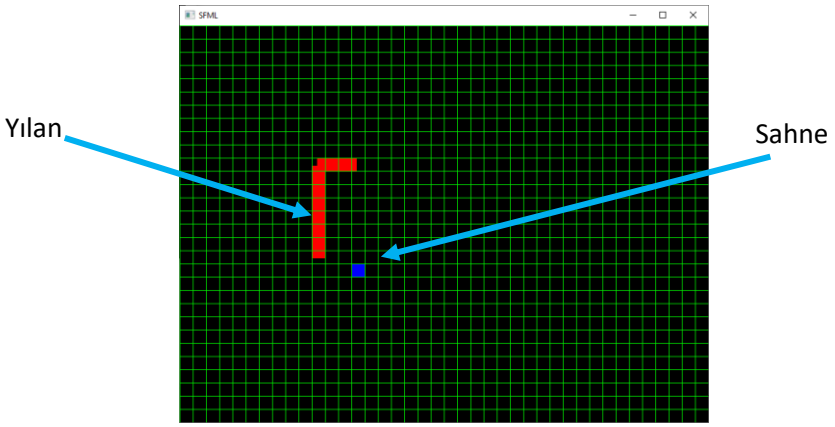
HAFTA 4: Yılan oyunu

Yılan Oyunu

Geliştireceğimiz oyunun menü tasarımını bir önceki bölümde yapmıştık. Şimdi sırada oyunu geliştirmekte. Yılan oyunu aşağıda görüldüğü gibi iki parçadan oluşacaktır. Her parçayı bir sınıf ile temsil edeceğiz. Sahne sınıfı sahne üzerindeki ızgarayı ve yemi temsil edecektir. Yilan sınıfı içerisinde her bir hücreyi temsil edecek YilanHucresi nesnelerine sahip olacaktır.



Şekil 4.1

Yılan Hücreleri

Yılan birden fazla kare şeklindeki hücrelerden oluşmaktadır. Bu kareleri temsil etmesi için **`YilanHucre`** adında bir sınıf tanımlayacağız. Bu sınıf bir kare çizmenin yanı sıra karenin hareket edeceği yönü de içerisinde tutabilmelidir. Aşağıda **`YilanHucre`** sınıfının tasarımı verilmiştir.

```
01 #pragma once
02 #pragma once
03 #include<SFML/Graphics.hpp>
04 enum class YON { BOS, SOL, SAG, YUKARI, ASAGI };
05 class YilanHucre
06 {
07 public:
08
09       YilanHucre();
10       void hareketEt(float boyut);
11       void yon(YON yeniYon);
12       void konum(sf::Vector2f yeniKonum);
13       YON  yonGetir();
14       sf::Vector2f konumGetir() const;
15 private:
16       YON m_yon;
17       sf::Vector2f m_konum;
18 };
```

Kod 4.1.

Yön bilgisini tutmak için bir **enum sınıfı** oluşturuldu. Bu sayede yönleri temsil eden isimleri hatırlamak zorunda kalmayacağız. **enum sınıflarını enum'dan** farklı yapan değerlere erişim yoludur. Örneğin aşağıda tanımlanmış **enum** değerlerine erişmek için isimlerini kullanmak yeterli olacaktır.

```
enum  YON { BOS, SOL, SAG, YUKARI, ASAGI };

YON a = BOS;
```

Aynı işlemi aşağıdaki gibi yapmaya çalışırsak derleyici hata verecektir.

```
enum class YON { BOS, SOL, SAG, YUKARI, ASAGI };

YON a = BOS;
```

Doğru kullanım aşağıdaki gibi olacaktır. Buradaki avantaj kullanıcıların değerleri temsil eden isimlere erişimini kolaylaştırmak ve hata yapmalarına izin vermemektir. Programcı olarak hangi yönlerin var olduğunu bilmemize gerek kalmadan **YON** ismini kullanarak var olan yön değerlerine erişebiliriz.

```
enum class YON { BOS, SOL, SAG, YUKARI, ASAGI };

YON a = YON::BOS;
```

YilanHucre sınıfı sadece iki tane veri üyesine sahiptir. Bu veriler hücrenin hareket edeceği yönü ve bulunduğu konumu tutacaktır. Akla şu soru gelebilir "Neden bu sınıf çizmesi gereken **RectangleShape** nesnesini tutmuyor". Her hücre için bir nesne tutmak kaynak kullanımı açısından pahalıya mal olacaktır. Bir **RectangleShape** nesnesi ile bir çerçevede de istediğimiz sayıda dörtgen çizebiliriz. Tek yapılması gereken çizim komutundan önce farklı koordinatlar vermektir. **Yilan** sınıfını tasarlarken bu konuya daha detaylı bir şekilde değineceğiz.

Kurucu fonksiyon hücreye başlangıçta varsayılan yönünü atamaktadır. Her yılan hücresi başlangıçta bu yöne sahip olacaktır. **hareket** fonksiyonu ise yılanın sahip olduğu yöne bağlı olarak yılanı hareket ettirecektir. Fonksiyon hareket miktarını argüman olarak almaktadır. Hücrenin hareket miktarı boyutu kadar olabileceği gibi farklı bir hareket miktarı da belirtilebileceğini düşünerek sınıf içerisine sabit bir değişken oluşturmaktan kaçındık. Amacımız hareket miktarı ile boyutunu bağımsız hale getirmektir. Ayrıca hücrelerin boyutunu **Yilan** sınıfı belirleyecektir. Hücrelerin boyutunu da onun tutmasına karar verdik. **YilanHucresi** sınıfına ait fonksiyonların içeriği Kod 4.2'de gösterilmiştir.

hareket fonksiyonu içerisinde öncelikle hücrenin yönünü tutan **m_yon** veri üyesinin değeri **switch** yapısı ile kontrol edilmektedir. Her yöne karşılık hücrenin koordinatındaki değişim gerçekleştirilmektedir. Örneğin hücre sağa gidecekse sadece **x** koordinatına hareket miktarı eklenmektedir. Dikey eksendeki hareketlerde dikkat edilmesi gereken **+y** ekseninin yukarıdan aşağı doğru olmasıdır. **m_konum** bir **sf::Vector2f** türünde bir nesne olduğu için toplama işlemi sırasında bu nesnenin kurucu fonksiyonu kullanılmıştır.

```cpp
#include "YilanHucre.h"

YilanHucre::YilanHucre()
{
	yon(YON::SAG);
}

void YilanHucre::hareketEt(float boyut)
{
	switch (m_yon)
	{
	case YON::SAG:
		m_konum += sf::Vector2f(boyut, 0);
		break;
	case YON::SOL:
		m_konum -= sf::Vector2f(boyut, 0);
		break;
	case YON::ASAGI:
		m_konum += sf::Vector2f(0, boyut);
		break;
	case YON::YUKARI:
		m_konum -= sf::Vector2f(0, boyut);
		break;
	}
}

void YilanHucre::yon(YON yeniYon)
{
	m_yon = yeniYon;
}

void YilanHucre::konum(sf::Vector2f yeniKonum)
{
	m_konum = yeniKonum;
}

sf::Vector2f YilanHucre::konumGetir() const
{
	return m_konum;
}

YON YilanHucre::yonGetir()
{
	return m_yon;
}
```

Kod 4.2.

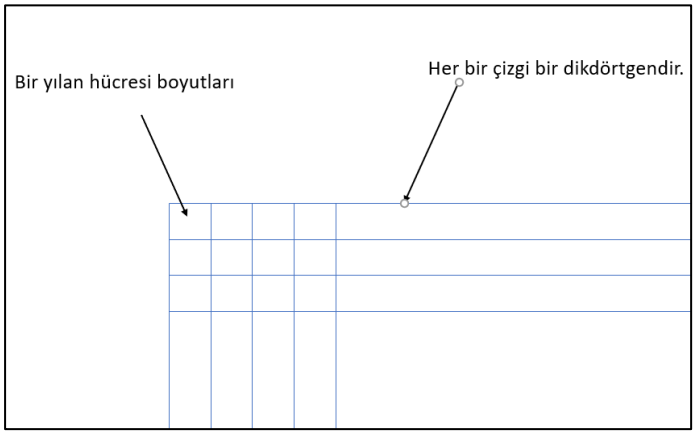
Sahne Sınıfı

Sahne sınıfı pencere üzerindeki ızgara ve yemi çizmek ile görevlidir. Ayrıca yem yenildiğinde yeni bir yem oluşturmakta **Sahne** sınıfın görevidir. **Oyun** sınıfı **Sahne** sınıfından bir nesneye sahip olacaktır. **Yilan** ile **Sahne** arasındaki ilişki, **Oyun** nesnesi tarafından sağlanacaktır. **Sahne** sınıfının prototipi aşağıdaki gibidir. **Sahne** prototipine bakınca çizimin nasıl yapılacağı ile ilgili bir fikir edinmek zor. Bunun öncelikle ızgaranın nasıl çizileceğinden bahsetmemiz gerekir.

```
01  #pragma once
02  #include<vector>
03  #include<SFML/Graphics.hpp>
04  #include<SFML/System.hpp>
05  class Sahne
06  {
07  public:
08        Sahne();
09        void olustur(float sahneGenislik,
10                     float sahneYukseklik,
11                     float hucreBoyut);
12        void ciz(sf::RenderWindow& pencere);
13        void yemOlustur();
14        sf::Vector2f yemKonumGetir();
15  private:
16        int          m_satirSayisi;
17        int          m_sutunSayisi;
18        float       m_hucreBoyutu;
20        sf::RectangleShape              m_satirHucresi;
21        sf::RectangleShape              m_sutunHucresi;
22        sf::RectangleShape              m_Yem;
23  };
```

Kod 4.3.

Sfml kütüphanesi doğru çizmek için bir araca sahip değildir (istenirse OpenGL ile yapılabilir). Bunun yerine doğru çizmek için dikdörtgen şekli kullanılacaktır. Her bir çizgi bir dikdörtgen ile temsil edilecektir. Çizgiler arası mesafe yılan hücresinin boyutu ile aynı olacaktır. Şekil 4.2'de sahne çizgilerine örnek gösterilmiştir.



Şekil 4.2

Sahne sınıfı, satır ve sütunları çizmek için iki tane **RectangleShape** nesnesi barındırmaktadır (20-21. Satırlar). Yemi çizmek için de yine bir **RectangleShape** nesnesi oluşturulmuştur (22.satır). 16 ve 17. satırlarda sahnedeki satir ve sütun sayılarını tutmak için iki değişken tanımlanmıştır. **m_hucreBoyutu** bir yılan hücresinin boyutunu tutacaktır. Bu değer **Oyun** nesnesi tarafından **Sahne** nesnesine verilecektir. **Sahne** sınıfının üyelerinden bahsettik şimdi fonksiyonlarını tanımlayalım.

Sahne sınıfının kurucusu öncelikle çizimde kullanılacak olan **RectangleShape** nesnelerinin rengini atamaktadır. Programcı olarak isterseniz bu renklerin dışarıdan değiştirilmesini sağlayacak metotları ekleyebilirsiniz. Kurucu fonksiyonun son satırında, **c++** standart kütüphanesindeki rastgele sayı üretme algoritmasını besleyecek olan ilk değeri atıyoruz. **srand** fonksiyonu algoritmayı besleyecek değeri atamakla görevlidir. Bu fonksiyon program sırasında sadece 1 defa çağrılmalıdır. Oyun da bir sahne olacağını düşünerek sahnenin kurucu fonksiyonunda bu çağrıyı gerçekleştirmek problem çıkarmayacaktır. Daha iyi bir çözüm için rastgele sayı üretme sınıfı tasarlanabilir. Program her çalıştığında **srand** fonksiyonuna farklı bir değer vermemiz gerekir aksi halde rastgele üreteceğimiz değerler hep aynı olacaktır. Örneğin ilk program çalıştığında sırayla 10,34,45 değerlerini ürettiysek bir sonraki çalıştırma da yine aynı değerleri aynı sıra ile elde ederiz. O yüzden **time** fonksiyonu kullanarak program her çalıştığında farklı değer üretilmesini sağlıyoruz.

olustur metodu üç parametre almaktadır. Parametrelerin değerleri **Oyun** nesnesi tarafından verilecektir. Bu değerler kullanılarak fonksiyonun ilk iki satırında sahnenin sütun ve satir sayısı hesaplanıp ilgili değişkenlere atanmaktadır. Hemen ardından yılan hücrelerinin boyutu saklanmaktadır. 18. Satırda çizilecek satırları temsil edecek olan dikdörtgenin boyutu belirtilmektedir. Dikdörtgenin genişliği doğal olarak sahnenin genişliği kadar olacaktır yüksekliğe 1 birim atanmıştır. Böylece dikdörtgen ekranda yatay bir çizgi şeklinde görünecektir. Aynı işlem 19.satırda sütun içinde yapılmıştır. 22.satırda **yemOlustur** metodu çağrılmaktadır. Bu metot yemin yeni koordinatlarını rastgele üretecektir.

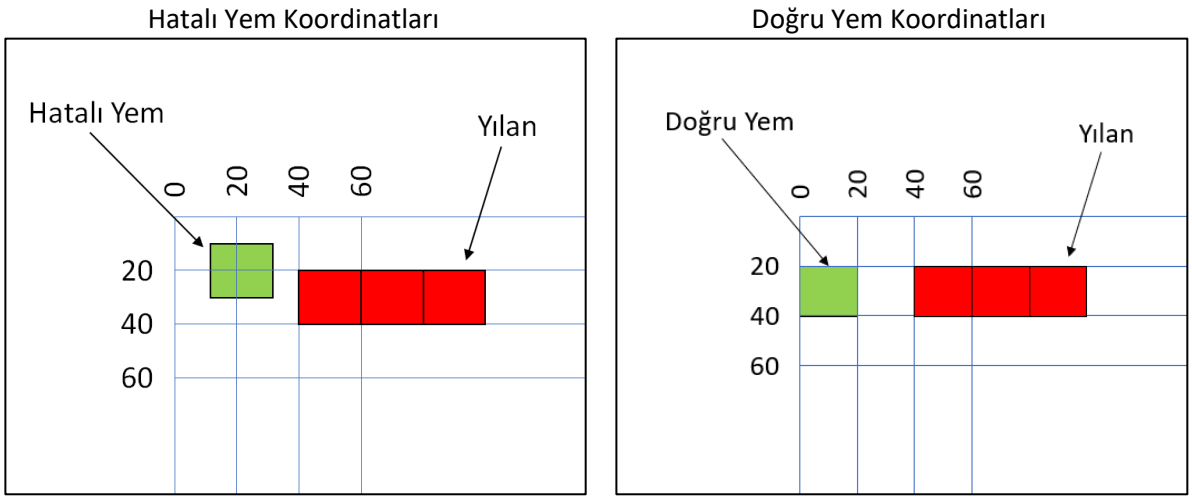
```
01 #include "Sahne.hpp"
02
03 Sahne::Sahne()
04 {
05       m_satirHucresi.setFillColor(sf::Color::Green);
06       m_sutunHucresi.setFillColor(sf::Color::Green);
07       m_Yem.setFillColor(sf::Color::Blue);
08       std::srand(std::time(nullptr));
09 }
10
11 void Sahne::olustur(float sahneGenislik,
12                     float sahneYukseklik,
13                      float hucreBoyut)
14 {
15       m_sutunSayisi = sahneGenislik / hucreBoyut;
16       m_satirSayisi = sahneYukseklik / hucreBoyut;
17       m_hucreBoyutu = hucreBoyut;
18       m_satirHucresi.setSize({ sahneGenislik,1 });
20       m_sutunHucresi.setSize({ 1,sahneYukseklik });
21       m_Yem.setSize({ hucreBoyut,hucreBoyut });
22       yemOlustur();
23 }
```

Kod 4.4.

yemolustur fonksiyonu ilk olarak yemin **x** ve **y** koordinatlarını üretmektedir. Bu işlem için öncelikle **std::rand** fonksiyonu kullanarak rastgele bir sayı üretiyoruz. Ürettiğimiz sayıların sahnenin genişliği ve yüksekliğini aşmaması gerekir. Ayrıca yemin koordinatları da yılan hücre boyutunun tam katları olmalıdır. Durumu daha iyi anlamak için Şekil 4.3'e göz atalım. Şekil de hücre boyutu 20 birim alınmıştır. Buna göre satır ve sütun çizgilerinin bir kısmı örnek olarak çizilmiştir. Yeşil olarak görünen yemin başlangıç koordinatları hücre boyutunun tam katı olmadığından satır ve sütun çizgileri işle çakışmaktadır. Yılanın satır ve sütün çizgilerine çarpmadan gideceği düşünülürse yem yılan ile tam kesişmeyecektir. Koordinatların tam kat olması için rastgele üretilen sayının satır ve sütun sayısından fazla olması engellenmektedir(3-4.satırlar). Koordinatlar bulunurken elde edilen bu değerler hücre boyutu ile çarpılmaktadır. Yemin koordinatları daima hücre boyutunun katı olacaktır.

```
01  void Sahne::yemOlustur()
02  {
03     float x = std::rand() % m_sutunSayisi;
04     float y = std::rand() % m_satirSayisi;
05     m_Yem.setPosition(sf::Vector2f(x * m_hucreBoyutu, y * m_hucreBoyutu));
06  }
07
08  sf::Vector2f Sahne::yemKonumGetir()
09  {
10         return m_Yem.getPosition();
11  }
```

Kod 4.5.



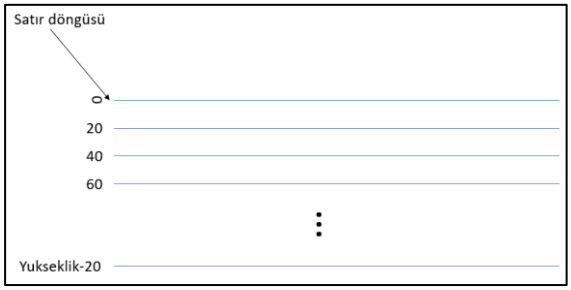
Şekil 4.3

Sahne sınıfının en karmaşık fonksiyonu **ciz** olacaktır. Fonksiyonun gövdesi Kod 4.6'da verilmiştir. İlk olarak yem çizdiriliyor (4.satır). Ardından iki döngü ile satır ve sütun çizgileri çizdiriliyor. İlk döngü satır çizgilerini çizdirmek ile görevlidir. Hücre boyutunun 20 olduğu durumda satırların çizilmesi Şekil 4.4'de gösterilmiştir. Döngü içerisinde öncelikle sıradaki satır çizgisinin **x** ve **y** koordinatı belirlenmektedir. Satırlar söz konusu olduğunda **x** değeri daima 0 olacaktır. **y** koordinatı ise her turda hücre boyutu kadar arttırılacaktır. Koordinat hesaplandıktan sonra satırı temsil eden dikdörtgen çizdirilmektedir. Yani her satır için aynı dikdörtgen farklı koordinatlarda tekrardan çizilmektedir. Sütun

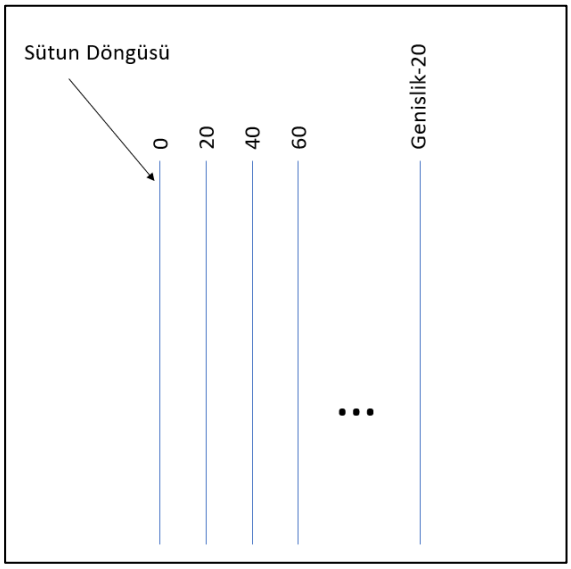
döngüsünde de neredeyse aynı işlemler yapılmaktadır. Tek fark olarak **y** koordinatı sabitken **x** koordinatı her turda artmaktadır.

```
01  void Sahne::ciz(sf::RenderWindow& pencere)
02  {
03          sf::Vector2f konum;
04          pencere.draw(m_Yem);
05          for (int satir = 0; satir < m_satirSayisi; satir++)
06          {
07                  konum.x = 0;
08                  konum.y = satir * m_hucreBoyutu;
09                  m_satirHucresi.setPosition(konum);
10                  pencere.draw(m_satirHucresi);
11          }
12          for (int sutun = 0; sutun < m_sutunSayisi; sutun++)
13          {
14                  konum.x = sutun * m_hucreBoyutu;
15                  konum.y = 0;
16                  m_sutunHucresi.setPosition(konum);
17                  pencere.draw(m_sutunHucresi);
18          }
20  }
```

Kod 4.6.



Şekil 4.4



Şekil 4.5

Şimdi sırada Sahne sınıfımızı denemekte. Sahneyi **Oyun** nesnesinin kontrol edeceğini düşünürsek öncelikle **Oyun** sınıfı içerisinde bir adet **Sahne** nesnesi eklemeliyiz. Ayrıca yılan hücre boyutunu tutması için bir değişken oluşturulmaktadır. Aşağıdaki kod parçasında bu işlem gösterilmektedir.

```
01 #pragma once
02 #include "Pencere.hpp"
03 #include "Buton.hpp"
04 #include "Sahne.hpp"
05 class Oyun
06 {
07 public:
08 //Sınıfın diğer kısımlarında değişim yok
09 //.....
10 public:
11 //Sınıfın diğer kısımlarında değişim yok
12 //.....
13    Sahne     m_sahne;
14    float m_hucreBoyutu;
15 };
```

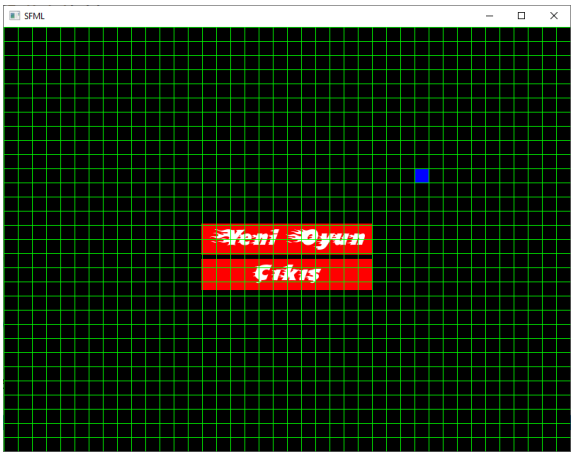
Kod 4.7.

Oyun sınıfının kurucu fonksiyonunun yeni hali aşağıdaki gibi olacaktır. Pencerenin boyutu alındıktan sonra bu değerler sahnenin **olustur** metoduna verilmektedir(7.satır). Hücre boyutu olarak şimdilik 20 değerini atadık. Sahne çiz fonksiyonunda sadece bir satır eklenecektir (13.satır).

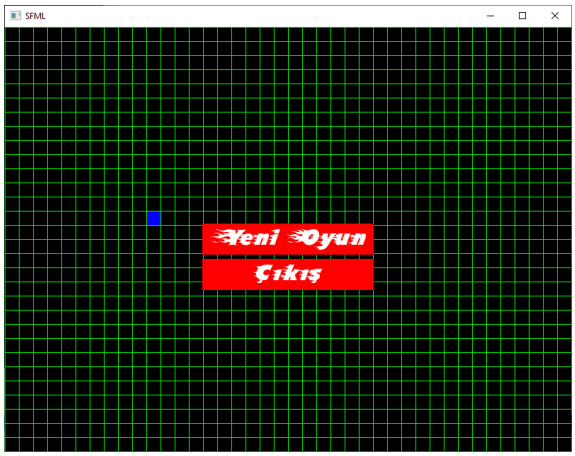
```
01 Oyun::Oyun()
02 {
03 // diğer kısımlarında değişim yok
04 //.....
05       m_hucreBoyutu = 20.0f;
06       auto boyut = m_pencere.pencereGetir().getSize();
07       m_sahne.olustur(boyut.x, boyut.y, m_hucreBoyutu);
08 }
09 void Oyun::sahneCiz()
10 {
11       m_pencere.cizimeBasla();
12       menuCiz();
13       m_sahne.ciz(m_pencere.pencereGetir());
14       m_pencere.cizimiBitir();
15 }
```

Kod 4.8.

Program çalıştırıldığında elde edilecek sonuç şekil 4.6'da verilmiştir. Sahnenin çiziminde herhangi problem olmadığı görülmektedir. Yem ve çizgiler doğru bir şekilde çizilmiştir. Problem sahnenin menünün üstüne çizilmesidir. Sahne çiziminin menünün altına gelmesini istiyorsak tek yapmamız gereken Kod 4.8'deki 12 ile 13. satırı yer değiştirmektir. Menü sahneden sonra çizildiği için problem kalmayacaktır (Şekil 4.7).



Şekil 4.6



Şekil 4.7

Menüye Dönüş

Sahnenin menüdeki seçimden sonra çizilmesini istiyorsak bir şekilde menüden geri besleme almamız gerekir. Örneğin çıkış butonuna basıldığında uygulama kapatılsın ve yeni oyun butonuna basıldığında sahne ile yılan çizilmeye başlasın. Bu işlemleri gerçekleştirebilmek için öncelikle **Oyun** sınıfında iki değişken oluşturulması gerekir. Aşağıda **Oyun** sınıfına eklenen iki değişken gösterilmektedir. Bu değişkenlerin değerleri kurucu fonksiyonda Kod 4.10'daki gibi atanacaktır. Değişkenler menüdeki tuşlara basılıp basılmadığını anlamamızı sağlayacaktır.

```
01 #pragma once
02 #include "Pencere.hpp"
03 #include "Buton.hpp"
04 #include "Sahne.hpp"
05 class Oyun
06 {
07 public:
08 //Sınıfın diğer kısımlarında değişim yok
09 //.....
10 private:
11 //Sınıfın diğer kısımlarında değişim yok
12 //.....
13       bool   m_kapalimi;
14       bool   m_yeniOyunTiklandimi;
15 };
```

Kod 4.9.

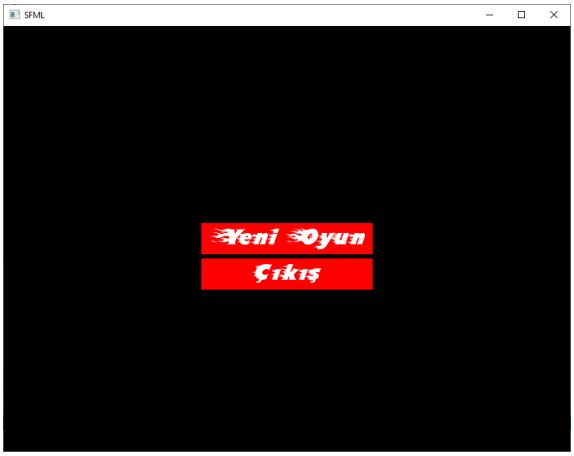
```
01 #include "Oyun.hpp"
02 #include<iostream>
03
04 Oyun::Oyun()
05 {
06       m_kapalimi = false;
07       m_yeniOyunTiklandimi = false;
08       // diğer kısımlarında değişim yok
09 }
```

Kod 4.10.

"Yeni Oyun" butonuna basılmadan sahnenin çizilmesini istemiyoruz. Bu yüzden Oyun sınıfına ait olan **sahneCiz** fonksiyonunu aşağıdaki gibi değiştireceğiz. Böylece **m_yeniOyunTiklandimi** değişkeninin değeri **true** olana kadar sahne çizilmeyecektir. Şekil 4.8'de program çalıştırıldığında elde edilen sonuç gösterilmiştir.

```
01  void Oyun::sahneCiz()
02  {
03        m_pencere.cizimeBasla();
04
05        if (m_yeniOyunTiklandimi)
06        {
07              m_sahne.ciz(m_pencere.pencereGetir());
08        }
09        else
10        {
11              menuCiz();
12        }
13
14        m_pencere.cizimiBitir();
15  }
```

Kod 4.11.

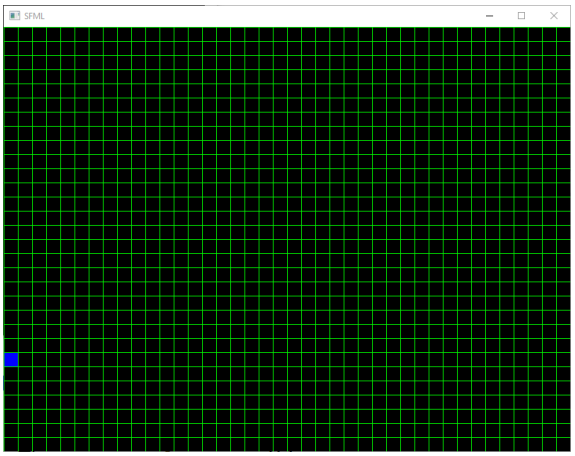


Şekil 4.8

Önceki bölümde, menüdeki butonlara tıklanınca çağrılacak iki fonksiyon belirlemiştik. **"Yeni Oyun"** butonuna tıklandığında **Oyun** sınıfının **btnYeniOyunTikla** fonksiyonu çağrılacaktır. Bu fonksiyonun gövdesini aşağıdaki gibi değiştireceğiz. Bu sayede "yeni oyun" butonu tıklandığında artık menü çizilmeyecek ve sadece sahne çizilecektir. Sonuç şekil 4.9'da gösterilmiştir.

```
01  void Oyun::btnYeniOyunTikla()
02  {
03        m_yeniOyunTiklandimi = true;
04  }
```

Kod 4.12.



Şekil 4.9

Oyun sınıfının **bittimi** fonksiyonu pencerenin kapatılıp kapatılmadığını belirtmek için kullanmıştık. **main** fonksiyonu içindeki döngümüz bu fonksiyonu kontrol ederek dönmektedir. Eğer çıkış butonunun tıklanma işlemini **bittimi** fonksiyonunun dönüş değerine bir şekilde bağlarsak butona basıldığında programın sonlanmasını sağlayabiliriz.

```
01 int main()
02 {
03         Oyun oyun;
04         while (!oyun.bittimi())
05         {
06                 oyun.girisKontrol();
07                 oyun.sahneGuncelle();
08                 oyun.sahneCiz();
09                 oyun.saatiYenidenBaslat();
10         }
11         return 0;
12 }
```

Kod 4.13.

Öncelikle çıkış butonuna basıldığında Oyun sınıfının **m_kapalimi** değişkeninin değerini **true** yapmamız gerekir. "Yeni oyun" butonunda olduğu gibi çıkış butonuna tıklandığında çağrılacak bir fonksiyonumuz bulunmaktadır. Bu fonksiyonun gövdesinde aşağıdaki gibi olacaktır. **Oyun** sınıfının **bittimi** fonksiyonunu da Kod 4.15'deki gibi değiştireceğiz. Bu sayede fonksiyon öncelikle çıkış butonuna basılıp basılmadığını kontrol etmektedir. Eğer tuşa basıldığına karar verirse **true** değeri döndürerek çağrıyı bitirmektedir. Aksi durumda önceden olduğu gibi pencerenin durumunu döndürmektedir.

```
01 void Oyun::btnCikisTikla()
02 {
03         m_kapalimi = true;
04 }
```

Kod 4.14.

```
01 bool Oyun::bittimi()
02 {
03       if (m_kapalimi)
04             return true;
05       return m_pencere.kapandimi();
06 }
```

Kod 4.15.

Şimdi programı çalışırken çıkış butonuna basarsanız program sonlandırılacaktır.

Yılan Sınıfı

Yilan sınıfı adında oyunumuzdaki yılanı temsil edecektir. Kod 4.16'da **Yilan** sınıfının ilk tanımı verilmiştir. Sınıf içerisinde yılana ait her bir hücrenin tutulacağı bir liste oluşturulmuştur (15.satır). **HucreListesi "YilanHucre.hpp"** dosyasının sonunda tanımlanmıştır. 16. Satırda bütün hücrelerin çiziminde kullanılacak olan **RectangleShape** nesnesi tanımlanmaktadır. 17. Satırda yılan hücrelerinin boyutunu tutacak değişken tanımlanmıştır.

```
      typedef std::vector<YilanHucre> HucreListesi;
```

```
01 #pragma once
02 #include "YilanHucre.h"
03 #include <SFML/Graphics.hpp>
04 class Yilan
05 {
06 public:
07       Yilan();
08       void  ciz(sf::RenderWindow& pencere);
09       void  kuyrugaEkle();
10       void  hareket(float mesafe);
11       float hucreBoyutGetir();
12       void  yonDegistir(YON yeniYon);
13 private:
14
15       HucreListesi       m_hucreler;
16       sf::RectangleShape m_hucreSekil;
17       float              m_hucreBoyutu;
18 };
```

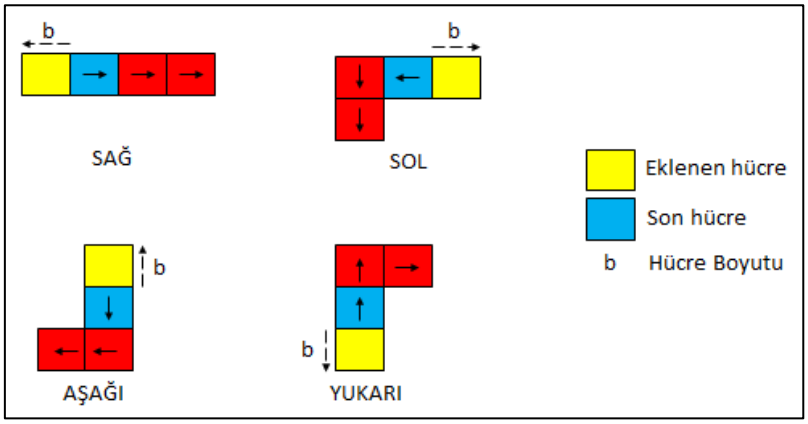
Kod 4.16.

Yilan sınıfının en önemli fonksiyonlarından birisi olan **kuyrugaEkle** Kod 4.17'de gösterilmiştir. Fonksiyon her çağrıldığında kuyruğun sonuna bir tane yılan hücresi eklemektedir. Kuyruğa ekleme işlemi yılanın son hücresinin yönü ve konumuna göre yapılmaktadır. Yeni hücre yılanın son hücresi ile aynı yönde olacaktır fakat konumu bir hücre boyutu tersi yönde olacaktır. Şekil 4.10'da yılanın son hücresinin farklı yönlere sahip olması durumunda yeni eklenecek hücrenin konumu gösterilmiştir. Bu kural ilk hücreyi eklerken uygulanmayacaktır. O yüzden 4. satırda fonksiyonumuz hücre sayısını kontrol etmektedir. Eğer hücre sayısı sıfır ise sadece ilk hücre listeye eklenmekte ve fonksiyon bitirilmektedir.

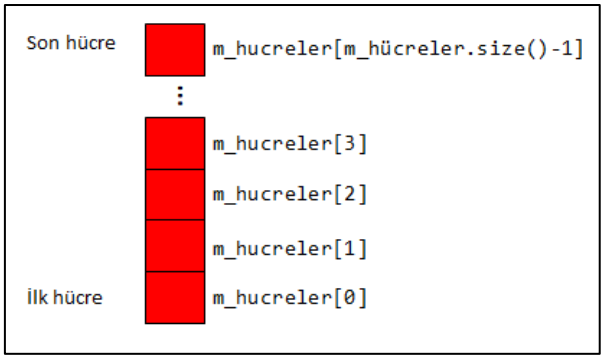
```
01  void Yilan::kuyrugaEkle()
02  {
03          int sonIndis = m_hucreler.size() - 1;
04          if (sonIndis < 0)
05          {
06                  m_hucreler.push_back(YilanHucre());
07                  return;
08          }
09          auto konum = m_hucreler[sonIndis].konumGetir();
10          auto yon = m_hucreler[sonIndis].yonGetir();
11          switch (yon)
12          {
13          case YON::SAG:
14                  konum.x -= m_hucreBoyutu;
15                  break;
16          case YON::SOL:
17                  konum.x += m_hucreBoyutu;
18                  break;
19          case YON::ASAGI:
20                  konum.y -= m_hucreBoyutu;
21                  break;
22          case YON::YUKARI:
23                  konum.y += m_hucreBoyutu;
24                  break;
25          }
26          YilanHucre yeniHucre;
27          yeniHucre.konum(konum);
28          yeniHucre.yon(yon);
29          m_hucreler.push_back(yeniHucre);
30  }
```

Kod 4.17.

Eğer fonksiyon 9.satıra gelirse yılanın bir veya daha fazla hücresi bulunmaktadır. Buna göre önce yılanın son hücresinin konumunu ve yönü ayrı değişkenlere atanmaktadır(Bu değişkenler yeni hücrenin konumu ve yönü olacaktır). Daha sonra bir **`switch`** ifadesi ile yılanın son hücresinin yönü yakalanmaktadır. Yönün her duruma göre son hücreden alınan konum bilgisi değiştirilmektir. Örneğin son hücre sağa doğru hareket ediyorsa konum değişkeni **x** koordinatında bir hücre boyutu kadar sola ötelenmektedir (14.satır). **`switch`** ifadesi bittikten sonra yeni hücremiz için bir nesne oluşturuyoruz ve bu nesneye güncellediğimiz koordinat bilgisi ile yon bilgisini atıyoruz. Son olarak hücre listesine yeni hücremizi ekliyoruz. **`m_hucreler`** listesi yılan hücrelerini konumlarına göre saklamaktadır. Örneğin listenin 0. elemanı yılanın ilk (baş) hücresidir. Şekil 4.11'de yılan hücrelerinin listeye yerleştirilme sırası gösterilmiştir.



Şekil 4.10



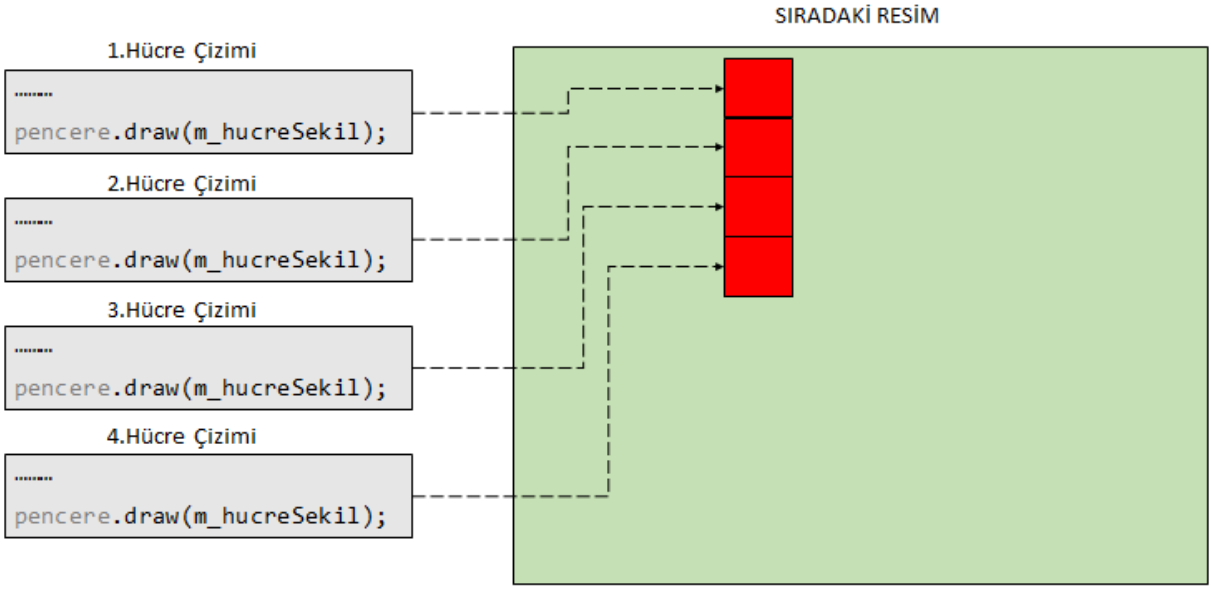
Şekil 4.11

Yilan sınıfına ait diğer fonksiyonlar aşağıdaki gibidir. Fonksiyonlar zamanla değişime uğraşacaktır. Öncelikle kurucu fonksiyonu inceleyelim. İlk olarak hücre boyutu atanmaktadır. Şimdilik hücre boyutunu sabit değer olarak yılan içerisinde atayacağız. Ardından yılan hücrelerinin rengi ve boyutu atanmaktadır. Son olarak yılana bir hücre eklemek için **kuyrugaEkle** fonksiyonu çağrılmaktadır. Başlangıç olarak yılan hücrelerinin hareketini doğru şekilde yapmasını amaçlıyoruz. O yüzden sadece 1 hücre eklendi.

```
01  Yilan::Yilan()
02  {
03        m_hucreBoyutu = 20.0f;
04        m_hucreSekil.setFillColor(sf::Color::Red);
05        m_hucreSekil.setSize({ m_hucreBoyutu,m_hucreBoyutu });
06        kuyrugaEkle();
07  }
08  void Yilan::ciz(sf::RenderWindow& pencere)
09  {
10        for (auto siradaki : m_hucreler)
11        {
12              m_hucreSekil.setPosition(siradaki.konumGetir());
13              pencere.draw(m_hucreSekil);
14        }
15  }
16  float Yilan::hucreBoyutGetir()
17  {
18        return m_hucreBoyutu;
19  }
20  void Yilan::hareket(float mesafe)
21  {
22        for (auto& siradaki : m_hucreler)
23              siradaki.hareketEt(mesafe);
24  }
25  void Yilan::yonDegistir(YON yeniYon)
26  {
27        m_hucreler[0].yon(yeniYon);
28  }
```

Kod 4.18.

ciz fonksiyonu yılanın sahip olduğu bütün hücreleri gezmektedir. Her hücre için **m_hucreSekil** dikdörtgen şekli bir defa çizilmektedir. Çizimden önce dikdörtgen şeklinin konumu sıradaki yılan hücresi ile aynı yapılmaktadır. Pencere nesnesi çizimi bitirmediği sürece aynı dikdörtgen şeklinden istediğimiz kadar çizebiliriz. Şekil 4.11'de birden fazla yılan hücresinin aynı **m_hucreSekil** nesnesi ile çizilmesi gösterilmiştir. Pencere çizimi bitirene kadar yapılan çizimler oluşturulacak resme eklenmektedir.



Şekil 4.11

Yılan Nesnesinin Oyuna Eklenmesi

Yılan sınıfını uygulamamıza **Oyun** sınıfı üzerinden ekleyeceğiz. Öncelikle Oyun sınıfına aşağıdaki kodda gösterilen 12.satırı ekleyeceğiz. Böylece **Oyun** sınıfımız bir adet **Yilan** nesnesine sahip olacaktır.

```
01  #pragma once
02  #include "Pencere.hpp"
03  #include "Buton.hpp"
04  #include "Sahne.hpp"
05  #include "Yilan.hpp"
06  class Oyun
07  {
08  //.....
09  private:
10  //Sınıfın diğer kısımlarında değişim yok
11  //.....
12        Yilan              m_yilan;
13  };
```

Kod 4.20.

Yılanın çizilebilmesi için **Oyun** sınıfının **sahneCiz** fonksiyonuna aşağıdaki kodda görüldüğü gibi 7. satır eklenecektir. Eğer program bu şekilde derlenip çalıştırılırsa yılanın çizildiği fakat hareket etmediği görülecektir.

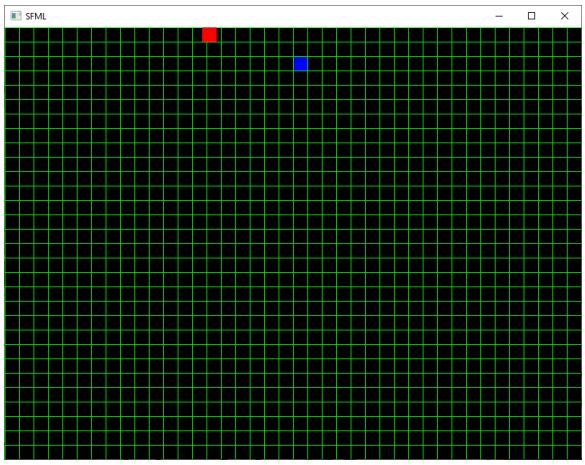
```
01  void Oyun::sahneCiz()
02  {
03          m_pencere.cizimeBasla();
04          if (m_yeniOyunTiklandimi)
05          {
06                  m_sahne.ciz(m_pencere.pencereGetir());
07                  m_yilan.ciz(m_pencere.pencereGetir());
08          }
09          else
10          {
11                  menuCiz();
12          }
13          m_pencere.cizimiBitir();
14  }
```

Kod 4.21.

Yılanın oyun içinde hareket edebilmesi için **Oyun** sınıfının **sahneGuncelle** fonksiyonu aşağıdaki gibi güncellenecektir. **hareket** fonksiyonuna mesafe olarak 1 verilmiştir. Fonksiyonun 1 saniyede 60 defa çağrılacağı düşünülürse yılanın tam 60 birimlik mesafe kat etmesine sebep olacaktır. Mesafe bir hücrenin boyut kadar yapılırsa yılanın kontrol edilemeyecek şekilde hızlı hareket ettiğini göreceğiz.

```
01  void Oyun::sahneGuncelle()
02  {
03          if (m_gecenSure.asSeconds() >= m_cerceveSuresi)
04          {
05
06                  m_gecenSure -= sf::seconds(m_cerceveSuresi);
07                  m_yilan.hareket(1);
08          }
09  }
```

Kod 4.22.



Şekil 4.12

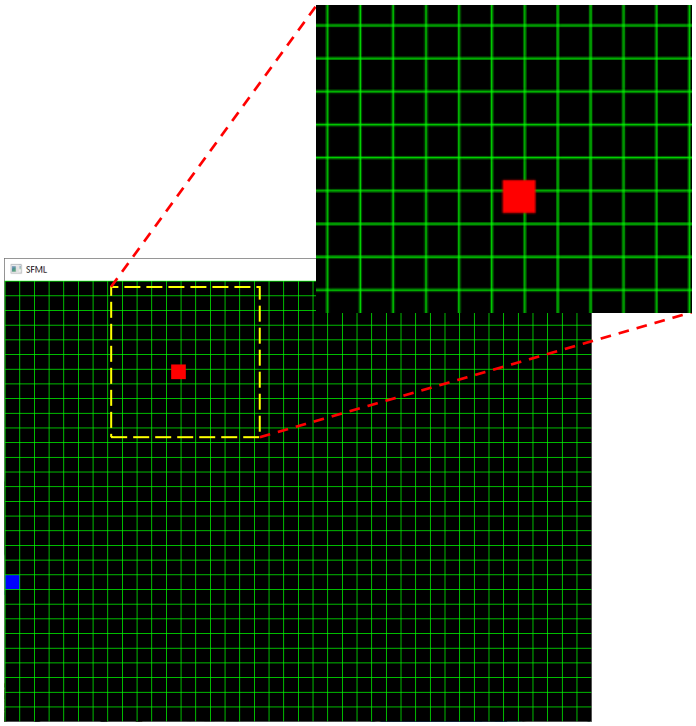
Yılanın Yön Değiştirmesi

Yılanın yön değiştirebilmesi **Oyun** sınıfı tarafından gerçekleştirilecektir. Öncelikle kullanıcın sol ve sağ tuşlara basmasına göre yılanın yönlerini değiştireceğiz. Bu işlemi **Oyun** sınıfının **girisKontrol** fonksiyonunda aşağıdaki gibi gerçekleştireceğiz. Uygulamayı çalıştırıp aşağı yön tuşuna bastığımızda Şekil 4.13'de gösterilen problem ile karşılaşılacaktır.

```
01 void Oyun::girisKontrol()
02 {
03       m_pencere.olayKontrol();
04
05       if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right))
06             m_yilan.yonDegistir(YON::SAG);
07       if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
08             m_yilan.yonDegistir(YON::SOL);
09       if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up))
10             m_yilan.yonDegistir(YON::YUKARI);
11       if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down))
12             m_yilan.yonDegistir(YON::ASAGI);
13 }
```

Kod 4.23.

Aşağıdaki şekilde yılan hücresinin yönü değiştirildiğindeki görüntüsü verilmiştir. Problemi daha iyi görünebilmesi için hücrenin etrafı yakınlaştırılmıştır. Dikkat edilecek olursa yılan hücresinin sahne çizgilerinin üzerinde hareket ettiği görülmektedir. Bunun temel sebebi yılanın her resim çizildiğindeki hareket miktarı yılanın boyutuna eşit olmamasıdır.

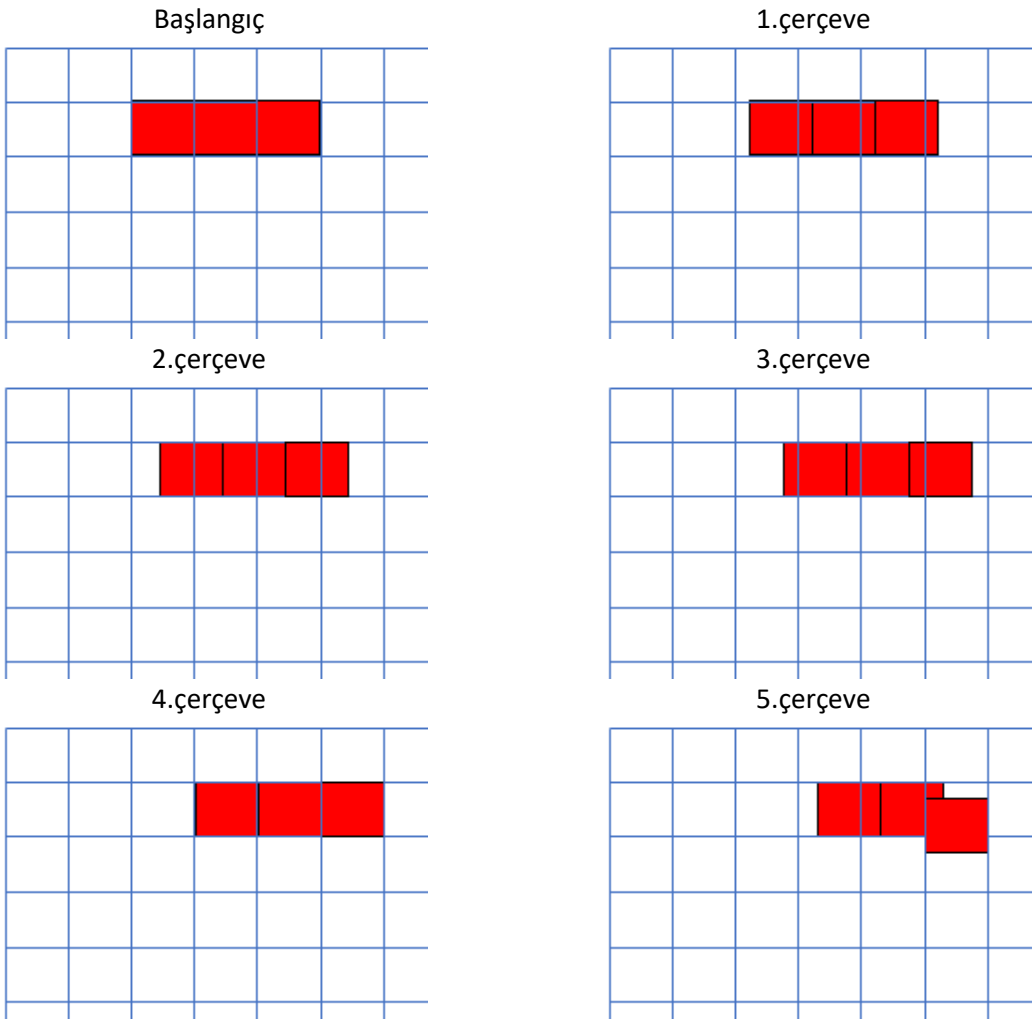


Şekil 4.13

Eğer **sahneGuncelle** fonksiyonunda yılanın hareketini 1 birim yerine yılan hücrelerinin boyutu olan 20 yaparsak problem çözülecektir. Fakat bu seferde yılan kontrol edilemeyecek şekilde hızlı hareket edecektir. Hızı düşürmek için **m_cerceveSuresi** değişenin değerini yükseltebiliriz. Örneğin Oyun sınıfının kurucu fonksiyonundaki **m_cerceveSuresine** değer atama işlemini aşağıdaki şekilde değiştirebiliriz. Bu işlem problemimizi çözecektir. Fakat şimdide yılan hücresinin hareketi akıcı değildir. Kimi oyunlar için problem olmasa da bizim yazacağımız oyunun daha akıcı hareket etmesini istiyoruz.

```
m_cerceveSuresi = 1.0f / 10.0f;
```

Hareketin akıcı olması için yılanın hareket süresini kısıtlamak yerine dönüş zamanını kısıtlamak daha mantıklı olacaktır. Yani yılanın yönünü değiştirme işlemi sadece belirli zaman aralıklarında yapılabilmelidir. Şekil 4.14'de yılanın hareketi boyutundan daha az miktar ile hareket etmektedir. Örneğin yılanın boyutu 20 birim olsun hareket miktarı ise 5 birim olduğunda yılanın kendi boyutu kadar yön alması için 4 defa **sahneGuncelle** fonksiyonunun çağrılması gerekir. Şekil 4.14'de bu işlemler gösterilmiştir. Her resim(çerçeve) oluşturulurken öncelikle **sahneGuncelle** çağrılmaktadır. Bu fonksiyon her çağrıldığında yılan hücrelerini 4 birim öteleyecektir. Bu süreç devam ederken yılan yönü sadece 5. çerçeve çizilmeden önce değiştirilebilmelidir. Aksi taktirde yılan sahne çizgileri arasında geçmeyecek ve çarpılma testleri daha karmaşık hale gelecektir. Peki bu işlemi nasıl gerçekleştireceğiz.



Şekil 4.14

Öncelikle **Oyun** sınıfı içerisinde yılanın bir hücre boyu kadar yol aldığını belirlememizi sağlayacak olan bir değişken oluşturmamız gerekiyor. Oyun sınıfına aşağıdaki değişkeni ekleyip kurucu fonksiyon içerisinde sıfır değeri atacağız. Yılan her hareket ettiğinde **m_alinanMesafe** değişkenine hareket miktarı eklenecektir. **m_alinanMesafe** hücre boyutu kadar olduğunda yön değiştirme işlemini gerçekleştireceğiz.

```
float m_alinanMesafe;
```

Yön değiştirme işlemi şu an için klavye yön tuşlarına basıldığında yapılmaktadır. Bunun yerine basılan tuş bilgisini yani istenilen yönü **Oyun** sınıfı içerisine ekleyeceğimiz bir değişken içerisinde saklayacağız. Bu değişken aşağıdaki gibi olacaktır. Oyun sınıfının kurucusunda bu değişkene **YON::SAG** değeri atanacaktır. Kod 4.24 ve Kod 4.25'de gösterilmiştir.

```
YON    m_siradakiYon;
```

```
01 #pragma once
02 #include "Pencere.hpp"
03 #include "Buton.hpp"
04 #include "Sahne.hpp"
05 #include "Yilan.hpp"
06 class Oyun
07 {
08 //.....
09 private:
10 //Sınıfın diğer kısımlarında değişim yok
11 //.....
12       YON    m_siradakiYon;
13       float m_alinanMesafe;
14 };
```

Kod 4.24.

```
01 #include "Oyun.hpp"
02 #include<iostream>
03
04 Oyun::Oyun()
05 {
06 //Sınıfın diğer kısımlarında değişim yok
07 //.....
08       m_alinanMesafe = 0.0f;
09       m_siradakiYon = YON::SAG;
10 }
```

Kod 4.25.

Oyun sınıfının **sahneGuncelle** fonksiyonu da Kod 4.26'daki gibi ve **girisKontrol** fonksiyonu da kod 4.27'deki gibi güncellenecektir. **girisKontrol** yön bilgisini **m_siradakiYon** değişkenine atamaktadır. **sahneGuncelle** fonksiyonunun 12. satırında yılan hücresinin aldığı yol devamlı olarak arttırılmaktadır. 6.satırda hücrenin aldığı yol bir hücre boyu kadar olduğu durum yakalanmaktadır. Koşul içerisinde yılanın yönü değiştirilmekte ve yılanın aldığı yön başlangıç durumuna getirilmektedir. Burada dikkat edilmesi gereken diğer bir husus yılanın bir hareket miktarı hücre boyutuna tam bölünmelidir aksi halde taşmalar olacaktır.

```
01  void Oyun::girisKontrol()
02  {
03        m_pencere.olayKontrol();
04
05        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right))
06              m_siradakiYon = YON::SAG;
07        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
08              m_siradakiYon = YON::SOL;
09        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up))
10              m_siradakiYon = YON::YUKARI;
11        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down))
12              m_siradakiYon = YON::ASAGI;
13  }
```

Kod 4.26.

```
01  void Oyun::sahneGuncelle()
02  {
03        if (m_gecenSure.asSeconds() >= m_cerceveSuresi)
04        {
05              m_gecenSure -= sf::seconds(m_cerceveSuresi);
06              if (m_alinanMesafe >= m_hucreBoyutu)
07              {
08                    m_yilan.yonDegistir(m_siradakiYon);
09                    m_alinanMesafe -= m_hucreBoyutu;
10              }
11              m_yilan.hareket(2);
12              m_alinanMesafe += 2;
13        }
14  }
```

Kod 4.27.

Yılan hareketi halledildikten sonra sırada yılanın hücre sayısını arttırmaktır. **Yılan** sınıfının kurucu fonksiyonunda **kuyrugaEkle** fonksiyonu aşağıdaki gibi 4 defa daha çağırarak yılanın hücre sayısını arttırıyoruz.

```
01  Yilan::Yilan()
02  {
03        m_hucreBoyutu = 20.0f;
04        m_hucreSekil.setFillColor(sf::Color::Red);
05        m_hucreSekil.setSize({ m_hucreBoyutu,m_hucreBoyutu });
06        for (int i = 0; i < 5; i++)
07              kuyrugaEkle();
08  }
```

Kod 4.28.

Programı çalıştırdığımızda sonuç Şekil 4.15'deki gibi olacaktır. Şekle dikkat edecek olursak, yılanın ilk hücresi aşağı giderken diğer hücreler sağa gitmeye devam etmektedir. Bunun sebebi yılan sınıfına ait **yonDegistir** fonksiyonu sadece yılanın ilk hücresinin yönünü değiştirmektedir.

Şekil 4.15

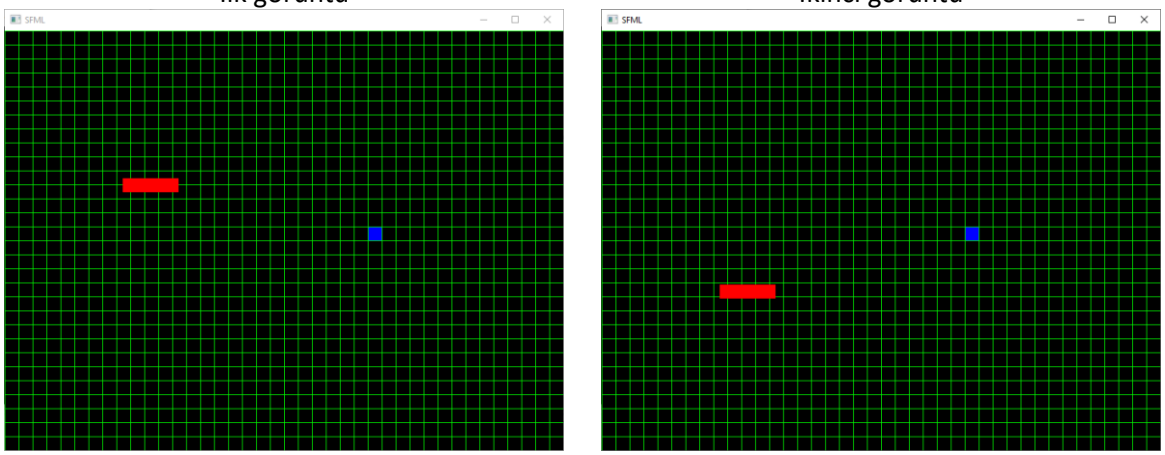
Bu problemin ortadan kalkması için ilk hücrenin yönünü diğer hücrelere de aktarmamız gerekir. Bu problemin üstesinden gelmek için **Yilan** sınıfının **hareketet** fonksiyonunu kullanabiliriz. Fonksiyon içerisinde ilk hücrenin yönünü bir sonrakine aktaracak bir kod satırını aşağıdaki gibi ekleyebiliriz. Değişimden sonra programı çalıştırıp yine aşağı tuşuna bastığımızda şekil 4.16'de gösterilen sonuç elde edilecektir. Bu sefer problemimiz ilk yılan hücresinin yönü bir seferde bütün hücrelere aktarılmıştır. Böylece yön değişimi aynı anda bütün hücreleri etkilemiştir.

```
01 void Yilan::hareketet(float mesafe)
02 {
03         int hucreSayisi = m_hucreler.size();
04         for (int i = 0; i < hucreSayisi-1; i++)
05         {
06                 YON yeniYon = m_hucreler[i].yonGetir();
07                 m_hucreler[i + 1].yon(yeniYon);
08         }
09         for (auto& siradaki : m_hucreler)
10         {
11                 siradaki.hareketEt(mesafe);
12         }
13 }
```

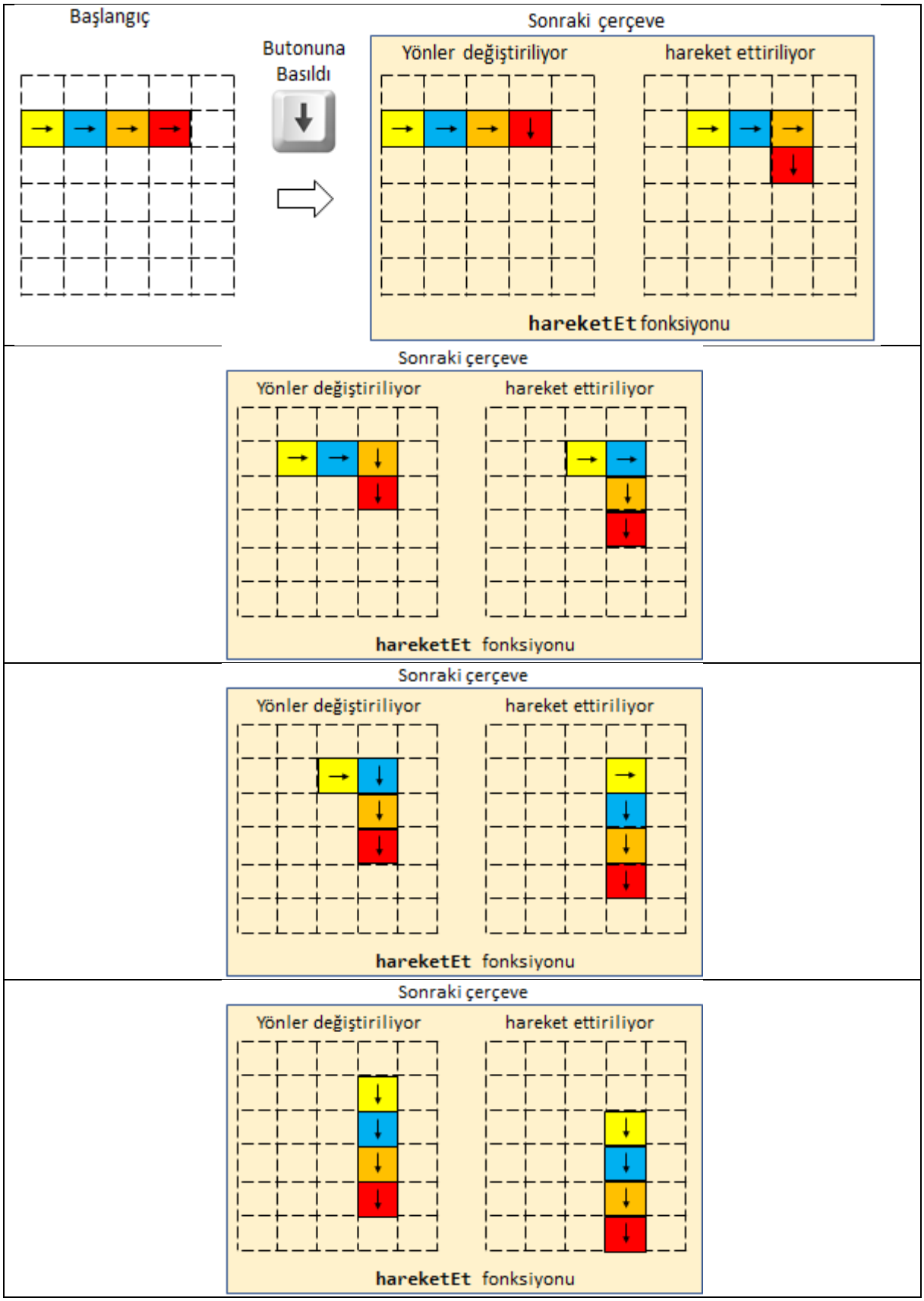
Kod 4.29.

İlk görüntü

İkinci görüntü



Şekil 4.16


Başlangıç
Butonuna Basıldı
Sonraki çerçeve
Yönler değiştiriliyor
hareket ettiriliyor
hareketEt fonksiyonu
Sonraki çerçeve
Yönler değiştiriliyor
hareket ettiriliyor
hareketEt fonksiyonu
Sonraki çerçeve
Yönler değiştiriliyor
hareket ettiriliyor
hareketEt fonksiyonu
Sonraki çerçeve
Yönler değiştiriliyor
hareket ettiriliyor
hareketEt fonksiyonu

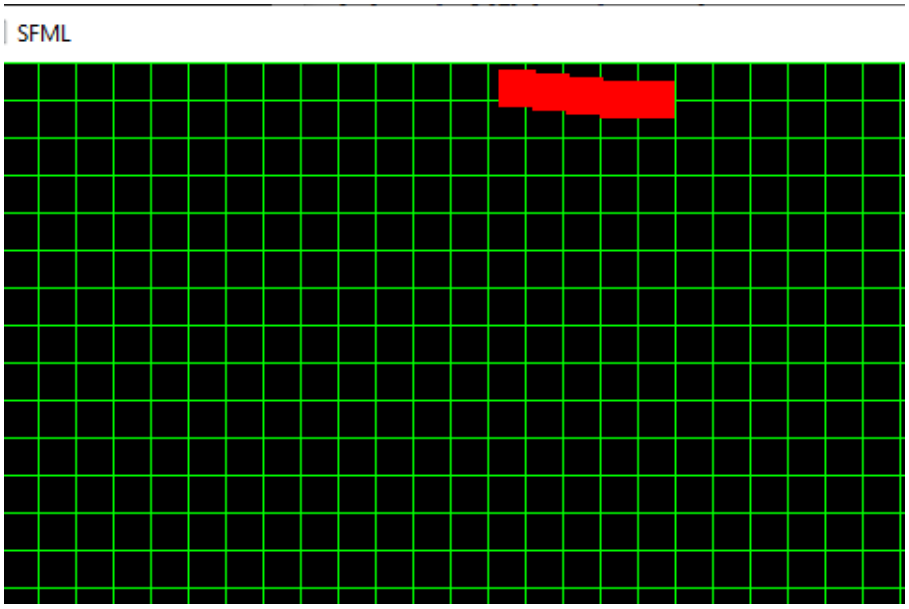
Şekil 4.17

Şekil 4.17'de sağa doğru hareket eden dört hücreye sahip bir yılan şekli görülmektedir. Kullanıcının aşağı tuşuna basmasıyla hücrelerin yönünün her bir çerçeve ile değişmesi gösterilmektedir. Dikkat edecek olursak her **hareketet** çağrıldığında sadece bir hücrenin yönü değişmektedir. Yönü değişen hücre ise baştan başlayıp sona doğru gitmektedir. Böyle bir dönüşümü sağlamak için hareket fonksiyonumuzdaki döngümüzü aşağıdaki gibi değiştirmemiz gerekmektedir.

```
01 void Yilan::hareket(float mesafe)
02 {
03         int hucreSayisi = m_hucreler.size();
04         for (int i = hucreSayisi - 1; i>0; i--)
05         {
06                 YON yeniYon = m_hucreler[i-1].yonGetir();
07                 m_hucreler[i].yon(yeniYon);
08         }
09         for (auto& siradaki : m_hucreler)
10         {
11                 siradaki.hareketEt(mesafe);
12         }
13 }
```

Kod 4.29.

hareketet fonksiyonunun yeni haliyle programı çalıştırıp yılanın yönünü aşağı olacak şekilde değiştirdiğimizde sonuç aşağıdaki gibi olacaktır. Dikkat edecek olursanız yönlerin aktarılması doğru olmasına rağmen aktarma işleminin çok hızlı gerçekleştiği görülmektedir. İlk hücrenin yön değişimi yılan hücre boyutu kadar mesafe aldığında gerçekleştirilmiştir. Aynı işlem yönlerin aktarımı içinde yapılmalıdır. Fakat hareket işlemi her çerçeve süresinde gerçekleşmelidir. Bu yüzden yön aktarımını hareketet fonksiyonundan çıkartmamız gerekmektedir.



Şekil 4.18

Yön aktarımı için **Yilan** sınıfından **yonleriAktar** sınıfı tanımlanıp gövdesi aşağıdaki gibi yapılacaktır.

```
01 void Yilan::yonleriAktar()
02 {
03       int hucreSayisi = m_hucreler.size();
04       for (int i = hucreSayisi - 1; i > 0; i--)
05       {
06             YON yeniYon = m_hucreler[i - 1].yonGetir();
07             m_hucreler[i].yon(yeniYon);
08       }
09 }
```

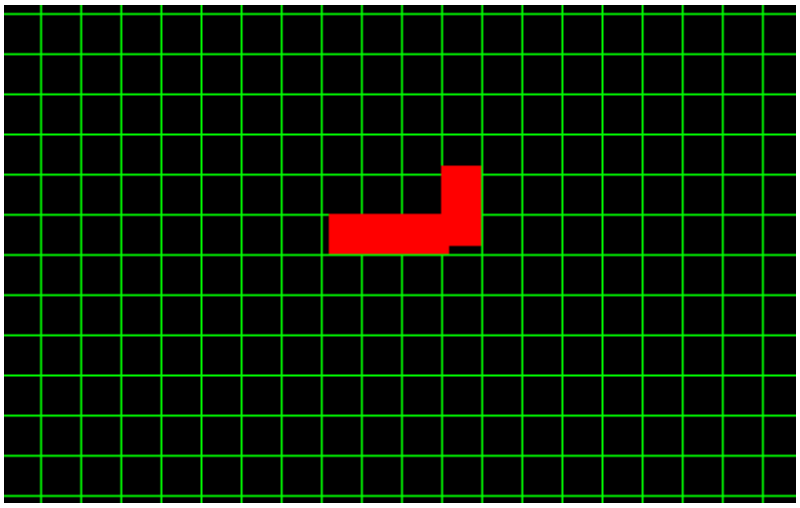
Kod 4.30.

Oyun sınıfına ait **sahneGuncelle** fonksiyonu da aşağıdaki gibi güncellenecektir. Yılanın yönlerini aktarma işlemi sadece alınan mesafe hücre boyutuna eşit olduğunda yapılacaktır.

```
01 void Oyun::sahneGuncelle()
02 {
03       if (m_gecenSure.asSeconds() >= m_cerceveSuresi)
04       {
05             m_gecenSure -= sf::seconds(m_cerceveSuresi);
06             if (m_alinanMesafe >= m_hucreBoyutu)
07             {
08                   m_yilan.yonleriAktar();
09                   m_yilan.yonDegistir(m_siradakiYon);
10                   m_alinanMesafe -= m_hucreBoyutu;
11             }
12             m_yilan.hareket(2);
13             m_alinanMesafe += 2;
14       }
15 }
```

Kod 4.31.

Şekil 4.19'da programın son hali görünmektedir. Yılan hücreleri yönlerini doğru şekilde aktarmaktadır. Yılan hücrelerinin her hareketi hücre boyutundan küçük olduğu için dönüş kısımlarında ufak bir gecikme olmaktadır.



Şekil 4.19

Yem Yeme İşlemi

Yılanın sahne üzerindeki yemi yediğini kontrol etme işlemi Oyun sınıfı tarafından yapılacaktır. **Yilan** ve **Sahne** sınıfları arasında direk bir ilişki yoktur. **Oyun** sınıfının, yem ile yılanın ilk hücresinin koordinatlarına ihtiyacı olacaktır. **Sahne** sınıfı içerisinde yemin koordinatlarını getiren bir fonksiyon bulunmasına karşın **Yilan** sınıfı içerisinde ilk hücrenin(baş) koordinatlarını getiren bir fonksiyon bulunmamaktadır. Bu yüzden **Yilan** sınıfında gövdesi aşağıdaki gibi tanımlanmış **basKonumGetir** isimli **public** erişime sahip bir fonksiyon eklenecektir.

```
sf::Vector2f Yilan::basKonumGetir()
{
	return m_hucreler[0].konumGetir();
}
```

Oyun sınıfı içerisinde yem ile yılanın ilk hücresinin çarpışıp çarpışmadığını kontrol eden gövdesi aşağıdaki gibi olan bir fonksiyon eklenecektir.

```
01 bool Oyun::yemYendimi()
02 {
03 	auto basKonum = m_yilan.basKonumGetir();
04 	auto yemKonum = m_sahne.yemKonumGetir();
05
06 	if (basKonum.x == yemKonum.x && basKonum.y == yemKonum.y)
07 		return true;
08 	return false;
09 }
```

Kod 4.32.

yemYendimi fonksiyonu **sahneGuncelle** fonksiyonunda aşağıdaki gibi çağrılacaktır.

```
01 void Oyun::sahneGuncelle()
02 {
03 	if (m_gecenSure.asSeconds() >= m_cerceveSuresi)
04 	{
05 		m_gecenSure -= sf::seconds(m_cerceveSuresi);
06 		if (m_alinanMesafe >= m_hucreBoyutu)
07 		{
08 			if (yemYendimi())
09 			{
10 				m_sahne.yemOlustur();
11 				m_yilan.kuyrugaEkle();
12 			}
13 			m_yilan.yonleriAktar();
14 			m_yilan.yonDegistir(m_siradakiYon);
15 			m_alinanMesafe -= m_hucreBoyutu;
16 		}
17 		m_yilan.hareket(2);
18 		m_alinanMesafe += 2;
19 	}
20 }
```

Kod 4.33.

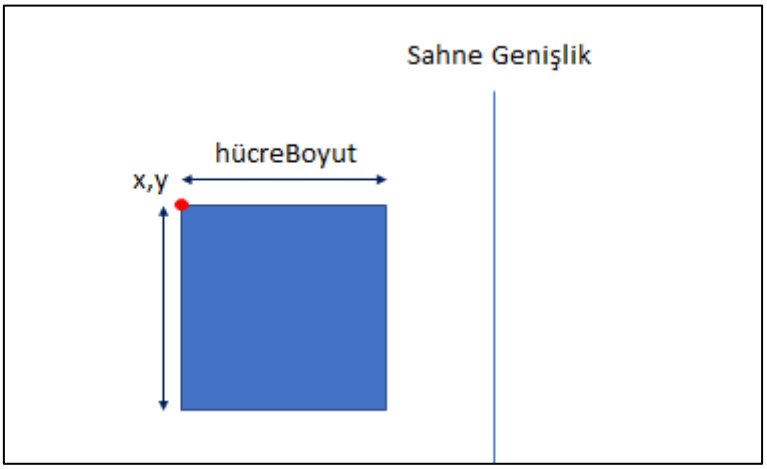
yemYendimi fonksiyonu **yilan** bir hücre boyutu kadar yol aldığında devreye girmektedir. Yön değiştirme mekanizması sadece bu durumda çalıştığından yeminde tam bu noktada kontrol edilmesi mantık olacaktır. Yılanın ilk hücresi yemin tam üstüne geldiğinde **yemYendimi** fonksiyonu **true** dönecektir. 8. Satırda bu fonksiyon bir koşul tarafından yakalanmaktadır. Koşulun gövdesi önce sahne ile iletişime geçecek ve yeni bir yem oluşturulmasını isteyecektir (var olan yemin koordinatları değiştirilecek). Ardından **yilan** ile iletişime geçip kuyruğuna bir eleman eklemesini isteyecektir.

Çarpışma Kontrolü

Sırada yılanın sahne ve kendisiyle yapacağı çarpışmaların kontrolünde. İlk olarak sahne çarpışmalarını kontrol edeceğiz. Bunun için **Oyun** sınıfı içerisinde **sahneIcindemi** fonksiyonu eklenecektir. Fonksiyon içerisinde önce yılanın ilk hücresinin koordinatı, ardından sahnenin boyutları getirilmektedir. Şekil 4.20'de bir yılan hücresi ve sahnenin genişlik sınırı gösterilmektedir. Hücrenin sahnenin dışına taşıp taşmadığını kontrol etmek için koordinat bilgilerinin yanı sıra hücrenin boyutunun da düşülmesi gerekmektedir.

```
01 bool Oyun::sahneIcindemi()
02 {
03         auto konum = m_yilan.basKonumGetir();
04         auto boyut = m_sahne.boyutGetir();
05
06         if (konum.x <= 0||
07                 konum.y<=0||
08                 konum.x+m_hucreBoyutu>=boyut.x||
09                 konum.y+m_hucreBoyutu>=boyut.y)
10         {
11                 return false;
12         }
13         return true;
14 }
```

Kod 4.34.



Şekil 4.20

Sahne boyutlarını getirmek için **Sahne** sınıfı içerisine aşağıda gövdesi verilen fonksiyon eklenecektir.

```
01 sf::Vector2f Sahne::boyutGetir()
02 {
03         sf::Vector2f boyut;
04         boyut.x = m_sutunSayisi * m_hucreBoyutu;
05         boyut.y = m_satirSayisi * m_hucreBoyutu;
06         return boyut;
07 }
```

Kod 4.35.

sahneGuncelle fonksiyonu içerisinde **sahneIcindemi** fonksiyon ile aşağıdaki gibi kontrol yapılmaktadır. **m_kapalimi** değişkeni **true** yapılarak **Oyun** sınıfının çizim döngüsünü sonlandırmasına sebep olmaktadır. Programı bu şekilde çalıştırırsak uygulamanın açıldığı gibi kapandığını göreceğiz. Bunun temel sebebi yılanın ilk hücresinin başlangıçta 0,0 noktasında bulunmasıdır. İlk hücrenin koordinatını başlangıçta bir hücre boyutu sağa alırsak problem çözülecektir (Kod 4.37).

```
01 void Oyun::sahneGuncelle()
02 {
03         if (m_gecenSure.asSeconds() >= m_cerceveSuresi)
04         {
05                 m_gecenSure -= sf::seconds(m_cerceveSuresi);
06                 if (m_alinanMesafe >= m_hucreBoyutu)
07                 {
08                         if (!sahneIcindemi())
09                         {
10                                 m_kapalimi = true;
11                                 return;
12                         }
13                         if (yemYendimi())
14                         {
15                                 m_sahne.yemOlustur();
16                                 m_yilan.kuyrugaEkle();
17                         }
18                         m_yilan.yonleriAktar();
19                         m_yilan.yonDegistir(m_siradakiYon);
20                         m_alinanMesafe -= m_hucreBoyutu;
21                 }
22                 m_yilan.hareket(2);
23                 m_alinanMesafe += 2;
24         }
25 }
```

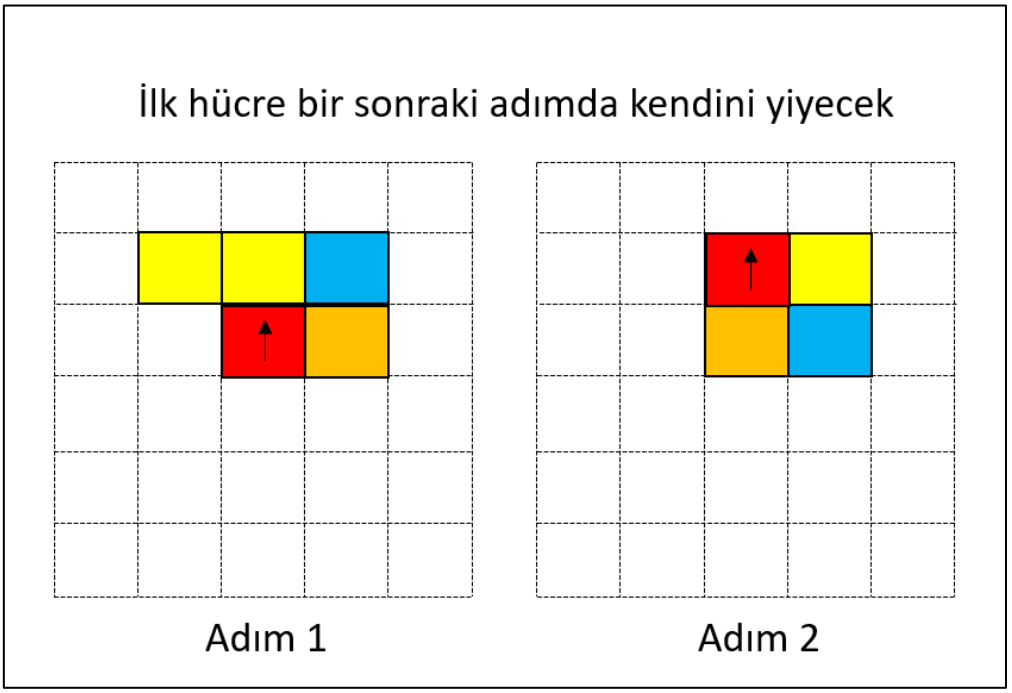
Kod 4.36.

```
01  void Yilan::kuyrugaEkle()
02  {
03          int sonIndis = m_hucreler.size() - 1;
04          if (sonIndis < 0)
05          {
06                  YilanHucre 28ücre;
07                  28ücre.konum(sf::Vector2f(m_hucreBoyutu, m_hucreBoyutu));
08                  m_hucreler.push_back(28ücre);
09                  return;
10          }
11          //Diğer kısımlar aynı
12  }
```

Kod 4.37.

Sınır kontrolü yapıldıktan sonra sırada yılanın kendini yemesinin kontrol edilmesindedir. Yılanın bütün hücreleri Yilan sınıfı içerisinde olduğu için kendini yeme kontrolünün Yilan sınıfı içerisinde yapılması daha mantıklı olacaktır.

kendiniYedimi fonksiyonu yılanın ilk hücresi ile diğer hücrelerinin koordinatlarını karşılaştıracaktır. Karşılaştırma sonucunda koordinatların eşit olduğu bir hücre bulunursa yılan kendisini yemiş demektir.



Şekil 4.21

```
01  bool Yilan::kendiniYedimi()
02  {
03          auto konum = basKonumGetir();
04          int hucreSayisi = m_hucreler.size();
05          for (int i = 1; i < hucreSayisi; i++)
06          {
07                  auto sirKonum = m_hucreler[i].konumGetir();
08                  if (konum.x == sirKonum.x && konum.y == sirKonum.y)
09                          return true;
10          }
11          return false;
12  }
```

Kod 4.38.

Şuan yılanımız kendisini yediğinde programımız sonlanmaktadır. Fakat yılanın yön değiştirmesinde bazı sıkıntılar mevcut. Örneğin yılan sağa giderken sol tuşuna basarsak yılan kendine doğru direk arkaya dönmektedir. Yön atamasında sınırlandırma getirmemiz gerekir. Yılan yatay giderken aşağı veya yukarı yön değiştirebilmesi fakat yatayda değiştirmesine izin vermemeliyiz. Aynı şekilde dikeyde giderken sadece yatay yöne dönmesine izin vermemiz gerekir. Yilan sınıfının yonDegistir fonksiyonu Kod 4.39'daki gibi değiştirilecektir.

```
01 void Yilan::yonDegistir(YON yeniYon)
02 {
03       auto yon = m_hucreler[0].yonGetir();
04       if (yeniYon == YON::ASAGI || yeniYon == YON::YUKARI)
05       {
06             if (yon == YON::SAG || yon == YON::SOL)
07                   m_hucreler[0].yon(yeniYon);
08       }
09       else
10       {
11             if (yon == YON::ASAGI ||yon == YON::YUKARI)
12                   m_hucreler[0].yon(yeniYon);
13       }
14 }
```

Kod 4.39.

Fonksiyon içerisinde yeni yönün yatay veya dikey yön olup olmadığı kontrol edilmektedir. Buna göre yılanın ilk hücresinin yönü de aynı şekilde yatay veya dikey olduğu kontrol ediliyor ve uygun durumda yön değiştiriliyor.