الدرس السابع عشر

أهلا بكم في الدرس السابع عشر من سلسلة دروس تعلم ال Xna, في هذا الدرس سوف نتحدث عن موضوع الجزيئات.

بعد أن قمنا بإضافة إكتشاف التصادمات إلى المشروع، نكون قد أنهينا تنفيذ الوظيفة الأساسية. ولكن على كل حال، مازال هناك بعض الاشياء الناقصة، مثل إنفجار جميل في حال صدم الصاروخ الأرض أو الآعبين الآخرين.

قبل عدة سنوات، لم تكن الإنفجارات ثنائية الأبعاد أكثر من رسم متحرك مجهز مسبقا. كملف Gif على سبيل المثال، يتم تضمينه في عدد من الإطارات "Frames" حيث يتم تشغيلهم فيها.

أما اليوم، بما أن قوة المعالجة للكمبيوترات إزدادت بشكل كبير، فإن كلا من الإنفجارات الثنائية و الثلاثية الأبعاد يتم بنائها من العديد من صور كرات نارية صغيرة باهتة، يتم وضعها فوق بعضها البعض. كل هذه الصور تبدأ من مركز الإنفجار، و تتحرك في إتجاهات عشوائية بينما يتم إخفائها تدريجيا مع مرور الوقت. تسمى هذه الصور الصغيرة بالجزيئات "Particles" و يسمى المحرك الذي يقوم بتحديثها بنظام الجزيئات "Particles System".

الجزيئات التي سوف نقوم بإستخدامها كبناء للإنفجار معروضه في الأسفل، و بإستطاعتك تنزيلها من الرابط هنا (خاضعة لرخصة MS Permissive License). قم بإستيرادها إلى المشروع و قم بإضافة المتغير التالي:

```
Texture2D explosionTexture;
```

وقم بتحميله في الدالة LoadContent :

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

لاحظ أن الصورة سوداء، مع وجود كرة نارية في المنتصف. سوف يتم مناقشة ذلك في نهاية الدرس.

قبل أن نتمكن من تصيير الجزيئات، هناك بعض الأشياء التي يجب أن نعرفها عنها. على سبيل المثال، سوف نحتاج أن نخزن المواقع لها، إضافة إلى تخزين معامل التحجيم. كما أننا سوف نحتاج لتخزين الإتجاه من أجل حساب الموقع التالي. و لكن ذلك ليس كل شيئ. سوف نحتاج لتكبير و تلاشي "Fade out" الجزيئات كلما مر الزمن, بمعنى اننا سوف نحتاج أن نخزن متى تم إنشاء الجزيئ، كم من الوقت يجب أن يبقى قبل أن يختفي نهائيا. في النهاية، بما أن الجزيئ يجب أن يتحرك بسرعه كبيرة في البداية و ببطئ في النهاية، سوف نقوم بتخزين التسارع.

لتخزين كل هذه المعلومات بطريقة مرتبة، سوف نقوم بتعريف تركيب "Struct"، بإسم ParticleData، التي سوف تخزن كل هذه البيانات. قم بإضافة الكود التالي في أعلى الكود؛ أسفل فضاء الأسماء "Namespace" :

```
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;
}
```

سوف نحتاج إلى قائمة، قادرة على تخزين كل الجزيئات الفعالة، إذن قم بإضافة المتغير التالي في أعلى الكود.

```
List<ParticleData> particleList = new List<ParticleData>();
```

بعدها، سوف نقوم بإنشاء دالة صغيرة، AddExplosion ، التي سوف تقوم بإنشاء بعض الجزيئات:

```
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);
 }
```

كلما بدئنا إنفجار جديد، هناك عدة اشياء يجب أن نتحكم بها. المنهجية المتبعة في هذه السلسلة من الدروس تعطينا تحكم كامل بالإنفجار: يجب أن نحدد أين يجب أن يبدأ الإنفجار، كم عدد الجزيئات في الإنفجار الواحد، اكبر حجم للإنفجار، و كم من الوقت يجب أن يستمر الإنفجار!

الدالة السابقة تقوم ببساطة بإستدعاء الدالة AddExplosionParticle لعدة مرات، في هذه الدالة يتم فعليا إضافة الجزيئات إلى القائمة:

```
 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;
 }
```

الدالة تبدأ بإنشاء كائن جديد فارغ من النوع ParticleData. هناك أشياء يمكننا تخزينها مباشرة في الكائن. موقع مركز الإنفجار يمكن تخزينه في الخاصية OriginalPosition، كما أن الموقع الحالي (الأول) للجزيئ يجب أن يكون نفسه مركز الأنفجار.

بعدها، بإستطاعتنا تخزين (وقت ولادة) birthtime الجزيئ، الذي سوف يلزمنا في ما بعد لكي نحدد عمر الجزيئ. أعلى عمر للجزيئ سوف يحدد متى يتم إنهاء و إخفاء الجزيئ. خاصية التحجيم Scaling سوف تحدد حجم الجزيئ، خاصية ال ModColor سوف تتيح لنا أن نجعل الجزيئ يختفي في نهاية حياته.

هنالك خاصية مهمة ناقصة: وهي إتجاه الجزيئ. إذا أردت أن تصنع نظام جزيئات حقيقي، هناك شيئ مهم يجب أن تهتم به: وهو العشوائية. بالرغم من أن كل الجزيئات في إنفجار واحد يجب أن تبدأ من نفس الموقع، لكن يجب أن تتحرك هذه الجزيئات في إتجاهات مختلفة. الطريقة الوحيدة للحصول على نتيجة جيده، هي تحديد هذه الإتجاهات بشكل عشوائي.

الكود التالي يقوم بإنشاء إتجاهات عشوائية. ليس الإتجاه فقط عشوائي، ولكن المسافة أيضا:

```
 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;
```

أولا، سوف نبدأ بتحديد كم المسافة التي سوف يسافرها الجزيئ حتى نهاية حياته. نريد قيمة عشوائية، بين 0 و أكبر حجم للإنفجار. يتم الحصول على هذا الرقم في السطر الأول.

سوف نستخدم ذلك كطول الإتجاه، الشيئ الآخر الذي يجب علينا عمله هو تدوير الجزيئ بزاوية عشوائية. يتم ذلك في الأسطر الثلاثة التاليه: الأول نقوم بتعريف متجه لتمثيل الطول الذي تم إنشائه في السطر الأول. بعدها، نقوم بإنشاء زاوية عشوائية، و نقوم بإستخدام هذه الزاوية لإنشاء مصفوفة تدوير، كما تم التوضيح في الدرس الثامن. هذه الدالة تستخدم لتدوير المتجه بزاوية عشوائية.

في هذه اللحظة، نحن نعلم كيف سيتم إزاحة الجزيئ حتى نهاية حياته. حتى الآن، قمنا بإستخدام متجه الإزاحة ذلك ليمثل الإتجاه "Direction" و التسارع، و هذا شيئ خاطئ طبعا كما سوف نرى في الدرس القادم إن شاء الله.

في النهاية, يجب إضافة الجزيئ الجديد إلى قائمة الجزيئات particleList:

```
particleList.Add(particle);
```

هذا كل شيئ في الدالة AddExplosionParticle. تذكر أن هذه الدالة يتم إستدعائها من الدالة AddExplosion، التي سوف يتم إستدعائها في حال إصطدام الصاروخ بأحد اللاعبين أو بالأرضية. هذا يعني أن المكان الأفضل لإستدعاء الدالة AddExplosion هو داخل الدالة CheckCollisions، حيث قمنا بكتابة ثلاث جمل شرطية منفصلة، كل منها مرتبطة بنوع من التصادمات مع الصاروخ.

إذهب إلى الدالة CheckCollisions و إبحث عن جملة الشرط الأولى، و التي يتم تنفيذها في حالة إكتشاف تصادم الصاروخ مع اللاعب. في داخل جملة الشرط تلك، قم بإضافة السطر التالي:

```
AddExplosion(playerCollisionPoint, 10, 80.0f, 2000.0f, gameTime);
```

سوف يؤدي ذلك إلى إضافة 10 جزيئات إلى ال particleList. حجم و مدة الإنفجار لا تهمنا فعليا في هذا الدرس، حيث سوف نقوم بتحديث الجزيئات في الدرس القادم.

في داخل جملة الشرط الثانية، قم بإضافة السطر التالي، الذي سوف يبدأ إنفجار صغير في كل مرة يصطدم الصاروخ بالأرض:

```
AddExplosion(terrainCollisionPoint, 4, 30.0f, 1000.0f, gameTime);
```

في هذا الدرس، سوف نقوم ببساطة برسم الجزيئات على الشاشة. بما أننا لم نحدث مواقعهم بعد, سوف يبقو بشكل ثابت في نفس نقطة التصادم. قم بإضافة الكود التالي في نهاية كود المشروع:

```
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,
```

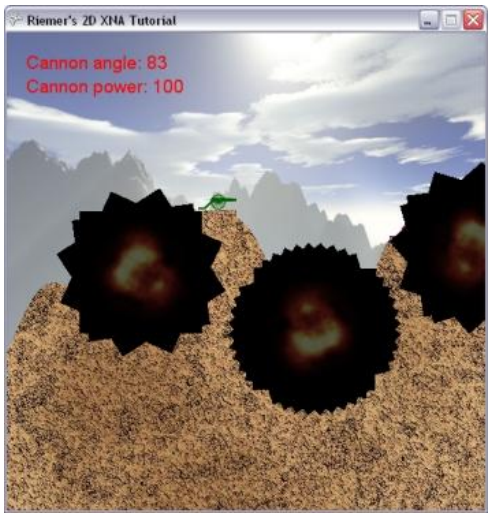
```
SpriteEffects.None, 1);
      }
 }
```

بما أننا لا نقوم بتغير شيئ في داخل القائمة particleList، بإمكاننا إستخدام الصيغة foreach للتكرار. بكل الأحوال أهمية إستخدام التكرار العادي هو وجود المتغير"i" الذي له قيمة مختلفة لكل جزيئ. قمنا بإستخدام المتغير "i" كقيمة لزاوية التدوير للصور، إذن كل جزئي إنفجار سوف يتم رسمه بدوران مختلف. (لاحظ أن 6.28 = 3.14*2=pi*2 و أن هنالك 360 درجة في الدائرة، إذن فإن كل زاوية تقابل 57 درجة => بمعنى أن كل جزيئ سوف يتم رسمه بدوران 57 درجة بالنسبة للجزيئ السابق.)

دعنا لا ننسى إستدعاء هذه الدالة من داخل دالة الرسم Draw، مباشرة قبل جملة SpriteBatch.End ضع السطر التالي:

```
DrawExplosion();
```

الآن قم بتشغيل الكود! في حال إصطدم الصاروخ بأحد اللاعبين أو بالأرضية، يجب أن يتم رسم الجزيئات على الشاشة. لاحظ أن النتيجة تعتمد على كونك إصطدمت بلاعب أو بالتضاريس، و هو كما قمنا بعمله في الدالة CheckCollisions بالضبط.



أنا متفق معك أن النتيجة النهائية ليست في الحقيقة ما نتوقعه. ولكننا بالتأكيد في الطريق الصحيح، وسوف يتم تحسين النتيجة في الدرسين السابقين، إن شاء الله.

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 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.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;
             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();
             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);
            }
        }
    }
}
```