Creating a Role Playing Game with XNA Game Studio 3.0
Part 32
Tile Map Editor - Part 5

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: http://www.jtmbooks.com/rpgtutorials/New2DRPG31.zip You can download the graphics from this link: Graphics.zip

I will be working on the editor again in this tutorial and the next one as well. The first thing that I will do is change the **Tileset** class. I am going to make it so that the constructor for the **Tileset** class takes a string for a parameter that will be the path to the tile set. The **Tileset** class will be responsible for reading itself in as well. Since most of the class is new I will give you the code for the entire class and then explain it.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Xna.Framework;
using System.Windows.Forms;
using System.IO;
using System.Xml;

namespace TileMapEditor
{
    public class Tileset
    {
        public string textureName;
        public int tileWidth, tileHeight;
        public List<Rectangle> tiles;
        public string TilesetTextureName;


        public Tileset(string tileSetName)
        {
            textureName = "";
            tileWidth = 0;
            tileHeight = 0;
            tiles = new List<Rectangle>();
            ProcessTileSet(tileSetName);
        }

        void ProcessTileSet(string filePath)
        {
            XmlDocument input = new XmlDocument();

            input.Load(filePath);
            foreach (XmlNode node in input.DocumentElement.ChildNodes)
            {
                if (node.Name == "TextureElement")
```

```
        {
            textureName = node.Attributes["TextureName"].Value;
        }
        if (node.Name == "TilesetDefinitions")
        {
            tileWidth = Int32.Parse(node.Attributes["TileWidth"].Value);
            tileHeight = Int32.Parse(node.Attributes["TileHeight"].Value);
        }

        if (node.Name == "TilesetRectangles")
        {
            List<Rectangle> rectangles = new List<Rectangle>();

            foreach (XmlNode rectNode in node.ChildNodes)
            {
                if (rectNode.Name == "Rectangle")
                {
                    Rectangle rect;
                    rect = new Rectangle(
                        Int32.Parse(rectNode.Attributes["X"].Value),
                        Int32.Parse(rectNode.Attributes["Y"].Value),
                        Int32.Parse(rectNode.Attributes["Width"].Value),
                        Int32.Parse(rectNode.Attributes["Height"].Value));
                    rectangles.Add(rect);
                }
            }

            tiles = rectangles;
        }
    }

    string fileName = Path.GetDirectoryName(filePath);
    fileName += @"\" + textureName;

    string[] extensions = { ".png", ".jpg", ".tga", ".bmp", ".gif" };
    bool found = false;

    foreach (string extension in extensions)
    {
        if (File.Exists(fileName + extension))
        {
            found = true;
            fileName += extension;
            break;
        }
    }

    if (found)
    {
        TilesetTextureName = fileName;
    }
    else
    {
        MessageBox.Show("Unrecognized image format in the tile set.",
            "Error",
            MessageBoxButtons.OK,
            MessageBoxIcon.Error);
        throw new Exception("Unable to load the tile set.");
    }
```

```
        }

    }
}
```

The first thing you will see is that I added in was three using statements. I used the **MessageBox** class in the **Tileset** class so I needed the using statement for **System.Windows.Forms**. I will be working with an **XML** document so I needed **System.Xml**. I will be doing some work with files and paths so I also needed the **System.IO** using statement. I also added in a string field, **textureName**, that will hold the name of the texture for the tile set.

The constructor for the class now takes a string parameter, the file name of the tile set with the full path. After initializing the fields the constructor then calls the **ProcessTileSet** method passing in the file name of the tile set with the full path to the tile set. The **ProcessTileSet** method should be very familiar by now. I had just copied and pasted the code.

The next thing that I did was change the **TileMap** class. First I add in a new field called **layerNames** that is a **List<string>** and it will hold names for the layers read in that will be added to **clbLayers**. What I did was add in a new constructor that takes as a parameter a string. It will then call a method to load a map, **LoadMap**, from disk similar to the method that I used to load the map into the game. The **LoadMap** method will call a **LoadLayer** method similar to the method that I used i nthe game to load in a layer of the map. I will give you the code for the constructor and then the code for the methods used to load the maps. Don't forget to add the field **layerNames** to the fields at the top of the class.

```
        public List<string> layerNames = new List<string>();

        public TileMap(string mapName)
        {
            LoadMap(mapName);
        }

        private void LoadMap(string mapName)
        {
            XmlDocument xmlDoc = new XmlDocument();

            try
            {
                xmlDoc.Load(mapName);
            }
            catch
            {
                throw new Exception("Unable to find the map.");

            }
            XmlNode rootNode = xmlDoc.FirstChild;

            if (rootNode.Name != "TileMap")
            {
                throw new Exception("Invalid tile map format!");
            }

            mapWidth = Int32.Parse(rootNode.Attributes["Width"].Value);
            mapHeight = Int32.Parse(rootNode.Attributes["Height"].Value);
```

```
        XmlNode layersNode = rootNode.FirstChild;

        if (layersNode.Name != "Layers")
        {
            throw new Exception("Invalid tile map format!");
        }

        TileMapLayer layer;
        int layerCounter = 0;

        foreach (XmlNode node in layersNode.ChildNodes)
        {
            if (node.Name == "Layer")
            {
                try
                {
                    layer = new TileMapLayer(mapWidth, mapHeight);
                    LoadLayer(node, layer);
                    layers.Add(layer);
                    layerCounter++;
                    layerNames.Add("Layer " + layerCounter.ToString());
                }
                catch
                {
                    throw new Exception("Error reading in map layer!");
                }
            }
        }
    }

    private void LoadLayer(XmlNode layerNode, TileMapLayer layer)
    {
        int rowCount = 0;
        foreach (XmlNode node in layerNode)
        {
            if (node.Name == "Row")
            {
                string row = node.InnerText;
                row.Trim();
                string[] cells = row.Split(' ');
                for (int i = 0; i < mapWidth; i++)
                    layer.SetTile(i, rowCount, Int32.Parse(cells[i]));
                rowCount++;
            }
        }
    }
```

The LoadMap method is almost the same as the **LoadMap** method from the **TileMap** class in the game. I had actually copied and pasted the code from the **TileMap** class in the game.

What is new in the **LoadMap** method is just before the foreach loop where I read in the layers of the map I have an integer variable called **layerCount**. I will use this variable to add a string to **layerNames** so that the layers will have a name in **clbLayers**.

In side the if in the foreach loop after I have successfully read in a layer and add it to the **layers** I increment **layerCount**. I then add a string to **layerNames** by taking "**Layers** " and then adding **layerCount** as a string. There was no change in the **LoadLayer** method.

Because I changed the **Tileset** class to be able to load itself I made a change to the **FrmNewMap** class. Right click the **FrmNewMap.cs** entry in the solution explorer and choose **View Code**. Find the **ProcessTileSet** method and remove it from the code. I added in a new string field to the class called **TilesetName** that will hold the name of the tile set. I changed the **btnOpenTileSet_Click** method. Inside the try-catch block instead of calling the **ProcessTileSet** method I call the constructor for the **Tileset** class passing in the file name selected by the user. I also set the **TilesetName** field to be the file name with out the extention. This will be that asset name that XNA uses. This is the new code.

```
public string TilesetName;

private void btnOpenTileSet_Click(object sender, EventArgs e)
{
    OpenFileDialog openFileDialog = new OpenFileDialog();

    openFileDialog.Filter = "(Tileset *.tset)|*.tset";
    openFileDialog.CheckFileExists = true;
    openFileDialog.CheckPathExists = true;
    openFileDialog.Multiselect = false;

    DialogResult dialogResult = openFileDialog.ShowDialog();

    if (dialogResult == DialogResult.OK)
    {
        try
        {
            tileset = new Tileset(openFileDialog.FileName);
            TilesetName =
                Path.GetFileNameWithoutExtension(openFileDialog.FileName);
            TilesetTextureName = tileset.TilesetTextureName;
            btnOK.Enabled = true;
        }
        catch
        {
            MessageBox.Show(
                "Error reading tileset.",
                "Error",
                MessageBoxButtons.OK,
                MessageBoxIcon.Error);
            tileset = null;
            btnOK.Enabled = false;
        }
    }
}
```

Now it is time to add in the functionality of the editor being able to read in a map. To handle being able to load in a map I first created an event handler for when the user clicks the **Open Map** option in the editor. I handled creating this in the constructor of the form. Again, when you type the += you can press tab twice to generate a stub for the method. This is the updated code for the constructor and the code for the method. I will explain the method after I have given you the code.

```
public Form1()
{
    InitializeComponent();
    layerToolStripMenuItem.Enabled = false;

    tileDisplay1.OnInitialize +=
        new EventHandler(tileDisplay1_OnInitialize);
```

```
    tileDisplay1.OnDraw +=
        new EventHandler(tileDisplay1_OnDraw);

    newMapToolStripMenuItem.Click +=
        new EventHandler(newMapToolStripMenuItem_Click);

    saveMapToolStripMenuItem.Enabled = false;
    saveMapToolStripMenuItem.Click +=
        new EventHandler(saveMapToolStripMenuItem_Click);

    openMapToolStripMenuItem.Click +=
        new EventHandler(openMapToolStripMenuItem_Click);

    exitToolStripMenuItem.Click +=
        new EventHandler(exitToolStripMenuItem_Click);

    nudCurrentTile.ValueChanged +=
        new EventHandler(nudCurrentTile_ValueChanged);
    nudCurrentTile.InterceptArrowKeys = false;
    Application.Idle += new EventHandler(Application_Idle);
}

void openMapToolStripMenuItem_Click(object sender, EventArgs e)
{
    OpenFileDialog openFileDialog = new OpenFileDialog();
    openFileDialog.AddExtension = true;
    openFileDialog.CheckFileExists = true;
    openFileDialog.CheckPathExists = true;
    openFileDialog.Filter = "(Tile Sets *.tset)|*.tset";
    openFileDialog.Multiselect = false;
    openFileDialog.ValidateNames = true;
    openFileDialog.Title = "Select tile set to open";

    DialogResult result = openFileDialog.ShowDialog();
    if (result == DialogResult.OK)
    {
        try
        {
            tileset = new Tileset(openFileDialog.FileName);
            texture = Texture2D.FromFile(GraphicsDevice,
                tileset.TilesetTextureName);
            tilesetBitmap = Bitmap.FromFile(tileset.TilesetTextureName);
        }
        catch
        {
            MessageBox.Show(
                "Error reading in tile set!",
                "Error",
                MessageBoxButtons.OK,
                MessageBoxIcon.Error);
            return;
        }
    }
    else
    {
        MessageBox.Show(
            "You must select a tile set to proceed!",
            "Error",
```

```
            MessageBoxButtons.OK,
            MessageBoxIcon.Error);
        return;
    }

    openFileDialog.Filter = "(Tile Maps *.tmap)|*.tmap";
    openFileDialog.Title = "Select map to open";
    result = openFileDialog.ShowDialog();
    if (result == DialogResult.OK)
    {
        try
        {
            tileMap = new TileMap(openFileDialog.FileName);
        }
        catch
        {
            MessageBox.Show(
                "Error reading in tile map!",
                "Error",
                MessageBoxButtons.OK,
                MessageBoxIcon.Error);
            return;
        }
    }
    clbLayers.Items.Clear();

    layerToolStripMenuItem.Enabled = true;
    saveMapToolStripMenuItem.Enabled = true;

    foreach (string layer in tileMap.layerNames)
        clbLayers.Items.Add(layer, true);

    clbLayers.SelectedIndex = clbLayers.Items.Count - 1;
    clbLayers.SelectionMode = SelectionMode.One;
    currentLayer = tileMap.layers[tileMap.layers.Count - 1];

    FillPictureBox(0);
    nudCurrentTile.Value = 0;
    nudCurrentTile.Maximum = tileset.tiles.Count - 1;
    this.Invalidate();
}
```

The first thing I do in the event handler is create an **OpenFileDialog** that I will use to open a tile set for the map. I set the **AddExtension** method to true so if the user doesn't type an extension for the file it will be appended to the file name. You want to make sure that the user provides a valid path and file name so I set the **CheckFileExists** and **CheckPathExists** properties to true as well. I set the **Filter** property so that the **OpenFileDialog** will display only files of type **tset.** You only want the user to be able to open one file so **Multiselect** is set to false. You want to make sure that file name entered by the user is valid so the **ValidateNames** property is set to true. The one other property I set is the **Title** property to **"Select tile set to open"** so that the user knows what type of file they are looking for. Finally I capture the result of the call to **ShowDialog** which displays the dialog and waits for the user to select a file.

I next check to see if the result of the call to **ShowDialog** was **DialogResult.OK** which means that the user selected a good file to work with. Inside the if statement there is a try catch block. In the try part I try and load in the tile set. In the catch part I report that there was an error reading in the tile

set and exit the method. Other wise I report that the user must open a tile set and exit the method.

I then change a couple properties of **openFileDialog**. The first one I set is the **Filter** property so that it will show files of type **tmap**. I also set the **TitleProperty** to "**Select map to open**". I then display the dialog and capture the result.

I then check if the result is **DialogResult.OK**, which means a good file of type **tmap** was found, I call the constructor of **TileMap** that takes as a parameter the selected file. If creating the **TileMap** object fails I display an error in a **MessageBox** and exit the method.

Next I do some processing of the map. The first thing that I do is clear **clbLayers** of all the items in it. The reason is because it will later control how the map is rendered in the editor. I then enable the **Save Map** option in the **File** menu and the **Layer** menu. I then add the layer names to **clbLayers** passing in true as well so that they will be checked.

I also set a few properties of **clbLayers**. I set the **SelectedIndex** property to the last item in the **CheckedListBox** by using the **Count** property of the **Items** collection and subtracting one. I also set the **SelectionMode** property to be **SelectionMode.One** to make sure that only one item can be selected in **clbLayers**. I also set **currentLayer** to be the last layer in in the map.

What remains is to take care of the preview of the tile being worked on and setting some properties of **nudCurrentTile** which controls the tile being drawn. I call **FillPictureBox** to set the preview of the tile that is being worked with passing 0 for the first tile. I set the **Value** property of **nudCurrentTile** to 0 meaning we are working on tile 0 in the tile set. I also set the **Maximum** property to the number of tiles minus one so an out of bounds exception is not thrown if you try and go past the last tile in the tile set. I also call the **Invalidate** method of the form to let it know that it should redraw itself.

I was going to add a little more to this tutorial but it turned out to be a little longer than I thought it would. I am working on a new tile set with more tiles to select from and I have already started coding the next part of Eyes of the Dragon. I encourage you to keep either visiting my site http://xna.jtmbooks.com or my blog, http://xna-rpg.blogspot.com for the latest news on my tutorials.