Creating a Role Playing Game with XNA Game Studio 3.0
Part 14
Adding Another Layer to the Tile Engine

To follow along with this tutorial you will have to have read the previous tutorials to understand much of what it going on. You can find a list of tutorials here: XNA 3.0 Role Playing Game Tutorials You will also find the latest version of the project on the web site on that page. If you want to follow along and type in the code from this PDF as you go, you can find the previous project at this link: Eyes of the Dragon - Version 13 You can download the graphics for the these tutorials at this link: Graphics.zip

This will be a fairly short tutorial to tell you the truth. What I did do was update the **TileMap** game component to show you how easy it is, with the method I used, to add a new layer to the map. To get started with this tutorial, if you are following along and typing this in as you go is to download the graphics file above. I had to change the tile set from being a JPEG to a PNG. I deleted the last tile in the tile set and add in a tile that had a lot of transparency to it. It is a tile with three rocks in it.

Soon, I think I am going to have to start work on a map editor. Being able to create your maps is a big part of creating a role playing game. I am in the process of trying to plan the best way to do it. I want to try and make it as full featured as possible.

To get started with this tutorial, after you have downloaded the newest version of the graphics, the first thing you will want to do is go to the **TileSets** folder in the **Content** folder. You will want to right click on **tileset1.jpg** and delete it. Then decompress the **Graphics.zip** and add the new **tileset1.png** file to the **TileSets** folder by right clicking it and selecting **Add Existing Item** and selecting the file.

All of the work I will be doing in this tutorial is in the **TileMap** game component. I had designed the tile engine so that it would be easy to add in new layers to the map. As I always do, I will give you the new code and then go over why I did what I did. This is the code for the new **TileMap** game component.

```
using System;
using System.Collections.Generic;
using System.Linq;
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.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;


namespace New2DRPG.CoreComponents
{
    /// <summary>
    /// This is a game component that implements IUpdateable.
    /// </summary>
    public class TileMap : Microsoft.Xna.Framework.DrawableGameComponent
    {
```

```
        List<TileMapLayer> tileMapLayers = new List<TileMapLayer>();

        Random random = new Random();
        SpriteBatch spriteBatch;

        public TileMap(int tileMapWidth, int tileMapHeight, Game game)
            : base(game)
        {
            spriteBatch =
                (SpriteBatch)Game.Services.GetService(typeof(SpriteBatch));

            TileMapLayer layer = new TileMapLayer(tileMapWidth, tileMapHeight);

            for (int x = 0; x < tileMapWidth; x++)
                for (int y = 0; y < tileMapHeight; y++)
                    layer.SetTile(x, y, random.Next(0, TileEngine.Tiles.Count -
1));

            tileMapLayers.Add(layer);

            layer = new TileMapLayer(tileMapWidth, tileMapHeight);

            for (int x = 0; x < tileMapWidth; x++)
                for (int y = 0; y < tileMapHeight; y++)
                {
                    layer.SetTile(x, y, -1);
                    if (random.Next(0, 50) == 0)
                    {
                        layer.SetTile(x, y, TileEngine.Tiles.Count - 1);
                    }
                }

            tileMapLayers.Add(layer);
        }

        /// <summary>
        /// Allows the game component to perform any initialization it needs to
before starting
        /// to run.  This is where it can query for any required services and load
content.
        /// </summary>
        public override void Initialize()
        {
            // TODO: Add your initialization code here

            base.Initialize();
        }

        /// <summary>
        /// Allows the game component to update itself.
        /// </summary>
        /// <param name="gameTime">Provides a snapshot of timing values.</param>
        public override void Update(GameTime gameTime)
        {
            // TODO: Add your update code here

            base.Update(gameTime);
        }
```

```
        public override void Draw(GameTime gameTime)
        {
            foreach (TileMapLayer layer in tileMapLayers)
            {
                for (int x = 0; x < TileEngine.TileMapWidth; x++)
                {
                    for (int y = 0; y < TileEngine.TileMapHeight; y++)
                    {
                        if (layer.GetTile(x, y) != -1)
                        {
                            spriteBatch.Draw(TileEngine.TileSet,
                                new Rectangle(x * TileEngine.TileWidth -
                                        TileEngine.CameraXPosition,
                                    y * TileEngine.TileHeight -
                                        TileEngine.CameraYPositon,
                                    TileEngine.TileWidth,
                                    TileEngine.TileHeight),
                                TileEngine.Tiles[layer.GetTile(x, y)],
                                Color.White);
                        }
                    }
                }
            }
            base.Draw(gameTime);
        }

        public void Hide()
        {
            Visible = false;
            Enabled = false;
        }

        public void Show()
        {
            Visible = true;
            Enabled = true;
        }
    }
}
```

You will see that all I had to do, to add in another layer, was to change the constructor and the **Draw** method. In the constructor, where I generated the random tile for the map, I took 1 off of the upper limit, since I had made the last tile in the tile set to be the partially transparent tile that I would use in the second layer. After creating the first layer and adding it to the list of layers, I created a second layer. What I did in this layer is initially set the tile index to -1. You may be screaming, you are going to get an **Index out of bounds** exception in the **Draw** method. I will get to the **Draw** method shortly. To determine if a tile in this layer is one of the rocks tile, I decided to use a 1 in 50 chance that the tile would be a rocks tile. To simulate that, I found a random number between 0 and 49. If the number chosen was 0, I set the tile to the last tile in the tile set, the rock tile.

You will see that in the **Draw** method, I added in a condition where I draw the tile on the screen. If the index of the tile in the layer is -1 I don't draw anything. This will do two things actually. It will keep the tile engine from generating an **Index out of bounds** exception. The other thing is it will help speed up the rendering process, a little. If you had the first tile in the tile set completely transparent and you used that instead of -1 XNA would draw the transparent tile. Using the -1 index and checking to see if the current index is -1 will speed up the rendering a little as you are not sending a call to the **Draw** method

of the **SpriteBatch** object. The fewer calls you make to the **Draw** method will help increase the rendering process. Later, I will also speed up the rendering process by only drawing the tiles that fit on the screen, plus or minus 1 or 2 tiles to keep the background color from showing through.

Well, that is it for this tutorial. I will try and have another tutorial available on the web site soon, so you can check out the blog that goes with these tutorials, http://xna-rpg.tutorials.com, for the latest news on these tutorials or the news page on my web site for the latest news on these tutorials.