الدرس التاسع عشر

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

في هذه اللحظة، و في حال قام الصاروخ بصدم الأرض أو أحد اللاعبين، يتم إنشاء بعض الجزيئات، و إضافتهم إلى القائمة و من ثم رسمهم على الشاشة بإستخدام مزج ألفا المضاف. ولكن حتى الآن لم يتم تحريك هذه الجزيئات حتى الآن، و هو ما سوف نفعله في هذا الدرس إن شاء الله.

60 مرة في الثانية، يجب أن نقوم بإعادة حساب الموقع الحالي لكل من الجزيئات. و الموقع الحالي لأي كائن يمكن حسابه من خلال 3 خصائص:

- **الموقع الإستهلالي "Initial Position"**: الذي يمثل موقع الجزيئ في بداية حياته.
- **السرعة الإستهلالية** (في المفاهيم الثنائية الأبعاد تسمى الإتجاه "Direction" أو السرعة "Velocity"): هذه السرعة تحدد كمية التغير في الموقع في كل ثانية.
- **التسارع**: التسارع يحدد **كم سوف تتغير السرعة في الثانية.**

جيد جدا، دعنا نمر على هذه القائمة مرة أخرى و نطبقها على الجزيئات الخاصة بالإنفجار:

- الموقع الإستهلالي (الإبتدائي): يتمثل بمركز الإنفجار الخاص بنا، و هو مكان حدوث التصادم على الشاشة.
- السرعة الإستهلالية (الإتجاه) : في الإنفجار يجب أن تكون السرعه في البداية عالية. في الألعاب ثنائية الأبعاد، بما أن الإتجاه يتكون من مركبتين X و Y ، إذن السرعة أيضا تتكون من مركبتين X و Y.
- التسارع: بما أن الجزيئ يبدأ بسرعة عالية، سوف نحتاج لأن نبطئهم مع مرور الوقت بحيث تصبح السرعة 0 في نهاية حياة الجزيئ. كما تلاحظ و كما سيتم شرحة لاحقا، هذا يعني أن إتجاه السرعة يجب أن يكون معكوس.

كما قلت سابقا، بإستطاعتك أن تجد الموقع الحالي بناء على الموقع الإستهلالي، السرعة الإستهلالية و التسارع. ذلك يتم بإستخدام المعادلة التالية:

$$pos_{current} = \frac{1}{2} * acceleration * time^2 + speed_{initial} * time + pos_{initial}$$

سوف نقوم بكتابة دالة جديدة، UpdateParticles، التي تقوم بالمرور على قائمة الجزيئات و تقوم بتحديث مواقعهم. دعنا نبدأ بالكود التالي:

```
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
        {
            //update current particle
        }
    }
}
```

هذه الدالة تقوم أولا بتخزين وقت اللعبة في المتغير now. بعدها، تمر على كل الجزيئات و تقوم بحساب العمر الحالي لكل جزيئ. إذا كان عمر الجزيئ أكبر من الحد الأعلى للأعمار، يتم حذفه من القائمة.

لاحظ أنه يتم المرور على القائمة بشكل عكسي. يفيدنا ذلك في حذف العناصرمن القائمة: لأنه إذا تم حذف العنصر i من القائمة، يتم تحريك مكان كل العناصر الأخرى التالية ل i في القائمة درجة واحدة للأمام. بحيث تصبح i+1 في الموقع i و تصبح i+2 في الموقع i+1. الآن، إذا قمنا بتحريك العناصر للأمام، سوف يتم في الدورة التالية للتكرار فحص العنصر i+1 مما يعني أننا تجاوزنا عن أحد العناصر.

دعنا الآن نستبدل السطر الخاص بالملاحظة "Update Current Particle" ببعض الكود الفعلي. في حالة لم يكن عمر الجزيئ تجاوز الحد (شباب يعني ☺)، يجب أن نقوم بتحديث الموقع. بإستخدام المعادلة السابقة، ذلك سهل جدا. ولكن في الحالة التي نتعامل فيها مع الوقت من المفروض أن يتم تحجيم الوقت ليصبح قيمة بين ال 0 و ال 1، بحيث أن ال 0 تعني 'البداية' و ال 1 تعني ال 'النهاية'. هذا سوف يسهل العديد من الحسابات كما سوف تلاحظ في نهاية الدرس. سوف نسمي قيمة الوقت تلك (بين ال 0 و ال 1) سوف نسميها بـ 'العمر النسبي'، relAge بشكل مختصر. ويمكن حسابة بسهولة، بالشكل التالي:

```
float relAge = timeAlive / particle.MaxAge;
```

بإمكانك التأكد بنفسك أن هذه القيمة تساوي 0 عندما يكون ال timeAlive = 0 ، و تكون 1 عندما تكون ال timeAlive = MaxAge .

سوف نقوم بإستخدام هذه القيمة بإعتبارها الزمن في المعادلة السابقة، و التي تصبح:

```
particle.Position = 0.5f * particle.Accelaration * relAge * relAge +
particle.Direction * relAge + particle.OrginalPosition;
```

بإمكانك أيضا التأكد بنفسك ان هذه هي نفس المعادلة السابقة الذكر. هذا الكود سوف يقوم بتحديث الكود بشكل صحيح لكل جزيئ من الجزيئات، في كل مرة يتم فيها إستدعاء هذه الدالة. ولكن هذا ليس كل شيئ نحتاج لتغيره: سوف نقوم أيضا بتحديث الشفافية و التحجيم للصورة. كلما إقترب الجزيئ من نهايته سوف نقوم بالتقليل من ظهور الجزيئ، و نقوم بتكبيره أيضا. بحيث عندما نقوم بعمل التأثيريين معا، سوف يبدو أن الجزيئ يتلاشى بشكل سلس.

بإمكاننا عمل تلاشي الجزيئ من خلال تقليل قوة لون الجزيئ. تذكر أن الألوان الخاصة بالجزيئات يتم إضافتها إلى بعضها البعض و إضافتها إلى المشهد. الآن، قبل أن نقوم بعمل ذلك، سوف نقوم بتقليل قيم الألوان لهذه الجزيئات، عن طريق ضربها برقم أقل من 1، اي بين ال 1 و ال 0.

تذكر في واحد من الدروس الأولى أن اللون يتم ضربه باللون المعدل "modulation color". و هو بالضبط ما نحن بصدد إستخدامه: في بداية الجزيئ، سوف نقوم بضرب ألوانه باللون المعدل الأبيض (1,1,1,1). في منتصف حياتها سوف نضرب ألوانه باللون (0.5,0.5,0.5,0.5) وفي نهاية حياته سوف نضربه باللون (0,0,0,0) و هو ما يؤدي لأن يختفي تأثير الجزيئ تماما عن الشاشة.

من أجل ذلك، سوف نحتاج أن نجد قيمة بين ال 0 و ال 1. و ذلك سهل بما أنه يوجد لدينا قيمة الوقت بين ال 0 و ال 1:

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

تأكد في الدالة DrawExplosion أن كل جزيئ يستخدم هذا اللون بإعتباره اللون التعديلي:

```
spriteBatch.Draw(explosionTexture, particle.Position, null, particle.ModColor,
i, new Vector2(256, 256), particle.Scaling, SpriteEffects.None, 1);
```

بعدها، يجب أن نتأكد أن الجزيئ يكبر حجمه كلما إقترب من نهاية عمره. سوف نجعل حجمه يعتمد على بعده عن نقطة المركز: كلما إبتعد عن المركز، كلما إزداد حجم الجزيئ.

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

السطرين الأولين يحسبان المسافة بين الموقع الحالي و الموقع الأصلي للجزيئ.

هذه المسافة تستخدم في تحديد حجم الجزيئ. الإزاحة بقيمة 50.0f مطلوبة، و إلا سوف يكون الحجم 0 في البداية، لأن المسافة تكون 0. القيم 50 و 200 تم إيجادهما بالتجربة. عندما تكون المسافة 150 سوف يكون التحجيم 1.

هذا كل شيئ في منطق الدالة UpdateParticles! في كل مرة يتم إستدعاء هذه الدالة، سوف يتم تحديث الموقع و الحجم و اللون التعديلي للجزيئ. ما زلنا بحاجة إلى حفظ الجزيئ المحدث إلى قائمة الجزيئات:

```
particleList[i] = particle;
```

دعنا أيضا نستدعي هذه الدالة من داخل دالة التحديث Update، بحيث يتم تحديث الجزيئات 60 مرة في الثانية تماما:

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

و كتعديل صغير، دعنا لا نقرأ شيئا من لوحة المفاتيح إذا كان الصاروخ طائرا أو إذا كان هناك إنفجار فعال. هذا سوف يتيح لنا التأكد أن اللاعب لن يغير طاقة أو زاوية المدفع الخاص به بعد قيامه بإطلاق الصاروخ. قم بتعديل السطر في الدالة Update:

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

عندما تقوم بتشغيل هذا الكود، ستلاحظ أن الإنفجارات قد جهزت!

في كل الأحوال، عندما تلقي نظرة فاحصة عليها (خصوصا الإنفجارات الكبيرة) سوف تلاحظ أن هنالك شيئا ليس صحيحا 100%: بالنسبة لنهاية الإنفجار، الجزيئات تصبح اسرع فاسرع! على نحو أسوأ، المواقع النهائية لها لا تحترم موقع الإنفجار الذي حددته.

ذلك بسبب الدالة AddExplosionParticle، قمنا بإختيار الإتجاه و التسارع للجزيئ بدون التفكير بهما. سوف نحتاج لنعيد التفكير مرة أخرى، وسوف يكون كل شيئ على ما يرام.

هناك محددين إثنين يساعداننا على إيجاد القيم الصحيحة. في نهاية الإنفجارات (عندما يكون العمر النسبيrelAge=1)، نريد:

- السرعة النهائية تساوي 0 تماما
- الموقع النهائي يجب أن يكون موقع البداية + متجه الإزاحة "displacement" ويتم حسابهم داخل الدالة AddExplosionParticle.

دعنا نضع ذلك في بعض المعادلات. بما أن التسارع يحدد التغير في السرعه بمرور الوقت، و ال relAge=1 في نهاية الجزيئ، بإستطاعتنا كتابة المحدد الأول كالتالي:

$$speed_{final} = 0 = speed_{initial} + acceleration * 1$$

أو

$$acceleration = -speed_{initial}$$

ما يعني أنه يجب علينا فقط إيجاد السرعة الإبتدائية. دعنا نكتب المحدد الثاني في معادلة:

$$pos_{initial} + displacement = \frac{1}{2} * acceleration * 1^2 + speed_{initial} * 1 + pos_{initial}$$

بإمكاننا شطب ال $pos_{Initial}$ من الجهتين، و إستبدال التسارع بالمحدد الأول:

$$displacement = \frac{1}{2} * (-speed_{initial}) + speed_{initial} = \frac{1}{2} speed_{initial}$$

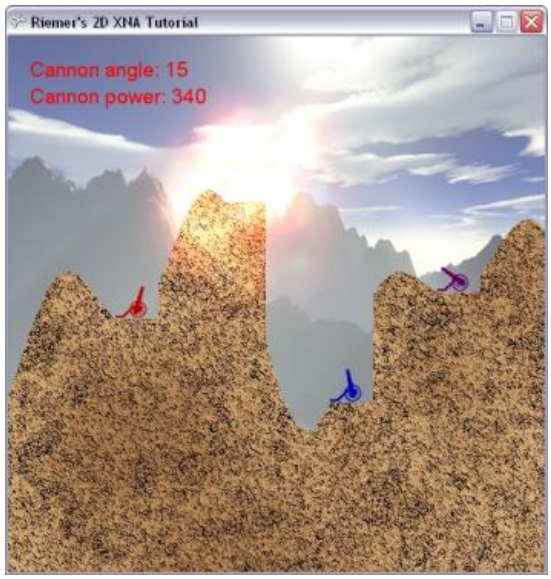
أو

$$acceleration = -speed_{initial}$$

هذا كل ما نحتاج لمعرفته! إذهب إلى الدالة AddExplosionParticle، و استبدل السطور التالية بحيث تعكس ما قمنا للتو بحسابه:

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

الآن عندما تقوم بتنفيذ الكود، سوف تلاحظ أن الإنفجارات تنمو بشكل أكبر فأكبر كما حددننا في المتغير explosionSize، و يبطئ حتى تصل سرعته إلى 0 في نهاية حياته!



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;

       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();

            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);
        }

        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;
                }
            }
        }

        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);
            }
        }
    }
}
```