Creating a Role Playing Game with XNA Game Studio 3.0
Part 31
Reading the Map

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/New2DRPG30.zip You can download the graphics from this link: Graphics.zip

In this tutorial I will be covering reading in the map from the editor but not using the Content Pipeline. For this tutorial you will need a map file. You can find the one that I used here. Make sure that you right click the game project and set it as the **Start Up Project**. I right clicked the **TileSets** folder in the **Content** folder and added in the map. Its file name was **tilemap.tmap**. If you used a different file name or added it to a different folder in the **Content** folder remember the name of the folder and the name of the map file. Now click the map. You will want to set two properties for the map in the **Properties Window**. You will want to set the **Build Action** to **None** and **Copy To Output Directory** to **Copy Always**. This step is important because the game will not be able to find the map and it the build action is not set to **None** the game will try and compile the map and will not know what to do with it.

Most of the work in this tutorial will be done in the **TileMap** class. What I will be doing is adding a new constuctor that takes the file name of the map with a **Tileset** object and the **Game** object. I will also be adding in two helper methods **LoadMap** and **LoadLayer**. The **LoadMap** method will be responsible for reading in the map and **LoadLayer** will load in the layers of the map. I will start with the new constructor for the **TileMap** class. Below the original constructor for the **TileMap** class add in this constructor. You will also want to add the following using statement to the using statements for the class.

```
using System.Xml;

public TileMap(string mapName, Tileset tileset, Game game)
    : base(game)
{
    spriteBatch = Game1.TileSpriteBatch;
    Content =
        (ContentManager)Game.Services.GetService(typeof(ContentManager));

    this.tileset = tileset;

    LoadContent();
    LoadMap(mapName);

    TileMapLayer layer = new TileMapLayer(mapWidth, mapHeight);

    for (int x = 0; x < mapWidth; x++)
        for (int y = 0; y < mapHeight; y++)
        {
            layer.SetTile(x, y, -1);
            if (random.Next(0, 50) < 5)
```

```
                {
                    layer.SetTile(x, y, tileset.Tiles.Count - 1);
                }
            }

        tileMapLayers.Add(layer);

        for (int i = 0; i < 2; i++)
        {
            ItemSprite item =
                new ItemSprite(game,
                    itemTexture,
                    new Vector2(random.Next(0, 8), random.Next(0, 8)));
            items.Add(item);
        }
    }
```

The constuctor should look very familiar as it is mostly the same code as the other constructor. The first change is that the constructor has as parameters a string for the file name **mapName**, a **TileSet** object and the **Game object.** After the call to **LoadContent** I call the **LoadMap** method that will load in the map. I then created a layer with rock tiles like in the other constructor and added it to the list of layers. I then created two **ItemSprite** objects and added them to the list of items.

Like I said in the **LoadMap** method I will handle reading in the map. I will also be handling reading in the layers in the **LoadLayers** method. This is the code for the two methods.

```
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;

    foreach (XmlNode node in layersNode.ChildNodes)
    {
        if (node.Name == "Layer")
        {
            try
            {
                layer = new TileMapLayer(mapWidth, mapHeight);
                LoadLayer(node, layer);
                tileMapLayers.Add(layer);
            }
            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 takes as a parameter the file name for the map. It then creates an **XmlDocument** called **xmlDoc** that I will read the map into. I try to read in the map inside of a try catch block. If reading the map fails I throw an exception to let you know what failed. I then have an **XmlNode** object called **rootNode**. I set that to the first child of **xmlDoc**. This is where the order in which things were written is important. If the first node is not the **TileMap** node then I throw an exception saying that the map was not in the proper format. I then use the **Int32.Parse** method to parse the values of the **Width** and **Height** attributes.

I then created an **XmlNode layersNode** and set it to **rootNode.FirstChild** which should be the **Layers** node. If it is not the **Layers** node I again throw an exception saying the map is not in the proper format. If everything works fine until now I have a **TileMapLayer** variable called **layer** that will hold the layer that is read in and be added to the list of layers for the map.

Next there is a foreach loop in which I will try and read in all of the layers of the map. There is a try catch block in which I will create a new **TileMapLayer** instance. I then call the **LoadLayer** method passing in the current **XmlNode** and **layer**. If the layer was read in successfully it is added to the list of layers of the map. If the layer was not read in successfully I thrown an exception that there was an error reading the map.

The **LoadLayer** method takes two parameters: an **XmlNode** object and a **TileMapLayer** object. Inside the method I have a local variable **rowCount** that will count the number of rows in the map. Inside a foreach loop I loop through all of the child nodes of **layerNode**.

In side of the loop there is an if statement to see if the name of the node is **Row**. If it is I then set **row** to be the **InnerText** of the node. I then call the **Trim** method of **row**. What this does is remove any leading or trailing spaces from the string. In the next line I create an array of strings called **cells** by using the **Split** method and passing in a space as the argument. What this will do is create an array with just the tile numbers for the tiles in the row. I next loop through the number of cells which should be **mapWidth**. Since this method is called from a try catch block if there is an error it will return back to **LoadMap** and it will catch the exception. Inside the for loop I call the **SetTile** method of the **TileMapLayer** class with **i** as the **x** argument, **rowCount** as the **y** argument and I use the **Parse** method of **Int32** to parse the string to an integer. After the for loop I increment **rowCount** so that when the next row is found when I call the **SetTile** method it will work on the next row of the map.

There is just one more thing to do. You actually have to use this new constructor in the game. That will take place in the constructor of the **ActionScreen** class. Instead of passing in the width and height of the map I will pass in the path of the map relative to the **Content** folder. This is the updated constructor.

```
public ActionScreen(Game game, SpriteFont gameFont, string tilesetName)
    : base(game)
{
    playerCharacter = new PlayerCharacter(game);
    this.gameFont = gameFont;

    this.tilesetName = tilesetName;
    LoadContent();
    tileMap = new TileMap(@"Content\TileSets\tilemap.tmap", tileset, game);
    Components.Add(tileMap);
    tileMap.Show();

    viewportWidth = TileEngine.ViewPortWidth;
    viewportHeight = TileEngine.ViewPortHeight;
    screenWidth = game.Window.ClientBounds.Width;
    screenHeight = game.Window.ClientBounds.Height;
}
```

I know this tutorial was a little short. The next tutorial will be a little longer as I will be working on the editor again. I will also have an updated tile set with many more tiles in it. I am already working on coding the next part of Eyes of the Dragon so 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.