksiyonu*

```
1  - (id) initWithAll :( int ) ekranGenislik  ekranYukseklik :( int ) ekranYukseklik{
2      self = [super init ];
3      if (self){
4          _yer.size.width = 0.15 * ekranGenislik ;
5          _yer.size.height = 0.05 * ekranGenislik ;
6          _yer.origin.x = ekranGenislik / 2 - _yer.size.width / 2;
7          _yer.origin.y = ekranYukseklik - 3.5 * _yer.size.height;
8      }
9      return self ;
10 }
```

özelliği 0.8 ile çarpılarak çubuğun %20 küçülmesi sağlanır (Satır 5). **yeniPozisyon** metodunda, dikdörtgenin sol koordinatı verilen **x** parametresine eşitlenir (Satır 8); böylece dikdörtgen, **x** koordinatına taşınmış olur.

Tablo 10.3: **Cubuk** *sınıfının* **buyult**, **kucult**, **yeniPozisyon** *ve* **yer** *metotları*

```
1  (void)buyult{
2      _yer.size.width *= 1.2;
3  }
4  -(void)kucult{
5      _yer.size.width *= 0.8;
6  }
7  -(void)yeniPozisyon:( int )x{
8      _yer.origin.x = x;
9  }
```

10.4.2 Tugla

Tugla, oyunda son kullanıcının kırarak yeni özellikler kazanabileceği tuğlaları hafızada tutan sınıftır (Tablo 10.4). **tip**, tuğlanın hangi özelliği kazandıracağını (Satır 2); **yer**, tuğlanın koordinatlarını (Satır 3); **kirildi**, tuğlanın kırılıp kırılmadığını (Satır 4); **vurmaSayisi** ise tuğlanın kaç kere vurulduğunu tutmaktadır (Satır 5). **Tugla** sınıfının kurucu fonksiyonu Tablo 10.5'te gösterilmiştir.

Tugla sınıfının **vuruldu** metodu Tablo 10.6'da gösterilmiştir. **vuruldu**, herhangi bir top söz konusu tuğlaya vurduğu zaman çağrılan 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 10.4: Tugla.h dosyası

```
1  @interface Tugla : NSObject
2  @property TuglaTip tip;
3  @property CGRect yer;
4  @property bool kirildi ;
5  @property int vurmaSayisi;
6  @end
7  typedef enum{
8      NORMAL = 1,
9      ZOR,
10     HIZLI,
11     YAVAS,
12     BUYUK,
13     KUCUK,
14     YASAM,
15     OLUM,
16     COKTOP
17 }TuglaTip;
```

Tablo 10.5: **Tugla** *sınıfının kurucu fonksiyonu*

```
1  - (id) initWithAll :( TuglaTip)tip  yer :( CGRect)yer{
2      self  = [super  init ];
3      if  ( self ){
4          _tip = tip;
5          _kirildi  = false;
6          _yer = yer;
7          _vurmaSayisi = 0;
8      }
9      return  self ;
10 }
```

Tablo 10.6: **Tugla** *sınıfının* **vuruldu** *metodu*

```
1  - (void)vuruldu{
2      _vurmaSayisi++;
3      if  (_tip == ZOR){
4          if  (_vurmaSayisi == 2){
5              _kirildi  = true;
6          }
7      } else  {
8          _kirildi  = true;
9      }
10 }
```

10.4.3 DusenTugla

DusenTugla, oyunda aşağıda doğru inen ve son kullanıcının yakalayarak yeni özellikler kazanabileceği tuğlaları hafızada tutan sınıftır (Tablo 10.7). **tip**, tuğlanın hangi özelliği kazandıracağını (Satır 2); **yer**, tuğlanın koordinatlarını (Satır 3); **hiz** ise tuğlanın hangi hızda düştüğünü saklar (Satır 4). **DusenTugla** sınıfının kurucu fonksiyonu Tablo 10.8'de gösterilmiştir.

Tablo 10.7: DusenTugla.h dosyası

```
1  @interface DusenTugla : NSObject
2  @property TuglaTip tip;
3  @property CGRect yer;
4  @property CGPoint hiz;
5  @end
```

Tablo 10.8: **DusenTugla** *sınıfının kurucu fonksiyonu*

```
1  - (id) initWithAll :( TuglaTip)tip  yer :( CGRect)yer yukseklik :( int ) yukseklik {
2      self  = [super  init ];
3      if  ( self ){
4          _tip = tip;
5          _yer = yer;
6          _hiz.x = 0;
7          _hiz.y = yukseklik  / 1000.0;
8      }
9      return  self ;
10 }
```

DusenTugla sınıfının **hareketEttir** ve **cubuklaTemas** metotları Tablo 10.9'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).

10.4.4 Top

Top, ekranda görülen herhangi bir topun özelliklerini hafızada tutan metottur (Tablo 10.10). Topun merkezi **merkez** adlı özellikte (Satır 2), hızı **hiz** adlı özellikte (Satır 3), yarıçapı da **yariCap** adlı özellikte (Satır 4) tutulmaktadır. **Top** sınıfının kurucu fonksiyonu Tablo 10.11'de gösterilmiştir. Top, ilk olarak kuzeybatı yönünde 45 derecelik bir açıyla hareket etmektedir (Satır 4-5). Topun yarıçapı ekranın %2'si kadardır (Satır 6).

Top sınıfının **sinirlarlaTemas**, **tuglaylaTemas** ve **cubuklaTemas** metotları Tablo 10.12'de gösterilmiştir. **sinirlarlaTemas**, topun ekranın sınırlarına çarpıp çarpmadığını kontrol eden metottur. Eğer top, ekranın sol veya sağ tarafına

Tablo 10.9: **DusenTugla** *sınıfının* **hareketEttir** *ve* **cubuklaTemas** *metotları*

```
1  -(void) hareketEttir{
2      _yer.origin.x += _hiz.x;
3      _yer.origin.y += _hiz.y;
4  }
5  -(bool)cubuklaTemas:(Cubuk*)cubuk{
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 }
```

Tablo 10.10: Top.h dosyası

```
1  @interface Top : NSObject
2  @property CGPoint merkez;
3  @property CGPoint hiz;
4  @property int yariCap;
5  @end
```

Tablo 10.11: **Top** *sınıfının kurucu fonksiyonu*

```
1  - (id) initWithAll :(int)cubukX cubukY:(int)cubukY ekranGenislik:(int) ekranGenislik{
2      self = [super init];
3      if (self){
4          _hiz.x = -ekranGenislik / 600.0;
5          _hiz.y = -ekranGenislik / 600.0;
6          _yariCap = ekranGenislik / 50;
7          _merkez.x = cubukX + _yariCap;
8          _merkez.y = cubukY - _yariCap;
9      }
10     return self;
11 }
```

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 10.12: **Top** *sınıfının* **sinirlarlaTemas**, **tuglaylaTemas** *ve* **cubuklaTemas** *metotları*

```
1  -(bool)sinirlarlaTemas :( int ) ekranGenislik  ekranYukseklik :( int )ekranYukseklik{
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 -(bool)tuglaylaTemas:(Tugla*)tugla{
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 -(bool)cubuklaTemas:(Cubuk*)cubuk{
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 10.13'te 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).

10.4.5 Seviye

Seviye sınıfının özellikleri Tablo 10.14'te ve kurucu fonksiyonu Tablo 10.15'te gösterilmiştir. **Seviye**, oyunun bir seviyesiyle ilgili bilgileri hafızada tutan sınıftır.

Tablo 10.13: **Top** *sınıfının* **hareketEttir**, **hizlandir** *ve* **yavaslat** *metotları*

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

Seviyeye ait tuğlalar **satirlar** adlı iki boyutlu dizide (Satır 2), **satirlar** dizisinin birinci boyutu **satir** adlı özellikte (Satır 3), ikinci boyutu **sutun** adlı özellikte (Satır 4), seviyenin numarası **seviyeNo** adlı özellikte (Satır 5), herhangi bir tuğlanın genişliği ve yüksekliği de **tuglaGenislik** ve **tuglaYukseklik** (Satır 1-2) adlı özelliklerde tutulmaktadır.

Tablo 10.14: Seviye.h dosyası

```
1  @interface Seviye : NSObject
2  @property NSMutableArray* satirlar;
3  @property int satir ;
4  @property int sutun;
5  @property int seviyeNo;
6  @end
```

Seviye sınıfının **bittiMi** ve **tugla** metotları Tablo 10.16'da gösterilmiştir. **bittiMi**, seviyenin bitip bitmediğini kontrol eden metottur. Seviyedeki her tuğla için (Satır 5), o tuğlanın kırılıp kırılmadığı kontrol edilir (Satır 6). Eğer herhangi bir tuğla kırılmadıysa seviye bitmemiştir (Satır 7). Tüm tuğlalar kırıldıysa seviye bitmiştir, metot **true** döndürür (Satır 11). **tugla**, verilen **satir** ve **sutun**'daki tuğlayı döndürür (Satır 14).

Seviye sınıfının **satirAyarla** metodu Tablo 10.17'de gösterilmiştir. **satirAyarla**, tuğla dizisinin bir satırını bir katardan ('1111111111' gibi) üreten metottur. Katardaki her i. karakter için (Satır 2), yeni bir dikdörtgen yaratılır (Satır 4-7). Katardaki karakter,

1 NORMAL tuğla yaratılır (Satır 10).

2 Kırılması ZOR tuğla yaratılır (Satır 13).

3 Kırıldığında, topları hızlandırabilecek tuğla yaratılır (Satır 16).

Tablo 10.15: **Seviye** *sınıfının kurucu fonksiyonu*

```
1  int tuglaGenislik;
2  int tuglaYukseklik;
3  - (id) initWithAll:(int)satir sutun:(int)sutun genislik:(int)genislik seviyeNo:(int)seviyeNo{
4      self = [super init];
5      if (self){
6          _satir = satir;
7          _sutun = sutun;
8          _seviyeNo = seviyeNo;
9          _satirlar = [[NSMutableArray alloc] initWithCapacity:satir];
10         for (int i = 0; i < satir; i++){
11             _satirlar[i] = [[NSMutableArray alloc] initWithCapacity:sutun];
12         }
13         tuglaGenislik = genislik / sutun;
14         tuglaYukseklik = 0.7 * tuglaGenislik;
15     }
16     return self;
17 }
```

Tablo 10.16: **Seviye** *sınıfının* **bittiMi** *ve* **tugla** *metotları*

```
1  -(bool)bittiMi{
2      Tugla* tugla;
3      for (int i = 0; i < _satir; i++){
4          for (int j = 0; j < _sutun; j++){
5              tugla = _satirlar[i][j];
6              if (!tugla.kirildi){
7                  return false;
8              }
9          }
10     }
11     return true;
12 }
13 -(Tugla*)tugla:(int)satir sutun:(int)sutun{
14     return _satirlar[satir][sutun];
15 }
```

4 Kırıldığında, topları yavaşlatabilecek tuğla yaratılır (Satır 19).

5 Kırıldığında, çubuğu büyültebilecek tuğla yaratılır (Satır 22).

6 Kırıldığında, çubuğu küçültebilecek tuğla yaratılır (Satır 25).

7 Kırıldığında, oyuncuya yeni bir hak verebilecek tuğla yaratılır (Satır 28).

8 Kırıldığında, oyuncunun bir hakkını azaltabilecek tuğla yaratılır (Satır 31).

9 Kırıldığında, yeni bir top üretebilecek tuğla yaratılır (Satır 34).

Tablo 10.17: **Seviye** *sınıfının* **satirAyarla** *metodu*

```
1  -(void) satirAyarla :( int )satirNo   satirBilgi  :( NSString*) satirBilgi {
2      for  (int  i  = 0; i  < [ satirBilgi   length ];  i++){
3          CGRect yer;
4          yer . origin .x = i  *  tuglaGenislik ;
5          yer . origin .y = satirNo  *  tuglaYukseklik ;
6          yer . size .width  =  tuglaGenislik ;
7          yer . size . height  =  tuglaYukseklik ;
8          switch  ([ satirBilgi   characterAtIndex : i ]){
9              case  '1':
10                 _satirlar [satirNo ][ i ]  =  [[Tugla  alloc ]  initWithAll :NORMAL yer:yer];
11                 break;
12             case  '2':
13                 _satirlar [satirNo ][ i ]  =  [[Tugla  alloc ]  initWithAll :ZOR yer:yer];
14                 break;
15             case  '3':
16                 _satirlar [satirNo ][ i ]  =  [[Tugla  alloc ]  initWithAll :HIZLI  yer : yer ];
17                 break;
18             case  '4':
19                 _satirlar [satirNo ][ i ]  =  [[Tugla  alloc ]  initWithAll :YAVAS yer:yer];
20                 break;
21             case  '5':
22                 _satirlar [satirNo ][ i ]  =  [[Tugla  alloc ]  initWithAll :BUYUK yer:yer];
23                 break;
24             case  '6':
25                 _satirlar [satirNo ][ i ]  =  [[Tugla  alloc ]  initWithAll :KUCUK yer:yer];
26                 break;
27             case  '7':
28                 _satirlar [satirNo ][ i ]  =  [[Tugla  alloc ]  initWithAll :YASAM yer:yer];
29                 break;
30             case  '8':
31                 _satirlar [satirNo ][ i ]  =  [[Tugla  alloc ]  initWithAll :OLUM yer:yer];
32                 break;
33             case  '9':
34                 _satirlar [satirNo ][ i ]  =  [[Tugla  alloc ]  initWithAll :COKTOP yer:yer];
35                 break;
36         }
37     }
38 }
```

10.4.6 Oyun

Oyun sınıfı Tablo 10.18'de gösterilmiştir. **Oyun**, oyunda var olan seviyeleri dosyadan okuyan ve hafızada tutan sınıftır. 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.

Tablo 10.18: **Oyun** *sınıfı*

```
- (id) initWithAll :( NSString*)dosyaAd genislik :( int ) genislik {
    self = [super init ];
    if ( self ){
        NSString *dosyaAdi, *dosyaIcerik , * satirBilgi ;
        NSScanner* ayirici ;
        int i, satir , sutun;
        dosyaAdi = [[NSBundle mainBundle] pathForResource:dosyaAd
                                                   ofType:@"txt"];
        dosyaIcerik = [NSString stringWithContentsOfFile:dosyaAdi
                                                encoding:NSUTF8StringEncoding
                                                   error :NULL];
        ayirici = [NSScanner scannerWithString:dosyaIcerik ];
        [ ayirici scanInt:& _seviyeSayisi ];
        _seviyeler = [[NSMutableArray alloc] initWithCapacity : _seviyeSayisi ];
        for (i = 0; i < _seviyeSayisi ; i++){
            [ ayirici scanInt:& satir ];
            [ ayirici scanInt:&sutun];
            Seviye* seviye = [[Seviye alloc ] initWithAll : satir sutun:sutun
             genislik : genislik seviyeNo: i ];
            for (int j = 0; j < satir ; j++){
                [ ayirici scanUpToString:@"\n" intoString:& satirBilgi ];
                [ seviye satirAyarla :j satirBilgi : satirBilgi ];
            }
             _seviyeler [ i ] = seviye ;
        }
    }
    return self ;
}
-(Seviye*)seviye :( int )pozisyon{
    return _seviyeler [ pozisyon ];
}
```

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.

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

```
3 10
9999999999
2222222222
9999999999
```

Önce verinin okunacağı dosya 'seviyeler.txt' bir **NSString** yapısına dönüştürülür (Satır 7-8). Bu aşamada dosya ismi **pathForResource** parametresi, dosya uzantısı ise **ofType** parametresi ile ifade edilir. Sonra, bu **NSString** yapısı ile dosya tek bir kataın içine okunur (Satır 9-11). Bu aşamada dosyanın adı **stringWithContentsOfFile** parametresi ile, dosyanın dil kodlaması ise **encoding** parametresi ile belirtilir. Ardından, bu **NSString** yapısı bir ayırıcıya (**NSScanner**) **scannerWithString** ile parametre olarak gönderilerek işlenmeye hazırlanır (Satır 12).

Dosyadan veri okunmaya hazırlandıktan sonra, seviye sayısı okunmakta ve **seviyeSayisi** özelliğinde tutulmaktadır (Satır 13). Her seviye için (Satır 15)

- Kaç satır ve sütun tuğla olduğu okunmaktadır (Satır 16-17).
- Yeni bir seviye için hafızada yer açılmakta (Satır 18-19) 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).

Tablo 10.19: Parametre.h dosyası

```
1  @interface Parametre : NSObject
2  @property int yasamSayisi;
3  @property Cubuk* cubuk;
4  @property NSMutableArray* dusenTuglalar;
5  @property NSMutableArray* toplar;
6  @property Seviye* seviye;
7  @property int puan;
8  @property int ekranGenislik;
9  @property int ekranYukseklik;
10 @end
```

10.4.7 Parametre

Oyuncunun kullanabileceği hak sayısı **yasamSayisi** adlı özellikte (Satır 2); oyuncunun sürükleyeceği çubuk **cubuk** adlı özellikte (Satır 3); ekranda o anda aşağı düşen tuğlalar **dusenTuglalar** adlı **NSMutableArray**'de (Satır 4); ekranda o anda var olan toplar **toplar** adlı **NSMutableArray**'de (Satır 5); oyuncunun oynadığı seviye **seviye** adlı özellikte (Satır 6); oyuncunun o ana kadar topladığı puan **puan** adlı özellikte (Satır 7); ekranın genişliği ve yüksekliği de **ekranGenislik** ve

ekranYukseklik adlı özelliklerde tutulmaktadır (Satır 8-9). **Parametre** sınıfının kurucu fonksiyonu Tablo 10.20'de gösterilmiştir. Oyunun başında oyuncunun 3 hakkı (Satır 6) ve 0 puanı bulunmaktadır (Satır 7).

Tablo 10.20: **Parametre** *sınıfının kurucu fonksiyonu*

```
1  - (id) initWithAll :( int ) genislik   yukseklik :( int ) yukseklik   seviye :( Seviye  *) seviye {
2      self  = [super  init ];
3      if  ( self ){
4          _ekranGenislik  =  genislik ;
5          _ekranYukseklik = yukseklik;
6          _yasamSayisi = 3;
7          _puan = 0;
8          [ self   yeniSeviye : seviye ];
9      }
10     return  self ;
11 }
```

Tablo 10.21: **Parametre** *sınıfının* **yeniSeviye**, **topSayisi**, **top**, **dusenTuglaSayisi** *ve* **dusenTugla** *metotları*

```
1  -(void)yeniSeviye :( Seviye*) seviye {
2      _seviye  =  seviye;
3      _cubuk = [[Cubuk alloc]  initWithAll : _ekranGenislik  ekranYukseklik : _ekranYukseklik];
4      _toplar  =  [[NSMutableArray alloc]  initWithCapacity :1];
5      _toplar[0]  =  [[Top alloc ]   initWithAll : _cubuk.yer. origin .x  cubukY:_cubuk.yer.origin .y
6      ekranGenislik : _ekranGenislik ];
7      _dusenTuglalar = [[NSMutableArray alloc]  init ];
8  }
9  -(int) topSayisi {
10     return  _toplar.count;
11 }
12 -(Top*)top:(int)pozisyon{
13     return  _toplar[pozisyon ];
14 }
15 -(int)dusenTuglaSayisi{
16     return  _dusenTuglalar.count;
17 }
18 -(DusenTugla*)dusenTugla:(int)pozisyon{
19     return  _dusenTuglalar[pozisyon ];
20 }
```

Parametre sınıfının **yeniSeviye**, **topSayisi**, **top**, **dusenTuglaSayisi** ve **dusenTugla** metotları Tablo 10.21'de gösterilmiştir. **yeniSeviye**, oyunda yeni bir seviyeye başlandığında çağrılan metottur. Çubuğu ekranın ortasına getirir (Satır 3), ekrandaki topları temizleyip (Satır 4), tek bir topu çubuğun sol üst köşesine konuşlandırır (Satır 5-6). **topSayisi**, ekrandaki topların (Satır 10), **dusenTuglaSayisi** ise düşen tuğlaların (Satır 16) sayısını döndürür. **top**, **toplar** dizisinin

belirli bir pozisyonundaki topu (Satır 13), **dusenTugla** ise **dusenTuglalar** dizisinin belirli bir pozisyonundaki düşen tuğlayı (Satır 19) döndürür.

Parametre sınıfının **topSinirlarlaTemasKontrol** metodu Tablo 10.22'de gösterilmiştir. Önce topun ekranın dışına çıkıp çıkmadığı kontrol edilir (Satır 3). Eğer top ekranın dışına çıktıysa, **toplar** dizisinden silinir (Satır 4) ve ekranda hiç top kalıp kalmadığına bakılır (Satır 5). Eğer ekranda hiç top kalmadıysa, oyuncu bir hak kaybeder (Satır 6). Bunun sonucunda eğer oyuncunun başka hakkı kaldıysa (Satır 7), yeni bir top, çubuğun sol üst köşesine gelecek şekilde yerleştirilir (Satır 8-9). Oyuncunun başka hakkı kalmadıysa metot **false** döndürür (Satır 12).

Tablo 10.22: **Parametre** *sınıfının* **topSinirlarlaTemasKontrol** *metodu*

```
1  -(bool)topSinirlarlaTemasKontrol:(Top*)top{
2      Top* yeniTop;
3      if ([top sinirlarlaTemas:_ekranGenislik ekranYukseklik:_ekranYukseklik]){
4          [_toplar removeObject:top];
5          if (_toplar.count == 0){
6              _yasamSayisi--;
7              if (_yasamSayisi > 0){
8                  yeniTop = [[Top alloc] initWithAll:_cubuk.yer.origin.x
9                  cubukY:_cubuk.yer.origin.y ekranGenislik:_ekranGenislik];
10                 [_toplar addObject:yeniTop];
11             } else {
12                 return false;
13             }
14         }
15     }
16     return true;
17 }
```

Parametre sınıfının **topTuglaylaTemasKontrol** metodu Tablo 10.23'te gösterilmiştir. Oyunun bu seviyesinde yer alan her tuğla için (Satır 6), bu tuğla eğer kırılmadıysa ve top tuğlayla temas ettiyse (Satır 7), söz konusu tuğla vurulmuştur (Satır 8). Eğer tuğla vurulduğunda kırıldıysa (Satır 9), tuğlanın tipine göre 10 veya 20 puan oyuncunun hanesine yazılır. Eğer tuğla ZOR tipinde bir tuğlaysa (Satır 12) 20 puan (Satır 13), başka bir tuğlaysa (Satır 10) 10 puan (Satır 11) oyuncunun hanesine eklenir. Kırılan tuğla özel bir tuğlaysa (ZOR veya NORMAL değilse), yeni bir düşen tuğla yaratılır (Satır 15-16) ve düşen tuğlalar listesine eklenir (Satır 17). Son tuğla da kırıldıysa (Satır 19), seviye biter (Satır 20) ve metot **true** döndürür.

Parametre sınıfının **topCubuklaTemasKontrol**, **elleCubukTemas** ve **cubukYeniX** metotları Tablo 10.24'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 10.23: **Parametre** *sınıfının* **topTuglaylaTemasKontrol** *metodu*

```
1  -(bool)topTuglaylaTemasKontrol:(Top*)top{
2      Tugla* tugla;
3      DusenTugla* dusenTugla;
4      for (int i = 0; i < _seviye.satir; i++){
5          for (int j = 0; j < _seviye.sutun; j++){
6              tugla = [_seviye tugla:i sutun:j];
7              if (![tugla kirildi] && [top tuglaylaTemas:tugla]){
8                  [tugla vuruldu];
9                  if ([tugla kirildi]){
10                     if (tugla.tip != ZOR)
11                         _puan += 10;
12                     else
13                         _puan += 20;
14                     if (tugla.tip != NORMAL && tugla.tip != ZOR){
15                         dusenTugla = [[DusenTugla alloc] initWithAll:tugla.tip
16                         yer:tugla.yer yukseklik:_ekranYukseklik];
17                         [_dusenTuglalar addObject:dusenTugla];
18                     }
19                     if ([_seviye bittiMi])
20                         return true;
21                 }
22             }
23         }
24     }
25     return false;
26 }
```

Tablo 10.24: **Parametre** *sınıfının* **topCubuklaTemasKontrol**, **elleCubukTemas** *ve* **cubukYeniX** *metotları*

```
1  -(bool)topCubuklaTemasKontrol:(Top*)top{
2      return [top cubuklaTemas:_cubuk];
3  }
4  -(bool)elleCubukTemas:(int)x y:(int)y{
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 -(void)cubukYeniX:(int)x{
13     if (x > 0 && x < _ekranGenislik - _cubuk.yer.size.width){
14         [_cubuk yeniPozisyon:x];
15     }
16 }
```

Parametre sınıfının **dusenTuglaCubuklaTemasKontrol** metodu Tablo 10.25'te gösterilmiştir. Düşen her tuğlanın (Satır 3) çubukla temas edip etmediği kontrol

Tablo 10.25: **Parametre** *sınıfının* **dusenTuglaCubuklaTemasKontrol** *metodu*

```
1  -(bool)dusenTuglaCubuklaTemasKontrol{
2      Top* yeniTop;
3      for (DusenTugla* dusenTugla in _dusenTuglalar){
4          if ([dusenTugla cubuklaTemas:_cubuk]){
5              switch (dusenTugla.tip){
6                  case HIZLI:
7                      for (Top* top in _toplar){
8                          [top hizlandir];
9                      }
10                     break;
11                 case YAVAS:
12                     for (Top* top in _toplar){
13                         [top yavaslat];
14                     }
15                     break;
16                 case BUYUK:
17                     [_cubuk buyult];
18                     break;
19                 case KUCUK:
20                     [_cubuk kucult];
21                     break;
22                 case YASAM:
23                     _yasamSayisi++;
24                     break;
25                 case OLUM:
26                     _yasamSayisi--;
27                     if (_yasamSayisi == 0)
28                         return false;
29                     break;
30                 case COKTOP:
31                     yeniTop = [[Top alloc] initWithAll:_cubuk.yer.origin.x
32                     cubukY:_cubuk.yer.origin.y ekranGenislik:_ekranGenislik];
33                     [_toplar addObject:yeniTop];
34                     break;
35                 default:
36                     break;
37             }
38             [_dusenTuglalar removeObject:dusenTugla];
39             break;
40         }
41     }
42     return true;
43 }
```

edilir (Satır 4). Eğer düşen tuğla çubukla temas ettiyse, tuğlanın tipine göre,

- HIZLI: Ekranda var olan tüm toplar (Satır 7) hızlandırılır (Satır 8).
- YAVAS: Ekranda var olan tüm toplar (Satır 12) yavaşlatılır (Satır 13).
- BUYUK: Çubuk büyültülür (Satır 17).

- KUCUK: Çubuk küçültülür (Satır 20).
- YASAM: Oyuncunun hak sayısı bir artırılır (Satır 23).
- OLUM: Oyuncunun hak sayısı bir azaltılır (Satır 26). Oyuncunun hak sayısı 0 kaldıysa, metot **false** döndürür.
- COKTOP: Yeni bir top çubuğun sol üst köşesine gelecek şekilde yaratılır (Satır 31-32) ve top listesine eklenir (Satır 33).

10.4.8 Ekran

Ekran sınıfının **tuglaRenk** metodu Tablo 10.26'da gösterilmiştir. **tuglaRenk**, gerek çizilen gerekse düşen tuğlaların renklerini belirtmek için kullanılmaktadır. Normal tuğlanın rengi siyah (Satır 4), kırılması zor tuğlanın rengi mor (Satır 6), top(lar)ın hızlanmasını sağlayan tuğlanın rengi mavi (Satır 8), yavaşlamasını sağlayan tuğlanın rengi camgöbeği (Satır 10), çubuğun büyümesini sağlayan tuğlanın rengi turuncu (Satır 12), küçülmesini sağlayan tuğlanın rengi yeşil (Satır 14), oyuncuya fazladan bir hak veren tuğlanın rengi mor (Satır 16), oyuncunun bir hakkını alan tuğlanın rengi kırmızı (Satır 18) ve son olarak oyundaki top sayısını bir artıran tuğlanın rengi de sarıdır (Satır 20).

Tablo 10.26: **Ekran** *sınıfının* **tuglaRenk** *metodu*

```
1  -(UIColor*) tuglaRenk:(TuglaTip)tip{
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 }
```

Oyunun belirli bir anını çizebilmek için Ekran sınıfı **UIView** sınıfını genişletmiş, ve **drawRect** metodununun da üstüne yazmıştır (Tablo 10.27 ve 10.28).

Tablo 10.27: **Ekran** *sınıfının* **drawRect** *metodu (1)*

```
1  - (void)drawRect:(CGRect)rect{
2      CGContextRef context;
3      Seviye* seviye;
4      Tugla* cizilenTugla;
5      Cubuk* cubuk;
6      Top* top;
7      DusenTugla* dusenTugla;
8      UIColor* renk;
9      int fontBuyukluk;
10     CGSize yaziBuyukluk;
11     context = UIGraphicsGetCurrentContext();
12     seviye = _parametre.seviye;
13     for (int i = 0; i < seviye.satir; i++){
14         for (int j = 0; j < seviye.sutun; j++){
15             cizilenTugla = [seviye tugla:i sutun:j];
16             if (![cizilenTugla kirildi]){
17                 renk = [self tuglaRenk:cizilenTugla.tip];
18                 CGContextSetFillColorWithColor(context, renk.CGColor);
19                 CGContextFillRect(context, cizilenTugla.yer);
20                 CGContextSetStrokeColorWithColor(context, [UIColor grayColor].CGColor);
21                 CGContextAddRect(context, cizilenTugla.yer);
22                 CGContextStrokePath(context);
23             }
24         }
25     }
26     cubuk = _parametre.cubuk;
27     CGContextSetFillColorWithColor(context, [UIColor darkGrayColor].CGColor);
28     CGContextFillRect(context, cubuk.yer);
29     CGContextSetStrokeColorWithColor(context, [UIColor grayColor].CGColor);
30     CGContextAddRect(context, cubuk.yer);
31     CGContextStrokePath(context);
32     for (int i = 0; i < [_parametre topSayisi]; i++){
33         top = [_parametre top:i];
34         CGRect alan = CGRectMake(top.merkez.x - top.yariCap, top.merkez.y - top.yariCap,
35         2 * top.yariCap, 2 * top.yariCap);
36         CGContextSetFillColorWithColor(context, [UIColor purpleColor].CGColor);
37          CGContextFillEllipseInRect (context, alan);
38     }
```

Ekranda sırasıyla aşağıdaki şekiller çizilmektedir:

- Tuğlalar: Oyunun seviyesine göre, tuğlaların bulunduğu satır ve sütun sayısı belirlenmekte (Satır 13-14), her olası satır ve sütun için, o pozisyonda bulunan tuğlanın (Satır 15) kırılıp kırılmadığına bakılmaktadır (Satır 16). Eğer tuğla kırılmadıysa, çizilecek tuğlanın rengi, **tuglaRenk** metoduyla belirlenmekte (Satır 17), ardından tuğlanın içi o renkte boyanmakta (Satır 18-19), tuğlanın dış kenarı ise gri renkte çizilmektedir (Satır 20-21).

- Çubuk: Çubuk, koyu gri renkte çizilmektedir (Satır 27). Önce tuğlanın içi

Tablo 10.28: **Ekran** *sınıfının* **drawRect** *metodu (2)*

```
39      for (int i = 0; i < [_parametre dusenTuglaSayisi]; i++){
40          dusenTugla = [_parametre dusenTugla:i];
41          renk = [self tuglaRenk:dusenTugla.tip];
42          CGContextSetFillColorWithColor(context, renk.CGColor);
43          CGContextFillEllipseInRect(context, dusenTugla.yer);
44      }
45      for (int i = 0; i < _parametre.yasamSayisi; i++){
46          CGRect alan = CGRectMake(5 + i * 2 * top.yariCap,
47            _parametre.cubuk.yer.origin.y + 1.4 * _parametre.cubuk.yer.size.height,
48            1.6 * top.yariCap, 1.6 * top.yariCap);
49          CGContextSetFillColorWithColor(context, [UIColor purpleColor].CGColor);
50          CGContextFillEllipseInRect(context, alan);
51      }
52      NSString* puan = [NSString stringWithFormat:@"%d", _parametre.puan];
53      fontBuyukluk = _parametre.ekranGenislik * 12 / 300;
54      yaziBuyukluk = [puan sizeWithFont:[UIFont systemFontOfSize:fontBuyukluk]];
55      CGContextSelectFont(context, [[UIFont systemFontOfSize:fontBuyukluk].fontName
56      UTF8String], fontBuyukluk, kCGEncodingMacRoman);
57      CGContextSetTextMatrix(context, CGAffineTransformMake(1.0,0.0, 0.0, -1.0, 0.0, 0.0));
58      CGContextShowTextAtPoint(context, _parametre.ekranGenislik - yaziBuyukluk.width,
59      _parametre.ekranYukseklik - 10 - yaziBuyukluk.height, [puan UTF8String], [puan length]);
60      NSString* seviyeYazi = [NSString stringWithFormat:@"Seviye␣%d",
61      _parametre.seviye.seviyeNo + 1];
62      fontBuyukluk = _parametre.ekranGenislik * 12 / 150;
63      yaziBuyukluk = [seviyeYazi sizeWithFont:[UIFont systemFontOfSize:fontBuyukluk]];
64      CGContextSelectFont(context, [[UIFont systemFontOfSize:fontBuyukluk].fontName
65      UTF8String], fontBuyukluk, kCGEncodingMacRoman);
66      CGContextShowTextAtPoint(context, _parametre.ekranGenislik / 2 - yaziBuyukluk.width / 2,
67      _parametre.ekranYukseklik / 2 - yaziBuyukluk.height / 2,
68      [seviyeYazi UTF8String], [seviyeYazi length]);
69  }
```

koyu gri renkle boyanmakta (Satır 28), ardından dış kenarı gri renkle çizilmektedir (Satır 29-30).

- Toplar: Toplar, pembe renkle çizilmektedir (Satır 36). Oyunun içindeki her top için (Satır 32), topu çevreleyen dikdörtgen belirlenmekte (Satır 34-35), ardından bu dikdörtgenin içine daire çizilmektedir (Satır 37).

- Düşen Tuğlalar: Her düşen tuğla için (Satır 40), düşen tuğlanın rengi **tuglaRenk** metoduyla belirlenmekte (Satır 41), ardından tuğla bu renkle eliptik olarak çizilmektedir (Satır 43).

- Haklar: Oyuncunun sahip olduğu haklar küçük daireler biçiminde sol alt köşeye çizilmektedir. Oyuncunun her hakkı için (Satır 45), hakka karşılık gelen daireyi çevreleyen dikdörtgen belirlenmekte (Satır 46-48), ardından bu dikdörtgenin içine pembe renkli daire çizilmektedir (Satır 50).

- Puan: Oyuncunun bu ana kadar kazandığı puan ekranın sağ alt köşesine yazılmaktadır. Puan yazısının büyüklüğü ekran genişliğinin %4'ü olacak şekilde ayarlanmaktadır (Satır 53-57). Ardından pembe renkle oyuncunun kazandığı puan ekranın sağ alt köşesine yazılmaktadır (Satır 58-59)
- Seviye: Oyuncunun oynadığı seviyenin yazı büyüklüğü ekran genişliğinin %8'i olacak şekilde ayarlanmaktadır (Satır 62-63). Ardından pembe renkle seviye numarası ekranı ortalayacak biçimde yazılmaktadır (Satır 66-68).

10.4.9 TuglaKirmaca

TuglaKirmaca sınıfının özellikleri ve **viewDidLoad** metodu Tablo 10.29'da gösterilmiştir. **TuglaKirmaca**, **UIView** 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. Ekranın genişliği ve yüksekliği **UIScreen** sınıfının **bounds** özelliğiyle elde edilmektedir (Satır 16-17). Ekranın her 10 milisaniyede bir tekrar çizilebilmesi için, **zamanlayici** adlı bir **NSTimer** tanımlanmış olup (Satır 23), her 10 milisaniyede bu sınıfa ait **onTimer** metodu çağrılmaktadır (Satır 24).

Tablo 10.29: **TuglaKirmaca** *sınıfının özellikleri ve* **viewDidLoad** *metodu*

```
1  NSTimer* zamanlayici = NULL;
2  bool hareketBasladi = false;
3  @implementation TuglaKirmaca
4  int ekranGenislik;
5  int ekranYukseklik;
6  int fark;
7  - (void)yeniSeviye{
8      _seviyeNo++;
9      [_ekran.parametre yeniSeviye:[_oyun seviye:_seviyeNo]];
10     zamanlayici = [NSTimer scheduledTimerWithTimeInterval:0.01 target: self
11     selector:@selector(onTimer) userInfo: nil repeats: YES];
12 }
13 - (void)viewDidLoad{
14     Parametre* oyunParametre;
15     [super viewDidLoad];
16     ekranGenislik = [[UIScreen mainScreen] bounds].size.width;
17     ekranYukseklik = [[UIScreen mainScreen] bounds].size.height;
18     _seviyeNo = 0;
19     _oyun = [[Oyun alloc] initWithAll:@"seviyeler" genislik:ekranGenislik];
20     oyunParametre = [[Parametre alloc] initWithAll:ekranGenislik yukseklik:ekranYukseklik
21     seviye:[_oyun seviye:0]];
22     _ekran.parametre = oyunParametre;
23     zamanlayici = [NSTimer scheduledTimerWithTimeInterval:0.01 target: self
24     selector:@selector(onTimer) userInfo: nil repeats: YES];
25 }
```

TuglaKirmaca sınıfının **onTimer** metodu Tablo 10.30'da gösterilmiştir. **onTimer**, her 10 milisaniyede bir çağrılan, ekranda hareket eden nesnelerin yerlerini değiştiren ve ekranı yeniden çizen metottur. Ekranda yerleri değişen iki tip nesne vardır. Bunlar,

- Toplar: Her top (Satır 5), hareket ettirilir (Satır 6) ve çubukla teması kontrol edilir (Satır 7). Eğer top bir tuğlayla temas ediyorsa ve temas eden tuğla son tuğlaysa (Satır 8), oynanan seviye sona ermiştir. Oyunda yeni seviyeye geçilir (Satır 10). Eğer top ekranın dışına çıktıysa ve sınırın dışına çıkan top son topsa (Satır 13), oyun sona erer (Satır 14).
- Düşen Tuğlalar: Her düşen tuğla (Satır 18), hareket ettirilir (Satır 19) ve çubukla teması kontrol edilir (Satır 20). Eğer düşen tuğla çubukla temas ediyorsa ve bu temasın sonucunda oyuncu son hakkını da kaybederse oyun sona erer (Satır 21).

Tablo 10.30: **TuglaKirmaca** *sınıfının* **onTimer** *metodu*

```
1  -(void)onTimer{
2      Top* top;
3      DusenTugla* dusenTugla;
4      for (int i = 0; i < [_ekran.parametre topSayisi]; i++){
5          top = [_ekran.parametre top:i];
6          [top hareketEttir];
7          [_ekran.parametre topCubuklaTemasKontrol:top];
8          if ([_ekran.parametre topTuglaylaTemasKontrol:top]){
9              [zamanlayici invalidate];
10             [self yeniSeviye];
11             return;
12         }
13         if (![_ekran.parametre topSinirlarlaTemasKontrol:top]){
14             [zamanlayici invalidate];
15         }
16     }
17     for (int i = 0; i < [_ekran.parametre dusenTuglaSayisi]; i++){
18         dusenTugla = [_ekran.parametre dusenTugla:i];
19         [dusenTugla hareketEttir];
20         if (![_ekran.parametre dusenTuglaCubuklaTemasKontrol])
21             [zamanlayici invalidate];
22     }
23     [_ekran setNeedsDisplay];
24 }
```

Son kullanıcı ekrana dokunduğunda, ekranda bir nesneyi sürüklediğinde ve bıraktığında **Pan Gesture Recognizer**'a bağladığımız **hareketEttir** metodu çağrılır (Tablo 10.31). Son kullanıcı,

- Ekrana dokunduğunda (Satır 3), dokunduğu yerde çubuk olup olmadığı kontrol edilir (Satır 4). Eğer dokunduğu yerde çubuk varsa, çubuk hare-

ket ettirilmeye başlamıştır (Satır 6). Eğer dokunduğu yerde çubuk yoksa (Satır 7), çubuk hareket etmiyordur (Satır 8).

- Ekrana dokunmayı bıraktığında (Satır 11), çubuğu artık hareket ettirmiyordur. Çubuğu bıraktığı yer ile ilk buluduğu yer arasındaki fark hesaplanır (Satır 13). Ekran **setNeedsDisplay** metodu kullanılarak yeniden çizilir (Satır 14).

- Ekranda çubuğu sürüklediğinde (Satır 18), çubuğun sürüklendiği yer ile ilk bulunduğu yer arasındaki fark hesaplanır (Satır 20). Ekran **setNeedsDisplay** metodu kullanılarak yeniden çizilir (Satır 21).

Tablo 10.31: **TuglaKirmaca** *sınıfının* **hareketEttir** *metodu*

```
1  - (IBAction)hareketEttir:(UIPanGestureRecognizer *) hareketAlgilayici {
2      CGPoint p = [hareketAlgilayici locationInView: hareketAlgilayici.view];
3      if (hareketAlgilayici.state == UIGestureRecognizerStateBegan){
4          if ([_ekran.parametre elleCubukTemas:p.x y:p.y]){
5              fark = p.x - [_ekran.parametre cubuk].yer.origin.x;
6              hareketBasladi = true;
7          } else {
8              hareketBasladi = false;
9          }
10     } else {
11         if (hareketAlgilayici.state == UIGestureRecognizerStateEnded){
12             if (hareketBasladi){
13                 [_ekran.parametre cubukYeniX:p.x - fark];
14                 [_ekran setNeedsDisplay];
15             }
16             hareketBasladi = false;
17         } else {
18             if (hareketAlgilayici.state == UIGestureRecognizerStateChanged){
19                 if (hareketBasladi){
20                     [_ekran.parametre cubukYeniX:p.x - fark];
21                     [_ekran setNeedsDisplay];
22                 }
23             }
24         }
25     }
26 }
```

10.5 Alıştırmalar

1. Yılan Oyunu: Yılan oyununu geliştiriniz.