Creating a Role Playing Game with XNA Game Studio 3.0
Part 27
Tile Map Editor – Part 1

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

So far everything is working great. We have a tile map being rendered and a sprite for the player to navigate through the world with. We are missing one thing though. In a role playing game you want to be able to read in the map for the player to explore. In this tutorial I will be covering getting started creating a tile map editor for the game. Instead of creating a separate solution for the tile map editor I'm going to add it as a project to the game so you can just set it as the startup project and run it right from Visual C#. This is going to be a multipart tutorial. In this part I will go over setting up the form and get it rendering a map.

The first thing you will want to do is load the last version of the game. Next you will want to right click the solution and select **Add new project**. Make sure that **Visual C#** is selected and choose **Windows Form Applicaton**. Name the new project **TileMapEditor**. This will create a new project and add it to the current solution. You are going to have to add a reference to the new project for the XNA framework. You can do this by either right clicking the project and selecting **Add reference** or clicking the project and from the **Project** menu selecting **Add reference**. In the dialog that appears you will want to select the **.NET** tab and scroll down until you find the **Microsoft.Xna.Framework** entry and then select it.

In order to render the map in a windows forms application I will be using a sample from the XNA Creators website. You can find the sample on this web page of the XNA Creators website: http://creators.xna.com/en-US/sample/winforms_series1 After you have downloaded the sample extract it to a directory and remember where you extracted it to. Right click the **TileMapEditor** project and select the **Add existing item** option. Navigate to the directory that you extracted the files to and locate the following files: **GraphicsDeviceControl.cs**, **GraphicsDeviceService.cs** and **ServiceContainer.cs**. While holding down the control key select those three files.

Now you will want to open the **GraphicsDeviceService.cs** file. At the moment it is in the wrong namespace, as well as all of the other files we added. I am going to show you a quick way to get them all into the right namespace. You will see the line:

```
namespace WinFormsGraphicsDevice
```

Place your cursor at the end of the line. Delete the **WinFormsGraphicsDevice** part and replace it with **TileMapEditor.** Before continuing on hold down the **shift** and **alt** key and press **F10**. This will bring up a little box. You will want to choose the **Rename** option in that box with your mouse. This will rename all occurances of **WinFormsGraphicsDevice** to **TileMapEditor**.

There is something else that we need to do. The **GraphicsDeviceControl** class is an abstract

class, like the **Sprite** class that I created. To be able to use this class we have to make a class that inherits from the **GraphicsDeviceControl** class. Right click the **TileMapEditor** project and add a new class called **TileDisplay**. I will give you the code for the **TileDisplay** class and then I will explain it.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace TileMapEditor
{
    class TileDisplay : GraphicsDeviceControl
    {
        public event EventHandler OnInitialize;
        public event EventHandler OnDraw;

        protected override void Initialize()
        {
            if (OnInitialize != null)
                OnInitialize(this, null);
        }

        protected override void Draw()
        {
            if (OnDraw != null)
                OnDraw(this, null);
        }
    }
}
```

The **GraphicsDeviceControl** class has two abstract methods **Initialize** and **Draw** that you must implement. That is why the **Initialize** and **Draw** methods in this class have the **override** keywords. To implement these methods in the tile map editor I will be using two event handlers **OnInitialize** and **OnDraw**. The **Initialize** method is used to implementaion of the initialization of the control. The **Draw** method is used to intialize the actual drawing code. This will allow us to create a control that we can render to using XNA code. The one last thing we need to do before we can use this is to build the project. Save the project and then press **F6** to build the project. Now there will be an option in the **Toolbox** window called **TileMapEditor Components**. The **Toolbox** is represented on the left hand side of the screen by a tab that says **Toolbox**. If the **Toolbox** is not visible you can open it by clicking **Toolbox** under the **View** menu.

Now we are ready to create our editor. Open the **Toolbox** and press the little pin icon to pin it as we are going to be using it a lot and it is easier to have it open. Click **Form1.cs** in the **TileMapEditor** project. You will want to click the **ViewDesigner** button  to bring up the properties for the form. Change the **Size** property of the form to **965, 705**. Change the **StartPosition** property to **Manual**. Set the **Text** property to **Tile Map Editor**. I don't want the form to be maximized so set the **MaximizeBox** property to false. I don't want the form to be resized either so set the **FormBorderStyle** property to **FixedSingle**. Finally set the **Location** propery to **0, 25**.

Now it is time to start adding controls to the form. In this application it wil be a good idea to have a menu for various functions: loading and saving maps, loading in a tile set, creating new maps and new layers and other things. So goto the **Toolbox** and drag a **Menustrip** control onto the form. For now we will want three menus in the **Menustrip**: **File**, **Tile Set** and **Layer**. If you are unfamiliar with

creating Windows forms applications with C# when you add a **Menustrip** control and you select it you will see **Type Here** in the **Menustrip**. This is where you will type the text you want for your menus. In the first item you will want to type **&File**. The **&** will make the **F** have an underline to tell the person using the program that you can press **alt+F** to open that menu option. The next one will be **&Tile Set** and the last one I will add is **&Layer**. Go back and click **&File**. In the **File** menu I added four options: **&New Map, &Open Map**, **&Save Map** and **E&xit**. There will be only one option for the moment under **&Tile Set** and that will be **&Open Tile Set**. For **Layer** there will be three options: **&New Layer**, **&Open Layer** and **&Save Layer**.

The next control that we will want is a **TileDisplay** control. Open the **TileMapEditor Component** item in the **Toolbox** and drag a **TileDisplay** control onto the form. Set its **Location** property to **5, 30** and its **Size** property to **800, 640**.

We will need a way to select the tile we will want to draw. I will do that using a **Picturebox** control and a **NumericUpDown** control. The **Picturebox** will display the current tile so you know what you are drawing and the **NumericUpDown** control will be used to select which tile to draw.

Drag a **PictureBox** control onto the form. There are a lot of properties to set for the **Picturebox**: (**Name)** is **pbCurrentTile**, **BackColor** is **White, BorderStyle** is **FixedSingle**, **Location** is **820, 42** and the **Size** property is **128, 128**.

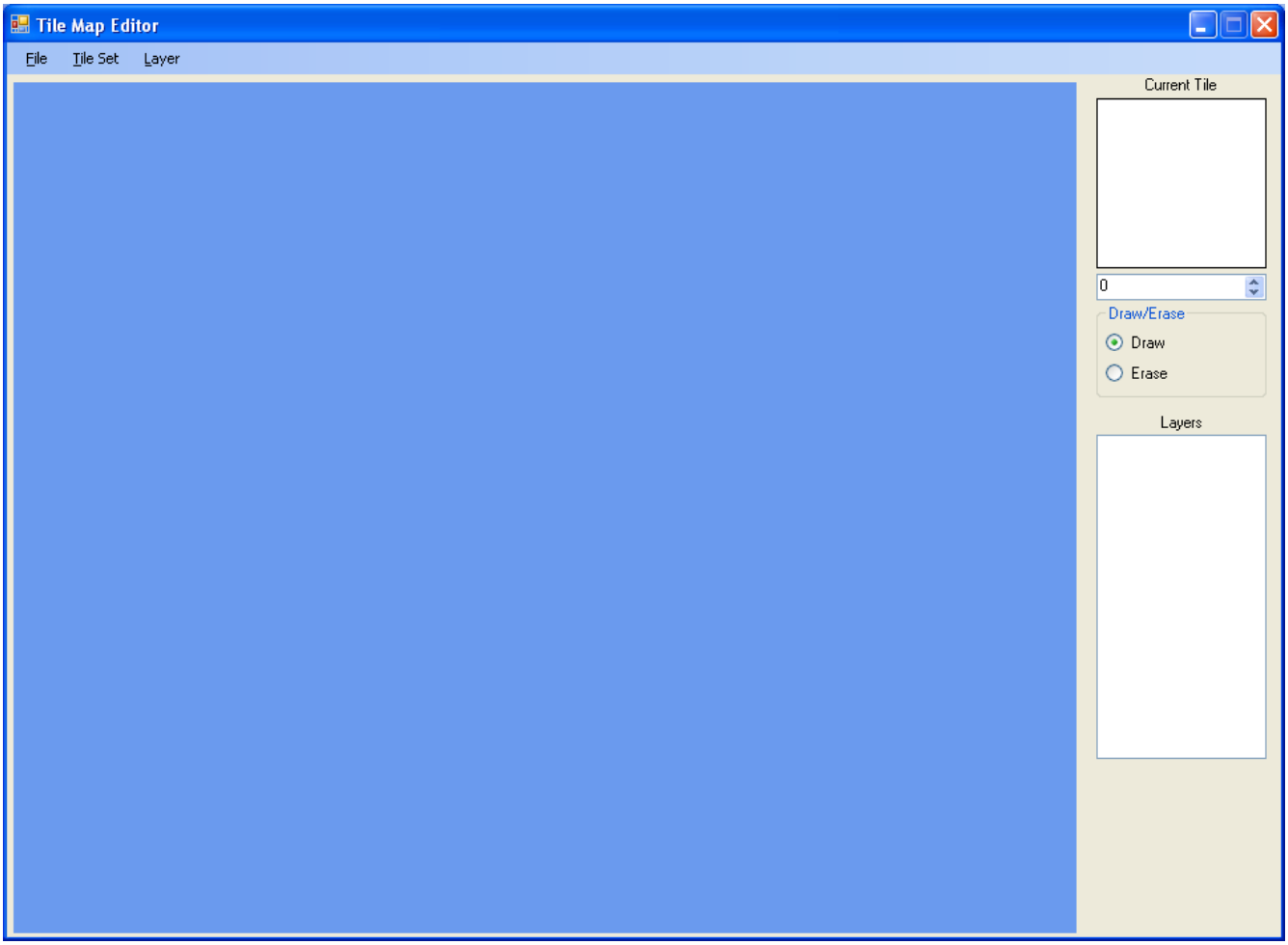
Now you will want to drag a **NumericUpDown** control onto the form. The properties for the **NumericUpDown** control are: **(Name)** is **nudCurrentTile**, **Location** is **820, 174** and **Size** is **128, 20**.

To control if the we are drawing or erasing tiles I will be using **RadioButtons**. I could have went with just a **CheckBox** but if you want to add in other options later using **RadioButtons** initially will make changing the code later on easier. To make coding the **RadioButtons** easier I will be using a group box. Drag a **GroupBox** onto the form and then drag two **RadioButtons** onto the **GroupBox**, not the form.

These are the properties for the **GroupBox**: **(Name)** is **gbDrawErase**, **Location** is **820, 197**, **Size** is **128, 70** and **Text** is **Draw/Erase**. The properties to change for the first **RadioButton** are: **(Name)** is **rbDraw**, **Checked** is **True**, and **Location** is **7, 20**. These are the properties to change for the second **RadioButton**: **(Name)** is **rbErase** and **Location** is **7, 43**.

There are three more controls that I want to add to the form two **Labels** and a **CheckedListBox**. The **CheckedListBox** will hold the list of layers and will be used to control which layer of the map we are working on and if that layer is visible in the editor. There are only three properties to set for the **CheckedListBox** control: **(Name)** is **clbLayer**, **Location** is **820, 273** and **Size** is **128, 244**.

The **Labels** are just for appearance purposes. Drag a **Label** control onto the form. Before doing anything else with the **Label** set the **AutoSize** property to **False**. This will just make centering the **Label** horizontally over top of the other controls easier. These are the other properties that you will want to change for the **Label**: **(Name)** is **lblCurrentTile**, **Location** is **820, 25**, **Size** is **128, 13**, **Text** is **Current Tile** and **TextAlign** will be **MiddleCenter**. Instead of draging a second **Label** onto the form hold down the **Control** key, select the **Label** and drag it down the form. This allows you to easily replicate controls on the form with similar properties. These are the properties to change for the new **Label**: **(Name)** is **lblLayers**, **Location** is **820, 279** and **Text** is **Layers**. The finished form should look something like this.



I am just going to add a little code to the form to initialize the **TileDisplay** control and clear the **TileDisplay** control. As always I will give you the code and then explain it. This is the code for the form so far.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;

namespace TileMapEditor
{
    public partial class Form1 : Form
    {
        SpriteBatch spriteBatch;

        public GraphicsDevice GraphicsDevice
        {
            get { return tileDisplay1.GraphicsDevice; }
        }

        public Form1()
        {
            InitializeComponent();
            tileDisplay1.OnInitialize +=
                  new EventHandler(tileDisplay1_OnInitialize);
            tileDisplay1.OnDraw +=
                  new EventHandler(tileDisplay1_OnDraw);
        }

        void tileDisplay1_OnInitialize(object sender, EventArgs e)
        {
            spriteBatch = new SpriteBatch(GraphicsDevice);
        }

        void tileDisplay1_OnDraw(object sender, EventArgs e)
        {
            GraphicsDevice.Clear(Color.CornflowerBlue);
        }

    }
}
```

Usually when you create a Windows forms application there is a using statement for the **System.Drawing** namespace. Some of the classes in that namespace will conflict with the XNA framework classes. Two in particular are the **Rectangle** and the **Color** classes that we will need in the rendering process. So I had to remove that namespace from the class. I will need the XNA framework for access to the **Rectangle** class for one and later I will need access to the **Vector2** class. Since I will be rendering I will need access to the **SpriteBatch** class as well as the **GraphicsDevice** class I will need the XNA Graphics namespace so there are using statements for the **Microsoft.Xna.Framework** and **Microsoft.Xna.Framework.Graphics**. Since I will eventually need a **SpriteBatch** object for rendering I added in a **SpriteBatch** object. The **TileDisplay** class has a **GraphicsDevice** object and to

make access to it easier I added in a get only property to get the **GraphicsDevice** object.

In the constructor I create the event handlers that will handle then intiailization of the of the **TileDisplay** control and the drawing of the **TileDisplay** control. When you are creating event handlers there is a short cut you can follow. After typing the object and the event handler when you type += you will be prompted to press the **Tab** key to insert the code for the event handler. Then you will prompted to press the **Tab** key to generate the code for the event handler. In the code for the generated event handler there will be a line of code that will throw an exception that the event handler has not been implemented. You can just delete that code and place you own code in it. You will have to add this code after the call to **InitializeComponent** because before that method call the controls do not exist and you will get an exception.

In the **OnInitialize** method I create the **SpriteBatch** object. In the **OnDraw** method all I do is call the **Clear** method of the **GraphicsDevice** to clear the graphics device. If you don't do that you will get some strange results in the **TileDisplay** control.

Well that is it for this tutorial. 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 these tutorials.