الدرس الحادي عشر

أهلا بكم في الدرس الحادي عشر من سلسلة دروس تعلم ال Xna , سوف نقوم في هذا الدرس بعمل رسم للألوان في خامة لكل بكسل على حدة.

في هذه اللحظة, لدينا الصاروخ طائر في الهواء, ولكن ليس هناك أي نوع من التصادم مع الأرض. قبل تمكننا من فعل ذلك سوف نحتاج أن نعرف بالضبط الإحداثيات الخاصة بالتضاريس. في هذه اللعبة هذا سهل إلى حد ما: لكل إحداثي X على الشاشة, يوجد إحداثي Y محدد للتضاريس.

لقد قمنا في الدروس السابقه بإرتكاب خطأ كبير (بشكل مقصود طبعا) ألا وهو أننا قمنا بإستخدام صورة للأرضيه طولها 500بكسل على محور Y. و لكننا في هذا الدرس سوف نستخدم طريقة افضل من تلك ألا وهي اننا سوف نقوم بتعرف إنحدار "Slope" التضاريس بأنفسنا , وسوف نقوم بإنشاء صورة الأرضيه الأمامية (التضاريس) بشكل ديناميكي.

يمكن عمل ذلك بإستخدام دالتين: GenerateTerrainContour الذي يقوم بإنشاء الإنحدارات , و CreateForeground الذي يقوم بإنشاء صورة الأرضيه الأمامية بناء على الإنحدار للتضاريس.سوف نبدأ بإستخدام دالتين بسيطتين بحيث يعطوننا نتائج بسيطة, و سوف نقوم في الدرس القادم بتحسين هذه الدوال من أجل الحصول على نتائج افضل, إن شاء الله.

ناتج دالة GenerateTerrainContour سوف يكون ببساطة مصفوفة من الأرقام الصحيحة "Integers" تخزن قيمة واحدة من الإحداثي Y لكل إحداثي X في الشاشة. بما أننا سوف نحتاج هذه المصفوفة في الكود الخاص بنا فيما بعد, إذن دعنا نضيف المتغير التالي في أعلى الكود:

```
int[] terrainContour;
```

بما أن قيم الإحداثيات Y هي أرقام صحيحة, لذلك إستخدمنا مصفوفة من الأرقام الصحيحه "ints" لتخزينها.

بعد ذلك في دالة GenerateTerrainContour , التي تقوم بتجهيز هذه المصفوفة و تعبئتها بالقيم. كما قلنا سابقا فإننا سوف نبدأ بنسخة بسيطة جدا :

```
private void GenerateTerrainContour()
{
    terrainContour = new int[screenWidth];

    for (int x = 0; x < screenWidth; x++)
        terrainContour[x] = screenHeight / 2;
}
```

سوف يحتوي المتغير TerrainContour على قيم بعدد قيم إحداثيات X الموجودة في الشاشة, يمكن إيجاد عرض الشاشة في المتغير screenWidth الذي قمنا بتعريفه في دروس سابقة. بعدها, لكل إحداثي X سوف نقوم بتخزين نفس الإحداثي Y. ذلك سوف ينتج لدينا تضاريس مسطحة أفقيا, في وسط الشاشة.

دعنا ننتقل إلى الدالة CreateForeground. هذه الدالة يجب أن تقوم بإنشاء خامة الأرضية الأمامية, بناء على محتويات المصفوفة terrainContour :

```
private void CreateForeground()
{
    Color[] foregroundColors = new Color[screenWidth * screenHeight];
```

```
     for (int x = 0; x < screenWidth; x++)
     {
         for (int y = 0; y < screenHeight; y++)
         {
             if (y > terrainContour[x])
                 foregroundColors[x + y * screenWidth] = Color.Green;
             else
                 foregroundColors[x + y * screenWidth] =
Color.TransparentBlack;
         }
     }
 }
```

السطر الأول يقوم بإنشاء مصفوفة, من أجل تخزين الألوان. قمنا بتجيهزها من أجل تخزين لون لكل بسكل على الشاشة.

بعدها, قمنا بالمرور على كل زوج من إحداثيات X و Y, لتغطية كل الإحداثيات الموجودة على الشاشة. لكل بكسل, قمنا بتفحص فيما إذا كان يقع فوق أو اسفل المنحدر الخاص بالتضاريس. إذا كانت اسفل الإنحدار (بمعنى أن الإحداثي Y **اكبر** من الإحداثي Y المخزن في المصفوفة terrainContour), نقوم بتخزين لون أخضر في المصفوفة foregroundColors. أما إذا كانت النقطة أعلى المنحدر الخاص بالأرض فنقوم بتخزين لون اسود شفاف كلون TransparantBlack, بمعنى أن الألوان أسفل هذه النقطة أو صورة الخلفية سوف تبقى ظاهرة.

في نهاية الدالة, لدنيا مصفوفة من الألوان, ولكن بقي علينا أن نقوم بإنشاء خامة "texture" لهذه الألوان. هذا سهل التنفيذ بكل الأحوال في ال Xna خصوصا, كل ما نحتاجه هو أن نضع السطرين التاليين في نهاية الدالة CreateForeground:

```
 foregroundTexture = new Texture2D(device, screenWidth, screenHeight,
1, TextureUsage.None, SurfaceFormat.Color);
 foregroundTexture.SetData(foregroundColors);
```

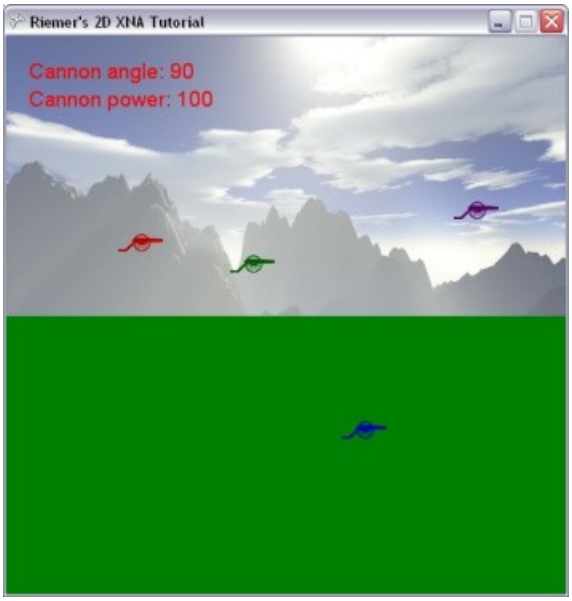
السطر الأول يقوم بإنشاء خامة فارغة. هذا يعني أن جزء من ذاكرة بطاقة الرسوميات سوف يتم حجزها, مساحة كافية لتخزين لون لكل بكسل على الشاشة. الوسيط الثاني و الثالث يقوم بتعريف كم عدد البكسلات التي نريد تخزينها في الخامة, أما الوسيط الأخير يعني أننا نريد ان نخزن لون في كل بكسل من الخامة. الوسيط الخاص بال "mipmapping" و "TextureUsage" خارج نطاق هذه الدروس لذا لن نتطرق لهما.

السطر الثاني يقوم عمليا بنسخ الألوان من المصفوفة إلى الذاكرة في بطاقة الرسوميات.

الآن في دالة LoadContent , إبحث عن السطر الذي يقوم بتحميل صورة الأرضية الأمامية "foreground" من الملف, و قم **بحذفها**. و قم بوضع الأسطر التالية في نهاية الدالة:

```
 GenerateTerrainContour();
 CreateForeground();
```

الآن قم بتشغيل الكود! يجب أن ترى كما في الصورة التالية:



كما نرى, تم حذف صورة الأرضية الأمامية. و النصف الأسفل مملوء باللون الأخضر. هذا يعني أننا نجحنا في إنشاء خامة, مكون من الألوان التي قمنا بتحديدها لكل بكسل.

في الدرس القادم, سوف نقوم بعمل بعض التعديلات على الإنحدار الخاص بالأرض, و على ألوان الأرضية لكي نتخلص من اللون الأخضر.

حاول حل التمرين التالي:

بدلا من أن تكون الأرضية مستقيمة في منتصف الشاشة, حاول رسمها من الركن العلوي الأيسر إلى الركن السفلي الأيمن.

طبعا لا تحفظ التغييرات لأننا نريد النتيجه كما هي الآن, من اجل الدرس السابق.

كود المشروع حتى اللحظة:

```
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;
    Texture2D smokeTexture;
    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;


   List<Vector2> smokeList = new List<Vector2> ();
   Random randomizer = new Random();

    int[] terrainContour;

    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");
       carriageTexture = Content.Load<Texture2D> ("carriage");
       cannonTexture = Content.Load<Texture2D> ("cannon");
       rocketTexture = Content.Load<Texture2D> ("rocket");
       smokeTexture = Content.Load<Texture2D> ("smoke");
       font = Content.Load<SpriteFont> ("myFont");
       SetUpPlayers();
       playerScaling = 40.0f / (float)carriageTexture.Width;

        GenerateTerrainContour();
        CreateForeground();
    }
```

```
        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);
        }

        private void GenerateTerrainContour()
        {
            terrainContour = new int[screenWidth];

            for (int x = 0; x < screenWidth; x++)
                terrainContour[x] = screenHeight / 2;
        }

        private void CreateForeground()
        {
            Color[] foregroundColors = new Color[screenWidth * screenHeight];

            for (int x = 0; x < screenWidth; x++)
            {
                for (int y = 0; y < screenHeight; y++)
                {
                    if (y > terrainContour[x])
                        foregroundColors[x + y * screenWidth] = Color.Green;
                    else
                        foregroundColors[x + y * screenWidth] = Color.TransparentBlack;
                }
            }

            foregroundTexture = new Texture2D(device, screenWidth, screenHeight, 1,
TextureUsage.None, SurfaceFormat.Color);
            foregroundTexture.SetData(foregroundColors);
        }

        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);

                for (int i = 0; i < 5; i++)
                {
                    Vector2 smokePos = rocketPosition;
                    smokePos.X += randomizer.Next(10) - 5;
                    smokePos.Y += randomizer.Next(10) - 5;
                    smokeList.Add(smokePos);
                }
            }
        }

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

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

```
            DrawSmoke();
            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);
        }

        private void DrawSmoke()
        {
            foreach (Vector2 smokePos in smokeList)
                spriteBatch.Draw(smokeTexture, smokePos, null, Color.White, 0, new
Vector2(40, 35), 0.2f, SpriteEffects.None, 1);
        }
    }
 }
```