

الدرس العاشر

أهلاً بكم في الدرس العاشر من سلسلة دروس تعلم 3D Xna (السلسلة الثانية)، في هذا الدرس سوف نقوم بإضافة الأهداف "Targets" إلى اللعبة.

بعد إعداد المشهد و إتمام طيران الطائرة، حان الوقت لإضافة بعض الأهداف من أجل الإطلاق عليها. في هذه الحالة، سوف نقوم باستخدام كرات بسيطة. كما فعلنا مع الطائرة، سوف نقوم بإستيراد نموذج الهدف من ملف.

بإمكانك تنزيل نموذج الهدف من [هذا الرابط](#) (أو من الملفات المرفقة في الدرس الأول). وهو عبارة عن كرة حمراء بسيطة، حجمها بالضبط يساوي 1. بعدها، قم بإضافة ملف ال X. إلى مجلد ال Content في المشروع، كما قمنا بذلك سابقاً.

بعدها، سوف نقوم بتعريف متغير في أعلى الكود يعبر عن أكبر عدد من الأهداف التي نريدها في اللعبة، إضافة إلى متغير لأجل تخزين نموذج الهدف في داخله.

```
const int maxTargets = 50;
Model targetModel;

List<BoundingSphere> targetList = new List<BoundingSphere> ();
```

سوف نستخدم المتغير الأخير من أجل متابعة الأهداف التي سوف نضيفها. كائن القائمة "List" في هذه الحالة مفيد بشكل ممتاز، حيث سيكون بإمكاننا إضافة و حذف الأهداف من القائمة أثناء تشغيل البرنامج. لكل هدف يلزمنا تخزين شيئين:

- موقع الهدف
- حجم الهدف

لذا، فإن كرة الحدود "BoundingSphere" كافية بشكل تام من أجل التعبير عن و تخزين الهدف! إضافة إلى ذلك، سيكون بإمكاننا لاحقاً استخدام تقنية إكتشاف التصادمات المدمجة في كائن ال "BoundingSphere" لفحص التصادمات.

أولاً، قم بإضافة السطر التالي إلى الدالة LoadContent، من أجل إستيراد ملف ال X. إلى داخل المتغير targetModel.

```
targetModel = LoadModel("target");
```

حيث يتم إستدعاء الدالة LoadModel التي تقوم بتعبئة المتغير targetModel، و تقوم بإستبدال كل التأثيرات في النموذج بنسخة من التأثيرات الخاصة بنا.

سوف نقوم بإضافة دالة جديدة، بإسم AddTargets، حيث سوف تقوم بإنشاء الأهداف و إضافتها إلى القائمة:

```
private void AddTargets()
{
    int cityWidth = floorPlan.GetLength(0);
```

```

int cityLength = floorPlan.GetLength(1);

Random random = new Random();

while (targetList.Count < maxTargets)
{
    int x = random.Next(cityWidth);
    int z = -random.Next(cityLength);
    float y = (float)random.Next(2000) / 1000f + 1;
    float radius = (float)random.Next(1000) / 1000f * 0.2f + 0.01f;

    BoundingBox newTarget = new BoundingBox(new
Vector3(x,y,z), radius);

    if (CheckCollision(newTarget) == CollisionType.None)
        targetList.Add(newTarget);
}
}

```

تبدأ هذه الدالة بقراءة حجم المدينة و إنشاء كائن جديد للأرقام العشوائية، حيث سوف يقوم بإنشاء أرقام عشوائية لتحديد مواقع و احجام الأهداف.

في كل دورة في التكرار، نقوم بإنشاء هدف جديد، و نفحص إذا ما كان متصادما مع أحد المباني و من ثم نقوم بإضافته إلى القائمة.

يتم ذلك من خلال إنشاء قيم X، Y و Z عشوائية، و حجم عشوائي للهدف الجديد. حيث نقوم بإستعمال هذه القيم لإنشاء كرة حدود "BoundingBox" جديدة. بعد أن نحصل على الكرة، نقوم ببساطة بتمريرها إلى الدالة CheckCollision بحيث نتأكد من عدم تصادمها مع أي من المباني في المدينة!

في وقت لاحق في هذا الدرس، سوف نقوم بتوسيع الدالة CheckCollision بحيث تقوم بفحص التصادم مع الأهداف، إذن عند إستدعاء هذه الدالة سوف نتأكد أنه لن يتصادم أي هدفين معا أبدا.

إذا كان الهدف المنشئ حديثا لا يتصادم مع أي مبنى أو مع أي من الأهداف الموجودة، يتم إضافته إلى القائمة.

قم بإستدعاء هذه الدالة من داخل الدالة Initialize:

```
AddTargets();
```

عند تشغيل هذا الكود ليس من المفروض أن تحصل أي مشكلة، ولكن لن نلاحظ أي إختلاف أيضا لأننا لم نرسم الأهداف بعد. قم بإضافة هذه الدالة إلى أسفل الكود:

```

private void DrawTargets()
{
    for (int i = 0; i < targetList.Count; i++)
    {
        Matrix worldMatrix =
Matrix.CreateScale(targetList[i].Radius) *
Matrix.CreateTranslation(targetList[i].Center);

```

```

        Matrix[] targetTransforms = new
Matrix[targetModel.Bones.Count];
        targetModel.CopyAbsoluteBoneTransformsTo(targetTransforms);
        foreach (ModelMesh mesh in targetModel.Meshes)
        {
            foreach (Effect currentEffect in mesh.Effects)
            {
                currentEffect.CurrentTechnique =
currentEffect.Techniques["Colored"];

currentEffect.Parameters["xWorld"].SetValue(targetTransforms[mesh.Parent
Bone.Index] * worldMatrix);

currentEffect.Parameters["xView"].SetValue(viewMatrix);

currentEffect.Parameters["xProjection"].SetValue(projectionMatrix);

currentEffect.Parameters["xEnableLighting"].SetValue(true);

currentEffect.Parameters["xLightDirection"].SetValue(lightDirection);

currentEffect.Parameters["xAmbient"].SetValue(0.5f);
            }
            mesh.Draw();
        }
    }
}

```

هذه الدالة تقوم بالضبط بما تقوم به الدالة DrawModel، بإستثناء أن الكود هنا يتكرر لكل هدف في قائمة الأهداف "targetList".

مصفوفة العالم التي يتم إنشاءها لكل هدف تحدد حجم و مكان ذلك الهدف. بعدها، يتم رسم الهدف بإستخدام التقنية "Colored"، كما هي الرؤوس في ملف ال X. الخاص بالهدف الذي يحتوي فقط على معلومات الألوان و المتجهات العمودية.

لا تنسى إستدعاء هذه الدالة من داخل الدالة Draw():

```
DrawTargets();
```

عند تشغيل الكود الآن سوف تلاحظ أنه تم رسم الأهداف في المدينة!

كل ما سنقوم بعمله الآن هو إكتشاف متى سوف تصطدم الطائرة بالأهداف. أيضا عندما يحصل ذلك، سوف يتم حذف الهدف من قائمة الأهداف. لذا قم بإضافة الكود التالي إلى الدالة CheckCollision() قبل الجملة الأخيرة طبعاً:

```

for (int i = 0; i < targetList.Count; i++)
{
    if (targetList[i].Contains(sphere) != ContainmentType.Disjoint)
    {
        targetList.RemoveAt(i);
        i--;
    }
}

```

```

        AddTargets();

        return CollisionType.Target;
    }
}

```

لكل هدف في قائمة الأهداف، نقوم بفحص إذا ما كان متصادما مع الكرة التي يتم تمريرها إلى الدالة (التي تمثل الطائرة أو أحد الكرات الأخرى). إذا حدث ذلك، نقوم بحذف ذلك الهدف من قائمة الأهداف و من ثم نقوم بإستدعاء الدالة AddTargets من أجل إضافة هدف جديد في موقع عشوائي. لا تنسى أننا نقوم بإرجاع نوع التصادم إلى الكود المستدعي!

ملاحظتين:

- عندما نقوم بحذف عنصر من القائمة، يجب أن نقوم بإنقاص العداد بواحد، لاحظ السطر الذي يحتوي على --i. إذا لم تقم بعمل ذلك لن يتم فحص العنصر التالي في القائمة!
- ذلك يفحص إذا ماكنت الطائرة متصادمة مع أحد الأهداف، لكن تأكد أيضا أنه لن يتم في حال من الأحوال وضع هدفين في نفس الموقع. سبب ذلك أننا نقوم بإستدعاء هذه الدالة (CheckCollision) من داخل الدالة AddTargets.

هذا كل شيء! عندما تقوم بتشغيل هذا الكود، سوف يتم رسم الأهداف في المدينة. و عندما تصطدم بأحد الأهداف يتم إعادة الطائرة إلى الموقع الأصلي.



ما الشيء الجيد في الأهداف إذا لم يكن بإمكاننا أن نطلق النار عليهم؟ 😊 في الدرس القادم سوف نقوم بإضافة بعض الرصاص. 😊

كود المشروع:

```

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 XNASeries2
{
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        enum CollisionType { None, Building, Boundary, Target }

        const int maxTargets = 50;
        int[] buildingHeights = new int[] { 0, 2, 2, 6, 5, 4 };

        GraphicsDeviceManager graphics;
        GraphicsDevice device;

        Effect effect;
        Vector3 lightDirection = new Vector3(3, -2, 5);
        Matrix viewMatrix;
        Matrix projectionMatrix;

        Texture2D sceneryTexture;
        Model xwingModel;
        Model targetModel;

        Vector3 xwingPosition = new Vector3(8, 1, -3);
        Quaternion xwingRotation = Quaternion.Identity;

        int[,] floorPlan;
        float gameSpeed = 1.0f;

        VertexBuffer cityVertexBuffer;
        VertexDeclaration texturedVertexDeclaration;

        BoundingBox[] buildingBoundingBoxes;
        BoundingBox completeCityBox;

        List<BoundingBox> targetList = new List<BoundingBox> ();

        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 XNA Tutorials -- Series 2";

            LoadFloorPlan();
            lightDirection.Normalize();
            SetUpBoundingBoxes();
            AddTargets();

            base.Initialize();
        }

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

            effect = Content.Load<Effect> ("effects");
            sceneryTexture = Content.Load<Texture2D> ("texturemap");
            xwingModel = LoadModel("xwing");
            targetModel = LoadModel("target");

            SetUpVertices();
            SetUpCamera();
        }

        private Model LoadModel(string assetName)

```

```

{
    Model newModel = Content.Load<Model> (assetName);
    foreach (ModelMesh mesh in newModel.Meshes)
        foreach (ModelMeshPart meshPart in mesh.MeshParts)
            meshPart.Effect = effect.Clone(device);
    return newModel;
}

private void LoadFloorPlan()
{
    floorPlan = new int[,]
    {
        {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,1,1,0,0,0,1,1,0,0,1,0,1},
        {1,0,0,1,1,0,0,0,1,0,0,0,1,0,1},
        {1,0,0,0,1,1,0,1,1,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,1,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,1,1,0,0,0,1,0,0,0,0,0,0,1},
        {1,0,1,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,1,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,1,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,1,0,0,0,1,0,0,0,0,0,1},
        {1,0,1,0,0,0,0,0,0,1,0,0,0,0,1},
        {1,0,1,1,0,0,0,0,1,1,0,0,0,1,1},
        {1,0,0,0,0,0,0,0,1,1,0,0,0,1,1},
        {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
    };

    Random random = new Random();
    int differentBuildings = buildingHeights.Length - 1;
    for (int x = 0; x < floorPlan.GetLength(0); x++)
        for (int y = 0; y < floorPlan.GetLength(1); y++)
            if (floorPlan[x, y] == 1)
                floorPlan[x, y] = random.Next(differentBuildings) + 1;
}

private void SetUpBoundingBoxes()
{
    int cityWidth = floorPlan.GetLength(0);
    int cityLength = floorPlan.GetLength(1);

    List<BoundingBox> bbList = new List<BoundingBox> ();
    for (int x = 0; x < cityWidth; x++)
    {
        for (int z = 0; z < cityLength; z++)
        {
            int buildingType = floorPlan[x, z];
            if (buildingType != 0)
            {
                int buildingHeight = buildingHeights[buildingType];
                Vector3[] buildingPoints = new Vector3[2];
                buildingPoints[0] = new Vector3(x, 0, -z);
                buildingPoints[1] = new Vector3(x + 1, buildingHeight, -z - 1);
                BoundingBox buildingBox = BoundingBox.CreateFromPoints(buildingPoints);
                bbList.Add(buildingBox);
            }
        }
    }
    buildingBoundingBoxes = bbList.ToArray();

    Vector3[] boundaryPoints = new Vector3[2];
    boundaryPoints[0] = new Vector3(0, 0, 0);
    boundaryPoints[1] = new Vector3(cityWidth, 20, -cityLength);
    completeCityBox = BoundingBox.CreateFromPoints(boundaryPoints);
}

private void AddTargets()
{
    int cityWidth = floorPlan.GetLength(0);
    int cityLength = floorPlan.GetLength(1);

    Random random = new Random();

```

```
while (targetList.Count < maxTargets)
{
    int x = random.Next(cityWidth);
    int z = -random.Next(cityLength);
    float y = (float)random.Next(2000) / 1000f + 1;
    float radius = (float)random.Next(1000) / 1000f * 0.2f + 0.01f;

    BoundingBox newTarget = new BoundingBox(new Vector3(x, y, z), radius);

    if (CheckCollision(newTarget) == CollisionType.None)
        targetList.Add(newTarget);
}

private void SetUpCamera()
{
    viewMatrix = Matrix.CreateLookAt(new Vector3(20, 13, -5), new Vector3(8, 0, -7), new
Vector3(0, 1, 0));
    projectionMatrix = Matrix.CreatePerspectiveFieldOfView(MathHelper.PiOver4,
device.Viewport.AspectRatio, 0.2f, 500.0f);
}

private void SetUpVertices()
{
    int differentBuildings = buildingHeights.Length - 1;
    float imagesInTexture = 1 + differentBuildings * 2;

    int cityWidth = floorPlan.GetLength(0);
    int cityLength = floorPlan.GetLength(1);

    List<VertexPositionNormalTexture> verticesList = new List<VertexPositionNormalTexture> ();
    for (int x = 0; x < cityWidth; x++)
    {
        for (int z = 0; z < cityLength; z++)
        {
            int currentbuilding = floorPlan[x, z];

            //floor or ceiling
            verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
buildingHeights[currentbuilding], -z), new Vector3(0, 1, 0), new Vector2((currentbuilding * 2 /
imagesInTexture, 1)));
            verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
buildingHeights[currentbuilding], -z - 1), new Vector3(0, 1, 0), new Vector2((currentbuilding * 2) /
imagesInTexture, 0)));
            verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1,
buildingHeights[currentbuilding], -z), new Vector3(0, 1, 0), new Vector2((currentbuilding * 2 + 1) /
imagesInTexture, 1)));

            verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
buildingHeights[currentbuilding], -z - 1), new Vector3(0, 1, 0), new Vector2((currentbuilding * 2) /
imagesInTexture, 0)));
            verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1,
buildingHeights[currentbuilding], -z - 1), new Vector3(0, 1, 0), new Vector2((currentbuilding * 2 + 1)
/ imagesInTexture, 0)));
            verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1,
buildingHeights[currentbuilding], -z), new Vector3(0, 1, 0), new Vector2((currentbuilding * 2 + 1) /
imagesInTexture, 1)));

            if (currentbuilding != 0)
            {
                //front wall
                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1, 0, -z - 1),
new Vector3(0, 0, -1), new Vector2((currentbuilding * 2) / imagesInTexture, 1)));
                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
buildingHeights[currentbuilding], -z - 1), new Vector3(0, 0, -1), new Vector2((currentbuilding * 2 - 1)
/ imagesInTexture, 0)));
                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x, 0, -z - 1), new
Vector3(0, 0, -1), new Vector2((currentbuilding * 2 - 1) / imagesInTexture, 1)));

                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
buildingHeights[currentbuilding], -z - 1), new Vector3(0, 0, -1), new Vector2((currentbuilding * 2 - 1)
/ imagesInTexture, 0)));
                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1, 0, -z - 1),
new Vector3(0, 0, -1), new Vector2((currentbuilding * 2) / imagesInTexture, 1)));
                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1,
buildingHeights[currentbuilding], -z - 1), new Vector3(0, 0, -1), new Vector2((currentbuilding * 2) /
imagesInTexture, 0)));

                //back wall
```

```
verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1, 0, -z), new
Vector3(0, 0, 1), new Vector2((currentbuilding * 2) / imagesInTexture, 1)));
verticesList.Add(new VertexPositionNormalTexture(new Vector3(x, 0, -z), new
Vector3(0, 0, 1), new Vector2((currentbuilding * 2 - 1) / imagesInTexture, 1)));
verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
buildingHeights[currentbuilding], -z), new Vector3(0, 0, 1), new Vector2((currentbuilding * 2 - 1) /
imagesInTexture, 0)));

verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
buildingHeights[currentbuilding], -z), new Vector3(0, 0, 1), new Vector2((currentbuilding * 2 - 1) /
imagesInTexture, 0)));
verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1,
buildingHeights[currentbuilding], -z), new Vector3(0, 0, 1), new Vector2((currentbuilding * 2) /
imagesInTexture, 0)));
verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1, 0, -z), new
Vector3(0, 0, 1), new Vector2((currentbuilding * 2) / imagesInTexture, 1)));

//left wall
verticesList.Add(new VertexPositionNormalTexture(new Vector3(x, 0, -z), new
Vector3(-1, 0, 0), new Vector2((currentbuilding * 2) / imagesInTexture, 1)));
verticesList.Add(new VertexPositionNormalTexture(new Vector3(x, 0, -z - 1), new
Vector3(-1, 0, 0), new Vector2((currentbuilding * 2 - 1) / imagesInTexture, 1)));
verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
buildingHeights[currentbuilding], -z - 1), new Vector3(-1, 0, 0), new Vector2((currentbuilding * 2 - 1)
/ imagesInTexture, 0)));

verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
buildingHeights[currentbuilding], -z - 1), new Vector3(-1, 0, 0), new Vector2((currentbuilding * 2 - 1)
/ imagesInTexture, 0)));
verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
buildingHeights[currentbuilding], -z), new Vector3(-1, 0, 0), new Vector2((currentbuilding * 2) /
imagesInTexture, 0)));
verticesList.Add(new VertexPositionNormalTexture(new Vector3(x, 0, -z), new
Vector3(-1, 0, 0), new Vector2((currentbuilding * 2) / imagesInTexture, 1)));

//right wall
verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1, 0, -z), new
Vector3(1, 0, 0), new Vector2((currentbuilding * 2) / imagesInTexture, 1)));
verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1,
buildingHeights[currentbuilding], -z - 1), new Vector3(1, 0, 0), new Vector2((currentbuilding * 2 - 1)
/ imagesInTexture, 0)));
verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1, 0, -z - 1),
new Vector3(1, 0, 0), new Vector2((currentbuilding * 2 - 1) / imagesInTexture, 1)));

verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1,
buildingHeights[currentbuilding], -z - 1), new Vector3(1, 0, 0), new Vector2((currentbuilding * 2 - 1)
/ imagesInTexture, 0)));
verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1, 0, -z), new
Vector3(1, 0, 0), new Vector2((currentbuilding * 2) / imagesInTexture, 1)));
verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1,
buildingHeights[currentbuilding], -z), new Vector3(1, 0, 0), new Vector2((currentbuilding * 2) /
imagesInTexture, 0)));
}
}

cityVertexBuffer = new VertexBuffer(device, verticesList.Count *
VertexPositionNormalTexture.SizeInBytes, BufferUsage.WriteOnly);

cityVertexBuffer.SetData<VertexPositionNormalTexture> (verticesList.ToArray());
texturedVertexDeclaration = new VertexDeclaration(device,
VertexPositionNormalTexture.VertexElements);
}

protected override void UnloadContent()
{
}

protected override void Update(GameTime gameTime)
{
    ProcessKeyboard(gameTime);
    float moveSpeed = gameTime.ElapsedGameTime.Milliseconds / 500.0f * gameSpeed;
    MoveForward(ref xwingPosition, xwingRotation, moveSpeed);

    BoundingBox xwingSpere = new BoundingBox(xwingPosition, 0.04f);
    if (CheckCollision(xwingSpere) != CollisionType.None)
    {
        xwingPosition = new Vector3(8, 1, -3);
        xwingRotation = Quaternion.Identity;
        gameSpeed /= 1.1f;
    }
}
```

```

        UpdateCamera();

        base.Update(gameTime);
    }

    private void UpdateCamera()
    {
        Vector3 campos = new Vector3(0, 0.1f, 0.6f);
        campos = Vector3.Transform(campos, Matrix.CreateFromQuaternion(xwingRotation));
        campos += xwingPosition;

        Vector3 camup = new Vector3(0, 1, 0);
        camup = Vector3.Transform(camup, Matrix.CreateFromQuaternion(xwingRotation));

        viewMatrix = Matrix.CreateLookAt(campos, xwingPosition, camup);
        projectionMatrix = Matrix.CreatePerspectiveFieldOfView(MathHelper.PiOver4,
device.Viewport.AspectRatio, 0.2f, 500.0f);
    }

    private void ProcessKeyboard(GameTime gameTime)
    {
        float leftRightRot = 0;

        float turningSpeed = (float)gameTime.ElapsedGameTime.TotalMilliseconds / 1000.0f;
        turningSpeed *= 1.6f * gameSpeed;
        KeyboardState keys = Keyboard.GetState();
        if (keys.IsKeyDown(Keys.Right))
            leftRightRot += turningSpeed;
        if (keys.IsKeyDown(Keys.Left))
            leftRightRot -= turningSpeed;

        float upDownRot = 0;
        if (keys.IsKeyDown(Keys.Down))
            upDownRot += turningSpeed;
        if (keys.IsKeyDown(Keys.Up))
            upDownRot -= turningSpeed;

        Quaternion additionalRot = Quaternion.CreateFromAxisAngle(new Vector3(0, 0, -1),
leftRightRot) * Quaternion.CreateFromAxisAngle(new Vector3(1, 0, 0), upDownRot);
        xwingRotation *= additionalRot;
    }

    private void MoveForward(ref Vector3 position, Quaternion rotationQuat, float speed)
    {
        Vector3 addVector = Vector3.Transform(new Vector3(0, 0, -1), rotationQuat);
        position += addVector * speed;
    }

    private CollisionType CheckCollision(BoundingSphere sphere)
    {
        for (int i = 0; i < buildingBoundingBoxes.Length; i++)
            if (buildingBoundingBoxes[i].Contains(sphere) != ContainmentType.Disjoint)
                return CollisionType.Building;

        if (completeCityBox.Contains(sphere) != ContainmentType.Contains)
            return CollisionType.Boundary;

        for (int i = 0; i < targetList.Count; i++)
        {
            if (targetList[i].Contains(sphere) != ContainmentType.Disjoint)
            {
                targetList.RemoveAt(i);
                i--;
                AddTargets();

                return CollisionType.Target;
            }
        }

        return CollisionType.None;
    }

    protected override void Draw(GameTime gameTime)
    {
        device.Clear(ClearOptions.Target | ClearOptions.DepthBuffer, Color.DarkSlateBlue, 1.0f, 0);

        DrawCity();
        DrawModel();

        DrawTargets();
    }

```

```
        base.Draw(gameTime);
    }

    private void DrawCity()
    {
        effect.CurrentTechnique = effect.Techniques["Textured"];
        effect.Parameters["xWorld"].SetValue(Matrix.Identity);
        effect.Parameters["xView"].SetValue(viewMatrix);
        effect.Parameters["xProjection"].SetValue(projectionMatrix);
        effect.Parameters["xTexture"].SetValue(sceneryTexture);
        effect.Parameters["xEnableLighting"].SetValue(true);
        effect.Parameters["xLightDirection"].SetValue(lightDirection);
        effect.Parameters["xAmbient"].SetValue(0.5f);
        effect.Begin();
        foreach (EffectPass pass in effect.CurrentTechnique.Passes)
        {
            pass.Begin();
            device.VertexDeclaration = texturedVertexDeclaration;
            device.Vertices[0].SetSource(cityVertexBuffer, 0,
            VertexPositionNormalTexture.SizeInBytes);
            device.DrawPrimitives(PrimitiveType.TriangleList, 0, cityVertexBuffer.SizeInBytes /
            VertexPositionNormalTexture.SizeInBytes / 3);
            pass.End();
        }
        effect.End();
    }

    private void DrawModel()
    {
        Matrix worldMatrix = Matrix.CreateScale(0.0005f, 0.0005f, 0.0005f) *
        Matrix.CreateRotationY(MathHelper.Pi) * Matrix.CreateFromQuaternion(xwingRotation) *
        Matrix.CreateTranslation(xwingPosition);

        Matrix[] xwingTransforms = new Matrix[xwingModel.Bones.Count];
        xwingModel.CopyAbsoluteBoneTransformsTo(xwingTransforms);
        foreach (ModelMesh mesh in xwingModel.Meshes)
        {
            foreach (Effect currentEffect in mesh.Effects)
            {
                currentEffect.CurrentTechnique = currentEffect.Techniques["Colored"];
                currentEffect.Parameters["xWorld"].SetValue(xwingTransforms[mesh.ParentBone.Index]
                * worldMatrix);
                currentEffect.Parameters["xView"].SetValue(viewMatrix);
                currentEffect.Parameters["xProjection"].SetValue(projectionMatrix);
                currentEffect.Parameters["xEnableLighting"].SetValue(true);
                currentEffect.Parameters["xLightDirection"].SetValue(lightDirection);
                currentEffect.Parameters["xAmbient"].SetValue(0.5f);
            }
            mesh.Draw();
        }
    }

    private void DrawTargets()
    {
        for (int i = 0; i < targetList.Count; i++)
        {
            Matrix worldMatrix = Matrix.CreateScale(targetList[i].Radius) *
            Matrix.CreateTranslation(targetList[i].Center);

            Matrix[] targetTransforms = new Matrix[targetModel.Bones.Count];
            targetModel.CopyAbsoluteBoneTransformsTo(targetTransforms);
            foreach (ModelMesh mesh in targetModel.Meshes)
            {
                foreach (Effect currentEffect in mesh.Effects)
                {
                    currentEffect.CurrentTechnique = currentEffect.Techniques["Colored"];
                    currentEffect.Parameters["xWorld"].SetValue(targetTransforms[mesh.ParentBone.I
                    ndex] * worldMatrix);
                    currentEffect.Parameters["xView"].SetValue(viewMatrix);
                    currentEffect.Parameters["xProjection"].SetValue(projectionMatrix);
                    currentEffect.Parameters["xEnableLighting"].SetValue(true);
                    currentEffect.Parameters["xLightDirection"].SetValue(lightDirection);
                    currentEffect.Parameters["xAmbient"].SetValue(0.5f);
                }
                mesh.Draw();
            }
        }
    }
}
```

