D ile SDL

Faruk Erdem Öncel

D ile SDL

D programlama dilinde SDL ile oyun programlama

Faruk Erdem Öncel

-2.1 Copyright © 2009-2015 Faruk Erdem Öncel

 [1]

1. http://creativecommons.org/licenses/by-nc-sa/3.0/us/
2. http://ddili.org/
3. http://creativecommons.org/licenses/by-nc-sa/3.0/us/
4. http://digitalmars.com/
5. http://ddili.org/copyright.html

İçindekiler

-1 Hoşgeldiniz

D ile 2D oyun programlama dersine hoşgeldiniz. Bu derste D dili kullanarak 2D oyun programlamaya bir başlangıç yapacağız.

Bir oyunun geliştirme aşamaları şu bölümlerden oluşuyor

- Oyun tasarımı ve senaryo
- Programlama
- Görsel Sanatlar
  (Çizim, resim ve diğer sanatsal içerik)

Biz bunlardan programlama ile ilgileneceğiz. Bunu yaparken programlama dili olarak D kullanacağız.

Kullanacağımız grafik kütüphanesi ise SDL (Simple Direct Media Layer)
http://www.libsdl.org[1]

Konuları takip edebilmek için sadece D programlama dilini bilmeniz yeterli. Her ne kadar SDL grafik kütüphanesi kullanıyor olsak da asıl amacım sadece 2D oyun programlama, oyun değişkenleri ile ilgili temel kavramları anlatmak olacak.

SDL ile ilgili konuları sadece gerektiğinde ve *uzmanlar için* bölümünde ek bilgi olarak anlatmayı düşünüyorum.

-1.1 Haydi başlayalım

Başlamak için şu basit adımları izlemeniz yeterli:

1. D derleyicisini kurun
2. SDL için geliştirme kütüphanelerini kurun
3. Örnek projeyi bilgisayarınıza indirin

gerekli araçları kuralım

1- D derleyicisi:
Ubuntu altında Digital Mars derleyicisinin son sürümünü
http://dlang.org/download.html[2]
adresinden indirerek otomatik olarak yazılım merkezi ile kurabilirsiniz.
Windows altında Digital Mars derleyicisinin son sürümünü
http://dlang.org/download.html[3]
adresinden otomatik olarak kurabilirsiniz.

2- SDL geliştirme kütüphaneleri:
SDL geliştirme kütüphanelerini kurmak için Ubuntu altında konsolda

```
$ sudo apt-get install libsdl1.2-dev libsdl-image1.2-dev
```

komutunu vermeniz yeterli.

Windows altında, SDL geliştirme kütüphaneleri örnek proje ile beraber geliyor. Bu yüzden SDL ile ilgili kütüphaneleri indirmenize gerek yok.

Ancak kolaylık olması açısından bin dizininde bulunan `*.dll` uzantılı dosyaları `C:\WINDOWS\system32` dizinine kopyalamak isteyebilirsiniz.

3- Örnek projeyi indirelim:
Örnek projeyi indirmek için tarayıcınızla

1. http://www.libsdl.org
2. http://dlang.org/download.html
3. http://dlang.org/download.html

https://github.com/erdemoncel/oyun[1]
adresini açıp sağ üst tarafta bulunan Downloads düğmesine tıklayıp Download .tar.gz ya da Download .zip seçerek projeyi indirmeniz yeterli

Ya da eğer *github* kullanımını biliyorsanız basitçe:

```
$ git clone git@github.com:erdemoncel/oyun.git
```

komutuyla projenin bir kopyasını bilgisayarınıza indirebilirsiniz.

-1.2 Oyun döngüsü

Bir oyunun temel bileşeni bir oyun döngüsüdür. İster çok gelişmiş ya da basit bir oyun olsun her oyun bir oyun döngüsü kullanır. Şimdi bir oyun döngüsünün yapısını inceleyelim.

İlklendir :

Oyunda kullandığımız oyun değişkenlerinin ilk değerlerini burada veriyoruz.

İçeriği Yükle :

Oyunumuzun ihtiyaç duyduğu 2D hareketli grafikler texture, oyuncu modelleri, ses efektleri ve müzik gibi içeriği bu bölümde yüklüyoruz,

Güncelle :

Burada oyuncunun klavye, oyun çubuğu ya da başka bir donanımla girdiği girdileri kontrol ediyor ve oyunun her karesinde oyun değişkenlerini değiştiriyoruz.

Çiz :

Burada oyunumuza grafik kartına ne gönderileceğini ve ekrana nasıl çizileceğini söylüyoruz

-1.3 Bir oyuncu oluşturmak

Ekranda bir görüntü oluşabilmesi için çeşitli bilgileri saklamalıyız. Bir oyuncunun konumu buna iyi bir örnek olabilir.

Şimdi kağıt üzerinde bir oyuncu oluşturmak için neler yapmamız gerektiğini düşünelim. Bunları kağıda yazın

İlklendir bölümüne oyuncumuzun ekran üzerinde başlangıç konumunu belirleyen bir işlev ekleyelim.

İlklendir

```
        oyuncuKonumunuBelirle()
```

Şimdi teknik olarak oyuncumuz mevcut. Oyunun `ilklendir()` metodunda sağladığımız bilgilerle artık oyun karakterimizi istediğimiz yere hareket ettirebiliriz.

Ama ekranda bir şey göremezsek bu iyi bir oyun sayılmaz

Ekrana bir şeyler çizebilmek için ilkönce çizeceğimiz animasyonu ya da grafiği yüklemeliyiz.

---

1. https://github.com/erdemoncel/oyun

İçerik Yükle

```
        içerik.yükle(dosya)
```

Daha sonra oyunumuzu ekran kartının belleğinde çiziyoruz.

Çiz

```
         oyuncuyu.çiz()
```

Oldukça basit. Peki oyunun içinde sürekli değişiklikler olmasını istersek ne yapmamız lazım.

Burada işte `güncelle()` metodu karşımıza çıkıyor. Güncelle metodunu kullanarak ilk değerlerini verdiğimiz oyun değişkenlerini istediğimiz gibi değiştirebiliyoruz. Güncelle metoduna bazı kodlar ekleyerek oyun içinde tutulan istediğimiz bilgiyi değiştirebiliriz.

Güncelle

```
         if (sağTuşBasılıMı)
            konumuGüncelle & animasyonuGüncelle
```

Örneğin klavyenin sağ tuşuna basınca oyun karakterimizin sağa doğru hareket etmesini istiyorsak `güncelle()` metodunda sağ tuşun basılı olup olmadığını denetliyoruz ve animasyonu güncelliyoruz.

Burada `animasyonuGüncelle()` kısmına sağa doğru yürüyen bir oyuncu animasyonu koyabiliriz.

-1.4 oyuncu.d dosyasının içinde

Artık oyunumuzu kodlamaya başlayabiliriz.

İlkönce daha önce indirmiş olduğunuz proje dosyalarını bir sıkıştırma programı ile açıp test klasörünün içine gelin. Burada herhangi bir editörle `oyuncu.d` isminde yeni bir dosya oluşturun. Ve içine şunları girin.

```
import sdl, vector2, cizici;

class Oyuncu
{
    // ilklendir
    this(Grafik2D grafik, Vector2 konum)
    {
    }

    void güncelle()
    {
    }

    void çiz(Çizici çizici)
    {

    }
}
```

Dikkat ederseniz biraz önce oyun döngüsünde bahsettiğimiz işlevlerin bazılarını **Oyuncu** sınıfımız için de yazmış olduk.

D derslerinden hatırlayacağınız gibi `this()` sınıfın kurucu işlevi ve sınıf üyelerine ilk değer ataması burada gerçekleşiyor. D'nin böyle bir özelliği olduğu için ekstradan bir `ilklendir()` işlevi yazmamıza gerek yok. **Oyuncu** sınıfının değişkenlerine ilk değer atamasını burada yapabiliriz.

Şimdi düşünelim. Bir oyun karakterinin ne gibi özellikleri olmalı?

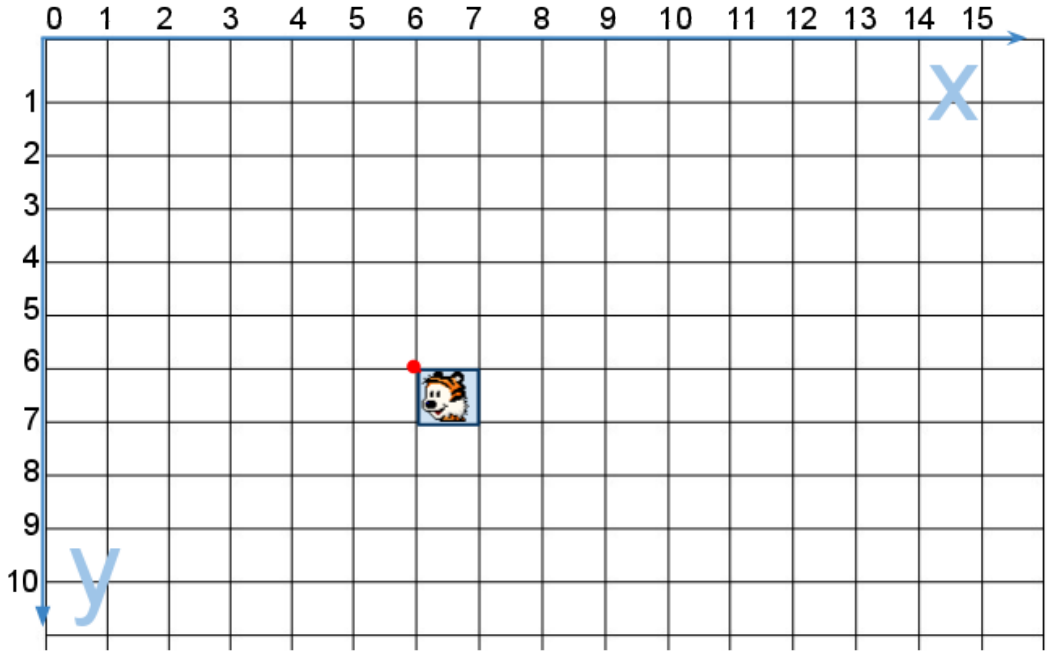
Herşeyden önce ekrana çizebileceğimiz bir 2D grafiğe ya da bir animasyona sahip olmalı. Ayrıca konumunu belirten koordinatlara sahip olmalı.

İşte **Oyuncu** sınıfımız bu yüzden parametre olarak bir 2 boyutlu grafik ve konum bilgisi alıyor. Burada şimdilik **Vector2** nin ekran üzerinde bir oyun nesnesinin x ve y koordinatlarını tutan basit bir yapı olduğunu bilmeniz yeterli.

Örneğin:

```
        auto oyuncuKonum = Vector2(6, 6);
```

diyerek aşağıdaki *Calvin and Hobbes* çizgi romanının sevimli karakteri *Hobbes*'un konumunu bir **Vector2** yapısında tutabiliriz.



Burada hemen farkedebileceğiniz bir şey SDL'in y ekseni alıştığımız koordinat sisteminin tersine aşağı doğru bakıyor.

Ayrıca koordinat merkezi olarak oyuncu grafiğinin sol üst köşesini alıyoruz. (resimdeki kırmızı nokta ile işaretlenmiş kısım)

`class Oyuncu`'nun altındaki ilk { den sonra bunları girin.

```
    // her oyuncunun bir 2D grafiği olmalı
    Grafik2D oyuncuGrafik;

    // oyuncunun ekranın sol üst köşesine göre göreceli konumu
    Vector2 konum;

    // oyuncunun durum
    bool aktif;

    // oyuncunun sağlık puanı
    int sağlık;

    // oyuncunun genişliği ve yüksekliği
    int genişlik;
    int yükseklik;
```

Vector2 yapısıyla bir nesnenin konumunun x ve y koordinatlarını tutabileceğimizi söylemiştik. **Grafik2D** ise grafik bilgisini tutan ve ekrana çizilebilen özel bir veri türü.

Oyuncumuzu çizdireceğimiz zaman oyuncuya ait **Grafik2D** iki boyutlu grafiğini **Vector2** ile belirtilen konuma çizdireceğiz.

Şimdi yapmamız gereken oyuncunun ilk konumunu belirtmek ve oyuncuya ait grafiğe bir ilk değer vermek.

Oyuncu sınıfının kurucu işlevini aşağıdaki gibi değiştirin.

```
    this(Grafik2D grafik, Vector2 konum)
    {
        oyuncuGrafik = grafik;

        // oyuncunun başlangıç konumu belirle
        this.konum = konum;

        // oyuncuyu aktif yapıyoruz
        aktif = true;

        // oyuncunun sağlık puanını belirle
        sağlık = 100;

        // her grafik nesnesinin bir w (genişlik) ve h (yükseklik) değeri
        // olduğu için oyuncumuzun genişlik ve yükseklik değerlerine bu
        // değerleri atayabiliriz
        genişlik = grafik.w;
        yükseklik = grafik.h;
    }
```

Oyuncumuzun ihtiyaç duyduğu ilk değerleri verdikten sonra artık oyuncuyu çizdirebiliriz. Bunu da oyuncunun çiz yöntemine bir **Çizici** nesnesi geçerek yapıyoruz.

Çizici nesnesinin yaptığı kendine geçilen bir iki boyutlu grafiği ekranın belirli bir konumuna çizdirmek. **Çizici** sınıfının ayrıntıların merak ediyorsanız src klasöründe bulan `cizici.d` modülünü inceleyebilirsiniz.

Oyuncu sınıfının `çiz` işlevini aşağıdaki gibi değiştirin.

```
    void çiz(Çizici çizici)
    {
        çizici.çiz(oyuncuGrafik, konum);
    }
```

Oyuncu sınıfıyla şimdilik işimiz bitti.

-1.5 oyun.d dosyasının içinde

test klasöründeki `oyun.d` dosyasını açın.

Kod içinde **Oyun** sınıfını bulduktan sonra `class Oyun : TemelOyun` un hemen altına { dan sonra oyuncuyu ekleyin.

```
    Oyuncu oyuncu;
```

Oyuncuyu ekrana çizebilmek için önce oyuncuya ait grafiği yüklemeli ve oyuncunun ilk konumunu belirlemeliyiz.

Sabitdisk üzerinde bulunan grafikleri okuyup oyuna yükleyeceğimiz için bunları `içerikYükle()` metodunun hemen altında yapabiliriz.

`super.içerikYükle()` nin altına bu kodu ekleyin.

```
        // oyuncuya ait içeriği yükle
        auto oyuncuKonum = Vector2(288, 203);
        oyuncu = new Oyuncu(içerik.yükle("penguen.bmp"), oyuncuKonum);
```

Burada `içerik.yükle` kısmı dikkatinizi çekmiş olabilir. Oyunumuzun içerik isimli bir yapısı var. Bir grafiği yüklemek için bu yapıya yüklemek istediğiniz 2D grafiğin ismini vermeniz yeterli.

`icerik.yukle()` işlevi bir **Grafik2D** döndürüyor. Ayrıca **Oyuncu** sınıfının kurucu işlevi de parametre olarak bir 2D grafik ve oyuncunun konumunu belirten bir vektör alıyordu.

Oyuncuyu ekrana çizmeye hazırız. Oyun sınıfının `ciz` işlevine bu kodu ekleyin.

```
        /// Nesneleri göster
        // oyuncuyu çiz
        oyuncu.çiz(çizici);
```

İşte bu kadar. Artık çizme bölümü de tamamlanmış oldu.

-1.6 Oyunu çalıştırmak

Programın kaynak kodunu derlemek için şu komutları girin:

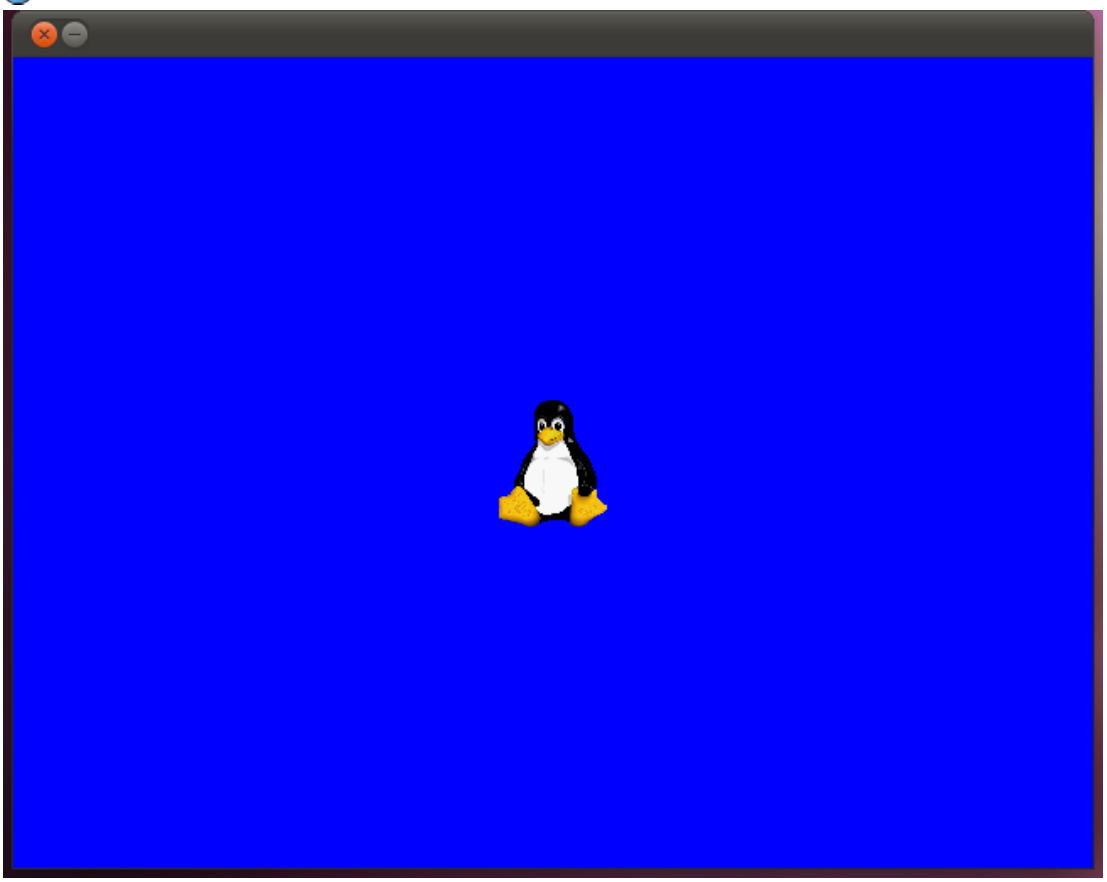
Ubuntu altında konsoldan:

```
$ make ornek
$ cd bin
$ ./ornek
```

Windows altında *Başlat->Çalıştır->* yolunu izleyerek buraya `cmd` yazın. Açılan konsolda:

```
make ornek -f win32.mak
cd bin
ornek.exe
```

Eğer her şey yolunda gittiyse ekranınınızda sevimli bir penguen görmeniz lazım



-1.7 Dersin kaynak kodu

Programı derlerken herhangi bir sorun çıktıysa merak etmeyin. Dersin kaynak kodunu buradan indirebilirsiniz.

-1.8 Uzman ipucu

Yazının başında da belirttiğim gibi SDL ile ilgili konuları bu bölümde anlatacağım. Bu yüzden bu bölümü okuyup okumamak sizin zevkinize kalmış

Amacımız ekranın arkaplan rengini değiştirmek. **TemelOyun** sınıfı içinde `//Ekranı temizle` yazan kısmın hemen altına gelin.

```
        SDL_FillRect(ekran, &ekran.clip_rect,
                     SDL_MapRGB(ekran.format, 0x00, 0x00, 0xFF));
```

Burada `SDL_MapRGB` işlevinin aldığı ilk parametre *piksellerin biçimi*, ikincisi de rengi oluşturan r g b değerleri. Yani *kırmızı yeşil mavi* değerleri. İşlevin kendisi de bir `uint` tamsayı döndürüyor.Yani bu kırmızı yeşil mavi değerlerinin birleşimi olan renk değerini SDL'de bir `uint` işaretsiz tamsayı ile ifade edebiliyoruz.

Eğer Gimp, Photoshop gibi çizim programları kullandıysanız renklerin karşılıklarının 0 ile 255 arasında değişen *RGB* değerler ile ifade edildiğini hatırlayacaksınız. Örneğin RGB(255, 0, 0) kırmızı gibi.

Biz de bu renk değeri kullanarak ekranı temizle işlevini, ekranı maviye boyayacak şekilde değiştireceğiz. Burada sizin de farketmiş olabileceğiniz gibi bu değerler onaltılık sayı sisteminde 0x şeklinde yazılmış. Bu yüzden ekranı maviye boyamak için ikinci renk değerini 0xFF olarak değiştirin.

Programı tekrar derlediğinizde artık ekranın maviye boyandığını göreceksiniz.

0 Oyunlarda hareket

Bu derste oyunlarda hareket konusu inceleyeceğiz. Daha önceki dersimizde vektörler için bir oyun nesnesinin konum bilgilerini tutan basit bir yapı olduğunu bilmeniz yeterli demiştik. Bu derste vektörler konusunda temel bazı bilgiler öğreneceğiz.

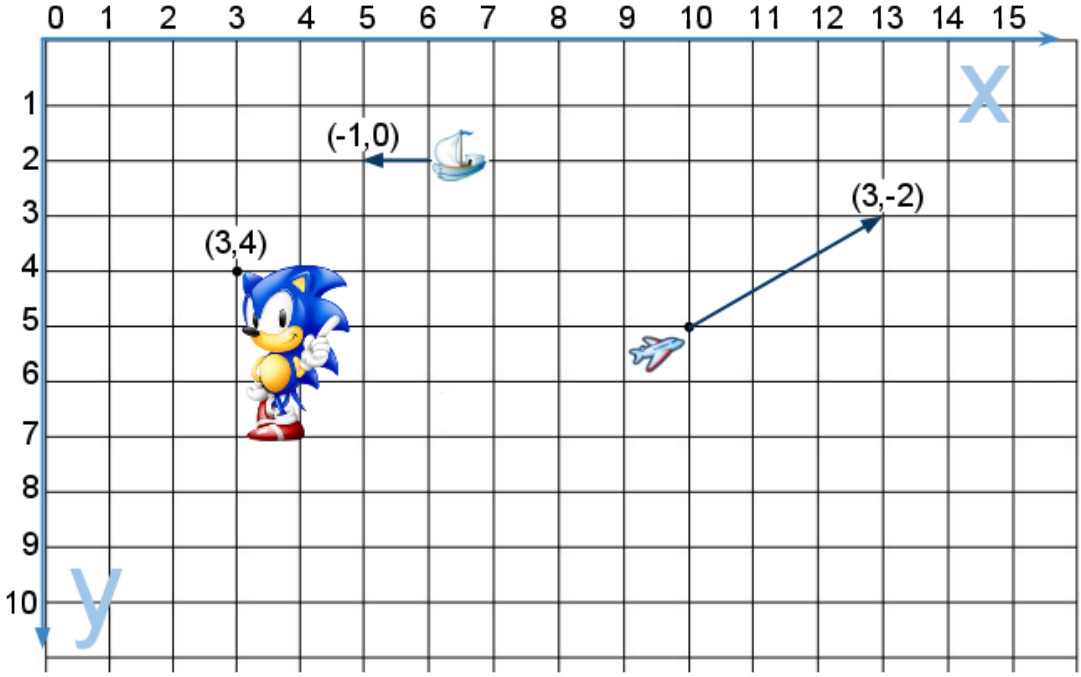
0.1 Lineer cebir nedir

Lineer cebir vektörleri inceleyen matematik koludur. Eğer oyunumuzda bir yarış arabasının hızı, kameranın yönü gibi bilgilere ihtiyacımız varsa vektörleri kullanmak durumundayız.

Lineer cebiri daha iyi anladığınız takdirde vektörlerin davranışları üzerinde daha fazla kontrol sahibi olabilirsiniz.

0.2 Vektör nedir?

Oyunlarda vektörler konum, yön ve hız bilgisi tutarlar. İşte size iki boyutlu örnekler



Konum vektörü Sonic oyun karakterinin merkezden 3 metre doğuda ve 4 metre güneyde durduğunu gösteriyor. Hız vektörü dakikada uçağın 2 kilometre yukarı ve 3 kilometre sağa hareket ettiğini gösteriyor. Yön vektörü geminin sola doğru işaret ettiğini gösteriyor

Sizin de görebileceğiniz üzere vektörün kendisi aslında bir dizi sayıdan oluşuyor, kullanıldığı yere göre bir anlam kazanıyor.Örneğin vektör (-1, 0) şekildeki gibi bir geminin yönü olabileceği gibi, bulunduğunuz konuma 1 kilometre uzaktaki bir binanın konumu ya da saatte 1km hızla hareket eden bir salyangozun hızı da olabilir.

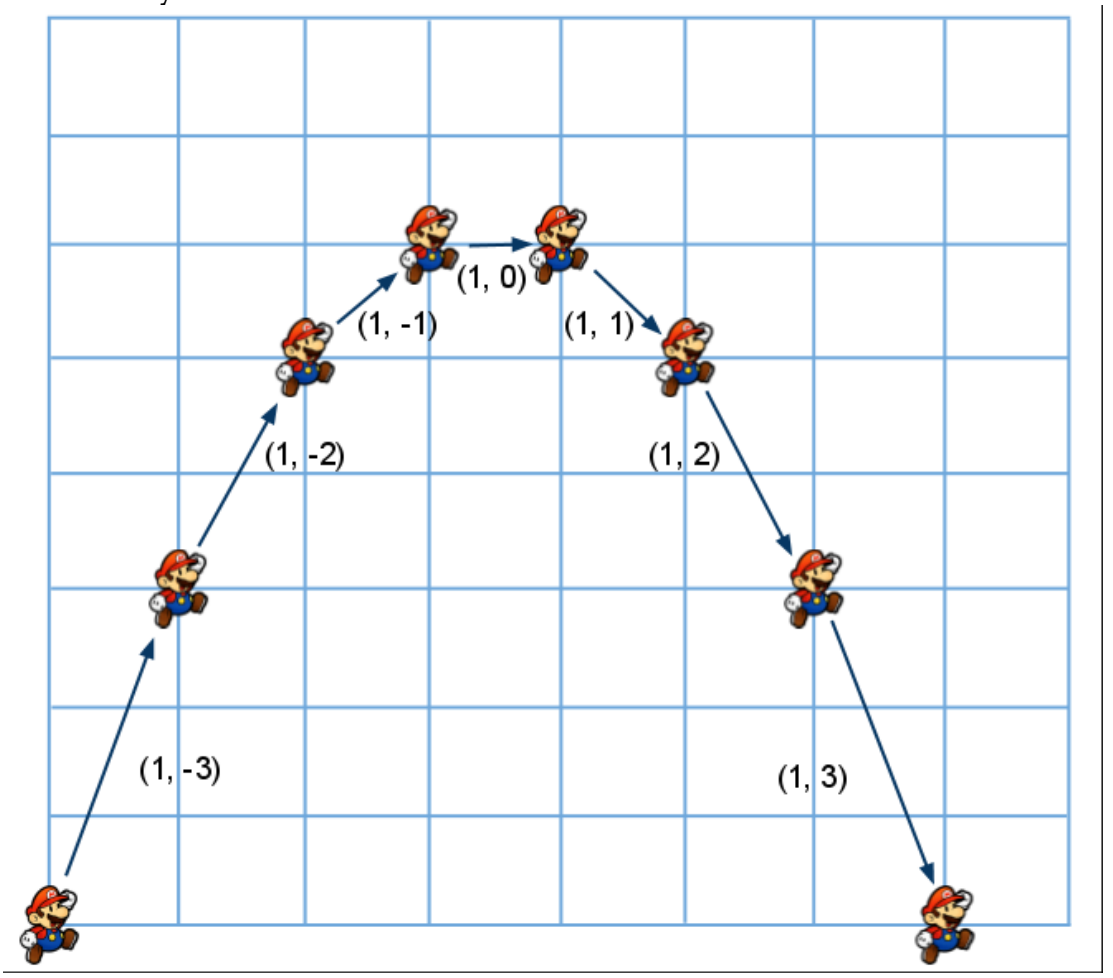
Bu yüzden vektörleri kullanırken birimlerini de belirlemek önemli. Şimdi temel olarak vektörlerin ne olduğunu öğrendiğimize göre onları nasıl kullanacağımızı öğrenmenin zamanı geldi

0.3 Vektörlerde toplama

Vektörleri toplamak için vektörün her bileşenini ayrı ayrı toplamak gerekiyor. Örneğin:

```
(0,1,4) + (3,-2,5) = (0+3, 1-2, 4+5) = (3,-1,9)
```

Peki vektörleri neden toplamak isteriz? Vektörleri toplamanın nedenlerinden bir tanesi fizik entegrasyonunu yapabilmek içindir. Oyun içinde her fiziksel nesne konum, hız, ivme gibi bilgileri tutabilmek için vektörleri kullanacaktır. Oyunun her karesinde (genellikle saniyenin 1/60'ı kadar bir sürede) hızı konuma ve ivmeyi hıza eklemeliyiz.



Burada gene SDL'in koordinat sistemine göre düşünüyoruz. Yani y ekseni aşağı doğru bakıyor. Yukarıya hareket etmek demek y değerinin azalması demek

Aslında Maryo ekranın altında durduğuna göre, konumu (0, 480) gibi bir değer olmalı (Ekran çözünürlüğümüzün 640x480 olduğunu varsayarsak) Ama biz hesaplamalarda kolaylık olsun diye bulunduğu konumu (0, 0) noktası kabul edeceğiz

İlk karede hızını (1, -3) konumuna (0, 0) ekleyerek yeni konumunu (1, -3) buluyoruz. Daha sonra ivmesini (0, 1) hıza (1, -3) ekleyerek yeni hızını (1, -2) buluyoruz

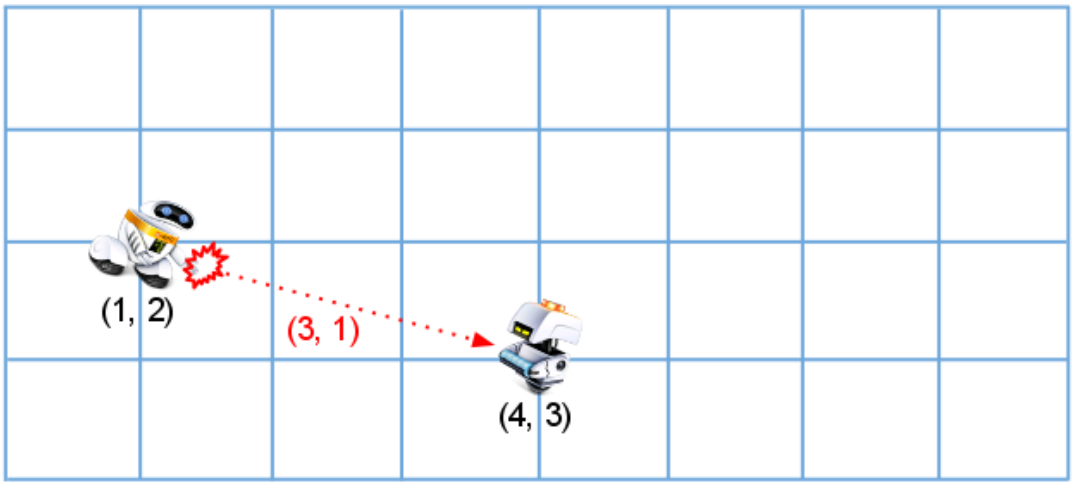
İkinci karede de aynısını yapıyoruz. Hızını (1, -2) konuma (1, -3) ekleyerek yeni konumu (2, -5) buluyoruz. Daha sonra ivmeyi (0, 1) hıza (1, -2) ekleyerek yeni hızı (1, -1) olarak buluyoruz.

Genellikle oyunlarda oyuncu karakterin ivmesini klavye veya oyun çubuğu kullanarak kontrol eder, oyun da yeni hız ve konum bilgilerini vektörleri toplayarak hesaplar

0.4 Vektörlerde çıkartma

Vektörlerde çıkartmayı da toplamaya benzer şekilde yapıyoruz. Vektörlerin çıkartarak bir konumdan bir diğer konumu işaret eden bir vektör buluyoruz. Örneğin oyuncumuz (1, 2) konumunda elinde bir lazer silahı ile duruyor ve düşman robotu (4, 3) noktasında olsun. Lazerin oyuncuyu vurabilmek için katetmesi gereken mesafeyi gösteren vektörü oyuncunun konumunu robotun konumundan çıkararak bulabilirsiniz.

```
(4,3)-(1,2) = (4-1, 3-2) = (3,1)
```



0.5 Skaler vektör çarpımı

Vektörler söz konusu olduğunda skaler derken vektörü oluşturan her sayıya skaler diyoruz. Örneğin (3,4) bir vektörken, 5 skaler. Oyunlarda bir vektörü bir skalerle çarpabilmek oldukça işimize yarayacak. Örneğin oyuncu hareket ederken oyuna daha gerçeklik katmak için oyuncunun hızına karşı gösterilen hava direncini hesaplayabiliriz. Bunu da oyuncunun hızını her karede 0.9 ile çarparak buluruz. Bunu yapmak için vektörün her bileşenini skaler ile çarpıyoruz. Eğer oyuncunun hızı (10,20) ise yeni hızı şu şekilde bulabiliriz:

```
0.9*(10,20) = (0.9*10, 0.9*20) = (9,18)
```

0.6 Kodlamaya başlayalım

Bu dersimize önceki derste kodladığımız kod üzerinde çok az değişiklik yaparak devam edeceğiz. Eğer daha önceki dersi kodladıysanız o dersin üzerinde değişiklikler yaparak devam edebilirsiniz. Eğer kodlamadıysanız buradan sıkıştırılmış dosyayı indirerek projenin ana dizininde açmanız yeterli. Bir önceki dersin kaynak kodlarını içeren test isminde bir dizin oluşacak.

0.7 oyun.d dosyasının içinde

`class TemelOyun`'un üzerinde bunları girin

```
class Top : Oyuncu
{
    this(Grafik2D grafik, Vector2 konum)
    {
        // Oyuncu sınıfının kurucu işlevini çağırıyoruz
        super(grafik, konum);
    }
}
```

Burada yaptığımız hiç bir şey yok aslında. **Top** sınıfını **Oyuncu** sınıfından türetiyoruz. **Top** sınıfının kurucu işlevinde de devraldığımız sınıfın kurucu işlevini çağırıyoruz.

Bilgilerinizi tazelemek için türeme ile ilgili dershanedeki bu dersi[1] tekrar gözden geçirmek isteyebilirsiniz

class Oyun: TemelOyun'un altındaki ilk [den sonra Oyuncu oyuncu kısmını aşağıdaki gibi değiştirin

```
    Top top;
    Vector2 topHızı;
```

Oyun sınıfının içerikYükle() işlevini aşağıdaki gibi değiştirin

```
        super.içerikYükle();

        // oyuncuya ait içeriği yükle
        auto topkonum = Vector2(0, 0);
        top = new Top(içerik.yükle("top.png", true), topkonum);
        topHızı = Vector2(2, 2);
```

Burada da ilk derste öğrendiklerimiz dışında yaptığımız tek şey içerik.yükle() işlevine geçtiğimiz ikinci parametre true saydam bir grafik yüklemek istediğimizi belirtiyor

Oyun sınıfının çiz işlevini de artık olmayan oyuncu nesnemiz yerine topu çizdirecek şekilde değiştirelim

```
        /// Nesneleri göster
        // topu çiz
        top.çiz(çizici);
```

Bu noktaya kadar yaptığımız basitçe penguen resmi yerine saydam bir top grafiği yüklemekti. Herhangi bir hata yapıp yapmadığınızı test etmek için Ubuntu altında şu komutları vermeniz yeterli

```
$ make ornek
$ cd bin
$ ./ornek
```

0.8 Topu hareket ettirmek

Oyun sınıfının güncelle() işlevinin en sonuna `// Oyun mantığı` yazan kısmın hemen altına topun güncellenme işlevini ekleyelim.

```
        // Oyun mantığı
        topuGüncelle();
```

Bu işlevin hemen altındaki] parantezinden sonra topun güncellenme işlevini ekleyelim

```
    private void topuGüncelle()
    {
        top.konum += topHızı;
```

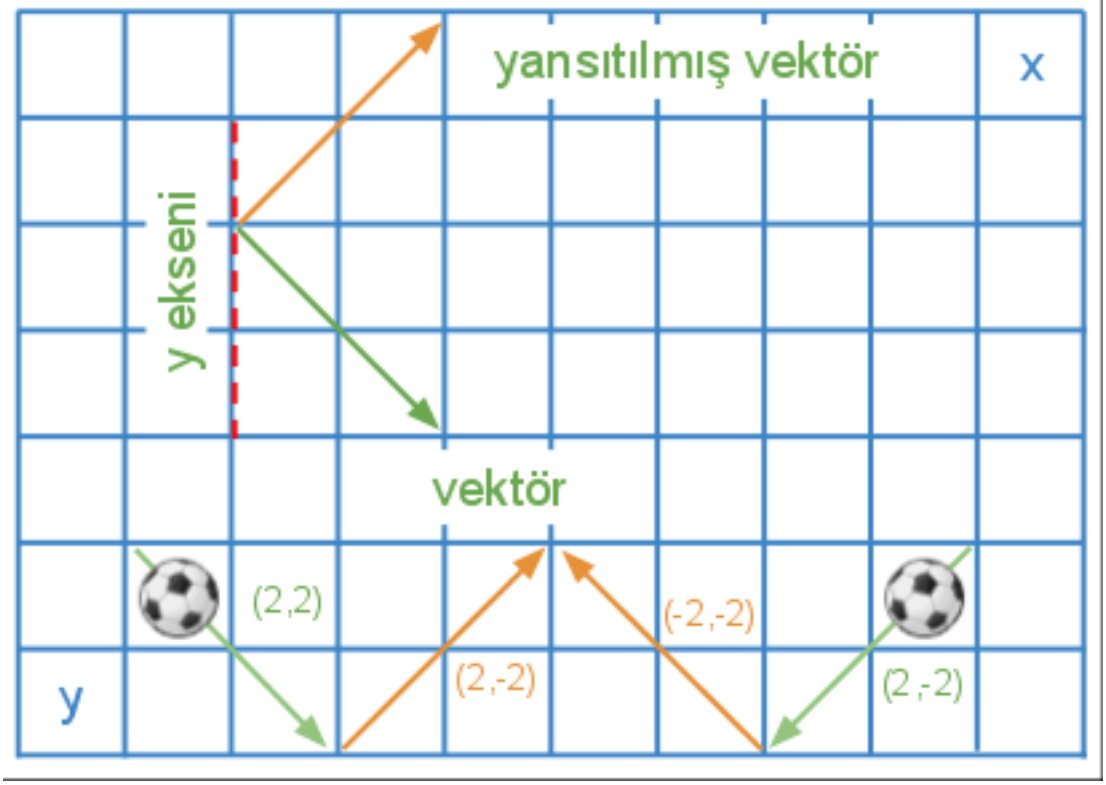
1. http://ddili.org/ders/d/tureme.html

```
        // alta mı geldik
        if (top.konum.y + top.yükseklik >= ekranyükseklik)
            // alta geldiysek hızı y ekseninde yansıtmamız yeterli
            topHızı.yansıt(Vector2(0, 1));

        // sağa mı geldik
        if (top.konum.x + top.genişlik >= ekrangenişlik)
            // sağa geldiysek hızı x ekseninde yansıtmamız yeterli
            topHızı.yansıt(Vector2(1, 0));

        // üste mi geldik
        if (top.konum.y <= 0)
            // üste geldiysek hızı y ekseninde yansıtıyoruz
            topHızı.yansıt(Vector2(0, 1));

        // sola mı geldik
        if (top.konum.x <= 0)
            // sola geldiysek hızı x ekseninde yansıtıyoruz
            topHızı.yansıt(Vector2(1, 0));
    }
```

Daha önce oyunun her karesinde hızı konuma ekliyoruz demiştik. İşte topun güncelle metodunda bunu yapıyoruz. Topun ilk konumu (0,0) noktasıydı. Oyun başladığında top sağa ve aşağı doğru hareket edecek

Alta gelip gelmediğimizi topun y konumunu, topun yüksekliğine ekleyerek hesaplayabiliriz. Eğer bu değer ekranyüksekliğine büyük ya da eşitse topu y ekseninde yansıtıyoruz. Top alt tarafa hem sağdan hem de soldan yaklaşabilir

Şimdi işin en püf noktasına geldik. Bir vektörü yansıtmak ne demek



Şekilde yeşil renkli vektörü y ekseni üzerinde yansıttığımız zaman elde ettiğimiz vektör turuncu renkte olan vektör. yansıt işlevine geçtiğimiz Vector2(0, 1) parametresi vektörü y ekseninde yansıtmak istediğimizi belirtiyor.

Bu vektörün uzunluğunun bir birim olduğunu kolayca görebiliriz. Uzunluğu bir birim olan vektörlere birim vektör deniliyor. Dikkat ederseniz topun sağdan ya da soldan gelse de y ekseninde yansıttığımızda istediğimiz hareketi elde edebiliyoruz

0.9 Oyunu çalıştırmak

Programın kaynak kodunu derlemek için şu komutları girin:

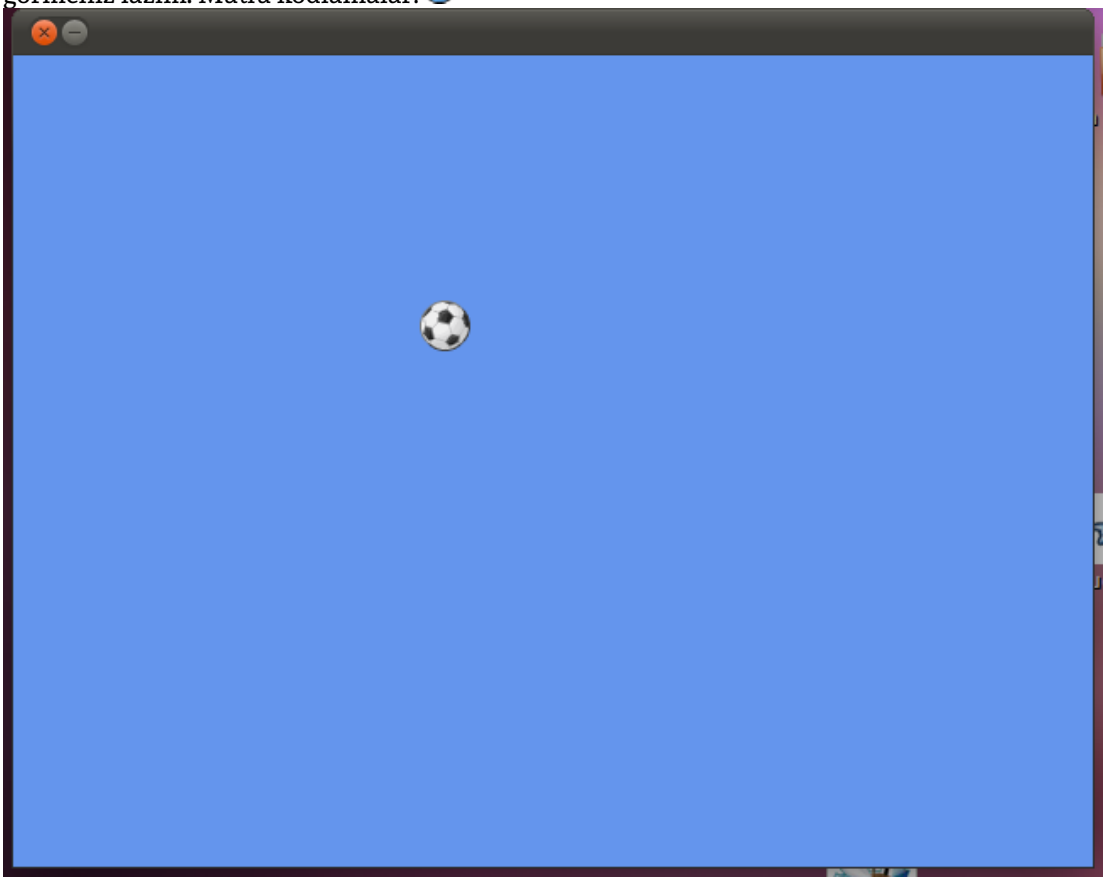
Ubuntu altında konsoldan:

```
$ make ornek
$ cd bin
$ ./ornek
```

Windows altında *Başlat->Çalıştır->* yolunu izleyerek buraya `cmd` yazın. Açılan konsolda:

```
make ornek -f win32.mak
cd bin
ornek.exe
```

Eğer her şey yolunda gittiyse ekranınınızda sağa sola hareket eden bir top görmeniz lazım. Mutlu kodlamalar!



Dizin