Shadow Monsters – MonoGame Tutorial Series
Chapter 13
Editor – Part One

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 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.

The main focus of this tutorial is getting started on the editor for the game. Before I get into the editor I want to refactor the GamePlayState. The Update method is getting very messy and updating it is a lot of code. So, I want to extract some of the code and place it into methods. Replace the GamePlayState with the following code.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using ShadowMonsters.Characters;
using ShadowMonsters.ShadowMonsters;
using ShadowMonsters.TileEngine;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Security.Cryptography;
using System.Text;
using System.Threading.Tasks;

namespace ShadowMonsters.GameStates
{
    public class GamePlayState : BaseGameState
    {
        private readonly Engine engine = new Engine(new Rectangle(0, 0, 1280, 720));
        private TileMap map;
        private Vector2 motion;
        private bool inMotion;
        private Rectangle collision;
        private ShadowMonsterManager monsterManager = new ShadowMonsterManager();
        private int frameCount = 0;
```

```
        public GamePlayState(Game game) : base(game)
        {
        }

        protected override void LoadContent()
        {
            MoveManager.FillMoves();
            ShadowMonsterManager.FromFile(@".\Content\ShadowMonsters.txt", content);
            Game1.Player.AddShadowMonster(ShadowMonsterManager.GetShadowMonster("water1"));
            Game1.Player.SetCurrentShadowMonster(0);
            Game1.Player.BattleShadowMonsters[0] = Game1.Player.GetShadowMonster(0);
            TileSet set = new TileSet();
            set.TextureNames.Add("tileset16-outdoors");
            set.Textures.Add(content.Load<Texture2D>(@"Tiles\tileset16-outdoors"));

            TileLayer groundLayer = new TileLayer(100, 100, 0, 1);
            TileLayer edgeLayer = new TileLayer(100, 100);
            TileLayer buildingLayer = new TileLayer(100, 100);
            TileLayer decorationLayer = new TileLayer(100, 100);

            for (int i = 0; i < 1000; i++)
            {
                decorationLayer.SetTile(random.Next(0, 100), random.Next(0, 100), 0,
random.Next(2, 4));
            }

            map = new TileMap(set, groundLayer, edgeLayer, buildingLayer, decorationLayer,
"level1");

            Character c = Character.FromString(GameRef,
"Paul,ninja_m,WalkDown,PaulHello,0,fire1,fire1,,,,,,dark1");
            c.Sprite.Position = new Vector2(2 * Engine.TileWidth, 2 * Engine.TileHeight);

            map.CharacterLayer.Characters.Add(new Point(2, 2), c);

            Merchant m = Merchant.FromString(GameRef,
"Bonnie,ninja_f,WalkLeft,BonnieHello,0,earth1,earth1,,,,,,");
            m.Sprite.Position = new Vector2(4 * Engine.TileWidth, 4 * Engine.TileHeight);
            m.Backpack.AddItem("Potion", 99);
            m.Backpack.AddItem("Antidote", 10);

            map.CharacterLayer.Characters.Add(new Point(4, 4), m);
            engine.SetMap(map);
            base.LoadContent();
        }

        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 override void Update(GameTime gameTime)
        {
```

```
            engine.Update(gameTime);
            frameCount++;
            HandleMovement(gameTime);

            if ((Xin.CheckKeyReleased(Keys.Space) ||
                Xin.CheckKeyReleased(Keys.Enter)) && frameCount >= 5)
            {
                frameCount = 0;
                StartInteraction();
            }

            if (Xin.CheckKeyReleased(Keys.I))
            {
                manager.PushState(GameRef.ItemSelectionState);
            }

            if ((Xin.CheckKeyReleased(Keys.B)) && frameCount >= 5)
            {
                frameCount = 0;
                StartBattle();
            }

            if (Xin.CheckKeyReleased(Keys.F1))
            {
                SaveGame();
            }
            if (Xin.CheckKeyReleased(Keys.F2))
            {
                LoadGame();
            }

            Engine.Camera.LockToSprite(map, Game1.Player.Sprite, new Rectangle(0, 0, 1280,
720));
            Game1.Player.Update(gameTime);

            base.Update(gameTime);
        }

        private void HandleMovement(GameTime gameTime)
        {
            if (Xin.KeyboardState.IsKeyDown(Keys.W) && !inMotion)
            {
                motion.Y = -1;
                Game1.Player.Sprite.CurrentAnimation = AnimationKey.WalkUp;
                Game1.Player.Sprite.IsAnimating = true;
                inMotion = true;
                collision = new Rectangle(
                    (int)Game1.Player.Sprite.Position.X,
                    (int)Game1.Player.Sprite.Position.Y - Engine.TileHeight * 2,
                    Engine.TileWidth,
                    Engine.TileHeight);
            }
            else if (Xin.KeyboardState.IsKeyDown(Keys.S) && !inMotion)
            {
                motion.Y = 1;
                Game1.Player.Sprite.CurrentAnimation = AnimationKey.WalkDown;
                Game1.Player.Sprite.IsAnimating = true;
                inMotion = true;
                collision = new Rectangle(
                    (int)Game1.Player.Sprite.Position.X,
```

```
                (int)Game1.Player.Sprite.Position.Y + Engine.TileHeight * 2,
                Engine.TileWidth,
                Engine.TileHeight);
        }
        else if (Xin.KeyboardState.IsKeyDown(Keys.A) && !inMotion)
        {
            motion.X = -1;
            Game1.Player.Sprite.CurrentAnimation = AnimationKey.WalkLeft;
            Game1.Player.Sprite.IsAnimating = true;
            inMotion = true;
            collision = new Rectangle(
                (int)Game1.Player.Sprite.Position.X - Engine.TileWidth * 2,
                (int)Game1.Player.Sprite.Position.Y,
                Engine.TileWidth,
                Engine.TileHeight);
        }
        else if (Xin.KeyboardState.IsKeyDown(Keys.D) && !inMotion)
        {
            motion.X = 1;
            Game1.Player.Sprite.CurrentAnimation = AnimationKey.WalkRight;
            Game1.Player.Sprite.IsAnimating = true;
            inMotion = true;
            collision = new Rectangle(
                (int)Game1.Player.Sprite.Position.X + Engine.TileWidth * 2,
                (int)Game1.Player.Sprite.Position.Y,
                Engine.TileWidth,
                Engine.TileHeight);
        }

        if (motion != Vector2.Zero)
        {
            motion.Normalize();
            motion *= (Game1.Player.Sprite.Speed *
(float)gameTime.ElapsedGameTime.TotalSeconds);
            Rectangle pRect = new Rectangle(
                    (int)(Game1.Player.Sprite.Position.X + motion.X),
                    (int)(Game1.Player.Sprite.Position.Y + motion.Y),
                    Engine.TileWidth,
                    Engine.TileHeight);

            if (pRect.Intersects(collision))
            {
                Game1.Player.Sprite.IsAnimating = false;
                inMotion = false;
                motion = Vector2.Zero;
            }

            foreach (Point p in engine.Map.CharacterLayer.Characters.Keys)
            {
                Rectangle r = new Rectangle(
                    p.X * Engine.TileWidth,
                    p.Y * Engine.TileHeight,
                    Engine.TileWidth,
                    Engine.TileHeight);

                if (r.Intersects(pRect))
                {
                    motion = Vector2.Zero;
                    Game1.Player.Sprite.IsAnimating = false;
                    inMotion = false;
```

```
            }
        }

        Vector2 newPosition = Game1.Player.Sprite.Position + motion;
        newPosition.X = (int)newPosition.X;
        newPosition.Y = (int)newPosition.Y;

        Game1.Player.Sprite.Position = newPosition;
        motion = Game1.Player.Sprite.LockToMap(
            new Point(
                map.WidthInPixels,
                map.HeightInPixels),
            motion);

        if (motion == Vector2.Zero)
        {
            Vector2 origin = new Vector2(
                    Game1.Player.Sprite.Position.X + Game1.Player.Sprite.Origin.X,
                    Game1.Player.Sprite.Position.Y + Game1.Player.Sprite.Origin.Y);
            Game1.Player.Sprite.Position = Engine.VectorFromOrigin(origin);
            inMotion = false;
            Game1.Player.Sprite.IsAnimating = false;
        }
    }
}

private void StartInteraction()
{
    foreach (Point s in engine.Map.CharacterLayer.Characters.Keys)
    {
        Character c = engine.Map.CharacterLayer.Characters[s];

        AnimationKey animation = Game1.Player.Sprite.CurrentAnimation;

        if (animation == AnimationKey.WalkLeft &&
            ((int)c.Sprite.Position.X > (int)Game1.Player.Sprite.Position.X ||
                (int)c.Sprite.Position.Y != (int)Game1.Player.Sprite.Position.Y))
        {
            continue;
        }

        if (animation == AnimationKey.WalkUp &&
            ((int)c.Sprite.Position.X != (int)Game1.Player.Sprite.Position.X ||
                (int)c.Sprite.Position.Y > (int)Game1.Player.Sprite.Position.Y))
        {
            continue;
        }

        if (animation == AnimationKey.WalkRight &&
            ((int)c.Sprite.Position.X < (int)Game1.Player.Sprite.Position.X ||
                (int)c.Sprite.Position.Y != (int)Game1.Player.Sprite.Position.Y))
        {
            continue;
        }

        if (animation == AnimationKey.WalkDown &&
            ((int)c.Sprite.Position.X != (int)Game1.Player.Sprite.Position.X ||
                (int)c.Sprite.Position.Y < (int)Game1.Player.Sprite.Position.Y))
        {
            continue;
```

```
        }

        float distance = Vector2.Distance(
            Game1.Player.Sprite.Origin + Game1.Player.Sprite.Position,
            c.Sprite.Origin + c.Sprite.Position);

        if (Math.Abs(distance) < Engine.TileWidth + Engine.TileWidth / 2)
        {
            manager.PushState(
                (ConversationState)GameRef.ConversationState);

            GameRef.ConversationState.SetConversation(c);
            GameRef.ConversationState.StartConversation();
            break;
        }
    }
}

private void StartBattle()
{
    foreach (Point s in engine.Map.CharacterLayer.Characters.Keys)
    {
        Character c = engine.Map.CharacterLayer.Characters[s];

        AnimationKey animation = Game1.Player.Sprite.CurrentAnimation;

        if (animation == AnimationKey.WalkLeft &&
            ((int)c.Sprite.Position.X > (int)Game1.Player.Sprite.Position.X ||
                (int)c.Sprite.Position.Y != (int)Game1.Player.Sprite.Position.Y))
        {
            continue;
        }

        if (animation == AnimationKey.WalkUp &&
            ((int)c.Sprite.Position.X != (int)Game1.Player.Sprite.Position.X ||
                (int)c.Sprite.Position.Y > (int)Game1.Player.Sprite.Position.Y))
        {
            continue;
        }

        if (animation == AnimationKey.WalkRight &&
            ((int)c.Sprite.Position.X < (int)Game1.Player.Sprite.Position.X ||
                (int)c.Sprite.Position.Y != (int)Game1.Player.Sprite.Position.Y))
        {
            continue;
        }

        if (animation == AnimationKey.WalkDown &&
            ((int)c.Sprite.Position.X != (int)Game1.Player.Sprite.Position.X ||
                (int)c.Sprite.Position.Y < (int)Game1.Player.Sprite.Position.Y))
        {
            continue;
        }

        float distance = Vector2.Distance(
            Game1.Player.Sprite.Origin + Game1.Player.Sprite.Position,
            c.Sprite.Origin + c.Sprite.Position);

        if (Math.Abs(distance) < Engine.TileWidth + Engine.TileWidth / 2 && c.Alive())
        {
```

```
            GameRef.StartBattleState.SetCombatants(Game1.Player, c);
            manager.PushState(GameRef.StartBattleState);
            break;
        }
    }
}

private void SaveGame()
{
    using (Aes aes = Aes.Create())
    {
        aes.IV = IV;
        aes.Key = Key;

        string path = Environment.GetFolderPath(
            Environment.SpecialFolder.ApplicationData);
        path += "\\ShadowMonsters\\";

        try
        {
            if (!Directory.Exists(path))
            {
                Directory.CreateDirectory(path);
            }
        }
        catch
        {
            // uh oh
        }

        try
        {
            ICryptoTransform encryptor = aes.CreateEncryptor(Key, IV);
            FileStream stream = new FileStream(
                path + "ShadowMonsters.sav",
                FileMode.Create,
                FileAccess.Write);
            using (CryptoStream cryptoStream = new CryptoStream(
                stream,
                encryptor,
                CryptoStreamMode.Write))
            {
                BinaryWriter writer = new BinaryWriter(cryptoStream);
                map.Save(writer);
                Game1.Player.Save(writer);
                writer.Close();
            }
            stream.Close();
            stream.Dispose();
        }
        catch
        {
            // uh oh
        }
    }
}

private void LoadGame()
{
    using (Aes aes = Aes.Create())
```

```
            {
                aes.IV = IV;
                aes.Key = Key;

                string path = Environment.GetFolderPath(
                    Environment.SpecialFolder.ApplicationData);
                path += "\\ShadowMonsters\\";

                try
                {
                    ICryptoTransform decryptor = aes.CreateDecryptor(Key, IV);
                    FileStream stream = new FileStream(
                        path + "ShadowMonsters.sav",
                        FileMode.Open,
                        FileAccess.Read);
                    using (CryptoStream cryptoStream = new CryptoStream(
                        stream,
                        decryptor,
                        CryptoStreamMode.Read))
                    {
                        BinaryReader reader = new BinaryReader(cryptoStream);
                        map = TileMap.Load(content, reader);
                        Game1.Player = Player.Load(GameRef, reader);
                        reader.Close();
                    }
                    stream.Close();
                    stream.Dispose();
                }
                catch (Exception exc)
                {
                    exc.ToString();
                }
            }
        }

        public override void Draw(GameTime gameTime)
        {
            base.Draw(gameTime);
            engine.Draw(gameTime, GameRef.SpriteBatch);
            GameRef.SpriteBatch.Begin(
                SpriteSortMode.Deferred,
                BlendState.AlphaBlend,
                SamplerState.PointClamp,
                null,
                null,
                null,
                Engine.Camera.Transformation);
            Game1.Player.Draw(gameTime);
            GameRef.SpriteBatch.End();
        }
    }
}
```

Nothing really complicated here. I pulled all of the movement code out of the Update method and placed it inside the HandleMovement method. It required a GameTime parameter because it normalizes the movement based on time. I extracted the code for starting a conversation and placed it into the StartInteraction method. I extracted the code for starting a battle into the StartBattle method. The save and load code has been placed into the SaveGame and LoadGame methods respectively.

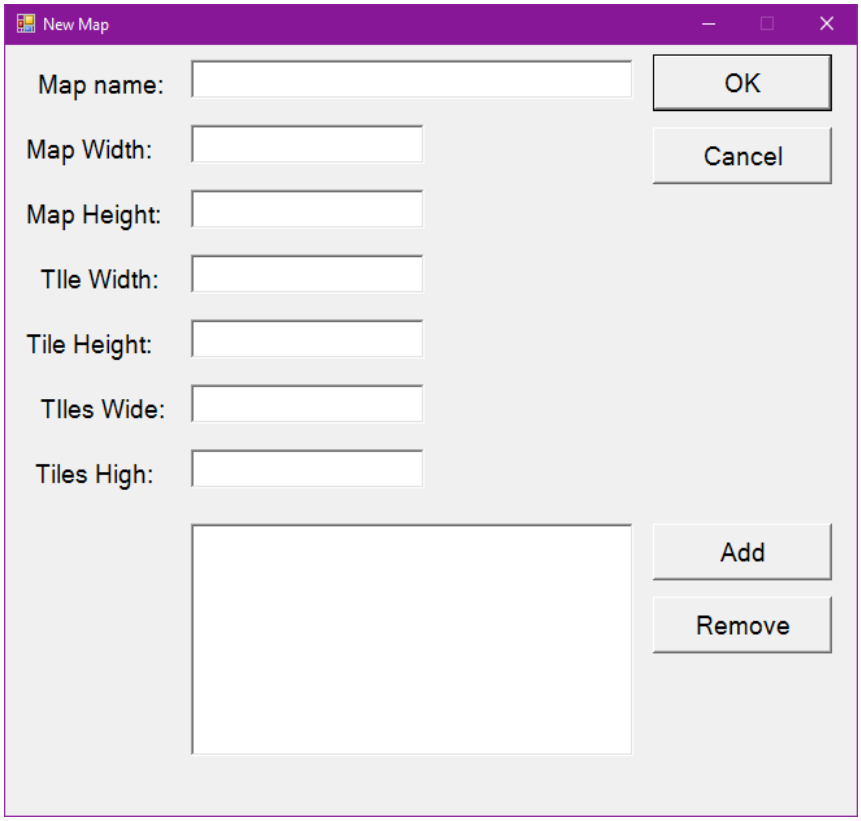
For the editor I'm going to add a second MonoGame project and add a reference to the System.Windows.Forms namespace. In the Solution Explorer right click the ShadowMonsters solution, select Add and then New Project. Search for MonoGame in the search box and select MionoGame Cross Platform Desktop Project. Now click the Next button. Enter ShadowEditor for the Name and click the Create button. In the new project that is created right click the Game1.cs file and select Rename. Change the name of the file to Game1.cs. If you are prompted to rename the class to Editor select Yes. Make sure that Program.cs is the following.

```
using System;

namespace ShadowEditor
{
    /// <summary>
    /// The main class.
    /// </summary>
    public static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            using (var game = new Editor())
                game.Run();
        }
    }
}
```

Now I'm going to add a couple references. Right click the ShadowEditor projecct in the Solution Explorer and select Add and then Reference. Click the Projects option on the left and select the ShadowMonsters project. Click the Assemblies option on the left and add System.Windows.Forms.

Now I'm going to add a form for creating new maps. This is what the form will look like when it is finished. It works by entering the name of the map, map width and height, the width of the tiles in the tile set, the height of the tiles in the tile set, the number of tiles wide the tile set is and the number of tiles high the tile set is. The different images for the tile set are added by clicking the Add button.



Right click the ShadowEditor project, select Add and then Form (Windows Form). Name this form NewMapForm. In the Solution Explorer expand the NewMapForm. Replace the contents of the NewMapForm.Designer.cs with the following code.

```
namespace ShadowEditor
{
    partial class NewMapForm
    {
        /// <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.txtName = new System.Windows.Forms.TextBox();
    this.btnOK = new System.Windows.Forms.Button();
    this.btnCancel = new System.Windows.Forms.Button();
    this.label2 = new System.Windows.Forms.Label();
    this.txtWidth = new System.Windows.Forms.TextBox();
    this.txtHeight = new System.Windows.Forms.TextBox();
    this.label3 = new System.Windows.Forms.Label();
    this.lbTIleSetImages = new System.Windows.Forms.ListBox();
    this.label4 = new System.Windows.Forms.Label();
    this.label5 = new System.Windows.Forms.Label();
    this.txtTileSetWidth = new System.Windows.Forms.TextBox();
    this.txtTileSetHeight = new System.Windows.Forms.TextBox();
    this.btnAdd = new System.Windows.Forms.Button();
    this.btnRemove = new System.Windows.Forms.Button();
    this.label6 = new System.Windows.Forms.Label();
    this.label7 = new System.Windows.Forms.Label();
    this.txtTilesWide = new System.Windows.Forms.TextBox();
    this.txtTilesHigh = new System.Windows.Forms.TextBox();
    this.SuspendLayout();
    //
    // label1
    //
    this.label1.AutoSize = true;
    this.label1.Location = new System.Drawing.Point(12, 9);
    this.label1.Name = "label1";
    this.label1.Size = new System.Drawing.Size(60, 13);
    this.label1.TabIndex = 0;
    this.label1.Text = "Map name:";
    //
    // txtName
    //
    this.txtName.Location = new System.Drawing.Point(78, 6);
    this.txtName.Name = "txtName";
    this.txtName.Size = new System.Drawing.Size(188, 20);
    this.txtName.TabIndex = 1;
    //
    // btnOK
    //
    this.btnOK.Location = new System.Drawing.Point(272, 4);
    this.btnOK.Name = "btnOK";
    this.btnOK.Size = new System.Drawing.Size(75, 23);
    this.btnOK.TabIndex = 17;
    this.btnOK.Text = "OK";
    this.btnOK.UseVisualStyleBackColor = true;
    //
    // btnCancel
    //
    this.btnCancel.Location = new System.Drawing.Point(272, 33);
    this.btnCancel.Name = "btnCancel";
    this.btnCancel.Size = new System.Drawing.Size(75, 23);
    this.btnCancel.TabIndex = 18;
    this.btnCancel.Text = "Cancel";
    this.btnCancel.UseVisualStyleBackColor = true;
```

```
//
// label2
//
this.label2.AutoSize = true;
this.label2.Location = new System.Drawing.Point(7, 35);
this.label2.Name = "label2";
this.label2.Size = new System.Drawing.Size(62, 13);
this.label2.TabIndex = 2;
this.label2.Text = "Map Width:";
//
// txtWidth
//
this.txtWidth.Location = new System.Drawing.Point(78, 32);
this.txtWidth.Name = "txtWidth";
this.txtWidth.Size = new System.Drawing.Size(100, 20);
this.txtWidth.TabIndex = 3;
//
// txtHeight
//
this.txtHeight.Location = new System.Drawing.Point(78, 58);
this.txtHeight.Name = "txtHeight";
this.txtHeight.Size = new System.Drawing.Size(100, 20);
this.txtHeight.TabIndex = 5;
//
// label3
//
this.label3.AutoSize = true;
this.label3.Location = new System.Drawing.Point(7, 61);
this.label3.Name = "label3";
this.label3.Size = new System.Drawing.Size(65, 13);
this.label3.TabIndex = 4;
this.label3.Text = "Map Height:";
//
// lbTIleSetImages
//
this.lbTIleSetImages.FormattingEnabled = true;
this.lbTIleSetImages.Location = new System.Drawing.Point(78, 192);
this.lbTIleSetImages.Name = "lbTIleSetImages";
this.lbTIleSetImages.Size = new System.Drawing.Size(188, 95);
this.lbTIleSetImages.TabIndex = 14;
//
// label4
//
this.label4.AutoSize = true;
this.label4.Location = new System.Drawing.Point(13, 87);
this.label4.Name = "label4";
this.label4.Size = new System.Drawing.Size(59, 13);
this.label4.TabIndex = 6;
this.label4.Text = "TIle Width:";
//
// label5
//
this.label5.AutoSize = true;
this.label5.Location = new System.Drawing.Point(7, 113);
this.label5.Name = "label5";
this.label5.Size = new System.Drawing.Size(61, 13);
this.label5.TabIndex = 8;
this.label5.Text = "Tile Height:";
//
// txtTileSetWidth
```

```
//
this.txtTileSetWidth.Location = new System.Drawing.Point(78, 84);
this.txtTileSetWidth.Name = "txtTileSetWidth";
this.txtTileSetWidth.Size = new System.Drawing.Size(100, 20);
this.txtTileSetWidth.TabIndex = 7;
//
// txtTileSetHeight
//
this.txtTileSetHeight.Location = new System.Drawing.Point(78, 110);
this.txtTileSetHeight.Name = "txtTileSetHeight";
this.txtTileSetHeight.Size = new System.Drawing.Size(100, 20);
this.txtTileSetHeight.TabIndex = 9;
//
// btnAdd
//
this.btnAdd.Location = new System.Drawing.Point(272, 192);
this.btnAdd.Name = "btnAdd";
this.btnAdd.Size = new System.Drawing.Size(75, 23);
this.btnAdd.TabIndex = 15;
this.btnAdd.Text = "Add";
this.btnAdd.UseVisualStyleBackColor = true;
//
// btnRemove
//
this.btnRemove.Location = new System.Drawing.Point(272, 221);
this.btnRemove.Name = "btnRemove";
this.btnRemove.Size = new System.Drawing.Size(75, 23);
this.btnRemove.TabIndex = 16;
this.btnRemove.Text = "Remove";
this.btnRemove.UseVisualStyleBackColor = true;
//
// label6
//
this.label6.AutoSize = true;
this.label6.Location = new System.Drawing.Point(13, 139);
this.label6.Name = "label6";
this.label6.Size = new System.Drawing.Size(61, 13);
this.label6.TabIndex = 10;
this.label6.Text = "TIles Wide:";
//
// label7
//
this.label7.AutoSize = true;
this.label7.Location = new System.Drawing.Point(11, 165);
this.label7.Name = "label7";
this.label7.Size = new System.Drawing.Size(57, 13);
this.label7.TabIndex = 12;
this.label7.Text = "Tiles High:";
//
// txtTilesWide
//
this.txtTilesWide.Location = new System.Drawing.Point(78, 136);
this.txtTilesWide.Name = "txtTilesWide";
this.txtTilesWide.Size = new System.Drawing.Size(100, 20);
this.txtTilesWide.TabIndex = 11;
//
// txtTilesHigh
//
this.txtTilesHigh.Location = new System.Drawing.Point(78, 162);
this.txtTilesHigh.Name = "txtTilesHigh";
```

```
            this.txtTilesHigh.Size = new System.Drawing.Size(100, 20);
            this.txtTilesHigh.TabIndex = 13;
            //
            // FormNew
            //
            this.AcceptButton = this.btnOK;
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(358, 309);
            this.Controls.Add(this.btnRemove);
            this.Controls.Add(this.btnAdd);
            this.Controls.Add(this.txtTilesHigh);
            this.Controls.Add(this.txtTileSetHeight);
            this.Controls.Add(this.txtTilesWide);
            this.Controls.Add(this.label7);
            this.Controls.Add(this.txtTileSetWidth);
            this.Controls.Add(this.label6);
            this.Controls.Add(this.label5);
            this.Controls.Add(this.label4);
            this.Controls.Add(this.lbTIleSetImages);
            this.Controls.Add(this.label3);
            this.Controls.Add(this.txtHeight);
            this.Controls.Add(this.txtWidth);
            this.Controls.Add(this.label2);
            this.Controls.Add(this.btnCancel);
            this.Controls.Add(this.btnOK);
            this.Controls.Add(this.txtName);
            this.Controls.Add(this.label1);
            this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedSingle;
            this.Margin = new System.Windows.Forms.Padding(2);
            this.MaximizeBox = false;
            this.Name = "FormNew";
            this.Text = "New Map";
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.TextBox txtName;
        private System.Windows.Forms.Button btnOK;
        private System.Windows.Forms.Button btnCancel;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.TextBox txtWidth;
        private System.Windows.Forms.TextBox txtHeight;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.ListBox lbTIleSetImages;
        private System.Windows.Forms.Label label4;
        private System.Windows.Forms.Label label5;
        private System.Windows.Forms.TextBox txtTileSetWidth;
        private System.Windows.Forms.TextBox txtTileSetHeight;
        private System.Windows.Forms.Button btnAdd;
        private System.Windows.Forms.Button btnRemove;
        private System.Windows.Forms.Label label6;
        private System.Windows.Forms.Label label7;
        private System.Windows.Forms.TextBox txtTilesWide;
        private System.Windows.Forms.TextBox txtTilesHigh;
    }
```

```
}
```

This is all code generated by Visual Studio so I'm not going to explain it. It was easier to replace the template code with the code generated in my editor than to drag the controls onto the form one by one and set the properties. Now to add the logic to the form. Right click the NewMapForm.cs in the Solution Explorer and select View Code. Replace the code with the following.

```
using Microsoft.Xna.Framework.Graphics;
using ShadowMonsters.TileEngine;
using System;
using System.IO;
using System.Windows.Forms;

namespace ShadowEditor
{
    public partial class NewMapForm : Form
    {
        public bool OkPressed { get; private set; }
        public TileSet TileSet { get; private set; }
        private GraphicsDevice graphics;
        public TileMap TileMap { get; private set; }

        public NewMapForm(GraphicsDevice graphics)
        {
            InitializeComponent();
            this.graphics = graphics;

            btnOK.Click += BtnOK_Click;
            btnCancel.Click += BtnCancel_Click;

            btnAdd.Click += BtnAdd_Click;
            btnRemove.Click += BtnRemove_Click;
        }

        private void BtnOK_Click(object sender, EventArgs e)
        {
            if (!int.TryParse(txtWidth.Text, out int mapWidth))
            {
                MessageBox.Show("Map Width must be numeric");
                return;
            }

            if (mapWidth <= 0)
            {
                MessageBox.Show("Map Width must be greater than zero.");
                return;
            }

            if (!int.TryParse(txtHeight.Text, out int mapHeight))
            {
                MessageBox.Show("Map Height must be numeric");
                return;
            }

            if (mapHeight <= 0)
            {
                MessageBox.Show("Map Height must be greater than zero.");
                return;
```

```
}

if (!int.TryParse(txtTileSetWidth.Text, out int tileWidth))
{
    MessageBox.Show("Tile Width must be numeric");
    return;
}

if (tileWidth <= 0)
{
    MessageBox.Show("Tile Width must be greater than zero.");
    return;
}

if (!int.TryParse(txtTileSetHeight.Text, out int tileHeight))
{
    MessageBox.Show("Tile Height must be numeric");
    return;
}

if (tileHeight <= 0)
{
    MessageBox.Show("Tile Height must be greater than zero.");
    return;
}

if (!int.TryParse(txtTilesWide.Text, out int tilesWide))
{
    MessageBox.Show("Tiles Wide must be numeric");
    return;
}

if (tilesWide <= 0)
{
    MessageBox.Show("Tile Wide must be greater than zero.");
    return;
}

if (!int.TryParse(txtTilesHigh.Text, out int tilesHigh))
{
    MessageBox.Show("Tiles High must be numeric");
    return;
}

if (tilesHigh <= 0)
{
    MessageBox.Show("Tiles High must be greater than zero.");
    return;
}

if (lbTIleSetImages.Items.Count == 0)
{
    MessageBox.Show("Map needs at least one image for the tile set");
    return;
}

TileLayer gr = new TileLayer(mapWidth, mapHeight, -1, -1);
TileLayer ed = new TileLayer(mapWidth, mapHeight, -1, -1);
TileLayer bu = new TileLayer(mapWidth, mapHeight, -1, -1);
TileLayer de = new TileLayer(mapWidth, mapHeight, -1, -1);
```

```
            TileSet = new TileSet(tilesWide, tilesHigh, tileWidth, tileHeight);

            foreach (object o in lbTIleSetImages.Items)
            {
                string s = o.ToString();
                FileStream stream = new FileStream(s, FileMode.Open);
                Texture2D t = Texture2D.FromStream(graphics, stream);
                TileSet.Textures.Add(t);
                TileSet.TextureNames.Add(Path.GetFileNameWithoutExtension(s));
            }

            TileMap = new TileMap(TileSet, gr, ed, bu, de, txtName.Text);
            OkPressed = true;
            Close();
        }

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

        private void BtnRemove_Click(object sender, EventArgs e)
        {
            if (lbTIleSetImages.SelectedIndex >= 0)
            {
                lbTIleSetImages.Items.RemoveAt(lbTIleSetImages.SelectedIndex);
            }
        }

        private void BtnAdd_Click(object sender, EventArgs e)
        {
            OpenFileDialog ofd = new OpenFileDialog
            {
                Filter = "Image Files|*.BMP;*.GIF;*.JPG;*.TGA;*.PNG"
            };
            ofd.Multiselect = true;

            DialogResult result = ofd.ShowDialog();

            if (result == DialogResult.OK)
            {
                foreach (string s in ofd.FileNames)
                    lbTIleSetImages.Items.Add(s);
            }
        }
    }
}
```

There are four autoproperties. OkPressed lets the editor know how the form was closed. TileSet holds the TileSet that is created. The graphics property is used for loading images as Texture2Ds. TileMap is the map that is generated if the user clicks the OK button.

The constructor requires a GraphicsDevice parameter. After initializing the components on the form the graphics property is assigned. It then wires the event handlers for the buttons on the form.

The event handler for the OK button does a lot of validation on the input fields. It checks that the inputs are numeric and that they are greater than 0. It then creates the layers for the map

and the tile set. It then loops over the images that we added for the tile sets. It gets a string from the item in the list box. It then opens a FileStream for creating a Texture2D. Next it creates a Texture2D from the stream. The texture is added to the tile set and the name of the file without extension is added to tile set. The map is created using the tile set, layers and the name of the map. OkPressed is set to true and the form is closed.

The event handler for the Cancel button just closes the form. The event handler for the Remove button checks to see if an item in the list box is selected. If it is the item with that index is removed from the list box.

The event handler for the Add button creates an OpenFileDialog to allow the user to select a file. The filter for the dialog allows for BMP, PNG, GIF and TGA files. I then grab the result of displaying the OpenFileDialog. If the OpenFileDialog was closed by clicking the OK button it loops over the selected file names and adds them to the list box.

The last thing that I'm going to do in this tutorial is wire creating a new map and some basic tile painting. Replace the contents of the Editor.cs file with the following code.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using ShadowMonsters;
using ShadowMonsters.TileEngine;
using WF = System.Windows.Forms;

namespace ShadowEditor
{
    /// <summary>
    /// This is the main type for your game.
    /// </summary>
    public class Editor : Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;
        int frameCount = 0;
        TileMap map;
        Camera camera;
        Rectangle viewPort = new Rectangle(0, 0, 64 * 18, 1080);

        public Editor()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
            new Engine(viewPort);
            camera = new Camera();

            graphics.PreferredBackBufferWidth = 1920;
            graphics.PreferredBackBufferHeight = 1080;
            graphics.ApplyChanges();
        }

        /// <summary>
        /// Allows the game to perform any initialization it needs to before starting to run.
        /// This is where it can query for any required services and load any non-graphic
        /// related content.  Calling base.Initialize will enumerate through any components
        /// and initialize them as well.
```

```
        /// </summary>
        protected override void Initialize()
        {
            // TODO: Add your initialization logic here
            Components.Add(new Xin(this));
            base.Initialize();
        }

        /// <summary>
        /// LoadContent will be called once per game and is the place to load
        /// all of your content.
        /// </summary>
        protected override void LoadContent()
        {
            // Create a new SpriteBatch, which can be used to draw textures.
            spriteBatch = new SpriteBatch(GraphicsDevice);

            // TODO: use this.Content to load your game content here
        }

        /// <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.N) && frameCount > 5)
            {
                NewMapForm form = new NewMapForm(GraphicsDevice);

                form.ShowDialog();

                if (form.OkPressed)
                {
                    map = form.TileMap;
                }

                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 (map != null && viewPort.Contains(Xin.MouseAsPoint))
            {
                if (Xin.MouseState.LeftButton == ButtonState.Pressed)
                {
                    map.GroundLayer.SetTile(tile.X, tile.Y, 0, 0);
                }

                if (Xin.MouseState.RightButton == ButtonState.Pressed)
                {
                    map.GroundLayer.SetTile(tile.X, tile.Y, -1, -1);
                }
            }

            base.Update(gameTime);
        }

        /// <summary>
        /// This is called when the game should draw itself.
        /// </summary>
        /// <param name="gameTime">Provides a snapshot of timing values.</param>
        protected override void Draw(GameTime gameTime)
        {
            GraphicsDevice.Clear(Color.CornflowerBlue);

            // TODO: Add your drawing code here

            if (map != null)
            {
                map.Draw(gameTime, spriteBatch, camera, true);
            }

            base.Draw(gameTime);
        }
    }
}
```

I added four fields to this class. The first is frameCount and is similar to what was used in the Update method of the GamePlayState. It will prevent multiple forms from popping up. The map field is the map being edited. The camera field is used for rendering the map and determining what tile to paint a tile in. Finally viewPort describes the area that the editor is.

In the constructor I create an Engine. I then create a Camera using the viewPort field that I created. I set the width of the window to 1920 and the height to 1080. I then apply the changes so that they will take effect. In the Initialize method I add a Xin to the list of components.

In the Update method I check if the editor is not that active window and if it is not I return out of the method. After checking for the exit criteria I increment the frameCount field. I then

check if the N key has been released and frameCount is greater than 5. I then create a NewMapForm and call its ShowDialog method. If the form was closed by pressing the OK button I assign the map field to the map that was created.

I then get the position of the mouse plus the camera's position. This is used to determine what tile to paint. It is calculated using the Vector2Cell method of the Engine class.

Next I check if the map field is not null and that the viewPort contains the mouse. If the left mouse button is pressed I call the SetTile method of the ground layer passing in the tile and 0 for the tile set and 0 for the tile index. If the right mouse button is pressed I call the SetTile method of the ground layer in the tile and -1 for the tile set and -1 to the tile index.

In the Draw method I check to see if the map is not null. If it is not I call the Draw method.

I'm going to wrap this tutorial up here. 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!