Creating a Role Playing Game with XNA Game Studio 3.0
Part 33
Tile Map Editor - Part 6
Using New Map In The Game

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/New2DRPG32.zip You can download the graphics from this link: Graphics.zip

I will be working on the editor again in this tutorial. The first thing you will want to do is download the new tile set. You can find the tile set at this link. I have included both the image and the **tset** file for the tile set. After you download and extract the files you can copy the **tileset1.tset** and **tilesetTexture1.png** files to the **TileSets** folder in the **Content** folder of the game. You can also right click the **TileSets** folder and select **Add Existing Item**, navigate to where you extracted the files and add them to the game.

To create a **tset** file right click the **TileSetGenerator** project and select **Set As StartUp Project**. Now press **F5** to build and run the project. What you will want to do is first click the **Load Image** button. Navigate to **TileSets** folder in your game and select the **tilesetTexture1.png** file. Now you will want to set the values in the numeric up and down controls. For this tile set the tiles are 128 pixels wide and 128 pixels high. This tile set is 8 tiles wide and 8 tiles high. You will now want to click the **Generate Tile Set** button to create the rectangles for the tile set. Now you will want to click the **Save Tile Set** button to save the tile set. Navigate to the **TileSets** folder in the **Content** folder of the game and select the **tileset1.tset** file.

Now it is time to start work on the editor. Right click the **TileMapEditor** project and select the **Set As StartUp Project** item. What I am going to do is create a small form to handle adding a new layer to the map. Click the **Project** item in the menu and select **Add Windows Form** to add a new form to the project. Name the form **FrmNewLayer**. Set the following properties for the form: **FormBorderStyle** is **FixedDialog**, **MaximizeBox** is **False**, **Size** is **246, 97**, **StartPostion** is **CenterParent** and **Text** is **New Layer**.

To the form I will be adding four controls: a **Label**, a **TextBox** and two **Buttons**. Drag a **Label** control onto the form and set these properties: **(Name)** to **lblLayerName**, **Location** to **3, 9** and **Text** to **Layer Name:**. Next you will drag a **TextBox** onto the form and set these properties: **(Name)** to **tbLayerName**, **Location** to **70, 6**, **Size** to**158, 20** and **TabIndex** to **1**. Drag a **Button** onto the form and set these properties: **(Name)** is **btnOK**, **Location** is **70, 32**, **TabIndex** is **2** and **Text** is **OK**. After dragging a second **Button** onto the form set the following properties: **(Name)** to **btnCancel**, **Location** to **153, 32**, **Tab Index** is **3** and **Text** is **Cancel**. Now click the title bar of the form and set property **AcceptButton** to **btnOK** and the property **CancelButton** to **btnCancel**. The finished form should look like this.

Right click **FrmNewLayer.cs** and select **View Code** to bring up the code for the form. I will give you the code for the form and then explain why I did what I did. This is the code.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace TileMapEditor
{
    public partial class FrmNewLayer : Form
    {
        public bool OKPressed = false;
        public string LayerName;

        public FrmNewLayer()
        {
            InitializeComponent();
            btnOK.Click += new EventHandler(btnOK_Click);
            btnCancel.Click += new EventHandler(btnCancel_Click);
        }

        void btnOK_Click(object sender, EventArgs e)
        {
            if (string.IsNullOrEmpty(tbLayerName.Text))
            {
                MessageBox.Show(
                    "You must enter a layer name",
                    "Error",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Error);
                return;
            }

            LayerName = tbLayerName.Text;
            OKPressed = true;
            this.Close();
        }

        void btnCancel_Click(object sender, EventArgs e)
        {
            OKPressed = false;
            this.Close();
        }

    }
```

```
}
```

This is a really simple form. I added two public fields to the form **OKPressed** and **LayerName**. As you probably guessed **OKPressed** will be set to true if the **OK** button was clicked. **LayerName** will hold the name for the layer. In the constructor of the form I create the event handlers for the **Click** event of the buttons. Again, you can save time by pressing the **Tab** key twice after typing the +=. The event handlers are both pretty simple. The **btnOK_Click** method checks to make sure there is text in **tbLayerName** using the **string.IsNullOrEmpty** method that will return true if the string is the empty string or if it is null. If that returns true I display an error message in a message box and exit the method. If it wasn't I set the **LayerName** field to the text in **tbLayerName**, set the **OKPressed** field to true and closes the form. The **btnCancel_Click method sets the OKPressed** field to false and closes the form.

Now right click **From1** in the solution explorer and choose the **View Code** option. I'm going to add in the ability to add new layers to the map, the ability to select which layer you want to work on and the ability to have layers visible. The first thing you will want to do is change the constructor to add in an event handler for when the **New Layer** menu item is clicked and when the user selects a new layer in the list of layers. I will do this in the constructor of the form. What I had done is just create the default event handlers for the **Click** event of **newLayerToolStripMenuItem** and for **clbLayers** I used the **SelectedIndexChanged** event handler. I will explain the handlers after you have seen 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);

    newLayerToolStripMenuItem.Click +=
        new EventHandler(newLayerToolStripMenuItem_Click);

    clbLayers.SelectedIndexChanged +=
        new EventHandler(clbLayers_SelectedIndexChanged);

    nudCurrentTile.ValueChanged +=
        new EventHandler(nudCurrentTile_ValueChanged);
    nudCurrentTile.InterceptArrowKeys = false;
```

```
    Application.Idle += new EventHandler(Application_Idle);
}

void newLayerToolStripMenuItem_Click(object sender, EventArgs e)
{
    FrmNewLayer frmNewLayer = new FrmNewLayer();

    frmNewLayer.ShowDialog();
    if (frmNewLayer.OKPressed)
    {
        TileMapLayer layer = new TileMapLayer(
            tileMap.MapWidth,
            tileMap.MapHeight);
        for (int x = 0; x < tileMap.MapWidth; x++)
            for (int y = 0; y < tileMap.MapHeight; y++)
                layer.SetTile(x, y, -1);

        tileMap.layers.Add(layer);
        currentLayer = tileMap.layers[tileMap.layers.Count - 1];
        clbLayers.Items.Add(frmNewLayer.LayerName, true);
        clbLayers.SelectedIndex = clbLayers.Items.Count - 1;
    }
}

void clbLayers_SelectedIndexChanged(object sender, EventArgs e)
{
    if (clbLayers.Items.Count != 0)
    {
        currentLayer = tileMap.layers[clbLayers.SelectedIndex];
    }
}
```

The **newLayerToolStripMenuItem_Click** method creates an instance of **FrmNewLayer** and displays it using the **ShowDialog** method so the program will wait until the user ends the dialog. It checks to see if the **OKPressed** field on the form was pressed. If it was it creates a new layer with the width and height of the map. It then sets all of the tiles to -1 so nothing will be drawn. It then adds the layer to the list of layers on the map. It then sets **currentLayer** to be the new layer. It then adds the **LayerName** filed to the items in **clbLayers** with true so the layer will be drawn. It also sets the **SelectedIndex** property of **clbLayers** to be the new layer. The **clbLayers_SelectedIndexChanged** method checks to make sure that there are layers in the map by checking the **Count** property of the **Items** collection of **clbLayers**. It then sets the **currentLayer** to be the selected layer in the map using the **SelectedIndex** property.

What I did next was work on controlling how the layers of the map are rendered to the display. All of this was done in the **Render** method. One thing I did do was extract the code for drawing the layer to a new method in the **Form1** class called **DrawLayer**. I also changed from using a foreach loop to a for loop for drawing the layers. The reason is that it will make things easier for knowing when to stop drawing layers and knowing if I should draw the layer. In the **Render** method the for loop goes from 0 to **clbLayers.SelectedIndex**. You will see that I used less than or equal to here instead of just less than. The reason is that you want to include the currently selected layer in the layers that you want to draw. For example if the first layer is selected, which would zero as the **SelectedIndex**, you would want to draw from zero up to but not including one. I also do not draw past the **SelectedIndex** property so you can see the layer that you are working on. I then call the **GetItemCheckState** method of **clbLayers** passing in **i**, the index of the for loop, and compare the result with **CheckState.Checked** which means that the selected item is checked and should be drawn. I then call the **DrawLayer** method

passing in **i.**

The code in the **DrawLayer** method should look familiar to you. There are two local variables in the method: **tile** and **tileRect**. The **tile** variable will hold which to tile to draw from the tile set. I decided to create one **Rectangle** object for rendering the layer instead of constantly creating them inside the nested for loops. I set **tileRect** to be a **Rectangle** with 0 for the **X** and **Y** values and for **Width** I use **Engine.TileWidth** and **Engine.TileHeight** for **Height**. Inside the loop I set **tile** to be the value of the **GetTile** method. Inside the nested for loop, where I actually draw the tile, I check to see if **tile** is not -1 which means it is a visible tile. If it is a visible tile I set the **X** and **Y** properties of **tileRect** and draw the tile.

```
private void Render()
{
    GraphicsDevice.Clear(Color.Black);
    if (tileMap != null)
    {
        for (int i = 0; i <= clbLayers.SelectedIndex; i++)
        {
            if (clbLayers.GetItemCheckState(i) == CheckState.Checked)
                DrawLayer(i);
        }
        DrawDisplay();
    }
}

private void DrawLayer(int layer)
{
    int tile;
    Rectangle tileRect = new Rectangle(
        0,
        0,
        Engine.TileWidth,
        Engine.TileHeight);

    spriteBatch.Begin(SpriteBlendMode.AlphaBlend);

    for (int y = 0; y < tileMap.MapHeight; y++)
        for (int x = 0; x < tileMap.MapWidth; x++)
        {
            tile = tileMap.layers[layer].GetTile(x, y);
            if (tile != -1)
            {
                tileRect.X = x * Engine.TileWidth
                    - (int)camera.Position.X;
                tileRect.Y = y * Engine.TileHeight
                    - (int)camera.Position.Y;
                spriteBatch.Draw(texture,
                        tileRect,
                        tileset.tiles[tile],
                        Color.White);
            }
        }

    spriteBatch.End();
}
```

What I will also add to the tutorial is being able to make see if the map has been changed and

then the user tries to close the form the editor will ask if you want to quit. I will use a bool variable called **isChanged** to add in this behavior. Add this field to the **Form1** class.

```
bool isChanged = false;
```

There is an event handler for a form that is triggered when the form is trying to close. This event can be canceled in the event handler. What I will do is check to see if the **isChanged** field is true. If it is I will show a message box with a **Yes**, **No** and **Cancel** button and check the result. I will create the event handler in the **Form1** class. This is the code for the updated constructor of **Form1** as well as the event handler to handle if the form is closing.

```
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);

    newLayerToolStripMenuItem.Click +=
        new EventHandler(newLayerToolStripMenuItem_Click);

    clbLayers.SelectedIndexChanged +=
        new EventHandler(clbLayers_SelectedIndexChanged);

    nudCurrentTile.ValueChanged +=
        new EventHandler(nudCurrentTile_ValueChanged);
    nudCurrentTile.InterceptArrowKeys = false;

    Application.Idle += new EventHandler(Application_Idle);

    this.FormClosing +=
        new FormClosingEventHandler(Form1_FormClosing);
}

void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
    e.Cancel = true;
    if (isChanged)
    {
        DialogResult result = MessageBox.Show(
```

```
            "The map has changed. Are you sure you want to close?",
            "Save map?",
            MessageBoxButtons.YesNoCancel,
            MessageBoxIcon.Question);

        if (result == DialogResult.Yes)
            e.Cancel = false;
    }
}
```

To control whether or not to cancel the event you use **e.Cancel**. If it is set to true the event will be canceled. Other wise the event will not be canceled. I then check to see if **isChanged** is true which means the map has been changed. If it is I display a message box with **Yes**, **No** and **Cancel** buttons. I then check to see if the result of the message box was the **Yes** button. If it was the user did not want to save the map and I set **e.Cancel** to false to cancel the event.

That is all good but it doesn't tell if the map was changed or not. That will require a little more work. You have to figure out when a map has been changed. One instance is when a new map is created so you will have to handle that. Another is when there is a map in the editor and the user presses the space key to paint a tile. The last one is if a new layer is added to a map. I will also, if saving the map is successful, set **isChanged** to false meaning the map has not been changed. I will also check to make sure if the user asks to open a map or create a new map that there were no changes to the current map.

The first one that I will work on is if the user draws or erases a tile on the display. You will handle that in the **Logic** method. What you would do is if the user had pressed the space bar to draw or erase a tile is set **isChecked** to true. This is the code for the **Logic** method.

```
private void Logic()
{
    tileDisplay1.Focus();
    if (tileMap != null)
    {
        keyState = Keyboard.GetState();
        if (CheckKey(XKeys.Right))
        {
            position.X += Engine.TileWidth;
        }
        else if (CheckKey(XKeys.Down))
        {
            position.Y += Engine.TileHeight;
        }
        else if (CheckKey(XKeys.Left))
        {
            position.X -= Engine.TileWidth;
        }
        else if (CheckKey(XKeys.Up))
        {
            position.Y -= Engine.TileHeight;
        }
        LockCursor();
        int tileX = (int)position.X / Engine.TileWidth;
        int tileY = (int)position.Y / Engine.TileHeight;

        tbCursorPosition.Text = "( " + tileX.ToString() + ", ";
        tbCursorPosition.Text += tileY.ToString() + " )";
        tbCursorPosition.Invalidate();
```

```
        if (CheckKey(XKeys.Space))
        {
            isChanged = true;
            if (rbDraw.Checked == true)
                currentLayer.SetTile(
                    tileX,
                    tileY,
                    (int)nudCurrentTile.Value);
            if (rbErase.Checked == true)
                currentLayer.SetTile(
                    tileX,
                    tileY,
                    -1);
        }
        oldKeyState = keyState;
    }
}
```

The easiest method to change next would be the **newLayerToolStripMenuItem_Click** method. What you would want to do is the user successfully added a new layer to the map is set **isChanged** to true. This is the new method.

```
void newLayerToolStripMenuItem_Click(object sender, EventArgs e)
{
    FrmNewLayer frmNewLayer = new FrmNewLayer();

    frmNewLayer.ShowDialog();
    if (frmNewLayer.OKPressed)
    {
        TileMapLayer layer = new TileMapLayer(
            tileMap.MapWidth,
            tileMap.MapHeight);
        for (int x = 0; x < tileMap.MapWidth; x++)
            for (int y = 0; y < tileMap.MapHeight; y++)
                layer.SetTile(x, y, -1);

        tileMap.layers.Add(layer);
        currentLayer = tileMap.layers[tileMap.layers.Count - 1];
        clbLayers.Items.Add(frmNewLayer.LayerName, true);
        clbLayers.SelectedIndex = clbLayers.Items.Count - 1;
        isChanged = true;
    }
}
```

The next thing I will do is add in changing **isChaged** to true if the user creates a new map. That will be handled in the **newMapToolStripMenuItem_Click** method. I will explain the code after you have seen it. This is the code for the method.

```
void newMapToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (isChanged)
    {
        DialogResult result = MessageBox.Show(
            "The map has changed. Are you sure you want to continue?",
            "Save map?",
            MessageBoxButtons.YesNo,
            MessageBoxIcon.Question);
```

```
        if (result == DialogResult.No)
            return;
    }

    frmNewMap.ShowDialog();

    if (frmNewMap.OkClicked)
    {
        tileset = frmNewMap.tileset;
        texture = Texture2D.FromFile(GraphicsDevice,
            frmNewMap.TilesetTextureName);
        tilesetBitmap = Bitmap.FromFile(frmNewMap.TilesetTextureName);
        tileMap = new TileMap(frmNewMap.MapWidth, frmNewMap.MapHeight);
        layerToolStripMenuItem.Enabled = true;
        saveMapToolStripMenuItem.Enabled = true;

        clbLayers.Items.Clear();
        clbLayers.Items.Add(frmNewMap.TileLayerName, true);
        clbLayers.SelectedIndex = 0;
        clbLayers.SelectionMode = SelectionMode.One;
        currentLayer = tileMap.layers[0];

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

What I do is check to see if **isChanged** is true. If it is I display a message box informing the user that the map has changed and ask if they want to continue with a **Yes** and **No** button. If the **No** button was pressed I exit the method using a return statement. Then in if the if statement where I check to see if the user hit the **OK** button at the bottom of the method I set **isChanged** to true meaning that there is now a new map in the editor and it has not been saved.

Changing it so that when the use opens a new map and there were changes to the previous map is similar to what was done above and is done in the **openMapToolStripMenuItem_Click** method. What you would do is check to see if **isChanged** is true. If it is true then display a message box asking if they want to continue with **Yes** and **No** buttons. If they choose the **No** button then exit the method. This is the code for the updated **openMapToolStripMenuItem_Click** method.

```
void openMapToolStripMenuItem_Click(object sender, EventArgs e)
{
    DialogResult saveChanges;
    if (isChanged)
    {
         saveChanges = MessageBox.Show(
            "The map has changed. Are you sure you want to continue?",
            "Save map?",
            MessageBoxButtons.YesNo,
            MessageBoxIcon.Question);

        if (saveChanges == DialogResult.No)
            return;
    }
```

```
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();
}
```

Well that is it for this tutorial. 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.