

الدرس السادس عشر

أهلاً بكم في الدرس السادس عشر من سلسلة دروس تعلم الـ Xna, في هذا الدرس سوف نقوم بتشغيل كل ما عملناه في إكتشاف التصادمات.

في هذا الدرس سوف نضع كل ما تعلمناه في الدرسين السابقين معاً، و سننتهي بإكتشاف تصادمات (على مستوى البكسل) يعمل بكفاءة. سوف نقوم بإنشاء دالتين، سوف تقوم الأولى بإكتشاف التصادم بين الصاروخ و الأرض من جهة، و بين الصاروخ واللاعبين من جهة أخرى. و الدالة الثانية تقوم بفحص فيما إذا كان الصاروخ مازال معروضاً على الشاشة.

بناءً على ما قمنا بتجهيزه في الدروس السابقة، من السهل أن نكتشف التصادم بين الصاروخ و التضاريس:

```
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;
}
```

قمنا بإنشاء مصفوفتي التحويل لصورة الصاروخ و التضاريس، قمنا بتمريرهم مع مصفوفات الألوان لكل من الصورتين إلى الدالة TexturesCollide. سوف ترجع هذه الدالة (-1,-1) إذا لم يتم إكتشاف تصادم، و إلا سوف يتم إرجاع إحداثي على الشاشة يمثل موقع التصادم. سوف تقوم الدالة CheckTerrainCollision بإرجاع هذه النتيجة.

بعدها سوف نقوم بكتابة الدالة CheckPlayersCollision، و التي تبدو أعقد قليلاً لأن هنالك أكثر من لاعب، ولأنه قد يكون هناك بعض اللاعبين ليسوا على قيد الحياة. إبدأ بالكود التالي:

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

أولاً، تم إنشاء المصفوفة الخاصة بالصاروخ، والتي سوف تكون نفسها لكل اللاعبين بعدها لكل نقوم بفحص كل لاعب فيما إذا كان ما يزال على قيد الحياة، وفيما إذا لم يكن هو اللاعب الذي قد أطلق الصاروخ، لأنه سوف يكون هناك تصادم مع اللاعب المطلق في لحظة إطلاق الصاروخ. إذا تحققت الشروط السابقة، سوف نقوم بإنشاء مصفوفة التحويل لحامل المدفع الخاص باللاعب الحالي، كما شرحنا في الدرس السابق.

تم تمرير المصفوفتين الخاصتين بالمدفع و الصاروخ إلى الدالة TexturesCollide، و النتيجة سوف يتم تخزينها في الكائن من نوع Vector2 المسمى carriageCollisionPoint. تذكر أن قيمتها سوف تكون (-1,-1) فيما إذا لم يكن هناك تصادم، وهو الذي نقوم بتفحصه في المقطع التالي من الكود، قم بوضع الكود التالي مباشرة بعد السطر الذي يتم فيه ملئ المتغير : carriageCollisionPoint

```
if (carriageCollisionPoint.X > -1)
{
    players[i].IsAlive = false;
    return carriageCollisionPoint;
}
```

إذا تم إكتشاف تصادم بين الصاروخ و اللاعب الحالي، سوف يتم تغيير قيمة المتغير IsAlive للاعب الحالي إلى false وسوف ترجع هذه الدالة نقطة التصادم، وبعدها يتم إيقاف تنفيذ الدالة.

إذا لم يتم إكتشاف تصادم مع حامل المدفع، يجب علينا أن نفحص فيما إذا كان هناك تصادم بين الصاروخ و المدفع الخاص باللاعب الحالي. ذلك يتم بنفس الطريقة، ولكننا سوف نقوم بإنشاء مصفوفة التحويل للمدفع بدلاً من حاملة المدفع:

```
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;
}
```

مرة أخرى، إذا تم إكتشاف تصادم بين المدفع و الصاروخ، سوف يتم إرجاع إحداثي التصادم على الشاشة. إذا لم يكن هناك تصادم سوف يستمر الكود في التنفيذ، و في الدورة الثانية للتكرار سوف يتم تنفيذ نفس عمليات الفحص السابقة للاعب التالي في حال لم يكتشف أي تصادم بين الصاروخ و أي لاعب، سوف تنتهي الحلقة التكرارية وسوف ترجع الدالة CheckPlayersCollision الإحداثي (-1,-1).

في النهاية، سوف نحتاج إلى دالة لكي نتأكد إذا ما كان الصاروخ داخل حدود الشاشة حتى الآن سيكون ذلك سهلاً:

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

    return rocketOutOfScreen;
}
```

هذه الدالة تفحص إذا ما كان الصاروخ أدنى الحد السفلي، أو على يسار الشاشة، أو على على يمين الشاشة. سوف تكون النتيجة صحيحة في حال كانت اي من الشروط السابقة صحيحة، وسوف ترجع الدالة القيمة true إلى الكود المستدعي.

حتى هذه اللحظة، قمنا بإنشاء 3 دوال مما يتيح لنا إكتشاف أي تصادم ممكن على الشاشة. كل ما نحتاجه هو دالة عامة تقوم بإستدعاء الدوال السابقة و معالجة نتائجها:

```
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();
    }

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

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

    if (rocketOutOfScreen)
    {
        rocketFlying = false;

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

هذه الدالة تستدعي الدوال الثلاث، و تقوم بتخزين نتائجهم في متغيرات. الجمل الشرطية الثلاثة تفحص فيما إذا قامت إحدى الدوال بإرجاع تصادم. إذا تم ذلك سوف نتوقف عن رسم الصاروخ، سوف يتم مسح ال smokeList ويتم تفعيل اللاعب التالي. في هذه اللحظة، الجمل الشرطية الثلاث تقوم بنفس العمل، و لكنني كتبتهم بشكل منفصل لأنه في الدروس القادمة سيكون لكل تصادم إنفجار مختلف.

هناك شيئين آخرين علينا عملهم: تعريف الدالة NextPlayer و إستدعاء إكتشاف التصادم من الدالة Update:

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

الدالة يجب عليها زيادة قيمة المتغير currentPlayer، و تفحص فيما إذا كان اللاعب الجديد حيا:

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

أولاً، يتم زيادة قيمة المتغير `currentPlayer`. بما أن هذا المتغير يجب أن لا يكون أكبر من قيمة `numberOfPlayers`، قمنا بأخذ باقي القسمة. كمثال، إذا كان `numberOfPlayers = 4`، عندما يتم زيادة الرقم 3 ليصبح 4، يتم إرجاع الرقم إلى 0 عن طريق باقي القسمة.

بعدها، نقوم بفحص فيما إذا كان اللاعب التالي على قيد الحياة. إذا لم يكن كذلك، نقوم بعملية الزيادة من جديد. السطر داخل الحلقة التكرارية `While` يقوم بنفس وظيفة السطر الأول والثاني تماماً: وضع `++` قبل المتغير `currentPlayer` يتيح زيادة قيمته قبل أن يتم أخذ باقي القسمة.

عندما نرجع من هذه الدالة، سوف يكون المتغير `currentPlayer` يحتوي على رقم اللاعب التالي الذي على قيد الحياة.

الآن، بعد إضافة كل ذلك إلى المشروع بشكل صحيح، يجب أن يكون الكود قادراً على إكتشاف التصادمات بين الصاروخ و أي كائن في طريقة!

ملاحظة سريعة على أداء هذه الطريقة: العديد من الحسابات سوف يتم تنفيذها على كل بكسل في الصورة الأولى. هذا يعني أنك عندما تريد أن تفحص التصادم بين صورتين، من الأفضل عليك أن تمرر الصورة الصغيرة في الوسيط الأول `tex1` و الصورة الكبرى في الوسيط الثاني `tex2`. علاوة على ذلك، بدلاً من أن تقوم بعملية الفحص بشكل أعمى بالتفصيل لكل الصور الموجودة، يجب عليك أن تفحص أولاً احتمالية تقاطع هذه الصور من خلال فحص إذا ما لظنت الحدود الخارجية لهذه الصور متقاطعة. إذا كانت الحدود الخارجية غير متقاطعة، سوف تكون متأكداً أنه لن يكون هناك تصادم، إذن سيكون من غير المجدي أن يتم إستدعاء الدالة `TexturesCollide`.

إذا كنت غير متأكد من مواقع بعض أجزاء الكود، ألق نظرة على كود المشروع بالكامل:

```
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;

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");
    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();
    }

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

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

    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;
    }
}

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), 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);
        }
    }
}

```