Eyes of the Dragon - XNA

Part 37

Map Editor Revisited

I'm writing these tutorials for the XNA 4.0 framework. Even though Microsoft has ended support for XNA it still runs on all supported operating systems and is an excellent learning tool. The code can also be ported over to MonoGame relatively easily and that is being supported by the MonoGame community.

The tutorials will make more sense if they are read in order. You can find the list of tutorials on the **Eyes of the Dragon** tutorials page of my web site. I will be making my version of the project available for download at the end of each tutorial. It will be included on the page that links to the tutorials. The solution is still in Visual Studio 2010 but there is no reason that if you have a later version configured correctly that you can upgrade it to one of those versions.

In this tutorial I will be updating the map editor to add in some of the new items that have been added to the game, such as animated tiles. Fire up your solution in Visual Studio. Right click the XLevlEditor project, right click it and select Set As StartUp Project.

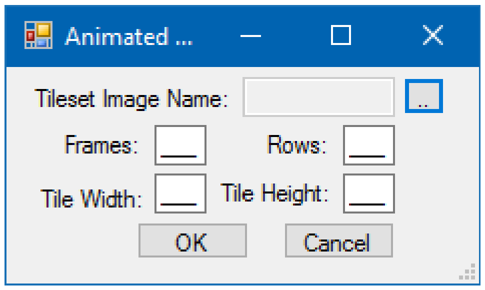
The first thing that I'm going to tackle is adding a few new items to the main map editor form. Double click the FormMain form to bring it up in design view. Now, select the Tileset menu item in the designer. Place your cursor in the Type Here box and enter &Animated Tileset. Select the tab control on the right side of the page. In the properties window for the control open the Tab Pages collection. Click the Add button to add a new tab. In the properties for the new tab set its (Name) property to tabAnimated and its text to Animated Tiles. Using the up and down arrows place this tab as the second tab in the list. Now add another new tab with the (Name) property of tabCollisions and the Text property Collision Layer. Place this tab as the fourth tab in the list.

Now, close the tab collection window. In the properties tab select the Animated Tiles tab. Drag a check box onto the tab, then a numeric up down box and a picture box onto the tab. Position the check box at the top of the tab. Set its (Name) property to chkPaintAnimated and its Text property to Paint animated tile. Place the numeric up down under the check box. Set its (Name) property to sbAnimatedTile and the Minimum, Maximum and Value properties to -1. Set the (Name) property of the picture box to pbAnimatedSet. Click on the Anchor property and select all four options. Set its SizeMode property to StretchImage. Now drag and position the box so that it takes up the rest of the tab.

Select the Collision Layer tab now. Drag a check box onto the tab and then a group box. Onto the group box drag two radio buttons. Change the (Name) property of the check box to chkPaintCollision and the Text property to Paint collisions. Change the Text property of the group box to Collision Types.

Change the (Name) property of the first radio button to rbNoCollision, the Text property to No Collision and the Checked property to True. Set those same properties for the second radio button to rbImpassable, Impassable and False.

I need to add a new form to the project to create an animated tile set and add it to the map. My finished form looked like this.



Set the Text property of the form to Animated Tileset and set the Text properties of the labels as seen in the image. For the text box for the image name set the (Name) property to tbTilesetImage and the Enabled property to False. For the top button set the (Name) property to btnSelectImage and the Text property to .... The other four boxes are MaskedTextBoxes. What I did is created the first one and set its properties. I then copied and pasted the other three. All of them have a the Mask property 000. Their (Name) properties are tbFrames, tbRows, tbTileWidth and tbTileHeight. The OK button has (Name) btnOK and Text OK. The Cancel button has (Name) btnCancel and Text Cancel. Open the code for this form and replace it with the following.

```
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;
using RpgLibrary.WorldClasses;

namespace XLevelEditor
{
    public partial class FormNewAnimatedTileset : Form
    {
        public AnimatedTilesetData Tileset { get; set; }

        public FormNewAnimatedTileset()
        {
            InitializeComponent();

            btnSelectImage.Click += new EventHandler(btnSelectImage_Click);
            btnOK.Click += new EventHandler(btnOK_Click);
            btnCancel.Click += new EventHandler(btnCancel_Click);
        }

        void btnSelectImage_Click(object sender, EventArgs e)
        {
            OpenFileDialog ofDialog = new OpenFileDialog();

            ofDialog.Filter = "Image Files|*.BMP;*.GIF;*.JPG;*.TGA;*.PNG";
            ofDialog.CheckFileExists = true;
            ofDialog.CheckPathExists = true;
            ofDialog.Multiselect = false;

            DialogResult result = ofDialog.ShowDialog();

            if (result == DialogResult.OK)
            {
                tbTilesetImage.Text = ofDialog.FileName;
            }
        }

        void btnOK_Click(object sender, EventArgs e)
        {
            if (string.IsNullOrEmpty(tbTilesetImage.Text))
            {
                MessageBox.Show("You must select an image for the tileset in order to 
proceed.");
                return;
            }

            int tileWidth = 0;
            int tileHeight = 0;
            int frames = 0;
            int rows = 0;

            if (!int.TryParse(tbTileWidth.Text, out tileWidth))
            {
                MessageBox.Show("Tile width must be an integer value.");
                return;
            }
            else if (tileWidth < 1)
            {
                MessageBox.Show("Tile width must me greater than zero.");
                return;
            }

            if (!int.TryParse(tbTileHeight.Text, out tileHeight))
            {
                MessageBox.Show("Tile height must be an integer value.");
                return;
            }
            else if (tileHeight < 1)
            {
                MessageBox.Show("Tile height must be greater than zero.");
                return;
            }

            if (!int.TryParse(tbFrames.Text, out frames))
            {
                MessageBox.Show("Frames must be an integer value.");
                return;
            }
            else if (frames < 1)
            {
                MessageBox.Show("Frames must me greater than zero.");
                return;
            }

            if (!int.TryParse(tbRows.Text, out rows))
            {
                MessageBox.Show("Rows must be an integer value.");
                return;
            }
            else if (rows < 1)
            {
                MessageBox.Show("Rows must be greater than zero.");
                return;
            }

            Tileset = new AnimatedTilesetData();
            Tileset.TilesetName = tbTilesetImage.Text;
            Tileset.TilesetImageName = tbTilesetImage.Text;
            Tileset.FramesAcross = frames;
            Tileset.TilesHigh = rows;
            Tileset.TileWidthInPixels = tileWidth;
            Tileset.TileHeightInPixels = tileHeight;

            this.Close();
        }

        void btnCancel_Click(object sender, EventArgs e)
        {
            Tileset = null;
            this.Close();
        }
    }
}
```

This is a very straight forward code behind for the form. There is a property that exposes an AnimatedTilesetData to be returned to the main form. The constructor wires the event handlers for the buttons. The event handler for the button to choose the image displays an OpenFileDialog to allow the user to select the image. If an image is selected the text property of the text box for the name is set to that value. The event handler for the OK button does some basic validation on the form to make sure all values have been filled out with appropriate values. If one of them isn't it displays a message and exits the handler. Once everything is validated I create a new AnimatedTilesetData using the values and assign it to the property to return the value back to the main form and then closes the form. The cancel button sets the AniamtedTilesetData property to null and closes the form.

I need to add a method to the TileMap class that allows me to set the AnimatedTileset for the map because the only way to set it is through the constructor. You could also make it public or expose it as get and set through a property. Add this method to the TileMap class.

```
public void AddAnimatedTileset(AnimatedTileset tileset)
{
    animatedSet = tileset;
}
```

The next thing that I will do is update the constructor to wire some event handlers in the main form. What I want to do is add a handler for the new menu item and the check event of the check boxes for painting collisions or animated tiles. I also set them to be disabled, temporarily, until it makes sense to be able to use them. They are also mutually exclusive. If the collision one is selected the animated one should be deselected, and the same in reverse. Update the constructor for FormMain and add these event handlers.

```
public FormMain()
{
    InitializeComponent();

    this.Load += new EventHandler(FormMain_Load);
    this.FormClosing += new FormClosingEventHandler(FormMain_FormClosing);

    tilesetToolStripMenuItem.Enabled = false;
    mapLayerToolStripMenuItem.Enabled = false;
    charactersToolStripMenuItem.Enabled = false;
    chestsToolStripMenuItem.Enabled = false;
    keysToolStripMenuItem.Enabled = false;
    chkPaintAnimated.Enabled = false;
    chkPaintCollision.Enabled = false;

    newLevelToolStripMenuItem.Click += new EventHandler(newLevelToolStripMenuItem_Click);
    newTilesetToolStripMenuItem.Click += new
EventHandler(newTilesetToolStripMenuItem_Click);
    newLayerToolStripMenuItem.Click += new EventHandler(newLayerToolStripMenuItem_Click);

    saveLevelToolStripMenuItem.Click += new EventHandler(saveLevelToolStripMenuItem_Click);

    openLevelToolStripMenuItem.Click += new EventHandler(openLevelToolStripMenuItem_Click);

    mapDisplay.OnInitialize += new EventHandler(mapDisplay_OnInitialize);
    mapDisplay.OnDraw += new EventHandler(mapDisplay_OnDraw);

    x1ToolStripMenuItem.Click += new EventHandler(x1ToolStripMenuItem_Click);
    x2ToolStripMenuItem.Click += new EventHandler(x2ToolStripMenuItem_Click);
    x4ToolStripMenuItem.Click += new EventHandler(x4ToolStripMenuItem_Click);
    x8ToolStripMenuItem.Click += new EventHandler(x8ToolStripMenuItem_Click);

    blackToolStripMenuItem.Click += new EventHandler(blackToolStripMenuItem_Click);
    blueToolStripMenuItem.Click += new EventHandler(blueToolStripMenuItem_Click);
    redToolStripMenuItem.Click += new EventHandler(redToolStripMenuItem_Click);
    greenToolStripMenuItem.Click += new EventHandler(greenToolStripMenuItem_Click);
    yellowToolStripMenuItem.Click += new EventHandler(yellowToolStripMenuItem_Click);
    whiteToolStripMenuItem.Click += new EventHandler(whiteToolStripMenuItem_Click);

    saveTilesetToolStripMenuItem.Click += new
EventHandler(saveTilesetToolStripMenuItem_Click);
    saveLayerToolStripMenuItem.Click += new EventHandler(saveLayerToolStripMenuItem_Click);

    openTilesetToolStripMenuItem.Click += new
EventHandler(openTilesetToolStripMenuItem_Click);
    openLayerToolStripMenuItem.Click += new EventHandler(openLayerToolStripMenuItem_Click);

    animatedTIlesetToolStripMenuItem.Click += new
EventHandler(animatedTIlesetToolStripMenuItem_Click);

    chkPaintAnimated.CheckChanged += new EventHandler(chkPaintAnimated_CheckChanged);
    chkPaintCollision.CheckChanged += new EventHandler(chkPaintCollision_CheckChanged);
}

private void animatedTIlesetToolStripMenuItem_Click(object sender, EventArgs e)
{
    FormNewAnimatedTileset frm = new FormNewAnimatedTileset();
    frm.ShowDialog();

    if (frm.Tileset == null)
        return;

    animatedSetData = frm.Tileset;

    try
    {
        GDIImage image = (GDIImage)GDIBitmap.FromFile(data.TilesetImageName);
        pbAnimatedSet.Image = image;

        Stream stream = new FileStream(data.TilesetImageName, FileMode.Open,
FileAccess.Read);

        Texture2D texture = Texture2D.FromStream(GraphicsDevice, stream);
        animatedSet = new AnimatedTileset(
                texture,
                animatedSetData.FramesAcross,
                animatedSetData.TilesHigh,
                animatedSetData.TileWidthInPixels,
                animatedSetData.TileHeightInPixels);

        if (map != null)
            map.AddAnimatedTileset(animatedSet);

        chkPaintAnimated.Enabled = true;
        stream.Close();
        stream.Dispose();
    }
    catch (Exception ex)
    {
        MessageBox.Show("Error reading file.\n" + ex.Message, "Error reading image");
        return;
    }
}

void chkPaintCollision_CheckChanged(object sender, EventArgs e)
{
    if (chkPaintCollision.Checked)
    {
        chkPaintAnimated.Checked = false;
    }
}

void chkPaintAnimated_CheckChanged(object sender, EventArgs e)
{
    if (chkPaintAnimated.Checked)
    {
        chkPaintCollision.Checked = false;
    }
}
```

The code for the constructor is pretty straight forward. All it does is set properties and wire the new event handlers. The event handlers for the paint check boxes are basically the same. It checks if that control is not checked. If it is not checked it checks it and then deselects the other check box.

The code for handling the dialog should be pretty familiar at this point. First, I create an instance of the form for creating animated tile sets and display it. If the dialog was cancelled I exit the method.

Next up there is a try-catch that attempts to read the texture and the image from the data. It also creates a new animated tile set. If a map exists then I call the AddAnimatedTileset method we just added to set the tile set to that object. Finally I enable the paint animated tiles check box. If there is a failure doing any of these tasks I show a message box and exit the method.

The next step will be to enable the check boxes if there is a valid map object and for the animated tile set that an animated tile set exists. I did that in the newLayerToolStripMenuItem_Click method. Update that code to the following.

```
void newLayerToolStripMenuItem_Click(object sender, EventArgs e)
{
    using (FormNewLayer frmNewLayer = new FormNewLayer(levelData.MapWidth,
levelData.MapHeight))
    {
        frmNewLayer.ShowDialog();
        if (frmNewLayer.OKPressed)
        {
            MapLayerData data = frmNewLayer.MapLayerData;

            if (clbLayers.Items.Contains(data.MapLayerName))
            {
                MessageBox.Show("Layer with name " + data.MapLayerName + " exists.",
"Existing layer");
                return;
            }

            MapLayer layer = MapLayer.FromMapLayerData(data);
            clbLayers.Items.Add(data.MapLayerName, true);
            clbLayers.SelectedIndex = clbLayers.Items.Count - 1;

            layers.Add(layer);

            if (map == null)
            {
                map = new TileMap(tileSets[0], animatedSet, (MapLayer)layers[0]);

                for (int i = 1; i < tileSets.Count; i++)
                    map.AddTileset(tileSets[1]);

                for (int i = 1; i < layers.Count; i++)
                    map.AddLayer(layers[1]);
            }

            charactersToolStripMenuItem.Enabled = true;
            chestsToolStripMenuItem.Enabled = true;
            keysToolStripMenuItem.Enabled = true;

            if (animatedSet != null)
                chkPaintAnimated.Enabled = true;

            chkPaintCollision.Enabled = true;
        }
    }
}
```

The next step will be to handle painting collisions and animated tiles. With the editor. To do that I updated the Logic method and added two new methods, PaintAnimated and PaintCollision which paint an animated tile and a collision type. Update the Logic method to the following and add the new methods.

```
private void Logic()
{
    if (layers.Count == 0)
        return;

    Vector2 position = camera.Position;

    if (trackMouse)
    {
        if (frameCount == 0)
        {
            if (mouse.X < Engine.TileWidth)
                position.X -= Engine.TileWidth;

            if (mouse.X > mapDisplay.Width - Engine.TileWidth)
                position.X += Engine.TileWidth;
            if (mouse.Y < Engine.TileHeight)
                position.Y -= Engine.TileHeight;

            if (mouse.Y > mapDisplay.Height - Engine.TileHeight)
                position.Y += Engine.TileHeight;

            camera.Position = position;
            camera.LockCamera();
        }

        position.X = mouse.X + camera.Position.X;
        position.Y = mouse.Y + camera.Position.Y;

        Point tile = Engine.VectorToCell(position);
        shadowPosition = new Vector2(tile.X, tile.Y);

        tbMapLocation.Text =
            "( " + tile.X.ToString() + ", " + tile.Y.ToString() + " )";

        if (isMouseDown && !chkPaintAnimated.Checked && !chkPaintCollision.Checked)
        {
            if (rbDraw.Checked)
                SetTiles(tile, (int)nudCurrentTile.Value, lbTileset.SelectedIndex);

            if (rbErase.Checked)
                SetTiles(tile, -1, -1);
        }
        else if (isMouseDown && chkPaintAnimated.Checked)
        {
            PaintAnimated(tile);
        }
        else if (isMouseDown && chkPaintCollision.Checked)
        {
            CollisionType cType = CollisionType.Passable;

            if (rbImpassable.Checked)
                cType = CollisionType.Impassable;

            PaintCollision(tile, cType);
        }
    }
}

private void PaintAnimated(Point tile)
{
    if ((int)sbAnimatedTile.Value == -1 &&
map.AnimatedTileLayer.AnimatedTiles.ContainsKey(tile))
    {
        map.AnimatedTileLayer.AnimatedTiles.Remove(tile);
        return;
    }

    if (map.AnimatedTileLayer.AnimatedTiles.ContainsKey(tile))
        map.AnimatedTileLayer.AnimatedTiles[tile].TileIndex = (int)sbAnimatedTile.Value;
    else
        map.AnimatedTileLayer.AnimatedTiles.Add(tile, new
AnimatedTile((int)sbAnimatedTile.Value, animatedSet.FrameCount));
}

private void PaintCollision(Point tile, CollisionType collisionValue)
{
    if (collisionValue == CollisionType.Passable &&
map.CollisionLayer.Collisions.ContainsKey(tile))
    {
        map.CollisionLayer.Collisions.Remove(tile);
        return;
    }

    if (map.CollisionLayer.Collisions.ContainsKey(tile))
        map.CollisionLayer.Collisions[tile] = collisionValue;
    else
        map.CollisionLayer.Collisions.Add(tile, collisionValue);
}
```

In the logic method I added in two else if statements after I check the isMouseDown. The first one checks if isMouseDown is selected and chkPaintAnimated is checked. If those conditions are true I call the PaintAnimated passing in the tile. The second else if checks if isMouseDown is true and chkPaintCollision is checked. In that case I create a CollisionType variable and set it to be Passable. Next there is an if statement that checks if rbImpassable is checked. If it is I update the variable to be Impassable. I then call PaintCollision passing in the tile and the type of collision.

In PaintAnimated I first check to see if the selected value in the numeric up down for the animated tile is -1 and then if an animated tile exists with the tile as a key. If both are true I remove the tile and exit the method. I then check if key exists for the tile. If it does I set the TileIndex property to the value of the numeric up down. If it does not exist I add a new item to the collection.

The PaintCollision method works similarly. I check to see if the collision type is passable and if the collision collection contains that key. If it does I remove the entry and exit the method. Then if the tile exists as a key update the value or if it does not add a new collision to the collection.

That leaves rendering the animated tiles and a visual cue that a tile has a collision associated with it. That will be done in the Render method. Update that method as follows and add the new method PaintCollision.

```
private void Render()
{
    for (int i = 0; i < layers.Count; i++)
    {
        spriteBatch.Begin(
            SpriteSortMode.Deferred,
            BlendState.AlphaBlend,
            SamplerState.PointClamp,
            null,
            null,
            null,
            camera.Transformation);

        if (clbLayers.GetItemChecked(i))
            layers[i].Draw(spriteBatch, camera, tileSets);

        Rectangle destination = new Rectangle(
            (int)shadowPosition.X * Engine.TileWidth,
            (int)shadowPosition.Y * Engine.TileHeight,
            brushWidth * Engine.TileWidth,
            brushWidth * Engine.TileHeight);

        Color tint = Color.White;
        tint.A = 1;

        if (animatedSet != null && map.AnimatedTileLayer.AnimatedTiles.Count > 0)
            map.AnimatedTileLayer.Draw(spriteBatch, animatedSet);

        if (map.CollisionLayer.Collisions.Count > 0)
```

```
            DrawCollisions();

        spriteBatch.Draw(shadow, destination, tint);
        spriteBatch.End();
    }

    DrawDisplay();
}

private void DrawCollisions()
{
    Color lowAlpha = Color.White;
    Texture2D collisionShadow = new Texture2D(GraphicsDevice, 32, 32, false,
SurfaceFormat.Color);

    Color[] data = new Color[collisionShadow.Width * collisionShadow.Height];
    Color tint = Color.White;

    for (int i = 0; i < collisionShadow.Width * collisionShadow.Height; i++)
        data[i] = tint;

    collisionShadow.SetData<Color>(data);

    foreach (Point p in map.CollisionLayer.Collisions.Keys)
    {
        Rectangle r = new Rectangle(p.X * Engine.TileWidth, p.Y * Engine.TileHeight,
Engine.TileWidth, Engine.TileHeight);

        if (map.CollisionLayer.Collisions[p] == CollisionType.Impassable)
        {
            lowAlpha = Color.Red;
            lowAlpha.A = 5;
        }

        spriteBatch.Draw(collisionShadow, r, lowAlpha);
    }
}
```

The first change to the Render method was after rendering the layers was to check that the animatedSet member variable is not null, which means the user created an animated tile set, and that there are animated tiles to be drawn. If both are true I call the Draw method of the animated tile layer. There is then another if statement that checks if there have been any tile collisions added. If there were I call the new method DrawCollisions.

In the DrawCollisions method I have a local variable, lowAlpha, that will be used to draw the collision type. I then create a new Texture2D that will be used to draw the collision. I create an array of Color that will hold the color data for the texture. I set that color that will be used to White. I then fill the array with the color and then fill the texture with that array.

There is then a foreach loop where I cycle over the key collection for the collisions. Inside I create a rectangle object using the key. There is then an if statement that checks what type of collision there is. I did that in the case that you add another type of collision type. Inside the if statement I set the tint color to red and half transparent. This works because the texture was created as solid white. That means whatever color I use in the call to draw the white will become that color. I use this often when I need a texture that is the same, other than the color. I draw it in shades of white and grey and tint it with a color. This allows for reuse of the base image and less assets.

If you were to build and run now you can place animated tiles and have them show up, but not update. Also, the tiles that have a collision associated with them will be drawn in semi-transparent red color.

I'm going to update the method for saving the level. What I've done is pass in the actual animated tile layer and collision layer to the MapData constructor. That save those new layers to disk. Update the saveLevelToolStripMenuItem_Click method to the following.

```
void saveLevelToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (map == null)
        return;

    List<MapLayerData> mapLayerData = new List<MapLayerData>();

    for (int i = 0; i < clbLayers.Items.Count; i++)
    {
        if (layers[i] is MapLayer)
        {
            MapLayerData data = new MapLayerData(
                clbLayers.Items[i].ToString(),
                ((MapLayer)layers[i]).Width,
                ((MapLayer)layers[i]).Height);

            for (int y = 0; y < ((MapLayer)layers[i]).Height; y++)
                for (int x = 0; x < ((MapLayer)layers[i]).Width; x++)
                    data.SetTile(
                        x,
                        y,
                        ((MapLayer)layers[i]).GetTile(x, y).TileIndex,
                        ((MapLayer)layers[i]).GetTile(x, y).Tileset);

            mapLayerData.Add(data);
        }
    }

    MapData mapData = new MapData(levelData.MapName, tileSetData, animatedSetData,
mapLayerData, map.CollisionLayer, map.AnimatedTileLayer);

    FolderBrowserDialog fbDialog = new FolderBrowserDialog();

    fbDialog.Description = "Select Game Folder";
    fbDialog.SelectedPath = Application.StartupPath;

    DialogResult result = fbDialog.ShowDialog();

    if (result == DialogResult.OK)
    {
        if (!File.Exists(fbDialog.SelectedPath + @"\Game.xml"))
        {
            MessageBox.Show("Game not found", "Error");
            return;
        }

        string LevelPath = Path.Combine(fbDialog.SelectedPath, @"Levels\");
        string MapPath = Path.Combine(LevelPath, @"Maps\");

        if (!Directory.Exists(LevelPath))
            Directory.CreateDirectory(LevelPath);

        if (!Directory.Exists(MapPath))
            Directory.CreateDirectory(MapPath);
        XnaSerializer.Serialize<LevelData>(LevelPath + levelData.LevelName + ".xml",
levelData);
        XnaSerializer.Serialize<MapData>(MapPath + mapData.MapName + ".xml", mapData);
    }
}
```

There is one quick fix that I need to make and that is add a private constructor to the AnimatedTile class that requires no parameters. Add this constructor to the AnimatedTile class in the AnimatedTileset file.

```
private AnimatedTile()
{
}
```

You can now run the editor and be able to create your maps, save them and then load them back in to the editor. I'm going to wrap up the tutorial here because I'd like to try and keep the tutorials to a reasonable length so that you don't have too much to digest at once. In the next tutorial I will cover adding characters to the level and likely loading the maps into the game rather than generating maps by hand.

So, I encourage you to visit my site, Game Programming Adventures , for the latest news on my tutorials or subscribe to my weekly newsletter. Use the Sign Up button the right side of the page to subscribe.

Good luck in your game programming adventures!
Jamie McMahon