XNA 4.0 RPG Tutorials

Part 18B

Finding Loot Part 2

I'm writing these tutorials for the new XNA 4.0 framework. The tutorials will make more sense if they are read in order. You can find the list of tutorials on the XNA 4.0 RPG tutorials page of my web site. I will be making my version of the project available for download at the end of each tutorial. It will be included on the page that links to the tutorials.

This is part B of tutorial 18. I was working on adding chests and keys to the editor and being able to read in chests at run time. At the end of the last tutorial I had add in new items to **FormMain** in the designer but I haven't added in the code to handle them. Right click **FormMain** in the editor and select **View Code**. This is the code for **FormMain**.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;

using RpgLibrary;
using RpgLibrary.CharacterClasses;
using RpgLibrary.ItemClasses;

namespace RpgEditor
{
    public partial class FormMain : Form
    {
        #region Field Region

        RolePlayingGame rolePlayingGame;
        FormClasses frmClasses;
        FormArmor frmArmor;
        FormShield frmShield;
        FormWeapon frmWeapon;
        FormKey frmKey;
        FormChest frmChest;

        static string gamePath = "";
        static string classPath = "";
        static string itemPath = "";
        static string chestPath = "";
        static string keyPath = "";

        #endregion

        #region Property Region

        public static string GamePath
        {
            get { return gamePath; }
        }
        public static string ClassPath
        {
            get { return classPath; }
        }


        public static string ItemPath
        {
            get { return itemPath; }
        }

        public static string ChestPath
        {
            get { return chestPath; }
        }

        public static string KeyPath
        {
            get { return keyPath; }
        }

        #endregion

        #region Constructor Region

        public FormMain()
        {
            InitializeComponent();

            this.FormClosing += new FormClosingEventHandler(FormMain_FormClosing);

            newGameToolStripMenuItem.Click += new EventHandler(newGameToolStripMenuItem_Click);
            openGameToolStripMenuItem.Click += new EventHandler(openGameToolStripMenuItem_Click);
            saveGameToolStripMenuItem.Click += new EventHandler(saveGameToolStripMenuItem_Click);
            exitToolStripMenuItem.Click += new EventHandler(exitToolStripMenuItem_Click);

            classesToolStripMenuItem.Click += new EventHandler(classesToolStripMenuItem_Click);
            armorToolStripMenuItem.Click += new EventHandler(armorToolStripMenuItem_Click);
            shieldToolStripMenuItem.Click += new EventHandler(shieldToolStripMenuItem_Click);
            weaponToolStripMenuItem.Click += new EventHandler(weaponToolStripMenuItem_Click);

            keysToolStripMenuItem.Click += new EventHandler(keysToolStripMenuItem_Click);
            chestsToolStripMenuItem.Click += new EventHandler(chestsToolStripMenuItem_Click);
        }

        #endregion

        #region Menu Item Event Handler Region

        void FormMain_FormClosing(object sender, FormClosingEventArgs e)
        {
            DialogResult result = MessageBox.Show(
                "Unsaved changes will be lost. Are you sure you want to exit?",
                "Exit?",
                MessageBoxButtons.YesNo,
                MessageBoxIcon.Warning);

            if (result == DialogResult.No)
                e.Cancel = true;
        }

        void newGameToolStripMenuItem_Click(object sender, EventArgs e)
        {
            using (FormNewGame frmNewGame = new FormNewGame())
            {
                DialogResult result = frmNewGame.ShowDialog();

                if (result == DialogResult.OK && frmNewGame.RolePlayingGame != null)
                {
                    FolderBrowserDialog folderDialog = new FolderBrowserDialog();
                    folderDialog.Description = "Select folder to create game in.";
                    folderDialog.SelectedPath = Application.StartupPath;

                    DialogResult folderResult = folderDialog.ShowDialog();

                    if (folderResult == DialogResult.OK)
                    {
                        try
                        {

                            gamePath = Path.Combine(folderDialog.SelectedPath, "Game");
                            classPath = Path.Combine(gamePath, "Classes");
                            itemPath = Path.Combine(gamePath, "Items");
                            keyPath = Path.Combine(gamePath, "Keys");
                            chestPath = Path.Combine(gamePath, "Chests");

                            if (Directory.Exists(gamePath))
                                throw new Exception("Selected directory already exists.");

                            Directory.CreateDirectory(gamePath);
                            Directory.CreateDirectory(classPath);
                            Directory.CreateDirectory(itemPath + @"\Armor");
                            Directory.CreateDirectory(itemPath + @"\Shield");
                            Directory.CreateDirectory(itemPath + @"\Weapon");
                            Directory.CreateDirectory(keyPath);
                            Directory.CreateDirectory(chestPath);

                            rolePlayingGame = frmNewGame.RolePlayingGame;
                            XnaSerializer.Serialize<RolePlayingGame>(gamePath + @"\Game.xml",
rolePlayingGame);
                        }
                        catch (Exception ex)
                        {
                            MessageBox.Show(ex.ToString());
                            return;
                        }

                        classesToolStripMenuItem.Enabled = true;
                        itemsToolStripMenuItem.Enabled = true;
                        keysToolStripMenuItem.Enabled = true;
                        chestsToolStripMenuItem.Enabled = true;
                    }
                }
            }
        }

        void openGameToolStripMenuItem_Click(object sender, EventArgs e)
        {
            FolderBrowserDialog folderDialog = new FolderBrowserDialog();

            folderDialog.Description = "Select Game folder";
            folderDialog.SelectedPath = Application.StartupPath;

            bool tryAgain = false;

            do
            {
                DialogResult folderResult = folderDialog.ShowDialog();
                DialogResult msgBoxResult;

                if (folderResult == DialogResult.OK)
                {
                    if (File.Exists(folderDialog.SelectedPath + @"\Game\Game.xml"))
                    {
                        try
                        {
                            OpenGame(folderDialog.SelectedPath);
                            tryAgain = false;
                        }
                        catch (Exception ex)
                        {
                            msgBoxResult = MessageBox.Show(
                                ex.ToString(),
                                "Error opening game.",
                                MessageBoxButtons.RetryCancel);

                            if (msgBoxResult == DialogResult.Cancel)
                                tryAgain = false;
                            else if (msgBoxResult == DialogResult.Retry)
                                tryAgain = true;
                        }
                    }
                    else
                    {

                        msgBoxResult = MessageBox.Show(
                            "Game not found, try again?",
                            "Game does not exist",
                            MessageBoxButtons.RetryCancel);

                        if (msgBoxResult == DialogResult.Cancel)
                            tryAgain = false;
                        else if (msgBoxResult == DialogResult.Retry)
                            tryAgain = true;
                    }
                }

            } while (tryAgain);
        }

        void saveGameToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (rolePlayingGame != null)
            {
                try
                {
                    XnaSerializer.Serialize<RolePlayingGame>(gamePath + @"\Game.xml",
rolePlayingGame);
                    FormDetails.WriteEntityData();
                    FormDetails.WriteItemData();
                    FormDetails.WriteChestData();
                    FormDetails.WriteKeyData();
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.ToString(), "Error saving game.");
                }
            }
        }

        void exitToolStripMenuItem_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        void classesToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (frmClasses == null)
            {
                frmClasses = new FormClasses();
                frmClasses.MdiParent = this;
            }

            frmClasses.Show();
            frmClasses.BringToFront();
        }

        void armorToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (frmArmor == null)
            {
                frmArmor = new FormArmor();
                frmArmor.MdiParent = this;
            }

            frmArmor.Show();
            frmArmor.BringToFront();
        }

        void shieldToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (frmShield == null)
            {
                frmShield = new FormShield();
                frmShield.MdiParent = this;
            }

            frmShield.Show();
            frmShield.BringToFront();
        }

        void weaponToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (frmWeapon == null)
            {
                frmWeapon = new FormWeapon();
                frmWeapon.MdiParent = this;
            }

            frmWeapon.Show();
            frmWeapon.BringToFront();
        }

        void chestsToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (frmChest == null)
            {
                frmChest = new FormChest();
                frmChest.MdiParent = this;
            }

            frmChest.Show();
            frmChest.BringToFront();
        }

        void keysToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (frmKey == null)
            {
                frmKey = new FormKey();
                frmKey.MdiParent = this;
            }

            frmKey.Show();
            frmKey.BringToFront();
        }

        #endregion

        #region Method Region

        private void OpenGame(string path)
        {
            gamePath = Path.Combine(path, "Game");
            classPath = Path.Combine(gamePath, "Classes");
            itemPath = Path.Combine(gamePath, "Items");
            keyPath = Path.Combine(gamePath, "Keys");
            chestPath = Path.Combine(gamePath, "Chests");
            if (!Directory.Exists(keyPath))
            {
                Directory.CreateDirectory(keyPath);
            }

            if (!Directory.Exists(chestPath))
            {
                Directory.CreateDirectory(chestPath);
            }

            rolePlayingGame = XnaSerializer.Deserialize<RolePlayingGame>(
                gamePath + @"\Game.xml");

            FormDetails.ReadEntityData();
            FormDetails.ReadItemData();
            FormDetails.ReadKeyData();
            FormDetails.ReadChestData();
            PrepareForms();
        }

        private void PrepareForms()
        {
            if (frmClasses == null)
            {
                frmClasses = new FormClasses();
                frmClasses.MdiParent = this;
            }

            frmClasses.FillListBox();

            if (frmArmor == null)
            {
                frmArmor = new FormArmor();
                frmArmor.MdiParent = this;
            }

            frmArmor.FillListBox();

            if (frmShield == null)
            {
                frmShield = new FormShield();
                frmShield.MdiParent = this;
            }

            frmShield.FillListBox();

            if (frmWeapon == null)
            {
                frmWeapon = new FormWeapon();
                frmWeapon.MdiParent = this;
            }

            frmWeapon.FillListBox();

            if (frmKey == null)
            {
                frmKey = new FormKey();
                frmKey.MdiParent = this;
            }

            frmKey.FillListBox();

            if (frmChest == null)
            {
                frmChest = new FormChest();
                frmChest.MdiParent = this;
            }

            frmChest.FillListBox();
            classesToolStripMenuItem.Enabled = true;
            itemsToolStripMenuItem.Enabled = true;
            keysToolStripMenuItem.Enabled = true;
            chestsToolStripMenuItem.Enabled = true;
        }

        #endregion
    }
}
```

Quite a few new additions to the code of **FormMain**. I added in fields for **FormKey** and **FormChest**. I also static fields for the path to keys and chests. The fields are **keyPath** and **chestPath**. The properties are **KeyPath** and **ChestPath**. The properties, like the other static properties, are read only.

What is new in the constructor is that I wire event handlers for **Click** events for the two new menu items, like the other menu items. In the event handler for the new game I create paths for **keyPath** and **chestPath** like the other paths. Keys and chests reside in their own directories. I also create the new directories like the other paths. I also enable the two new menu items. The event handler for the **Click** event for saving a game calls two static methods that I haven't written yet on **FormDetails** called **WriteChestData** and **WriteKeyData**. As you can guess they will write out chest and key data.

In the event handlers for the **Click** event of the new menu items I first check to see if the form for that menu item is null. If it is null then I create a new form and set its **MdiParent** property to be the current form. I then call the **Show** and **BringToFront** methods to display the forms and bring them to the front.

Since the directory structure for a game has changed in the **OpenGame** method I check to see if the directories pointed to by **keyPath** and **chestPath** don't exist. If they don't exists then I create them. I then call the **ReadKeyData** and **ReadChestData** methods of **FormDetails** that I also haven't written yet and will get to soon.

The last change is in the **PrepareForms** method. I check to see if the forms are null. If they are null I create new instances and set their **MdiParent** properties to **this**. I also call their **FillListBox** methods to fill them with the appropriate items.

I need to update the code for **FormDetails**. I have to add the write and read methods for writing and reading chests and keys. Right click **FormDetails** and select **View Code** to view the code. Add the following methods.

```
public static void WriteKeyData()
{
    foreach (string s in ItemManager.KeyData.Keys)
    {
        XnaSerializer.Serialize<KeyData>(
            FormMain.KeyPath + @"\" + s + ".xml",
            ItemManager.KeyData[s]);
    }
}

public static void WriteChestData()
{
    foreach (string s in ItemManager.ChestData.Keys)
    {
        XnaSerializer.Serialize<ChestData>(
            FormMain.ChestPath + @"\" + s + ".xml",
            ItemManager.ChestData[s]);
    }
}
```

```
public static void ReadKeyData()
{
    string[] fileNames = Directory.GetFiles(FormMain.KeyPath, "*.xml");

    foreach (string s in fileNames)
    {
        KeyData keyData = XnaSerializer.Deserialize<KeyData>(s);
        itemManager.KeyData.Add(keyData.Name, keyData);
    }
}

public static void ReadChestData()
{
    string[] fileNames = Directory.GetFiles(FormMain.ChestPath, "*.xml");

    foreach (string s in fileNames)
    {
        ChestData chestData = XnaSerializer.Deserialize<ChestData>(s);
        itemManager.ChestData.Add(chestData.Name, chestData);
    }
}
```

The code is similar to the other writing and reading code. For writing in a foreach loop I loop through all of the keys in the **ItemManager** for that type of data. I then call the **Serialize** method of the **XnaSerializer** class with the right data type, path to the file, and the actual item to write. To create the path you take the path to that type of object and append a \ the name of the item and .xml for the extension. To read in the items you first need to get all of the items of that type in the directory. You use the **GetFiles** method of the **Directory** class to call all of the files with an xml extension. In a foreach loop I loop through all of the file names. I then call the **Deserialize** method of the **XnaSerilializer** class with the proper data type and the file name storing the object in a variable. I then add the object to the **ItemManager** passing in the name of the object for the key and the actual object for the value.

Right click the **RpgEditor** and select **Set As StartUp Project** if the editor isn't already the start up project. Build and run to launch the editor. Once you've done that select the **Open** menu item and navigate to your game data from tutorial 14. I'm going to add a couple keys and chests to the editor. Add the following keys and chests then save the game.

Keys

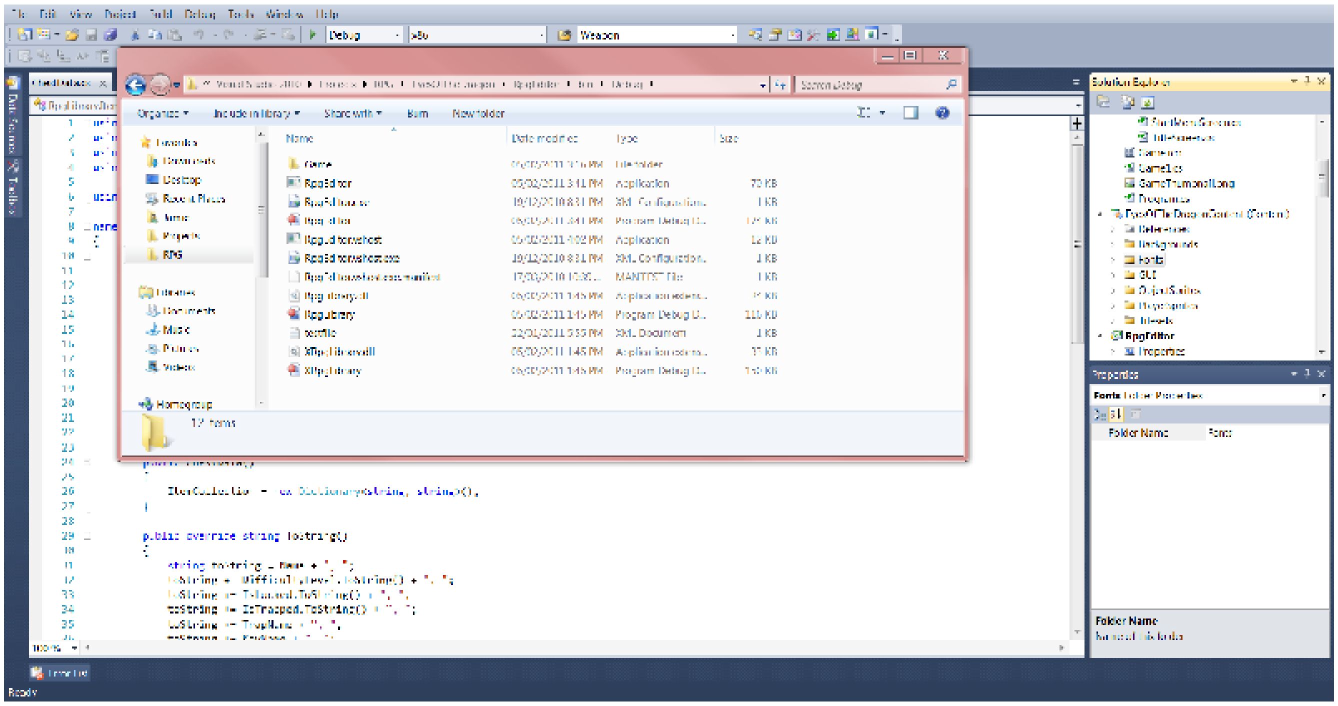
Key Name	Key Type
Rusty Key	
Golden Key	Golden Key

Chests

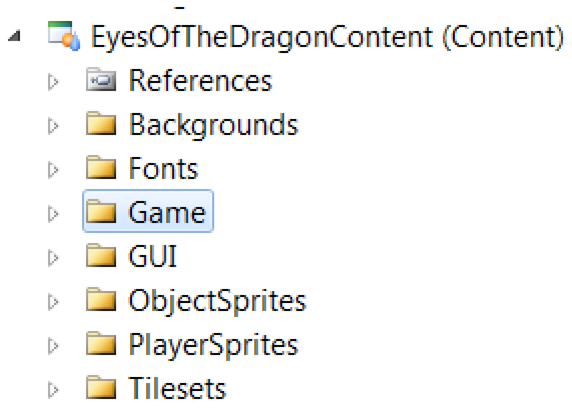
Name	Difficulty Level	Locked	Key Name	Key Type	Keys Required	Trapped	Trap Name	Min Gold	Max Gold	Item Collection
Rusty Chest	Normal	Yes	Rusty Key		1	No		100	150	
Gold Chest	Normal	Yes	Golden Key	Golden Key	1	No		200	250	
Big Gold Chest	Normal	Yes	Golden Key	Golden Key	2	No		500	750	
Plain Chest	Easy	Yes			0	No		10	50	
Broken Crate	Normal	No			0	No		5	10	

The next step will be to get a chest we created with the editor into the game. The first step will be to add the game data to the **EyesOfTheDragonContent** project. The easiest way will be to drag the

Game folder from windows explorer onto the **EyesOfTheDragonContent** project. Open a windows explorer window so that Visual Studio is visible below it and browse to where your **Game** directory can be found as seen below. Sorry it isn't all that clear. Now, drag the **Game** folder onto the **EyesOfTheDragonContent** project.



You should now have a **Game** folder in your **EyesOfTheDragonContent** project as seen in the next image. Right click the **EyesOfTheDragon** project and select **Set As StartUp Project** to make your game the start up project. You will need to add a reference to the **EyesOfTheDragonContent** project for your **RpgLibrary** to compile the XML files into XNB files that can be read using the **Content Pipeline**. Right click the **EyesOfTheDragonContent** project and select **Add Reference**. From the **Projects** tab select **RpgLibrary**.



EyesOfTheDragonContent (Content)
References
Backgrounds
Fonts
Game
GUI
ObjectSprites
PlayerSprites
Tilesets

The last thing I want to do is show how easy it is to read in a **ChestData** object using the **Content Pipeline**. Change the **CreateWorld** method of the **CharacterGeneratorScreen** to the following.

```
private void CreateWorld()
{
    Texture2D tilesetTexture = Game.Content.Load<Texture2D>(@"Tilesets\tileset1");
    Tileset tileset1 = new Tileset(tilesetTexture, 8, 8, 32, 32);

    tilesetTexture = Game.Content.Load<Texture2D>(@"Tilesets\tileset2");
    Tileset tileset2 = new Tileset(tilesetTexture, 8, 8, 32, 32);

    List<Tileset> tilesets = new List<Tileset>();
    tilesets.Add(tileset1);
    tilesets.Add(tileset2);

    MapLayer layer = new MapLayer(100, 100);

    for (int y = 0; y < layer.Height; y++)
    {
        for (int x = 0; x < layer.Width; x++)
        {
            Tile tile = new Tile(0, 0);

            layer.SetTile(x, y, tile);
        }
    }

    MapLayer splatter = new MapLayer(100, 100);

    Random random = new Random();

    for (int i = 0; i < 100; i++)
    {
        int x = random.Next(0, 100);
        int y = random.Next(0, 100);
        int index = random.Next(2, 14);

        Tile tile = new Tile(index, 0);
        splatter.SetTile(x, y, tile);
    }

    splatter.SetTile(1, 0, new Tile(0, 1));
    splatter.SetTile(2, 0, new Tile(2, 1));
    splatter.SetTile(3, 0, new Tile(0, 1));

    List<MapLayer> mapLayers = new List<MapLayer>();
    mapLayers.Add(layer);
    mapLayers.Add(splatter);

    TileMap map = new TileMap(tilesets, mapLayers);
    Level level = new Level(map);

    ChestData chestData = Game.Content.Load<ChestData>(@"Game\Chests\Plain Chest");

    Chest chest = new Chest(chestData);

    BaseSprite chestSprite = new BaseSprite(
        containers,
        new Rectangle(0, 0, 32, 32),
        new Point(10, 10));

    ItemSprite itemSprite = new ItemSprite(
        chest,
        chestSprite);
    level.Chests.Add(itemSprite);

    World world = new World(GameRef, GameRef.ScreenRectangle);
    world.Levels.Add(level);
    world.CurrentLevel = 0;

    GamePlayScreen.World = world;
}
```

As you can see I replaced the code where I created a **ChestData** object by hand with a call to **Load** of the **ContentManager** of our **Game** class. I passed in the **Plain Chest** that I created in the editor. We went through a little work to get there but it is well worth it in the end. Most classes that you create can be serialized using the **IntermediateSerializer** and then read in using the **Content Pipeline** at run time with out having to create a custom **Content Pipeline** extension.

I'm going to end this tutorial here. The plan was to show how easy it is to read in our custom content at run time using the **Content Pipeline**. I encourage you to visit the news page of my site, XNA Game Programming Adventures, for the latest news on my tutorials.

Good luck in your game programming adventures!

Jamie McMahon