XNA

Games maken met de XNA bibliotheek

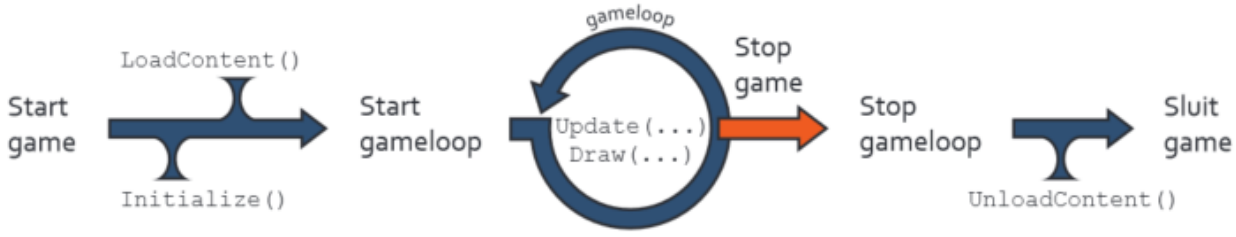
XNA (**X**NA's **N**ot **A**cronymed) Game Studio Express is een door Microsoft ontwikkelde en in december 2006 uitgebrachte game-ontwikkelomgeving, voornamelijk bedoeld voor studenten en hobbyisten. **XNA Game Studio** Express wordt geleverd als een (gratis) uitbreiding voor Visual C# Express 2005 en maakt het mogelijk om games te ontwerpen voor Windows, Xbox 360, Zune media spelers en Windows Phone 7 & 8. XNA Game Studio Express is opgebouwd rond het **XNA Framework**, die is gebaseerd op het .NET Framework.

Het is uitgerust met een set **Managed code** en een uitgebreide klassenbibliotheek die is geoptimaliseerd voor game development. Een ander component, de XNA Framework Content Pipeline toolset maakt het ontwerpers mogelijk om gemakkelijk 2D/3D in de te ontwerpen games op te nemen.

Het XNA Framework zorgt ervoor, naast alle handige dingen die aangeboden worden voor alle managed talen, dat er een bepaald niveau van abstractie bovenop communicatie met **DirectX** wordt gelegd. DirectX9 in het geval van XNA. DirectX is een framework wat er voor zorgt dat je op een uniforme en dus makkelijke manier met de videokaart(en), geluidskaart(en), input controllers en alle aan gaming gerelateerde elementen op een Windows operating systeem kan praten.

Application Life Cycle

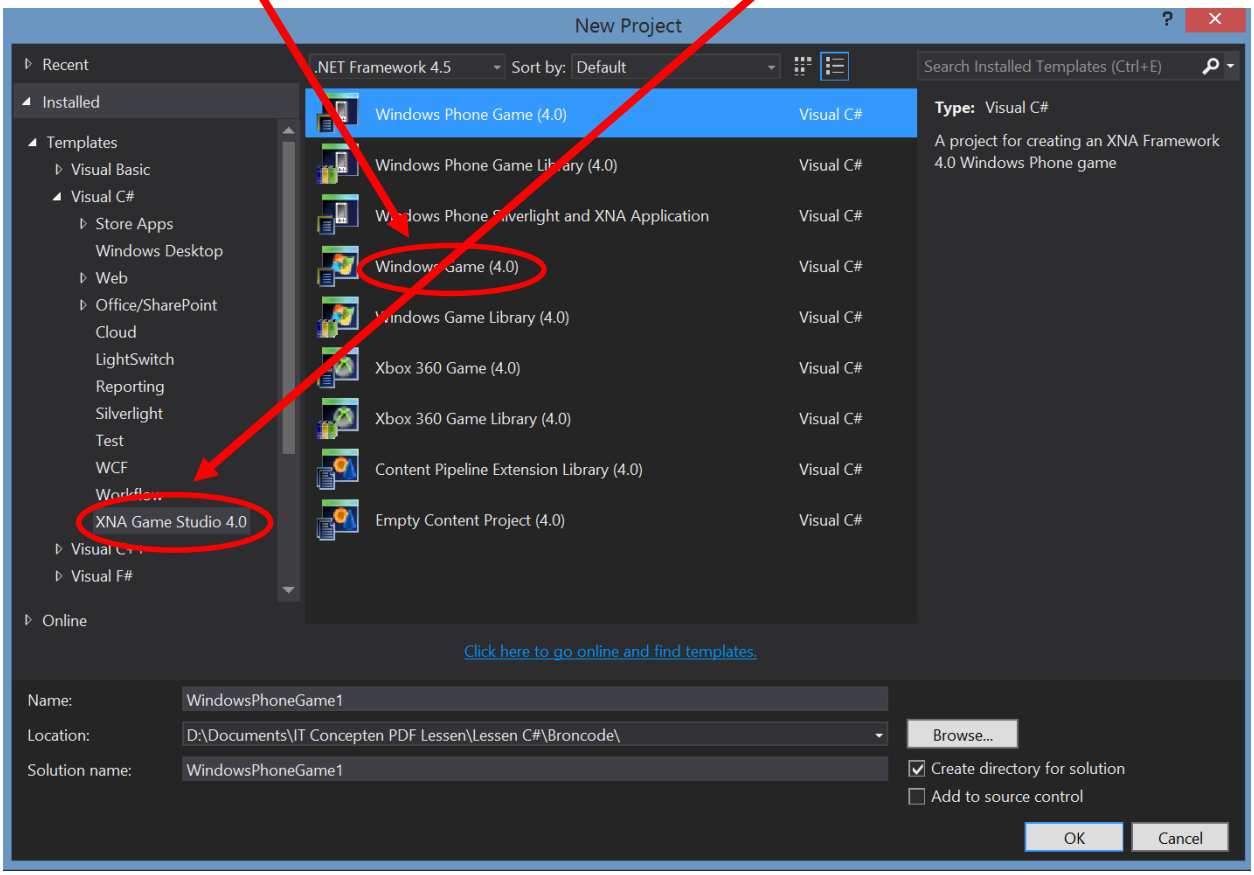
Het is misschien wel even handig om een korte schets te geven van de **application life cycle** van een XNA applicatie. Die ziet er zo uit:



Met de pijl die in het rond gaat, zie je Draw en Update constant worden aangeroepen. Dat is wat genoemd wordt de Game Loop waarin je applicatie leeft. Deze, en de andere functies worden hieronder verder besproken.

XNA Bekijken

Zorg dat je de laatste versie van XNA (versie 4.0) geïnstalleerd hebt. Start Visual Studio, maak een nieuw project aan, kies XNA Game Studio 4.0 en dan Windows Game (4.0).



Geef het project een naam (bijvoorbeeld EersteXNA), en geef het pad aan waar het project moet worden opgeslagen.

De code

We beginnen in een afgeleide class genaamd Game1. Afgeleid van een class die XNA ons aanbiedt. Een Microsoft.Xna.Framework.Game class. Deze basis class is waar het allemaal mee begint voor een XNA applicatie. Als je gebruik wil maken van XNA zal je minimaal een class moeten maken die afgeleid is van de XNA Game class. Je kunt ook een class maken die is afgeleid is van een class van de XNA Game class (polymorphism).

XNA

Bij XNA moet je dan specificeren welke class er tijdens het opstarten van je applicatie gebruikt moeten worden. Het bestand, **Program.cs**, ziet er bij een XNA 4.0 project zo uit:

```
using System;

namespace EersteXNA
{
#if WINDOWS || XBOX
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        static void Main(string[] args)
        {
            using (Game1 game = new Game1())
            {
                game.Run();
            }
        }
    }
#endif
}
```

Iets anders dan die van een console of Windows Forms applicatie.

In het Solution Explorer paneel kun je zien dat er door XNA nog wat andere bestanden zijn aangemaakt en dat er bestanden niet aanwezig zijn die bijvoorbeeld bij de Windows Forms applicatie wel waren, zoals Form1.cs.

Het bestand **Game1.cs**, bevat het skelet voor het maken van een spel. We gaan de code die, automatisch, gegenereerd is nader bekijken. Ook zal je merken dat er al redelijk wat commentaar in de code is aangebracht om aan te geven wat waar moet gebeuren.

De code in het bestand Game1.cs kan verdeel worden in de volgende delen:

- **Using …**
  Hier worden de verschillende bibliotheken die in het programma nodig zijn gedefinieerd.

```
sing System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
```

- **Namespace …**
  Hier binnen komt alle broncode te staan voor het spel. De vijf virtuele functies die vanuit de XNA Game class zijn voorgeschreven en de constructor. Van de klasse.

  We bekijken de verschillende klasen:

- **Game1()**
  Dit is de constructor voor de Game1 klasse.

```
public Game1()
{
    graphics = new GraphicsDeviceManager(this);
    Content.RootDirectory = "Content";
}
```

- **Initialize()**
  Deze method wordt gebruikt om alle spel elementen te initialiseren.

  Dit is de plek waar al de XNA gerelateerde initialisatie gedaan worden. Maak hierbij een scheiding tussen de constructor en deze functie, waarbij de constructor alleen maar gebruikt wordt om mijn member variabelen aan te maken en mogelijk wat zaken in te stellen die absoluut niet met XNA te maken hebben. Alle andere zaken gebeuren in de Intialize functie. **Let er wel op** dat je ook daadwerkelijk de base implementatie aanroept. Die is belangrijk want die zorgt ervoor dat alle gamecomponents (waar we het nog niet over hebben gehad) ook worden aangeroepen. Het is ook nog belangrijk waar je die base methode aanroept. **Het moet op het einde van de methode komen.**

```
protected override void Initialize()
{
    // TODO: Add your initialization logic here

    base.Initialize();
}
```

- **LoadContent()**
  De functie LoadContent is de aangewezen plek is om je **content** (plaatjes, geluid, etc.) te laden. Normaliter is dat de content die je via de **Content Pipeline** hebt geladen. XNA 4.0 maakt hier normaal de **SpriteBatch** aan.

```
protected override void Initialize()
{
    // TODO: Add your initialization logic here

    base.Initialize();
}
```

- **UnloadCintent()**
  Ook UnloadContent is een geen complexe functie. Alle content die je geladen hebt, moet je hier unloaden (verwijderen). We hebben nog geen content, dus de functie is leeg.

```
protected override void UnloadContent()
{
    // TODO: Unload any non ContentManager content here
}
```

- **Update()**
  Met de Update functie komen wij bij één van de twee hoofdfuncties aan binnen een XNA applicatie. In de Update functie is het de bedoeling dat je input van de gebruiker afhandelt. In het standaard voorbeeld ziet het er zo uit:

```
protected override void Update(GameTime gameTime)
{
    // Allows the game to exit
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
        this.Exit();

    // TODO: Add your update logic here

    base.Update(gameTime);
}
```

Je kan wel een beetje raden wat hier gebeurt. Als speler 1 op de back knop drukt op de gamepad, dan stopt de applicatie. Jammer dat er op de Windows Phone geen gamepad zit en dus ook geen back knop, maar het standaard Game Studio voorbeeld stopt dit er wel zo in.

- **Draw()**
  Belangrijke functie nummer twee, de Draw functie wordt, net zoals de Update functie, constant aangeroepen. In de Draw functie is het de bedoeling dat je al je objecten die je op het scherm wil zien tekent. Klinkt simpel en zo is het in principe ook. In de standaard applicatie ziet de functie er zo uit:

```
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // TODO: Add your drawing code here

    base.Draw(gameTime);
}
```

Elke keer dat de Draw functie wordt aangeroepen, zal het scherm worden 'leeggemaakt' met de standaard basis kleur CornflowerBlue ($6495ED RGB).

Compileer en start het programma door in het menu DEBUG te kiezen voor de optie *Start Without Debugging of gebruik de toetsen combinatie* [CTRL +F5].

Het programma zet een licht blauw venster op het scherm en doet verder niets. Dit is het eerste XNA programma.

PONG

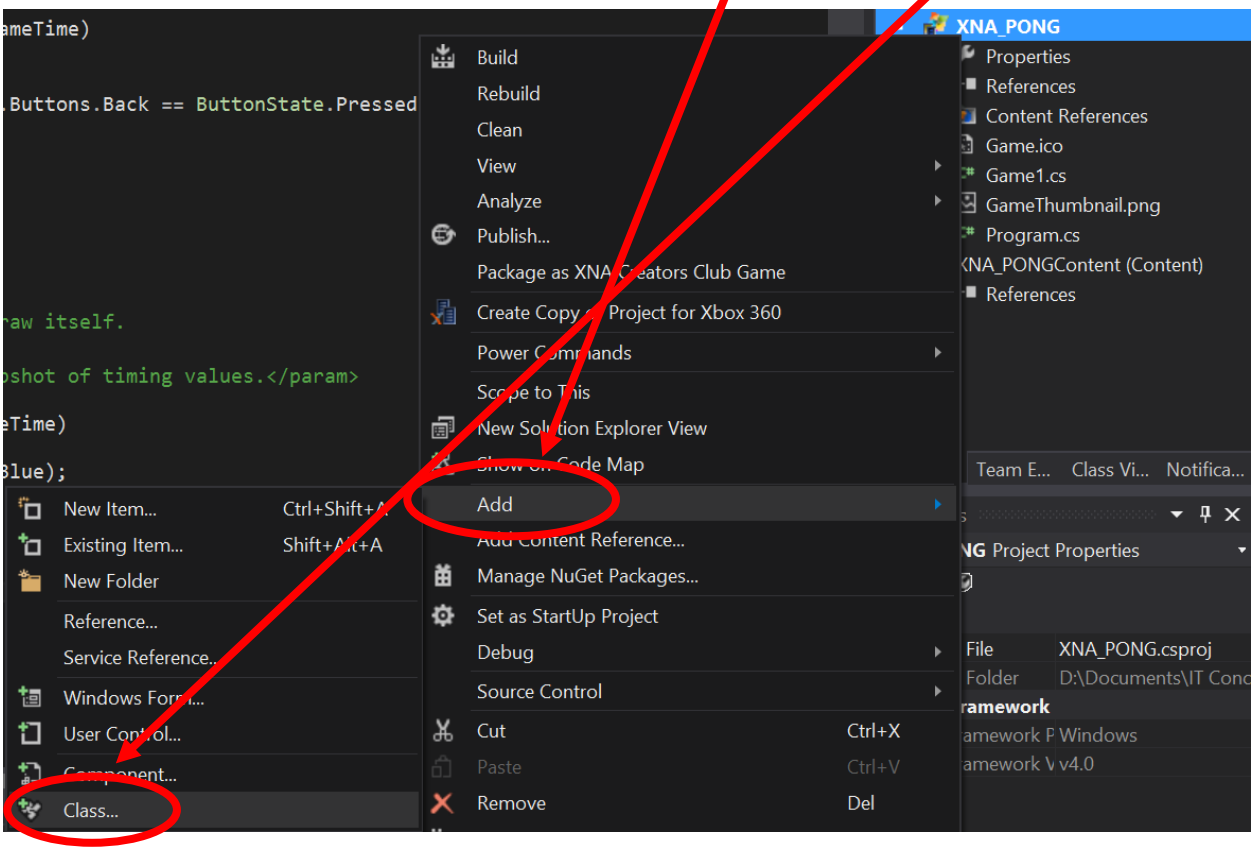
PONG is één van de eerste computer spellen ooit gemaakt. Het is het eerste spel dat commercieel succes boekte. Dit simpel "tennis-achtige" spel gebruikt twee peddels (paddles) en een bal. Het doel is om van de tegenstander te winnen door als eerste 10 punten te halen. Het Originele spel is ontwikkeld door Allan Alcorn en in 1972 uitgebracht door Atari. PONG wordt gezien als het spel dat de video game industrie heeft gestart.

Maken van PONG met XNA en C#

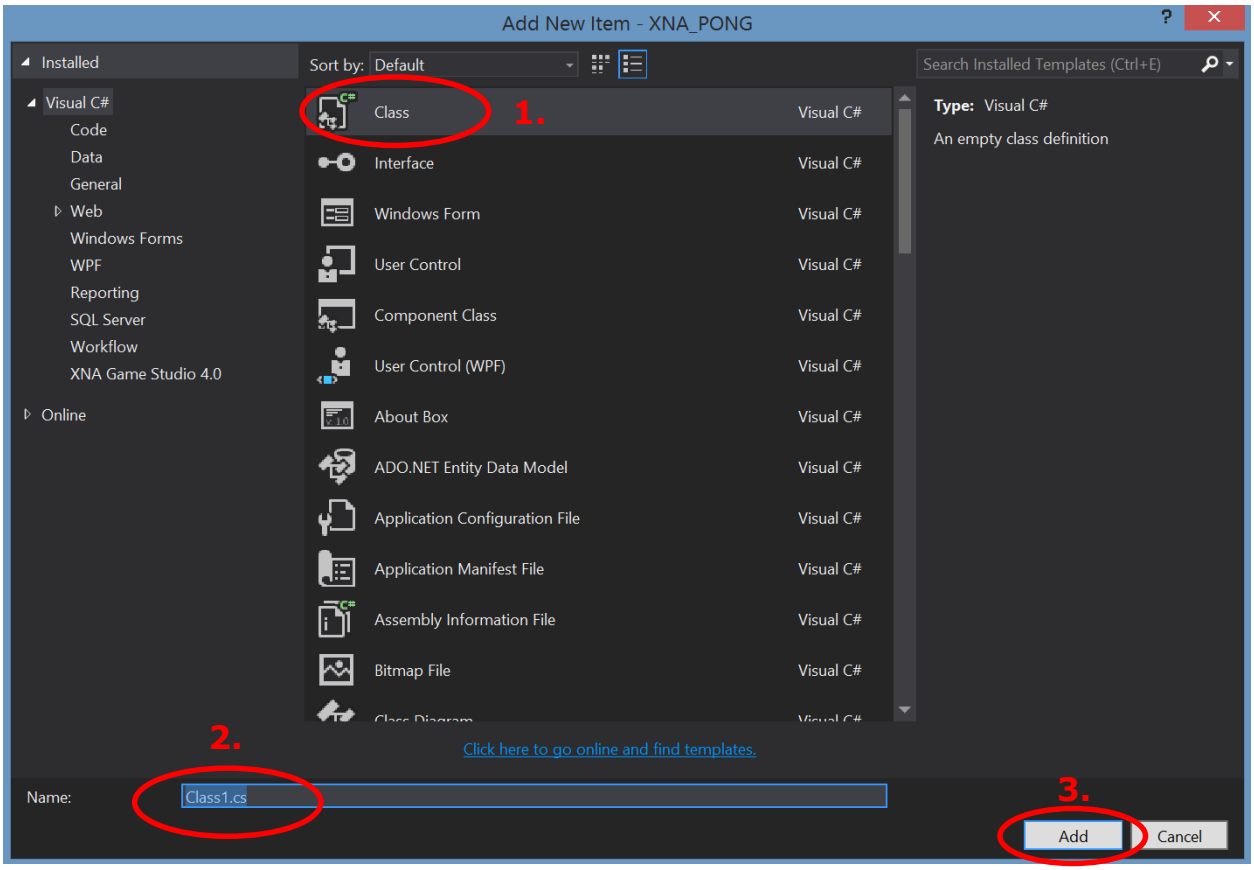
Start Visual Studio en begin een nieuw XNA project.

Ball Klasse

Maak eerst een klasse Ball.cs. Dit kun je doen door in de *Solutuion Explore* rechts te klikken en in het menu de keuzes *Add* en dan *Class* te kiezen.



Na dat je Class gekozen hebt krijg je het volgende dialoogvenster te zien:



Kies de optie Class en geef het de naam Ball. Klik daarna op de Add knop. Visual Studio maakt het bestand Ball.cs aan, plaats het in de Solution Explorer en opent het in de editor venster.
Voer de volgende code in de Ball klasse in:

```
//position of ball
public Point pos;
public int h_speed, v_speed;

//constructor - position
public Ball(int x, int y)
{
    pos = new Point(x, y);

    Random rand = new Random();
    h_speed = rand.Next(3, 7);
    if (rand.Next(0, 2) == 0) h_speed *= -1;

    rand = new Random();
    v_speed = rand.Next(3, 7);
    if (rand.Next(0, 2) == 0) v_speed *= -1;
}
```

Het kan zijn dat er een rode slinger onder het woord Point komt te staan. Dit geeft een warning/error aan. Point is namelijk een onderdeel van de XNA bibliotheek.

Als je boven in het bestand kijkt waar de **using** opdrachten staan, blijkt dat er geen verwijzing naar de XNA Framework wordt gemaakt. Type de volgende code onderaan het lijst je met al aanwezige using opdrachten om het probleem te verhelpen.

```
using Microsoft.Xna.Framework;
```

Paddle klasse

Herhaal de stappen om een nieuw leeg klasse bestand te maken. Noem deze klasse *Paddle.cs*.

Voer de onderstaande code bij de klasse Paddle in. Vergeet niet om ook de *using.Microsoft.Xna.Framework;* boven aan toe te voegen.

```
class Paddle
{
    //position of paddle
    public Point pos;
    public int speed;

    //constructor - position
    public Paddle(int x, int y)
    {
        pos = new Point(x, y);
        speed = 3;
    }
}
```

Game klasse

Ga terug naar de Game klasse en definieer de variabelen en objecten die we nodige hebben voor het spel. De code moet er als volgt uit zien:

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


    Texture2D t_paddle1, t_paddle2, t_ball;
    SpriteBatch spriteBatch;

    Paddle paddle1;
    Paddle paddle2;
    Ball ball;

    Random rand = new Random();
    KeyboardState currentState;
    GamePadState currentPad;
```

Texture2D is een klasse die een 2D rooster van texels. Een texel representeert de kleinste maat van een texture dat gelezen of geschreven kan worden naar een GPU. Simpel gezegd een plaatje.

Dus t_paddle1 en t_paddle2 zullen straks de afbeeldingen voor de peddles bevatten en t_ball die van de bal.

De variabelen paddle1 en paddle2 representeren een instantie van de klasse Paddle. De variabele ball representeert een instantie van de klasse Ball.

De variabele rand bewaard een willekeurig getal.

De variabele currentState representeert een instantie van het object KeyBoardState.

De variabele currentPad representeert een instantie van het object GamePadState.

UpdateBall() method

Er is een method nodig die de verplaatsing van de bal controleert. Dat doen we met de UpdateBall() method. Schrijf de code voor deze klasse onder de sluit accolade van de Draw method in the Game1.cs klasse. De code voor deze klasse ziet er als volgt uit:

```
void UpdateBall()
{
    //update positions
    ball.pos.X += ball.h_speed;
    ball.pos.Y += ball.v_speed;

    //check for boundaries
    //bottom
    if (ball.pos.Y > (Window.ClientBounds.Height - 10 - t_ball.Height))
        ball.v_speed *= -1;

    //top
    if (ball.pos.Y < 10)
        ball.v_speed *= -1;
}
```

De x en y posities van de bal worden aangepast met de waarden die staan in ball.h_speed en ball.v_speed. Deze zijn eigenschappen afkomstig van de klasse Ball.cs.

UpdatePaddles()

Verder wordt in dit stuk code gecontroleerd of de bal niet buiten de boven en onder grenzen komt van het speel venster.
Op dezelfde manier moet er een method komen die de peddels controleert. Daarvoor wordt de method UpdatePaddles() gebruikt. De code in de functie is van commentaar voor zien en duidelijk te begrijpen. Die code is als volgt:

```
void UpdatePaddles()
{
    //get keyboard keys
    currentState = Keyboard.GetState();
    Keys[] currentKeys = currentState.GetPressedKeys();
    //check for up and down arrow keys
    foreach (Keys key in currentKeys)
    {
        if (key == Keys.Up)
            paddle1.pos.Y -= paddle1.speed;
        if (key == Keys.Down)
            paddle1.pos.Y += paddle1.speed;
        if (key == Keys.Escape)
            this.Exit();
    }

    //paddle to move according to ball
    if (paddle2.pos.Y + (t_paddle2.Height / 2) > ball.pos.Y)
        paddle2.pos.Y -= paddle2.speed;
    else if (paddle2.pos.Y + (t_paddle2.Height / 2) < ball.pos.Y)
        paddle2.pos.Y += paddle2.speed;

    //check boundaries
    if (paddle1.pos.Y <= 10) paddle1.pos.Y = 10;
    if (paddle2.pos.Y <= 10) paddle2.pos.Y = 10;

    if (paddle1.pos.Y + t_paddle1.Height >= Window.ClientBounds.Height - 10) paddle1.
pos.Y = Window.ClientBounds.Height - t_paddle1.Height - 10;
    if (paddle2.pos.Y + t_paddle2.Height >= Window.ClientBounds.Height - 10) paddle2.
pos.Y = Window.ClientBounds.Height - t_paddle2.Height - 10;
}
```

Initialize()

In de initialisatie methode, die automatisch is gegenereerd, moet een aanroep gemaakt worden naar de methode ResetGame() om het spel te kunnen (her)starten. De aanpassingen is als volgt:

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

LoadContent()

In de LoadContent methode moeten de afbeeldingen van de peddels en de bal worden geladen in de respectievelijke variabeklen. De methode ziet er na de aanpassingen als volgt uit:

```
protected override void LoadContent()
{
    // Create a new SpriteBatch, which can be used to draw textures.
    t_paddle1 = Content.Load<Texture2D>(@"paddle");
    t_paddle2 = Content.Load<Texture2D>(@"paddle");
    t_ball = Content.Load<Texture2D>(@"ball");
    spriteBatch = new SpriteBatch(GraphicsDevice);

    // TODO: use this.Content to load your game content here
}
```

Draw()

In de Draw() methode moet er wat code geplaatst worden die er voor zorgt dat de **sprites** (afbeeldingen) die we gebruiken op het scherm van het spel worden getekend. De Draw methode ziet er na de aanpassingen als volgt uit:

```
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // TODO: Add your drawing code here
    //spriteBatch.Begin(SpriteBlendMode.AlphaBlend);
    spriteBatch.Begin();
    spriteBatch.Draw(t_paddle1, new Rectangle(paddle1.pos.X, paddle1.pos.Y, t_paddle1
.Width, t_paddle1.Height), Color.White);
    spriteBatch.Draw(t_paddle2, new Rectangle(paddle2.pos.X, paddle2.pos.Y, t_paddle2
.Width, t_paddle2.Height), Color.White);
    spriteBatch.Draw(t_ball, new Rectangle(ball.pos.X, ball.pos.Y, t_ball.Width, t_ba
ll.Height), Color.White);
    spriteBatch.End();


    base.Draw(gameTime);
}
```

Update()
Als laatste moet er code worden toegevoegd in de Update() methode die de methodes aanroepen zodat het spel kan worden uitgevoerd.

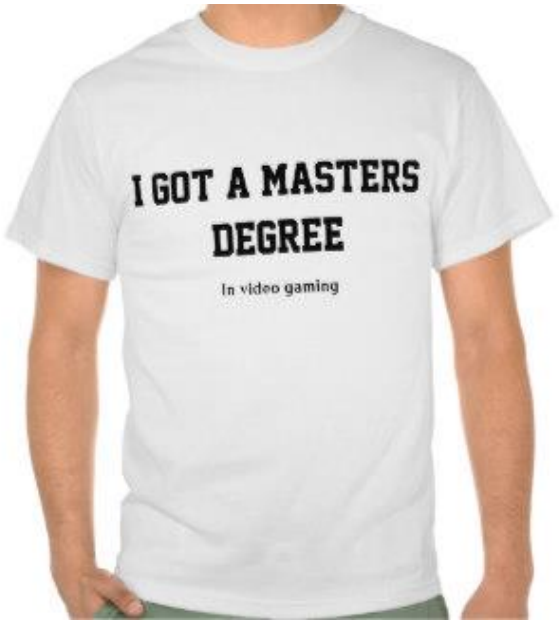
```
protected override void Update(GameTime gameTime)
{
    // Allows the game to exit
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
        this.Exit();

    // TODO: Add your update logic here
    UpdateBall();
    UpdatePaddles();
    CheckCollisions();

    base.Update(gameTime);
}
```

Game ON!
Het spel is nu klaar en kan gespeeld worden. Gebruik de pijltjes toetsen [↑↓] om de linker peddel te bewegen. De andere wordt door het programma gecontroleerd.

Uitbreidingen:
Het spel is nu speelbaar maar er zouden best nog wat dingen aan kunnen worden uitgebreid zoals:

- Een score teller die op het scherm wordt getoond.
- Het aantal levens dat een speler heeft.
- De moeilijkheidsgraad van het spel. De bal kan bij voorbeeld sneller gaan bewegen elke keer nadat het 10 keer op en neer gegaan is.
- Effecten kunnen aan de bal worden gegeven.

- Geluid kan worden toegevoegd.
- Een hoogste score zou kunnen worden bijgehouden, etc.



Probeer nu zelf een aantal van deze functie toe te voegen en het spel uit te breiden.
Je hebt nu een basis kennis gekregen om een computer spel te maken. Begin simpel en breidt dan steeds verder uit. De mogelijkheden zijn alleen beperkt door je eigen creativiteit.

Succes!