# Creating a Role Playing Game with XNA Game Studio 3.0
# Part 6
# Extending the Tile Engine

To follow along with this tutorial you will have to have read the previous tutorials to understand much of what it going on. You can find a list of tutorials here: XNA 3.0 Role Playing Game Tutorials You will also find the latest version of the project on the web site. If you want to follow along and type in the code from this PDF as you go, you can find the previous project at this link: XNA RPG 5

Today I'm going to make a few changes to the tile engine. Instead of using the array of integers for the tile engine I'm going to make it so that you can use layers. Each layer for now will be a little simple. All it is going to be is a 2D array of integers. Later I'm going to create a tile class. This class will hold more information about the tile. Integers will be fine for now though. I just want to display the tile.

The first thing I want to do is create a map layer class. This layer class will hold the information for the layer. For now it is a little simple but I created it so it can be extended to hold more information later. You can go a head and add a new class to the **CoreComponents** folder called **TileMapLayer**. This is the code for the class:

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

namespace New2DRPG.CoreComponents
{
    class TileMapLayer
    {
        int[,] map;

        public TileMapLayer(int x, int y)
        {
            map = new int[y, x];
        }

        public int TileMapWidth
        {
            get { return map.GetLength(1); }
        }

        public int TileMapHeight
        {
            get { return map.GetLength(0); }
        }

        public void SetTile(int x, int y, int tileIndex)
        {
            map[y, x] = tileIndex;
        }
```

```
        public int GetTile(int x, int y)
        {
            return map[y, x];
        }
    }
}
```

As you can see there is only the array for the map, the constuctor, a few properties and a method to get and set a tile in the map. I don't think there are any mysteries in this class. So I'm going to move on.

The next thing you will want to do is create a map class. I dediced to make the map class to be a **DrawableGameComponent**. It will do the drawing of the layers of the map. You could have done all of the layer drawing in the tile engine. Go a head and add a new **GameComponent** to the **CoreComponents** folder. As always I will give you the new code and then try and explain why I did what I did. This is the code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;


namespace New2DRPG.CoreComponents
{
    /// <summary>
    /// This is a game component that implements IUpdateable.
    /// </summary>
    public class TileMap : Microsoft.Xna.Framework.DrawableGameComponent
    {
        List<TileMapLayer> tileMapLayers = new List<TileMapLayer>();

        Random random = new Random();
        SpriteBatch spriteBatch;

        public TileMap(int tileMapWidth, int tileMapHeight, Game game)
            : base(game)
        {
            spriteBatch =
                (SpriteBatch)Game.Services.GetService(typeof(SpriteBatch));

            TileMapLayer layer = new TileMapLayer(tileMapWidth, tileMapHeight);

            for (int x = 0; x < 50; x++)
                for (int y = 0; y < 50; y++)
                    layer.SetTile(x, y, random.Next(0, 4));
            tileMapLayers.Add(layer);
        }
```

```
        public override void Initialize()
        {
            // TODO: Add your initialization code here

            base.Initialize();
        }

        public override void Update(GameTime gameTime)
        {
            // TODO: Add your update code here

            base.Update(gameTime);
        }

        public override void Draw(GameTime gameTime)
        {
            foreach (TileMapLayer layer in tileMapLayers)
            {
                for (int x = 0; x < TileEngine.TileMapWidth; x++)
                {
                    for (int y = 0; y < TileEngine.TileMapHeight; y++)
                    {
                        spriteBatch.Draw(TileEngine.TileSet,
                            new Rectangle(x * TileEngine.TileWidth -
                                    TileEngine.CameraXPosition,
                                y * TileEngine.TileHeight -
                                    TileEngine.CameraYPositon,
                                TileEngine.TileWidth,
                                TileEngine.TileHeight),
                            TileEngine.Tiles[layer.GetTile(x, y)],
                            Color.White);
                    }
                }
            }

            base.Draw(gameTime);
        }

        public virtual void Hide()
        {
            Visible = false;
            Enabled = false;
        }

        public virtual void Show()
        {
            Visible = true;
            Enabled = true;
        }
    }
}
```

The first thing I did was change it from **GameComponent** to **DrawableGameComponent**. The next thing you will see is that there is a **List** of tile layers. You will have to add the layers in the right order. The bottom layer needs to be added first. Then you would add new layers in the order you want them drawn. There is a **Random** object and a **SpriteBatch** object. The **Random** object will be used to

generate a random map and of course the **SpriteBatch** object will be used to draw the map.

For now the constructor takes the height and the width of the map to be created and a **Game** object. The consructor gets the **SpriteBatch** object and creates a new layer with the new width and height. I then created a new layer and the used the **SetTile** method to set the tiles to a random tile. Finally I added it to the **List** of tiles.

In the **Draw** method I used a **foreach** loop to go through all of the layers. Then inside of this loop I draw the layer. I did it the same as in the tile engine. You will see that there are many refrences to **static** objects in the **TileEngine** object. I have one for the tile set, the width and the height of the map in tiles, the height and width of the tiles, the X and Y position of the camera and the source rectangles of the tiles in the tile set.

There are also methods to show and hide the form. They are like all the other methods that do that. They just change the value of the **Visible** and the **Enabled** methods.
Now it is time to change your attention to the **TileEngine** class. There were a lot of changes to the **TileEngine**. I will show you the updated code for the **TileEngine** and then explain the changes.

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;


namespace New2DRPG.CoreComponents
{
    /// <summary>
    /// This is a game component that implements IUpdateable.
    /// </summary>
    public class TileEngine : Microsoft.Xna.Framework.DrawableGameComponent
    {
        SpriteBatch spriteBatch;
        static Texture2D tileset;

        static Rectangle[] tiles = new Rectangle[4];

        KeyboardState oldState;
        KeyboardState newState;

        TileMap map;

        List<GameComponent> childComponents = new List<GameComponent>();

        static int tileWidth = 80;
        static int tileHeight = 60;
```

```
static int tileMapWidth;
static int tileMapHeight;

static int screenWidth;
static int screenHeight;

static int mapWidthInPixels;
static int mapHeightInPixels;

static float cameraX;
static float cameraY;

Random random = new Random();
Camera camera = new Camera();

public TileEngine(Game game, Texture2D tileset, int tileMapWidth,
    int tileMapHeight)
    : base(game)
{
    map = new TileMap(tileMapWidth, tileMapHeight, game);
    childComponents.Add(map);

    spriteBatch =
        (SpriteBatch)Game.Services.GetService(typeof(SpriteBatch));

    TileEngine.tileset = tileset;
    TileEngine.tileMapWidth = tileMapWidth;
    TileEngine.tileMapHeight = tileMapHeight;

    tiles[0] = new Rectangle(0, 0, 128, 128);
    tiles[1] = new Rectangle(128, 0, 128, 128);
    tiles[2] = new Rectangle(0, 128, 128, 128);
    tiles[3] = new Rectangle(128, 128, 128, 128);

    screenWidth = game.Window.ClientBounds.Width;
    screenHeight = game.Window.ClientBounds.Height;
}

public static int TileMapWidth
{
    get { return tileMapWidth; }
}

public static int TileMapHeight
{
    get { return tileMapHeight; }
}

public static int TileWidth
{
    get { return tileWidth; }
}

public static int TileHeight
{
    get { return tileHeight; }
```

```
}

public static int ScreenWidth
{
    get { return screenWidth; }
}

public static int ScreenHeight
{
    get { return screenHeight; }
}

public static int MapWidthInPixels
{
    get { return mapWidthInPixels; }
}

public static int MapHeightInPixels
{
    get { return mapHeightInPixels; }
}

public static int CameraXPosition
{
    get { return (int)cameraX; }
}

public static int CameraYPositon
{
    get { return (int)cameraY; }
}

public static Texture2D TileSet
{
    get { return tileset; }
}

public static Rectangle[] Tiles
{
    get { return tiles; }
}

public override void Initialize()
{
    // TODO: Add your initialization code here

    base.Initialize();
}

public override void Update(GameTime gameTime)
{
    base.Update(gameTime);

    mapWidthInPixels = TileMapWidth * tileWidth;
    mapHeightInPixels = TileMapHeight * tileHeight;

    newState = Keyboard.GetState();
```

```
Vector2 motion = new Vector2();

if (newState.IsKeyDown(Keys.Up) || newState.IsKeyDown(Keys.NumPad8))
{
    motion.Y--;
}

if (newState.IsKeyDown(Keys.Right) || newState.IsKeyDown(Keys.NumPad6))
{
    motion.X++;
}

if (newState.IsKeyDown(Keys.Down) || newState.IsKeyDown(Keys.NumPad2))
{
    motion.Y++;
}

if (newState.IsKeyDown(Keys.Left) || newState.IsKeyDown(Keys.NumPad4))
{
    motion.X--;
}

if (newState.IsKeyDown(Keys.NumPad9))
{
    motion.X++;
    motion.Y--;
}

if (newState.IsKeyDown(Keys.NumPad3))
{
    motion.X++;
    motion.Y++;
}

if (newState.IsKeyDown(Keys.NumPad1))
{
    motion.X--;
    motion.Y++;
}

if (newState.IsKeyDown(Keys.NumPad7))
{
    motion.X--;
    motion.Y--;
}

if (motion != Vector2.Zero)
{
    motion.Normalize();
}

camera.Position += motion * camera.Speed;

camera.LockCamera();
TileEngine.cameraX = camera.Position.X;
TileEngine.cameraY = camera.Position.Y;
```

```
            oldState = newState;
        }

        public override void Draw(GameTime gameTime)
        {
            foreach (GameComponent child in childComponents)
            {
                if ((child is DrawableGameComponent) &&
                    ((DrawableGameComponent)child).Visible)
                {
                    ((DrawableGameComponent)child).Draw(gameTime);
                }
            }

            base.Draw(gameTime);
        }

        public virtual void Show()
        {
            Enabled = true;
            Visible = true;
            foreach (GameComponent child in childComponents)
            {
                child.Enabled = true;
                if ((child is DrawableGameComponent))
                    ((DrawableGameComponent)child).Visible = true;
            }
        }

        public virtual void Hide()
        {
            Enabled = false;
            Visible = false;
            foreach (GameComponent child in childComponents)
            {
                child.Enabled = false;
                if ((child is DrawableGameComponent))
                    ((DrawableGameComponent)child).Visible = false;
            }
        }

        public virtual void Pause()
        {
            Enabled = !Enabled;
            foreach (GameComponent child in childComponents)
            {
                child.Enabled = !child.Enabled;
            }
        }
    }
}
```

The first change you will see is that I made the tile set static. Since there is only going to be one tile set available at a time this was a good idea. I didn't make it public because I didn't want it modified out of the tile engine. I also made the source rectangles of the tiles in the tile set static and again didn't make them public so they can't be changed outside of the class.

Of course I got rid of the array of integers for the map. I added a **TileMap** object. Since this is a game component, I also created a list of game components for the tile engine. So I can easily get the X and Y position of the camera I created two static floats to X and Y position of the camera.

The constructor is just a little different. I created a new map and added it to the list of child game components. I also set the static variales, the tile set map width and height and source rectangles. Finally I get the height and width of the screen.

There are a total of twelve properties in this class. That might seem like a lot. Using the idea of data encapsulation (keeping data of an object in side a class) I like to only allow certain variables in a class to be able to be changed outside the class. There are properties for the width and height of the map in tiles, the width and the height of the tiles, the width and the height of the screen, the height and the width of the map in pixels. Since when you are drawing the tile map you are using rectangles I made the properties for the camera position integers. The **TileSet** property returns a **Texture2D**. The last one is a little interesting. There are two ways you can get the source rectangles. One way is to get all of them, like this:

```
Rectangle[] sourceRectangles = TileEngine.Tiles;
```

The other way is to just get one element is like I did in the **Draw** method of the **TileMap** class. You just use the index like you would an array.

```
Rectangle sourceRectangle = TileEngine.Tiles[index];
```

There is one change to the **Update** method. In the **Update** method after I lock the camera to the screen, I set the **cameraX** and **cameraY** variables.

There was a change in the **Draw** method. I removed the code to draw the map, as you can see it was moved to the **TileMap** class. I just copied the code from the **Draw** method in the **GameScreen** class. What this code does is loop through each component in the list of children components. If the component is a drawable component and it is visible it will call the draw method of the component.

I also changed the **Show**, **Hide** and **Pause** methods. I updated them so that when they are called they will show, hide or pause all of the children.

Well, that is it for another tutorial. Keep on coming back, I will try and have two or three ready each week.