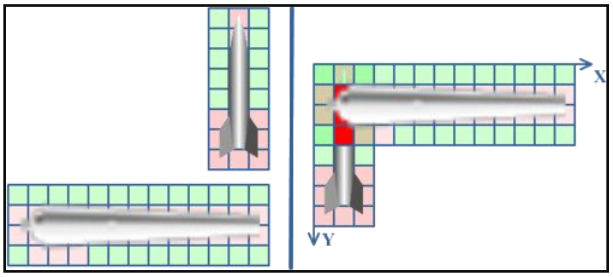
الدرس الرابع عشر

أهلا بكم في الدرس الرابع عشر من سلسلة دروس تعلم ال Xna, في هذا الدرس سوف أقوم بالشرح عن موضوع إكتشاف التصادمات "Collision Detection".

بما أن الصاروخ الآن طائر في الشاشة, و كل نقطة من التضاريس معروفة في الكود, نحن جاهزين لأكثر المواضيع تحديا في برمجة الألعاب, كما انه من المواضع الشيقة في هذه السلسلة: وهو إكتشاف التصادمات.

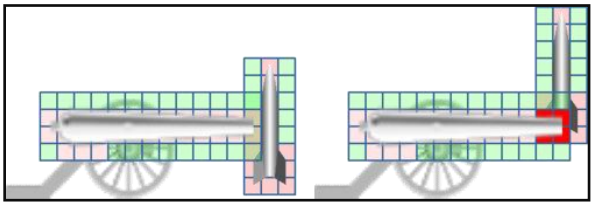
الفكرة العامه عن إكتشاف التصادمات بين صورتين هي بسيطة جدا: بحيث لكل بكسل في الصورة الأولى, نفحص فيما إذا كانت تتصادم مع أي بكسل في الصورة الثانية.

تذكر دائما أن الصورة عبارة عن مستطيل. ولكن في الحقيقة القليل من العناصر لها أشكال مستطيلة، لان العديد من الصور تحتوي على العديد من المناطق الشفافة. هذا ما هو واضح في الصورة التاليه على اليسار. لاحظ أن المناطق الشفافة تم رسمها باللون الأخضر، و المناطق الغير شفافة باللون الأحمر الفاتح (لا تسميها وردي). سوف نعود لهذا لاحقا.



قبل أن نستطيع فحص التصادم بين صورتين، سوف نحتاج لأن نتأكد أنهما في موقعهما الصحيح؛ و أنه قد تم تحريكهما إلى الموقع الصحيح على الشاشة. إذا لم نقم بوضعهم في المكان الصحيح قبل تفحص التصادمات، سوف يتم محاذاة البكسلات العلوية- اليسرى للصورتين و سوف يكون لدينا تصادم. وهذا ما يظهر في الصورة السابقة اليمنى.

دعنا نفترض أن الصورتين في موقعهما الصحيح إذن سوف يكون الصاروخ أمام المدفع, كما يظهر في الصورة اليسرى التالية:



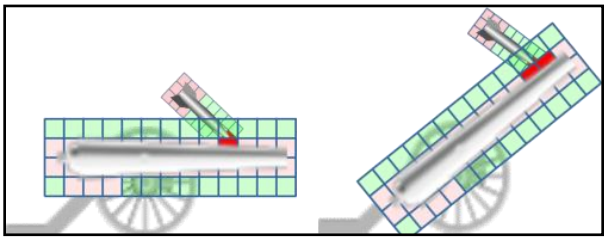
رغم أن الصورتين في الجزء الأيسر من الصورة السابقه تتقاطعان, لكننا نرى أن الصاروخ و المدفع عمليا لا يتقاطعان. هذا بسبب أن البكسلات المتقاطعه في الصورتين هناك واحده منهم على الأقل شفافة.

عند مقارنة بكسلين متصادمين، إذا كان واحد منهم شفاف، لا يكون هناك تصادم بين الكائنات الفعلية. فقط عندما يكون كلا من البكسلين في الصورتين الأولى و الثانية غير شفافين، يكون هنالك تصادم حقيقي. وهذا ما يوضحه الجدول التالي:

Pixel Image 1 Transparent?	Pixel Image 2 Transparent?	Collision ?
Yes	Yes	No
No	Yes	No
Yes	No	No
No	No	Yes

مثال على التصادم الحقيقي معروض على الجزء الأيمن من الصورة السابقة. كما تلاحظ أن البكسلات الأربعة الأخيرة من صورة المدفع غير شفافات, كما أنهم يتصادمون مع أربع بكسلات غير شفافة من صورة الصاروخ.

هاذا على أية حال، ليس كل ما سنقولة عن إكتشاف التصادم في الأبعاد الثنائية. في الصورة السابقة, لم يتم تحجيم أو تدوير أي من الصور الثنائية الأبعاد. عندما يتصادم الصاروخ مع المدفع، سوف يتم تصغيره و تدويره. هذه الحاله تظهر في الصورة التاليه على اليسار.



الجزء الأيمن من هذه الصورة يعرض أكثر الحالات تعقيدا: بحيث أن كلا من الصورتين قد تم تحجيمهم أو تدويرهم. سوف نواجه هذه الحاله في لعبتنا.

السؤال الحقيقي في هذا الدرس هو كيف يمكننا أن نكتشف التصادم في حالات مماثلة. هذه الخوارزمية لكيفية قيامنا بذلك:

```
For each pixel in image A:
{
    Find the screen coordinate occupied by this pixel
    Find the pixel in image B that is also rendered to this position
    If both pixels are non-transparent
    {
        COLLISION!
    }
}
```

هذا موضح في الصورة السابقة. حيث تعرض هذه الصورة مدفع قد تم عليه تدوير، و صاروخ قد تم عليه تدوير إضافة إلى التحجيم. قمت بإضافة الصور الأصلية في الركن الأيسر العلوي كمرجع.

جميل جدا، ولكن ' كيف يمكننا إيجاد الإحداثيات على الشاشة المرسوم عليها هذه البكسلات؟ ' ذلك يتم بإستخدام المصفوفات. تذكر، أن المصفوفات ليست شيئا سحريا، ولكن في الحقيقة هي شبكة بحجم 4 * 4 من الأرقام. ولكن كل ما أريد أن تعرفه عن المصفوفات في هذه النقطة: أن المصفوفة يمكنها تخزين الموقع، الدوران، و التحجيم لصورة معينة.

عندما يتم تخزين الموقع، التحجم و الدوران في المصفوفة، يمكنك إستخدامها لإيجاد الإحداثيات النهائية لأي بكسل على الشاشة. كما أنه من المهم في هذه الحالة أيضا: إذا قمنا بأخذ المعكوس لهذه المصفوفة، و لدينا أي إحداثي سيكون بإستطاعتنا إيجاد الإحداثي في الصورة الأصلية.

الآن سوف نمر على الخوارزمية السابقة. بدلا من تطبيقها على كل البكسلات في الصورة A , نستطيع أن نناقش ذلك لبكسل واحد؛ إفترض أنه البكسل الأسفل الأيمن (3,8) للصورة. كخطوة أولى, نريد أن نجد على أي من إحداثيات الشاشة سوف يتم رسم هذا البكسل. نستطيع إيجاد ذلك من خلال ضرب (= تحويل "Transforming") إحداثيات البكسل (3,8) بالمصفوفة الخاصة بالصورة. هذا سوف يعطينا الإحداثيات التي سوف يرسم عليها البكسل على الشاشة !

الخطوة التالية هي العكس: لدينا إحداثيات الشاشة، ونريد أن نعرف أي بكسل من من الصورة B مرسوم في هذه الإحداثيات. ذلك يتم من خلال تحويل إحداثيات الشاشة عن طريق عكس "inverse" المصفوفة الخاصة بالصورة B. هذا سوف يعطينا الإحداثيات في الصورة الأصلية B , التي هي (13,2) في هذه الحالة.

بعدها, نرى أن كلا من البكسلين في الصورتين A و B غير شفافين, إذن نكتشف أن الصورتين تتصادمان معا على الشاشة!

دعنا الآن ننفذ ذلك بإستخدام بعض الكود. سوف نقوم بإنشاء دالة، TexturesCollide، التي سوف تقرر فيما إذا ما كانت الصورتين متصادمات. الدالة سوف تحتاج اربع أشياء لكي تعمل بشكل صحيح:

- مصفوفة ثنائية البعد تحتوي ألوان الصورة الأولى
- مصفوفة التحويل الخاصة بالصورة الأولى
- مصفوفة ثنائية البعد تحتوي ألوان الصورة الثانية
- مصفوفة التحويل الخاصة بالصورة الثانية

إذن يمكننا الآن كتابة الكود التالي:

```
 private Vector2 TexturesCollide(Color[,] tex1, Matrix mat1, Color[,] tex2,
Matrix mat2)
 {
     return new Vector2(-1, -1);
 }
```

الدالة لن تحدد فقط فيما إذا كان هناك تصادم، ولكن إذا كان هناك تصادم سوف تقوم بإرجاع الموقع على الشاشة الذي تم فيه ذلك التصادم. إذا لم يكن هنالك تصادم سوف ترجع الإحداثي (-1,-1).

دعنا نترجم "لكل بكسل في الصورة A" إلى كود ال C# ، قم بإضافة الكود التالي إلى أعلى الدالة:

```
 int width1 = tex1.GetLength(0);
 int height1 = tex1.GetLength(1);
 for (int x1 = 0; x1 < width1; x1++)
 {
     for (int y1 = 0; y1 < height1; y1++)
     {
     }
 }
```

لكل بكسل في الصورة الأولى, سوف نقوم أولا بتحديد الإحداثي المقابل على الشاشة. ذلك يتم عن طريق تحويل الإحداثي X,Y الأصلي بالمصفوفة الخاصة بالصورة الأولى، إذن قم بإضافة الكود التالي في التكرار الداخلي:

```
 Vector2 pos1 = new Vector2(x1,y1);
 Vector2 screenCoord = Vector2.Transform(pos1, mat1);
```

الآن لدينا الإحداثي على الشاشة للبكسل الحالي، دعنا نقوم بإيجاد البكسل المرتبط بهذا الإحداثي في الصورة الثانية الأصلية. بطبيعة الحال هذا عكس ما فعلناه قبل ثانيتين ☺، إذن الآن نحن نحتاج أن نقوم بتحويل إحداثي الشاشة عن طريق عكس المصفوفة الخاصة بالصورة الثانية:

```
 Matrix inverseMat2 = Matrix.Invert(mat2);
 Vector2 pos2 = Vector2.Transform(screenCoord, inverseMat2);
```

حسناً، في هذه اللحظة لدينا موقعين في الصورتين الأصليتين، كما أننا نعلم أنهما مرسومان في نفس الموقع على الشاشة. كل ما يلزمنا لعمله هو أن نفحص إذا ما كان كل من البكسلين غير شفافين:

```
if (tex1[x1, y1].A > 0)
 {
     if (tex2[x2, y2].A > 0)
     {
         return screenCoord;
     }
 }
```

الذي يفحص فيما إذا كان مكون الألفا "Alpha" لكل من الصورتين أكبر من صفر. إذا كان البكسلين غير شفافين إذن نقوم بإرجاع إحداثي الشاشة (سنقوم بتعريف x2,y2 لاحقا).

الآن في حالة عدم إكتشاف تصادم، سوف نقوم بإرجاع (-1,-1) إلى الكود المستدعي. بعدها، سوف نجعل برنامجنا يفسر الإحداثي (-1,-1) أنه لم يتم إكتشاف تصادم. إذن ضع الكود التالي في نهاية الدالة السابقة إذا لم يكن موجود أصلا:

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

هذا يفي بالوظيفة الأساسية للدالة. في حين أنه ربما يكون هناك حالات خطيره في هذه الدالة، بشكل عام قد تؤدي إلى إنهيار البرنامج، لأن بعض البكسلات في الصورة A قد تقع خارج الصورة B. في الصور السابقة على سبيل المثال: البكسل الأول في صورة الصاروخ (في المنطقة العلوية اليسرى في الصاروخ) لا يقابله اي بكسل في الصورة B, إذن السطر التالي سوف يؤدي إلى إنهيار البرنامج عند ذلك البكسل:

```
if (tex2[x2, y2].A > 0)
```

هذه المشكلة سهلة الحل, من خلال تفحص إذا ما كانت الإحداثيات الخاصة بالصورة A لا تقع خارج الصورة B، اضف الكود في التكرار الداخلي:

```
int width2 = tex2.GetLength(0);
int height2 = tex2.GetLength(1);
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)
            {
                return screenCoord;
            }
        }
    }
}
```

من المفروض أن يعمل ذلك بشكل جيد. سوف نقوم بإنهاء الدرس بتنظيف الدالة قليلا, كما أنه يوجد بعض الأشياء التي يمكننا أن نقوم بتحسينها. لنبدأ بـ ، كلا من المتغيرات width2 و ال height2 داخل جملة التكرار الداخلية لا يتغيران أبدا، إذن يجب أن نقوم بوضعهم قبل بداية التكرار و حسابهم مرة واحده فقط.

هناك تحسين آخر يمكننا فعله. في الدالة، لكل بكسل في الصورة A قمنا بعمل تحويلين "Transformation": من الصورة A إلى إحداثيات الشاشة، و من إحداثيات الشاشه إلى الصورة B. و لكن من الأفضل إسبدالهم بتحويل واحد من الصورة الأولى إلى الثانية.

بإستخدام المصفوفات، ذلك سهل. حاليا نحن نقوم بالتحويل بإستخدام mat1، وبعدها بإستخدام معكوس (mat2). ولكننا بإستطاعتنا الحصول على المصفوفة التي تجمع كلا المصفوفتين السابقتين، بكل بساطة عن طريق ضربهما معا. ضع الكود التالي في أعلى الدالة السابقة:

```
Matrix mat1to2 = mat1 * Matrix.Invert(mat2);
```

بما أن المصفوفة mat1to2 هي ناتج دمج المصفوفتين mat1 و معكوس mat2، إذن تحويل الإحداثيات من الصورة الأولى بإستخدام هذه المصفوفة سوف يعطينا مباشرة الإحداثيات داخل الصورة الثانية!

هذا يتيح لنا التحويل من pos1 إلى pos2 بإستخدام مصفوفة تحويل واحدة! هذا واضح في الأسفل حيث قمت بعرض الكود النهائي الكامل للدالة TexturesCollide :

```
 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);
 }
```

لاحظ أيضا أننا لم نقم بإستعمال إحداثيات الشاشة أبدا. كذلك, عندما يتم إكتشاف تصادم، سوف نحتاج إلى أن نحول الإحداثي الخاص بالصورة الأولى بإستخدام mat1 من أجل الحصول على إحداثيات الشاشة، بحيث يمكن إرجاعها للكود المستدعي.

لا يوجد صورة للمشروع في هذا الدرس، بالرغم من أننا قمنا بإضافة بعض الكود إلى المشروع. في الدرس القادم سوف نرى كيف يمكننا الإستفاده من وظيفة إكتشاف التصادم التي قمنا بإضافتها، لفحص التصادم بين أي صورتين.

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 PlayerData
    {
        public Vector2 Position;
        public bool IsAlive;
        public Color Color;
        public float Angle;
        public float Power;
    }

    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;
        GraphicsDevice device;

        int screenWidth;
        int screenHeight;

        Texture2D backgroundTexture;
        Texture2D foregroundTexture;
        Texture2D carriageTexture;
        Texture2D cannonTexture;
        Texture2D rocketTexture;
        Texture2D smokeTexture;
        Texture2D groundTexture;
        SpriteFont font;

        PlayerData[] players;
        int numberOfPlayers = 4;
        float playerScaling;
        int currentPlayer = 0;

        bool rocketFlying = false;
        Vector2 rocketPosition;
        Vector2 rocketDirection;
        float rocketAngle;
        float rocketScaling = 0.1f;


       List<Vector2> smokeList = new List<Vector2> ();
       Random randomizer = new Random();
       int[] terrainContour;

       public Game1()
       {
           graphics = new GraphicsDeviceManager(this);
           Content.RootDirectory = "Content";
       }

       protected override void Initialize()
       {
           graphics.PreferredBackBufferWidth = 500;
           graphics.PreferredBackBufferHeight = 500;
           graphics.IsFullScreen = false;
           graphics.ApplyChanges();
           Window.Title = "Riemer's 2D XNA Tutorial";
```

```
            base.Initialize();
        }

        protected override void LoadContent()
        {
            device = graphics.GraphicsDevice;
            spriteBatch = new SpriteBatch(device);

            screenWidth = device.PresentationParameters.BackBufferWidth;
            screenHeight = device.PresentationParameters.BackBufferHeight;


            backgroundTexture = Content.Load<Texture2D> ("background");
            carriageTexture = Content.Load<Texture2D> ("carriage");
            cannonTexture = Content.Load<Texture2D> ("cannon");
            rocketTexture = Content.Load<Texture2D> ("rocket");
            smokeTexture = Content.Load<Texture2D> ("smoke");
            groundTexture = Content.Load<Texture2D> ("ground");
            font = Content.Load<SpriteFont> ("myFont");
            playerScaling = 40.0f / (float)carriageTexture.Width;
            GenerateTerrainContour();
            SetUpPlayers();
            FlattenTerrainBelowPlayers();
            CreateForeground();
        }

        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);
        }

        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();
            UpdateRocket();

            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);
         }

         protected override void Draw(GameTime gameTime)
         {
             graphics.GraphicsDevice.Clear(Color.CornflowerBlue);

             spriteBatch.Begin();
             DrawScenery();
             DrawPlayers();
             DrawText();
             DrawRocket();
             DrawSmoke();
             spriteBatch.End();
```

```
            base.Draw(gameTime);
        }

        private void DrawScenery()
        {
            Rectangle screenRectangle = new Rectangle(0, 0, screenWidth, screenHeight);
            spriteBatch.Draw(backgroundTexture, screenRectangle, Color.White);
            spriteBatch.Draw(foregroundTexture, screenRectangle, Color.White);
        }

        private void DrawPlayers()
        {
            foreach (PlayerData player in players)
            {
                if (player.IsAlive)
                {
                    int xPos = (int)player.Position.X;
                    int yPos = (int)player.Position.Y;
                    Vector2 cannonOrigin = new Vector2(11, 50);

                    spriteBatch.Draw(cannonTexture, new Vector2(xPos + 20, yPos - 10), null,
player.Color, player.Angle, cannonOrigin, playerScaling, SpriteEffects.None, 1);
                    spriteBatch.Draw(carriageTexture, player.Position, null, player.Color, 0, new
Vector2(0, carriageTexture.Height), playerScaling, SpriteEffects.None, 0);
                }
            }
        }

        private void DrawText()
        {
            PlayerData player = players[currentPlayer];
            int currentAngle = (int)MathHelper.ToDegrees(player.Angle);
            spriteBatch.DrawString(font, "Cannon angle: " + currentAngle.ToString(), new Vector2(20,
20), player.Color);
            spriteBatch.DrawString(font, "Cannon power: " + player.Power.ToString(), new Vector2(20,
45), player.Color);
        }

        private void DrawRocket()
        {
            if (rocketFlying)
                spriteBatch.Draw(rocketTexture, rocketPosition, null, players[currentPlayer].Color,
rocketAngle, new Vector2(42, 240), 0.1f, 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);
        }
    }
}
```