

الدرس الثالث عشر

أهلاً بكم في الدرس الثالث عشر من سلسلة دروس تعلم الـ Xna, في هذا الدرس سوف نقوم بتحويل الخامة إلى مصفوفة ألوان.

بما أن الإنحدارات قد جهزت أخيراً, نستطيع أن نستمر و نفعل شيئاً بخصوص لون الأرضية الأخضر. بدلاً من استخدام لون ثابت، لكل بكسل في التضاريس سوف نقوم بقراءة اللون المقابل لهذا البكسل في صورة خلفية موجودة مسبقاً، و استخدام هذا اللون في التضاريس الخاصة بنا. ستواجهنا مشكلة في حال كان حجم التضاريس المنشئة بشكل تلقائي أكبر من حجم الصورة الموجودة سابقاً، لهذا السبب سوف نحتاج لصورة يمكن تكرارها 'tileable'. هذا يعني أنك لن ترى حافة عندما تقوم بوضع نسختين من الصورة إلى جانب بعضهما البعض.

مثل هذه الصورة يمكنك تنزيلها من الرابط التالي (او الملفات المرفقة) [هنا](#).

كما نفعل دائماً عند إستيراد صورة إلى المشروع، بعد إستيراد الصورة، قم بإضافة المتغير في أعلى الكود:

```
Texture2D groundTexture;
```

قم بتجهيز المتغير في دالة الـ LoadContent :

```
groundTexture = Content.Load<Texture2D> ("ground");
```

الآن، بدلاً من تحديد لون ثابت سوف نقوم بنقل معلومات اللون من هذه الصورة إلى الخامة foregroundTexture. على كل حال لفعل ذلك، يجب أن نستطيع الوصول إلى الألوان المخزنة في الصورة groundTexture. هذا عكس ما قمنا بفعله قبل عدة دروس: هناك قمنا بإنشاء خامة من مصفوفة ألوان، هنا نريد أن ننسخ البيانات من الخامة إلى مصفوفة الألوان.

ليس من الصعب عمل ذلك، دعنا نبدأ بإضافة الدالة التالية:

```
private Color[,] TextureTo2DArray(Texture2D texture)
{
    Color[] colors1D = new Color[texture.Width * texture.Height];
    texture.GetData(colors1D);
}
```

كما قلنا سابقاً، نريد أن تستقبل هذه الدالة كائن من النوع Texture2D، إستخراج الألوان و إرجاعها كمصفوفة ثنائية الأبعاد. استخدام مصفوفة ثنائية الأبعاد سوف يفيدنا فيما بعد، بحيث أن كل لون في هذه المصفوفة سوف يكون مرتبط ببكسل في الصورة ثنائية الأبعاد.

الآن، هذه الدالة تقوم ببساطة بإنشاء مصفوفة ألوان ببعد واحد، قادرة على تخزين لون واحد لكل بكسل في الصورة (الطول * العرض في الصورة). بعدها تقوم بنسخ معلومات الألوان من الخامة إلى المصفوفة.

ببساطة، هذا كل شيء نحتاج لفعله، إلا أننا نحتاج هذه البيانات في مصفوفة ثنائية البعد وليست أحادية. لذا قم بإضافة الأسطر التالي في نهاية الدالة السابقة :

```
Color[,] colors2D = new Color[texture.Width, texture.Height];
for (int x = 0; x < texture.Width; x++)
    for (int y = 0; y < texture.Height; y++)
        colors2D[x, y] = colors1D[x + y * texture.Width];
```

```
return colors2D;
```

قمنا أولاً بإنشاء و تجهيز المصفوفة ثنائية البعد، كبيره بشكل كافي لتخزين لون واحد لكل بكسل في الخامة.

بعدها قمنا بنسخ البيانات من المصفوفة أحادية البعد إلى الموقع المقابل في داخل المصفوفة ثنائية البعد. في النهاية، قمنا بإرجاع هذه المصفوفة إلى الكود المستدعي لهذه الدالة.

الآن في داخل الدالة CreateForeground سوف نقوم باستخدام الألوان الموجودة في هذه الدالة، بدلا من اللون الأخضر. إذا إذهب إلى الدالة CreateForeground ، و أضف الكود التالي في السطر الأول:

```
Color[,] groundColors = TextureTo2DArray(groundTexture);
```

الآن إبحث عن السطر التالي، الذي يقوم بوضع لون أخضر لكل بكسل في إنحدارات التضاريس:

```
foregroundColors[x + y * screenWidth] = Color.Green;
```

وقم باستبداله بالسطر التالي:

```
foregroundColors[x + y * screenWidth] = groundColors[x, y];
```

الذي سوف يقوم بنسخ اللون الخاص بالبكسل نفسه في خامة الأرضية في داخل المصفوفة foregroundTexture.

هذا سوف يعمل بشكل جيد، إلا إذا قمنا بزيادة طول أو عرض الشاشة أكثر من عرض وطول خامة الأرضية لحل هذه المشكلة سوف نأخذ باقي القسمة لكل منهما، يمكن تفسير باقي القسمة "modulo" بالمثال التالي. إذا كانت ال groundWidth = 400 ؛ إذن باقي قسمة ال 100 على ال 400 سوف يكون 100، بينما باقي قسمة 500 على 400 سوف يكون أيضا 100. باقي قسمة 532 على 400 يساوي 132، و باقي قسمة 389 على 400 هو 389. بمعنى أنها تغطي كل مجال الأرقام [groundWidth, 0] ، و هو ما نحتاجه بالضبط.

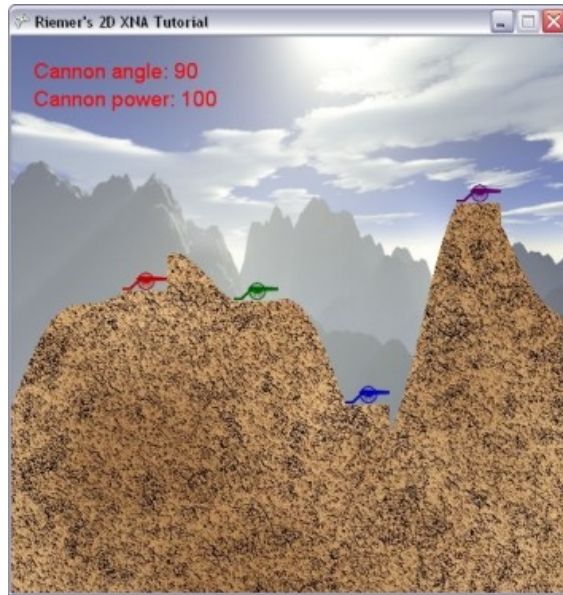
في لغة C# عملية باقي القسمة يمكن تنفيذها من خلال الإشارة '%'، إذن بهذه الطريقة نستطيع معرفة باقي قسمة X على ال groundWidth :

```
x % groundWidth
```

سوف نحتاج لفعل نفس الشيء مع الإحداثي y، لكي نتأكد انها لن تكون أكبر من إرتفاع صورة الأرضية. إذن هذا ما يجب فعله في النهاية:

```
foregroundColors[x + y * screenWidth] = groundColors[x %  
groundTexture.Width, y % groundTexture.Height];
```

عند تشغيل الكود الآن، يجب أن يكون لديك نتيجة مثل التالية:



قبل درسين، تعلمنا كيف ننشئ كائن Texture2D من مصفوفة ألوان ثنائية البعد. في هذا الدرس، تعلمنا العكس: كيف نحصل على مصفوفة ألوان ثنائية البعد من خامه Texture2D. هذا يعني أنك الآن تعرف كل شيء عن معالجة ألوان الخامات على مستوى البكسل.

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;
        Texture2D smokeTexture;
        Texture2D groundTexture;
        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");

        groundTexture = Content.Load<Texture2D> ("ground");

        font = Content.Load<SpriteFont> ("myFont");
        playerScaling = 40.0f / (float)carriageTexture.Width;
        GenerateTerrainContour();
        SetUpPlayers();
        FlattenTerrainBelowPlayers();
        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[i].Position = new Vector2();
        }
    }

```

```

        players[i].Position.X = screenWidth / (numberOfPlayers + 1) * (i + 1);
        players[i].Position.Y = terrainContour[(int)players[i].Position.X];
    }
}

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

    double rand1 = randomizer.NextDouble() + 1;
    double rand2 = randomizer.NextDouble() + 2;
    double rand3 = randomizer.NextDouble() + 3;

    float offset = screenHeight / 2;
    float peakheight = 100;
    float flatness = 70;

    for (int x = 0; x < screenWidth; x++)
    {
        double height = peakheight / rand1 * Math.Sin((float)x / flatness * rand1 + rand1);
        height += peakheight / rand2 * Math.Sin((float)x / flatness * rand2 + rand2);
        height += peakheight / rand3 * Math.Sin((float)x / flatness * rand3 + rand3);
        height += offset;
        terrainContour[x] = (int)height;
    }
}

private void FlattenTerrainBelowPlayers()
{
    foreach (PlayerData player in players)
    {
        if (player.IsAlive)
        {
            for (int x = 0; x < 40; x++)
            {
                terrainContour[(int)player.Position.X + x] =
terrainContour[(int)player.Position.X];
            }
        }
    }
}

private void CreateForeground()
{
    Color[,] groundColors = TextureTo2DArray(groundTexture);
    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] = groundColors[x % groundTexture.Width,
y % groundTexture.Height];
            }
            else
            {
                foregroundColors[x + y * screenWidth] = Color.TransparentBlack;
            }
        }
    }

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

private Color[,] TextureTo2DArray(Texture2D texture)
{
    Color[] colors1D = new Color[texture.Width * texture.Height];
    texture.GetData(colors1D);

    Color[,] colors2D = new Color[texture.Width, texture.Height];
    for (int x = 0; x < texture.Width; x++)
    {
        for (int y = 0; y < texture.Height; y++)
        {
            colors2D[x, y] = colors1D[x + y * texture.Width];
        }
    }

    return colors2D;
}

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);
    }
}

```