Farseer Physics Engine

Farseer Physics Engine

Farseer Physics Engine jest silnikiem fizycznym napisanym dla platformy .NET. Został on zainspirowany przez silnik Box2D znany przede wszystkim z wykorzystania na platformie Flash.
Z jego pomocą można w łatwy sposób symulować oddziaływania fizyczne różnych ciał oraz wykrywać kolizje. Silnik zawiera wiele przydatnych narzędzi ułatwiających zadanie programistom.

Silnik można pobrać na stronie:
http://farseerphysics.codeplex.com/

Działanie

W rzeczywistości, przy pomocy silnika tworzony jest niewidzialny świat, w którym występują wszystkie oddziaływania fizyczne. Sam silnik nie oferuje żadnych możliwości wizualizacji, jednak twórcy silnika sugerują wykorzystanie XNA (z którego będziemy korzystać w tej prezentacji), w którym zostały napisane przykładowe programy dołączone do archiwum z silnikiem.

Należy jednak pamiętać o tym, że Farseer Physics Engine wykorzystuje system metryczny w celu określenia parametrów obiektów, natomiast XNA oczekuje wartości podanych w pikselach. Programista musi zatem zadbać o poprawną konwersję jednostek.

Zaczynamy

Pierwszym krokiem jest stworzenie świata, w którym będą tworzone obiekty. W tym celu tworzymy pole klasy typu World:

```
World swiat;
```

A następnie przypisujemy do niego nową instancję naszego świata

```
swiat = new World(new Vector2(0f, 9.82f));
```

Podany parametr jest wektorem, określającym grawitację, jaka będzie występowała w naszym świecie.

Uaktualnianie świata

Programista sam musi zadbać o uaktualnienie utworzonego świata. W tym celu najlepiej wykorzystać funkcję Update wykorzystywaną przez XNA. Przechodzimy zatem do funkcji:

```
protected override void Update(GameTime gameTime)
```

Wewnątrz niej dodajemy linijkę:

```
swiat.Step((float)gameTime.ElapsedGameTime.TotalMilliseconds * 0.001f);
```

Co dalej?

Mamy zatem pusty świat, w którym póki co nic się nie dzieje. W ramach testu stwórzmy zatem jakieś podłoże, będące statycznym, nieporuszającym się ciałem. Pomimo tego, że ciało statyczne nie porusza się, nadal ma ono swoje parametry fizyczne takie jak tarcie czy restytucja i oddziałuje ono z pozostałymi ciałami.

W celu ułatwienia konwersji jednostek pomiędzy pikselami oraz metrami zadeklarujmy sobie następującą stałą:

```
private const float MeterInPixels = 16f;
```

Tworzenie ciała

Obiektem stanowiącym nasze ciało jest obiekt typu Body. Zadeklarujmy zatem pole przechowujące ciało naszego podłoża:

```
Body podloze;
```

Następnym krokiem jest stworzenie nowego ciała oraz przypisanie go do naszego pola. Takie ciało nie ma jednak zdefiniowanego kształtu.

Tworzenie kształtu

Farseer Physics Engine zawiera łatwą w użyciu klasę, która samodzielnie stworzy nową instancję ciała a następnie przypisze jej kształt. Załóżmy zadem, że nasze ciało ma mieć kształt prostokąta.

```
podloze = BodyFactory.CreateRectangle(swiat, 1280 / MeterInPixels, 76f / MeterInPixels, 1f);
```

Pierwszym parametrem jest świat, w którym zostanie stworzony obiekt. Drugim parametrem jest rozmiar obiektu, natomiast trzeci parametr jest jego skalą.

Parametry podłoża

Mamy zatem nasze ciało. Musimy jednak określić gdzie to ciało się znajduje oraz jakie są jego parametry fizyczne.

```
podloze.Position = new Vector2(640 / MeterInPixels, 700 / MeterInPixels);
podloze.BodyType = BodyType.Static;
podloze.Restitution = 0.1f;
podloze.Friction = 1.5f;
```

Wyświetlanie podłoża

Jak było to wcześniej wspomniane sam silnik nie oferuje żadnych możliwości wizualizacji. Skorzystajmy zatem z XNA w celu wyświetlenia naszego podłoża. Zadeklarujmy następującą zmienną

```
public Texture2D TPodloze;
```

A następnie wczytajmy do niej teksturę:

```
TPodloze = Content.Load<Texture2D>(@"Sprites\groundSprite");
```

Wyświetlanie podłoża

W tym momencie warto wspomnieć o kolejnej „niezgodności” pomiędzy XNA oraz Farseer Physics Engine. XNA określa pozycję według lewego górnego rogu obiektu. Z kolei Farseer wyznacza środek obiektu. W celu skorygowania tego faktu deklarujemy zmienną:

```
Vector2 groundOrigin = new Vector2(TPodloze.Width / 2f, TPodloze.Height / 2f);
```

Wyświetlanie podłoża

Ostatnim krokiem jest wyświetlenie naszego obiektu.

Wewnątrz metody Draw() wpisujemy:

```
spriteBatch.Begin();
spriteBatch.Draw(TPodloze, podloze.Position * MeterInPixels, null,
    Color.White, 0f, groundOrigin, 1f, SpriteEffects.None, 0f);
spriteBatch.End();
```

Co dalej?

Mamy zatem nasze podłoże. Następnym krokiem będzie dodanie dwóch ciał znajdujących się nad ziemią, które zderzą się ze sobą, a następnie spadną na nasze podłoże. Cały proces odbywa się analogicznie jak w przypadku naszego podłoża, jedyną różnicą będzie ustawienie typu ciał na Dynamic a nie Static.

Tekstura

Wczytujemy nową teksturę:

```
public Texture2D TBox;
```

```
TBox = Content.Load<Texture2D>(@"Sprites\boxSprite");
```

I od razu w metodzie Draw() tworzymy do wykorzystania na później zmienną korygującą pozycję:

```
Vector2 boxOrigin = new Vector2(TBox.Width / 2f, TBox.Height / 2f);
```

Tworzenie ciał

Tworzymy dwa ciała:

```
Body box1;
Body box2;
```

I przypisujemy im kształty:

```
box1 = BodyFactory.CreateRectangle
    (swiat, TBox.Width / MeterInPixels, TBox.Height / MeterInPixels, 1f);
box2 = BodyFactory.CreateRectangle
    (swiat, TBox.Width / MeterInPixels, TBox.Height / MeterInPixels, 1f);
```

Parametry pierwszego ciała

Pierwszemu ciału przypisujemy parametry:

```
box1.Position = new Vector2(400 / MeterInPixels, 400 / MeterInPixels);
box1.BodyType = BodyType.Dynamic;
box1.Restitution = 0.3f;
box1.Friction = 2f;
box1.Mass = 5f;
box1.Rotation = 3f;
box1.LinearVelocity = new Vector2(10f, 0);
```

Parametry drugiego ciała

Natomiast drugiemu ciału przypisujemy parametry:

```
box2.Position = new Vector2(600 / MeterInPixels, 400 / MeterInPixels);
box2.BodyType = BodyType.Dynamic;
box2.Restitution = 0.3f;
box2.Friction = 2f;
box2.Mass = 3f;
box2.Rotation = -5f;
box2.LinearVelocity = new Vector2(-10f, 0);
```

Wizualizacja

Ostatnim krokiem jest zadbanie o wizualizację:

```
spriteBatch.Begin();
spriteBatch.Draw
    (TBox, box1.Position * MeterInPixels, null, Color.White,
    box1.Rotation, boxOrigin, 1f, SpriteEffects.None, 0f);
spriteBatch.Draw
    (TBox, box2.Position * MeterInPixels, null, Color.White,
    box2.Rotation, boxOrigin, 1f, SpriteEffects.None, 0f);
spriteBatch.End();
```

Koniec

To byłoby wszystko jeżeli chodzi o tą prezentację, jednak sam silnik oferuje znacznie większe możliwości. Opis niektórych funkcjonalności silnika można znaleźć na stronie CodePlexu'u. Warto także przeanalizować kod przykładowych aplikacji dołączonych do silnika które w bardzo ciekawy sposób prezentują możliwości silnika.