HAFTA 2: SFML Kütüphanesine İlk Bakış

Pencere oluşturma

Grafik uygulamaları için öncelikle bir pencere oluşturmamız gerekmektedir. Her işletim sisteminde bu işlem farklılık gösterebilmektedir. Sfml kütüphanesi pencere oluşturma işlemlerini yönetmek için **RenderWindow** sınıfını kullanmaktadır. Bu sınıfa erişmek için **sf** isim uzayını kullanmamız gerekmektedir. **RenderWindow** sayesinde programcılar işletim sistemi ile direk haberleşmek zorunda kalmadan basit bir ara yüz ile pencere oluşturup kullanabilmektedir. Aşağıdaki kod satırı bir pencere nesnesi oluşturmak için kullanılmaktadır.

```
	sf::RenderWindow window(sf::VideoMode(300, 300), "ilk SFML Programi!");
```

Pencere oluşturduktan sonra pencere içerisinde bir şekil çizmek için bir daire şeklini aşağıdaki gibi oluşturuyoruz. **CircleShape** sınıfının kurucu fonksiyonu dairenin çapını parametre olarak alabilmektedir.

```
	sf::CircleShape shape(100.f);
```

Sadece pencere nesnesini oluşturmak yeterli olmayacaktır. Pencere uygulamaları pencere kapatılana kadar devamlı olarak dönen bir ana döngüye sahiptirler. Bu döngünün her turunda pencereye gelen mesajlar işlenir , ardından pencere bu mesajlara göre güncellenir ve son olarak gerekiyorsa pencere çizilir. Sfml uygulamalarında da aynı şekilde bir ana döngü bulunacaktır. Bu döngünün örneği aşağıda verilmiştir. Döngü koşulu pencerenin açık olup olmadığını kontrol etmektedir. Pencere açık ise döngü işlemlerine devam edecektir. Bu döngü oyun uygulamalarında çizim döngüsü olarak da adlandırılmaktadır.

```
	while (window.isOpen())
	{
		//Kodlar
	}
```

Döngü içerisinde pencere üzerinde gerçekleşen olayları (fare tıklanması, klavye tuşuna basılması vb.) kontrol edilmesi gerekmektedir. Pencerede gerçekleşen olayları kontrol etmek için **RenderWindow** sınıfının **pollEvent** metodunu kullanmamız yeterli olacaktır. Bu metodun prototipi aşağıdaki gibidir.

```
	bool sf::Window::pollEvent(Event& event)
```

pollEvent metodu, olay bilgisini parametresi aracılığıyla kullanıcıya vermektedir. Metodu kullanmadan önce bir adet **Event** nesnesi oluşturmamız gerekmektedir. Aşağıdaki kod bloğunda pencerede meydana gelen olay okunmakta ve ardından olayın türü kontrol edilmektedir. Olayın türü "**Closed**" ise pencerenin kapatma tuşuna basılmış demektir bu yüzden bizde oluşturduğumuz pencerenin "**close**" metodunu kullanarak penceremizi kapatıyoruz.

```
sf::Event event;
window.pollEvent(event)
if (event.type == sf::Event::Closed)
	window.close();
```

Dikkat etmemiz gereken diğer bir hususta pencerede birden fazla olayın gerçekleşebilmesidir. **pollEvent** metodu her çağrıldığında sıradaki olayı kullanıcıya getirmektedir. Eğer bir olay bulamazsa **false** değeri getirmektedir. Bu göz önünde bulundurularak bütün olayları okumak için aşağıdaki döngüyü oluşturmamız gerekmektedir.

```
	sf::Event event;
	while (window.pollEvent(event))
	{
		if (event.type == sf::Event::Closed)
			window.close();
	}
```

Çizim

Oyun uygulamalarında çizim işlemi klasik pencere uygulamalarına göre farklılık gösterebilmektedir. Oyunlarda görüntünün devamlı olarak güncellenmesi gerekmektedir. Akıcı bir görüntü elde edebilmek için görüntü elde etme hızının saniyede 30-60 arası bir değerde olması tercih edilmektedir. Klasik pencere uygulamalarında bu şekilde bir görüntü hızına ihtiyaç duyulmaz.

Çizim işlemi için öncelikle çizim alanının temizlenmesi gerekmektedir. Bunun için **RenderWindow** sınıfının **clear** metodu kullanılmaktadır. Temizleme işleminin ardından pencere de gösterilecek şekiller çizilecektir. Çizim için **RenderWindow** sınıfının **draw** metodu kullanılacaktır. (Çizimi yapacağımız pencerenin metodunu çağırmamız gerekmektedir). Çizim işlemleri yapıldıktan sonra elde edilen görüntünün pencere üzerinde gösterilmesini sağlamak için **display** metodu kullanılmaktadır. Bu işlemler aşağıdaki kod bloğunda sırasıyla gösterilmektedir. Kod 2.1 de temel bir Sfml uygulamasını kaynak kodu gösterilmektedir.

```
	window.clear();
	window.draw(shape);
	window.display();
```

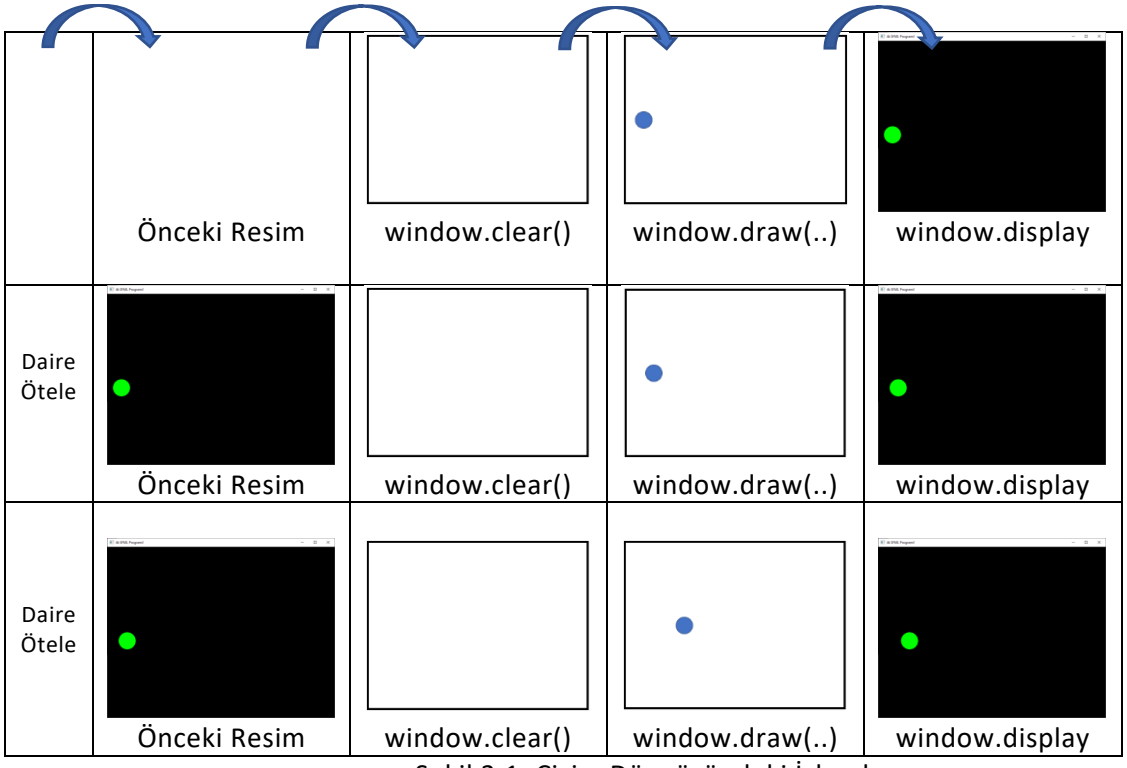
```
01 #include <SFML/Graphics.hpp>
02 int main()
03 {
04 	sf::RenderWindow window(sf::VideoMode(200, 200), "ilk Program");
05 	sf::CircleShape shape(100.f);
06 	shape.setFillColor(sf::Color::Green);
07 	while (window.isOpen())
08 	{
09 		sf::Event event;
10 		while (window.pollEvent(event))
11 		{
12 			if (event.type == sf::Event::Closed)
13 				window.close();
14 		}
15 		window.clear();
16 		window.draw(shape);
17 		window.display();
18 	}
19 	return 0;
20 }
```

Kod 2.1.

Çizim Döngüsü

Çizim işleminde oyunumuzun anlık görüntülerini pencere üzerinde göstereceğiz. Örneğin ekranda bir dairemiz olsun ve bu dairenin zamanla sağa gitmesini istiyoruz. Bunun için görüntü oluşturulmadan önce dairenin konumunu 1 birim arttırmamız yeterli olacaktır. Böylece her elde edilen görüntüde daire biraz daha sağda bulunacaktır. Aynı işlem devamlı olarak yapıldığında daire yavaş yavaş sağa doğru hareket ediyormuş gibi görünecektir.

Şekil 2.1 de çizim işleminin devamlı olarak yapılmasına bir örnek gösterilmiştir. Burada yapılan işlemlerin sonuçları hemen sağında gösterilmiştir. Her bir satır döngünün 1 turunu temsil etmektedir. İşlemlerin sırası soldan sağa doğrudur. Tur atılırken öncelikle daire sağa doğru ötelenmektedir (x koordinatı arttırılıyor). Ardından önceki resmi **`clear`** metoduyla temizliyoruz. Daha sonra **`draw`** metoduyla daireyi yeni konumunda çiziyoruz. Son olarak da elde ettiğimiz görüntüyü pencerede gösteriyoruz. Bu işlemler her turda devamlı olarak yapıldığında dairenin sağa doğru hareket ettiği izlenimi oluşacaktır. Kod 2.2 de bu işlemler gösterilmektedir.



Şekil 2.1. Çizim Döngüsündeki İşlemler

```
01  #include <SFML/Graphics.hpp>
02
03  int main()
04  {
05          sf::RenderWindow window(sf::VideoMode(800, 600), "SFML");
06
07          sf::CircleShape shape(30.f);
08
09          float x = 10;
10
11          shape.setFillColor(sf::Color::Green);
12          while (window.isOpen())
13          {
14                  sf::Event event;
15                  while (window.pollEvent(event))
16                  {
17                          if (event.type == sf::Event::Closed)
18                                  window.close();
19                  }
20                  shape.setPosition(x, 300);
21
22                  window.clear();
23                  window.draw(shape);
24                  window.display();
25
26                  x += 1;
27          }
28          return 0;
29  }
```

Kod 2.2.

Farklı Şekillerin Çizimi

Dörtgen çizmek için **RectangleShape** sınıfını kullanılmaktadır. Öncelikle bu sınıftan bir nesne oluşturulması gerekmektedir. Sınıf parametre olarak **sf::Vector2f** nesnesi almaktadır. Eğer parametre girilmezse değerleri 0 olan bir nesne otomatik olarak atanacaktır. Aşağıdaki kod parçasında bir **RectangleShape** nesnesi oluşturulmaktadır. Kurucuya parametre olarak başlatıcı listesi ile oluşturulan bir **Vector2f** nesnesi verilmektedir. Başlatıcı Listesi (initializer list) **c++ 11** standarttı ile gelmiş bir özelliktir. Oluşturulan **Vector2f** nesnesinin **x** ve **y** elemanlarına sırası ile 50 ve 30 değerleri atanacaktır.

```
        sf::RectangleShape dortgen({ 50,30 });
```

Şeklin rengi yine **setFillColor** fonksiyonu ile atanmaktadır. Şeklin sınırlarını çizdirmek istersek **setOutLineThickness** fonksiyonu aracılığıyla sınır kalınlığını belirtmemiz gerekmektedir. Başlangıçta bu kalınlık varsayılan olarak 0 kabul edilmektedir.

```
        dortgen.setOutlineThickness(10.0f);
```

Sınır çizgileri varsayılan olarak beyaz renge sahiptir. Eğer rengi değiştirilmek istenirse **setOutlineColor** fonksiyonu kullanılabilir.

```
01  #include <SFML/Graphics.hpp>
02
03  int main()
04  {
05        sf::RenderWindow window(sf::VideoMode(800, 600), " SFML!");
06        sf::CircleShape shape(30.f);
07        sf::RectangleShape dortgen({ 50,30 });
08        float x = 10;
09        shape.setFillColor(sf::Color::Green);
10        dortgen.setFillColor(sf::Color::Red);
11        dortgen.setOutlineThickness(10.0f);
12        dortgen.setOutlineColor(sf::Color(250, 150, 100));
13        while (window.isOpen())
14        {
15              sf::Event event;
16              while (window.pollEvent(event))
17              {
18                    if (event.type == sf::Event::Closed)
19                          window.close();
20              }
21              shape.setPosition(x, 300);
22              dortgen.setPosition(300, x);
23              window.clear();
24              window.draw(shape);
25              window.draw(dortgen);
26              window.display();
27
28              x += 0.1;
29        }
30        return 0;
31  }
```

Kod 2.3.

Zaman

Kod 2.2 çalıştırıldığında dairenin sağa doğru çok hızlı bir şekilde hareket ettiğini göreceğiz. Bunun sebebi çizim döngüsünün çok hızlı tur atmasıdır. Hız problemini çözmek için 26. Satırdaki kodu aşağıdaki gibi değiştirebiliriz. Bu sayede x koordinatının zamanla artışını azaltmış olacağız. Kodu tekrar derleyip sonucu inceleyin. Buradaki temel problem döngünün bir turu bitirme süresinin çok düşük olmasıdır. Eğer ki kodumuzu daha pratik hale getirmek istersek artış miktarını bir önceki tur için geçen süre ile ilişkilendirebiliriz.

```
x += 0.1;
```

Sfml kütüphanesi zaman kontrolü için **Clock** sınıfı barındırmaktadır. **Clock** sınıfından bir nesne oluşturduğumuzda otomatik olarak zamanı tutmaya başlayacaktır. Bu süre **clock** nesnesi var olduğu sürece artmaktadır. **restart** metodu ile zaman tekrardan başlatılabilmektedir. Bu metodun dönüş değeri saatin başlangıcından metot çağrılana kadar ki geçen süreyi veren bir **Time** nesnesidir. Kod 2.4'de **Clock** sınıfının kullanımı gösterilmiştir. 8.satır da **Clock** sınıfından bir nesne oluşturuluyor. 13.satırda **Time** türünden bir nesne oluşturuluyor. Bu nesne geçen süreyi tutacaktır. Döngünün sonunda **Clock** nesnesinin **restart** metodu çağrılıyor böylece bir önceki çağrıdan bu yana geçen süreyi elde ediyoruz. Ayrıca zaman tekrardan sıfırlanmaktadır.30. satırda ise geçen süreyi saniyeye çevirip 100 ile çarpıyoruz. Elde ettiğimiz sonucu da şeklin x koordinatına

ekliyoruz. Saniye değeri çok küçük bir rakam olacağı için onu 100 ile çarpmayı uygun gördük. İsterseniz bu çarpanı değiştirerek sonuçları inceleyebilirsiniz.

```
01  #include <SFML/Graphics.hpp>
02  int main()
03  {
04        sf::RenderWindow window(sf::VideoMode(800, 600), " SFML!");
05        sf::CircleShape shape(30.f);
06
07         sf::Clock saat;
08
09         float x = 10;
10        shape.setFillColor(sf::Color::Green);
11
12        sf::Time gecenSure;
13
14        while (window.isOpen())
15        {
16              sf::Event event;
17              while (window.pollEvent(event))
18              {
19                    if (event.type == sf::Event::Closed)
20                          window.close();
21              }
22
23              shape.setPosition(x, 300);
24              window.clear();
25              window.draw(shape);
26              window.display();
27
28              gecenSure = saat.restart();
29              x += 100* gecenSure.asSeconds();
30        }
31        return 0;
32  }
```

Kod 2.4.

Çerçeve Sayısı

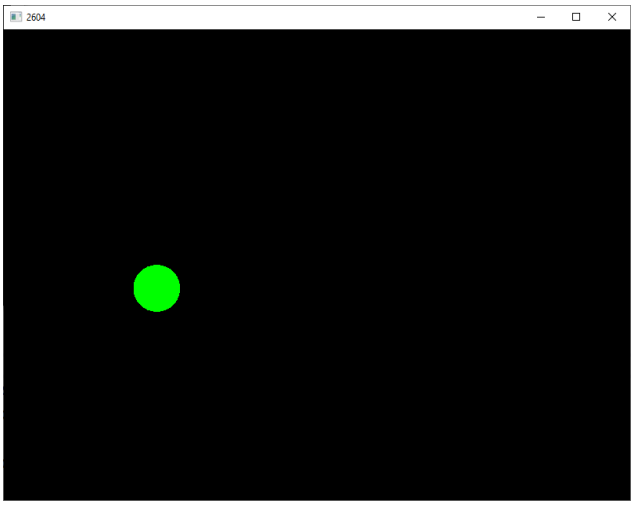
Oyun programlama da çerçeve oyunun bir anlık görüntüsünü temsil etmektedir. Notlar içerisinde çerçeve yerine resim kelimesi de değişimli olarak kullanılmaktadır. Çerçeve sayısı da (Frame per second) bir saniyede gösterebildiğimiz çerçeve sayısını temsil etmektedir. Kimi oyunlar çerçeve sayısına bir sınır getirmezken bazı oyunlar çerçeve sayısını sabit tutmaktadır. Oyunun akıcı olabilmesi için tavsiye edilen en az 30 çerçeve elde edilmesidir. Bu değer 30 FPS olarak da gösterilir. Günümüzde ideal olan çerçeve sayısı da 60 FPS olarak belirlenmiştir.

Programımız içerisinde fps değerini bulmak için aşağıdaki formülü kullanabiliriz. Kod 2.5 içerisinde fps değerini hesaplayıp pencerenin başlığında gösteren programın kodu verilmiştir. 29. satırda bir çerçevenin çizilmesi için geçen süre hesaplanmaktadır. 30.satırda fps değeri hesaplanmıştır. Dikkat edecek olursanız fps tam sayı olarak tanımlanmıştır. Bu sayede virgülden sonraki kısımlar göz ardı edilecektir. Programı çalıştırıp sonucunu inceleyin

$$\text{fps} = \frac{1}{1 \text{ çerçeve için harcanan süre}}$$

```cpp
#include <SFML/Graphics.hpp>
#include<iostream>
#include<string>
int main()
{
	sf::RenderWindow window(sf::VideoMode(800, 600), "SFML!");

	sf::CircleShape shape(30.f);
	sf::Clock saat;
	float x = 10;

	shape.setFillColor(sf::Color::Green);

	sf::Time gecenSure;

	while (window.isOpen())
	{
		sf::Event event;
		while (window.pollEvent(event))
		{
			if (event.type == sf::Event::Closed)
				window.close();
		}
		shape.setPosition(x, 300);
		window.clear();
		window.draw(shape);
		window.display();

		float turSuresi = saat.restart().asSeconds();
		int fps = (1 / turSuresi);
		window.setTitle(std::to_string(fps));

		gecenSure += sf::seconds(turSuresi);
		x += 100 * turSuresi;
	}
	return 0;
}
```

Kod 2.5.



Şekil 2.2. Çerçeve sayısı hesaplama

Çerçeve sayısı kullanılan bilgisayara göre değişiklik gösterecektir. Dikkat edecek olursanız fps değeri çok süratli bir şekilde değişmektedir. Bunun temel sebebi fps değerinin devamlı olarak hesaplanıp ekrana çıkartılmasıdır. Hesaplama süreside çok kısa sürdüğü için çok hızlı bir şekilde sonuçta değişmektedir. Kod 2.4'deki 29,30 ve 31 satırlarını aşağıdaki gibi değiştirirsek fps gösterme hızını yavaşlatabiliriz. Fps sayısı her 200 milisaniyede bir hesaplanmaktadır. Tabi buradaki hesap sadece 200 milisaniye içerisindeki son çerçeve süresini hesaplayıp ekrana çıkartmaktadır.

```
float turSuresi = saat.restart().asSeconds();
if (gecenSure.asMilliseconds() >= 200.0f)
{
	gecenSure -= sf::milliseconds(200.0f);
	int fps = (1 / turSuresi);
	window.setTitle(std::to_string(fps));
}
```

Çerçeve Sayısını Sabitleme

Oyun uygulamaların da çerçeve arasındaki sürelerin sabitlenmesi oyun geliştirme sürecini hızlandırmaktadır. Günümüze kadar birçok oyun bu yöntemi kullanarak animasyon, fizik ve çizim işlemleri gerçekleştirmiştir. Bu ders içerisinde de sfml kütüphanesi uygulamalarında çerçeve sayısını 60 olarak sabitleyeceğiz. Bu işlem için sfml kütüphanesinin bir fonksiyonu bulunmasına rağmen kendi kodumuzu kullanarak bu işlemi gerçekleştireceğiz. Öncelikle 1 çerçevenin süresini bulmamız gerekiyor bu işlemin formülünü daha önce vermiştik. Ona göre aşağıdaki kod satırı bize bir çerçevenin süresini verecektir.

```
float cerceveSuresi = 1.0f / 60.0f;
```

Çerçeve süresini sabitlemek için çizim döngüsünün her turunda geçen sürenin çerçeve süresine eşit olup olmadığı kontrol edilmesi gerekmektedir. Bunun içinde aşağıdaki koşul yeterli olacaktır. Koşul içerisinde ise yani bir çerçeve süresi geçmiş ise geçen süreden çerçeve süresi çıkartılacaktır bu sayede yeni çerçevenin süresi tekrardan başlatılmış olacaktır. Kod 2.6. içerisinde koşul içinde çizilecek şeklin koordinatları güncellenmektedir. Çizim işlemi ve giriş kontrolü bu koşul dışında tutulacaktır.

```
if (gecenSure.asSeconds() >= cerceveSuresi)
{
	//işlemler
	gecenSure -= sf::seconds(cerceveSuresi);
}
```

```cpp
#include <SFML/Graphics.hpp>

int main()
{
	sf::RenderWindow window(sf::VideoMode(800, 600), " SFML !");

	sf::CircleShape shape(30.f);
	sf::Clock saat;
	float x = 10;
	float cerceveSuresi = 1.0f / 60.0f;
	shape.setFillColor(sf::Color::Green);

	sf::Time gecenSure;

	while (window.isOpen())
	{
		sf::Event event;
		while (window.pollEvent(event))
		{
			if (event.type == sf::Event::Closed)
				window.close();
		}
		if (gecenSure.asSeconds() >= cerceveSuresi)
		{
			x += 1;
			shape.setPosition(x, 300);
			gecenSure -= sf::seconds(cerceveSuresi);
		}

		window.clear();
		window.draw(shape);
		window.display();

		gecenSure += saat.restart();
	}
	return 0;
}
```

Kod 2.6.

Giriş Kontrolü

Oyun uygulamalarını kullanmak için klavye, fare veya joystick gibi araçlar kullanılmaktadır. Bu derste de klavye ve fare kullanılacaktır. Sfml klavye ve fare girişlerini kontrol etmek için **Keyboard** ve **Mouse** isimli iki sınıfa sahiptir. Bu sınıfları kullanarak işletim sisteminin karmaşık kütüphanelerini öğrenmek zorunda kalmıyoruz.

Keyboard sınıfı oldukça basittir ve **isKeyPressed** isimli tek bir fonksiyon içermektedir. Fonksiyonun statik olarak tanımlandığı için **Keyboard** sınıfından bir nesne oluşturmak zorunda değiliz. Aşağıdaki kod bloğunda klavyenin sol tuşuna basılıp basılmadığı kontrol edilmektedir. Eğer basıldı ise **isKeyPressed** fonksiyon **true** değeri döndürecek ve koşul içerisine girilecektir.

```cpp
	if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
	{

	}
```

Kodumuzdaki dairenin hareketini sağ ve sol tuşun basılışına göre düzenlemek istersek sonuç Kod 2.7'daki gibi olacaktır. Öncelikle **x** koordinatının artış miktarı 9.satırda tanımlanan bir değişkene bağlanmıştır. Bu değişkenin değeri sağ tuşa basıldığında +1.0f, sol tuşa basıldığın da -1.0f olacaktır. Böylece sol tuşa basıldığında şeklin **x** koordinatından 1 değeri çıkartılmış olacaktır. Kodu çalıştırıp sonuçlarını inceleyin.

```
01  #include <SFML/Graphics.hpp>
02
03  int main()
04  {
05        sf::RenderWindow window(sf::VideoMode(800, 600), " SFML !");
06        sf::CircleShape shape(30.f);
07        sf::Clock saat;
08
09        float x = 10;
10
11        float artis = 1.0f;
12        float cerceveSuresi = 1.0f / 60.0f;
13        shape.setFillColor(sf::Color::Green);
14        sf::Time gecenSure;
15        while (window.isOpen())
16        {
17              sf::Event event;
18              while (window.pollEvent(event))
19              {
20                    if (event.type == sf::Event::Closed)
21                          window.close();
22              }
23              if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right))
24              {
25                    artis = 1.0f;
26              }
27              if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
28              {
29                    artis = -1.0f;
30              }
31              if (gecenSure.asSeconds() >= cerceveSuresi)
32              {
33                    x += artis;
34                    shape.setPosition(x, 300);
35                    gecenSure -= sf::seconds(cerceveSuresi);
36              }
37              window.clear();
38              window.draw(shape);
39              window.display();
40              gecenSure += saat.restart();
41        }
42        return 0;
43  }
```

Kod 2.7.

Nesneye Dayalı Programlama

Şu ana kadar öğrendiğimiz yapıları nesneler ile ifade etmek uygulama geliştirmek için çok daha pratik olacaktır. Başlangıç olarak pencere işlemlerini gerçekleştirecek bir **Pencere** sınıfı oluşturacağız. **Pencere** sınıfının prototipi aşağıdaki gibidir. Pencere sınıfına ait metotların gövdeleri ise Kod 2.9 de gösterilmiştir.

```
01 #pragma once
02 #include<SFML/Graphics.hpp>
03
04 class Pencere
05 {
06 public:
07       Pencere();
08       void ayarla(unsigned int genislik,
09                   unsigned int yukseklik,
10                   sf::String baslik);
11       void cizimeBasla();
12       void cizimiBitir();
13       bool kapandimi();
14       void olayKontrol();
15       void ciz(sf::Drawable& sekil);
16 private:
17       sf::RenderWindow m_window;
18 };
```

Kod 2.8. Pencere.hpp dosyasının içeriği

```
01 #include "Pencere.hpp"
02
03 Pencere::Pencere()
04 {
05       ayarla(800, 600, "SFML");
06 }
07 void Pencere::ayarla(unsigned int genislik,
08       unsigned int yukseklik, sf::String baslik)
09 {
10       m_window.create(sf::VideoMode(genislik, yukseklik), baslik);
11 }
12 void Pencere::cizimeBasla()
13 {
14       m_window.clear();
15 }
16 void Pencere::cizimiBitir()
17 {
18       m_window.display();
19 }
20 bool Pencere::kapandimi()
21 {
22       return !m_window.isOpen();
23 }
24 void Pencere::ciz(sf::Drawable& sekil)
25 {
26       m_window.draw(sekil);
27 }
```

Kod 2.9. Pencere.cpp -1

```
28  void Pencere::olayKontrol()
29  {
30          sf::Event event;
31          while (m_window.pollEvent(event))
32          {
33                  if (event.type == sf::Event::Closed)
34                          m_window.close();
35          }
36  }
```

Kod 2.9. Pencere.cpp -2

ayarla metodu pencere oluşturmak için kullanılmıştır. **RenderWindow** nesnesi oluşturmak ile işletim sistemine ait olan bir pencereyi oluşturmanın farklı işlemler olduğunu dikkate almalıyız. Önceki örneklerimizde **RenderWindow** nesnesini oluştururken kurucu metot ile pencere oluşturulmaktaydı. Yeni sınıfımızda ise bu işlemi **RenderWindow** sınıfına ait olan **create** metodu ile yapıyoruz. Kod 2.9'da **create** metodu parametreleri ile çağrılmaktadır(10.satır).

```
01  #include <SFML/Graphics.hpp>
02  #include "Pencere.hpp"
03  int main()
04  {
05          Pencere yeniPencere;
06
07          sf::CircleShape shape(30.f);
08          sf::Clock saat;
09          float x = 10;
10          float artis = 1.0f;
11          float cerceveSuresi = 1.0f / 60.0f;
12          shape.setFillColor(sf::Color::Green);
13          sf::Time gecenSure;
14
15          while (!yeniPencere.kapandimi())
16          {
17                  yeniPencere.olayKontrol();
18
19                  if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right))
20                          artis = 1.0f;
21                  if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
22                          artis = -1.0f;
23                  if (gecenSure.asSeconds() >= cerceveSuresi)
24                  {
25                          x += artis;
26                          shape.setPosition(x, 300);
27                          gecenSure -= sf::seconds(cerceveSuresi);
28                  }
29
30                  yeniPencere.cizimeBasla();
31                  yeniPencere.ciz(shape);
32                  yeniPencere.cizimiBitir();
33
34                  gecenSure += saat.restart();
35          }
36          return 0;
37  }
```

Kod 2.10. main.cpp

Pencere sınıfının ana program içerisindeki kullanımı Kod 2.10 içerisinde gösterilmektedir. 15.satırda döngü koşulu olarak pencere nesnesinin **kapandimi** metodu çağrılmaktadır. Bu metot pencerenin kapatılıp kapatılmadığını kontrol etmektedir. 17. Satırda ise pencere üzerinde gerçekleşen olaylar kontrol edilmektedir. 30-32 satırlarda çizim işlemi gerçekleştirilmektedir. İlerleyen haftalarda bu işlemler daha pratik hale gelecektir.

Oyun Sınıfı

Oyun sınıfını uygulamamızı temsil edecek ve uygulama içerisinde kullanacağımız temel araçları içerisinde barındıracaktır. **Oyun** sınıfının prototipi aşağıdaki gibidir. **Oyun** sınıfı içerisinde Pencere sınıfını ve çizilecek şekilleri de barındırmaktadır. **Oyun** sınıfını tanımladıktan sonra **main.cpp** içeriği kod 2.12'deki gibi olacaktır.

```
01 #pragma once
02
03 #include "Pencere.hpp"
04
05 class Oyun
06 {
07 public:
08       Oyun();
09       ~Oyun();
10       void girisKontrol();
11       void sahneGuncelle();
12       void sahneCiz();
13
14       void saatiYenidenBaslat();
15       bool bittimi();
16 private:
17       Pencere            m_pencere;
18       sf::Clock          m_saat;
19       sf::Time           m_gecenSure;
20       sf::CircleShape    m_sekil;
21       float m_artis;
22       float m_konumX;
23       float m_cerceveSuresi;
24 };
```

Kod 2.11. Oyun.cpp

```
01 #include <SFML/Graphics.hpp>
02 #include "Oyun.hpp"
03 int main()
04 {
05
06       Oyun oyun;
07       while (!oyun.bittimi())
08       {
09             oyun.girisKontrol();
10             oyun.sahneGuncelle();
11             oyun.sahneCiz();
12             oyun.saatiYenidenBaslat();
13       }
14       return 0;
15 }
```

Kod 2.12. main.cpp

girisKontrol fonksiyonu klavye ve fare kontrolünü yaparken **sahneGuncelle** fonksiyonu da çizilecek şekillerin konumlarını güncellemektedir. **sahneCiz** fonksiyonu sahne içerisindeki bütün şekillerin çizilmesini sağlayacaktır. Şu an için sadece bir daire şeklimiz bulunuyor. Bu şekilden bir nesnede yine **Oyun** sınıfı içerisinde bulunacaktır. **Oyun** sınıfına ait fonksiyonların gövdeleri **Oyun.cpp** içerisinde bulunmaktadır. Bu dosyanın içeriği Kod 2.13'de gösterilmiştir.

```
01 #include "Oyun.hpp"
02
03 Oyun::Oyun()
04 {
05       m_sekil.setRadius(30);
06       m_sekil.setFillColor(sf::Color::Green);
07       m_artis = 1.0f;
08       m_konumX = 10.0f;
09       m_cerceveSuresi = 1.0f / 60.0f;
10 }
11 Oyun::~Oyun()
12 {
13 }
14 void Oyun::girisKontrol()
15 {
16       if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right))
17             m_artis = 1.0f;
18       if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
19             m_artis = -1.0f;
20 }
21 void Oyun::sahneGuncelle()
22 {
23       if (m_gecenSure.asSeconds() >= m_cerceveSuresi)
24       {
25             m_konumX += m_artis;
26             m_sekil.setPosition(m_konumX, 300);
27             m_gecenSure -= sf::seconds(m_cerceveSuresi);
28       }
29 }
30 void Oyun::sahneCiz()
31 {
32       m_pencere.cizimeBasla();
33       m_pencere.ciz(m_sekil);
34       m_pencere.cizimiBitir();
35 }
36 void Oyun::saatiYenidenBaslat()
37 {
38       m_gecenSure += m_saat.restart();
39 }
40 bool Oyun::bittimi()
41 {
42       return m_pencere.kapandimi();
43 }
```

Kod 2.13. Oyun.cpp

Dikkat edecek olursanız **main** fonksiyonu içerisindeki işlemler oldukça azalmış ve daha okunabilir hale gelmiştir. Ayrıca fonksiyon içerisinde sadece Oyun nesnesi ile iletişime geçilmektedir. Nesneye dayalı programlama tekniğinin en büyük avantajlarından birisi kodun

daha kolay okunup anlaşılabilmesini sağlamasıdır. Sınıfı tasarlamak zor olsa da kullanması çok daha kolay olmaktadır.