الدرس التاسع

أهلا بكم في الدرس التاسع من سلسلة دروس تعلم ال Xna , سوف نقوم في هذا الدرس بتحريك الصاروخ.

في هذا الدرس, سوف نرى كيف بإستطاعتنا نقل الصاروخ و إطلاقه. من السهل تحريك الصاروخ, بما أننا قمنا سابقا بحساب الإتجاه في الدرس السابق. على كل حال, يلزمنا أن يتأثر مسار الصاروخ بالجاذبية, بمعنى أن ما تطلقه إلى الأعلى سوف يعود إلى الأسفل في النهاية. كنتيجة لذلك, بما أن إتجاه الصاروخ سوف يتغير, يجب أن نقوم بتعديل الزاوية التي سوف يرسم الصاروخ بها في كل مره يتغير إتجاهه.

دعنا ننشئ دالة جديدة لكي تقوم بهذه الوظيفة. بما أن هذه الدالة لن ترسم شيئا, ولكنا نحتاج إلى إستدعائها بشكل متكرر إذن سوف نقوم بإستدعائها من الدالة Update , بهذا سوف نتأكد أن الدالة تم إستدعائها 60 مره في الثانية.

```
private void UpdateRocket()
{
    if (rocketFlying)
    {
        rocketPosition += rocketDirection;
    }
}
```

في كل مره يتم إستدعاء هذه الدالة أثناء إنطلاق (طيران) الصاروخ سوف يتم إضافة إتجاه الصاروخ إلي الموقع الحالي له. بما أن إتجاه الصاروخ لا يتغير حاليا, إذن سيتحرك على خط مستقيم. قم بإستدعاء هذه الدالة من داخل الدالة Update :

```
UpdateRocket();
```

الآن قم بتشغيل الكود و أطلق الصاروخ! سوف يطير الصاروخ في خط مستقيم بسرعة تعتمد على طاقة إطلاق الصاروخ

هذا إنجاز إلى حد ما, ولكنه لا يبدو حقيقيا كما أنه لا يسقط على الأرض حتى لو قمت بإطلاقه بطاقة قليلة. بناء على ذلك سوف نقوم بإضافة بعض التأثر بالجاذبية. بحيث أن قوة الجاذبية سوف تقوم بسحب إتجاه الصاروخ للأسفل. قم بإستبدال الكود في دالة UpdateRocket بالكود التالي , طبعا داخل جملة الشرط:

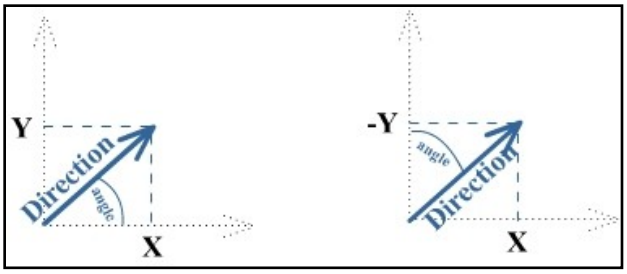
```
Vector2 gravity = new Vector2(0, 1);
rocketDirection += gravity / 10.0f;
rocketPosition += rocketDirection;
```

إتجاه الجاذبية يكون على طول محور Y الموجب, أي إلى الأسفل. في كل مره يتم إستدعاء هذه الدالة (60 مره في الثانية) يتم إضافة جزء من الإتجاه إلى الأسفل إلى إتجاه الصاروخ الأصلي.

الآن قم بتشغيل هذا الكود و قم بتجربة التغييرات! سترى أن الصاروخ يتحرك على منحنى حقيقي, و لكن هناك مشكلة أخرى: و هي أن الصاروخ لا يقوم بالإستداره بناء على الحركة !

و ذلك لان ال SpriteBatch يقوم برسم الصاروخ بعد تدويره بناء على القيمة rocketAngle, و عندما قمنا بتغيير إتجاه الصاروخ rocketDirection لم نقم بتغير قيمة الزاوية بعد. إذن ما نحتاج معرفته هو : **إذا كان الإتجاه معلوما, ما هي الزاوية المرتبطه بهذا الإتجاه.؟**

لإيجاد الزاوية, ألق نظرة على الصورة التالية, على الجانب الأيسر:



الآن, إرجع إلى ايام الثانوية ☺ , كان هنالك بعض من القوانين البسيطة التي يجب أن تكون قد حفظتهم. هذا واحد منهم:

"في المثلث القائم الزاوية, تستطيع أن تجد قيمة الزاوية عن طريق تقسيم طول الضلع المقابل على طول الضلع المجاور, و بعدها أخذ الظل المعكوس "Atan" لناتج القسمة".

تم إشتقاق ذلك من تعريف الجيب و جيب التمام و الظل.

بتطبيق ذلك على مثالنا, تستطيع إيجاد الزاوية بين الإتجاه و محور X المجاور عن طريق أخذ الظل المعكوس "Atan" لل Direction.Y/Direction.X . هذا ما وضح في الرسم السابق الجهة اليسرى: إبتداء من من الإتجاه المعروض هناك, و نريد أن نجد الزاوية بين الإتجاه و محور ال X . ذلك سوف يتم من خلال الكود التالي :

```
Math.Atan(Direction.Y/Direction.X)
```

هذه هي الحالة العامة, و هي التي تعلمناها في رياضيات الثانوية. في حالة الصاروخ الخاص بنا, هناك فرقين بسيطين قمت بتعليمهم بالخط العريض:

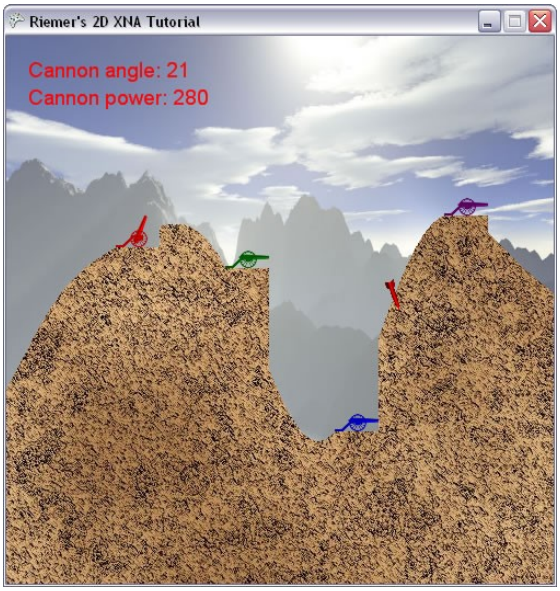
- أننا نريد الزاوية بين الإتجاه ومحور ال **Y**
- في ال Xna , المحور إلى الأعلى يمثل المحور Y **السالب**

المشكلة الأولى يمكن حلها من خلال تغيير مواقع X و Y , و الأخرى عن طريق إعطاء الإشارة السالبة إلى قيمة ال Y. إذن هذا ما يلزمنا من الكود:

```
 rocketAngle = (float)Math.Atan2(rocketDirection.X, -
rocketDirection.Y);
```

بدلا من إستخدام الداله Atan, سوف نحتاج إلى إستخدام الدالة Atan2, و الفرق بينهما أن في الدالة Atan , يجب عليك إرسال ناتج عملية القسمة كوسيط. بما انه من المعروف أن Y-/X- لها نفس نتيجة X/Y , سيتطلب هذا عملية فحص إضافية "if-check". عملية الفحص متوفرة في الدالة Atan2, مما يتيح لنا تحديد كل من المكونين بشكل منفصل.

هذا كل شيئ في هذا الدرس, قم بتشغيل الكود ولاحظ كيف يقوم الصاروخ بتغيير دورانه بناء على الإتجاه!



في هذا الدرس و سابقه, رأينا كيفية إشتقاق الزاوية من الإتجاه و العكس. سوف نحتاج هذه المعلومات في برمجة الألعاب الثنائية و الثلاثية الأبعاد.

التمرين : و اضح أننا بحاجة لمراجعة تمارين الثانوية لكي نستطيع برمجة الألعاب ☺ .

كود المشروع حتى الآن:

```
using System;
using System.Collections.Generic;
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.Net;
using Microsoft.Xna.Framework.Storage;

namespace XNAtutorial
{
    public struct PlayerData
    {
        public Vector2 Position;
        public bool IsAlive;
        public Color Color;
        public float Angle;
        public float Power;
    }

    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;
        GraphicsDevice device;

        int screenWidth;
        int screenHeight;

        Texture2D backgroundTexture;
        Texture2D foregroundTexture;
        Texture2D carriageTexture;
        Texture2D cannonTexture;
        Texture2D rocketTexture;
        SpriteFont font;

        PlayerData[] players;
        int numberOfPlayers = 4;
        float playerScaling;
        int currentPlayer = 0;

        bool rocketFlying = false;
        Vector2 rocketPosition;
        Vector2 rocketDirection;
        float rocketAngle;
        float rocketScaling = 0.1f;

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

```
        }

        protected override void Initialize()
        {
            graphics.PreferredBackBufferWidth = 500;
            graphics.PreferredBackBufferHeight = 500;
            graphics.IsFullScreen = false;
            graphics.ApplyChanges();
            Window.Title = "Riemer's 2D XNA Tutorial";

            base.Initialize();
        }

        protected override void LoadContent()
        {
            device = graphics.GraphicsDevice;
            spriteBatch = new SpriteBatch(device);

            screenWidth = device.PresentationParameters.BackBufferWidth;
            screenHeight = device.PresentationParameters.BackBufferHeight;


           backgroundTexture = Content.Load<Texture2D> ("background");
           foregroundTexture = Content.Load<Texture2D> ("foreground");
           carriageTexture = Content.Load<Texture2D> ("carriage");
           cannonTexture = Content.Load<Texture2D> ("cannon");
           rocketTexture = Content.Load<Texture2D> ("rocket");
           font = Content.Load<SpriteFont> ("myFont");
           SetUpPlayers();
           playerScaling = 40.0f / (float)carriageTexture.Width;
       }

       private void SetUpPlayers()
       {
           Color[] playerColors = new Color[10];
           playerColors[0] = Color.Red;
           playerColors[1] = Color.Green;
           playerColors[2] = Color.Blue;
           playerColors[3] = Color.Purple;
           playerColors[4] = Color.Orange;
           playerColors[5] = Color.Indigo;
           playerColors[6] = Color.Yellow;
           playerColors[7] = Color.SaddleBrown;
           playerColors[8] = Color.Tomato;
           playerColors[9] = Color.Turquoise;

           players = new PlayerData[numberOfPlayers];
           for (int i = 0; i < numberOfPlayers; i++)
           {
               players[i].IsAlive = true;
               players[i].Color = playerColors[i];
               players[i].Angle = MathHelper.ToRadians(90);
               players[i].Power = 100;
           }

           players[0].Position = new Vector2(100, 193);
           players[1].Position = new Vector2(200, 212);
           players[2].Position = new Vector2(300, 361);
           players[3].Position = new Vector2(400, 164);
       }

       protected override void UnloadContent()
       {
       }

       protected override void Update(GameTime gameTime)
       {
           if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
               this.Exit();

           ProcessKeyboard();

            UpdateRocket();

            base.Update(gameTime);
        }

        private void ProcessKeyboard()
        {
            KeyboardState keybState = Keyboard.GetState();
            if (keybState.IsKeyDown(Keys.Left))
                players[currentPlayer].Angle -= 0.01f;
            if (keybState.IsKeyDown(Keys.Right))
                players[currentPlayer].Angle += 0.01f;

            if (players[currentPlayer].Angle > MathHelper.PiOver2)
                players[currentPlayer].Angle = -MathHelper.PiOver2;
            if (players[currentPlayer].Angle < -MathHelper.PiOver2)
                players[currentPlayer].Angle = MathHelper.PiOver2;

            if (keybState.IsKeyDown(Keys.Down))
                players[currentPlayer].Power -= 1;
            if (keybState.IsKeyDown(Keys.Up))
                players[currentPlayer].Power += 1;
            if (keybState.IsKeyDown(Keys.PageDown))
```

```
                players[currentPlayer].Power -= 20;
            if (keybState.IsKeyDown(Keys.PageUp))
                players[currentPlayer].Power += 20;

            if (players[currentPlayer].Power > 1000)
                players[currentPlayer].Power = 1000;
            if (players[currentPlayer].Power < 0)
                players[currentPlayer].Power = 0;

            if (keybState.IsKeyDown(Keys.Enter) || keybState.IsKeyDown(Keys.Space))
            {
                rocketFlying = true;
                rocketPosition = players[currentPlayer].Position;
                rocketPosition.X += 20;
                rocketPosition.Y -= 10;
                rocketAngle = players[currentPlayer].Angle;
                Vector2 up = new Vector2(0, -1);
                Matrix rotMatrix = Matrix.CreateRotationZ(rocketAngle);
                rocketDirection = Vector2.Transform(up, rotMatrix);
                rocketDirection *= players[currentPlayer].Power / 50.0f;
            }
        }

        private void UpdateRocket()
        {
            if (rocketFlying)
            {
                Vector2 gravity = new Vector2(0, 1);
                rocketDirection += gravity / 10.0f;
                rocketPosition += rocketDirection;
                rocketAngle = (float)Math.Atan2(rocketDirection.X, -rocketDirection.Y);
            }
        }

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

            spriteBatch.Begin();
            DrawScenery();
            DrawPlayers();
            DrawText();
            DrawRocket();
            spriteBatch.End();

            base.Draw(gameTime);
        }

        private void DrawScenery()
        {
            Rectangle screenRectangle = new Rectangle(0, 0, screenWidth, screenHeight);
            spriteBatch.Draw(backgroundTexture, screenRectangle, Color.White);
            spriteBatch.Draw(foregroundTexture, screenRectangle, Color.White);
        }

        private void DrawPlayers()
        {
            foreach (PlayerData player in players)
            {
                if (player.IsAlive)
                {
                    int xPos = (int)player.Position.X;
                    int yPos = (int)player.Position.Y;
                    Vector2 cannonOrigin = new Vector2(11, 50);

                    spriteBatch.Draw(cannonTexture, new Vector2(xPos + 20, yPos - 10), null, player.Color,
player.Angle, cannonOrigin, playerScaling, SpriteEffects.None, 1);
                    spriteBatch.Draw(carriageTexture, player.Position, null, player.Color, 0, new Vector2(0,
carriageTexture.Height), playerScaling, SpriteEffects.None, 0);
                }
            }
        }

        private void DrawText()
        {
            PlayerData player = players[currentPlayer];
            int currentAngle = (int)MathHelper.ToDegrees(player.Angle);
            spriteBatch.DrawString(font, "Cannon angle: " + currentAngle.ToString(), new Vector2(20, 20),
player.Color);
            spriteBatch.DrawString(font, "Cannon power: " + player.Power.ToString(), new Vector2(20, 45),
player.Color);
        }

        private void DrawRocket()
        {
            if (rocketFlying)
                spriteBatch.Draw(rocketTexture, rocketPosition, null, players[currentPlayer].Color, rocketAngle, new
Vector2(42, 240), 0.1f, SpriteEffects.None, 1);
        }
    }
}
```