XNA 4.0 RPG Tutorials

Part 19

Skills Continued

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.

In tutorial 15 I added in some place holder classes for skills, talents, and spells. In this tutorial I'm going to flesh out skills more. As I mentioned in tutorial 15 skills can be learned by any intelligent creature. The way I'm handling skills is that skills have a rank associated with them between 1 and 100. A rank of 0 means that the character has no knowledge of that skill. A rank of 100 means the character is master at that skill. When a character is first created they get 25 points to spend on skills. Every time a character levels up they get 10 additional points to spend on skills. There is also a difficulty involved when a character tries to apply a skill. It can be very easy to use a skill or hard even for a master. To successfully use a skill a random number between 1 and 100 is generated. If the character's rank in the skill plus the difficulty plus any modifiers is greater than or equal to the random number the character is successful in using the skill. Skills will have modifiers based on a primary attribute, like strength, and the character's class. A thief will have a better understanding of poisons than a fighter.

So, what exactly are the skills I'm planning on implementing? I'm going to implement some crafting skills. Crafting skills require a recipe to use and the reagents required by the recipe. It is the recipes that have a difficulty associated with them. Making a simple health potion wouldn't be too difficult but creating a deadly poison would be quite challenging. I'm going to be adding a bartering skill as well that will help the character deal with trading. Be careful using it though. If you upset the merchant you could get tossed out of their shop! I will add a few others as well.

The first step will be to flesh out the **Skill** and **SkillData** classes. Like the other **Data** classes **SkillData** will define a basic skill, with out any modifiers. The **Skill** class will be created and be specific to a character. Change the **SkillData** class to the following.

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

namespace RpgLibrary.SkillClasses
{
    public class SkillData
    {
        #region Field Region

        public string Name;
        public string PrimaryAttribute;
        public Dictionary<string, int> ClassModifiers;

        #endregion
        #region Constructor Region

        public SkillData()
        {
            ClassModifiers = new Dictionary<string, int>();
        }

        #endregion

        #region Virtual Method Region

        public override string ToString()
        {
            string toString = Name + ", ";
            toString += PrimaryAttribute;

            foreach (string s in ClassModifiers.Keys)
                toString += ", " + s + "+" + ClassModifiers[s].ToString();

            return toString;
        }

        #endregion
    }
}
```

There are three fields in the class. There is **Name** for the name of the skill. **PrimaryAttribute** is the primary attribute that affects the skill. **ClassModifiers** is a **Dictionary<string, int>**. The **string** is the name of the class and the **int** is the amount of the modifier. The **IntermediateSerializer** class does allow for **Dictionary** fields to be serialized and deserialzed so there is no worries about that. I included an override of the **ToString** method, for use in the editor.

Speaking of the editor now would be a good time to add skills to the editor. For the next little bit you will want the editor to be the start up project. Right click the **RpgEditor** in the solution explorer and select **Set As StartUp Project**. You are going to want to add two forms to the **RpgEditor**. One to hold all of the skills in the game and one creating and editing skills. Right click the **RpgEditor** in the solution explorer, select **Add** and then **Windows Form**. Name the new form **FormSkill**.

First, set the **Size** property of **FormSkill** to be the **Size** property of **FormDetails**. Set the **Text** property to be **Skills** and the **MinimizeBox** property to **False**. Right click **FormSkill** in the solution explorer and select **View Code**. Modify the code of **FormSkill** to inherit from **FormDetails**.

```
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;

namespace RpgEditor
{
    public partial class FormSkill : FormDetails
    {
        public FormSkill()
        {
            InitializeComponent();
        }
    }
}
```

Before I get to the form for creating and editing a skill you will want to update **FormDetails** to include a **SkillDataManager** field. You will also want to add methods to read and write skills. Right click **FormDetails** in the solution explorer and select **View Code**. First, add this field to the **Field Region** and property to the **Property Region**. You will also want a using statement for the **SkillClasses** name space in the **RpgLibrary**.

```
using RpgLibrary.SkillClasses;

protected static SkillDataManager skillManager;

public static SkillDataManager SkillManager
{
    get { return skillManager; }
}
```

To the **Method Region** you will want to add these methods to write and read the skill data.

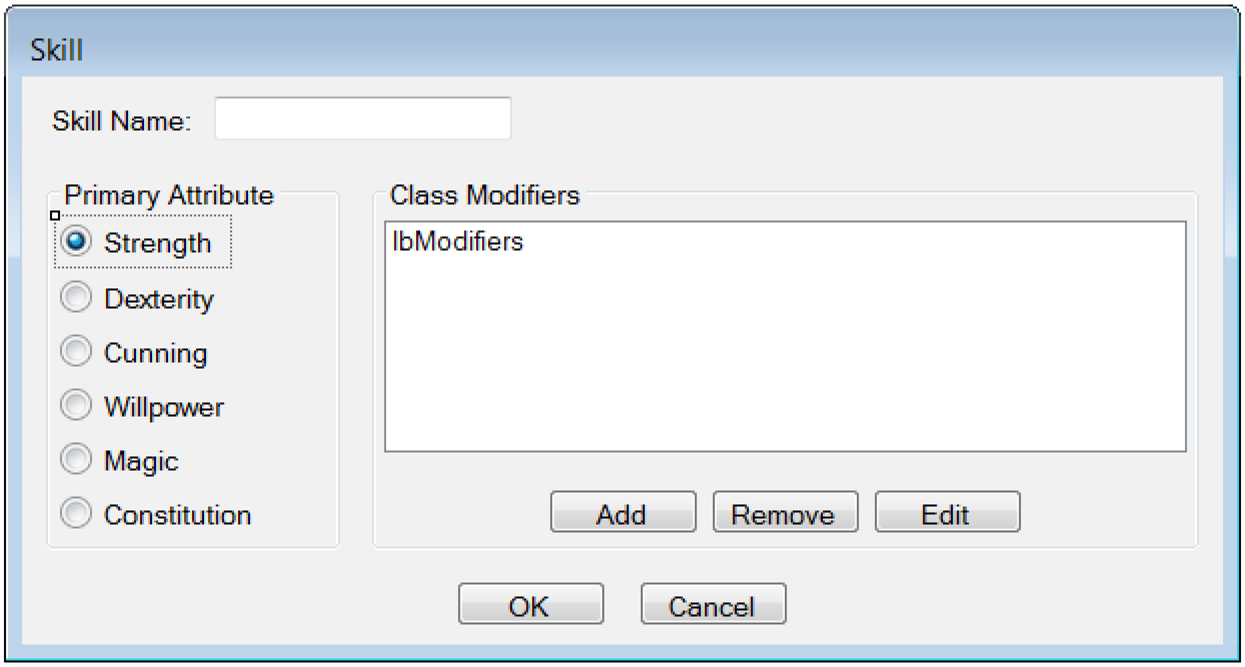
```
public static void WriteSkillData()
{
    foreach (string s in SkillManager.SkillData.Keys)
    {
        XnaSerializer.Serialize<SkillData>(
            FormMain.SkillPath + @"\" + s + ".xml",
            SkillManager.SkillData[s]);
    }
}

public static void ReadSkillData()
{
    skillManager = new SkillDataManager();

    string[] fileNames = Directory.GetFiles(FormMain.SkillPath, "*.xml");

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

Visual Studio is going to give you two errors, that **FormMain.SkillPath** does not exist. Let's fix that in **FormMain**. Right click **FormMain** in the solution explorer and select **View Code**. First, you are going to want a field for **FormSkill** and a field for the path to write skills. You will also want to add a property to expose the path for skills. Change the **Field** and **Property** regions to the following.

```
#region Field Region

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

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

```
#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; }
}

public static string SkillPath
{
    get { return skillPath; }
}

#endregion
```

The next step is going to be designing the form for creating and editing the individual skills. Right click **RpgEditor**, select **Add** and then **Windows Form**. Name this new form **FormSkillDetails**. My form in the solution explorer appears next.



First, you will want to make the form bigger to house all of the controls. You can also set the **ControlBox** property to **False**, the **StartPosition** to **CenterParent**, and the **Text** property to **Skill**. First, drag a **Label** onto the form and set its **Text** property to **Skill Name:**. Drag a **Text Box** beside the **Label** and set its **Name** property to **tbName**. Drag a **Group Box** onto the form next and change the size. You can set its **Text** property to **Primary Attribute**. Onto the **Group Box** you will want to drag on six **Radio Buttons**. For the first one set the **Name** property to **rbStrength** and the **Text** property to **Strength**. Also, set the **Checked** property of the **Radio Button** to **True**. For the other **Radio Buttons** set their **Name** and **Text** properties to the following, in order: **rbDexterity** and **Dexterity**, **rbCunning** and **Cunning**, **rbWillpower** and **Willpower**, **rbMagic** and **Magic**, and **rbConstitution** and **Constitution**. I then dragged another **Group Box** onto the form and made it larger. I also set the **Text** property of the **Group Box** to **Class Modifiers**. I dragged a **List Box** onto the **Group Box** and set its **Name** property to **lbModifiers**. Under the **List Box**, but still on the **Group Box**, I dragged three **Buttons**. I set their **Name** and **Text** properties to the following, again in order: **btnAdd** and **Add**, **btnRemove** and **Remove**, and **btnEdit** and **Edit**. Under the **Group Boxes** I dragged on two more **Buttons**. The first one I set **Name** to **btnOK** and **Text** to **OK**. The second I set **Name** to **btnCancel** and **Text** to **Cancel**.

I'm going to add in some of the logic to the form now. Right click **FormSkillDetails** in the solution explorer and select **View Code**. The code for **FormSkillDetails** follows next.

```
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 RpgLibrary.SkillClasses;

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

        SkillData skill;

        #endregion

        #region Property Region

        public SkillData Skill
        {
            get { return skill; }
            set { skill = value; }
        }

        #endregion

        #region Constructor Region

        public FormSkillDetails()
        {
            InitializeComponent();

            this.Load += new EventHandler(FormSkillDetails_Load);
            this.FormClosing += new FormClosingEventHandler(FormSkillDetails_FormClosing);

            btnOK.Click += new EventHandler(btnOK_Click);
            btnCancel.Click += new EventHandler(btnCancel_Click);
        }

        #endregion

        #region Form Event Handler Region

        void FormSkillDetails_Load(object sender, EventArgs e)
        {
            if (Skill != null)
            {
                tbName.Text = Skill.Name;
                switch (Skill.PrimaryAttribute.ToLower())
                {
                    case "strength":
                        rbStrength.Checked = true;
                        break;
                    case "dexterity":
                        rbDexterity.Checked = true;
                        break;
                    case "cunning":
                        rbCunning.Checked = true;
                        break;
                    case "willpower":
                        rbWillpower.Checked = true;
                        break;
                    case "magic":
                        rbMagic.Checked = true;
                        break;
                    case "constitution":
                        rbConstitution.Checked = true;
                        break;
                }

                foreach (string s in Skill.ClassModifiers.Keys)
                {
                    string data = s + ", " + Skill.ClassModifiers[s].ToString();
                    lbModifiers.Items.Add(data);
                }
            }
        }

        void FormSkillDetails_FormClosing(object sender, FormClosingEventArgs e)
        {
            if (e.CloseReason == CloseReason.UserClosing)
            {
                e.Cancel = true;
            }
        }

        #endregion

        #region Button Event Handler Region

        void btnOK_Click(object sender, EventArgs e)
        {
            if (string.IsNullOrEmpty(tbName.Text))
            {
                MessageBox.Show("You must provide a name for the skill.");
                return;
            }

            SkillData newSkill = new SkillData();

            newSkill.Name = tbName.Text;

            if (rbStrength.Checked)
                newSkill.PrimaryAttribute = "Strength";
            else if (rbDexterity.Checked)
                newSkill.PrimaryAttribute = "Dexterity";
            else if (rbCunning.Checked)
```

```
                newSkill.PrimaryAttribute = "Cunning";
            else if (rbWillpower.Checked)
                newSkill.PrimaryAttribute = "Willpower";
            else if (rbMagic.Checked)
                newSkill.PrimaryAttribute = "Magic";
            else if (rbConstitution.Checked)
                newSkill.PrimaryAttribute = "Constitution";

            skill = newSkill;
            this.FormClosing -= FormSkillDetails_FormClosing;
            this.Close();
        }

        void btnCancel_Click(object sender, EventArgs e)
        {
            skill = null;
            this.FormClosing -= FormSkillDetails_FormClosing;
            this.Close();
        }

        #endregion
    }
}
```

There is a lot that you've seen here before. There is a using statement to bring the **SkillClasses** classes into scope. There is a field of type **SkillData** and a property to expose the field. The constructor wires event handlers for the **Load** and **FormClosing** events of the form. It also wires handlers for **btnOK** and **btnCancel**. I didn't wire the handlers for the other buttons. That I will do in a future tutorial. The **Load** event handler checks to see if the **Skill** property is not null. If it isn't I set the **Text** property of **tbName** to the **Name** field. There is a switch on the the **PrimaryAttribute** field of the skill converted to a lower case string. The cases check for each of the primary attributes. If that value is found I set the **Checked** property of the appropriate **Radio Button** to **True**. In a foreach loop I loop over all of the keys in the **ClassModifiers** dictionary. I create a string consisting of the class name, a comma, and the value of the modifier as a string. I then add the string to the **Items** of **lbModifier**. There is nothing you haven't seen in the **FormClosing** event handler. The **Click** handler of **btnOK** does a little validation on the form. If the **Text** property of **tbName** is null or empty I show a message box stating it must have a value and break out of the method. I create a new instance of **SkillData** and set the **Name** field to be the **Text** property of **tbName**. For determining what the primary attribute should be there is a series of if-else-it statements that check the **Checked** property of the radio buttons. I set the **PrimaryAttribute** field base on which radio button was checked. I then set the **skill** field and then unsubscribe from the **FormClosing** event. Finally the form closes. I believe the **Click** event handler for **btnCancel** doesn't need to be explained either.

I'm going to add in the code for dealing with **FormSkill** now. Right click **FormSkill** in the solution explorer and select **View Code**. The code for that form follows next.

```
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.SkillClasses;

namespace RpgEditor
{
    public partial class FormSkill : FormDetails
    {

        #region Constructor Region

        public FormSkill()
        {
            InitializeComponent();

            btnAdd.Click += new EventHandler(btnAdd_Click);
            btnEdit.Click += new EventHandler(btnEdit_Click);
            btnDelete.Click += new EventHandler(btnDelete_Click);
        }

        #endregion

        #region Button Event Handler Region


        void btnAdd_Click(object sender, EventArgs e)
        {
            using (FormSkillDetails frmSkillDetails = new FormSkillDetails())
            {
                frmSkillDetails.ShowDialog();

                if (frmSkillDetails.Skill != null)
                {
                    AddSkill(frmSkillDetails.Skill);
                }
            }
        }

        void btnEdit_Click(object sender, EventArgs e)
        {
            if (lbDetails.SelectedItem != null)
            {
                string detail = lbDetails.SelectedItem.ToString();
                string[] parts = detail.Split(',');
                string entity = parts[0].Trim();

                SkillData data = skillManager.SkillData[entity];
                SkillData newData = null;

                using (FormSkillDetails frmSkillData = new FormSkillDetails())
                {
                    frmSkillData.Skill = data;
                    frmSkillData.ShowDialog();

                    if (frmSkillData.Skill == null)
                        return;

                    if (frmSkillData.Skill.Name == entity)
                    {
                        skillManager.SkillData[entity] = frmSkillData.Skill;
                        FillListBox();
                        return;
                    }

                    newData = frmSkillData.Skill;
                }

                DialogResult result = MessageBox.Show(
                    "Name has changed. Do you want to add a new entry?",
                    "New Entry",
                    MessageBoxButtons.YesNo);

                if (result == DialogResult.No)
                    return;

                if (skillManager.SkillData.ContainsKey(newData.Name))
                {
                    MessageBox.Show("Entry already exists. Use Edit to modify the entry.");
                    return;
                }

                lbDetails.Items.Add(newData);
                skillManager.SkillData.Add(newData.Name, newData);
            }
        }

        void btnDelete_Click(object sender, EventArgs e)
        {
            if (lbDetails.SelectedItem != null)
            {
                string detail = (string)lbDetails.SelectedItem;
                string[] parts = detail.Split(',');
                string entity = parts[0].Trim();

                DialogResult result = MessageBox.Show(
                    "Are you sure you want to delete " + entity + "?",
                    "Delete",
                    MessageBoxButtons.YesNo);

                if (result == DialogResult.Yes)
                {
                    lbDetails.Items.RemoveAt(lbDetails.SelectedIndex);
                    skillManager.SkillData.Remove(entity);

                    if (File.Exists(FormMain.SkillPath + @"\" + entity + ".xml"))
                        File.Delete(FormMain.SkillPath + @"\" + entity + ".xml");
                }
            }
        }

        #endregion

        #region Method Region

        public void FillListBox()
        {
            lbDetails.Items.Clear();

            foreach (string s in FormDetails.SkillManager.SkillData.Keys)
                lbDetails.Items.Add(FormDetails.SkillManager.SkillData[s]);
        }

        private void AddSkill(SkillData skillData)
        {
            if (FormDetails.SkillManager.SkillData.ContainsKey(skillData.Name))
            {
                DialogResult result = MessageBox.Show(
                    skillData.Name + " already exists. Overwrite it?",
                    "Existing armor",
                    MessageBoxButtons.YesNo);

                if (result == DialogResult.No)
                    return;

                skillManager.SkillData[skillData.Name] = skillData;
                FillListBox();
                return;
            }

            skillManager.SkillData.Add(skillData.Name, skillData);
            lbDetails.Items.Add(skillData);
        }

        #endregion
    }
}
```

You may be getting tired of seeing this code over and over again. It is a lot like the code for the other forms that house all of a specific type of data. The main difference is that instead of working with items this form works with skills. The basics were to add using statements to bring some classes into scope for the form. I then wired the event handlers for the buttons.

In the **Click** event of the **Add** button I create a new **FormSkillDetails** in a using statement so it will be disposed of when I'm done with it. I then display the form using **ShowDialog** so the user must finish with the form before working on the editor again. If the **Skill** property of the form is not null I call the **AddSkill** method.

The **Click** event of the **Edit** button checks to see if the **SelectedItem** property of **lbDetails** is not null. It then splits the item into the different parts to get the **Name** of the skill. I then get the skill from the **SkillDataManager**. I also have another local variable of **SkillData** that is set to null. I create a new form of type **FormSkillDetails** in a using statement so it will be disposed of when I close the form. I then set the **Skill** property of form to be the skill retrieved from the **SkillDataManager**. It is is null the user canceled the edit and I break out of the method. If the **Name** of the skill on the form is the same as the name of the skill from the list box I update the skill in the skill data manager and call the **FillListBox** method to update it and return out of the method. Otherwise I set the **newData** local variable to be the **Skill** property of the form. I display a message box stating that the name of the skill has changed and if the user wants to add a new skill with that name. If the response is no I break out of the method. If there is already a skill with the new name I display a message box and break out of the method. Otherwise I add the new skill to **lbDetails** and **skillManager**.

The **Click** event of the **Delete** button checks to see if the **SelectedItem** property of **lbDetails** is not null like before. I get the name of the selected item as before and display a message box asking if the entry should be deleted. If the answer was yes I remove the entry from the list box and from the skill data manager class. I also check to see if a file exists in the directory containing skills. If it does exist I delete it. I will point out I missed something in tutorial 18. I neglected to update the code for deleting items in **FormKey** and **FormChest**. I will update that code shortly.

The **FillListBox** method works like previous versions of the **FillListBox** method. It just works with **SkillData** rather than other data classes. Same with the **AddSkill** method. It just works with skills rather than other data classes.

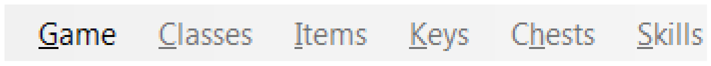
For **FormKey** you want to check the directory for keys, not a directory inside the directory for items. Change the **btnDelete_Click** method of **FormKey** to the following.

```
void btnDelete_Click(object sender, EventArgs e)
{
    if (lbDetails.SelectedItem != null)
    {
        string detail = (string)lbDetails.SelectedItem;
        string[] parts = detail.Split(',');
        string entity = parts[0].Trim();

        DialogResult result = MessageBox.Show(
            "Are you sure you want to delete " + entity + "?",
            "Delete",
            MessageBoxButtons.YesNo);

        if (result == DialogResult.Yes)
        {
            lbDetails.Items.RemoveAt(lbDetails.SelectedIndex);
            itemManager.KeyData.Remove(entity);
            if (File.Exists(FormMain.KeyPath + @"\" + entity + ".xml"))
                File.Delete(FormMain.KeyPath + @"\" + entity + ".xml");
        }
    }
}
```

The same thing is true for **FormChest**. Rather than keys you want to work with chests. Change the **btnDelete_Click** method in **FormChest** to the following.

```
void btnDelete_Click(object sender, EventArgs e)
{
    if (lbDetails.SelectedItem != null)
    {
        string detail = (string)lbDetails.SelectedItem;
        string[] parts = detail.Split(',');
        string entity = parts[0].Trim();

        DialogResult result = MessageBox.Show(
            "Are you sure you want to delete " + entity + "?",
            "Delete",
            MessageBoxButtons.YesNo);

        if (result == DialogResult.Yes)
        {
            lbDetails.Items.RemoveAt(lbDetails.SelectedIndex);
            itemManager.ChestData.Remove(entity);

            if (File.Exists(FormMain.ChestPath + @"\" + entity + ".xml"))
                File.Delete(FormMain.ChestPath + @"\" + entity + ".xml");
        }
    }
}
```

The next thing to do is to update the main form to work with skills. You will want to add a new menu item for skills. Right click **FormMain** and select **View Designer** to bring up the design view. Like you did in the last tutorial you want to add a new menu item. Beside the **Chests** item add a new item **&Skills**. Set the **Enabled** property to be **False** as well. Your menu bar should now look like this.



You can close the design view and go back to the code view. If you closed the code view before open it again by right clicking **FormMain** and selecting **View Code**. You will want to add in an event handler for the new menu item. Change the constructor of **FormMain** to the following. Add the following method just below the other handlers for menu item clicks.

```
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);

    skillsToolStripMenuItem.Click += new EventHandler(skillsToolStripMenuItem_Click);
}

void skillsToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (frmSkill == null)
    {
        frmSkill = new FormSkill();
        frmSkill.MdiParent = this;
    }

    frmSkill.Show();
    frmSkill.BringToFront();
}
```

The handler works like the other handlers. Check to see if the form is null. If it is null create one and set the **MdiParent** to be **this**. Then you call the **Show** method to show the form and the **BringToFront** method to make it the top form.

You are going to want to update the code for creating a new game, saving a game, and opening a game. I will start with creating a new game. Change the code for the click event handler of the new game menu item to the following.

```
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");
                    skillPath = Path.Combine(gamePath, "Skills");

                    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);
                    Directory.CreateDirectory(skillPath);

                    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;
                skillsToolStripMenuItem.Enabled = true;
            }
        }
    }
}
```

The changes were that I make a new path like the other paths. I create a directory like the others as well. I also set the **Enabled** property of the menu item to **true** as well.

For saving a game you will want to write out the **SkillData** in the **SkillDataManager**. All you need to do is to call the static method **WriteSkillData** of **FormDetails**. Change the click event handler for the save menu item to the following.

```
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();
            FormDetails.WriteSkillData();
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.ToString(), "Error saving game.");
        }
    }
}
```

There are two parts to opening games. You will want to change the **OpenGame** and **PrepareForms** methods. You can change those methods to the following.

```
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");
    skillPath = Path.Combine(gamePath, "Skills");

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

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

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

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

    FormDetails.ReadEntityData();
    FormDetails.ReadItemData();
    FormDetails.ReadKeyData();
    FormDetails.ReadChestData();
    FormDetails.ReadSkillData();

    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();
    if (frmSkill == null)
    {
        frmSkill = new FormSkill();
        frmSkill.MdiParent = this;
    }

    frmSkill.FillListBox();

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

So, what are the changes to **OpenGame**. The first is you want to create a path for skills. Like for chests and keys and since skills are new I check to see if a directory for skills exists. If it doesn't I create a directory for them. I then call the **ReadSkillData** method of **FormDetails** to read in the skills.

The change to **PrepareForms** is I check to see if **frmSkills** is null. If it is I create a new form and set the **MdiParent** to be **this** the current form. I then call the **FillListBox** method of **FormSkill** to fill it with items. I set the **Enabled** property of the menu item to **True** as well.

So, build and run the editor. Choose to open a game from the **Game** menu and navigate to the **EyesOfTheDragonContent** folder where you added the **Game** to last time. Bring up the **Skills** form and add the following skills.

Skill Name	Primary Attribute
Bartering	Cunning
Herbalism	Magic
Poison Making	Cunning
Trap Making	Dexterity

The first skill, **Bartering**, is influenced by a character's **Cunning**. If you recall **Cunning** measures a character's ability to read a situation, pick up on subtleties and reason. **Bartering** can improve interaction with merchants. They will give you a lower price if you successfully barter with them when you are buying something for example. If you fail drastically they might throw you out of their store! **Herbalism** is combining ingredients to make magical potions and is influenced by a character's **Magic** attribute. The higher the character's magic the better quality the potion will be. **Poison Making** allows the player to create poisons. They are more mundane than potions so it required great cunning when crafting a poison. **Trap Making** allows the character to craft traps. It takes a lot of dexterity to keep a trap from going off while you are crafting it.

I'm going to end this tutorial here. The plan was to add in more to dealing with skills in the game. 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