Eyes of the Dragon - XNA

Part 36

Animated Tiles

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.

In this tutorial I will be switching modes to work on the editor instead of the game.

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.

Initially my plan for this tutorial was to add animated tiles to the game and place them with the editor. I'm going to tackle the editor part in the next tutorial because of the number of changes that were made to get animated tiles into the game.

The first thing we will need are some classes that represent animated tiles, in general. I will still be using the concept of tile sets where animated tiles are concerned. The way that I will implement them is that all animated tiles are on a single row of an image. The animation will move across the row and then wrap around to the beginning. This will mean you need one row for each animated tile.

Now, right click the TileEngine folder, select Add and then Class. Name this new class AnimatedTileset. The code for that class will follow. I actually include the AnimatedTile class in this file as well, though you could create separate files for both.

```
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 XRpgLibrary.TileEngine
{
    public class AnimatedTile
    {
```

```
        private int framesPerSecond;

        public int TileIndex;
        public int CurrentFrame;
        public int FrameCount;
        public TimeSpan Elapsed;
        public TimeSpan Length;

        public int FramesPerSecond
        {
            get { return framesPerSecond; }
            set
            {
                if (value < 1)
                    framesPerSecond = 1;
                else if (value > 60)
                    framesPerSecond = 60;
                else
                    framesPerSecond = value;
               Length = TimeSpan.FromSeconds(1 / (double)framesPerSecond);
            }
        }

        public AnimatedTile(int tileIndex, int frames)
        {
            TileIndex = tileIndex;
            CurrentFrame = 0;
            framesPerSecond = 8;
            FrameCount = frames;
            Elapsed = TimeSpan.Zero;
            Length = TimeSpan.Zero;

            Length = TimeSpan.FromSeconds(1 / (double)framesPerSecond);
        }

        public void Update(GameTime gameTime)
        {
            Elapsed += gameTime.ElapsedGameTime;

            if (Elapsed >= Length)
            {
                Elapsed = TimeSpan.Zero;
                CurrentFrame = (CurrentFrame + 1) % FrameCount;
            }
        }
    }

    public class AnimatedTileset
    {
        #region Fields and Properties

        Texture2D image;
        int tileWidthInPixels;
        int tileHeightInPixels;
        int frameCount;
        int tilesHigh;
        List<Rectangle[]> sourceFrames = new List<Rectangle[]>();

        #endregion

        #region Property Region

        public Texture2D Texture
        {
            get { return image; }
            private set { image = value; }
```

```
        }

        public int TileWidth
        {
            get { return tileWidthInPixels; }
            private set { tileWidthInPixels = value; }
        }

        public int TileHeight
        {
            get { return tileHeightInPixels; }
            private set { tileHeightInPixels = value; }
        }

        public int FrameCount
        {
            get { return frameCount; }
            private set { frameCount = value; }
        }

        public int TilesHigh
        {
            get { return tilesHigh; }
            private set { tilesHigh = value; }
        }

        public List<Rectangle[]> SourceFrames
        {
            get { return sourceFrames; }
        }

        #endregion

        #region Constructor Region

        public AnimatedTileset(Texture2D image, int frameCount, int tilesHigh, int
tileWidth, int tileHeight)
        {
            Texture = image;
            TileWidth = tileWidth;
            TileHeight = tileHeight;
            FrameCount = frameCount;
            TilesHigh = tilesHigh;

            for (int y = 0; y < tilesHigh; y++)
            {
                Rectangle[] frames = new Rectangle[frameCount];

                for (int x = 0; x < frameCount; x++)
                {
                    frames[x] = new Rectangle(
                        x * tileWidth,
                        y * tileHeight,
                        tileWidth,
                        tileHeight);
                }

                SourceFrames.Add(frames);
            }
        }

        #endregion

        #region Method Region
        #endregion
    }
```

```
}
```

The first class is AnimatedTile for the aniamted tiles. I was originally going to use a struct for this but I found that the Update method did not behave as I would have expected so I made it a class instead. There is a private field, framesPerSecond, that determines how often the animation will changed each second. Next are a few public fields. It would be better practice to make them public and expose them as properties but for demo purposes this is okay. The member variables represent the index of the tile in the animated tile set, what frame is currently being displayed, the number of frames for the animation, how much time has elapsed since the last update and the amount of time that needs to pass before the animation updates.

I did add a public property, FramesPerSecond, that has a getter and a setter. The getter simply return the member variable. The setter though does some validation. Since this is an animate object the number of times the frame changes each section should technically be below 2 but I just check to make sure it is 1 or greater. If it is less than 1 I set the value to 1. Similarly I check that the maximum number of frames is greater than 60 and cap it at 60. Typically for animation you don't want much more than 30 frames, which is close to a movie, but since the maximum frame rate for the game is 60 I cap it there. Otherwise I set the framesPerSecond to be the value passed in. After the validation I set the Length member to 1 divided by the number of frames.

The constructor just initializes the member variables based on the values passed in. After initializing the other members I use the same formula as in the property to calculate the length of time each frame should be displayed before moving onto the next frame.

I also included an Update method to update the animation. It adds the elapsed time passed to the Elapsed member variable. I then compare that to the Length member to see if it greater than or equal to. If it is I reset that member variable to zero time elapsed and calculate the next frame to be displayed. Because I used the modulus operator the value will range between the 0 and the number of frames minus 1, which corresponds with the indexes for each frame.

The next class represents an animated tile set. As I mentioned I'm implementing this where each animated tile is on its own row in the tile set. There are member variables for the image for the tile set, the width of the tile set in pixels, the height of the tile set in pixels, the frame count for each row of tiles, the number of tiles high the set is and a list of rectangles for each row.

Next up are public properties for each member variable where the getter is public and the setter is private. They are meant to give access to other classes that need it but keep being able to change the values inside the class. The one exception is the list of source rectangle. There is no setter for that property. The constructor then initializes values based on the parameters passed in. Then there is a loop that will loop over each row in the tile set. Inside that loop I create an array of Rectangle objects that hold the source rectangles for that row. Next is another loop that loops for the number of frames for the row. It then assigns the source rectangle for that frame using the values passed in.

We need a data class that will be used when reading saved maps from the game. I added that class to the TilesetData class rather than creating a new file. Update that class to the following code.

```
using System;
```

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

namespace RpgLibrary.WorldClasses
{
    public class TilesetData
    {
        public string TilesetName;
        public string TilesetImageName;
        public int TileWidthInPixels;
        public int TileHeightInPixels;
        public int TilesWide;
        public int TilesHigh;
    }

    public class AnimatedTilesetData
    {
        public string TilesetName;
        public string TilesetImageName;
        public int TileWidthInPixels;
        public int TileHeightInPixels;
        public int FramesAcross;
        public int TilesHigh;
    }
}
```

This class is basically a replica of the other class in this file. It is made up of just member variables that hold the name of the tile set, the name of the tile set image, the width of tiles in pixels, the height of the tile in pixels, the number of frames in each row and the number of rows.

The next change that I made was adding in a class that would represent an animated tile layer. Right click the TileEngine folder in the XRpgLibrary project, select Add and then Class. Name this new class AnimatedTileLayer. The code for that class follows next.

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

namespace XRpgLibrary.TileEngine
{
    public class AnimatedTileLayer
    {
        private Dictionary<Point, AnimatedTile> animatedTiles = new Dictionary<Point,
AnimatedTile>();

        [ContentSerializer]
        public Dictionary<Point, AnimatedTile> AnimatedTiles
        {
            get { return animatedTiles; }
            private set { animatedTiles = value; }
        }

        public AnimatedTileLayer()
        {
        }

        public void Update(GameTime gameTime)
```

```
        {
            foreach (Point p in AnimatedTiles.Keys)
                AnimatedTiles[p].Update(gameTime);
        }

        public void Draw(SpriteBatch spriteBatch, AnimatedTileset tileSet)
        {
            Rectangle destination = new Rectangle(0, 0, Engine.TileWidth,
Engine.TileHeight);

            foreach (Point p in AnimatedTiles.Keys)
            {
                destination.X = p.X * Engine.TileWidth;
                destination.Y = p.Y * Engine.TileHeight;

                spriteBatch.Draw(tileSet.Texture, destination,
tileSet.SourceFrames[AnimatedTiles[p].TileIndex][AnimatedTiles[p].CurrentFrame],
Color.White);
            }
        }
    }
}
```

This class is similar to the CollisionLayer class that I implemented a few tutorials back. The animated tiles are stored in a dictionary with the Point for the tile as the key and the animated tile as the value. I expose the member variable using a property with a public get and a private set. I also marked with the attribute to tell the IntermediateSeralizer to serialize it.

The class doesn't hold any other data so I just included a default public constructor that will be used when deserializing the class and making new layers. The Update method loops over each of the collection of animated tiles and calls their Update methods. There is also a draw method to draw the layer that takes a SpriteBatch parameter and a AnimatedTileset parameter. Inside the method I create a destination Rectangle that will be used to position the tile. I then loop over the collection of tiles. Using the key I position the tile by multiplying the key by the width and height of the tiles in the engine. I now draw the tile using the destination rectangle and the source rectangle. That is found by using the TileIndex of the tile for the first index and the CurrentFrame of the tile for the second.

The next thing that I had to do is update the MapData class to add the new classes to the map. What I did was add in member variables and marked them with an attribute to make them optional which will allow loading maps that did not contain these new fields. I also updated the constructor to initialize the values. Here are the changes that I made.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using XRpgLibrary.TileEngine;

namespace RpgLibrary.WorldClasses
{
    public class MapData
    {
        public string MapName;
        public MapLayerData[] Layers;
        public TilesetData[] Tilesets;
```

```
        [ContentSerializer(Optional = true)]
        public AnimatedTilesetData AnimatedTileset;

        [ContentSerializer(Optional = true)]
        public AnimatedTileLayer AnimatedTiles;

        [ContentSerializer(Optional=true)]
        public CollisionLayer Collisions;

        private MapData()
        {
        }

        public MapData(string mapName, List<TilesetData> tilesets, AnimatedTilesetData
animatedSet, List<MapLayerData> layers, CollisionLayer collisionLayer, AnimatedTileLayer
animatedLayer)
        {
            MapName = mapName;
            AnimatedTileset = animatedSet;
            Tilesets = tilesets.ToArray();
            Layers = layers.ToArray();
            Collisions = collisionLayer;
            AnimatedTiles = animatedLayer;
        }
    }
}
```

The next step will be to update the TileMap class to have an AnimatedTileLayer. What I did was add new member variables for an AnimatedTileset and an AnimatedTileLayer. The constructors were updated to accept these new fields and assign them. The Update method was updated to call the Update method of the AnimatedTileLayer and the Draw method was updated to call the Draw method if there is an AnimatedTileset object. Update the TileMap 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;

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

        List<Tileset> tilesets;
        AnimatedTileset animatedSet;
        List<ILayer> mapLayers;
        CollisionLayer collisionLayer;
        AnimatedTileLayer animatedTileLayer;

        static int mapWidth;
        static int mapHeight;

        #endregion

        #region Property Region

        public CollisionLayer CollisionLayer
        {
```

```
            get { return collisionLayer; }
        }

        public AnimatedTileLayer AnimatedTileLayer
        {
            get { return animatedTileLayer; }
        }

        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, AnimatedTileset animatedTileSet, MapLayer
baseLayer, MapLayer buildingLayer, MapLayer splatterLayer, CollisionLayer collisionLayer,
AnimatedTileLayer animatedLayer)
        {
            this.tilesets = tilesets;
            this.animatedSet = animatedTileSet;
            this.mapLayers = new List<ILayer>();
            this.collisionLayer = collisionLayer;
            this.animatedTileLayer = animatedLayer;

            mapLayers.Add(baseLayer);

            AddLayer(buildingLayer);
            AddLayer(splatterLayer);

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

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

            collisionLayer = new CollisionLayer();
            animatedTileLayer = new AnimatedTileLayer();
            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);
            }

            animatedTileLayer.Update(gameTime);
        }

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

            if (animatedSet != null)
                animatedTileLayer.Draw(spriteBatch, animatedSet);
        }

        #endregion
    }
}
```

The change was in the World class. It needed to be updated to call the Update method of the current map so that the AnimatedTileLayer would update the tiles. Change the World 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 Microsoft.Xna.Framework.Input;

using RpgLibrary.CharacterClasses;
using RpgLibrary.ItemClasses;

using XRpgLibrary.TileEngine;
using XRpgLibrary.SpriteClasses;

namespace XRpgLibrary.WorldClasses
{
    public class World : DrawableGameComponent
    {
        #region Graphic Field and Property Region

        Rectangle screenRect;

        public Rectangle ScreenRectangle
        {
            get { return screenRect; }
```

```
        }

        #endregion

        #region Item Field and Property Region

        ItemManager itemManager = new ItemManager();

        #endregion

        #region Level Field and Property Region

        readonly List<Level> levels = new List<Level>();
        int currentLevel = -1;

        public List<Level> Levels
        {
            get { return levels; }
        }

        public int CurrentLevel
        {
            get { return currentLevel; }
            set
            {
                if (value < 0 || value >= levels.Count)
                    throw new IndexOutOfRangeException();

                if (levels[value] == null)
                    throw new NullReferenceException();

                currentLevel = value;
            }
        }

        public TileMap CurrentMap
        {
            get { return levels[currentLevel].Map; }
        }

        #endregion

        #region Constructor Region

        public World(Game game, Rectangle screenRectangle)
            : base(game)
        {
            screenRect = screenRectangle;
        }

        #endregion

        #region Method Region

        public override void Update(GameTime gameTime)
        {
            CurrentMap.Update(gameTime);
        }

        public override void Draw(GameTime gameTime)
        {
            base.Draw(gameTime);
        }

        public void DrawLevel(GameTime gameTime, SpriteBatch spriteBatch, Camera camera)
        {
```

```
            levels[currentLevel].Draw(gameTime, spriteBatch, camera);
        }

        #endregion
    }
}
```

These changes have broken a few methods in the level editor. Anywhere that a tile map or tile map data is created there is no longer a valid constructor because they now take as parameters the new animated tile classes. First, add the following two member variables with the others in FormMain in the XLevelEditor project.

```
        AnimatedTileset animatedSet;
        AnimatedTilesetData animatedSetData;
```

Now, in the newLayerToolStripMenuItem_Click method when creating the map if it does not exist we need to pass in an animated tile set as well as the other parameters. I just used the animatedSet field that I just added. Update that method 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;
        }
    }
}
```

The next change was int the openLevelToolStripMenuItem_Click method. For now I added in reading the texture for the animated tile set and passing the the new member variable to the constructor. Update that method as follows.

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

```
    }

    Stream textureStream = new FileStream(mapData.AnimatedTileset.TilesetImageName,
FileMode.Open, FileAccess.Read);
    Texture2D aniamtedTexture = Texture2D.FromStream(GraphicsDevice, textureStream);
    animatedSet = new AnimatedTileset(aniamtedTexture, 8, 1, 64, 64);

    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], animatedSet, (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;
}
```

This is really just enough to allow the project to build so that we can test the animated tile in the game. Once I move onto the editor I will flesh it out more. The last method to update will be the openLayerToolStripMenuItem_Click method. This method creates a map object if there is no map object. I just updated that call to include the animated tile set member variable that I added. Here is the updated code.

```
void openLayerToolStripMenuItem_Click(object sender, EventArgs e)
{
    OpenFileDialog ofDialog = new OpenFileDialog();
    ofDialog.Filter = "Map Layer Data (*.mldt)|*.mldt";
    ofDialog.CheckPathExists = true;
    ofDialog.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], animatedSet, (MapLayer)layers[0]);

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

I also needed to update the saveLevelToolStripMenuItem_Click method because I create a new MapData object and the signature for that constructor changed. Update that 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, new CollisionLayer(), new 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);
    }
}
```

Those are the required changes that I had to make in order to have the level editor project build. I also needed to update the CreateWorld method in the CharacterGeneratorScreen and LoadGameScreen. First we need at least one animated tile. I visited Reiner's Tilesets and download his animated townthings tile set from this page http://www.reinerstilesets.de/2d-grafiken/2d-animated/. I then modified the camp file tile and the flame so that we could use it in the game. You can download those files with this link: Updated Tilesets.

After you have downloaded the files and extracted them right click on the Tilesets folder in the content project for the game, select Add and then Existing Item. Navigate to the tile sets and add them to the project. You will get a warning that the one tile set exists, just allow it to replace the existing file.

Open the code for the CharacterGeneratorScreen class and replace the CreateWorld method with this new version. You can also replace the same method in the LoadGameScreen class with this new version.

```
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);

    tilesetTexture = Game.Content.Load<Texture2D>(@"Tilesets\fire-tiles");
    AnimatedTileset animatedSet = new AnimatedTileset(tilesetTexture, 8, 1, 64, 64);

    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(5, 0, new Tile(14, 0));
    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, animatedSet, layer);
    map.AddTileset(tileset2);
    map.AddLayer(splatter);

    map.CollisionLayer.Collisions.Add(new Point(1, 0), CollisionType.Impassable);
    map.CollisionLayer.Collisions.Add(new Point(3, 0), CollisionType.Impassable);

    map.AnimatedTileLayer.AnimatedTiles.Add(new Point(5, 0), new AnimatedTile(0, 8));

    Level level = new Level(map);

    ChestData chestData = Game.Content.Load<ChestData>(@"Game\Chests\Plain Chest");

    Chest chest = new Chest(chestData);

    BaseSprite chestSprite = new BaseSprite(
        containers,
        new Rectangle(0, 0, 32, 32),
        new Point(10, 10));

    ItemSprite itemSprite = new ItemSprite(
        chest,
        chestSprite);
    level.Chests.Add(itemSprite);

    World world = new World(GameRef, GameRef.ScreenRectangle);
    world.Levels.Add(level);
    world.CurrentLevel = 0;

    AnimatedSprite s = new AnimatedSprite(
        GameRef.Content.Load<Texture2D>(@"SpriteSheets\Eliza"),
        AnimationManager.Instance.Animations);

    s.Position = new Vector2(5 * Engine.TileWidth, 5 * Engine.TileHeight);

    EntityData ed = new EntityData("Eliza", 10, 10, 10, 10, 10, 10,  "20|CON|12", "16|WIL|16",
"0|0|0");
    Entity e = new Entity("Eliza", ed, EntityGender.Female, EntityType.NPC);
```

```
    NonPlayerCharacter npc = new NonPlayerCharacter(e, s);
    npc.SetConversation("eliza1");

    world.Levels[world.CurrentLevel].Characters.Add(npc);
    GamePlayScreen.World = world;

    CreateConversation();


((NonPlayerCharacter)world.Levels[world.CurrentLevel].Characters[0]).SetConversation("eliza1");
}
```

What the new code does is first load the texture for the animated tile, fire-tiles. It then creates an animated tile set with the values 8, 1, 64 and 64 because the set is 8 tiles wide, 1 tile high and the tiles are 64 by 64. The next change is that when I call the constructor to create a new map I pass in the animated set that I just created. Then after add the collisions to the collision layer I add a new animated tile at position (5, 0).

At this point you should be able to build and run the game. Once you get to the game play screen you should see the camp file with animated flames.

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. 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