Shadow Monsters – MonoGame Tutorial Series
Chapter 16
Editor – Part Four

This tutorial series is about creating a Pokemon style game with the MonoGame Framework called Shadow Monsters. The tutorials will make more sense if You read them in order as each tutorial builds on the previous tutorials. You can find the list of tutorials on my blog: Shadow Monsters. The source code for each tutorial will be available as well. I will be using Visual Studio 2019 Community for the series. The code should compile on the 2013, 2015 and 2017 versions as well.

I also want to mention that I assume You have a basic understanding of C# and MonoGame. If You don't I recommend that You learn basic C# and work with MonoGame a little. Enough to know the basics of propertys, properties, methods, classes and the MonoGame framework.

In this tutorial we will be continuing with the editor. The first thing that I will be covering is creating portals. Like characters and merchants I created a form for listing portals on the map and a form for creating/editing portals. I will start with the form for creating portals. Right click the ShadowEditor project in the Solution Explorer, select Add then Form (Windows Forms). Name this new form PortalForm. Open the PortalForm.Designer.cs file by expanding the node in front of PortalForm.cs in the Solution Explorer. Replace the contents with the following code.

```
namespace ShadowEditor
{
    partial class PortalForm
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise,
false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }
```

```
#region Windows Form Designer generated code

/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
    this.label1 = new System.Windows.Forms.Label();
    this.TxtDestination = new System.Windows.Forms.TextBox();
    this.label2 = new System.Windows.Forms.Label();
    this.TxtDestinationTile = new System.Windows.Forms.TextBox();
    this.label3 = new System.Windows.Forms.Label();
    this.TxtSourceTile = new System.Windows.Forms.TextBox();
    this.BtnOK = new System.Windows.Forms.Button();
    this.BtnCancel = new System.Windows.Forms.Button();
    this.TxtName = new System.Windows.Forms.TextBox();
    this.label4 = new System.Windows.Forms.Label();
    this.textBox2 = new System.Windows.Forms.TextBox();
    this.SuspendLayout();
    //
    // label1
    //
    this.label1.AutoSize = true;
    this.label1.Location = new System.Drawing.Point(25, 37);
    this.label1.Name = "label1";
    this.label1.Size = new System.Drawing.Size(86, 13);
    this.label1.TabIndex = 3;
    this.label1.Text = "Destination map:";
    //
    // TxtDestination
    //
    this.TxtDestination.Location = new System.Drawing.Point(117, 34);
    this.TxtDestination.Name = "TxtDestination";
    this.TxtDestination.Size = new System.Drawing.Size(100, 20);
    this.TxtDestination.TabIndex = 5;
    //
    // label2
    //
    this.label2.AutoSize = true;
    this.label2.Location = new System.Drawing.Point(32, 63);
    this.label2.Name = "label2";
    this.label2.Size = new System.Drawing.Size(79, 13);
    this.label2.TabIndex = 6;
    this.label2.Text = "Destination tile:";
    //
    // TxtDestinationTile
    //
    this.TxtDestinationTile.Location = new System.Drawing.Point(117, 60);
    this.TxtDestinationTile.Name = "TxtDestinationTile";
    this.TxtDestinationTile.Size = new System.Drawing.Size(100, 20);
    this.TxtDestinationTile.TabIndex = 7;
    //
    // label3
    //
    this.label3.AutoSize = true;
    this.label3.Location = new System.Drawing.Point(51, 89);
    this.label3.Name = "label3";
    this.label3.Size = new System.Drawing.Size(60, 13);
    this.label3.TabIndex = 8;
    this.label3.Text = "Source tile:";
    //
    // TxtSourceTile
    //
    this.TxtSourceTile.Location = new System.Drawing.Point(117, 86);
    this.TxtSourceTile.Name = "TxtSourceTile";
    this.TxtSourceTile.Size = new System.Drawing.Size(100, 20);
    this.TxtSourceTile.TabIndex = 9;
    //
    // BtnOK
    //
    this.BtnOK.Location = new System.Drawing.Point(244, 5);
    this.BtnOK.Name = "BtnOK";
    this.BtnOK.Size = new System.Drawing.Size(75, 23);
    this.BtnOK.TabIndex = 10;
    this.BtnOK.Text = "OK";
    this.BtnOK.UseVisualStyleBackColor = true;
    this.BtnOK.Click += new System.EventHandler(this.BtnOK_Click);
    //
    // BtnCancel
    //
    this.BtnCancel.Location = new System.Drawing.Point(244, 34);
    this.BtnCancel.Name = "BtnCancel";
    this.BtnCancel.Size = new System.Drawing.Size(75, 23);
    this.BtnCancel.TabIndex = 0;
    this.BtnCancel.Text = "Cancel";
    this.BtnCancel.UseVisualStyleBackColor = true;
    this.BtnCancel.Click += new System.EventHandler(this.BtnCancel_Click);
    //
    // TxtName
    //
    this.TxtName.Location = new System.Drawing.Point(117, 8);
    this.TxtName.Name = "TxtName";
    this.TxtName.Size = new System.Drawing.Size(100, 20);
    this.TxtName.TabIndex = 4;
    //
    // label4
    //
    this.label4.AutoSize = true;
    this.label4.Location = new System.Drawing.Point(32, 11);
    this.label4.Name = "label4";
    this.label4.Size = new System.Drawing.Size(66, 13);
    this.label4.TabIndex = 0;
    this.label4.Text = "Portal name:";
    //
    // textBox2
    //
    this.textBox2.Location = new System.Drawing.Point(117, -27);
    this.textBox2.Name = "textBox2";
    this.textBox2.Size = new System.Drawing.Size(100, 20);
    this.textBox2.TabIndex = 1;
    //
    // FormPortal
    //
    this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
    this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
    this.ClientSize = new System.Drawing.Size(337, 147);
    this.Controls.Add(this.BtnCancel);
    this.Controls.Add(this.BtnOK);
```

```
            this.Controls.Add(this.label3);
            this.Controls.Add(this.label2);
            this.Controls.Add(this.TxtSourceTile);
            this.Controls.Add(this.TxtDestinationTile);
            this.Controls.Add(this.textBox2);
            this.Controls.Add(this.TxtName);
            this.Controls.Add(this.label4);
            this.Controls.Add(this.TxtDestination);
            this.Controls.Add(this.label1);
            this.Name = "FormPortal";
            this.Text = "FormPortal";
            this.Load += new System.EventHandler(this.FormPortal_Load);
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.TextBox TxtDestination;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.TextBox TxtDestinationTile;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.TextBox TxtSourceTile;
        private System.Windows.Forms.Button BtnOK;
        private System.Windows.Forms.Button BtnCancel;
        private System.Windows.Forms.TextBox TxtName;
        private System.Windows.Forms.Label label4;
        private System.Windows.Forms.TextBox textBox2;
    }
}
```

Since it is generated code I'm not going to go over. Now open the code for PortalForm by selecting it in the SolutionExplorer and pressing F7 or right clicking it and selecting View Code. Replace the generated code with the following code.

```
using ShadowMonsters.TileEngine;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace ShadowEditor
{
    public partial class PortalForm : Form
    {
        public Portal Portal { get; set; }
        public bool OkPressed { get; private set; }
        public string PortalName { get; internal set; }

        public PortalForm(string name = null, Portal portal = null)
        {
            InitializeComponent();
            if (name != null && portal != null)
            {
                TxtName.Enabled = false;
                TxtName.Text = name;
                TxtDestination.Text = portal.DestinationLevel;
                TxtDestinationTile.Text = portal.DestinationTile.X + ":" +
portal.DestinationTile.Y;
                TxtSourceTile.Text = portal.SourceTile.X + ":" + portal.SourceTile.Y;
            }
        }

        private void FormPortal_Load(object sender, EventArgs e)
        {
            BtnOK.Click += BtnOK_Click;
            BtnCancel.Click += BtnCancel_Click;
        }

        private void BtnCancel_Click(object sender, EventArgs e)
        {
            OkPressed = false;
            Close();
        }

        private void BtnOK_Click(object sender, EventArgs e)
        {
            if (string.IsNullOrEmpty(TxtName.Text))
            {
                MessageBox.Show("You must enter the name of the portal.");
                return;
            }

            if (string.IsNullOrEmpty(TxtDestination.Text))
            {
                MessageBox.Show("You must enter a destination level.");
                return;
            }

            if (string.IsNullOrEmpty(TxtDestinationTile.Text))
            {
                MessageBox.Show("You must enter the destination tile.");

                return;
            }

            if (string.IsNullOrEmpty(TxtSourceTile.Text))
            {
                MessageBox.Show("You must enter the source3 tile.");
                return;
            }

            string[] source = TxtSourceTile.Text.Split(':');
            string[] dest = TxtDestinationTile.Text.Split(':');

            Portal = new Portal(
                new Microsoft.Xna.Framework.Point(
                    int.Parse(source[0]),
                    int.Parse(source[1])
                    ),
                new Microsoft.Xna.Framework.Point(
                    int.Parse(dest[0]),
                    int.Parse(dest[1])
                    ),
                TxtDestination.Text);

            PortalName = TxtName.Text;
            OkPressed = true;

            Close();
        }
    }
}
```

There is a Portal property to return a created portal if the OK button has been clicked. There is also a string property that returns the name of the portal. Finally there is a bool that returns if the OK button was clicked.

The constructor takes an optional string parameter and an optional Portal parameter. If both are not null the fields of the form are filled out. The Portal Name is set to the string parameter. I disable the text box because I don't want it edited. The destination map is set to the destination map of the Portal passed in. The source and destination tiles are set to the X and Y coordinates of the source and destination tiles of the portal separated by colons.

The Load event handler for the form wires the event handler for the Click event of the OK and Cancel buttons.

The event handler for the Click event of the OK button validates the inputs. That the fields have been populated. It then splits the source and destination fields into their parts on the colon. I then create a portal passing in a new point using the parts of the source tile, a new point using the parts of the destination tile and the destination map. The PortalName property is set to the Text property of TxtName and OKPressed is set to true. Finally the form is closed.

The event handler for the Click event of the Cancel button sets OKPressed to false. It then close the form.

Now I will add the list for for portals. Right click the ShadowEditor project in the Solution Explorer, select Add and then Form (Windows Forms). Name this new form PortalListForm. Open the PortalListForm.Designer.cs file and replace the code with the following.

```
namespace ShadowEditor
{
    partial class PortalListForm
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise, 
false.</param>
        protected override void Dispose(bool disposing)
```

```
{
    if (disposing && (components != null))
    {
        components.Dispose();
    }
    base.Dispose(disposing);
}

#region Windows Form Designer generated code

/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
    this.LBPortals = new System.Windows.Forms.ListBox();
    this.BtnAdd = new System.Windows.Forms.Button();
    this.BtnEdit = new System.Windows.Forms.Button();
    this.BtnDelete = new System.Windows.Forms.Button();
    this.SuspendLayout();
    //
    // LBCharacters
    //
    this.LBPortals.FormattingEnabled = true;
    this.LBPortals.Location = new System.Drawing.Point(12, 12);
    this.LBPortals.Name = "LBCharacters";
    this.LBPortals.Size = new System.Drawing.Size(287, 277);
    this.LBPortals.TabIndex = 0;
    //
    // BtnAdd
    //
    this.BtnAdd.Location = new System.Drawing.Point(305, 12);
    this.BtnAdd.Name = "BtnAdd";
    this.BtnAdd.Size = new System.Drawing.Size(75, 23);
    this.BtnAdd.TabIndex = 1;
    this.BtnAdd.Text = "Add";
    this.BtnAdd.UseVisualStyleBackColor = true;
    //
    // BtnEdit
    //
    this.BtnEdit.Location = new System.Drawing.Point(306, 42);
    this.BtnEdit.Name = "BtnEdit";
    this.BtnEdit.Size = new System.Drawing.Size(75, 23);
    this.BtnEdit.TabIndex = 2;
    this.BtnEdit.Text = "Edit";
    this.BtnEdit.UseVisualStyleBackColor = true;
    //
    // BtnDelete
    //
    this.BtnDelete.Location = new System.Drawing.Point(306, 71);
    this.BtnDelete.Name = "BtnDelete";
    this.BtnDelete.Size = new System.Drawing.Size(75, 23);
    this.BtnDelete.TabIndex = 3;
    this.BtnDelete.Text = "Delete";
    this.BtnDelete.UseVisualStyleBackColor = true;
    //
    // FormCharacterList
    //
    this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
```

```
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(394, 305);
            this.Controls.Add(this.BtnDelete);
            this.Controls.Add(this.BtnEdit);
            this.Controls.Add(this.BtnAdd);
            this.Controls.Add(this.LBPortals);
            this.Name = "PortalListForm";
            this.Text = "PortalListForm";
            this.ResumeLayout(false);

        }

        #endregion

        private System.Windows.Forms.ListBox LBPortals;
        private System.Windows.Forms.Button BtnAdd;
        private System.Windows.Forms.Button BtnEdit;
        private System.Windows.Forms.Button BtnDelete;
    }
}
```

Since it is generated code I'm not going to explain it. Open the code for ShadowListForm by selecting it in the Solution Explorer and pressing the F7 key or right clicking the ShadowListForm and selecting View Code. Replace the code with the following.

```
using ShadowMonsters.TileEngine;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace ShadowEditor
{
    public partial class PortalListForm : Form
    {
        private readonly TileMap map;
        private readonly Editor game;

        public PortalListForm(TileMap map, Editor game)
        {
            InitializeComponent();
            this.map = map;
            this.game = game;

            foreach (var p in map.PortalLayer.Portals.Keys)
            {
                LBPortals.Items.Add(p);
            }

            BtnAdd.Click += BtnAdd_Click;
            BtnEdit.Click += BtnEdit_Click;
            BtnDelete.Click += BtnDelete_Click;
        }
        private void BtnAdd_Click(object sender, EventArgs e)
        {
            PortalForm form = new PortalForm();
            form.ShowDialog();

            if (form.OkPressed)
            {
                map.PortalLayer.AddPortal(form.PortalName, form.Portal);
                LBPortals.Items.Add(form.PortalName);
            }
        }

        private void BtnEdit_Click(object sender, EventArgs e)
        {
            if (LBPortals.SelectedIndex >= 0)
            {
                PortalForm form = new PortalForm(
                    LBPortals.SelectedItem.ToString(),
                    map.PortalLayer.Portals[LBPortals.SelectedItem.ToString()]);

                form.ShowDialog();

                if (form.OkPressed)
                {
                    map.PortalLayer.Portals[form.PortalName] = form.Portal;
                }
            }
        }

        private void BtnDelete_Click(object sender, EventArgs e)
        {
            if (LBPortals.SelectedIndex >= 0)
            {
                map.PortalLayer.Portals.Remove(LBPortals.SelectedItem.ToString());
                LBPortals.Items.RemoveAt(LBPortals.SelectedIndex);
            }
        }
    }
}
```

There are a TileMap field and a Editor field that provide access to the map and the editor. The constructor sets the fields to the parameters passed in. It then loops over all of the keys in the Portals on the portal layer. The key is then added to the list of portals on the form. It then wires the event handlers for the Click event of the Add, Edit and Delete button.

The Click event handler for the Add button creates a PortalForm and calls the DisplayDialog method to display the form. If the OK button was pressed on the form the portal is added passing in the PortalName and Portal properties of the form. It then adds the PortalName property to the list box of portals.

The Click event handler for the Edit button checks to see if an item in the list box of portals has been selected. If it has it creates a PortalForm passing in the name of the portal and the portal. It then calls the ShowDialog method. If the OK button was pressed it sets the portal by that name to the Portal of the form.

The Click event handler for the Delete button checks to see if there is an item selected in the list box. If there is an item selected in the list box I remove the portal using the SelectedItem property of the list box cast to a string because items in list boxes are objects. I then use the RemoveAt method of the items collection passing in the SelectedIndex property.

The next thing that I'm going to add is planned shadow monster battles like the legendary pokemon battles in the Pokemon game. I did that by adding another layer to the TileMap class. Right click the TileMap folder in the Solution Explorer, select Add and then Class. Name this new class MonsterLayer. Here is the code.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;
using ShadowMonsters.ShadowMonsters;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ShadowMonsters.TileEngine
{
    public class MonsterLayer
    {
        #region Field Region

        private Dictionary<Point, ShadowMonster> monsters = new Dictionary<Point,
ShadowMonster>();

        #endregion

        #region Property Region

        public Dictionary<Point, ShadowMonster> Monsters
        {
            get { return monsters; }
        }

        #endregion

        #region Constructor Region
        #endregion

        #region Method Region

        public void Draw(GameTime gameTime, SpriteBatch spriteBatch, Camera camera)
        {
            spriteBatch.Begin(
                SpriteSortMode.Deferred,
                BlendState.AlphaBlend,
                SamplerState.PointClamp,
                null,
                null,
                null,
                camera.Transformation);

            foreach (Point p in monsters.Keys)
            {
                spriteBatch.Draw(
                    monsters[p].Texture,
                    new Rectangle(
                        p.X * Engine.TileWidth,
                        p.Y * Engine.TileHeight,
                        Engine.TileWidth,
                        Engine.TileHeight),
                    null,
                    Color.White);
            }

            spriteBatch.End();
        }

        public void Save(BinaryWriter writer)
        {
            writer.Write(monsters.Count);

            foreach (Point p in monsters.Keys)
            {
                writer.Write(p.X);
                writer.Write(p.Y);
                monsters[p].Save(writer);
            }
        }

        public static MonsterLayer Load(ContentManager content, BinaryReader reader)
        {
            MonsterLayer layer = new MonsterLayer();

            int count = reader.ReadInt32();

            for (int i = 0; i < count; i++)
            {
                Point p = new Point(reader.ReadInt32(), reader.ReadInt32());
                ShadowMonster monster = ShadowMonster.Load(content, reader.ReadString());

                layer.Monsters.Add(p, monster);
            }

            return layer;
        }
        #endregion
    }
}
```

There is a field monsters that is a Dictionary<Point, ShadowMonster> that holds the shadow monsters on the map. There is also a property to expose its value. The Draw method draws the shadow monsters at the tile they were created on. The Save method requires a BinaryReader parameter. It writes the number of shadow monsters. It then loops over the collection writing the coordinates separately then calling the Save method of the shadow monster. The Load method requires a ContentManager parameter and a BinaryReader parameter. It creates a new layer. The number of shadow monsters is read in. It loops that many times. It creates a Point by reading in the X and Y coordinates. It then calls the Load method of the ShadowMonster class passing in the ContentManager and the string for the shadow monster. The ShadowMonster is then added to the collection of shadow monsters using the point for a key.

I made changes to the TileMap class to handle the new layer. If you have existing maps that you want to keep I would follow this process. Update the Save method but don't change the Load method. Run the editor and load the map. Once the map is loaded save it and exit the editor. Now that the map format has been updated update the Load method to handle the new format. There were several changes to the TileMap class so I'm just going to give the code for the entire class. Here is the code.

```
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;
using System.IO;
using ShadowMonsters.Characters;

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

        string mapName;
        TileLayer groundLayer;
        TileLayer edgeLayer;
        TileLayer buildingLayer;
        TileLayer decorationLayer;
        CharacterLayer characterLayer;
        CollisionLayer collisionLayer;
        PortalLayer portalLayer;
        MonsterLayer monsterLayer;

        int mapWidth;
        int mapHeight;

        TileSet tileSet;

        #endregion

        #region Property Region

        public string MapName
        {
            get { return mapName; }
            private set { mapName = value; }
        }

        public TileSet TileSet
        {
            get { return tileSet; }
            set { tileSet = value; }
        }

        public TileLayer GroundLayer
        {
            get { return groundLayer; }
            set { groundLayer = value; }
        }
```

```
public TileLayer EdgeLayer
{
    get { return edgeLayer; }
    set { edgeLayer = value; }
}


public TileLayer BuildingLayer
{
    get { return buildingLayer; }
    set { buildingLayer = value; }
}

public int MapWidth
{
    get { return mapWidth; }
}

public int MapHeight
{
    get { return mapHeight; }
}

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

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

public CharacterLayer CharacterLayer => characterLayer;

public CollisionLayer CollisionLayer => collisionLayer;

public PortalLayer PortalLayer => portalLayer;

public MonsterLayer MonsterLayer => monsterLayer;

#endregion

#region Constructor Region

private TileMap()
{
    characterLayer = new CharacterLayer();
    collisionLayer = new CollisionLayer();
    portalLayer = new PortalLayer();
    monsterLayer = new MonsterLayer();
}

private TileMap(TileSet tileSet, string mapName)
    : this()
{
    this.tileSet = tileSet;
    this.mapName = mapName;
}
```

```
public TileMap(
    TileSet tileSet,
    TileLayer groundLayer,
    TileLayer edgeLayer,
    TileLayer buildingLayer,
    TileLayer decorationLayer,
    string mapName)
    : this(tileSet, mapName)
{
    this.groundLayer = groundLayer;
    this.edgeLayer = edgeLayer;
    this.buildingLayer = buildingLayer;
    this.decorationLayer = decorationLayer;

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

#endregion

#region Method Region

public void SetGroundTile(int x, int y, int set, int index)
{
    groundLayer.SetTile(x, y, set, index);
}

public Tile GetGroundTile(int x, int y)
{
    return groundLayer.GetTile(x, y);
}

public void SetEdgeTile(int x, int y, int set, int index)
{
    edgeLayer.SetTile(x, y, set, index);
}

public Tile GetEdgeTile(int x, int y)
{
    return edgeLayer.GetTile(x, y);
}

public void SetBuildingTile(int x, int y, int set, int index)
{
    buildingLayer.SetTile(x, y, set, index);
}

public Tile GetBuildingTile(int x, int y)
{
    return buildingLayer.GetTile(x, y);
}

public void SetDecorationTile(int x, int y, int set, int index)
{
    decorationLayer.SetTile(x, y, set, index);
}

public Tile GetDecorationTile(int x, int y)
{
```

```
            return decorationLayer.GetTile(x, y);
        }

        public void FillEdges()
        {
            for (int y = 0; y < mapHeight; y++)
            {
                for (int x = 0; x < mapWidth; x++)
                {
                    edgeLayer.SetTile(x, y, -1, -1);
                }
            }
        }

        public void FillBuilding()
        {
            for (int y = 0; y < mapHeight; y++)
            {
                for (int x = 0; x < mapWidth; x++)
                {
                    buildingLayer.SetTile(x, y, -1, -1);
                }
            }
        }

        public void FillDecoration()
        {
            for (int y = 0; y < mapHeight; y++)
            {
                for (int x = 0; x < mapWidth; x++)
                {
                    decorationLayer.SetTile(x, y, -1, -1);
                }
            }
        }

        public void Update(GameTime gameTime)
        {
            if (groundLayer != null)
                groundLayer.Update(gameTime);

            if (edgeLayer != null)
                edgeLayer.Update(gameTime);

            if (buildingLayer != null)
                buildingLayer.Update(gameTime);

            if (decorationLayer != null)
                decorationLayer.Update(gameTime);

            characterLayer.Update(gameTime);
        }

        public void Draw(GameTime gameTime, SpriteBatch spriteBatch, Camera camera, bool debug
= false)
        {
            if (groundLayer != null)
                groundLayer.Draw(gameTime, spriteBatch, tileSet, camera);

            if (edgeLayer != null)
```

```
                edgeLayer.Draw(gameTime, spriteBatch, tileSet, camera);

            characterLayer.Draw(gameTime, spriteBatch, camera);

            if (buildingLayer != null)
                buildingLayer.Draw(gameTime, spriteBatch, tileSet, camera);

            if (decorationLayer != null)
                decorationLayer.Draw(gameTime, spriteBatch, tileSet, camera);

            if (debug)
            {
                collisionLayer.Draw(spriteBatch, camera);
                portalLayer.Draw(spriteBatch, camera);
            }

            monsterLayer.Draw(gameTime, spriteBatch, camera);
        }

        public bool Save(BinaryWriter writer)
        {
            writer.Write(mapName);

            characterLayer.Save(writer);
            tileSet.Save(writer);
            edgeLayer.Save(writer);
            groundLayer.Save(writer);
            decorationLayer.Save(writer);
            buildingLayer.Save(writer);
            portalLayer.Save(writer);
            collisionLayer.Save(writer);
            monsterLayer.Save(writer);

            return true;
        }

        public static TileMap Load(ContentManager content, BinaryReader reader)
        {
            TileMap map = new TileMap();

            map.mapName = reader.ReadString();
            map.characterLayer = CharacterLayer.Load(content, reader);
            map.tileSet = TileSet.Load(content, reader);
            map.edgeLayer = TileLayer.Load(reader);
            map.groundLayer = TileLayer.Load(reader);
            map.decorationLayer = TileLayer.Load(reader);
            map.buildingLayer = TileLayer.Load(reader);
            map.portalLayer = PortalLayer.Load(reader);
            map.collisionLayer = CollisionLayer.Load(reader);
            map.monsterLayer = MonsterLayer.Load(content, reader);

            map.mapWidth = map.groundLayer.Width;
            map.mapHeight = map.groundLayer.Height;

            return map;
        }
        #endregion
    }
}
```

There is a new MonsterLayer field and property that exposes its value. In the Draw method I call the Draw method of the MonsterLayer passing in the required parameters. In the Save method after saving the other layers I call the Save method of the layer. In the Load method I call the Load method of the layer after loading all of the other layers.

Now I'm going to add the forms for adding shadow monsters to the map. Like I did for characters and merchants I added forms for creating a shadow monster and a form that holds the list of shadow monsters on the map. The form for creating a shadow monster works by entering the name of the shadow monster and the source tile. Right click the ShadowEditor projects in the Solution Explorer, select Add and then Form (Windows Forms). Name this form ShadowMonsterForm. Open the ShadowMonster.Designer.cs file by expanding the node beside the form. Replace the code with the following.

```
namespace ShadowEditor
{
    partial class ShadowMonsterForm
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise,
false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.label1 = new System.Windows.Forms.Label();
            this.label2 = new System.Windows.Forms.Label();
            this.TxtShadowMonster = new System.Windows.Forms.TextBox();
            this.TxtSource = new System.Windows.Forms.TextBox();
            this.BtnOK = new System.Windows.Forms.Button();
            this.BtnCancel = new System.Windows.Forms.Button();
            this.SuspendLayout();
            //
            // label1
            //
            this.label1.AutoSize = true;
            this.label1.Location = new System.Drawing.Point(2, 21);
            this.label1.Margin = new System.Windows.Forms.Padding(6, 0, 6, 0);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(166, 25);
            this.label1.TabIndex = 0;
            this.label1.Text = "Shadow Monster:";
            //
            // label2
            //
            this.label2.AutoSize = true;
            this.label2.Location = new System.Drawing.Point(50, 73);
            this.label2.Margin = new System.Windows.Forms.Padding(6, 0, 6, 0);
            this.label2.Name = "label2";
            this.label2.Size = new System.Drawing.Size(118, 25);
            this.label2.TabIndex = 1;
            this.label2.Text = "Source Tile:";
            //
            // TxtShadowMonster
            //
            this.TxtShadowMonster.Location = new System.Drawing.Point(178, 16);
            this.TxtShadowMonster.Margin = new System.Windows.Forms.Padding(6, 6, 6, 6);
            this.TxtShadowMonster.Name = "TxtShadowMonster";
            this.TxtShadowMonster.Size = new System.Drawing.Size(180, 29);
            this.TxtShadowMonster.TabIndex = 2;
            //
            // TxtSource
            //
            this.TxtSource.Location = new System.Drawing.Point(178, 68);
            this.TxtSource.Margin = new System.Windows.Forms.Padding(6, 6, 6, 6);
            this.TxtSource.Name = "TxtSource";
            this.TxtSource.Size = new System.Drawing.Size(180, 29);
            this.TxtSource.TabIndex = 3;
            //
            // BtnOK
            //
            this.BtnOK.Location = new System.Drawing.Point(370, 7);
            this.BtnOK.Margin = new System.Windows.Forms.Padding(6, 6, 6, 6);
            this.BtnOK.Name = "BtnOK";
            this.BtnOK.Size = new System.Drawing.Size(138, 42);
            this.BtnOK.TabIndex = 4;
            this.BtnOK.Text = "OK";
            this.BtnOK.UseVisualStyleBackColor = true;
            //
            // BtnCancel
            //
            this.BtnCancel.Location = new System.Drawing.Point(370, 68);
            this.BtnCancel.Margin = new System.Windows.Forms.Padding(6, 6, 6, 6);
            this.BtnCancel.Name = "BtnCancel";
            this.BtnCancel.Size = new System.Drawing.Size(138, 42);
            this.BtnCancel.TabIndex = 6;
            this.BtnCancel.Text = "Cancel";
            this.BtnCancel.UseVisualStyleBackColor = true;
            //
            // ShadowMonsterForm
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(11F, 24F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(552, 120);
            this.Controls.Add(this.BtnCancel);
            this.Controls.Add(this.BtnOK);
            this.Controls.Add(this.TxtSource);
            this.Controls.Add(this.TxtShadowMonster);
```

```
            this.Controls.Add(this.label2);
            this.Controls.Add(this.label1);
            this.Margin = new System.Windows.Forms.Padding(6, 6, 6, 6);
            this.Name = "ShadowMonsterForm";
            this.Text = "FormShadowMonster";
            this.Load += new System.EventHandler(this.FormShadowMonster_Load);
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.TextBox TxtShadowMonster;
        private System.Windows.Forms.TextBox TxtSource;
        private System.Windows.Forms.Button BtnOK;
        private System.Windows.Forms.Button BtnCancel;
    }
}
```

I'm not going to explain the generated code. Select the ShadowMonsterForm in the Solution Explorer and press the F7 key or right click the ShadowMonsterForm in the Solution Explorer and select View Code to bring up the code. Replace the code with 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.Threading.Tasks;
using System.Windows.Forms;

namespace ShadowEditor
{
    public partial class ShadowMonsterForm : Form
    {
        public string ShadowMonster { get; set; }
        public string Source { get; set; }
        public bool OKPressed { get; internal set; }

        public ShadowMonsterForm()
        {
            InitializeComponent();
        }

        public ShadowMonsterForm(string avatar, string source)
        {
            InitializeComponent();

            TxtShadowMonster.Text = avatar;
            TxtSource.Text = source;
        }

        private void FormShadowMonster_Load(object sender, EventArgs e)
        {
```

```
        BtnOK.Click += BtnOK_Click;
        BtnCancel.Click += BtnCancel_Click;
    }

    private void BtnOK_Click(object sender, EventArgs e)
    {
        if (string.IsNullOrEmpty(TxtShadowMonster.Text))
        {d
            MessageBox.Show("You must enter the name of the avatar.");
            return;
        }

        if (string.IsNullOrEmpty(TxtSource.Text))
        {
            MessageBox.Show("You must enter the source tile.");
            return;
        }

        if (!TxtSource.Text.Contains(':'))
        {
            MessageBox.Show("Source tile must be seperated by a :.");
            return;
        }

        string[] parts = TxtSource.Text.Split(':');

        if (!int.TryParse(parts[0], out int xcoord))
        {
            MessageBox.Show("X-coordinate must be numeric.");
            return;
        }

        if (xcoord < 0)
        {
            MessageBox.Show("X coordinate must not be negative.");
            return;
        }

        if (!int.TryParse(parts[1], out int ycoord))
        {
            MessageBox.Show("Y coordinate must be numeric.");
            return;
        }

        if (ycoord < 0)
        {
            MessageBox.Show("Y coordinate must not be negative.");
            return;
        }

        ShadowMonster = TxtShadowMonster.Text;
        Source = TxtSource.Text;
        OKPressed = true;
        Close();
    }

    private void BtnCancel_Click(object sender, EventArgs e)
    {
        OKPressed = false;
        Close();
        }
    }
}
```

There are properties to expose the values of the form. There is also a property to expose if the OK button was pressed like the other forms. There are two constructors for the class. The first requires no parameters. The second constructor takes two strings as parameters. The first is the name of the shadow monster and the second is the source tile with the X and Y coordinates separated by a colon. It then sets the Text properties of the fields to the values passed in. The event handler for the Load event of the form wires the handlers for the Click events of the OK and Cancel buttons.

The event handler for the Click event of the OK button I check to see if Text property of TxtShadowMonst and TxtSource are empty or null. If either is I display a message box stating the fields must be filled and exit the method. If the source tile does not contain a colon I display a message box and exit the method. I then split the source tile into its parts based on the colon. If parsing the first part of the source tile fails I display an error message and exit the method. I then check to see if the value is negative and if it is I exit the method. If parsing the second part of the source tile I display an error message and exit the method. If the value is negative I display a message and exit the method. If the validation succeeds I set the properties to the Text properties of the corresponding text boxes. I set the OKPressed to true and close the form.

In the event handler for the Click event of the Cancel button I set the OKPressed property to false. Then I close the form.

Next I added the form that displays lists. Right click the ShadowEditor project in the Solution Explorer, select Add then Form (Windows Forms). Name the form ShadowMonsterListForm. Expand the node beside the ShadowMonsterListForm then open the code for the designer. Replace the code with the following.

```
namespace ShadowEditor
{
    partial class ShadowMonsterListForm
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise, 
false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }
```

```
#region Windows Form Designer generated code

/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
    this.LBMonsters = new System.Windows.Forms.ListBox();
    this.BtnAdd = new System.Windows.Forms.Button();
    this.BtnEdit = new System.Windows.Forms.Button();
    this.BtnDelete = new System.Windows.Forms.Button();
    this.SuspendLayout();
    //
    // LBCharacters
    //
    this.LBMonsters.FormattingEnabled = true;
    this.LBMonsters.Location = new System.Drawing.Point(12, 12);
    this.LBMonsters.Name = "LBCharacters";
    this.LBMonsters.Size = new System.Drawing.Size(287, 277);
    this.LBMonsters.TabIndex = 0;
    //
    // BtnAdd
    //
    this.BtnAdd.Location = new System.Drawing.Point(305, 12);
    this.BtnAdd.Name = "BtnAdd";
    this.BtnAdd.Size = new System.Drawing.Size(75, 23);
    this.BtnAdd.TabIndex = 1;
    this.BtnAdd.Text = "Add";
    this.BtnAdd.UseVisualStyleBackColor = true;
    //
    // BtnEdit
    //
    this.BtnEdit.Location = new System.Drawing.Point(306, 42);
    this.BtnEdit.Name = "BtnEdit";
    this.BtnEdit.Size = new System.Drawing.Size(75, 23);
    this.BtnEdit.TabIndex = 2;
    this.BtnEdit.Text = "Edit";
    this.BtnEdit.UseVisualStyleBackColor = true;
    //
    // BtnDelete
    //
    this.BtnDelete.Location = new System.Drawing.Point(306, 71);
    this.BtnDelete.Name = "BtnDelete";
    this.BtnDelete.Size = new System.Drawing.Size(75, 23);
    this.BtnDelete.TabIndex = 3;
    this.BtnDelete.Text = "Delete";
    this.BtnDelete.UseVisualStyleBackColor = true;
    //
    // FormCharacterList
    //
    this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
    this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
    this.ClientSize = new System.Drawing.Size(394, 305);
    this.Controls.Add(this.BtnDelete);
    this.Controls.Add(this.BtnEdit);
    this.Controls.Add(this.BtnAdd);
    this.Controls.Add(this.LBMonsters);
    this.Name = "FormCharacterList";
    this.Text = "FormCharacterList";
```

```
            this.ResumeLayout(false);

        }

        #endregion

        private System.Windows.Forms.ListBox LBMonsters;
        private System.Windows.Forms.Button BtnAdd;
        private System.Windows.Forms.Button BtnEdit;
        private System.Windows.Forms.Button BtnDelete;
    }
}
```

Now I will add the logic to the form. Right click the ShadowMonsterListForm.cs file in the Solution Explorer and select View Code or select the ShadowMonsterListForm.cs file in the Solution Explorer and press the F7 key. Replace the code with the following.

```
using ShadowMonsters.ShadowMonsters;
using ShadowMonsters.TileEngine;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace ShadowEditor
{
    public partial class ShadowMonsterListForm : Form
    {
        private readonly TileMap map;
        private readonly Editor game;

        public ShadowMonsterListForm(TileMap map, Editor game)
        {
            InitializeComponent();

            this.map = map;
            this.game = game;

            foreach (var m in map.MonsterLayer.Monsters.Values)
            {
                LBMonsters.Items.Add(m.Name);
            }

            BtnAdd.Click += BtnAdd_Click;
            BtnEdit.Click += BtnEdit_Click;
            BtnDelete.Click += BtnDelete_Click;
        }

        private void BtnAdd_Click(object sender, EventArgs e)
        {
            ShadowMonsterForm form = new ShadowMonsterForm();
            form.ShowDialog();

            if (form.OKPressed)
            {
                string[] parts = form.Source.Split(':');
                map.MonsterLayer.Monsters.Add(
                    new Microsoft.Xna.Framework.Point(
                        int.Parse(parts[0]),
                        int.Parse(parts[1])),

ShadowMonsterManager.GetShadowMonster(form.ShadowMonster.ToLowerInvariant()));
                LBMonsters.Items.Add(form.ShadowMonster);
            }
        }

        private void BtnEdit_Click(object sender, EventArgs e)
        {
            if (LBMonsters.SelectedIndex < 0 || LBMonsters.Items.Count == 0)
            {
                return;
            }

            if (LBMonsters.SelectedItem.ToString() == "Unknown")
            {
                return;
            }

            foreach (var p in map.MonsterLayer.Monsters.Keys)
            {
                if (map.MonsterLayer.Monsters[p].Name.ToLowerInvariant() ==
                    LBMonsters.SelectedItem.ToString().ToLowerInvariant())
                {
                    ShadowMonsterForm form = new ShadowMonsterForm(
                        LBMonsters.SelectedItem.ToString(),
                        p.X + ":" + p.Y);

                    form.ShowDialog();

                    if (form.OKPressed)
                    {
                        map.MonsterLayer.Monsters[p] =
                            ShadowMonsterManager.GetShadowMonster(
                                form.ShadowMonster.ToLowerInvariant());
                        LBMonsters.Items[LBMonsters.SelectedIndex] = form.ShadowMonster;
                        break;
                    }
                }
            }
        }

        private void BtnDelete_Click(object sender, EventArgs e)
        {
            if (LBMonsters.SelectedIndex < 0)
            {
                return;
            }

            foreach (var p in map.MonsterLayer.Monsters.Keys)
            {
                if (map.MonsterLayer.Monsters[p].Name.ToLowerInvariant() ==
                    LBMonsters.SelectedItem.ToString().ToLowerInvariant())
                {
                    map.MonsterLayer.Monsters.Remove(p);
                    break;
                }
            }

            LBMonsters.Items.RemoveAt(LBMonsters.SelectedIndex);
        }
    }
}
```

There are TileMap and Editor fields like the other list forms. The constructor requires a TileMap and Editor parameter. It sets the fields to the values passed in. It loops over the shadow monsters on the layer. It adds the Name property to the list box. Next it wires the Click event for the Add, Edit and Delete buttons.

In the event handler for the Click event of the Add button I create a ShadowMonsterForm then call the ShowDialog method on the form to display it. If the OKPressed property of the form is true I split the Source property into its parts on the colon. I then add the monster to the layer creating a Point from the parts and getting the shadow monster from the ShadowMonsterManager passing in the ShadowMonster property converted to a lower case invariant string.

The Click event handler of the Edit button I check to see if the SelectedIndex of the list box is less than zero or the Count property is 0. If either is true I exit the method. In a foreach loop I loop over the keys in the Monsters collection of the monster layer. I then compare the name of the monster converted to lower case invariant to the selected item of the list box. If they are equal I create a ShadowMonsterForm passing in the SelectedItem of the list box and the coordinates of the key separated by a colon. Next I call the ShowDialog method to display the form. If OKPressed on the form is true I set the monster with the key to be the shadow monster with the name as a lower invariant string. Because the collection has been modified I break out of the loop.

In the Click event handler for the Delete button I check to see if the SelectedIndex is less than zero and if it is I exit the method. I then loop over the keys of the Monsters collection. If the Name property of the monster and the selected item as lower invariant strings are the same I remove the monster with the key and exit the loop. I then remove the item with the selected index of the list box.

I had to update the LoadContent method of the Editor class to have access to the shadow monsters. I call the FillMoves of the MoveManager and the FromFile method of the ShadowMonsterManager. Update the LoadContent method of the Editor class to the following.

```
        protected override void LoadContent()
        {
            // Create a new SpriteBatch, which can be used to draw textures.
            spriteBatch = new SpriteBatch(GraphicsDevice);

            MoveManager.FillMoves();
            ShadowMonsterManager.FromFile(@".\Content\ShadowMonsters.txt", Content);

            controls = new ControlManager(Content.Load<SpriteFont>(@"InterfaceFont"))
            {
```

```
    AcceptInput = true
};

Texture2D texture = new Texture2D(
    GraphicsDevice,
    64,
    64);

Color[] area = new Color[64 * 64];

for (int i = 0; i < area.Length; i++)
{
    area[i] = new Color(255, 255, 255, 128);
}

texture.SetData(area);

CollisionLayer.Texture = texture;

area = new Color[64 * 64];

for (int i = 0; i < area.Length; i++)
{
    area[i] = new Color(0, 0, 255, 128);
}

texture.SetData(area);

PortalLayer.Texture = texture;

Texture2D background = new Texture2D(
    graphics.GraphicsDevice,
    100,
    150);

Color[] buffer = new Color[100 * 150];

for (int i = 0; i < buffer.Length; i++)
{
    buffer[i] = Color.White;
}

background.SetData(buffer);

Texture2D cursor = new Texture2D(
    GraphicsDevice,
    100,
    12);

buffer = new Color[100 * 12];

for (int i = 0; i < buffer.Length; i++)
{
    buffer[i] = Color.Blue;
}

cursor.SetData(buffer);

pbTileset = new PictureBox(
    new Texture2D(
                graphics.GraphicsDevice,
                512,
                512),
            new Rectangle(
                64 * 18,
                128,
                1920 - 64 * 18,
                512));
        pbPreview = new PictureBox(
            new Texture2D(
                graphics.GraphicsDevice,
                64,
                64),
            new Rectangle(
                64 * 18,
                25,
                64,
                64));

        lbLayers = new ListBox(
            background,
            cursor)
        {
            Position = new Vector2(64 * 18, 800),
            TabStop = false
        };

        lbLayers.Items.Add("Ground");
        lbLayers.Items.Add("Edge");
        lbLayers.Items.Add("Decorations");
        lbLayers.Items.Add("Building");

        lbTileSets = new ListBox(
            background,
            cursor)
        {
            Position = new Vector2(64 * 18 + 150, 800),
            TabStop = false
        };

        lbTileSets.SelectionChanged += LbTileSets_SelectionChanged;

        controls.Add(lbTileSets);
        controls.Add(lbLayers);
        controls.Add(pbTileset);
        controls.Add(pbPreview);
    }
```

I made a couple updates to the editor. I added different brush sizes, the default 1 x 1 along with a 2 x 2, 4 x 4 and 8 x 8. I also wired displaying the ShadowMonsterListForm. The last thing that I implemented painting collisions. Add the following fields and update the Update method of the Editor class to the following.

```
    bool Paint = true;
    int brushSize = 1;

    protected override void Update(GameTime gameTime)
    {
        Point tile;
```

```
            if (!IsActive)
                return;

            if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed ||
Keyboard.GetState().IsKeyDown(Keys.Escape))
                Exit();

            frameCount++;

            if (Xin.CheckKeyReleased(Keys.D1))
            {
                brushSize = 1;
            }

            if (Xin.CheckKeyReleased(Keys.D2))
            {
                brushSize = 2;
            }

            if (Xin.CheckKeyReleased(Keys.D4))
            {
                brushSize = 4;
            }

            if (Xin.CheckKeyReleased(Keys.D8))
            {
                brushSize = 8;
            }
            if (Xin.CheckKeyReleased(Keys.N) && frameCount > 5)
            {
                NewMapForm form = new NewMapForm(GraphicsDevice);

                form.ShowDialog();

                if (form.OkPressed)
                {
                    map = form.TileMap;
                    pbTileset.Image = map.TileSet.Textures[0];
                    pbPreview.Image = map.TileSet.Textures[0];

                    lbTileSets.Items.Clear();
                    foreach (string s in map.TileSet.TextureNames)
                        lbTileSets.Items.Add(s);

                    lbTileSets.SelectedIndex = 0;
                    pbPreview.SourceRectangle =
                        map.TileSet.SourceRectangles[0];
                    pbTileset.SourceRectangle = new Rectangle(
                        0,
                        0,
                        map.TileSet.Textures[0].Width,
                        map.TileSet.Textures[0].Height);
                }

                frameCount = 0;
            }

            Vector2 position = new Vector2
            {
                X = Xin.MouseAsPoint.X + camera.Position.X,
                Y = Xin.MouseAsPoint.Y + camera.Position.Y
            };

            tile = Engine.VectorToCell(position);

            if (Xin.CheckKeyReleased(Keys.P))
            {
                Paint = !Paint;
            }

            if (map != null && viewPort.Contains(Xin.MouseAsPoint))
            {
                if (Xin.MouseState.LeftButton == ButtonState.Pressed)
                {
                    if (Paint)
                    {
                        switch (lbLayers.SelectedIndex)
                        {
                            case 0:
                                for (int i = 0; i < brushSize; i++)
                                {
                                    for (int j = 0; j < brushSize; j++)
                                    {
                                        map.SetGroundTile(tile.X + i, tile.Y + j,
lbTileSets.SelectedIndex, selectedTile);
                                    }
                                }
                                break;
                            case 1:
                                for (int i = 0; i < brushSize; i++)
                                {
                                    for (int j = 0; j < brushSize; j++)
                                    {
                                        map.SetEdgeTile(tile.X + i, tile.Y + j,
lbTileSets.SelectedIndex, selectedTile);
                                    }
                                }
                                break;
                            case 2:
                                for (int i = 0; i < brushSize; i++)
                                {
                                    for (int j = 0; j < brushSize; j++)
                                    {
                                        map.SetDecorationTile(tile.X + i, tile.Y + j,
lbTileSets.SelectedIndex, selectedTile);
                                    }
                                }
                                break;
                            case 3:
                                for (int i = 0; i < brushSize; i++)
                                {
                                    for (int j = 0; j < brushSize; j++)
                                    {
                                        map.SetBuildingTile(tile.X + i, tile.Y + j,
lbTileSets.SelectedIndex, selectedTile);
                                    }
                                }
                                break;
                        }
```

```
        }
        else
        {
            if (!map.CollisionLayer.Collisions.ContainsKey(
                new Point(
                    tile.X,
                    tile.Y)))
            {
                map.CollisionLayer.Collisions.Add(
                    new Point(
                    tile.X,
                    tile.Y),
                    ShadowMonsters.TileEngine.CollisionType.Impassable);
            }
        }
    }

    if (Xin.MouseState.RightButton == ButtonState.Pressed)
    {
        if (Paint)
        {
            switch (lbLayers.SelectedIndex)
            {
                case 0:
                    for (int i = 0; i < brushSize; i++)
                    {
                        for (int j = 0; j < brushSize; j++)
                        {
                            map.SetGroundTile(tile.X + i, tile.Y + j, -1, -1);
                        }
                    }
                    break;
                case 1:
                    for (int i = 0; i < brushSize; i++)
                    {
                        for (int j = 0; j < brushSize; j++)
                        {
                            map.SetEdgeTile(tile.X + i, tile.Y + j, -1, -1);
                        }
                    }
                    break;
                case 2:
                    for (int i = 0; i < brushSize; i++)
                    {
                        for (int j = 0; j < brushSize; j++)
                        {
                            map.SetDecorationTile(tile.X + i, tile.Y + j, -1, -1);
                        }
                    }
                    break;
                case 3:
                    for (int i = 0; i < brushSize; i++)
                    {
                        for (int j = 0; j < brushSize; j++)
                        {
                            map.SetBuildingTile(tile.X + i, tile.Y + j, -1, -1);
                        }
                    }
                    break;
            }
```

```
                }
                else
                {
                    if (map.CollisionLayer.Collisions.ContainsKey(
                        new Point(
                            tile.X,
                            tile.Y)))
                    {
                        map.CollisionLayer.Collisions.Remove(
                        new Point(
                            tile.X,
                            tile.Y));
                    }
                }
            }
        }

        if (pbTileset != null &&
            pbTileset.DestinationRectangle.Contains(
            Xin.MouseAsPoint) &&
            Xin.CheckMouseReleased(MouseButtons.Left))
        {
            Point previewPoint = new Point(
                Xin.MouseAsPoint.X - pbTileset.DestinationRectangle.X,
                Xin.MouseAsPoint.Y - pbTileset.DestinationRectangle.Y);
            float xScale = (float)map.TileSet.Textures[lbTileSets.SelectedIndex].Width /
                pbTileset.DestinationRectangle.Width;

            float yScale = (float)map.TileSet.Textures[lbTileSets.SelectedIndex].Height /
                pbTileset.DestinationRectangle.Height;

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

            Point clickedTile = new Point(
                tilesetPoint.X / map.TileSet.TileWidth,
                tilesetPoint.Y / map.TileSet.TileHeight);

            selectedTile = clickedTile.Y * map.TileSet.TilesWide + clickedTile.X;
            pbPreview.SourceRectangle =
                map.TileSet.SourceRectangles[selectedTile];
        }

        if (Xin.CheckKeyReleased(Keys.C) && frameCount > 5 && map != null)
        {
            CharacterListForm frm = new CharacterListForm(map, this);
            frm.ShowDialog();
        }

        if (Xin.CheckKeyReleased(Keys.M) && frameCount > 5 && map != null)
        {
            MerchantListForm frm = new MerchantListForm(map, this);
            frm.ShowDialog();
        }

        if (Xin.CheckKeyReleased(Keys.D) && frameCount > 5 && map != null)
        {
            PortalListForm frm = new PortalListForm(map, this);
            frm.ShowDialog();
        }

        if (Xin.CheckKeyReleased(Keys.S) && frameCount > 5 && map != null)
        {
            ShadowMonsterListForm frm = new ShadowMonsterListForm(map, this);
            frm.ShowDialog();
        }
        if (Xin.CheckKeyReleased(Keys.F1) && frameCount > 5)
        {
            WF.SaveFileDialog sfd = new WF.SaveFileDialog();
            sfd.Filter = "Tile Map (*.map)|*.map";
            WF.DialogResult result = sfd.ShowDialog();

            if (result == WF.DialogResult.OK)
            {
                SaveMap(sfd.FileName);
            }
        }

        if (Xin.CheckKeyReleased(Keys.F2) && frameCount > 5)
        {
            WF.OpenFileDialog ofd = new WF.OpenFileDialog();
            ofd.Filter = "Tile Map (*.map)|*.map";
            WF.DialogResult result = ofd.ShowDialog();

            if (result == WF.DialogResult.OK)
            {
                LoadMap(ofd.FileName);

                pbTileset.Image = map.TileSet.Textures[0];
                pbPreview.Image = map.TileSet.Textures[0];

                lbTileSets.Items.Clear();
                foreach (string s in map.TileSet.TextureNames)
                    lbTileSets.Items.Add(s);

                lbTileSets.SelectedIndex = 0;
                pbPreview.SourceRectangle =
                    map.TileSet.SourceRectangles[0];
                pbTileset.SourceRectangle = new Rectangle(
                    0,
                    0,
                    map.TileSet.Textures[0].Width,
                    map.TileSet.Textures[0].Height);
            }
        }

        HandleScrollMap();
        controls.Update(gameTime);
        base.Update(gameTime);
    }
```

The paint field is a bool that determines if you are painting tiles or you are painting collisions. I'm going to wrap this tutorial up here. The brushSize is the size of the brush. If the 1 key has been released the brushSize is set to 1. If the 2 key has been released the brushSize is set to 2. If the 4 key has been released the brushSize is set to 4. If the 8 key has been released the brushSize is set to 8.

The code for painting has changed. For the left mouse button there is a check if Paint is true. If it is I paint tiles. In the paint code there are nested for loops that loop for brushSize. The calls to set tile add the loop indexes. If Paint is false I check to see if there is not I add a collision. For the right mouse button there is a check to see if Paint is true. If it is the painting code works the same as for the left button with the nested loops. If it is false I check to see if there is a key with the tile. If there is I remove the collision.

There is a check to see if the S key has been released, frameCount is less than 5 and map is not null. If all are true I create a ShadowMonsterListForm and call the ShowDialog method to display the form.

I will be starting work on the next tutorial shortly. Keep checking back on the blog for news on that tutorial. I hope to have it up in the next week or so.

I wish You the best in Your MonoGame Programming Adventures!