

الدرس الثامن عشر

أهلاً بكم في الدرس الثامن عشر من سلسلة دروس تعلم الـ Xna, في هذا الدرس سوف نتحدث عن موضوع مزج ألفا المضافة "Additive alpha blending".

المشكلة الرئيسية في نتيجة الدرس السابق هو أنه تم رسم صورة الانفجار كما هي بالأصل في الصورة الأصلية. بمعنى أنه تم رسم الخلفية و الحدود السوداء للصورة فوق المحتويات الموجودة في الذاكرة الخلفية "backbuffer" لبطاقة الرسومات. في هذا الدرس سوف نقوم بإصلاح هذه المشكلة من خلال تشغيل مزج ألفا المضافة قبل رسم الجزئيات.

أولاً سنتحدث قليلاً عن الموضوع. في مزج ألفا نحن مهتمين بالحصول على لون جديد للبكسل، اعتماداً على اللونين التاليين:

- اللون الأصلي SourceColor : اللون الجديد، اللون الناتج من الصورة التي يتم رسمها.
- اللون الهدف DestinationColor : لون البكسل السابق الموجود في الذاكرة الخلفية و الذي سوف نقوم بالرسم فوقه.

يتم مزج كلا من اللونين معاً من أجل الحصول على اللون الجديد للبكسل. في الـ Xna يتم ضرب كل من الـ SourceColor و الـ DestinationColor بمعامل:

$$\text{NewColor} = \text{SrcBlend} * \text{SrcColor} + \text{DestBlend} * \text{DestColor}$$

بإستطاعتنا تعريف كلا من الـ SrcBlend و الـ DestBlend بأنفسنا. إفتراضياً، الـ Xna يتوقع أن تحتوي الصورة على معلومات الشفافية لكل بكسل. بعد قنوات الـ R,G و B يوجد قناة الـ A (Alpha)، التي تحتوي على معلومات الشفافية. بعض تنسيقات الصور (مثل PNG) تقوم بتخزين هذه المعلومات، و البعض الآخر (مثل JPG) لا يدعم ذلك.

هذا ما يأخذنا إلى نظام المزج الإفتراضي في الـ Xna. بشكل إفتراضي الـ SpriteBatch يستخدم قيمة ألفا كـ SrcBlend. القيمة المعكوسة، (1-SrcBlend) يستخدم كـ DestBlend.

$$\text{NewColor} = \text{SrcAlpha} * \text{SrcColor} + (1 - \text{SrcAlpha}) * \text{DestColor}$$

مثال على ذلك صورة حامل المدفع، فهي غير شفافة في كل بكسلات حامل المدفع، و شفافة في كل البكسلات الأخرى. إذن لكل بكسل من بكسلات حامل المدفع لدينا التالي:

$$\begin{aligned}\text{NewColor} &= 1 * \text{SrcColor} + 0 * \text{DestColor} \\ \text{NewColor} &= \text{SrcColor}\end{aligned}$$

ما يعني أن بكسلات المدفع سوف يتم كتابتها فوق الألوان الموجودة سابقاً في الذاكرة الخلفية "backbuffer"، كما يجب أن تكون بالضبط.

أما بالنسبة للبكسلات الأخرى في الصورة، بحيث A=0 نحصل على:

$$\begin{aligned}\text{NewColor} &= 0 * \text{SrcColor} + 1 * \text{DestColor} \\ \text{NewColor} &= \text{DestColor}\end{aligned}$$

إن هذه البكسلات لن تغير أي شيء في الذاكرة الخلفية. البكسلات التي لها قيمة ألفا بين ال 0 و ال 1 سوف تنتج مزيج من اللون الجديد (المصدر) و اللون القديم (الهدف).

هذا يكفي عن مزج ألفا حتى الآن، دعنا نناقش الحالة الخاصة بجزيئات الانفجار التي لدينا. قبل رسمهم على الشاشة، سوف نقوم بتغيير ال SrcBlend و ال DestBlend الخاصة بنا بطريقة تجعل ال Xna تضيف الألوان في داخل الانفجار إلى الألوان الموجودة مسبقا في الذاكرة الخلفية.

دعنا نرى كيف يعمل ذلك. لاحظ المناطق السوداء الكبيرة في صورة الانفجار. عمليا الأسود هو ليس لون أصلا إنما هو ناتج من عدم وجود أي ألوان أخرى من ال RGB= 0,0,0. إذن عندما نقوم بإضافة هذا اللون إلى اللون الموجود في الذاكرة الخلفية، لن يكون له أي تأثير. إذن يتبقى لدينا الألوان الموجودة في وسط الصورة (الأحمر - أصفر). هذه البكسلات، عندما يتم رسمها، سوف تقوم بإضافة ال RGB الخاصة بها إلى اللون الموجود سابقا في الذاكرة الخلفية. في حال قمنا برسمهم لمرات متكررة فوق بعضهم البعض سوف نحصل على بقعة بلون أبيض - مصفر مشرق، ناتجة من إضافة ألوان الانفجار معا!

السؤال المتبقي هو: كيف سوف نقوم بإضافة ألوان الصور إلى بعضها البعض؟ الجواب هو من خلال إعطاء معاملي المزج القيمة 1 :

```
NewColor = 1*SrcColor + 1*DestColor
NewColor = SrcColor + DestColor
```

دعنا نرى ما يحصل. في اللحظة قبل رسم صور الانفجار، تحتوي الذاكرة الخلفية على المشهد و المدافع. الآن عندما يتم رسم الصورة الأولى، سوف يتم إضافة ألوانها إلى ألوان البكسلات المقابلة لها في الذاكرة الخلفية. كنتيجة، سوف يتم إضافة بعض من اللون الأصفر و الأحمر للمنطقة في الذاكرة الخلفية المقابلة لمركز الانفجار.

الآن عندما يتم رسم الجزئي الثاني، سوف يتم إضافة ألوانه مرة أخرى إلى الذاكرة الخلفية. بعد رسم الجزئي الثاني، سوف يزداد اللونين الأحمر - أصفر. سوف يزداد هذا الوهج الأحمر الأصفر في كل مره يتم رسم الجزئي فيها.

إذن كيف سوف نقوم بإعطاء معاملات المزج تلك القيمة 1؟ بإمكاننا عمل ذلك في ال Xna من خلال وضع الاسطر التالية قبل رسم أي شيء (لا تفعل الآن):

```
device.RenderState.AlphaBlendEnable = true;
device.RenderState.SourceBlend = Blend.One;
device.RenderState.DestinationBlend = Blend.One;
```

بكل الأحوال، ال SpriteBatch لديها خاصية المزج المضاف مبنية داخليا، بما أن ذلك يستخدم بشكل متكرر في برمجة الألعاب الثنائية الأبعاد. يمكن تفعيلها من الدالة SpriteBatch.Begin. في حال قمت بتفعيل خاصية المزج المضاف في الدالة SpriteBatch.Begin سوف يتم رسم كل الصور بعدها بهذه الطريقة (مالم تستخدم SpriteSortMode.Deferred - لن نشرحها هنا-)، في حالتنا هنا نريد أن نفعل المزج المضاف في صور الانفجارات فقط.

يمكن حل ذلك باستخدام SpriteBatch آخر. أو بما أننا نريد أن نرسم جزيئات الانفجارات كآخر عنصر في المشهد، سوف نقوم برسمها في كتلة SpriteBatch.Begin...SpriteBatch.End منفصلة:

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

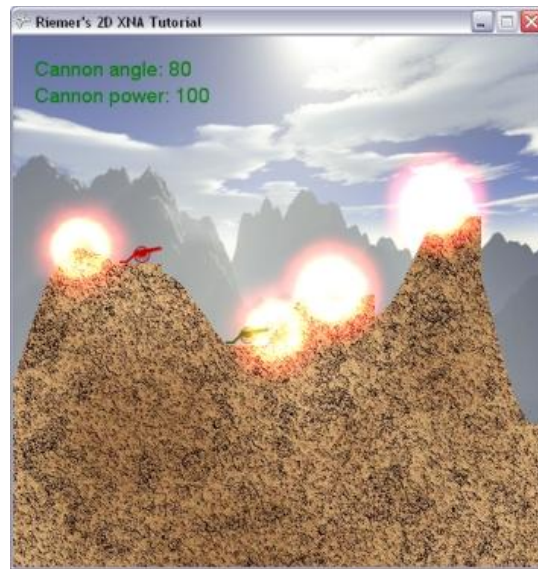
```

    spriteBatch.Begin(SpriteBlendMode.Additive, SpriteSortMode.Deferred,
SaveStateMode.None);
    DrawExplosion();
    spriteBatch.End();

```

بعد أن إنتهى ال spriteBatch من رسم المشهد الرئيسي بإستخدام الإعدادات الافتراضية، قمنا ببديئه مرة أخرى مع تفعيل خاصية المزج المضاف. بإستخدام هذه الإعدادات، يتم فقط رسم جزيئات الانفجارات، و إضافة ألوانها بشكل فعال إلى محتويات المشهد الموجودة مسبقا في الذاكرة الخلفية.

عندما تقوم بتشغيل الكود سوف تكون صورة الانفجار أنعم بشكل كبير، كما هو واضح في الصورة التالية:



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 ParticleData
    {
        public float BirthTime;
        public float MaxAge;
        public Vector2 OrginalPosition;
        public Vector2 Accelaration;
        public Vector2 Direction;
        public Vector2 Position;
        public float Scaling;
        public Color ModColor;
    }

    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;
        Texture2D explosionTexture;
        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<ParticleData> particleList = new List<ParticleData>();
        List<Vector2> smokeList = new List<Vector2> ();           Random randomizer = new Random();
        int[] terrainContour;

        Color[,] rocketColorArray;
        Color[,] foregroundColorArray;
        Color[,] carriageColorArray;
        Color[,] cannonColorArray;

        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");
            explosionTexture = Content.Load<Texture2D> ("explosion");
        }
    }

```

```

        playerScaling = 40.0f / (float)carriageTexture.Width;
        GenerateTerrainContour();
        SetUpPlayers();
        FlattenTerrainBelowPlayers();
        CreateForeground();

        rocketColorArray = TextureTo2DArray(rocketTexture);
        carriageColorArray = TextureTo2DArray(carriageTexture);
        cannonColorArray = TextureTo2DArray(cannonTexture);
    }

    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);
    foregroundColorArray = TextureTo2DArray(foregroundTexture);
}

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();

    if (rocketFlying)
    {
        UpdateRocket();
        CheckCollisions(gameTime);
    }

    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);
        }
    }
}

private Vector2 TexturesCollide(Color[,] tex1, Matrix mat1, Color[,] tex2, Matrix mat2)
{
    Matrix mat1to2 = mat1 * Matrix.Invert(mat2);

    int width1 = tex1.GetLength(0);
    int height1 = tex1.GetLength(1);
    int width2 = tex2.GetLength(0);
    int height2 = tex2.GetLength(1);

    for (int x1 = 0; x1 < width1; x1++)
    {
        for (int y1 = 0; y1 < height1; y1++)
        {
            Vector2 pos1 = new Vector2(x1, y1);
            Vector2 pos2 = Vector2.Transform(pos1, mat1to2);

            int x2 = (int)pos2.X;
            int y2 = (int)pos2.Y;
            if ((x2 >= 0) && (x2 < width2))
            {
                if ((y2 >= 0) && (y2 < height2))
                {
                    if (tex1[x1, y1].A > 0)
                    {
                        if (tex2[x2, y2].A > 0)
                        {
                            Vector2 screenPos = Vector2.Transform(pos1, mat1);
                            return screenPos;
                        }
                    }
                }
            }
        }
    }

    return new Vector2(-1, -1);
}

private Vector2 CheckTerrainCollision()
{
    Matrix rocketMat = Matrix.CreateTranslation(-42, -240, 0) *
Matrix.CreateRotationZ(rocketAngle) * Matrix.CreateScale(rocketScaling) *
Matrix.CreateTranslation(rocketPosition.X, rocketPosition.Y, 0);
    Matrix terrainMat = Matrix.Identity;
    Vector2 terrainCollisionPoint = TexturesCollide(rocketColorArray, rocketMat,
foregroundColorArray, terrainMat);
    return terrainCollisionPoint;
}

private Vector2 CheckPlayersCollision()
{
    Matrix rocketMat = Matrix.CreateTranslation(-42, -240, 0) *
Matrix.CreateRotationZ(rocketAngle) * Matrix.CreateScale(rocketScaling) *
Matrix.CreateTranslation(rocketPosition.X, rocketPosition.Y, 0);
    for (int i = 0; i < numberOfPlayers; i++)
    {
        PlayerData player = players[i];
        if (player.IsAlive)
    }
}

```

```

        {
            if (i != currentPlayer)
            {
                int xPos = (int)player.Position.X;
                int yPos = (int)player.Position.Y;

                Matrix carriageMat = Matrix.CreateTranslation(0, -carriageTexture.Height, 0) *
Matrix.CreateScale(playerScaling) * Matrix.CreateTranslation(xPos, yPos, 0);
                Vector2 carriageCollisionPoint = TexturesCollide(carriageColorArray,
carriageMat, rocketColorArray, rocketMat);
                if (carriageCollisionPoint.X > -1)
                {
                    players[i].IsAlive = false;
                    return carriageCollisionPoint;
                }

                Matrix cannonMat = Matrix.CreateTranslation(-11, -50, 0) *
Matrix.CreateRotationZ(player.Angle) * Matrix.CreateScale(playerScaling) *
Matrix.CreateTranslation(xPos + 20, yPos - 10, 0);
                Vector2 cannonCollisionPoint = TexturesCollide(cannonColorArray, cannonMat,
rocketColorArray, rocketMat);
                if (cannonCollisionPoint.X > -1)
                {
                    players[i].IsAlive = false;
                    return cannonCollisionPoint;
                }
            }
        }
        return new Vector2(-1, -1);
    }

    private bool CheckOutOfScreen()
    {
        bool rocketOutOfScreen = rocketPosition.Y > screenHeight;
        rocketOutOfScreen |= rocketPosition.X < 0;
        rocketOutOfScreen |= rocketPosition.X > screenWidth;

        return rocketOutOfScreen;
    }

    private void CheckCollisions(GameTime gameTime)
    {
        Vector2 terrainCollisionPoint = CheckTerrainCollision();
        Vector2 playerCollisionPoint = CheckPlayersCollision();
        bool rocketOutOfScreen = CheckOutOfScreen();

        if (playerCollisionPoint.X > -1)
        {
            rocketFlying = false;

            smokeList = new List<Vector2> ();
            NextPlayer();
            AddExplosion(playerCollisionPoint, 10, 80.0f, 2000.0f, gameTime);
        }

        if (terrainCollisionPoint.X > -1)
        {
            rocketFlying = false;

            smokeList = new List<Vector2> ();
            NextPlayer();
            AddExplosion(terrainCollisionPoint, 4, 30.0f, 1000.0f, gameTime);
        }

        if (rocketOutOfScreen)
        {
            rocketFlying = false;

            smokeList = new List<Vector2> ();
            NextPlayer();
        }
    }

    private void NextPlayer()
    {
        currentPlayer = currentPlayer + 1;
        currentPlayer = currentPlayer % numberOfPlayers;
        while (!players[currentPlayer].IsAlive)
        {
            currentPlayer = ++currentPlayer % numberOfPlayers;
        }
    }
}

```

```

private void AddExplosion(Vector2 explosionPos, int numberOfParticles, float size, float
maxAge, GameTime gameTime)
{
    for (int i = 0; i < numberOfParticles; i++)
        AddExplosionParticle(explosionPos, size, maxAge, gameTime);
}

private void AddExplosionParticle(Vector2 explosionPos, float explosionSize, float maxAge,
GameTime gameTime)
{
    ParticleData particle = new ParticleData();

    particle.OriginalPosition = explosionPos;
    particle.Position = particle.OriginalPosition;

    particle.BirthTime = (float)gameTime.TotalGameTime.TotalMilliseconds;
    particle.MaxAge = maxAge;
    particle.Scaling = 0.25f;
    particle.ModColor = Color.White;

    float particleDistance = (float)randomizer.NextDouble() * explosionSize;
    Vector2 displacement = new Vector2(particleDistance, 0);
    float angle = MathHelper.ToRadians(randomizer.Next(360));
    displacement = Vector2.Transform(displacement, Matrix.CreateRotationZ(angle));

    particle.Direction = displacement;
    particle.Accelaration = 3.0f * particle.Direction;

    particleList.Add(particle);
}

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

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

    spriteBatch.Begin(SpriteBlendMode.Additive, SpriteSortMode.Deferred, SaveStateMode.None);
    DrawExplosion();
    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, player.Scaling, SpriteEffects.None, 1);
            spriteBatch.Draw(carriageTexture, player.Position, null, player.Color, 0, new
Vector2(0, carriageTexture.Height), player.Scaling, 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), rocketScaling, 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);
}

private void DrawExplosion()
{
    for (int i = 0; i < particleList.Count; i++)
    {
        ParticleData particle = particleList[i];
        spriteBatch.Draw(explosionTexture, particle.Position, null, particle.ModColor, i, new
Vector2(256, 256), particle.Scaling, SpriteEffects.None, 1);
    }
}
}
}

```