Shadow Monsters – MonoGame Tutorial Series
Chapter 21
Creating Shadow Monsters

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. 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 want to mention though that the series is released as Creative Commons 3.0 Attribution. It means that you are free to use any of the code or graphics in your own game, even for commercial use, with attribution. Just give credit to Cynthia McMahon and add a link to my site, https://mygameprogrammingadventures.blogspot.com. Screenshots of your project and/or a video of game play would be appreciated.

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 fields, properties, methods, classes and the MonoGame framework.

This tutorial is going to bounce back to creating content. Specifically we will be creating shadow monsters and moves. I will also be encrypting the shadow monster file so that players can't edit it to make their shadow monsters stronger. It's always a good idea to not have your content as plain text.

To get started we need a form for creating a shadow monster. Right click the ShadowEditor project in the Solution Explorer, select Add and then Form (Windows Form). Name this new form DefinitionForm. Expand the DefinitionForm node in the Solution Explorer and open the DefinitionForm.Designer.cs file. Replace the code with the following.

```
namespace ShadowEditor
{
    partial class DefinitionForm
    {
        /// <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.TxtKey = new System.Windows.Forms.TextBox();
            this.TxtName = new System.Windows.Forms.TextBox();
            this.TxtCost = new System.Windows.Forms.TextBox();
            this.TxtLevel = new System.Windows.Forms.TextBox();
            this.TxtAttack = new System.Windows.Forms.TextBox();
            this.cboElement = new System.Windows.Forms.ComboBox();
            this.label2 = new System.Windows.Forms.Label();
            this.label3 = new System.Windows.Forms.Label();
            this.label4 = new System.Windows.Forms.Label();
            this.label5 = new System.Windows.Forms.Label();
            this.label6 = new System.Windows.Forms.Label();
            this.TxtDefense = new System.Windows.Forms.TextBox();
            this.label7 = new System.Windows.Forms.Label();
            this.TxtSpeed = new System.Windows.Forms.TextBox();
            this.label8 = new System.Windows.Forms.Label();
            this.label9 = new System.Windows.Forms.Label();
            this.TxtHealth = new System.Windows.Forms.TextBox();
            this.LBMoves = new System.Windows.Forms.ListBox();
            this.label10 = new System.Windows.Forms.Label();
            this.BtnAdd = new System.Windows.Forms.Button();
            this.BtnRemove = new System.Windows.Forms.Button();
            this.BtnOK = new System.Windows.Forms.Button();
            this.BtnCancel = new System.Windows.Forms.Button();
            this.CboMoves = new System.Windows.Forms.ComboBox();
            this.label11 = new System.Windows.Forms.Label();
            this.TxtMoveLevel = new System.Windows.Forms.TextBox();
            this.label12 = new System.Windows.Forms.Label();
            this.SuspendLayout();
            //
            // label1
            //
            this.label1.AutoSize = true;
            this.label1.Location = new System.Drawing.Point(44, 15);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(28, 13);
            this.label1.TabIndex = 0;
            this.label1.Text = "Key:";
            //
            // TxtKey
            //
            this.TxtKey.Location = new System.Drawing.Point(78, 12);
            this.TxtKey.Name = "TxtKey";
            this.TxtKey.Size = new System.Drawing.Size(121, 20);
            this.TxtKey.TabIndex = 1;
            //
            // TxtName
            //
            this.TxtName.Location = new System.Drawing.Point(78, 38);
            this.TxtName.Name = "TxtName";
```

```
this.TxtName.Size = new System.Drawing.Size(121, 20);
this.TxtName.TabIndex = 2;
//
// TxtCost
//
this.TxtCost.Location = new System.Drawing.Point(78, 89);
this.TxtCost.Name = "TxtCost";
this.TxtCost.Size = new System.Drawing.Size(121, 20);
this.TxtCost.TabIndex = 3;
//
// TxtLevel
//
this.TxtLevel.Location = new System.Drawing.Point(78, 115);
this.TxtLevel.Name = "TxtLevel";
this.TxtLevel.Size = new System.Drawing.Size(121, 20);
this.TxtLevel.TabIndex = 4;
//
// TxtAttack
//
this.TxtAttack.Location = new System.Drawing.Point(78, 141);
this.TxtAttack.Name = "TxtAttack";
this.TxtAttack.Size = new System.Drawing.Size(121, 20);
this.TxtAttack.TabIndex = 5;
//
// cboElement
//
this.cboElement.FormattingEnabled = true;
this.cboElement.Location = new System.Drawing.Point(78, 62);
this.cboElement.Name = "cboElement";
this.cboElement.Size = new System.Drawing.Size(121, 21);
this.cboElement.TabIndex = 6;
//
// label2
//
this.label2.AutoSize = true;
this.label2.Location = new System.Drawing.Point(-1, 41);
this.label2.Name = "label2";
this.label2.Size = new System.Drawing.Size(73, 13);
this.label2.TabIndex = 7;
this.label2.Text = "Display name:";
//
// label3
//
this.label3.AutoSize = true;
this.label3.Location = new System.Drawing.Point(24, 70);
this.label3.Name = "label3";
this.label3.Size = new System.Drawing.Size(48, 13);
this.label3.TabIndex = 8;
this.label3.Text = "Element:";
//
// label4
//
this.label4.AutoSize = true;
this.label4.Location = new System.Drawing.Point(41, 92);
this.label4.Name = "label4";
this.label4.Size = new System.Drawing.Size(31, 13);
this.label4.TabIndex = 9;
this.label4.Text = "Cost:";
//
// label5
```

```
//
this.label5.AutoSize = true;
this.label5.Location = new System.Drawing.Point(37, 118);
this.label5.Name = "label5";
this.label5.Size = new System.Drawing.Size(36, 13);
this.label5.TabIndex = 10;
this.label5.Text = "Level:";
//
// label6
//
this.label6.AutoSize = true;
this.label6.Location = new System.Drawing.Point(31, 144);
this.label6.Name = "label6";
this.label6.Size = new System.Drawing.Size(41, 13);
this.label6.TabIndex = 11;
this.label6.Text = "Attack:";
//
// TxtDefense
//
this.TxtDefense.Location = new System.Drawing.Point(78, 167);
this.TxtDefense.Name = "TxtDefense";
this.TxtDefense.Size = new System.Drawing.Size(121, 20);
this.TxtDefense.TabIndex = 12;
//
// label7
//
this.label7.AutoSize = true;
this.label7.Location = new System.Drawing.Point(22, 170);
this.label7.Name = "label7";
this.label7.Size = new System.Drawing.Size(50, 13);
this.label7.TabIndex = 13;
this.label7.Text = "Defense:";
//
// TxtSpeed
//
this.TxtSpeed.Location = new System.Drawing.Point(78, 193);
this.TxtSpeed.Name = "TxtSpeed";
this.TxtSpeed.Size = new System.Drawing.Size(121, 20);
this.TxtSpeed.TabIndex = 14;
//
// label8
//
this.label8.AutoSize = true;
this.label8.Location = new System.Drawing.Point(24, 196);
this.label8.Name = "label8";
this.label8.Size = new System.Drawing.Size(41, 13);
this.label8.TabIndex = 15;
this.label8.Text = "Speed:";
//
// label9
//
this.label9.AutoSize = true;
this.label9.Location = new System.Drawing.Point(32, 222);
this.label9.Name = "label9";
this.label9.Size = new System.Drawing.Size(41, 13);
this.label9.TabIndex = 16;
this.label9.Text = "Health:";
//
// TxtHealth
//
```

```
this.TxtHealth.Location = new System.Drawing.Point(78, 219);
this.TxtHealth.Name = "TxtHealth";
this.TxtHealth.Size = new System.Drawing.Size(121, 20);
this.TxtHealth.TabIndex = 17;
//
// LBMoves
//
this.LBMoves.FormattingEnabled = true;
this.LBMoves.Location = new System.Drawing.Point(78, 245);
this.LBMoves.Name = "LBMoves";
this.LBMoves.Size = new System.Drawing.Size(120, 95);
this.LBMoves.TabIndex = 18;
//
// label10
//
this.label10.AutoSize = true;
this.label10.Location = new System.Drawing.Point(24, 245);
this.label10.Name = "label10";
this.label10.Size = new System.Drawing.Size(42, 13);
this.label10.TabIndex = 19;
this.label10.Text = "Moves:";
//
// BtnAdd
//
this.BtnAdd.Location = new System.Drawing.Point(204, 245);
this.BtnAdd.Name = "BtnAdd";
this.BtnAdd.Size = new System.Drawing.Size(75, 23);
this.BtnAdd.TabIndex = 20;
this.BtnAdd.Text = "Add";
this.BtnAdd.UseVisualStyleBackColor = true;
//
// BtnRemove
//
this.BtnRemove.Location = new System.Drawing.Point(204, 274);
this.BtnRemove.Name = "BtnRemove";
this.BtnRemove.Size = new System.Drawing.Size(75, 23);
this.BtnRemove.TabIndex = 21;
this.BtnRemove.Text = "Remove";
this.BtnRemove.UseVisualStyleBackColor = true;
//
// BtnOK
//
this.BtnOK.Location = new System.Drawing.Point(204, 12);
this.BtnOK.Name = "BtnOK";
this.BtnOK.Size = new System.Drawing.Size(75, 23);
this.BtnOK.TabIndex = 22;
this.BtnOK.Text = "OK";
this.BtnOK.UseVisualStyleBackColor = true;
//
// BtnCancel
//
this.BtnCancel.Location = new System.Drawing.Point(205, 41);
this.BtnCancel.Name = "BtnCancel";
this.BtnCancel.Size = new System.Drawing.Size(75, 23);
this.BtnCancel.TabIndex = 23;
this.BtnCancel.Text = "Cancel";
this.BtnCancel.UseVisualStyleBackColor = true;
//
// CboMoves
//
```

```
this.CboMoves.FormattingEnabled = true;
this.CboMoves.Location = new System.Drawing.Point(78, 346);
this.CboMoves.Name = "CboMoves";
this.CboMoves.Size = new System.Drawing.Size(121, 21);
this.CboMoves.TabIndex = 24;
//
// label11
//
this.label11.AutoSize = true;
this.label11.Location = new System.Drawing.Point(32, 349);
this.label11.Name = "label11";
this.label11.Size = new System.Drawing.Size(37, 13);
this.label11.TabIndex = 25;
this.label11.Text = "Move:";
//
// TxtMoveLevel
//
this.TxtMoveLevel.Location = new System.Drawing.Point(78, 373);
this.TxtMoveLevel.Name = "TxtMoveLevel";
this.TxtMoveLevel.Size = new System.Drawing.Size(120, 20);
this.TxtMoveLevel.TabIndex = 26;
//
// label12
//
this.label12.AutoSize = true;
this.label12.Location = new System.Drawing.Point(4, 376);
this.label12.Name = "label12";
this.label12.Size = new System.Drawing.Size(69, 13);
this.label12.TabIndex = 27;
this.label12.Text = "Unlocked At:";
//
// DefinitionForm
//
this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
this.ClientSize = new System.Drawing.Size(291, 404);
this.Controls.Add(this.label12);
this.Controls.Add(this.TxtMoveLevel);
this.Controls.Add(this.label11);
this.Controls.Add(this.CboMoves);
this.Controls.Add(this.BtnCancel);
this.Controls.Add(this.BtnOK);
this.Controls.Add(this.BtnRemove);
this.Controls.Add(this.BtnAdd);
this.Controls.Add(this.label10);
this.Controls.Add(this.LBMoves);
this.Controls.Add(this.TxtHealth);
this.Controls.Add(this.label9);
this.Controls.Add(this.label8);
this.Controls.Add(this.TxtSpeed);
this.Controls.Add(this.label7);
this.Controls.Add(this.TxtDefense);
this.Controls.Add(this.label6);
this.Controls.Add(this.label5);
this.Controls.Add(this.label4);
this.Controls.Add(this.label3);
this.Controls.Add(this.label2);
this.Controls.Add(this.cboElement);
this.Controls.Add(this.TxtAttack);
this.Controls.Add(this.TxtLevel);
```

```
            this.Controls.Add(this.TxtCost);
            this.Controls.Add(this.TxtName);
            this.Controls.Add(this.TxtKey);
            this.Controls.Add(this.label1);
            this.Name = "DefinitionForm";
            this.Text = "DefinitionForm";
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.TextBox TxtKey;
        private System.Windows.Forms.TextBox TxtName;
        private System.Windows.Forms.TextBox TxtCost;
        private System.Windows.Forms.TextBox TxtLevel;
        private System.Windows.Forms.TextBox TxtAttack;
        private System.Windows.Forms.ComboBox cboElement;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.Label label4;
        private System.Windows.Forms.Label label5;
        private System.Windows.Forms.Label label6;
        private System.Windows.Forms.TextBox TxtDefense;
        private System.Windows.Forms.Label label7;
        private System.Windows.Forms.TextBox TxtSpeed;
        private System.Windows.Forms.Label label8;
        private System.Windows.Forms.Label label9;
        private System.Windows.Forms.TextBox TxtHealth;
        private System.Windows.Forms.ListBox LBMoves;
        private System.Windows.Forms.Label label10;
        private System.Windows.Forms.Button BtnAdd;
        private System.Windows.Forms.Button BtnRemove;
        private System.Windows.Forms.Button BtnOK;
        private System.Windows.Forms.Button BtnCancel;
        private System.Windows.Forms.ComboBox CboMoves;
        private System.Windows.Forms.Label label11;
        private System.Windows.Forms.TextBox TxtMoveLevel;
        private System.Windows.Forms.Label label12;
    }
}
```

This code was generated by Visual Studio when I added controls to the form and set their name and text where applicable. I'm not going to explain it. Open the code view for the form by switching to the tab and pressing F7 or right clicking the form in the Solution Explorer and selecting View Code. Replace the code with the following.

```
using ShadowMonsters.ShadowMonsters;
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;
```

```csharp
namespace ShadowEditor
{
    public partial class DefinitionForm : Form
    {
        public string ShadowMonster { get; private set; }
        public bool OkPressed { get; private set; }

        public DefinitionForm()
        {
            InitializeComponent();

            this.Load += DefinitionForm_Load;

            foreach (var v in Enum.GetNames(typeof(ShadowMonsterElement)))
            {
                cboElement.Items.Add(v);
            }

            BtnOK.Click += BtnOK_Click;
            BtnCancel.Click += BtnCancel_Click;
            BtnAdd.Click += BtnAdd_Click;
            BtnRemove.Click += BtnRemove_Click;
        }

        public DefinitionForm(ShadowMonster m) : this()
        {
            TxtKey.Text = m.Name;
            TxtName.Text = m.DisplayName;
            cboElement.SelectedItem = m.Element;
            TxtCost.Text = m.Cost.ToString();
            TxtAttack.Text = m.BaseAttack.ToString();
            TxtDefense.Text = m.BaseDefense.ToString();
            TxtSpeed.Text = m.BaseSpeed.ToString();
            TxtHealth.Text = m.BaseHealth.ToString();

            foreach (IMove move in m.KnownMoves.Values)
            {
                LBMoves.Items.Add(move.Name + ":" + move.UnlockedAt);
            }
        }

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

            if (string.IsNullOrEmpty(TxtKey.Text))
            {
                MessageBox.Show("You must enter the name of the shadow monster.");
                return;
            }

            if (string.IsNullOrEmpty(TxtCost.Text)
                || int.TryParse(TxtCost.Text, out int cost))
            {
                MessageBox.Show("You must enter a numeric value for cost");
                return;
            }

            if (string.IsNullOrEmpty(TxtLevel.Text)
                || int.TryParse(TxtLevel.Text, out int level))
            {
                MessageBox.Show("You must enter a numeric value for level,");
                return;
            }

            if (string.IsNullOrEmpty(TxtAttack.Text)
                || int.TryParse(TxtAttack.Text, out int attack))
            {
                MessageBox.Show("You must enter a numeric value for attack,");
                return;
            }

            if (string.IsNullOrEmpty(TxtDefense.Text)
                || int.TryParse(TxtDefense.Text, out int defense))
            {
                MessageBox.Show("You must enter a numeric value for defense.");
                return;
            }

            if (string.IsNullOrEmpty(TxtSpeed.Text)
                || int.TryParse(TxtSpeed.Text, out int speed))
            {
                MessageBox.Show("You must enter a numeric value for speed,");
                return;
            }

            if (string.IsNullOrEmpty(TxtHealth.Text)
                || int.TryParse(TxtHealth.Text, out int health))
            {
                MessageBox.Show("You must enter a numeric value for health,");
                return;
            }

            StringBuilder sb = new StringBuilder();

            sb.Append(TxtKey.Text + "," + TxtName.Text + "," + cost + ",");
            sb.Append(attack + "," + defense + "," + speed + "," + health + ",0,0");

            foreach (object o in LBMoves.Items)
            {
                string s = o.ToString();
                sb.Append("," + s);
            }

            ShadowMonster = sb.ToString();
            OkPressed = true;
            Close();
        }

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

        private void BtnAdd_Click(object sender, EventArgs e)
        {
            LBMoves.Items.Add(CboMoves.SelectedItem.ToString() + ":" + TxtMoveLevel.Text);

        }

        private void BtnRemove_Click(object sender, EventArgs e)
        {
            if (LBMoves.SelectedIndex < 0)
                return;

            LBMoves.Items.RemoveAt(LBMoves.SelectedIndex);
        }

        private void DefinitionForm_Load(object sender, EventArgs e)
        {
            MoveManager.FillMoves();

            foreach (IMove move in MoveManager.Moves.Values)
            {
                CboMoves.Items.Add(move.Name);
            }
        }
    }
}
```

There are public properties ShadowMonster that returns a string the can be used in FromString of the ShadowMonster class to create a shadow monster. There is a bool property OkPressed that communicates how the form was closed.

There are two constructors in the class. In the first constructor I wire the Load event handler and populate a combo box that holds the elements for a shadow monster using the GetNames method of the Enum class. I then wire the event handlers for the click events of the buttons on the form.

The second constructor takes a ShadowMonster parameter. It calls the first constructor to wire the event handlers and fill the combo box. Then it sets the form controls to the values passed in.

The event handler for the OK button I validate that the form fields have values and that the numeric fields are numeric. If the validation succeeds I create a StringBuilder. I call Append twice passing in the value of the controls. For source tile I pass in (0, 0) because we are using separate images and not a single image. I then loop over the moves and add them to the string. I set the ShadowMonster property to the ToString of the string builder. OkPressed is set to true then the form is closed.

The event handler for the Click event of the Cancel button just closes the form. The event handler for the Click event of the Add button adds the concatenation of the selected move, a colon and the move level text box. You should probably do some validation here.

In the Click event handler for the Remove button I check to make sure there is no selected item. If there is not I leave the method. Otherwise I remove the selected index from the items collection.

In the event handler for the Load event of the form I call MoveManager.FillMoves to be sure there are moves to work with. Then I loop over the Values property of the collection and add the Name property of the move to the Items collection of CboMoves.

Now that we have the details form we need a list, or master, to hold the shadow monsters. Right click the ShadowEditor project in the Solution Explorer, select Add and then Form (Windows Forms). Name this new for DefinitionListForm. Expand the node beside DefinitionListForm then open the DefinitionListForm.Designer.cs file. Replace the contents with the following code.

```
namespace ShadowEditor
{
    partial class DefinitionListForm
    {
        /// <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.LBDefinitions = 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.BtnImport = new System.Windows.Forms.Button();
            this.BtnExport = new System.Windows.Forms.Button();
            this.SuspendLayout();
            //
            // LBDefinitions
            //
            this.LBDefinitions.FormattingEnabled = true;
            this.LBDefinitions.Location = new System.Drawing.Point(12, 12);
            this.LBDefinitions.Name = "LBDefinitions";
            this.LBDefinitions.Size = new System.Drawing.Size(287, 277);
            this.LBDefinitions.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;
            //
            // BtnImport
            //
            this.BtnImport.Location = new System.Drawing.Point(305, 100);
            this.BtnImport.Name = "BtnImport";
            this.BtnImport.Size = new System.Drawing.Size(75, 23);
            this.BtnImport.TabIndex = 4;
            this.BtnImport.Text = "Import";
            this.BtnImport.UseVisualStyleBackColor = true;
            //
            // BtnExport
            //
            this.BtnExport.Location = new System.Drawing.Point(307, 129);
            this.BtnExport.Name = "BtnExport";
            this.BtnExport.Size = new System.Drawing.Size(75, 23);
            this.BtnExport.TabIndex = 5;
            this.BtnExport.Text = "Export";
            this.BtnExport.UseVisualStyleBackColor = true;
            //
            // DefinitionListForm
            //
            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.BtnExport);
            this.Controls.Add(this.BtnImport);
            this.Controls.Add(this.BtnDelete);
            this.Controls.Add(this.BtnEdit);
            this.Controls.Add(this.BtnAdd);
            this.Controls.Add(this.LBDefinitions);
            this.Name = "DefinitionListForm";
            this.Text = "Shadow Monster Definitions";
            this.ResumeLayout(false);

        }
```

```
        #endregion

        private System.Windows.Forms.ListBox LBDefinitions;
        private System.Windows.Forms.Button BtnAdd;
        private System.Windows.Forms.Button BtnEdit;
        private System.Windows.Forms.Button BtnDelete;
        private System.Windows.Forms.Button BtnImport;
        private System.Windows.Forms.Button BtnExport;
    }
}
```

Right click the DefinitionListForm in the Solution Explorer and select View Code or select it in the Solution Explorer and press F7. Replace the contents of the file with the following code.

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

namespace ShadowEditor
{
    public partial class DefinitionListForm : Form
    {
        private readonly byte[] IV = new byte[]
        {
            067, 197, 032, 010, 211, 090, 192, 076,
            054, 154, 111, 023, 243, 071, 132, 090
        };

        private readonly byte[] Key = new byte[]
        {
            067, 090, 197, 043, 049, 029, 178, 211,
            127, 255, 097, 233, 162, 067, 111, 022,
        };

        public DefinitionListForm()
        {
            InitializeComponent();

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

            BtnImport.Click += BtnImport_Click;
            BtnExport.Click += BtnExport_Click;

            foreach (var v in ShadowMonsterManager.ShadowMonsterList.Keys)
            {
                LBDefinitions.Items.Add(v);
            }
        }

        private void BtnImport_Click(object sender, EventArgs e)
        {
            OpenFileDialog ofd = new OpenFileDialog
            {
                Filter = "Definitions (*.txt)|*.txt"
            };

            DialogResult result = ofd.ShowDialog();

            if (result == DialogResult.OK)
            {
                //LoadShadowMonsters(ofd.FileName);
                using (Stream stream = new FileStream(ofd.FileName, FileMode.Open,
FileAccess.Read))
                {
                    try
                    {
                        using (TextReader reader = new StreamReader(stream))
                        {
                            try
                            {
                                string lineIn = "";

                                do
                                {
                                    lineIn = reader.ReadLine();
                                    if (lineIn != null)
                                    {
                                        ShadowMonster monster =
ShadowMonster.FromString(lineIn);
                                        if (!
ShadowMonsterManager.ShadowMonsterList.ContainsKey(monster.Name.ToLowerInvariant()))
                                        {

ShadowMonsterManager.ShadowMonsterList.Add(monster.Name.ToLowerInvariant(), monster);
                                            LBDefinitions.Items.Add(monster.Name);
                                        }
                                    }
                                } while (lineIn != null);
                            }
                            catch (Exception exc)
                            {
                                exc.GetType();
                            }
                            finally
                            {
                                if (reader != null)
                                    reader.Close();
                            }
                        }
                    }
                    catch (Exception exc)
                    {
                        exc.GetType();
                    }
                    finally
                    {
```

```
                    if (stream != null)
                        stream.Close();
                }
            }
        }
    }

    private void LoadShadowMonsters(string filename)
    {
        using (Aes aes = Aes.Create())
        {
            aes.IV = IV;
            aes.Key = Key;

            try
            {
                ICryptoTransform decryptor = aes.CreateDecryptor(Key, IV);
                FileStream stream = new FileStream(
                    filename,
                    FileMode.Open,
                    FileAccess.Read);
                using (CryptoStream cryptoStream = new CryptoStream(
                    stream,
                    decryptor,
                    CryptoStreamMode.Read))
                {
                    using (BinaryReader reader = new BinaryReader(cryptoStream))
                    {
                        int count = reader.ReadInt32();

                        for (int i = 0; i < count; i++)
                        {
                            string data = reader.ReadString();
                            reader.ReadInt32();
                            ShadowMonster monster = ShadowMonster.FromString(data);
                            ShadowMonsterManager.AddShadowMonster(monster.Name, monster);
                            LBDefinitions.Items.Add(monster.Name);
                        }
                    }
                }
            }
            catch (Exception exc)
            {
                MessageBox.Show(exc.ToString());
            }
        }
    }

    private void BtnExport_Click(object sender, EventArgs e)
    {
        SaveFileDialog sfd = new SaveFileDialog
        {
            Filter = "Definitions (*.txt)|*.txt"
        };

        DialogResult result = sfd.ShowDialog();

        if (result != DialogResult.OK)
            return;
            using (Aes aes = Aes.Create())
            {
                aes.IV = IV;
                aes.Key = Key;

                try
                {
                    ICryptoTransform encryptor = aes.CreateEncryptor(Key, IV);
                    FileStream stream = new FileStream(
                        sfd.FileName,
                        FileMode.Create,
                        FileAccess.Write);
                    using (CryptoStream cryptoStream = new CryptoStream(
                        stream,
                        encryptor,
                        CryptoStreamMode.Write))
                    {
                        BinaryWriter writer = new BinaryWriter(cryptoStream);

                        writer.Write(ShadowMonsterManager.ShadowMonsterList.Count);

                        foreach (ShadowMonster shadowMonster in
ShadowMonsterManager.ShadowMonsterList.Values)
                        {
                            writer.Write(shadowMonster.ToString());
                            writer.Write(-1);
                        }

                        writer.Close();
                    }
                    stream.Close();
                    stream.Dispose();
                }
                catch (Exception exc)
                {
                    MessageBox.Show(exc.Message);
                }
            }

        }

        private void BtnEdit_Click(object sender, EventArgs e)
        {
            if (LBDefinitions.SelectedIndex < 0)
                return;

            DefinitionForm frm = new DefinitionForm(
                ShadowMonsterManager.ShadowMonsterList[LBDefinitions.SelectedItem.ToString()]);
            frm.ShowDialog();

            if (!frm.OkPressed)
                return;

            ShadowMonsterManager.ShadowMonsterList[LBDefinitions.SelectedItem.ToString()] =
                ShadowMonster.FromString(frm.ShadowMonster);
        }

        private void BtnDelete_Click(object sender, EventArgs e)
        {
            if (LBDefinitions.SelectedIndex < 0)
```

```
                return;

            ShadowMonsterManager.ShadowMonsterList.Remove(LBDefinitions.SelectedItem.ToString());
            LBDefinitions.Items.Remove(LBDefinitions.SelectedItem);
        }

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

            if (frm.OkPressed)
            {
                ShadowMonster m = ShadowMonster.FromString(frm.ShadowMonster);
                ShadowMonsterManager.AddShadowMonster(m.Name.ToLowerInvariant(), m);
                LBDefinitions.Items.Add(m.Name);
            }
        }
    }
}
```

There are copies of the IV and Key arrays have been copied from the GamePlayState for encrypting and decrypting the file. In the constructor I wire the event handlers for the Add, Delete, Edit, Import and Export buttons. I then loop over the keys in the ShadowMonsterList dictionary and adds them to the list box that holds definitions.

In the Click event handler for BtnImport I create a OpenFileDialog with a *.txt filter so we can browse for the definition file. I capture the return value of showing the dialog. If it OK I go to the code for loading shadow monsters. There is a commented out a call to the LoadShadowMonsters method for now because the file is currently not encrypted. Once the file is encrypted I will remove the current loading code and call the method instead. I copied the code for loading shadow monsters from the ShadowMonsterManager.FromFile method. I didn't call it because I did a little extra processing. First it creates a FileStream that points to the file name from the OpenFileDialog. It then creates a TextReader to read the file inside a try-catch block. I then read the file line by line. If a line is not null I call the FromString method of the ShadowMonster class passing in the line. This is a call to an overload of the method that I added. If a shadow monster by that name does not exist in the dictionary I call the AddShadowMonster method. I then add the shadow monster to the definitions list box. There are catches to catch any exceptions. For now they do nothing useful but in the future we will do something meaningful with the exceptions. In the finally clauses I close the reader and the stream.

The LoadShadowMonsters method accepts the filename for the shadow monster definition file. It creates an Aes instance. It sets the IV and Key properties of the Aes instance. It creates a decryptor passing in the Key and IV. It opens a file stream for reading using the file name. It then creates a CryptoStream with read access passing in the decryptor. Because I'm encrypting the files now I use a BinaryReader instead of a TextReader. It reads the count of shadow monsters in the file. It loops that many times. It reads the definition as a string then reads a delimiter. I create the ShadowMonster using string just read in, without passing in a ContentManager. I call AddShadow monster on the ShadowMonsterManager passing in the Name property of the shadow monster and monster. I then add the shadow monster to the list of shadow monsters. If there is an exception I display it in a message box.

The Click event handler for BtnExport creates a SaveFileDialog to allow us to browse to the path to save with. I then capture the result in a local variable. If that variable is not the OK result I exit out of the method. I create an Aes and initialize the IV and key as before. In a try-catch block I create the encryptor using the CreateEncryptor method of the Aes class passing in the IV and key. I open a file stream for writing and create new. I create a CryptoStream to encrypt the file stream as it is being written. I create a BinaryWriter to write the file. I write the number of shadow monsters in the dictionary. I then loop over the Values property of the ShadowMonsterList. I write the shadowMonster using the ToString method that I will add shortly. I then write a delimiter. I close the writer and close and dispose the stream. If an exception occurred it is displayed in a message box.

In the event handler for the Click event of BtnEdit I check to see if there is no selected item. If there is not I exit out of the method. I create a DefinitionForm passing in the shadow monster that is the selected item.I call the ShowDialog method. If the OkPressed property of the form is false I exit the method. I then set the shadow monster with the selected key to be the shadow monster from the form using the FromString method.

The Click event handler for BtnDelete checks to see if there is no selected item and exits the method if there is not. It calls the Remove method on the ShadowMonstList property of the ShadowMonsterManager to remove the shadow monster and the Remove method of the Items collection of the list box.

The Click event handler for BtnAdd I create a DefinitionForm and then call the ShowDialog method on it. If the OkPressed property is true I create a shadow monster using the FromString method that takes a single parameter. I call AddShadowMonster on the ShadowMonsterManager to add it to the list of shadow monsters. I add the Name property of the monster to the list box.

I added an overload of the FromString method to the ShadowMonster class. I also added an override of the ToString method. I also added a Cost property. Add the following code to the ShadowMonster class.

```
        public int Cost
        {
            get { return costToBuy; }
        }

        public override string ToString()
        {
            StringBuilder sb = new StringBuilder();

            sb.Append(Name + "," + DisplayName + "," + Element.ToString() + "," + costToBuy +
",");
            sb.Append(Level + "," + attack + "," + defense + "," + speed + "," + health +
",0,0");

            foreach(IMove move in KnownMoves.Values)
            {
                sb.Append("," + move.Name + ":" + move.UnlockedAt);
            }
            return sb.ToString();
        }
```

The ToString method appends the fields together using a string builder. It then loops over the values in the knownMoves dictionary. For each move I append the name and level it unlocks at separated by a colon. The FromString method is a copy of the other FromString method minus loading the texture for the shadow monster. I did this because we don't know where the content folder is for the second project.

Now I need to update the Editor class in the ShadowEditor project. I added code to display the DefinitionListForm if the Q key has been released. I chose Q because S is used for adding shadow monsters to the map and M is used for adding merchants. Change the Update 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;

        lbWorld = new ListBox(
            background,
            cursor)
        {
            Position = new Vector2(64 * 18 + 325, 800),
            TabStop = true,
            Enabled = true,
            HasFocus = true
        };

        lbWorld.SelectionChanged += LbWorld_SelectionChanged;

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

    private void LbWorld_SelectionChanged(object sender, EventArgs e)
    {
        world.ChangeMap(lbWorld.SelectedItem);
        pbTileset.Image = world.Map.TileSet.Textures[0];
        pbPreview.Image = world.Map.TileSet.Textures[0];

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

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

        camera.Position = Vector2.Zero;
        camera.LockCamera(world.Map, viewPort);
    }

    private void LbTileSets_SelectionChanged(object sender, System.EventArgs e)
    {
        pbTileset.Image = world.Map.TileSet.Textures[lbTileSets.SelectedIndex];
        pbPreview.Image = world.Map.TileSet.Textures[lbTileSets.SelectedIndex];
    }

    /// <summary>
    /// UnloadContent will be called once per game and is the place to unload
    /// game-specific content.
    /// </summary>
    protected override void UnloadContent()
    {
        // TODO: Unload any non ContentManager content here
        }

        /// <summary>
        /// Allows the game to run logic such as updating the world,
        /// checking for collisions, gathering input, and playing audio.
        /// </summary>
        /// <param name="gameTime">Provides a snapshot of timing values.</param>
        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)
                {
                    world.Maps.Add(form.TileMap.MapName, form.TileMap);
                    world.ChangeMap(form.TileMap.MapName);

                    lbWorld.Items.Add(world.Map.MapName);

                    camera.Position = Vector2.Zero;
                    camera.LockCamera(world.Map, viewPort);

                    pbTileset.Image = world.Map.TileSet.Textures[0];
                    pbPreview.Image = world.Map.TileSet.Textures[0];

                    lbTileSets.Items.Clear();
                    foreach (string s in world.Map.TileSet.TextureNames)
```

```
                        lbTileSets.Items.Add(s);

                    lbTileSets.SelectedIndex = 0;
                    pbPreview.SourceRectangle =
                        world.Map.TileSet.SourceRectangles[0];
                    pbTileset.SourceRectangle = new Rectangle(
                        0,
                        0,
                        world.Map.TileSet.Textures[0].Width,
                        world.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 (world != null && world.Maps.Count > 0 && 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++)
                                    {
                                        world.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++)
                                    {
                                        world.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++)
                                    {
                                        world.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++)
                                    {
                                        world.Map.SetBuildingTile(tile.X + i, tile.Y + j,
lbTileSets.SelectedIndex, selectedTile);
                                    }
                                }
                                break;
                        }
                    }
                    else
                    {
                        if (!world.Map.CollisionLayer.Collisions.ContainsKey(
                            new Point(
                                tile.X,
                                tile.Y)))
                        {
                            world.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++)
                                    {
                                        world.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++)
                                    {
                                        world.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++)
                                    {
                                        world.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++)
                                    {
                                        world.Map.SetBuildingTile(tile.X + i, tile.Y + j, -1,
-1);
                                    }
                                }
                                break;
                        }
                    }
                    else
                    {
                        if (world.Map.CollisionLayer.Collisions.ContainsKey(
                            new Point(
                                tile.X,
                                tile.Y)))
                        {
                            world.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)world.Map.TileSet.Textures[lbTileSets.SelectedIndex].Width /
                    pbTileset.DestinationRectangle.Width;

                float yScale =
(float)world.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 / world.Map.TileSet.TileWidth,
                    tilesetPoint.Y / world.Map.TileSet.TileHeight);
                selectedTile = clickedTile.Y * world.Map.TileSet.TilesWide + clickedTile.X;
                pbPreview.SourceRectangle =
                    world.Map.TileSet.SourceRectangles[selectedTile];
            }

            if (Xin.CheckKeyReleased(Keys.C) && frameCount > 5 && world.Maps.Count > 0)
            {
                CharacterListForm frm = new CharacterListForm(world.Map, this);
                frm.ShowDialog();
            }

            if (Xin.CheckKeyReleased(Keys.M) && frameCount > 5 && world.Maps.Count > 0)
            {
                MerchantListForm frm = new MerchantListForm(world.Map, this);
                frm.ShowDialog();
            }

            if (Xin.CheckKeyReleased(Keys.D) && frameCount > 5 && world.Maps.Count > 0)
            {
                PortalListForm frm = new PortalListForm(world.Map, this);
                frm.ShowDialog();
            }

            if (Xin.CheckKeyReleased(Keys.S) && frameCount > 5 && world.Maps.Count > 0)
            {
                ShadowMonsterListForm frm = new ShadowMonsterListForm(world.Map, this);
                frm.ShowDialog();
            }

            if (Xin.CheckKeyReleased(Keys.Q) && frameCount > 5)
            {
                DefinitionListForm frm = new DefinitionListForm();
                frm.ShowDialog();
            }
            if (Xin.CheckKeyReleased(Keys.W) && frameCount > 5)
            {
                if (world != null)
                {
                    WF.DialogResult result = WF.MessageBox.Show("Proceeding will delete
existing world. Are you sure you want to continue?", "Continue?", WF.MessageBoxButtons.YesNo);
                    if (result == WF.DialogResult.No)
                    {
                        return;
                    }
                }

                WorldForm form = new WorldForm();
                form.ShowDialog();

                if (!form.OKPressed)
                {
                    return;
                }

                world = form.World;
            }
            if (Xin.CheckKeyReleased(Keys.F1) && frameCount > 5)
            {
                WF.SaveFileDialog sfd = new WF.SaveFileDialog();
                sfd.Filter = "World (*.wrld)|*.wrld";
                WF.DialogResult result = sfd.ShowDialog();

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

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

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

                    pbTileset.Image = world.Map.TileSet.Textures[0];
                    pbPreview.Image = world.Map.TileSet.Textures[0];

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

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

                    camera.Position = Vector2.Zero;
                    camera.LockCamera(world.Map, viewPort);
                }
            }

            HandleScrollMap();
            controls.Update(gameTime);

            if (world != null  && world.Maps.Count > 0)
            {
                world.Update(gameTime);
            }

            base.Update(gameTime);
        }
```

Now build and run the editor. When it loads up press the Q key to load the definition. The list box should be populated with the six shadow monsters. Click the Export button and navigate to the ShadowMonsters\ContentFolder. Select the ShadowMonsters.txt file and over write it with the encrypted copy.

I need to update the DetailListForm and ShadowMonsterManager to now use the code for encrypted files when loading. For the Click event handler of the Import button I uncommented the call to LoadShadowMonsters and deleted the loading code. Update the BtnImport_Click method to the follow.

```
        private void BtnImport_Click(object sender, EventArgs e)
        {
            OpenFileDialog ofd = new OpenFileDialog
            {
                Filter = "Definitions (*.txt)|*.txt"
            };

            DialogResult result = ofd.ShowDialog();

            if (result == DialogResult.OK)
            {
                LoadShadowMonsters(ofd.FileName);
            }
        }
```

I replaced the existing code from the ShadowMonsterManager with the code from the LoadShadowMonsters method, making sure to pass the content manager as a parameter. Add the following using statement, fields and update the FromFile method of the ShadowMonsterManager to the following.

```
using System.Security.Cryptography;

        private static readonly byte[] IV = new byte[]
        {
            067, 197, 032, 010, 211, 090, 192, 076,
            054, 154, 111, 023, 243, 071, 132, 090
        };

        private static readonly byte[] Key = new byte[]
        {
            067, 090, 197, 043, 049, 029, 178, 211,
            127, 255, 097, 233, 162, 067, 111, 022,
        };

        public static void FromFile(string fileName, ContentManager content)
        {
            using (Aes aes = Aes.Create())
            {
                aes.IV = IV;
                aes.Key = Key;

                try
                {
                    ICryptoTransform decryptor = aes.CreateDecryptor(Key, IV);
                    FileStream stream = new FileStream(
                        fileName,
                        FileMode.Open,
                        FileAccess.Read);
                    using (CryptoStream cryptoStream = new CryptoStream(
                        stream,
                        decryptor,
                        CryptoStreamMode.Read))
                    {
                        using (BinaryReader reader = new BinaryReader(cryptoStream))
                        {
                            int count = reader.ReadInt32();

                            for (int i = 0; i < count; i++)
                            {
                                string data = reader.ReadString();
                                reader.ReadInt32();
                                ShadowMonster monster = ShadowMonster.FromString(data,
content);

                                ShadowMonsterManager.AddShadowMonster(monster.Name, monster);
                            }
                        }
                    }
                }
                catch
                {
                }
            }
        }
```

Now you can easily add in new shadow monsters to the game and the file is encrypted so players can't go peeking and making changes to your code. I'm going to add some new moves in this tutorial before wrapping up. They will automatically be picked up by the editor so you don't have to worry about that.

Right click the ShadowMonsters folder in the ShadowMonsters project, select Add and then Class. Name this new class Bubble. Here is the code.

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

namespace ShadowMonsters.ShadowMonsters
{
    public class Bubble : IMove
    {
        #region Field Region

        private int unlockedAt = 3;
        private bool unlocked = false;
        private int duration = 0;

        #endregion

        #region Property Region

        public string Name => "Bubble";

        public Target Target => Target.Enemy;

        public MoveType MoveType => MoveType.Attack;

        public MoveElement MoveElement => MoveElement.Water;

        public int UnlockedAt { get => unlockedAt; set => unlockedAt = value; }

        public bool Unlocked => unlocked;
        public int Duration { get => duration; set => duration = value; }

        public int Attack => 0;

        public int Defense => 0;

        public int Speed => 0;

        public int Health => MoveManager.Random.Next(20, 30);

        #endregion

        #region Method Region

        public object Clone()
        {
            Bubble move = new Bubble
            {
                duration = this.duration,
                unlocked = this.unlocked,
                unlockedAt = this.unlockedAt
            };

            return move;
        }

        public void Unlock()
        {
            unlocked = true;
        }

        #endregion
    }
}
```

The class implements the IMove interface. For brevity's sake I only added three fields. One for duration, which is set to 0, one for unlocked that is set to false and one for unlockedAt which is set at 3. There is a get only property that returns the name of the move. Target returns Enemy, MoveType is Attack, Element is Water. Unlocked at is a get/set which gets the unlocked field and sets it to the value passed in. Unlocked is a get only for the unlocked field. Duration is a get/set for the duration field. Attack, Defense and Speed all return 0. Health returns a number between 20 and 29. There is no constructor because everything was initialized inline. The Clone method creates a new instance of Bubble assigning the fields of the created instance to the current instance. The Unlock method just sets unlocked to true.

The rest of the classes follow the same flow. The difference is in their values. The different one is Heal that sets the move type to Heal and the target to Self. I will add them one by one but I'm not going to explain them like above.

Right click the ShadowMonsters folder under the ShadowMonsters project, select Add and then class. Name this new class Flare. Here is the code.

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

namespace ShadowMonsters.ShadowMonsters
{
    public class Flare : IMove
    {
        #region Field Region

        private int unlockedAt = 3;
        private bool unlocked = false;
        private int duration = 0;

        #endregion

        #region Property Region

        public string Name => "Flare";

        public Target Target => Target.Enemy;

        public MoveType MoveType => MoveType.Attack;

        public MoveElement MoveElement => MoveElement.Fire;

        public int UnlockedAt { get => unlockedAt; set => unlockedAt = value; }

        public bool Unlocked => unlocked;

        public int Duration { get => duration; set => duration = value; }

        public int Attack => 0;

        public int Defense => 0;

        public int Speed => 0;

        public int Health => MoveManager.Random.Next(20, 30);

        #endregion

        #region Method Region

        public object Clone()
        {
            Flare move = new Flare
            {
                duration = this.duration,
                unlocked = this.unlocked,
                unlockedAt = this.unlockedAt
            };

            return move;
        }

        public void Unlock()
        {
            unlocked = true;
        }
        #endregion
    }
}
```

Right click the ShadowMonsters folder in the ShadowMonsters project, select Add and then Class. Name this new class Gust. Here is the code.

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

namespace ShadowMonsters.ShadowMonsters
{
    public class Gust : IMove
    {
        #region Field Region

        private int unlockedAt = 3;
        private bool unlocked = false;
        private int duration = 0;

        #endregion

        #region Property Region

        public string Name => "Gust";

        public Target Target => Target.Enemy;

        public MoveType MoveType => MoveType.Attack;

        public MoveElement MoveElement => MoveElement.Wind;

        public int UnlockedAt { get => unlockedAt; set => unlockedAt = value; }

        public bool Unlocked => unlocked;

        public int Duration { get => duration; set => duration = value; }

        public int Attack => 0;

        public int Defense => 0;

        public int Speed => 0;

        public int Health => MoveManager.Random.Next(20, 30);

        #endregion

        #region Method Region

        public object Clone()
        {
            Gust move = new Gust
            {
                duration = this.duration,
                unlocked = this.unlocked,
                unlockedAt = this.unlockedAt
            };

            return move;
        }

        public void Unlock()
        {
            unlocked = true;
        }

        #endregion
    }
}
```

Right click the ShadowMonsters project under the ShadowMonsters project in the Solution Explorer, select Add and then class. Name this new class Heal. Here is the code.

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

namespace ShadowMonsters.ShadowMonsters
{
    public class Heal : IMove
    {
        #region Field Region

        private int unlockedAt = 3;
        private bool unlocked = false;
        private int duration = 0;

        #endregion

        #region Property Region

        public string Name => "Heal";

        public Target Target => Target.Enemy;

        public MoveType MoveType => MoveType.Heal;

        public MoveElement MoveElement => MoveElement.Light;

        public int UnlockedAt { get => unlockedAt; set => unlockedAt = value; }

        public bool Unlocked => unlocked;

        public int Duration { get => duration; set => duration = value; }

        public int Attack => 0;

        public int Defense => 0;

        public int Speed => 0;

        public int Health => MoveManager.Random.Next(25, 50);
        #endregion

        #region Method Region

        public object Clone()
        {
            Heal move = new Heal
            {
                duration = this.duration,
                unlocked = this.unlocked,
                unlockedAt = this.unlockedAt
            };

            return move;
        }

        public void Unlock()
        {
            unlocked = true;
        }

        #endregion
    }
}
```

Right click the ShadowMonsters folder under the ShadowMonsters project in the Solution Explorer, select Add and then Class. Name this new class RockToss. Here is the code for the class.

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

namespace ShadowMonsters.ShadowMonsters
{
    public class RockToss : IMove
    {
        #region Field Region

        private int unlockedAt = 3;
        private bool unlocked = false;
        private int duration = 0;

        #endregion

        #region Property Region

        public string Name => "Rock Toss";

        public Target Target => Target.Enemy;

        public MoveType MoveType => MoveType.Attack;

        public MoveElement MoveElement => MoveElement.Earth;

        public int UnlockedAt { get => unlockedAt; set => unlockedAt = value; }
        public bool Unlocked => unlocked;

        public int Duration { get => duration; set => duration = value; }

        public int Attack => 0;

        public int Defense => 0;

        public int Speed => 0;

        public int Health => MoveManager.Random.Next(20, 30);

        #endregion

        #region Method Region

        public object Clone()
        {
            RockToss move = new RockToss
            {
                duration = this.duration,
                unlocked = this.unlocked,
                unlockedAt = this.unlockedAt
            };

            return move;
        }

        public void Unlock()
        {
            unlocked = true;
        }

        #endregion
    }
}
```

Right click the ShadowMonsters folder under the ShadowMonsters project in the Solution Explorer, select Add and then Class. Name this new class Shade. Here is the code.

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

namespace ShadowMonsters.ShadowMonsters
{
    public class Shade : IMove
    {
        #region Field Region

        private int unlockedAt = 3;
        private bool unlocked = false;
        private int duration = 0;

        #endregion
        #region Property Region

        public string Name => "Shade";

        public Target Target => Target.Enemy;

        public MoveType MoveType => MoveType.Attack;

        public MoveElement MoveElement => MoveElement.Dark;

        public int UnlockedAt { get => unlockedAt; set => unlockedAt = value; }

        public bool Unlocked => unlocked;

        public int Duration { get => duration; set => duration = value; }

        public int Attack => 0;

        public int Defense => 0;

        public int Speed => 0;

        public int Health => MoveManager.Random.Next(20, 30);

        #endregion

        #region Method Region

        public object Clone()
        {
            Shade move = new Shade
            {
                duration = this.duration,
                unlocked = this.unlocked,
                unlockedAt = this.unlockedAt
            };

            return move;
        }

        public void Unlock()
        {
            unlocked = true;
        }

        #endregion
    }
}
```

The last thing to do is to update the MoveManager. In the FillMoves method you need to add in the new classes. Replace the FillMoves method with the following.

```
        public static void FillMoves()
        {
            allMoves.Clear();
            allMoves.Add("Tackle", new Tackle());
            allMoves.Add("Block", new Block());
            allMoves.Add("Bubble", new Bubble());
            allMoves.Add("Flare", new Flare());
            allMoves.Add("Gust", new Gust());
            allMoves.Add("Heal", new Heal());
            allMoves.Add("Rock Toss", new RockToss());
            allMoves.Add("Shade", new Shade());
        }
```

I am going to wrap this tutorial up here as we've covered a lot and I don't want to start a new topic at this point. 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!