Eyes of the Dragon Tutorials
Part 31
Tile Engine Update

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 this tutorial I'm going to make an update to the tile engine and in doing so the levels. I started doing something in my tile engines that I really liked. I created an abstract base that represents a layer of the map. I also have a class that inherits from this class that represents a layer of tiles. I also have other layers like an item layer and a sprite layer. On the item layer I have things like chests that the player interacts with. On the sprite layer you can have animated tiles that will be updated each frame of the game. You are unlimited in the types of layers you can have. All of the layers can still be managed by the map class. Instead of using the abstract class I'm going to use an interface instead.

To get started you will want to add a class that represents a layer. Right click the **TileEngine** folder in the **MGRpgLibrary** folder, select **Add** and then **Interface**. Name this new interface **ILayer**. The code for that file follows next.

```
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;

namespace MGRpgLibrary.TileEngine
{
    public interface ILayer
    {
        void Update(GameTime gameTime);
        void Draw(SpriteBatch spriteBatch, Camera camera, List<Tileset> tilesets);
    }
}
```

A simple interface that has two methods associated with it. There are also some using statements to bring classes into scope. The first method is an **Update** method that allows a layer to update itself, useful for animated tiles and sprites. The second is a **Draw** method that allows a layer to draw itself. For a layer to draw itself it needs a **SpriteBatch**, **Camera**, and a **List<Tileset>**.

The next step is to update the **MapLayer** class to implement the **ILayer** interface. Update the **MapLayer** class to the following.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using RpgLibrary.WorldClasses;
```

```
namespace MGRpgLibrary.TileEngine
{
    public class MapLayer : ILayer
    {
        #region Field Region

        Tile[,] layer;

        #endregion

        #region Property Region

        public int Width
        {
            get { return layer.GetLength(1); }
        }

        public int Height
        {
            get { return layer.GetLength(0); }
        }

        #endregion

        #region Constructor Region

        public MapLayer(Tile[,] map)
        {
            this.layer = (Tile[,])map.Clone();
        }

        public MapLayer(int width, int height)
        {
            layer = new Tile[height, width];

            for (int y = 0; y < height; y++)
            {
                for (int x = 0; x < width; x++)
                {
                    layer[y, x] = new Tile(0, 0);
                }
            }
        }

        #endregion

        #region Method Region

        public Tile GetTile(int x, int y)
        {
            return layer[y, x];
        }

        public void SetTile(int x, int y, Tile tile)
        {
            layer[y, x] = tile;
        }

        public void SetTile(int x, int y, int tileIndex, int tileset)
```

```
        {
            layer[y, x] = new Tile(tileIndex, tileset);
        }

        public void Update(GameTime gameTime)
        {
        }

        public void Draw(SpriteBatch spriteBatch, Camera camera, List<Tileset> tilesets)
        {
            Point cameraPoint = Engine.VectorToCell(camera.Position * (1 / camera.Zoom));
            Point viewPoint = Engine.VectorToCell(
                new Vector2(
                    (camera.Position.X + camera.ViewportRectangle.Width) * (1 / camera.Zoom),
                    (camera.Position.Y + camera.ViewportRectangle.Height) * (1 /
camera.Zoom)));

            Point min = new Point();
            Point max = new Point();

            min.X = Math.Max(0, cameraPoint.X - 1);
            min.Y = Math.Max(0, cameraPoint.Y - 1);
            max.X = Math.Min(viewPoint.X + 1, Width);
            max.Y = Math.Min(viewPoint.Y + 1, Height);

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

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

                for (int x = min.X; x < max.X; x++)
                {
                    tile = GetTile(x, y);

                    if (tile.TileIndex == -1 || tile.Tileset == -1)
                        continue;

                    destination.X = x * Engine.TileWidth;

                    spriteBatch.Draw(
                        tilesets[tile.Tileset].Texture,
                        destination,
                        tilesets[tile.Tileset].SourceRectangles[tile.TileIndex],
                        Color.White);
                }
            }
        }

        public static MapLayer FromMapLayerData(MapLayerData data)
        {
            MapLayer layer = new MapLayer(data.Width, data.Height);

            for (int y = 0; y < data.Height; y++)
            {
                for (int x = 0; x < data.Width; x++)
                {
                    layer.SetTile(
                        x,
```

```
                        y,
                        data.GetTile(x, y).TileIndex,
                        data.GetTile(x, y).TileSetIndex);
                }
            }

            return layer;
        }
        #endregion
    }
}
```

Not many changes here. The first is **MapLayer** now implements the **ILayer** interface. The only other change is that I included a blank **Update** method, to complete implementing the interface as there was already a **Draw** method that matches the interface.

Next I will update the **TileMap** class to use the interface. There were a number of changes to the **TileMap** class. The new **TileMap** class follows next.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;

namespace MGRpgLibrary.TileEngine
{
        public class TileMap
    {
        #region Field Region

        List<Tileset> tilesets;
        List<ILayer> mapLayers;
        static int mapWidth;
        static int mapHeight;

        #endregion

        #region Property Region

        public static int WidthInPixels
        {
            get { return mapWidth * Engine.TileWidth; }
        }

        public static int HeightInPixels
        {
            get { return mapHeight * Engine.TileHeight; }
        }

        #endregion

        #region Constructor Region

        public TileMap(List<Tileset> tilesets, MapLayer baseLayer, MapLayer buildingLayer,
            MapLayer splatterLayer)
```

```
{
    this.tilesets = tilesets;
    this.mapLayers = new List<ILayer>();

    mapLayers.Add(baseLayer);

    AddLayer(buildingLayer);
    AddLayer(splatterLayer);

    mapWidth = baseLayer.Width;
    mapHeight = baseLayer.Height;
}

public TileMap(Tileset tileset, MapLayer baseLayer)
{
    tilesets = new List<Tileset>();
    tilesets.Add(tileset);

    mapLayers = new List<ILayer>();
    mapLayers.Add(baseLayer);

    mapWidth = baseLayer.Width;
    mapHeight = baseLayer.Height;
}

#endregion

#region Method Region
public void AddLayer(ILayer layer)
{
    if (layer is MapLayer)
    {
        if (!((MapLayer)layer).Width == mapWidth && ((MapLayer)layer).Height ==
            mapHeight))
        {
            throw new Exception("Map layer size exception");
        }
    }

    mapLayers.Add(layer);
}

public void AddTileset(Tileset tileset)
{
    tilesets.Add(tileset);
}

public void Update(GameTime gameTime)
{
    foreach (ILayer layer in mapLayers)
    {
        layer.Update(gameTime);
    }
}

public void Draw(SpriteBatch spriteBatch, Camera camera)
{
    foreach (MapLayer layer in mapLayers)
    {
        layer.Draw(spriteBatch, camera, tilesets);
```

```
            }
        }
        #endregion
    }
}
```

The first change is that instead of having a **List<MapLayer>** I now have a **List<ILayer>** for the layers in the map. The next change is in the constructors. Instead of taking a **List<MapLayer>** the first constructor now takes three **MapLayer** objects. Eventually I will expand it to take more objects but it works for now. The constructor assigns the **tilesets** field and creates a new **List<ILayer>** for the layers. I then add the **baseLayer** parameter to the **List<ILayer>**. To add the other layers I use the **AddLayer** method to confirm that layers are the same size. I then set the **mapWidth** and **mapHeight** fields using the **baseLayer.Width** and **baseLayer.Height** respectively.

The next constructor still takes a **Tileset** and **MapLayer** object as arguments. The only real changes are that I renamed the **MapLayer** parameter to **baseLayer** and instead of creating a **List<MapLayer>** I create a **List<ILayer>**.

I also modified the **AddLayer** method. It now takes an **ILayer** parameter rather than a **MapLayer** parameter. Since both constructors require a **MapLayer** as a parameter I check to see if the type of layer being added is a **MapLayer**. There is then an if statement that checks if the width and height of the layer are not the height and width of the map. **MapLayers** must be the same width as the base layer so if the size of both are not the size of the base I raise an exception. I then add the layer to the list of layers.

That broke the **LoadGameScreen** and **CharacterGeneratorScreen** classes because we were using the constructor that passes in a lists of tile sets and map layers in the **CreateWorld** methods. First, change the **CreateWorld** method of **LoadGameScreen** to the following.

```
private void CreateWorld()
{
    Texture2D tilesetTexture = Game.Content.Load<Texture2D>(@"Tilesets\tileset1");
    Tileset tileset1 = new Tileset(tilesetTexture, 8, 8, 32, 32);

    tilesetTexture = Game.Content.Load<Texture2D>(@"Tilesets\tileset2");

    Tileset tileset2 = new Tileset(tilesetTexture, 8, 8, 32, 32);
    MapLayer layer = new MapLayer(100, 100);

    for (int y = 0; y < layer.Height; y++)
    {
        for (int x = 0; x < layer.Width; x++)
        {
            Tile tile = new Tile(0, 0);
            layer.SetTile(x, y, tile);
        }
    }

    MapLayer splatter = new MapLayer(100, 100);
    Random random = new Random();

    for (int i = 0; i < 100; i++)
    {
```

```
            int x = random.Next(0, 100);
            int y = random.Next(0, 100);
            int index = random.Next(2, 14);

            Tile tile = new Tile(index, 0);

            splatter.SetTile(x, y, tile);
        }

        splatter.SetTile(1, 0, new Tile(0, 1));
        splatter.SetTile(2, 0, new Tile(2, 1));
        splatter.SetTile(3, 0, new Tile(0, 1));

        TileMap map = new TileMap(tileset1, layer);

        map.AddTileset(tileset2);
        map.AddLayer(splatter);

        Level level = new Level(map);

        World world = new World(GameRef, GameRef.ScreenRectangle);

        world.Levels.Add(level);
        world.CurrentLevel = 0;

        GamePlayScreen.World = world;
    }
```

The flow is basically the same. I create the two tile sets and the two map layers. I just removed the two **List<T>** variables as they aren't needed. I then use the second constructor, the one that takes a **Tileset** and **ILayer** arguments. To create the **TileMap** object. I then use the **AddTileset** and **AddLayer** methods of **TileMap** to add the second tile set and layer.

The changes to the **CreateWorld** method of **CharacterGeneratorScreen** were the same as in the **LoadGameScreen**. You can change that method to the following.

```
private void CreateWorld()
{
    Texture2D tilesetTexture = Game.Content.Load<Texture2D>(@"Tilesets\tileset1");
    Tileset tileset1 = new Tileset(tilesetTexture, 8, 8, 32, 32);

    tilesetTexture = Game.Content.Load<Texture2D>(@"Tilesets\tileset2");

    Tileset tileset2 = new Tileset(tilesetTexture, 8, 8, 32, 32);
    MapLayer layer = new MapLayer(100, 100);

    for (int y = 0; y < layer.Height; y++)
    {
        for (int x = 0; x < layer.Width; x++)
        {
            Tile tile = new Tile(0, 0);
            layer.SetTile(x, y, tile);
        }
    }

    MapLayer splatter = new MapLayer(100, 100);
    Random random = new Random();
```

```
    for (int i = 0; i < 100; i++)
    {
        int x = random.Next(0, 100);
        int y = random.Next(0, 100);
        int index = random.Next(2, 14);

        Tile tile = new Tile(index, 0);

        splatter.SetTile(x, y, tile);
    }

    splatter.SetTile(1, 0, new Tile(0, 1));
    splatter.SetTile(2, 0, new Tile(2, 1));
    splatter.SetTile(3, 0, new Tile(0, 1));

    TileMap map = new TileMap(tileset1, layer);

    map.AddTileset(tileset2);
    map.AddLayer(splatter);

    Level level = new Level(map);

    World world = new World(GameRef, GameRef.ScreenRectangle);

    world.Levels.Add(level);
    world.CurrentLevel = 0;

    GamePlayScreen.World = world;
}
```

The other thing that was broken was **FormMain** in the **XLevelEditor** project. The first thing to change however is the **layers** field. Instead of a **List<MapLayer>** I use a **List<ILayer>**. Change the **layers** field to the this.

```
public static List<ILayer> layers = new List<ILayer>();
```

The first thing that was broken was the **newLayerToolStripMenuItem_Click** method. It was where I created a map if the **map** field was null. I had to create the map using the second constructor that takes a **Tileset** and a **MapLayer**. Update the **newLayerToolStringMenuItem_Click** method.

```
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], (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;
        }
    }
}
```

So, what changed here is the code for the if statement that checks if the **map** field is null. I use the constructor that takes a single **TileSet** and a **ILayer**. There is then a two for loops. They start from 1 instead of 0 because the first object is already in the map and you down want it added again. The first loop adds in tile sets and the second adds in layers.

The next thing that was broken is the **SetTiles** method. The layers are no longer just **MapLayers** so I can't use the methods and properties associated with **MapLayers**. Change the **SetTiles** method to the follow.

```
private void SetTiles(Point tile, int tileIndex, int tileset)
{
    int selected = clbLayers.SelectedIndex;
    if (layers[selected] is MapLayer)
    {
        for (int y = 0; y < brushWidth; y++)
        {
            if (tile.Y + y >= ((MapLayer)layers[selected]).Height)
                break;
            for (int x = 0; x < brushWidth; x++)
            {
                if (tile.X + x < ((MapLayer)layers[selected]).Width)
                    ((MapLayer)layers[selected]).SetTile(
                    tile.X + x,
                    tile.Y + y,
                    tileIndex,
                    tileset);
            }
        }
    }
}
```

What the method does is check to see if the selected layer is a **MapLayer**. If it is a **MapLayer** I cast

the **ILayer** to a **MapLayer** so I can use the methods and properties of the **MapLayer** class.

The **saveLevelToolStripMenuItem_Click** was broken as well. Again it was because I'm using **ILayer** instead of **MapLayer** to hold information. What I did was in the for loop that loops through the layers is check to see if the layer is a **MapLayer**. If it is a **MapLayer** I cast the layer to be a **MapLayer**. Change the **saveLevelToolStripMenuItem_Click** 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, 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);
    }
}
```

The same problem is in the **saveLayerToolStripMenuItem_Click** method. You can change that method to the following.

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

    if (layers[clbLayers.SelectedIndex] is MapLayer)
    {
        SaveFileDialog sfDialog = new SaveFileDialog();
        sfDialog.Filter = "Map Layer Data (*.mldt)|*.mldt";
        sfDialog.CheckPathExists = true;
        sfDialog.OverwritePrompt = true;
        sfDialog.ValidateNames = true;

        DialogResult result = sfDialog.ShowDialog();

        if (result != DialogResult.OK)
            return;

        MapLayerData data = new MapLayerData(
            clbLayers.SelectedItem.ToString(),
            ((MapLayer)layers[clbLayers.SelectedIndex]).Width,
            ((MapLayer)layers[clbLayers.SelectedIndex]).Height);

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

        try
        {
            XnaSerializer.Serialize<MapLayerData>(sfDialog.FileName, data);
        }
        catch (Exception exc)
        {
            MessageBox.Show(exc.Message, "Error saving map layer data");
        }
    }
}
```

Opening levels and layers is also broken. For now I'm going to make the assumption that all levels

being loaded are made of **MapLayers**. When I add other layer types I will have to update the saving and loading code. It is unfortunate that so much got broken but it will be well worth the headache later on. First the **openLevelToolStripMenuItem_Click** method. You can change that method to the following.

```
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();

    levelData = newLevel;

    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[0], (MapLayer)layers[0]);

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

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

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

The change here is where I create the map. I did it as before, after creating all of the map layers and tile sets. I create a **TileMap** using the constructor that takes a single tile set and map layer. I then loop through the remaining tile sets and add them using the **AddTileset** method. I then do the same for the layers using the **AddLayer** method.

That just leaves the **openLayerToolStripMenuItem_Click** method. You can change that method to the following.

```
void openLayerToolStripMenuItem_Click(object sender, EventArgs e)
{
    OpenFileDialog ofDialog = new OpenFileDialog
    {
        Filter = "Map Layer Data (*.mldt)|*.mldt",
        CheckPathExists = true,
        CheckFileExists = true
    };

    DialogResult result = ofDialog.ShowDialog();
```

```
    if (result != DialogResult.OK)
    {
        return;
    }

    MapLayerData data = null;

    try
    {
        data = XnaSerializer.Deserialize<MapLayerData>(ofDialog.FileName);
    }
    catch (Exception exc)
    {
        MessageBox.Show(exc.Message, "Error reading map layer");
        return;
    }

    for (int i = 0; i < clbLayers.Items.Count; i++)
    {
        if (clbLayers.Items[i].ToString() == data.MapLayerName)
        {
            MessageBox.Show("Layer by that name already exists.", "Existing layer");
            return;
        }
    }

    clbLayers.Items.Add(data.MapLayerName, true);
    layers.Add(MapLayer.FromMapLayerData(data));

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

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

The change again is in creating a new map if the **map** field is null. I create a map using the constructor that takes a tile set and a map layer. I then loop through any other tile sets and add them to the map.

That's it for this tutorial. It covers the original tutorial with few modifications. 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