Eric van Feggelen is Senior Programmer bij Avanade

Game development met het Microsoft XNA Framework

TOEVOEGING VAN DE CONTENT PIPELINE AAN XNA IS WAARDEVOL

Microsoft heeft onlangs een nieuw framework uitgebracht waarmee het mogelijk is zelf games te schrijven voor Windows en de Xbox 360-console. De ontwikkelomgeving, Microsoft XNA Game Studio Express (GSE), is een aanvulling op C# Express 2005 en kan gratis gedownload worden. In dit artikel nemen we in vogelvlucht de basisonderdelen van het XNA Framework door en maken we gebruik van de Content Pipeline om een 3d-model uit de Spacewar-starterkit in een eigen project te renderen. Dit doen we 'met een knipoog' naar de XNA Programming Guide door gebruik te maken van GameComponents en serviceproviders.

Microsoft XNA, Cross-platform Next-generation Architecture, is het laatste game development-framework dat Microsoft heeft uitgebracht. Je kunt XNA beschouwen als de opvolger van Managed DirectX. XNA maakt gebruikt van het .NET Framework 2.0. XNA is momenteel primair gericht op studenten en hobbyisten om de ontwikkelkosten van games te drukken en de populariteit van de Xbox 360-console op te schroeven. Je kunt met Microsoft XNA Game Studio Express (GSE) dan ook XNA-projecten naar Xbox 360-consoles deployen en draaien met de XNA Game Launcher. De XNA Game Launcher voor de Xbox is te downloaden via Xbox Live Marketplace. XNA Game Launcher is een aangepaste versie van het Compact Framework waar het XNA Framework in zit. Doordat het allemaal op .NET-technologie draait, is het gemakkelijk te begrijpen en zit er een aantal handige features in zoals incremental deployments en remote debugging. Hoewel je C# Express 2005 en GSE gratis kunt downloaden, kun je alleen deployen naar een Xbox 360 met een lidmaatschap van de XNA-creators club. Dit abonnement kost $99 per jaar. Een XNA-project kun je ook compilen en draaien op een Windows-pc. In versie 1.0 van GSE is het niet mogelijk games te maken voor Windows Vista, deze ondersteuning wordt in een later stadium toegevoegd.

Het framework

Het XNA Framework bestaat uit een aantal fundamentele API's, netjes onderverdeeld in namespaces, die nodig zijn voor het ontwikkelen van games. Hieronder staat een overzicht van de belangrijkste namespaces en een korte omschrijving:

Microsoft.Xna.Framework -
Algemene zaken zoals game-timers en loops
Microsoft.Xna.Framework.Audio -
Low-level API's voor XACT-aansturing
Microsoft.Xna.Framework.Content -
Runtime componenten voor de content pipeline
Microsoft.Xna.Framework.Graphics -
Low-level API's voor 3d-hardware-acceleratie
Microsoft.Xna.Framework.Input -
Input voor toetsenbord, muis en Xbox 360-controller
Microsoft.Xna.Framework.Storage -
Classes voor het schrijven en lezen van bestanden

De Game-class

In het XNA Framework is de Game-class zonder twijfel de belangrijkste. Deze class vormt de basis van ieder XNA-project. Een XNA-project begint altijd met een class die is afgeleid van de Game-class. Deze class dient een aantal virtual methods te overriden, namelijk Initialize, Update en Draw. Om een game te starten, moet de inherited Run-method in de afgeleide class aangeroepen worden, dit gebeurt meestal in de entrypoint van de applicatie. Codevoorbeeld 1 toont een voorbeeld van een lege afgeleide Game-class die wordt aangeboden in een nieuw XNA-project.
Natuurlijk valt je direct op dat er twee methods zijn die ik niet eerder heb benoemd, namelijk *LoadGraphicsContent* en *UnloadGraphicsContent*. Deze methods worden na de Initialize aangeroepen en kunnen worden gebruikt om content te laden. Content in GSE is een belangrijk onderwerp, GSE heeft een Content Pipeline waar ontwikkelaars in design-time content-onderdelen kunnen inladen en compileren naar binaire formaten waar het XNA Framework sneller mee overweg kan tijdens de runtime. Het tweede dat je ongetwijfeld is opgevallen, zijn de twee members graphics en content, instanties van
de *GraphicsDeviceManager-* en *ContentManager-classes.*

GraphicsDeviceManager-class

De GraphicsDeviceManager-class doet exact wat de naam omschrijft, het beheren van een device. Het creëren van een instantie van deze class is voldoende om een device, een handle en een viewport te maken, dit was in Managed DirectX (MDX) veel meer werk. De constructor van de GraphicsDeviceManager vereist een reference naar de game-instantie, zodat door middel van events communicatie tussen de GraphicsDeviceManager- en de Game-class mogelijk wordt.

ContentManager-class

De ContentManager-class beheert de runtime content. Dit betekent dat hij de binaire outputbestanden van de design-time Content Pipeline inlaadt, en daar typesafe toegang aan geeft. De Content Pipeline kan momenteel vijf verschillende

typen content verwerken;

1. Grafisch – 3d-content (.3ds, .fbx, .x)
2. Grafisch – 2d-content (.jpg, .png, .tga, .dds)
3. Geluid en media – XACT (.xap)
4. Materialen – HLSL (.fx)
5. XML – data in alle relevante vormen (.xml)

De ContentManager beheert ook de levenslijn van de resources en hij zal alles netjes opruimen wanneer een instantie wordt gedisposed.

GameComponents

Volgens het application model is het de bedoeling dat de methods Update en Draw gebruikt worden voor game-logic en het tekenen van de frames. De Load- en UnloadGraphicsContent-methods zijn bedoeld om content in te laden, bij voorkeur via de content (ContentManager-class) member. Naarmate een XNA-game wordt uitgebreid, worden de methods Update en Draw in de Game-class complexer en onoverzichtelijker. XNA biedt in de vorm van GameComponents een oplossing voor dit probleem. GameComponents voegen op een abstracte manier functionaliteit toe aan een game. Hiervoor hoeven alleen maar objecten, die zijn afgeleid van de GameComponent-class, toegevoegd te worden aan de Component-collectie van de game-instantie. De methods *Game.Update* en *Game.Draw* roepen de respectieve *GameComponent.Update*- en *GameComponent.Draw*-methods aan voor elke instantie. Met deze techniek kan componentspecifieke logica eenvoudig opgedeeld worden in verschillende GameComponents.

```
public class Game1 : Microsoft.Xna.Framework.Game
{
  GraphicsDeviceManager graphics;
  ContentManager content;

  public Game1()
  {
    graphics = new GraphicsDeviceManager(this);
    content = new ContentManager(Services);
  }

  protected override void Initialize()
  {
    // TODO: Add your initialization logic here
    base.Initialize();
  }

  protected override void LoadGraphicsContent
    (bool loadAllContent)
  {
    if (loadAllContent)
    {
      //TODO: Load any ResourceManagementMode.Automatic content
    }
    // TODO: Load any ResourceManagementMode.Manual content
  }
  protected override void UnloadGraphicsContent
    (bool unloadAllContent)
  {
    if (unloadAllContent == true)
    {
      content.Unload();
    }
  }

  protected override void Update(GameTime gameTime)
  {
    // TODO: Add your update logic here
    base.Update(gameTime);
  }

  protected override void Draw(GameTime gameTime)
  {
    graphics.GraphicsDevice.Clear(Color.CornflowerBlue);
    // TODO: Add your drawing code here
    base.Draw(gameTime);
  }
}
```

Codevoorbeeld 1.

```
public class DrawableGameComponent : GameComponent, IDrawable
```

Codevoorbeeld 2.

Een GameComponent bouwen

Er zijn twee typen GameComponents, de components die alleen logica bevatten en components die ook een visuele representatie hebben. Het XNA Framework heeft hier een onderscheid in gemaakt via twee classes, de *GameComponent*-class en de *DrawableGameComponent*-class. De laatste is simpelweg afgeleid van de eerste en implementeert een interface genaamd *IDrawable* waar de method Game.Draw op controleert voordat de *DrawableGameComponent.Draw*-method wordt aangeroepen zoals te zien is in codevoorbeeld 2.

Wanneer via de Solution Explorer een nieuwe GameComponent wordt aangemaakt, ziet deze er uit als in codevoorbeeld 3. DrawableGameComponents kunnen alleen gemaakt worden door de base-class van een GameComponent handmatig aan te passen.

Een nuttig voorbeeld van een GameComponent is een Frames Per Second (FPS) counter; een getal dat aangeeft hoe vaak de Update-method van de game per seconde wordt uitgevoerd. Hiervoor kan een GameComponent gebruikt worden die dit getal naar de titelbalk van het window schrijft, codevoorbeeld 4 geeft dit weer. Let goed op de private members van de class die boven de Update-method gedeclareerd zijn. De Update-method hoogt ieder keer de count op, tot er een seconde is verstreken. De count wordt op nul gezet en de tijd wordt opgeslagen, zodat er een nieuwe seconde berekend kan worden. Codevoorbeeld 5 geeft weer hoe een GameComponent in gebruik kan worden genomen, door hem aan de Components-collectie van de Game-class toe te voegen.

Events

Natuurlijk is het niet de meest gangbare oplossing om direct de titel van het window aan te passen, omdat componenten niet in staat horen te zijn een game op deze manier te beïnvloeden. Extra informatie in de titel gaat nu verloren of zal beheerd moeten worden in de GameComponent, een plek waar dit niet hoort.

```
public class GameComponent1 : GameComponent
{
  public GameComponent1(Game game)
    : base(game)
  {
    // TODO: Construct any child components here
  }

  public override void Initialize()
  {
    // TODO: Add your initialization code here
    base.Initialize();
  }

  public override void Update(GameTime gameTime)
  {
    // TODO: Add your update code here
    base.Update(gameTime);
  }
}
```

Codevoorbeeld 3.

Een betere oplossing is het gebruik van events. Hiermee kan een method worden gebruikt in de Game-class die de titel van het window aanpast.
Als we beginnen met een delegate die het aantal FPS meegeeft aan de method, zoals te zien is in codevoorbeeld 6, is er alleen nog een event met een aanroep nodig in de GameComponent. In codevoorbeeld 7 en 8 staan de declaratie van het event en de aanroep. Codevoorbeeld 8 kan worden gebruikt in de Update-method van de GameComponent in plaats van de regel die de titel van het window aanpast.
Tot slot zal het event geregistreerd moeten worden in de Game-class op de instantie van de GameComponent. Door deze laatste aanpassingen kan de GameComponent in verschillende games worden gebruikt en zich op verschillende manieren manifesteren. Codevoorbeelden 9 en 10 tonen code om de laatstgenoemde stappen te realiseren.

3d-modellen

Om de boel iets visueler te maken, kan er een GameComponent gemaakt worden die de Content Pipeline gebruikt om een 3d-model in te laden en te renderen. Hiervoor dien je een DrawableGameComponent te gebruiken. De snelste manier om dit te bereiken is het lenen van een 3d-model van de Spacewar Starterkit. Volg de onderstaande stappen om dit te doen:

1. Gebruik GSE om een nieuw XNA-project te maken dat is gebaseerd op de Spacewar-template.
2. Save de solution op een plaats waar de bestanden gemakkelijk gevonden kunnen worden.
3. Gebruik in het bestaande project de Solution Explorer om een Existing Item toe te voegen.
4. Zoek de Spacewar-solution op en selecteer de file /Content/Models/p1_wedge.fbx

Wanneer de solution nu wordt gebuild, zal de design-time Content Pipeline het fbx-bestand compileren naar een binair formaat dat gemakkelijk door de Content Pipeline-runtime in het XNA Framework kan worden geladen. In dit geval treden er build-errors op, omdat de Content Pipeline een aantal textures mist die in het 3d-model wordt gebruikt. Deze textures moeten ook toegevoegd worden aan de solution totdat deze er ongeveer uitziet als in afbeelding 1. De textures worden alleen gebruikt in dit model en het is niet nodig om deze een speciale Build Action te geven. Zo lang de textures maar in de solution zitten en naar de juiste output-directory worden gekopieerd, kan de Content Pipeline het 3d-model compileren.

De solution kan nu wel gebuild worden. De Content Pipeline maakt een nieuw binair bestand aan dat overeenkomt met de AssetName van het content-item. In dit geval is dat "p1_wedge.xnb". De Content Pipeline-runtime (ofwel de ContentManager) kan deze file nu inladen tijdens runtime.
Net zoals de Game-class, hebben GameComponents Unload- en LoadGraphicsContent-methods. Deze kun je in de DrawableGameComponent overschrijven en uitbreiden om je 3d-model in te laden. Het probleem daarbij is dat een GameComponent geen lokale reference heeft naar de ContentManager, deze is namelijk private gedeclareerd in de Game-class. Er is een tweede probleem. Wanneer deze member public gedeclareerd wordt, dan is de content nog niet toegankelijk, omdat de GameComponent geen reference heeft naar de game-instantie, maar naar een instantie van het base-type.

Game-services gebruiken

Om het bovengenoemde probleem te overbruggen, zijn er opnieuw een aantal verschillende mogelijkheden. Mijn voorkeur gaat uit naar het gebruik van services. Via services wordt namelijk de afhankelijkheid tussen de Game-instantie en de GameComponent verbroken, omdat services worden geregistreerd in de base Game-class via een type. Codevoorbeeld 11 toont hoe een service opgehaald wordt aan de hand van een interface-type.

Zoals te zien is in het codevoorbeeld retourneert de GetService-method een object van het genoemde type. Het maakt dus niet uit wat voor object er in de service zit, als de juiste interface maar wordt geïmplementeerd. Het is aan de ontwikkelaar om ervoor te zorgen dat de interface de juiste mogelijkheden biedt. Om voor

```
// Private members
private TimeSpan previousTick;
private int count;

// Update method
public override void Update(GameTime gameTime)
{
  // Check if more than a second has elapsed
  if (gameTime.TotalGameTime.TotalSeconds -
    this.previousTick.TotalSeconds> 1)
  {
    // Update the window title
    this.Game.Window.Title = this.count.ToString();

    // Save current time
    this.previousTick = gameTime.TotalGameTime;

    // Reset count
    this.count = 0;
  }
  else
  {
    // Increment the count
    this.count++;
  }
  // Call base
  base.Update(gameTime);
}
```

Codevoorbeeld 4.

```
MeasureFPS measureFPS = new MeasureFPS(this);
this.Components.Add(measureFPS);
```

Codevoorbeeld 5.

```
public delegate void MeasureFPSEventHandler(int totalFrames);
```

Codevoorbeeld 6.

```
public event MeasureFPSEventHandler OnUpdateFPS;
```

Codevoorbeeld 7.

```
if (this.OnUpdateFPS != null)
this.OnUpdateFPS(this.count);
```

Codevoorbeeld 8.

```
MeasureFPS measureFPS = new MeasureFPS(this);
measureFPS.OnUpdateFPS += new MeasureFPSEventHandler
  (measureFPS_OnUpdateFPS);
this.Components.Add(measureFPS);
```

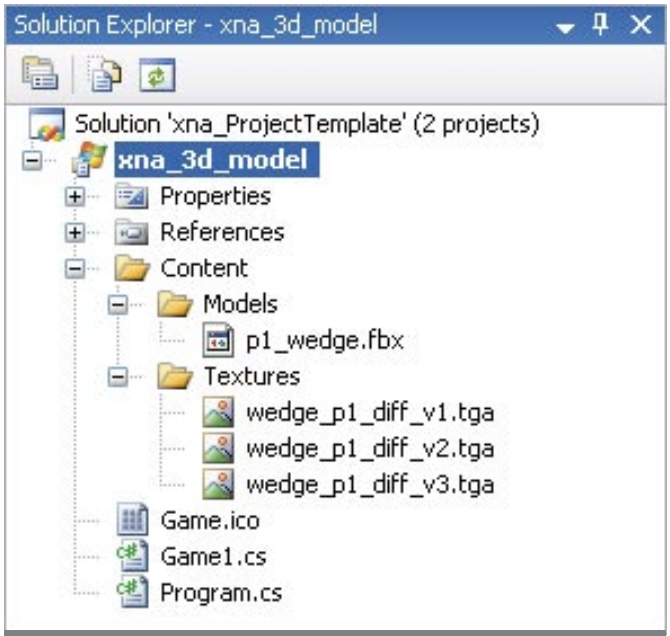
Codevoorbeeld 9.

```
private void measureFPS_OnUpdateFPS(int totalFrames)
{
  this.Window.Title = String.Format("FPS: {0}",
    totalFrames.ToString());
}
```

Codevoorbeeld 10.

```
IGraphicsDeviceService graphicsDeviceService = (IGraphicsDeviceService)
  this.Game.Services.GetService(typeof(IGraphicsDeviceService));
```

Codevoorbeeld 11.



Afbeelding 1. De solution in het XNA-3d-model

elkaar te krijgen dat de ContentManager uit de Game-class kan worden gebruikt, moet de game-instantie geregistreerd worden als service. Codevoorbeeld 12 geeft een interface weer die een ContentManager-property afdwingt. Als de Game-class deze interface implementeert, wordt er een (read-only) property toegevoegd die een ContentManager-object teruggeeft, zoals in codevoorbeeld 12 is te zien. Let op hoe de service wordt geregistreerd in de constructor via het interface-type, met een reference naar de instantie van de game via het keyword *this*.

Nadat de service is geregistreerd, kan deze opgehaald worden in de GameComponent. Hiervoor moet de method LoadGraphicsContent aangepast worden om de game-instantie op te halen, boxed als een IcontentManagerService-type. Het maakt dus niet uit dat de game een interface-type heeft, omdat deze interface toegang geeft tot de ContentManager-property die de content-member teruggeeft. In codevoorbeeld 13 wordt de content-member opgehaald via de service en wordt het model ingeladen via de AssetName van het Content Model.

```
public interface IContentManagerService
{
  ContentManager ContentManager { get; }
}

public class Game1 : Game, IContentManagerService
{
  GraphicsDeviceManager graphics;
  ContentManager content;

  public Game1()
  {
    graphics = new GraphicsDeviceManager(this);
    content = new ContentManager(Services);

    Services.AddService(typeof(IContentManagerService), this);
  }

  public ContentManager ContentManager
  {
    get
    {
      return this.content;
    }
  }
}
```

Codevoorbeeld 12.

Het tekenen van het model

Nu het model is ingeladen, dient de method DrawGameComponent.Draw gebruikt te worden om het model op het scherm te tekenen. Omdat de Content Pipeline het model naar een formaat heeft gecompileerd dat het framework begrijpt, is daar niet veel code voor nodig. Een 3d-model heeft enkele belangrijke properties die de visuele representatie beïnvloeden. Dit zijn de matrices World, View en Projection. Deze drie matrices kunnen rotaties, posities, schaling en projecties opslaan die worden toegepast op elke vertex van het model. Een vertex is een 3-dimensionaal punt (x,y,z) in een model. De drie matrices worden vermenigvuldigd met elke vertex door middel van een vertex-shader die door de Content Pipeline voor dit model is aangemaakt. Codevoorbeeld 14 toont de method Draw van de GameComponent, en vult de matrices World, View, Projection voor elke mesh in het model. Een mesh is een opeenvolging van vertices die gerenderd worden via dezelfde shader. Het 3d-model bestaat in dit geval uit drie verschillende meshes. Uiteindelijk wordt de mesh naar de backbuffer van het device gerenderd met de method Mesh.Draw. Meer is er niet nodig om dit model te tekenen. Vergeet niet de GameComponent die je gebruikt voor het 3d-model toe te voegen aan de Components-collectie van de Game-class. De identity-matrix (eenheidsmatrix) zorgt er in codevoorbeeld 14 voor dat het model geen eigen transformatie heeft ten opzichte van de camera. Maar er zou bijvoorbeeld een rotatie-matrix gebruikt kunnen worden om het model te roteren. De matrix-class heeft hier een aantal statische methods voor zoals de method *Matrix.CreateRotationY(float radians)*.
Helaas is er te weinig ruimte in dit artikel om dieper in te gaan op de 3d-aspecten van XNA. De documentatie van XNA en MDX bevat gelukkig voldoende informatie om de gemiddelde .NET-ontwikkelaar snel op weg te helpen.

```
IContentManagerService contentManagerService=(IContentManagerService)
  this.Game.Services.GetService(typeof(IContentManagerService));

this.model = contentManagerService.ContentManager.Load<Model>
  (@"Content\Models\p1_wedge");
```

Codevoorbeeld 13.

```
public override void Draw(GameTime gameTime)
{
  foreach (ModelMesh mesh in this.model.Meshes)
  {
    foreach (BasicEffect effect in mesh.Effects)
    {
      effect.World = Matrix.Identity;

      effect.View = Matrix.CreateLookAt(new Vector3
        (0f, 50f, 5000f), Vector3.Zero, Vector3.Up);

      effect.Projection = Matrix.CreatePerspectiveFieldOfView
        (MathHelper.ToRadians(45f), 640/480, 1f, 10000f);
    }

    mesh.Draw();
  }

  base.Draw(gameTime);
}
```

Codevoorbeeld 14.

Tot slot

Ik ben van mening dat Microsoft er goed aan heeft gedaan een nieuw game-framework te lanceren dat gebaseerd is op .NET-technologie. De stap voor .NET-ontwikkelaars, studenten en hob-

byisten is ineens een stuk kleiner geworden, en ik verwacht ook dat XNA snel zal uitgroeien met een grote repository aan starterkits, games, snippets en code. Ook de toevoeging van de Content Pipeline aan XNA is zeer waardevol. Dankzij de standaard formaten die XNA gebruikt, is het in Visual C# Express heel gemakkelijk om importers te schrijven die naar dit formaat compileren, mocht dat nodig zijn.

Aan de slag met XNA

Om te starten met XNA heb je een aantal zaken nodig. Ten eerste een machine met Windows XP Professional met SP2. Hierop dient in ieder geval de laatste SDK van DirectX 9.0c en het XNA Framework te draaien. Om te kunnen ontwikkelen, heb je Microsoft Visual C# Express nodig en Microsoft Game Studio Express. De links staan allemaal op de officiële homepage van XNA; zie referenties.

Eric van Feggelen is Senior Programmer bij Avanade (http://www.avanade.com/), een samenwerkingsverband van Microsoft en Accenture. Voor vragen of opmerkingen is Eric te bereiken op ericva@avanade.com of via zijn blog op http://www.fegelein.com/.

Referenties

Microsoft XNA Officiële: http://msdn.microsoft.com/directx/XNA/default.aspx
Microsoft XNA Forums: http://forums.microsoft.com/MSDN/ShowForum.aspx?ForumID=846
Microsoft XNA Team Blog: http://blogs.msdn.com/xna/default.aspx
Microsoft XNA FAQ: http://msdn.microsoft.com/directx/xna/faq/
Microsoft XNA Documentation: http://msdn2.microsoft.com/en-us/library/bb200104.aspx