الدرس العشرون

أهلا بكم في الدرس العشرون من سلسلة دروس تعلم ال Xna, في هذا الدرس سوف نتحدث عن حفر الأرض نتيجة الإنفجار.

بالرغم من أن الإنفجارات التي قمنا بتجهيزها في الدروس الثلاث السابقة تبدو جميلة، و لكن ليس لها أي تأثيرات على التضاريس حتى الآن. و هو ما سنقوم بعمله.

بالرغم من أنه لن يكون هناك أي مفاهيم حقيقية جديدة في هذا الدرس، إلا أنني قررت إضافته إلى هذه السلسلة لبرمجة الألعاب الثنائية الأبعاد بإستخدام Xna ليكون مثالا آخر على إستعمال المصفوفات. كل ما نحتاج إلى إضافته هو دالة بإسم AddCrater. بما أننا نريد أن تأخذ هذه الحفر شكل صورة الإنفجار، سوف نحتاج أن نمرر مصفوفة الألوان ثنائية الأبعاد لهذه الصورة. أيضا، لمعرفة أين على الشاشة يجب أن تظهر الحفرة، و ما هو الحجم الذي يجب أن تظهر به سوف نحتاج لتمرير مصفوفة التحويل الخاصة بصورة الإنفجار.

إبدأ بالكود التالي:

```
private void AddCrater(Color[,] tex, Matrix mat)
{
    int width = tex.GetLength(0);
    int height = tex.GetLength(1);

    for (int x = 0; x < width; x++)
    {
        for (int y = 0; y < height; y++)
        {
            if (tex[x, y].R > 10)
            {
                Vector2 imagePos = new Vector2(x, y);
                Vector2 screenPos = Vector2.Transform(imagePos, mat);

                int screenX = (int)screenPos.X;
                int screenY = (int)screenPos.Y;
            }
        }
    }
}
```

نقوم أولا بإيجاد العرض و الطول الخاص بصورة الإنفجار (بالبكسل)، مما يتيح لنا المرور على كل البكسلات في الصورة بإستخدام حلقتي تكرار. لكل بكسل، نتأكد أولا أن هذا البكسل ليس أسود بشكل كامل. تذكر أنه دائما المناطق في مركز صورة الإنفجار تحتوي على بعض اللون، أما الحدود الخارجية فهي سوداء تماما. نحن نريد فقط أن نحذف التضاريس من الأجزاء من الصورة التي تحتوي على الإنفجار.

بعدها، نحسب إحداثيات البكسل على الشاشة (يمثل هذا الموقع المكان الذي سوف يتم فيه رسم البكسل على الشاشة). يتم ذلك عن طريق تحويل إحداثيات الصورة بإستخدام مصفوفة التحويل الخاصة بالصورة. بما أن الشاشة لدينا لا تحتوي على بكسلات بإحداثي (4.121,23.423)، نقوم بتحويل إحداثيات ال X و ال Y إلى أعداد صحيحة (لاحظ أن هناك طريقة أدق ولكنها أبطئ لعمل التحويل و هي من خلال تقريب إحداثيات X و Y).

أخيرا، إذا كان إحداثي الشاشة ليس خارج منطقة الشاشة، نقوم بفحص فيما إذا كان البكسل سوف يتم رسمه في التضاريس في هذه الحالة نقوم بخفض التضاريس في هذه النقطة:

```
if ((screenX) > 0 && (screenX < screenWidth))
     if (terrainContour[screenX] < screenY)
     terrainContour[screenX] = screenY;
```

هذا كل شيئ من أجل الدالة، كل ما نحتاجه هو إستدعائها بشكل صحيح. أولا نحتاج أن نخزن مصفوفة ألوان ثنائية الأبعاد، لذلك قمنا بتعريف المتغير التالي في أعلى الكود:

```
Color[,] explosionColorArray;
```

وقم بملئها في نهاية الدالة LoadContent:

```
explosionColorArray = TextureTo2DArray(explosionTexture);
```

بعد ذلك، سوف نقوم بإستدعاء الدالة AddCrater. يجب أن يتم ذلك مرة واحدة لكل إنفجار، إذن المكان الأفضل لفعل ذلك هو الدالة AddExplosion. لإنشاء المصفوفة، بإستطاعتنا إستخدام الموقع و الحجم الخاص بالإنفجار. لإضافة بعض العشوائية، سوف نقوم بإضافة تدوير عشوائي إلى المصفوفة، إذن إبدأ بإضافة الكود التالي في نهاية الدالة AddExplosion:

```
float rotation = (float)randomizer.Next(10);
```

بعدها، نحن جاهزون لإنشاء مصفوفة التحويل الخاصة بصورة الإنفجار. هذه المصفوفة لها بالضبط نفس الشكل الذي قمنا بشرحة قبل درسين:

- أولا، نظام الإحداثيات الخاص بصورة الإنفجار تم تحريكه إلى مركز الإنفجار.
- بعدها، تم تصغيره، بناء على حجم الإنفجار و حجم صورة الإنفجار.
- بعدها، تم تدويره بزاوية عشوائية.
- في النهاية، تم تحريك نظام إحداثيات ال XY الخاص بصورة الإنفجار، بحيث أنه تم تحريك مركز الصورة إلى المكان الذي كانت فيه نقطة الأصل الخاصة بنظام إحداثي ال XY للصورة.

بالكود:

```
Matrix mat = Matrix.CreateTranslation(-explosionTexture.Width / 2, -
explosionTexture.Height / 2, 0) * Matrix.CreateRotationZ(rotation) *
Matrix.CreateScale(size / (float)explosionTexture.Width * 2.0f) *
Matrix.CreateTranslation(explosionPos.X, explosionPos.Y, 0);
```

الآن بعد حصولنا على مصفوفة الألوان ثنائية الأبعاد و المصفوفة الخاصة بالإنفجار، بإستطاعتنا إستدعاء الدالة AddCrater:

```
AddCrater(explosionColorArray, mat);
```

هذا كل شيئ! سوف يتم تشوية التضاريس نتيجة الإنفجار. بكل الأحوال، عند قيامك بتشغيل الكود لن ترى أي تغيرات ☹ . سبب ذلك أننا ما زلنا بحاجة إلى تحديث خامة التضاريس foregroundTexture. قبل أن نفعل ذلك، سوف نقوم بتحديث مواقع الآعبين في حال تم تغيير التضاريس تحتهم في حالة إصابتها:

```
for (int i = 0; i < players.Length; i++)
    players[i].Position.Y = terrainContour[(int)players[i].Position.X];
FlattenTerrainBelowPlayers();
CreateForeground();
```

الآن عندما تقوم بتشغيل الكود، يجب ان ترى حفره جديدة في كل مره يصطدم فيها الصاروخ بالأرض أو بأحد اللآعبين. لاحظ أن حجم الحفرة يعتمد على حجم الإنفجار.

الكود 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 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;

        Color[,] explosionColorArray;

        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);

            explosionColorArray = TextureTo2DArray(explosionTexture);
        }

        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();

            if ((!rocketFlying) && (particleList.Count == 0))
                ProcessKeyboard();

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

            if (particleList.Count > 0)
                UpdateParticles(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);


            float rotation = (float)randomizer.Next(10);
            Matrix mat = Matrix.CreateTranslation(-explosionTexture.Width / 2, -explosionTexture.Height / 2,
0) * Matrix.CreateRotationZ(rotation) * Matrix.CreateScale(size / (float)explosionTexture.Width * 2.0f) *
Matrix.CreateTranslation(explosionPos.X, explosionPos.Y, 0);
            AddCrater(explosionColorArray, mat);

            for (int i = 0; i < players.Length; i++)
                players[i].Position.Y = terrainContour[(int)players[i].Position.X];
            FlattenTerrainBelowPlayers();
            CreateForeground();
        }

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

            particle.OrginalPosition = explosionPos;
            particle.Position = particle.OrginalPosition;

            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 * 2.0f;
            particle.Accelaration = -particle.Direction;

            particleList.Add(particle);
```

```
        }

        private void UpdateParticles(GameTime gameTime)
        {
            float now = (float)gameTime.TotalGameTime.TotalMilliseconds;
            for (int i = particleList.Count - 1; i >= 0; i--)
            {
                ParticleData particle = particleList[i];
                float timeAlive = now - particle.BirthTime;

                if (timeAlive > particle.MaxAge)
                {
                    particleList.RemoveAt(i);
                }
                else
                {
                    float relAge = timeAlive / particle.MaxAge;
                    particle.Position = 0.5f * particle.Accelaration * relAge * relAge + particle.Direction *
relAge + particle.OrginalPosition;

                    float invAge = 1.0f - relAge;
                    particle.ModColor = new Color(new Vector4(invAge, invAge, invAge, invAge));

                    Vector2 positionFromCenter = particle.Position - particle.OrginalPosition;
                    float distance = positionFromCenter.Length();
                    particle.Scaling = (50.0f + distance) / 200.0f;

                    particleList[i] = particle;
                }
            }
        }

        private void AddCrater(Color[,] tex, Matrix mat)
        {
            int width = tex.GetLength(0);
            int height = tex.GetLength(1);

            for (int x = 0; x < width; x++)
            {
                for (int y = 0; y < height; y++)
                {
                    if (tex[x, y].R > 10)
                    {
                        Vector2 imagePos = new Vector2(x, y);
                        Vector2 screenPos = Vector2.Transform(imagePos, mat);

                        int screenX = (int)screenPos.X;
                        int screenY = (int)screenPos.Y;

                        if ((screenX) > 0 && (screenX < screenWidth))
                            if (terrainContour[screenX] < screenY)
                                terrainContour[screenX] = screenY;
                    }
                }
            }
        }

        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, 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);
        }

        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);
            }
        }
    }
}
```