الدرس الأخير

أهلا بكم في الدرس الثاني و العشرين من سلسلة دروس تعلم ال Xna, في هذا الدرس سوف نتحدث عن خيارات الدقة (أبعاد الشاشة) في اللعبة.

بعد تجهيز كل هذه الميزات في لعبتنا الخاصة، هناك شيئ واحد إضافي مهم يجب أن نبقيه في بالنا: كيف سوف تبدو اللعبة في حال قمنا بتشغيلها على جهاز آخر. عندما تقوم بتشغيل اللعبة في شاشة ثابتة الحجم، لن يؤدي ذلك إلى مشاكل. ولكن في حال كانت أبعاد شاشة اللعبة تعتمد على أبعاد شاشة جهاز اللاعب، سوف تحتاج لأن تقوم ببعض التغييرات على الكود. وإلا، إذا قمت بتشغيل اللعبة في شاشة أكبر، سوف تملأ الشاشة المنطقة العلوية اليسرى من الشاشةفقط.

عندما نتعامل مع أبعاد متغيرة للشاشة، هناك ببساطة حلين إثنين:

- بإستطاعتك أن تعيد تحجيم كل العناصر و الصور الموجودة في اللعبة لكي تتوافق مع الأبعاد الحالية للشاشة. في حال كانت الأبعاد أكبر، قد يؤدي ذلك إلى أن تبدو الصور بشكل أبشع قليلا لأنها قد تعرضت (للمط).
- بإمكانك أن تستغل جميع البكسلات الموجودة. في حالة اللعبة الخاصة بنا، سوف يؤدي ذلك إلى وجود تضاريس اكبر، و إلى مسافة أكبر بين اللاعبين مما سيصعب أن يصيب كل منهم الآخر.

بما أن كل من الطريقتين لها سلبياتها و إيجابياتها، دعنا نناقش كلا منهما. سوف نجعل من السهل التحويل بين كل منهما، بإستخدام متغير منطقي عام، الذي يجب أن نضيفه إلى قائمة المتغيرات الخاصة بنا، إضافة إلى الحجم الأصلي للشاشة.

```
const bool resultionIndependent = false;
Vector2 baseScreenSize = new Vector2(800, 600);
```

عندما يكون المتغير resultionIndependent له القيمة true، سوف يتم إتباع المنهجية الأولى: سوف نقوم بتحجيم كامل اللعبة من (600,800) ليتلائم مع أبعاد الشاشة الحالية. لاحظ أن ذلك قد يؤدي إلى تغيير نسبة الطول إلى العرض: إذا كانت أبعاد شاشة المستخدم (800,800)، سوف يتم شد (مط) اللعبة بشكل عمودي. أما إذا كان المتغير resultionIndependent له القيمة false فسوف يتم إتباع المنهجية الثانية.

دعنا أولا نقوم بعمل تعديل بسيط في دالة ال LoadContent. هنا نقوم بإعطاء قيم للمتغيرين screenWidth و screenHeight، الذين يحددان حجم الشاشة التي يستخدمها الكود. عندما نقوم بعمل التحجيم سوف نحتاج لأن نعرف حجم الشاشة الأصلي، حيث سوف يتم مط المشهد الناتج في داخل الدالة Draw .

```
if (resultionIndependent)
{
     screenWidth = (int)baseScreenSize.X;
     screenHeight = (int)baseScreenSize.Y;
}
else
{
     screenWidth = device.PresentationParameters.BackBufferWidth;
     screenHeight = device.PresentationParameters.BackBufferHeight;
}
```

عندما يكون المتغير resultionIndependent له القيمة false، سوف تجري الامور بالطريقة القديمة: سوف يقوم الكود بإنشاء الخامات بنفس الحجم للأبعاد الأصلية. سوف يؤدي ذلك إلى وجود تضاريس اكبر, بالإمكان رسمها على الشاشة.

التغيرات الأخيره التي يلزمنا أن نقوم بها هي في الدالة Draw. في حال كان المتغير resultionIndependent له القيمة true، سوف نحتاج للقيام بتحجيم كامل للمشهد بحيث يطابق حجم الشاشة تماما. إذن، سوف نقوم أولا بحساب قيمة التحجيم

التي نحتاجها للمشهد. بما أن التحجيم العمودي و التحجيم الأفقي بالإمكان أن يكونو مختلفين، سوف نحتاج لإيجاد قيمتين إثنتين. بدلا من تخزينهم في متغير من النوع Vector2 سوف نقوم بتخزينهم في Vector3، حيث سوف يتم إعطاء المركب الثالث القيمة 1، بما أننا سوف نحتاج فيما بعد ل Vector3 و ليس Vector2.

ضع هذا الكود في أعلى الدالة Draw :

```
 Vector3 screenScalingFactor;
 if (resultionIndependent)
 {
     float horScaling = (float)device.PresentationParameters.BackBufferWidth /
baseScreenSize.X;
     float verScaling = (float)device.PresentationParameters.BackBufferHeight /
baseScreenSize.Y;
     screenScalingFactor = new Vector3(horScaling, verScaling, 1);
 }
 else
 {
     screenScalingFactor = new Vector3(1, 1, 1);
 }
```

في حال كان المتغير resultionIndependent له القيمة true، سوف نحتاج لإيجاد معامل التحجيم. معامل التحجيم الأفقي يتم إيجاده عن طريق قسمة العرض الحقيقي لشاشة المستخدم، على الأبعاد الخاصة باللعبة. تأكد من ذلك بنفسك، إذا كانت الأبعاد لها نفس قيمة أبعاد اللعبة الأصلية، سوف يكون معامل التحجيم 1. إذا كانت الأبعاد الحالية للعبة أكبر، سوف يكون معامل التحجيم أكبر من 1، ما يعني أنه سوف يتم مط المشهد. إذا كانت أبعاد الشاشة الحالية اكبر من أبعاد مشهد اللعبة، سوف يكون معامل التحجيم أقل من 1، إذن سوف يتم تصغير المشهد.

أما إذا كان المتغير resultionIndependent له القيمة false، يجب أن يتم رسم المشهد على الشاشة كما هو، حيث سوف يكون معامل التحجيم 1، للأفقي و العمودي.

بعدها، سوف نحتاج لطريقة لتحجيم المشهد كاملا. بإمكاننا تعديل معاملات التحجيم و المواقع لكل إستدعاءات SpriteBatch.Draw، ولكن هناك طريقة أسهل بكثير: الدالة SpriteBatch.Begin تتيح لنا أن نعمل تحويل عام. هذا التحويل العام الذي سوف نستخدمه يمثل معامل التحجيم الذي قمنا بحسابة، ولكن بإمكانك إستخدامه بسهولة من أجل عمل تدوير للمشهد كاملا، مثلا في حال كنت تريد ضبط اللعبة على جهاز ال Zune.

هذا التحويل العام سوف يتم وصفه، مثل أي تحويل، بإستخدام مصفوفة. قم بإضافة الكود التالي إلى الدالة Draw:

```
 Matrix globalTransformation = Matrix.CreateScale(screenScalingFactor);
```

لإضافة هذا التحويل في الدالة SpriteBatch.Begin، سوف نستخدم الصورة الأكثر تعقيدا للدالة، الصورة التي تستقبل 4 وسائط. قم بإستبدال الإستدعاء الأول للدالة SpriteBatch.Begin بهذا السطر:

```
 spriteBatch.Begin(SpriteBlendMode.AlphaBlend, SpriteSortMode.Immediate,
SaveStateMode.None, globalTransformation);
```

هذا السطر يتيح مزج ألفا بسيط وليس المضاف، و يحدد التحجيم بإستخدام مصفوفة التحويل العامة. لن أتعرض للوسيط الثاني و الثالث هنا.

في المرة الثانيه التي يتم فيها إستخدام ال SpriteBatch، نريده أن يستخدم مزج ألفا المضافة، إذن قم بتغيير السطر إلى التالي:

```
 spriteBatch.Begin(SpriteBlendMode.Additive, SpriteSortMode.Deferred,
SaveStateMode.None, globalTransformation);
```

هذا كل شيئ الآن حاول تغير القيم الخاصة بالمتغير baseScreenSize، إضافة إلى تغيير حجم الشاشة. بإمكانك أيضا أن تعطي المتغير graphics.IsFullScreen القيمة true. قم بتغيير قيم المتغير resultionIndependent بين ال true و ال false، ولاحظ الفرق!

بهذا ننهي سلسلة الدروس الخاصة ببرمجة الألعاب ثنائية الأبعاد في ال Xna. أرجو أن تكون قد تمتعت بها كما تمتعت أنا بكتابتها، و أنك قد تعلمت شيئا جديدا.

لديك الحرية الكاملة لعمل أي تعديل\ زيادة على الكود، بما أن هذه هي الطريقة الأفضل للتعلم كيف تعمل الأشياء. بعد ذلك، يجب أن تكون جاهزا اكثر لتبدأ ببرمجة لعبة ثنائية الأبعاد خاصة بك!

إذا شعرت بأنك قد أتقنت غالبية المفاهيم و الوظائف المستعرضة في هذه السلسلة، أنا أنصحك بشدة لكي تلقي نظرة على السلسلة الأولى من سلسلة دروس برمجة الألعاب ثلاثية الأبعاد بإستخدام Xna!

تم بحمد الله

المترجم:

في نهاية ترجمتي لهذه السلسلة أرجو أن أكون قد وفقت في هذه العملية و أن اكون قد أفدت البعض على تعلم شيئ جديد في عالم برمجة الألعاب.

طبعا النسخة المترجمة من السلسلة مفتوحة للجميع و بإمكان أي أحد أن ينقلها و يوزعها في أي مكان أو موقع، ولكن بشرط ذكر المترجم (خلدون خالد- الفريق العربي للبرمجة) ولا مانع من وضع رابط ☺

بالنسبة للسلاسل التعليمية الخاصة ببرمجة الألعاب ثلاثية الأبعاد سوف أحاول أن ابدأ بترجمتها في الأيام القادمة (ليس وعدا ☺) ولكن حسب وقت الفراغ إن شاء الله.

و السلام عليكم ورحمة الله و بركاته.

الكود النهائي:

```
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;

       AudioEngine audioEngine;
       WaveBank waveBank;
       SoundBank soundBank;


        const bool resultionIndependent = false;
        Vector2 baseScreenSize = new Vector2(800, 600);

        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);

            if (resultionIndependent)
            {
                screenWidth = (int)baseScreenSize.X;
                screenHeight = (int)baseScreenSize.Y;
            }
            else
            {
                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);

            audioEngine = new AudioEngine("Content/xactProject.xgs");
            waveBank = new WaveBank(audioEngine, "Content/myWaveBank.xwb");
            soundBank = new SoundBank(audioEngine, "Content/mySoundBank.xsb");
        }

        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);

            audioEngine.Update();

            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);
            soundBank.PlayCue("hitcannon");
        }

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

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

        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);


             Vector3 screenScalingFactor;
             if (resultionIndependent)
             {
                 float horScaling = (float)device.PresentationParameters.BackBufferWidth / baseScreenSize.X;
                 float verScaling = (float)device.PresentationParameters.BackBufferHeight / baseScreenSize.Y;
                 screenScalingFactor = new Vector3(horScaling, verScaling, 1);
             }
             else
             {
                 screenScalingFactor = new Vector3(1, 1, 1);
             }

             Matrix globalTransformation = Matrix.CreateScale(screenScalingFactor);

             spriteBatch.Begin(SpriteBlendMode.AlphaBlend, SpriteSortMode.Immediate, SaveStateMode.None,
globalTransformation);
             DrawScenery();
             DrawPlayers();
             DrawText();
             DrawRocket();
             DrawSmoke();
             spriteBatch.End();

             spriteBatch.Begin(SpriteBlendMode.Additive, SpriteSortMode.Deferred, SaveStateMode.None,
globalTransformation);
             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);
            }
        }
    }
}
```