# Eyes of the Dragon Tutorials
# Part 24
# Level Editor – Part Two

I'm writing these tutorials for the MonoGame 3.8 framework using Visual Studio 2019. 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 page of my web blog. I will be making each version of the project available on GitHub here. It will be included on the page that links to the tutorials.

In tutorial 23 I got the basics of a level editor up and running. You could create new levels that included the map. You could add tilesets to the map as well as layers to the map. In this tutorial I'm going to do a little more work on the editor.

The first thing I want to do is to modify the design of the **Tiles** tab. I want to display the tile the mouse is over. The first step is to modify the properties of **lbTileset**. I want to make it shorter. Set the following properties:

**lbTileset**

| Property | Value |
|---|---|
| Size | 180, 212 |

Below that I dragged on a **Label** and a **Text Box**. Set these properties for those two controls.

**Label**

| Property | Value |
|---|---|
| (Name) | lblCursor |
| AutoSize | FALSE |
| Location | 8, 567 |
| Size | 180, 23 |
| Text | Map Location |
| TextAlign | TopCenter |

**Text Box**

| Property | Value |
|---|---|
| (Name) | tbMapLocation |
| Enabled | FALSE |
| Location | 8, 590 |
| Size | 180, 22 |

You can now set the **Text** property of **tbMapLocation** in the **Logic** method. Update the **Logic** method to the following.

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

```
    Vector2 position = camera.Position;

    if (trackMouse)
    {
        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);

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

        if (isMouseDown)
        {
            if (rbDraw.Checked)
            {
                layers[clbLayers.SelectedIndex].SetTile(
                    tile.X,
                    tile.Y,
                    (int)nudCurrentTile.Value,
                    lbTileset.SelectedIndex);
            }

            if (rbErase.Checked)
            {
                layers[clbLayers.SelectedIndex].SetTile(
                    tile.X,
                    tile.Y,
                    -1,
                    -1);
            }
        }
    }
}
```

All I did was create a string using the **X** and **Y** properties of the **Point** called **tile** that is the current tile the mouse is in. You could set the **TextAlign** property of **tbMapLocation** to center the text if you want. Choosing the tile you want to draw using the **Numeric Up Down** can be a real pain in the you know what. What would be nicer is if you could also click on the **Picture Box** that holds the tileset image and select that tile. Part of the problem with that is that the tileset image in the **Picture Box** has been scaled.

Unless the tileset is the same size as the **Picture Box** you need to do a little math. Update the code for the **Load** event of the form to wire an event handler for the **MouseDown** event for **pbTilesetPreview** and add in this handler to the **Tile Tab Event Handler**. I also set the **TextAlign** property for **tbMapLocation** to center the text. You want **MouseDown** because it uses **MouseEventArgs** that has information about the mouse unlike the **Click** event.

```
void FormMain_Load(object sender, EventArgs e)
{
    lbTileset.SelectedIndexChanged += new EventHandler(lbTileset_SelectedIndexChanged);
    nudCurrentTile.ValueChanged += new EventHandler(nudCurrentTile_ValueChanged);

    Rectangle viewPort = new Rectangle(0, 0, mapDisplay.Width, mapDisplay.Height);

    camera = new Camera(viewPort);
    engine = new Engine(32, 32);

    controlTimer.Tick += new EventHandler(controlTimer_Tick);
    controlTimer.Enabled = true;
    controlTimer.Interval = 200;

    tbMapLocation.TextAlign = HorizontalAlignment.Center;

    pbTilesetPreview.MouseDown += new MouseEventHandler(pbTilesetPreview_MouseDown);
}

void pbTilesetPreview_MouseDown(object sender, MouseEventArgs e)
{
    if (lbTileset.Items.Count == 0)
        return;

    if (e.Button != System.Windows.Forms.MouseButtons.Left)
        return;

    int index = lbTileset.SelectedIndex;

    float xScale = (float)tileSetImages[index].Width /
        pbTilesetPreview.Width;
    float yScale = (float)tileSetImages[index].Height /
        pbTilesetPreview.Height;

    Point previewPoint = new Point(e.X, e.Y);

    Point tilesetPoint = new Point(
        (int)(previewPoint.X * xScale),
        (int)(previewPoint.Y * yScale));

    Point tile = new Point(
        tilesetPoint.X / tileSets[index].TileWidth,
        tilesetPoint.Y / tileSets[index].TileHeight);

    nudCurrentTile.Value = tile.Y * tileSets[index].TilesWide + tile.X;
}
```

Not much new in the event handler for the form loading. I just wire the handler and set a property for **tbMapLocation**. It is in the handler for **MouseDown** of **pbTilesetPreview** where the interesting code is. First, I check to see that there is a tileset loading comparing the **Count** property of the **Items** collection of **lbTileset** to zero. If there is no tileset I exit the method. Also, if the left mouse button did

not trigger the event I exit the method. I capture the **SelectedIndex** property of **lbTileset** because I needed to use it a lot. As I mentioned if the size of the tileset image is not the size of the image of the picture box there was scaling done. To get the scaling factor for the X coordinate, I take the **Width** of the tileset, cast to a float, divided by the **Width** of the picture box. If you don't cast the width of the tileset to a float first you will be doing integer division and you want the digits after the decimal place. The scaling factor for the Y coordinate is done using the **Height** of the tileset and picture box. I then get the location of the mouse as a **Point** using the **X** and **Y** properties of the **MouseEventArgs**. I then get what that **Point** would be if scaled using **xScale** and **yScale**. To find which tile that point is in, using X and Y coordinates, you do like you do in the **Engine** class to determine what cell a **Vector2** is in. You divide the **X** by the **Width** and the **Y** by the **Height**. Tiles are created going left to right first and then top to bottom. To find the value of the tile in one dimension you take the **Y** coordinate, multiply it by the number of tiles wide, and add the **X** coordinate. I set that value to the **Value** property of **nudCurrentTile**. Since the **ValueChanged** event handler is wired for that, even if it is changed in the program, the event handler will be triggered. That will fill the preview picture box with the right prieve from the tileset.

If you start changing the size of the **MapDisplay** you may end up seeing the blue background, not what is desired. You will want what is displayed to change with what is visible. To do that I'm going to subscribe to the **SizeChanged** event of the **MapDisplay**. I will subscribe to that in the **Load** event of the form. Modify the handler for that event to the following and add this handler to the **Form Event Handler** region.

```
void FormMain_Load(object sender, EventArgs e)
{
    lbTileset.SelectedIndexChanged += new EventHandler(lbTileset_SelectedIndexChanged);
    nudCurrentTile.ValueChanged += new EventHandler(nudCurrentTile_ValueChanged);

    Rectangle viewPort = new Rectangle(0, 0, mapDisplay.Width, mapDisplay.Height);

    camera = new Camera(viewPort);
    engine = new Engine(32, 32);

    controlTimer.Tick += new EventHandler(controlTimer_Tick);

    controlTimer.Enabled = true;
    controlTimer.Interval = 200;

    tbMapLocation.TextAlign = HorizontalAlignment.Center;

    pbTilesetPreview.MouseDown += new MouseEventHandler(pbTilesetPreview_MouseDown);
    mapDisplay.SizeChanged += new EventHandler(mapDisplay_SizeChanged);
}

void mapDisplay_SizeChanged(object sender, EventArgs e)
{
    Rectangle viewPort = new Rectangle(0, 0, mapDisplay.Width, mapDisplay.Height);
    Vector2 cameraPosition = camera.Position;
        camera = new Camera(viewPort, cameraPosition);

    camera.LockCamera();
    mapDisplay.Invalidate();
}
```

What the event handler does is create a new camera and tell the map display to redraw itself. I create new rectangle the size of the map display. I then save the current position of the camera. I then create the new camera passing in the new rectangle and the position of the old camera. I then call the **LockCamera** method to lock the camera. Finally I call the **Invalidate** method to tell the **MapDisplay** to refresh itself.

What I also want to do is add in a grid that can be toggled on and off as well as a cursor for the mouse. If there is a map layer I will draw the currently selected tile under the image of the cursor as well. For that you are going to require two images. One for the grid and one for the mouse cursor. You can download them from https://cynthiamcmahon.ca/blog/downloads/mapimages.zip. Download the images and extract them to a folder. Right click the **XLevelEditor** project, select **Add** and then **New Folder**. Name this new folder **Content**. Drag the two files above onto that folder. No matter which approach you took in the properties set the **Copy to Output Directory** property to **Copy Always**.

Before I code the logic I want to add another menu item, a View menu item. Beside the **&Key** entry add in an entry **&View**. Under that add an item **&Display Grid**. Set the **Checked** and **CheckOnClick** properties for that menu item to **True**. I also dragged my **&View** menu item to be beside the **&Level** item.

The rendering will lbe done in the **MapDisplay** class. Replace the code in the **MapDisplay** class with the following. If you get an error trying to open it just press F7 to bring up the code for it.

```
using MGRpgLibrary.TileEngine;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace XLevelEditor
{
    public class MapDisplay : MonoGame.Forms.Controls.MonoGameControl
    {
        Texture2D grid;
        Texture2D cursor;
        MouseState mouseState;

        protected override void Initialize()
        {
            try
            {
                using (Stream stream = new FileStream(@"Content\grid.png", FileMode.Open,
                  FileAccess.Read))
                {
                    grid = Texture2D.FromStream(GraphicsDevice, stream);
                    stream.Close();
                }
                using (Stream stream = new FileStream(@"Content\cursor.png", FileMode.Open,
```

```
                    FileAccess.Read))
                {
                    cursor = Texture2D.FromStream(GraphicsDevice, stream);
                    stream.Close();
                }
            }
            catch (Exception exc)
            {
                System.Windows.Forms.MessageBox.Show(exc.Message, "Error reading images");
                grid = null;
                cursor = null;
            }

            base.Initialize();
        }

        protected override void Update(GameTime gameTime)
        {
            base.Update(gameTime);
        }

        protected override void Draw()
        {
            base.Draw();

            if (FormMain.map == null)
                return;

            Editor.spriteBatch.Begin(
                SpriteSortMode.Deferred,
                BlendState.AlphaBlend,
                SamplerState.PointClamp,
                null,
                null,
                null,
                FormMain.camera.Transformation);

            for (int i = 0; i < FormMain.layers.Count; i++)
                FormMain.layers[i].Draw(Editor.spriteBatch, FormMain.camera,
FormMain.tileSets);

            Editor.spriteBatch.End();
            DrawDisplay();
        }
        private void DrawDisplay()
        {
            if (FormMain.layers.Count == 0)
                return;

            Rectangle destination = new Rectangle(
                0,
                0,
                Engine.TileWidth,
                Engine.TileHeight);

            mouseState = Mouse.GetState();

            if (FormMain.DrawGrid)
            {
                int maxX = this.Width / Engine.TileWidth + 1;
```

```
                int maxY = this.Height / Engine.TileHeight + 1;

                Editor.spriteBatch.Begin();

                for (int y = 0; y < maxY; y++)
                {
                    destination.Y = y * Engine.TileHeight;

                    for (int x = 0; x < maxX; x++)
                        Editor.spriteBatch.Draw(grid, destination, Color.White);

                }
                Editor.spriteBatch.End();
            }

            Editor.spriteBatch.Begin();

            destination.X = mouseState.X;
            destination.Y = mouseState.Y;

            Editor.spriteBatch.Draw(
                FormMain.tileSets[FormMain.SelectedTileset].Texture,
                destination,

FormMain.tileSets[FormMain.SelectedTileset].SourceRectangles[FormMain.SelectedTile],
                Color.White);

            Editor.spriteBatch.Draw(cursor, destination, Color.White);
            Editor.spriteBatch.End();
        }
    }
}
```

There are three new fields in the class. One for the grid, one for the cursor and one for the mouse. In the Initialize method I load the textures in a try-catch block. It is best that when you are doing anything that could crash your application in a try-catch block to recover.

Inside the try-catch block are using statements where I create a new **Stream** for the image files in the **\Content** folder, that is why I had you set the **Copy to Output Directory** property so they will be there. Inside the using statements I use the **FromStream** method of the **Texture2D** class to read in the files. I then close the stream. It will automatically be disposed when the block ends. If there is an exception I display the message in a message box with a title. I also set both of the fields to be null. I will still have the editor work with out the images but not draw them. That is much nicer than not having the editor work at all.

Drawing the grid and the mouse cursor and preview will be done in a method that I will call from the **Draw** method. The new part of the **Draw** method just calls the **DrawDisplay** method after rendering the map. The **DrawDisplay** method will exit if the **layer** count is zero because there is no map being drawn. It then creates a **Rectangle** object, **destination**, that has the width and height from the **Engine** class. There is then an if statement that checks a property that has to be added to **FormMain** to determine if the grid should be drawn. If it is true I will draw the grid over the map. I find out how many items to draw across by dividing the **Width** of the map display by the **Width** of a tile on the screen and add 1. For the number to draw down I divide the **Height** of the map display by the **Height**

of a tile on the screen.

I call **Begin** on **spriteBatch** and then there is a set of nested loops much like when you draw a tile map. The out loop will loop through all of the rows and the inner loop the columns. In the outer loop I set the **Y** property of **destination** and in the inner loop I set the **X** property of **destination** just like when tiling. After the loops I call the **End** method of the **spriteBatch**. In between calls to **Begin** and **End** I draw the preview tile and the cursor image. I set the **X** and **Y** properties of destination to the **X** and **Y** properties of the mouse. For drawing the tile I get the **Texture2D** using the **tileSets** field and the **SelectedTileset** property of **FormMain** that I haven't added yet.. To select the source rectangle I again use the **SelectedTileset** property of **FormMain** and the **SelecteTile** property that I have to add next.

Add the following properties to the **FormMain** class.

```
        public static bool DrawGrid { get; set; } = true;

        public static int SelectedTileset
        {
            get; set;
        }

        public static int SelectedTile
        {
            get; set;
        }
```

Just some static properties that will return values based on properties of the form. Setting their values are done in the event handlers for the controls they are tied to. In the **FormLoad** method add the following code and this new event handlers.

```
void FormMain_Load(object sender, EventArgs e)
{
    lbTileset.SelectedIndexChanged += new EventHandler(lbTileset_SelectedIndexChanged);
    nudCurrentTile.ValueChanged += new EventHandler(nudCurrentTile_ValueChanged);
    displayGridToolStripMenuItem.CheckedChanged += DisplayGridToolStripMenuItem_CheckedChanged;
    Rectangle viewPort = new Rectangle(0, 0, mapDisplay.Width, mapDisplay.Height);

    camera = new Camera(viewPort);
    engine = new Engine(32, 32);

    controlTimer.Tick += new EventHandler(controlTimer_Tick);

    controlTimer.Enabled = true;
    controlTimer.Interval = 200;

    tbMapLocation.TextAlign = HorizontalAlignment.Center;

    pbTilesetPreview.MouseDown += new MouseEventHandler(pbTilesetPreview_MouseDown);
    mapDisplay.SizeChanged += new EventHandler(mapDisplay_SizeChanged);
}

private void DisplayGridToolStripMenuItem_CheckedChanged(object sender, EventArgs e)
{
    FormMain.DrawGrid = displayGridToolStripMenuItem.Checked;
```

```
}
```

It is pretty straight forward. The event handler is wired with the other event handlers. The event handler just sets the **FormMain.DrawGrid** property to the **Checked** property of the **displayGridToolStripMenuItem**.

We need to update the **FillPreviews** method to set the **SelectedTileset** and **SelectedTile** properties. Replace that methos with the following.

```
private void FillPreviews()
{
    int selected = lbTileset.SelectedIndex;
    int tile = (int)nudCurrentTile.Value;

    FormMain.SelectedTileset = selected;
    FormMain.SelectedTile = tile;

    GDIImage preview = (GDIImage)new GDIBitmap(pbTilePreview.Width, pbTilePreview.Height);

    GDIRectangle dest = new GDIRectangle(0, 0, preview.Width, preview.Height);
    GDIRectangle source = new GDIRectangle(
        tileSets[selected].SourceRectangles[tile].X,
        tileSets[selected].SourceRectangles[tile].Y,
        tileSets[selected].SourceRectangles[tile].Width,
        tileSets[selected].SourceRectangles[tile].Height);

    GDIGraphics g = GDIGraphics.FromImage(preview);

    g.DrawImage(tileSetImages[selected], dest, source, GDIGraphicsUnit.Pixel);

    pbTilesetPreview.Image = tileSetImages[selected];
    pbTilePreview.Image = preview;
}
```

**Bixel wa**s a reader of my tutorials and was an active member of my old forum. He had a nice idea for not always having to enter in values to the new level and new tileset forms. I'm going to take his suggestion and add them in here. Right click **FormNewLevel** in the solution explorer and select **View Code**. Change the constructor to the following and add the following method to the **Constructor** region.

```
public FormNewLevel()
{
    InitializeComponent();

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

    SetDefaultValues();
}

private void SetDefaultValues()
{
    tbLevelName.Text = "Starting Level";
    tbMapName.Text = "Village";
    mtbWidth.Text = "100";
    mtbHeight.Text = "100";
```

```
}
```

The new code just gives the controls on the form some starting values. I'm also going to change the and design for **FormNewTileset**. You don't need to input the number of tiles wide and tiles high a tile set is.

You can calculate those values with the other values on the form. Right click **FormNewTileset** and select **View Designer** to bring up design view of the form. While holding down <SHIFT> select the **Labels** and **Masked Text Boxes** associated with tiles wide and tiles high and press the <Delete> key to remove them. Move the buttons up a little and change the size of the form. My form appears on the next page.

That is going to break the code of the **btnOK_Click** method. I'm also going to set some default values for the form as well. Right click **FormNewTileset** in the solution explorer and select **View Code**. Change the code for that form to 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 FormNewTileset : Form
    {
        #region Field Region

        bool okPressed;
        TilesetData tilesetData;

        #endregion
        #region Property Region

        public TilesetData TilesetData
        {
            get { return tilesetData; }
        }

        public bool OKPressed
        {
            get { return okPressed; }
        }

        #endregion

        #region Constructor Region

        public FormNewTileset()
        {
            InitializeComponent();
```

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

    SetDefaultValues();
}

private void SetDefaultValues()
{
    tbTilesetName.Text = "Village Tileset";
    mtbTileWidth.Text = "32";
    mtbTileHeight.Text = "32";
}

#endregion

#region Button Event Handler Region

void btnSelectImage_Click(object sender, EventArgs e)
{
    OpenFileDialog ofDialog = new OpenFileDialog
    {
        Filter = "Image Files|*.BMP;*.GIF;*.JPG;*.TGA;*.PNG",
        CheckFileExists = true,
        CheckPathExists = true,
        Multiselect = false
    };

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

void btnOK_Click(object sender, EventArgs e)
{
    if (string.IsNullOrEmpty(tbTilesetName.Text))
    {
        MessageBox.Show("You must enter a name for the tileset.");
        return;
    }

    if (string.IsNullOrEmpty(tbTilesetImage.Text))
    {
        MessageBox.Show("You must select an image for the tileset.");
        return;
    }

    int tileWidth = 0;
    int tileHeight = 0;
    int tilesWide = 0;
    int tilesHigh = 0;

    if (!int.TryParse(mtbTileWidth.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(mtbTileHeight.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;
            }

            Image tileSet = (Image)Bitmap.FromFile(tbTilesetImage.Text);

            tilesWide = tileSet.Width / tileWidth;
            tilesHigh = tileSet.Height / tileHeight;

            tilesetData = new TilesetData
            {
                TilesetName = tbTilesetName.Text,
                TilesetImageName = tbTilesetImage.Text,
                TileWidthInPixels = tileWidth,
                TileHeightInPixels = tileHeight,
                TilesWide = tilesWide,
                TilesHigh = tilesHigh
            };

            okPressed = true;

            this.Close();
        }

        void btnCancel_Click(object sender, EventArgs e)
        {
            okPressed = false;

            this.Close();
        }
        #endregion
    }
}
```

Like on the other form the constructor calls a method, **SetDefaultValues**, to set some values for the form. That method sets the **Text** property of **tbTilesetName** to **Village Tileset** and the **Text** property of **mtbTileWidth** and **mtbTileHeight** to **32**. In the handler for the **Click** event of **btnOK** I removed the code that checked the values of tiles wide and tiles high. After validating the form I read in the image for the tile set using the **FromFile** method of the **Bitmap** class. I get **tilesWide** by dividing the **Width** of the image by **tileWidth**. Similarly, for **tilesHigh** I divide the **Height** of the image by **tileHeight**. The rest of the code is the same.

The next step will be writing out a level and reading it back in. To do that I'm going to add in a class like I did to the other editor to serialize and deserialize the data classes. Right click the **XLevelEditor** project select **Add** and then **Class**. Name this new class **XnaSerializer**. The code for that class follows

next.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Xml;
using Microsoft.Xna.Framework.Content.Pipeline.Serialization.Intermediate;

namespace XLevelEditor
{
    static class XnaSerializer
    {
        public static void Serialize<T>(string filename, T data)
        {
            XmlWriterSettings settings = new XmlWriterSettings
            {
                Indent = true
            };

            using (XmlWriter writer = XmlWriter.Create(filename, settings))
            {
                IntermediateSerializer.Serialize<T>(writer, data, null);
            }
        }

        public static T Deserialize<T>(string filename)
        {
            T data;

            using (FileStream stream = new FileStream(filename, FileMode.Open))
            {
                using (XmlReader reader = XmlReader.Create(stream))
                {
                    data = IntermediateSerializer.Deserialize<T>(reader, null);
                }
            }

            return data;
        }
    }
}
```

The code is the same as before. There are generic methods for serializing and deserializing objects using the **IntermediateSerializer** class. Since it is the same as before I'm not going to go into the code here.

Before I get to saving maps I need to add a little logic into **FormMain**. I need to keep track of the file names for the tile set images. To do that I'm going to add a **List<TilesetData>** that will hold the tile sets that are added to a map. Add the following field to the **Field** region of **FormMain** and change the handler for the **Click** event of the new tileset menu item to the following.

```
List<TilesetData> tileSetData = new List<TilesetData>();

void newTilesetToolStripMenuItem_Click(object sender, EventArgs e)
{
```

```
    using (FormNewTileset frmNewTileset = new FormNewTileset())
    {
        frmNewTileset.ShowDialog();

        if (frmNewTileset.OKPressed)
        {
            TilesetData data = frmNewTileset.TilesetData;

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

                tileSetImages.Add(image);

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

                Texture2D texture = Texture2D.FromStream(GraphicsDevice, stream);

                Tileset tileset = new Tileset(
                    texture,
                    data.TilesWide,
                    data.TilesHigh,
                    data.TileWidthInPixels,
                    data.TileHeightInPixels);

                tileSets.Add(tileset);
                tileSetData.Add(data);

                if (map != null)
                    map.AddTileset(tileset);

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

            lbTileset.Items.Add(data.TilesetName);

            if (lbTileset.SelectedItem == null)
                lbTileset.SelectedIndex = 0;

            mapLayerToolStripMenuItem.Enabled = true;
        }
    }
}
```

All the new code does is create a new **List<TilesetData>** called **tileSetData**. In the handler for the **Click** event of the new tileset menu item after I add the new tileset to **tileSets** I also add it to the new field **tileSetData**.

I also want to add an overload of the **SetTile** method of **MapLayerData** that takes four parameters instead of three. I will be passing in the **X** and **Y** coordinates of the tile as well as the index of the tile in the tileset and the tileset. Add the following method to **MapLayerData**.

```
public void SetTile(int x, int y, int tileIndex, int tileSet)
{
    Layer[y * Width + x] = new Tile(tileIndex, tileSet);
}
```

It works the same as the other **SetTile** method but just creates a new **Tile** rather than assigning the **Tile** that was passed in. Now I can handle writing out the data for a level. In the constructor for **FormMain** you want to wire a handler for the **Click** event for **saveLevelToolStripMenuItem**. I added a new region at the bottom of **FormMain** for menu items related to saving called **Save Menu Item Event Handler**. Change the constructor for **FormMain** to the following and the following code to the new region.

```
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;
    newLevelToolStripMenuItem.Click += new EventHandler(newLevelToolStripMenuItem_Click);
    newTilesetToolStripMenuItem.Click += new EventHandler(newTilesetToolStripMenuItem_Click);
    newLayerToolStripMenuItem.Click += new EventHandler(newLayerToolStripMenuItem_Click);
    saveLevelToolStripMenuItem.Click += new EventHandler(saveLevelToolStripMenuItem_Click);
}

#region Save Menu Item Event Handler Region

void saveLevelToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (layers.Count == 0)
        return;

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

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

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

        mapLayerData.Add(data);
    }

    MapData mapData = new MapData(levelData.MapName, tileSetData, mapLayerData);
```

```
    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);
    }
}

#endregion
```

What the new method does is check to see if the field **map** is null. If there is no map there is nothing really to save. I then create a **List<MapLayerData>** to hold the **MapLayerData** for each of the layers in the map. There is then a for loop that loops through all of the items the **Items** of **clbLayers**. Inside the loop I create a new **MapLayerData** object. I pass in the **Items** object for the associated index converted to a string because the items are stored as objects, and the **Width** and **Height** of the associated **MapLayer**. There is then a set of nest loops that will loop through all of the tiles in the layer. I call the **SetTile** method the I just wrote for the **MapLayerData** class passing in the loop variables for the **x** and **y**. For the **TileIndex** I use the **TileIndex** property after calling **GetTile** on the layer. You can string things like this together to make life easier. For the **Tileset** I use the **Tileset** property from **GetTile** on the the layer.

I then create a new instance of **MapData** use the **MapName** property of the **LevelData** object, the **List<TilesetData>** I created earlier and the **List<MapLayerData>** that I just created.
To save the level I display a **FolderBrowserDialog** setting the **Description** property to **Select Game Folder** and setting the **SelectedPath** to the start up path of the application. I capture the result of the dialog in the variable **result**. If the result of the dialog was that the user pressed OK I check to see if the **Game.xml** file exists. If it doesn't then I display an error message that the game wasn't found and exit the method. I then create two paths. The first path is for the levels in the game, **LevelPath**, and the second is for the maps in the game, **MapPath**. I then check to see if these paths exists. If they don't exist I create them. I then call the **Serialize** method of the **XnaSerializer** class to serialize the level and the map.

The next step is to reverse the process and read in a level and a map. In the constructor you will want to wire an event handler for **openLevelToolStripMenuItem**. I added another region to deal with reading in data. Change the constructor for **FormMain** to the following and add in the following region.

```
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;
    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);
}

void openLevelToolStripMenuItem_Click(object sender, EventArgs e)
{
    OpenFileDialog ofDialog = new OpenFileDialog
    {
        Filter = "Level Files (*.xml)|*.xml",
        CheckFileExists = true,
        CheckPathExists = true
    };

    DialogResult result = ofDialog.ShowDialog();

    if (result != DialogResult.OK)
        return;

    string path = Path.GetDirectoryName(ofDialog.FileName);

    LevelData newLevel = null;
    MapData mapData = null;

    try
    {
        newLevel = XnaSerializer.Deserialize<LevelData>(ofDialog.FileName);
        mapData = XnaSerializer.Deserialize<MapData>(path + @"\Maps\" + newLevel.MapName +
            ".xml");
    }
    catch (Exception exc)
    {
        MessageBox.Show(exc.Message, "Error reading level");
        return;
    }

    tileSetImages.Clear();
    tileSetData.Clear();
    tileSets.Clear();
    layers.Clear();
```

```
    lbTileset.Items.Clear();
    clbLayers.Items.Clear();

    foreach (TilesetData data in mapData.Tilesets)
    {
        Texture2D texture = null;

        tileSetData.Add(data);
        lbTileset.Items.Add(data.TilesetName);

        GDIImage image = (GDIImage)GDIBitmap.FromFile(data.TilesetImageName);
        tileSetImages.Add(image);

        using (Stream stream = new FileStream(data.TilesetImageName, FileMode.Open,
            FileAccess.Read))
        {
            texture = Texture2D.FromStream(GraphicsDevice, stream);

            tileSets.Add(
                new Tileset(
                    texture,
                    data.TilesWide,
                    data.TilesHigh,
                    data.TileWidthInPixels,
                    data.TileHeightInPixels));
        }
    }

    foreach (MapLayerData data in mapData.Layers)
    {
        clbLayers.Items.Add(data.MapLayerName, true);
        layers.Add(MapLayer.FromMapLayerData(data));
    }

    lbTileset.SelectedIndex = 0;
    clbLayers.SelectedIndex = 0;
    nudCurrentTile.Value = 0;

    map = new TileMap(tileSets, layers);

    tilesetToolStripMenuItem.Enabled = true;
    mapLayerToolStripMenuItem.Enabled = true;
    charactersToolStripMenuItem.Enabled = true;
    chestsToolStripMenuItem.Enabled = true;
    keysToolStripMenuItem.Enabled = true;
}
```

The code of interest is in the **Click** event handler for the level open menu item. I first create an object of type **OpenFileDialog** to browse for levels. Levels were stored as .xml files so I set the **Filter** property of **ofDialog** to display just .xml files. I set the **CheckFileExists** and **CheckPathExists** properties to true. I then capture the result of the **ShowDialog** method. If the result was not the user hitting the OK button I exit out of the method. To load all information about the level I need to capture the directory the level sits in. Use the **GetDirectoryName** method of the **Path** class to do that. I then create two local variables to read the data into. The variable **newLevel** will hold the **LevelData** and **mapData** will hold the **MapData**. In a try-catch block I try and deserialize the level and the map. If an exception was thrown I display it in a message box and exit the method. I then call the **Clear** method on a lot of collections. The **tileSetImages** is the GDI+ images of the tilesets, **tileSetData** is the

**TilesetData** for the tilesets, **tileSets** is the actual tilesets, **layers** is the map layers, and the last two are the **List Box** of tilesets and the **Checked List Box** for controlling map layers.
I guess I probably should have done the next part in a try-catch as well. In a foreach loop I loop through all of the **TilesetData** objects in the **Tilesets** field of the **MapData** read in. Inside the loop is a **Texture2D** to hold the texture for the tileset. I add the current **TilesetData** object to **tileSetData** and the name of the tileset data to **Items** collection of **lbTileset**. This is where I probably should have added a try-catch block. I use the **FormFile** method of the GDI+ **Bitmap** class to read in the image associated with the **TilesetData**. I then add that image to **tileSetImages**. The next block of code is a using statement where I create a **Stream** to the image for the tileset to read it in as a **Texture2D**. If you don't use **FileAccess.Read** you can get an exception that the file is in use if you are running from the debugger. I use the **FromStream** method to read the image in as a **Texture2D** and then add a new **Tileset** to the **tileSets** field.

Now that I'm done with the tilesets its time to do the layers. There is a foreach loop that will loop through all of the layers in **mapData**. I add the **MapLayerName** of the layer to the **Items** collection of **clbLayers** passing in true as well so that it will be checked. I then use the **FromMapLayerData** method of the **MapLayer** class to add a new layer to the **layers** field. I then set the **SelectedIndex** of **lbTileset** and **clbLayers** to 0 as well as the **Value** property of **nudCurrentTile**. I then create a new **TileMap** object using the **tileSets** and **layers** fields. The last thing I do is set the **Enabled** property of the other menu items to true.

That's it for this tutorial. It covers the original tutorial with many modifications. We can not save our levels and read them back in. Also, some modifications to drawing tiles. So, I encourage you to visit my blog at, https://cynthiamcmahon.ca/blog/, for the latest news on my tutorials and other goodness.

Good luck in your Game Programming Adventures!

*Cynthia*