Pong-peli, vaihe 7

Tässä vaiheessa lisäämme peliin pistelaskun. Pong-pelissä pelaaja saa pisteen kun pallo ohittaa toisen pelaajan mailan.

1. Aliohjelma laskureita varten

Pisteiden laskemiseksi tarvitsemme jonkinlaisen laskurin, jossa pitää yllä pelaajan pisteitä. Lisäksi laskuri täytyisi nollata pelin alussa. Tässä olisi yksi kokonaisuus pelin alustamisessa. Tehdäänpä siitä aliohjelma (toteutetaan se hiukan myöhemmin):

```
void LisaaLaskurit()
{
   // ...
}
```

Kutsutaan tuota aliohjelmaa `Begin`-aliohjelmasta:

```
protected override void Begin()
{
    LuoKentta();
    AsetaOhjaimet();
    LisaaLaskurit();
    AloitaPeli();
}
```

2. Laskurin luominen

Laskureita tulee peliin kaksi, yksi kummallekin pelaajalle. Koska molemmat laskurit luodaan arvatenkin lähes samalla tavalla, kannattaa laskurin luomisesta tehdä aliohjelma:

```
IntMeter LuoPisteLaskuri()
{
    IntMeter laskuri = new IntMeter( 0 );
    laskuri.MaxValue = 10;
    return laskuri;
}
```

Aliohjelman paluuarvon tyyppi on `IntMeter`, mikä tarkoittaa laskuria joka laskee kokonaisluvuilla (kokonaisluvun tyyppi on `int`). Laskurin luomisessa parametrina on laskurin oletusarvo, mikä pistelaskun ollessa kyseessä on luonnollisesti nolla. Laskurille voi asettaa maksimiarvon, jonka jälkeen laskuri lopettaa pisteiden laskemisen, asetetaan maksimiarvo vaikkapa kymmeneen.

Pisteiden laskemiseen riittäisi toki pelkkä kokonaisluku, mutta `IntMeter`-tyyppiä käyttämällä päästään helpommalla kun pisteitä halutaan esittää ruudulla, kuten kohta nähdään.

3. Pisteiden esittäminen

Pelkkä `IntMeter`-olio ei vielä näytä pisteitä, vaan ainoastaan säilyttää lukuarvon muistissa. Pisteiden esittämiseksi ruudulla tarvitaan vielä tekstikenttiä. Sellaisia ovat `Label`-tyyppiset oliot. Tehdäänpä sellainen laskurin ohessa. Koska pelaajien laskurit tulevat eri kohtaan ruutua, viedään ruutukoordinaatit parametrina aliohjelmalle. Näillä muutoksilla aliohjelma näyttää tältä:

```
IntMeter LuoPisteLaskuri( double x, double y )
{
    IntMeter laskuri = new IntMeter( 0 );
    laskuri.MaxValue = 10;
    Label naytto = new Label();
    naytto.BindTo( laskuri );
    naytto.X = x;
    naytto.Y = y;
    naytto.TextColor = Color.White;
    Add( naytto );
    return laskuri;
}
```

`Label` ei itse sisällä yhtään lukua, mutta se osaa näyttää laskurin arvon. Tätä varten sille pitää kertoa, minkä laskurin lukua se näyttää, kutsulla `naytto.BindTo( laskuri )`. Kun tekstikenttä tietää laskurin, näytössä näkyy aina laskurin oikea arvo silloinkin kun laskurin arvoa jossakin kohtaa muutetaan.

Koska tekstikentän teksti on oletuksena musta, se ei erottuisi kovin hyvin mustasta taustasta. Niinpä asetetaankin näytön väri (`naytto.ValueColor`) valkoiseksi (`Color.White`).

Kun pistenäyttö on luotu, täytyy se vielä lopuksi lisätä ruudulle. Tämä tapahtuu yksinkertaisesti aliohjelman `Add` (suom. lisää) kutsulla.

4. Laskureiden lisääminen peliin

Koska laskureihin täytyy päästä käsiksi kohta, kun pisteitä lasketaan, tehdään laskureista attribuutteja (attribuutti näkyy kaikille saman luokan aliohjelmille) muiden attribuuttien seuraksi. Miksei näyttöjä tarvitse lisätä attribuuteiksi?

```
public class Peli : PhysicsGame
{
    Vector nopeusYlos = new Vector( 0, 200 );
    Vector nopeusAlas = new Vector( 0, -200 );

    PhysicsObject pallo;
    PhysicsObject maila1;
    PhysicsObject maila2;
```

```
    IntMeter pelaajan1Pisteet;
    IntMeter pelaajan2Pisteet;


    protected override void Begin()
    {
        LuoKentta();
        AsetaOhjaimet();
        LisaaLaskurit();
        AloitaPeli();
    }

    // ...
```

Kun meillä on aliohjelma laskurin luomiseksi, voidaan pistelaskurit tehdä kutsumalla sitä.

```
void LisaaLaskurit()
{
    pelaajan1Pisteet = LuoPisteLaskuri( Screen.Left + 100.0, Screen.Top - 100.0 );
    pelaajan2Pisteet = LuoPisteLaskuri( Screen.Right - 100.0, Screen.Top - 100.0 );
}
```

`LuoPisteLaskuri`-aliohjelmassahan ensimmäinen parametri oli x-koordinaatti ja toinen y-koordinaatti. Ensimmäisen laskurin x-koordinaatti on tässä laskettu laskemalla yhteen ruudun vasemman reunan x-koordinaatti (`Screen.Left`) sekä vakioarvo `100.0`. Näin laskuri tulee ruudun vasemmasta reunasta hiukan oikealle. Y-koordinaatti lasketaan samaan tapaan ruudun yläreunasta (`Screen.Top`) lukien. Toinen laskuri muuten sama, mutta laskuri tulee ruudun oikeaan reunaan.

Kun nyt ajat ohjelman, pitäisi ruunun yläreunassa näkyä kaksi laskuria, jotka näyttävät arvoa 0.

5. Törmäyksen käsittely

Jotta voisimme kasvattaa pisteitä, täytyisi tietää milloin pallo ohittaa jommankumman mailan. Tämä onnistuu siten, että tarkkaillaan sitä kun pallo osuu johonkin. Jos pallo osuu kentän vasempaan tai oikeaan reunaan (reunathan lisättiin kenttään aikaisemmin), voidaan kasvattaa toisen pelaajan pisteitä.

Törmäyksiin reagoimista varten kirjastossa on aliohjelma nimeltä `AddCollisionHandler`. Se ottaa kaksi parametria: fysiikkaolio, jonka törmäyksiin halutaan reagoida sekä aliohjelma, jota kutsutaan olion törmätessä.

Lisää seuraavanlainen kutsu `LuoKentta`-aliohjelmaan, sen jälkeen kun pallo on luotu ja lisätty tasoon:

```
AddCollisionHandler( pallo, KasittelePallonTormays );
```

Aliohjelman, jossa törmäys käsitellään, **täytyy olla** aina seuraavanlainen: Paluuarvo on `void` (eli ei palauteta mitään) ja parametreina on kaksi `PhysicsObject`-luokan oliota. Ensimmäinen parametri on se olio jonka törmäyksiä kuunnellaan, eli törmääjä (meillä se on siis pallo). Toinen parametri on törmäyksen kohde, jota ei vielä tunneta.

Aliohjelman voi toki nimetä vapaasti, kunhan sama nimi annetaan parametrina `AddCollisionHandler`-kutsussa.

```
void KasittelePallonTormays( PhysicsObject pallo, PhysicsObject kohde )
{
}
```

Kun törmäyksen käsittelevään aliohjelmaan tullaan, tiedetään vasta että pallo on törmännyt **johonkin**.

Jotta selviää mihin se on törmännyt täytyy tehdä hiukan vertailua. Törmäys vasempaan reunaan selviää tarkastamalla onko törmäyksen kohde ja vasen reuna yksi ja sama olio. Miten tämä onnistuisi?

Jotta vertailu olisi helppo suorittaa, meidän täytyy taas hiukan muuttaa aiempaa koodia. Kun aiemmin loimme kenttään kaikki reunat kutsumalla `Level.CreateBorders`-aliohjelmaa, emme saaneet paluuarvona minkäänlaista viittausta luotuihin reunoihin. Jokaiselle reunalle on kuitenkin olemassa oma aliohjelma, joka luo sen. Lisäksi nuo aliohjelmat palauttavat luodun olion, jonka voimme vaikkapa sijoittaa johonkin muuttujaan. Esimerkiksi pelkkä vasen reuna luodaan aliohjelmakutsulla `Level.CreateLeftBorder`. Tämä aliohjelma ei ota vastaan parametreja.

Etsi `LuoKentta`-aliohjelmasta seuraava rivi:

```
Level.CreateBorders( 1.0, false );
```

Korvataan tämä rivi. Sen paikalle luodaan reunat uudestaan yksitellen. Tehdään ensin vasen reuna tähän tapaan:

```
PhysicsObject vasenReuna = Level.CreateLeftBorder();
vasenReuna.Restitution = 1.0;
vasenReuna.IsVisible = false;
```

Aliohjelma `CreateLeftBorder` ei ota vastaan parametreja kuten `CreateBorders`, joten reunan ominaisuudet täytyy muuttaa halutuiksi jälkikäteen. `CreateLeftBorder`:in paluuarvona on luotu reunaolio. Koska reuna on tyyppiä `PhysicsObject`, sijoitetaan `CreateLeftBorder`:in palauttama olio sen tyyppiseen muuttujaan nimeltä vasenReuna. Muuttujan avulla reunan ominaisuuksia voidaan muuttaa.

Tee vasemman reunan luonnin perään **samalla tavalla** oikea reuna (`CreateRightBorder`), alareuna (`CreateBottomBorder`) ja yläreuna (`CreateTopBorder`).

Nyt törmäyksen kohteen tarkistaminen on helppoa. Voidaan tehdä yksinkertainen yhtäsuuruusvertailu ==-merkintää käyttäen (Muista, yhtäsuuruus on C#:ssa kaksi =-merkkiä!) kuten tässä:

```
if ( kohde == vasenReuna )
{
    // ...
}
```

Kun törmäyksen kohde on tiedossa, on pisteiden kasvattaminen helppoa. Jos pallo osuu oikeaan reunaan, kasvatetaan pelaajan 1 pisteitä yhdellä. `IntMeter`-tyyppisen laskurin sisältämään arvoon pääsee käsiksi sen `Value`-ominaisuuden kautta.

Vasempaan reunaan törmäys käsitellään samaan tapaan, nyt vain kasvatetaan pelaajan 2 pisteitä. Näin törmäyksen käsittelystä saadaan seuraavan näköinen:

```
void KasittelePallonTormays( PhysicsObject pallo, PhysicsObject kohde )
{
```

```
    if ( kohde == oikeaReuna )
    {
        pelaajan1Pisteet.Value += 1;
    }
    else if ( kohde == vasenReuna )
    {
        pelaajan2Pisteet.Value += 1;
    }
}
```

Äskeisessä koodissa tuli vastaan vielä yksi uusi asia: toisen `if`-lauseen edessä on sana `else` (suom. muuten).

Tämä tarkoittaa sitä, että `else`-sanan jälkeen tuleva `if`-lause suoritetaan vain, jos sitä edeltävän `if`-lauseen ehto oli epätosi. Tässä tapauksessa `else` ei ole aivan välttämätön, sillä jos pallo osuu kentän oikeaan reunaan, ei se voi samalla osua vasempaan reunaan.

Noista kahdesta `if`-lauseesta ei siis koskaan suoriteta molempia samalla kertaa. On myös hyvin mahdollista, että kumpaakaan niistä ei suoriteta, jos vaikkapa pallo osuu mailaan.

Jotta oikeaan ja vasempaan reunaan päästään käsiksi `KasittelePallonTormays`-aliohjelmassa, ne täytyy taas lisätä attribuuttien joukkoon, josta ne näkyvät kaikille aliohjelmille. Tee siis vielä seuraavat muutokset:

```
public class Peli : PhysicsGame
{
    Vector nopeusYlos = new Vector( 0, 200 );
    Vector nopeusAlas = new Vector( 0, -200 );

    PhysicsObject pallo;
    PhysicsObject maila1;
    PhysicsObject maila2;


    PhysicsObject vasenReuna;
    PhysicsObject oikeaReuna;


    IntMeter pelaajan1Pisteet;
    IntMeter pelaajan2Pisteet;

    // ...

    void LuoKentta()
    {
        pallo = new PhysicsObject( 40.0, 40.0 );
        pallo.Shape = Shapes.Circle;
        pallo.X = -200.0;
        pallo.Y = 0.0;
        pallo.Restitution = 1.0;
        Add( pallo );
        AddCollisionHandler( pallo, KasittelePallonTormays );

        maila1 = LuoMaila( Level.Left + 20.0, 0.0 );
        maila2 = LuoMaila( Level.Right - 20.0, 0.0 );


        vasenReuna = Level.CreateLeftBorder();
```

```
        vasenReuna.Restitution = 1.0;
        vasenReuna.IsVisible = false;

        oikeaReuna = Level.CreateRightBorder();

        oikeaReuna.Restitution = 1.0;
        oikeaReuna.IsVisible = false;
        PhysicsObject ylaReuna = Level.CreateTopBorder();
        ylaReuna.Restitution = 1.0;
        ylaReuna.IsVisible = false;
        PhysicsObject alaReuna = Level.CreateBottomBorder();
        alaReuna.Restitution = 1.0;
        alaReuna.IsVisible = false;

        Level.BackgroundColor = Color.Black;

        Camera.ZoomToLevel();
    }
    // ...
```

6. Hienosäätöä

Pallo ei ehkä nyt käyttäydy aivan toivomallamme tavalla. Voit yrittää hienosäätää pelin pelattavuutta esimerkiksi pallon ominaisuuksia muuttamalla. Tutki mitä tekevät sen `KineticFriction` ja `CanRotate` ominaisuudet ja kokeile muuttaa niitä. Vaikuttaako pelikokemukseen?

7. Lopputulos

Pong-pelin koodi kaikkine mailoineen ja pistelaskuineen näyttää nyt suunnilleen tältä:

Error: Macro Include(source:/trunk/esimerkit/Pong/Vaihe7/Peli.cs) failed

`No node trunk/esimerkit/Pong/Vaihe7/Peli.cs at revision 4058`