Shadow Monsters – MonoGame Tutorial Series
Chapter 17
World

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.

In this tutorial I will be updating the game and editor to use a world instead of a map. A world is a collection of maps. Right click the TileEngine folder under the ShadowMonster project in the Solution Explorer, select Add and then Class. Name this new class World. Here is the code.

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

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

        private readonly Dictionary<string, TileMap> maps = new Dictionary<string, TileMap>();
        private string currentMapName;
        private Portal startingMap;

        #endregion

        #region Property Region

        public Dictionary<string, TileMap> Maps
```

```
        {
            get { return maps; }
        }

        public TileMap Map
        {
            get { return maps[currentMapName]; }
        }

        public string CurrentMapName
        {
            get { return currentMapName; }
        }

        public Portal StartingMap
        {
            get { return startingMap; }
        }

        #endregion

        #region Constructor Region

        private World()
        {
        }

        public World(Portal portal)
        {
            startingMap = portal;
        }

        #endregion

        #region Method Region

        public void ChangeMap(Portal portal)
        {
            if (maps.ContainsKey(portal.DestinationLevel))
            {
                currentMapName = portal.DestinationLevel;
            }
        }

        public void ChangeMap(string mapName)
        {
            if (maps.ContainsKey(mapName))
            {
                currentMapName = mapName;
            }
        }

        public void Update(GameTime gameTime)
        {
            maps[currentMapName].Update(gameTime);
        }

        public void Draw(GameTime gameTime, SpriteBatch spriteBatch, Camera camera, bool debug
= false)
        {
```

```
            maps[currentMapName].Draw(gameTime, spriteBatch, camera, debug);
        }

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

            foreach (TileMap map in maps.Values)
            {
                map.Save(writer);
            }
        }

        public static World Load(ContentManager content, BinaryReader reader)
        {
            World world = new World();

            Portal p = Portal.Load(reader);

            int count = reader.ReadInt32();

            for (int i = 0; i < count; i++)
            {
                TileMap map = TileMap.Load(content, reader);
                world.Maps.Add(map.MapName, map);
            }

            world.ChangeMap(p);

            return world;
        }
        #endregion
    }
}
```

There are three fields in this class. The Dictionary<string, TileMap> holds the maps in the world. currentMapName is the name of the current map. There is a Portal, startingMap, that is the destination on the first map that will be loaded. There a read only properties that expose the fields. There are properties that expose the values of the fields. There is also a property that returns the current map.

There are two constructors. The first is private and is used for loading in worlds. The second takes a Portal parameter and sets the startingMap field.

There are two ChangeMap methods that are used to change the map. The first takes a Portal parameter. It checks to see if the DestinationLevel property of the Portal passed in is a Key. If it is I set the currentMapName field to the DestinationLevel property of the portal. The second takes a string parameter that is the key in the maps collection. It checks to see if the key passed in is in the collection. If it is it updates the currentMap field to the value passed in.

There is also an Update method that accepts a GameTime parameter. It calls the Update method of the current map.

There is a Draw method that accepts a GameTime, SpriteBatch, Camera and optional bool parameter. It calls the Draw method of the current map.

The Save method requires a BinaryWriter. It calls the Save method of the startingPortal field passing in the BinraryWriter. It then writes the number of maps. Next it loops over all of the maps and calls their Save method.

The Load method requires a ContentManager parameter and a BinaryReader parameter. It creates a new World instance using the private constructor. The next step is to call the Load method of the Portal class to load the portal. Now it reads the number of maps. Looping for the number of maps it calls the Load method of the TileMap passing in the required parameters. After loading the map they are added to the map collection.

Now I will update the GamePlayState to use the World class that I just added. There were many changes so I will give the code for the entire class. Update the GamePlayState to the following.

```
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 World world;
        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));
            }

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

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

            map.CharacterLayer.Characters.Add(c.SourceTile, c);

            Merchant m = Merchant.FromString(GameRef,
"Bonnie,ninja_f,WalkLeft,BonnieHello,0,4:4,earth1,earth1,,,,,,");
            m.Sprite.Position = new Vector2(
                m.SourceTile.X * Engine.TileWidth,
                m.SourceTile.Y * Engine.TileHeight);

            m.Backpack.AddItem("Potion", 99);
            m.Backpack.AddItem("Antidote", 10);

            map.CharacterLayer.Characters.Add(m.SourceTile, m);

            world = new World(new Portal(new Point(10, 10), new Point(10, 10), map.MapName));
            world.Maps.Add(map.MapName, map);
            world.ChangeMap(map.MapName);
            Game1.Player.Sprite.Position = new Vector2(
                world.StartingMap.SourceTile.X * Engine.TileWidth,
                world.StartingMap.SourceTile.Y * Engine.TileHeight);

            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)
        {
            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(
        world.Maps[world.CurrentMapName],
        Game1.Player.Sprite,
        new Rectangle(0, 0, 1280, 720));
    world.Update(gameTime);
    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 world.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(
                world.Map.WidthInPixels,
                world.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 world.Map.CharacterLayer.Characters.Keys)
    {
        Character c = world.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 world.Map.CharacterLayer.Characters.Keys)
    {
        Character c = world.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);
                world.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);
                        world = World.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);
            world.Draw(gameTime, GameRef.SpriteBatch, Engine.Camera);
            GameRef.SpriteBatch.Begin(
                SpriteSortMode.Deferred,
                BlendState.AlphaBlend,
                SamplerState.PointClamp,
                null,
                null,
                null,
                Engine.Camera.Transformation);
            Game1.Player.Draw(gameTime);
            GameRef.SpriteBatch.End();
        }
    }
}
```

There is a new World field. In the LoadContent method after creating the test map I create a new World with a Portal with a SourceTile of 10,10, a DestinationTile of 10,10 and a DestinationLevel of map.MapName. The map is then added to the Maps collection. It then calls the ChangeMap method passing in map.MapName. It then sets the Position of the player's sprite to the SourceTile times the width and height of tiles.

The Update method now calls LockToSprite passing in the current map of the world instead of the engine. It also calls the Update method of the world.

The HandleMovement method now loops over the characters of the current map of the world instead of the engine. It also uses the WidthInPixels and HeightInPixels property of the current map of the world. StartInteraction now uses the world instead of the engine of the world when looping over the characters. It also uses the world when getting the character. StartBattle now uses the world instead of the engine as well. In SaveGame I call Save on the world field instead of engine.Map. In LoadGame I call Load of the World class instead of the TileMap class. The Draw method now calls the Draw method of the World class.

I will move on to the editor now. To start with I added in a form for creating a new world. Right click the ShadowEditor project in the Solution Explorer, select Add and then Form (Windows Forms). Name this new form WorldForm. Expand the node beside WorldForm and open the WorldForm.Designer.cs file. Replace the code with the following.

```
namespace ShadowEditor
{
    partial class WorldForm
    {
        /// <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.TxtSourceTile = new System.Windows.Forms.TextBox();
            this.label2 = new System.Windows.Forms.Label();
            this.TxtDestinationTile = new System.Windows.Forms.TextBox();
            this.label3 = new System.Windows.Forms.Label();
            this.TxtDestinationLevel = new System.Windows.Forms.TextBox();
            this.BtnOK = new System.Windows.Forms.Button();
            this.BtnCancel = new System.Windows.Forms.Button();
            this.SuspendLayout();
            //
            // label1
            //
            this.label1.AutoSize = true;
```

```
this.label1.Location = new System.Drawing.Point(49, 13);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(64, 13);
this.label1.TabIndex = 0;
this.label1.Text = "Source Tile:";
//
// TxtSourceTile
//
this.TxtSourceTile.Location = new System.Drawing.Point(119, 10);
this.TxtSourceTile.Name = "TxtSourceTile";
this.TxtSourceTile.Size = new System.Drawing.Size(100, 20);
this.TxtSourceTile.TabIndex = 1;
//
// label2
//
this.label2.AutoSize = true;
this.label2.Location = new System.Drawing.Point(30, 39);
this.label2.Name = "label2";
this.label2.Size = new System.Drawing.Size(83, 13);
this.label2.TabIndex = 2;
this.label2.Text = "Destination Tile:";
//
// TxtDestinationTile
//
this.TxtDestinationTile.Location = new System.Drawing.Point(119, 36);
this.TxtDestinationTile.Name = "TxtDestinationTile";
this.TxtDestinationTile.Size = new System.Drawing.Size(100, 20);
this.TxtDestinationTile.TabIndex = 3;
//
// label3
//
this.label3.AutoSize = true;
this.label3.Location = new System.Drawing.Point(21, 65);
this.label3.Name = "label3";
this.label3.Size = new System.Drawing.Size(92, 13);
this.label3.TabIndex = 4;
this.label3.Text = "Destination Level:";
//
// TxtDestinationLevel
//
this.TxtDestinationLevel.Location = new System.Drawing.Point(119, 62);
this.TxtDestinationLevel.Name = "TxtDestinationLevel";
this.TxtDestinationLevel.Size = new System.Drawing.Size(100, 20);
this.TxtDestinationLevel.TabIndex = 6;
//
// btnOK
//
this.BtnOK.Location = new System.Drawing.Point(241, 8);
this.BtnOK.Name = "btnOK";
this.BtnOK.Size = new System.Drawing.Size(75, 23);
this.BtnOK.TabIndex = 7;
this.BtnOK.Text = "OK";
this.BtnOK.UseVisualStyleBackColor = true;
//
// btnCancel
//
this.BtnCancel.Location = new System.Drawing.Point(241, 37);
this.BtnCancel.Name = "btnCancel";
this.BtnCancel.Size = new System.Drawing.Size(75, 23);
this.BtnCancel.TabIndex = 8;
```

```
            this.BtnCancel.Text = "Cancel";
            this.BtnCancel.UseVisualStyleBackColor = true;
            //
            // WorldForm
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(350, 100);
            this.Controls.Add(this.BtnCancel);
            this.Controls.Add(this.BtnOK);
            this.Controls.Add(this.TxtDestinationLevel);
            this.Controls.Add(this.label3);
            this.Controls.Add(this.TxtDestinationTile);
            this.Controls.Add(this.label2);
            this.Controls.Add(this.TxtSourceTile);
            this.Controls.Add(this.label1);
            this.Name = "World Form";
            this.Text = "World Form";
            this.Load += new System.EventHandler(this.WorldForm_Load);
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.TextBox TxtSourceTile;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.TextBox TxtDestinationTile;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.TextBox TxtDestinationLevel;
        private System.Windows.Forms.Button BtnOK;
        private System.Windows.Forms.Button BtnCancel;
    }
}
```

The next step is to wire the logic for the form. Select the form in the Solution Explorer and press F7 to bring up the code view. Replace the code with the following code.

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

namespace ShadowEditor
{
    public partial class WorldForm : Form
    {
        public World World { get; set; }
        public bool OKPressed { get; set; }

        public WorldForm()
        {
```

```
            InitializeComponent();
        }

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

        private void BtnOK_Click(object sender, EventArgs e)
        {
            if (string.IsNullOrEmpty(TxtSourceTile.Text))
            {
                MessageBox.Show("You must enter a source tile.");
                return;
            }

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

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

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

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

            if (!int.TryParse(src[1], out int ycoord))
            {
                MessageBox.Show("Source tile y-coordinate must be numeric.");
                return;
            }

            Microsoft.Xna.Framework.Point s = new Microsoft.Xna.Framework.Point(xcoord,
ycoord);

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

            if (!int.TryParse(dest[1], out ycoord))
            {
                MessageBox.Show("Destination tile y-coordinate myst be numeric.");
                return;
            }

            Microsoft.Xna.Framework.Point d = new Microsoft.Xna.Framework.Point(xcoord,
```

```
ycoord);

            Portal p = new Portal(s, d, TxtDestinationLevel.Text);
            World = new World(p);

            OKPressed = true;
            Close();
        }

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

There is a World property that is get and set. The same is true for the bool property OKPressed. In the Load event handler of the form I wire the event handlers for the Click event of the OK and Cancel buttons.

In the event handler of the Click event for the OK button I check to see if the source tile is null or empty. If it is I exit the method. I do the same for destination tile and the destination level. I split the source tile and destination tile text into its parts based on the colon.
Now I will update the editor to use the World class. The changes are extensive so I will give the code for the entire class. If converting any of the parts to an integer fails I display an error message and exit the method. I also create Points for the source and destination tiles. I then create a Portal and a World using the Portal. I set OKPressed to true and close the form.

In the event handler of the Click event for the Cancel button I set OKPressed to fals and close the form.

Time to turn our attention to the Editor class. There were a lot of changes so I will give you the code for the entire class.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using ShadowMonsters;
using ShadowMonsters.Controls;
using ShadowMonsters.ShadowMonsters;
using ShadowMonsters.TileEngine;
using System;
using System.IO;
using System.Security.Cryptography;
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;
World world;
Camera camera;
Rectangle viewPort = new Rectangle(0, 0, 64 * 18, 1080);
ControlManager controls;

PictureBox pbTileset;
PictureBox pbPreview;
ListBox lbLayers;
ListBox lbTileSets;
ListBox lbWorld;
int selectedTile;
bool Paint = true;
int brushSize = 1;

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

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

    IsMouseVisible = true;
}

/// <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));
    Game1.BuildAnimations();
    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);

    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.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);
}

private void HandleScrollMap()
{
    if (world == null || world.Maps.Count == 0)
    {
        return;
    }

    if (Xin.MouseAsPoint.X > 64 * 17 && Xin.MouseAsPoint.X < 64 * 18)
    {
        camera.Position = new Vector2(camera.Position.X + 8, camera.Position.Y);
    }

    if (Xin.KeyboardState.IsKeyDown(Keys.Right))
    {
        camera.Position = new Vector2(camera.Position.X + 8, camera.Position.Y);
    }

    if (Xin.MouseAsPoint.Y > 1080 - 64 && Xin.MouseAsPoint.Y < 1080)
    {
        camera.Position = new Vector2(camera.Position.X, camera.Position.Y + 8);
    }

    if (Xin.KeyboardState.IsKeyDown(Keys.Down))
    {
        camera.Position = new Vector2(camera.Position.X, camera.Position.Y + 8);
    }

    if (Xin.MouseAsPoint.X < 64)
```

```
            {
                camera.Position = new Vector2(camera.Position.X - 8, camera.Position.Y);
            }

            if (Xin.KeyboardState.IsKeyDown(Keys.Left))
            {
                camera.Position = new Vector2(camera.Position.X - 8, camera.Position.Y);
            }

            if (Xin.MouseAsPoint.Y < 64)
            {
                camera.Position = new Vector2(camera.Position.X, camera.Position.Y - 8);
            }

            if (Xin.KeyboardState.IsKeyDown(Keys.Up))
            {
                camera.Position = new Vector2(camera.Position.X, camera.Position.Y - 8);
            }

            if (world != null && world.Maps.Count > 0)
            {
                camera.LockCamera(world.Map, viewPort);
            }
        }

        private void SaveWorld(string fileName)
        {
            using (Aes aes = Aes.Create())
            {
                aes.IV = IV;
                aes.Key = Key;

                try
                {
                    ICryptoTransform encryptor = aes.CreateEncryptor(Key, IV);
                    FileStream stream = new FileStream(
                        fileName,
                        FileMode.Create,
                        FileAccess.Write);
                    using (CryptoStream cryptoStream = new CryptoStream(
                        stream,
                        encryptor,
                        CryptoStreamMode.Write))
                    {
                        BinaryWriter writer = new BinaryWriter(cryptoStream);
                        world.Save(writer);
                        writer.Close();
                    }
                    stream.Close();
                    stream.Dispose();
                }
                catch (Exception exc)
                {
                    WF.MessageBox.Show("There was an error saving the world.Map." +
exc.ToString());
                }
            }
        }

        private void LoadWorld(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))
                    {
                        BinaryReader reader = new BinaryReader(cryptoStream);

                        world = World.Load(Content, reader);
                        lbWorld.Items.Clear();

                        foreach (TileMap m in world.Maps.Values)
                        {
                            lbWorld.Items.Add(m.MapName);
                        }

                        reader.Close();
                    }

                    stream.Close();
                    stream.Dispose();
                }
                catch (Exception exc)
                {
                    WF.MessageBox.Show("There was a problem loading the world.Map. " +
exc.ToString());
                }
            }
        }

        /// <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 (world != null && world.Maps.Count > 0)
            {
                world.Draw(gameTime, spriteBatch, camera, true);
            }

            spriteBatch.Begin();
            controls.Draw(spriteBatch);
```

```
            spriteBatch.End();

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

There is a new ListBox that will hold the maps in the world. The LoadContent method creates the list box. It is the height and width as the other list boxes and it position is on the right of the other list boxes. It also has TabStop, Enabled and HasFocus set to true. It then wires the SelectionChanged event. It also adds it to the ControlManager.

The SelectionChanged event handler for the lbWorld list box calls the ChangeMap method of the World class. It sets the Image property of pbTileset and pbPreview to the first texture in the tile set of the map. It clears the items in lbTileSets then adds the texture names back in. Next it sets the SelectedIndex of lbTileSets to the first item. The source rectangle for the pbPreview to be the first source rectangle of the tile set. The next thing that it does is set the source rectangle of pbTileset to be the entire texture of the first texture in the tile set.

Instead of using the map field the SelectionChanged event handler for lbTileSets uses the new world field.

In the Update method the code for creating a new map was changed. If OKPressed is true then the map is added to the Maps collection of the world field. Then ChangeMap is called passing in the name of the map. The name of the map is then added to lbWorlds. The camera is then set to the zero vector and locked to the map. Like in the SelectionChanged event handler for lbWorld the Image properties are set to the first texture of the tile set of the map. It also resets lbTileSets and the source rectangles of the picture boxes.

Where I used to check for map != null I check for world != null and world.Maps.Count > 0. The paint code now uses world.Map instead of map. When creating the list forms I pass in world.Map instead of map.

There is now a check to see if the W key was released and if frameCount is greater than five. There is a check to see if world is not null. If it is not I capture the result of a message box asking if the user wants to continue. If the result is No I exit the method. I then create a WorldForm and call the ShowDialog method to display it. If OKPressed is false I exit the method. The world field is then set to the World created by the form.

In the code for displaying the SaveFileDialog I update the filter to display World (*.wrld) and the extension to *.wrld. If the OK button was pressed on the dialog I call SaveWorld instead of SaveMap. The SaveMap method was removed.

In the code for displaying the LoadFileDialog I updated the filter so it would display World (*.wrld) and the extension *.wrld. If the OK button was press on the dialog I call LoadWorld instead of LoadMap. The LoadMap method was removed. It then resets the GUI to use the new map like before. The camera is also reset to Vector2.Zero and locked.

There is then a check to see if the map is not null and that there is a map on the world. If both are true I call the Update method of the world field.

HandleScrollMap now checks to see that world is null or there are no maps. If either is true it exits the method. There is a check to see if world is not null and that there is a map before calling LockCamera though it should never happen.

The SaveWorld method works the same as SaveMap method when it comes to creating and encrypting the file. The difference is that it now calls world.Save instead of map.Save.

The LoadWorld method works similarly to the LoadMap method. It opens the file and decrypts it like before. It calls World.Load instead of TileMap.Load to load the world. After loading the world I reset lbWorld and add the maps in the loaded world. It then resets the GUI the same way as the SelectionChanged event handler of lbWorld.

The Draw method check to see that world is not null and that the Count property of world.Maps is greater than zero. If that is true I call the Draw method of the world field.

I am 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!