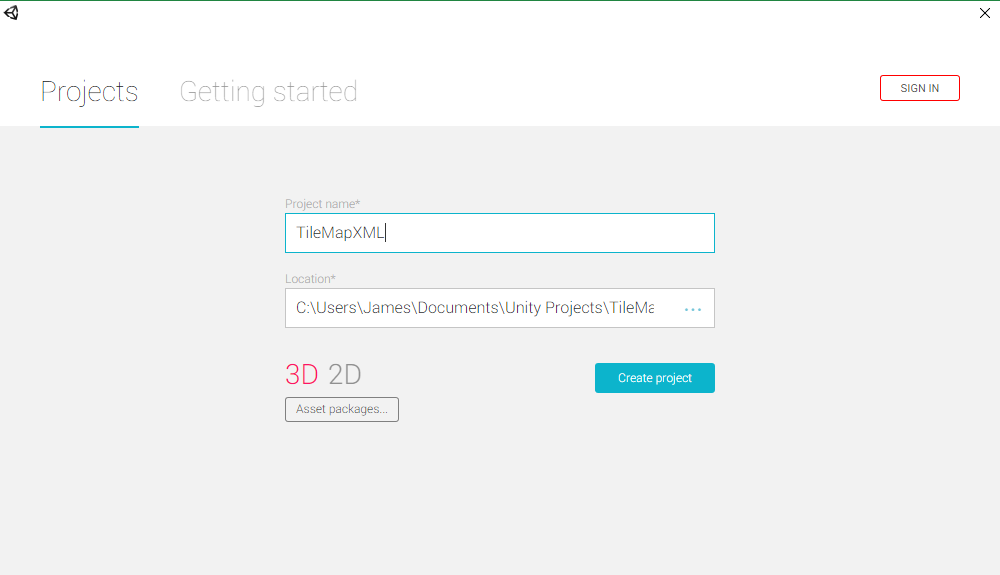
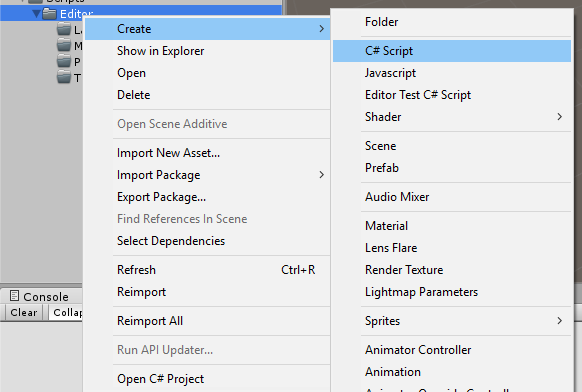
# Initial Unity Setup

So let’s get started by creating a new Unity Project, I called mine TileMapXML.

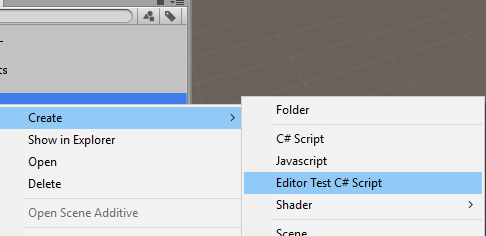


Next let’s start building our file structure. Create a folder called TileMapXML. Inside TileMapXML create a folder called Maps and a Folder called Scripts. We will be placing all of our maps in the maps folder and all of our scripts in the scripts folder. Now in the Maps folder create a folder called Tilesets, this is where all of our tileset textures and saved tilesets from Tiled will go. In the Scripts folder create a new folder called Editor, this is a special folder that Unity use for all of your Editor scripts, these scripts will not be included in your run time package. We will be placing all of the scripts representing the TMX file format in the editor folder. This means that you will not be able to load a TMX file when the game runs. We will be creating Scriptable objects using the editor, these scriptable objects are what we will load at run time. This will keep the file size of the game smaller, allowing you more resources to run your game. This is important if you are targeting a mobile platform like IOS or Android. In TileMapXML->Scripts->Editor create 4 more folders call them Layers, Map, Properties, and Tileset. In TileMapXML->Scripts->Editor->Layers create a folder called Objects. This completes the folder structure to help keep things organized.

In TileMapXML->Scripts->Editor create a new C# Script called TMX.



In TileMapXML->Scripts->Editor create a new Editor Test C# Script called TMXTest.



We will be using this to control the test that run in the Editor Test Runner. The test runner uses NUnit for testing. We will be creating test as we go to make sure whatever map that we are using loads all of its data correctly and gives us an easy indication if the map is compatible with our scripts or not.

Now all we need to do is create all of the C# scripts that represent the TMX file format. We will be referencing <http://doc.mapeditor.org/reference/tmx-map-format/>.

In TileMapXML->Scripts->Editor->Map create a new C# script called TMXMap. We will be adding other C# scripts to this folder latter when we get to building the map, enums that represent the attributes on the map i.e orientation.

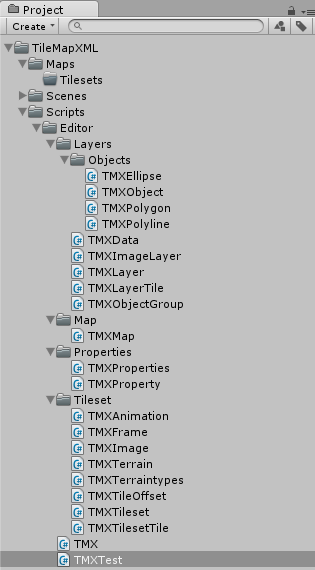
In TileMapXML->Scripts->Editor->Tileset create 8 new C# script called TMXTileset, TMXTileOffset, TMXImage, TMXTerrainTypes, TMXTerrain, TMXTilesetTile, TMXAnimation, and TMXFrame.

In TileMapXML->Scripts->Editor->Layers create 5 new C# script called TMXLayer, TMXData, TMXLayerTile, TMXObjectGroup, and TMXImageLayer.

In TileMapXML->Scripts->Editor->Layers->Objects create 4 new C# script called TMXObject, TMXEllipse, TMXPolygon, and TMXPolyline.

In TileMapXML->Scripts->Editor->Properties create 2 new C# script called TMXProperties, and TMXProperty.

When you are done you should have something similar to this.



Now open all of the Scripts except for TMXTest and remove all of the Unity generated code

using UnityEngine;

using System.Collections;

: MonoBehaviour

// Use this for initialization

void Start () {

}

// Update is called once per frame

void Update () {

}

Make sure to add the following using statements

using System.Collections.Generic;

using System.Xml.Serialization;

And put them in the appropriate namespaces TileMapXML for TMXMap and TMX, TileMapXML.Tileset for all of the scripts in the Tileset folder, TileMapXML.Layers for all of the scripts in the Layers folder, TileMapXML.Layers.Objects for all of the scripts in the Objects folder, and TileMapXML.Properties for the scripts in the properties folder.

My TMXMap script looks like

using System.Collections.Generic;

using System.Xml.Serialization;

using TileMapXML.Properties;

namespace TileMapXML

{

/// <summary>

/// <map>

/// • version: The TMX format version, generally 1.0.

/// • orientation: Map orientation.

/// Tiled supports "orthogonal", "isometric", "staggered" (since 0.9)

/// and "hexagonal" (since 0.11).

/// • renderorder: The order in which tiles on tile layers are rendered.

/// Valid values are right-down(the default), right-up, left-down and left-up.

/// In all cases, the map is drawn row-by-row.

/// (since 0.10, but only supported for orthogonal maps at the moment)

/// • width: The map width in tiles.

/// • height: The map height in tiles.

/// • tilewidth: The width of a tile.

/// • tileheight: The height of a tile.

/// • hexsidelength: Only for hexagonal maps.

/// Determines the width or height (depending on the staggered axis)

/// of the tile's edge, in pixels.

/// • staggeraxis: For staggered and hexagonal maps,

/// determines which axis("x" or "y") is staggered. (since 0.11)

/// • staggerindex: For staggered and hexagonal maps,

/// determines whether the "even" or "odd" indexes along the staggered axis are shifted. (since 0.11)

/// • backgroundcolor: The background color of the map.

/// (since 0.9, optional, may include alpha value since 0.15 in the form #AARRGGBB)

/// • nextobjectid: Stores the next available ID for new objects.

/// This number is stored to prevent reuse of the same ID after objects have been removed. (since 0.11)

///

/// The tilewidth and tileheight properties determine the general grid size of the map.

/// The individual tiles may have different sizes.

/// Larger tiles will extend at the top and right(anchored to the bottom left).

///

/// A map contains three different kinds of layers.

/// Tile layers were once the only type, and are simply called layer,

/// object layers have the objectgroup tag and image layers use the imagelayer tag.

/// The order in which these layers appear is the order in which the layers are rendered by Tiled.

///

/// Can contain: properties, tileset, layer, objectgroup, imagelayer

/// </summary>

public class TMXMap

{

#region attributes

#endregion

}//public class TMXMap

}//namespace TileMapXML

Notice that at the top of each script I copied the information from <http://doc.mapeditor.org/reference/tmx-map-format> into a summary block. This is something that most IDE’s like MonoDevelop and VisualStudio will use when displaying the intillisense about the class. This is not required it is just helpful to have some useful information about the class and how it is used. Also notice that I have added a region called attributes as well. This is where all of the XML attributes will be going and helps for code organization, this is also not required.