XNA 4.0 RPG Tutorials

Part 29

Resistances and Weaknesses

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 going to be a rather short tutorial where I cover adding in resistances and weaknesses to different types of damage. The first thing I did was update the **DamageType** enumeration in the **DamageEffectData** class. Update the **DamageType** enumeration to the following.

```
public enum DamageType { Weapon, Poison, Disease, Fire, Earth, Water, Air }
```

A bit of a change. I replaced **Crushing**, **Piercing** and **Slashing** with one value, **Weapon**, that represents damage done with a weapon. I did that because I'd have to go back and make a lot of modifications to the item classes and editors for weapons and armor to have different resistances and weaknesses. I just didn't want to make that many changes. You could modify weapons to do different types of damage and armors have different strengths and weaknesses against different damage types. I also renamed **Ice** to **Water** and **Lightning** to **Air** just to be a little more generic.

I also want to add in four classes. Two classes are for the weaknesses that a character has and the other two for the resistances that a character has. Right click the **EffectClasses** folder, select **Add** and then **Class**. Name the new class **WeaknessData**. Repeat that procedure three more times and name the new classes **Weakness**, **ResistanceData** and **Resistance**. The code for those classes is next.

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

namespace RpgLibrary.EffectClasses
{
    public class WeaknessData
    {
        #region Field Region

        public DamageType WeaknessType;
        public int Amount;

        #endregion

        #region Property Region
        #endregion

        #region Constructor Region
        #endregion

        #region Method Region
        #endregion
```

```
        #region Virtual Method Region
        #endregion
    }
}
```

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

namespace RpgLibrary.EffectClasses
{
    public class Weakness
    {
        #region Field Region

        DamageType weakness;
        int amount;

        #endregion

        #region Property Region

        public DamageType WeaknessType
        {
            get { return weakness; }
            private set { weakness = value; }
        }

        public int Amount
        {
            get { return amount; }
            private set
            {
                if (value < 0)
                    amount = 0;
                else if (value > 100)
                    amount = 100;
                else
                    amount = value;
            }
        }

        #endregion

        #region Constructor Region

        public Weakness(WeaknessData data)
        {
            WeaknessType = data.WeaknessType;
            Amount = data.Amount;
        }

        #endregion

        #region Method Region

        public int Apply(int damage)
        {
            return (damage + damage * amount / 100);
        }

        #endregion

        #region Virtual Method Region
        #endregion
    }
}
```

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

namespace RpgLibrary.EffectClasses
{
    public class ResistanceData
    {
        #region Field Region

        public DamageType ResistanceType;
        public int Amount;

        #endregion

        #region Property Region
        #endregion

        #region Constructor Region
        #endregion

        #region Method Region
        #endregion

        #region Virtual Method Region
        #endregion
    }
}
```

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

namespace RpgLibrary.EffectClasses
{
    public class Resistance
    {
        #region Field Region

        DamageType resistance;
        int amount;

        #endregion

        #region Property Region

        public DamageType ResistanceType
        {
            get { return resistance; }
            private set { resistance = value; }
        }

        public int Amount
        {
            get { return amount; }
            private set
            {
                if (value < 0)
                    amount = 0;
                else if (value > 100)
                    amount = 100;
                else
                    amount = value;
            }
        }

        #endregion
```

```
        #region Constructor Region

        public Resistance(ResistanceData data)
        {
            ResistanceType = data.ResistanceType;
            Amount = data.Amount;
        }

        #endregion

        #region Method Region

        public int Apply(int damage)
        {
            return (damage - damage * amount / 100);
        }

        #endregion

        #region Virtual Method Region
        #endregion
    }
}
```

First important note is that I change the namespace from **XRpgLibrary** to **RpgLibrary**. The data classes are basically the same other than the names of the first field. The **WeaknessData** class has **WeaknessType** and **ResistanceData** has **ResistanceType**. Those fields are what the weakness or resistance is applied to. If it is set to **DamageType.Fire** for example the entity is either weak against fire or strong against fire. I had thought of making it so that if a resistance is greater than 100% that it would heal the target. That would be something you could do but I ended up deciding against it. The other field in the data classes is **Amount** which is the amount of the resistance or weakness.

The **Weakness** class has two fields **weakness** and **amount**. The **weakness** field is the type of weakness and the **amount** field is the percentage of the weakness. An entity will have a percentage that they are weak to. A fire type entity may have a high weakness to water where they will take 25% more damage from a water attack. A resistance works in reverse. A fire type entity will be strong against a fire based attack taking 75% less damage from a fire based attack. There are get and set accessors to assign and get the values of the **weakness** and **amount** fields, **WeaknessType** and **Amount** respectively. The set part for both is private because you don't want the values to change after initially set.

The constructor takes a **WeaknessData** parameter that is the weakness to be created. It just sets the fields of the class using the values of the parameter passed in.

There is also a method in the class called **Apply** that will apply a weakness to the damage passed in. A weakness to an attack type will increase the damage caused by that attack type. To calculate the damage done after the weakness is applied you take the original damage and add the damage times the amount divided by 100. That is just a basic percentage formula.

The **Resistance** class has two fields as well that are similar to the **Weakness** class **resistance** and **amount**. The **resistance** field is the type of resistance and the **amount** field is the percentage of the resistance like in the **Weakness** class. There are get and set accessors to assign and get the values of the **resistance** and **amount** fields, **ResistanceType** and **Amount** respectively. The set part for both is private because you don't want the values to change after they are set initially.
The constructor takes a **ResistanceData** parameter that describes the resistance to be created. It then sets the fields for the class using the accessors and the values passed in.

There is an **Apply** method in the **Resistance** class as well that takes the damage that the resistance is applied to. To apply the resistance you use a similar formula as the **Apply** method of the **Weakness** class. You take the original damage and subtract the damage times the amount divided by 100. That will reduce the damage by the percentage of the resistance.

You will need a way to track the weaknesses and resistances of the entities in the game. The best place to do that would be in the **Entity** class. I'm going to add a new region to the **Entity** class. You will also want to add in a using statement for the **RpgLibrary.EffectClasses** namespace to the **Entity** class. I added my region after the **Calculated Attribute Field and Property Region**. Add the following to the **Entity** class.

```
using RpgLibrary.EffectClasses;

#region Resistance and Weakness Field and Property Region

private readonly List<Resistance> resistances;

public List<Resistance> Resistances
{
    get { return resistances; }
}

private readonly List<Weakness> weaknesses;

public List<Weakness> Weaknesses
{
    get { return weaknesses; }
}

#endregion
```

The fields that I added are **List<T>** for both resistances and weaknesses that are readonly. There are also public properties to expose the fields that are get only. The field **resistances** and property **Resistances** are for resistances and **weaknesses** and **Weaknesses** are for weaknesses. What you need to do now is modify the private constructor of the **Entity** class to initialize these new fields. Modify the private constructor of the **Entity** class to the following.

```
private Entity()
{
    Strength = 10;
    Dexterity = 10;
    Cunning = 10;
    Willpower = 10;
    Magic = 10;
    Constitution = 10;

    health = new AttributePair(0);
    stamina = new AttributePair(0);
    mana = new AttributePair(0);

    skills = new Dictionary<string, Skill>();
    spells = new Dictionary<string, Spell>();
    talents = new Dictionary<string, Talent>();

    skillModifiers = new List<Modifier>();
    spellModifiers = new List<Modifier>();
    talentModifiers = new List<Modifier>();

    resistances = new List<Resistance>();
    weaknesses = new List<Weakness>();
}
```

There is one more thing to do with weaknesses and resistances. You need to update the **Apply** method of **DamageEffect** to call the **Apply** method of any resistances and weaknesses. You would do that after rolling the damage but before checking to see if the damage is less than one. The order that I will apply them is first weaknesses and then resistances. It would be possible for an entity to have items with resistances and weaknesses that counter act each other. Both still should be applied though. There is the possibility that a lot of resistances will negate all damage and multiple weaknesses would greatly increase the damage being done. When you are designing weaknesses and resistances for your game you will have to be careful to balance them out so they don't make the game too hard or too easy when an entity has the appropriate items. Change the **Apply** method of the **DamageEffect** class to the following.

```
public override void Apply(Entity entity)
{
    int amount = modifier;

    for (int i = 0; i < numberOfDice; i++)
        amount += Mechanics.RollDie(dieType);

    foreach (Weakness weakness in entity.Weaknesses)
        if (weakness.WeaknessType == damageType)
            amount = weakness.Apply(amount);

    foreach (Resistance resistance in entity.Resistances)
        if (resistance.ResistanceType == damageType)
            amount = resistance.Apply(amount);

    if (amount < 1)
        amount = 1;

    switch (attackType)
    {
        case AttackType.Health:
            entity.Health.Damage((ushort)amount);
            break;
        case AttackType.Mana:
            entity.Mana.Damage((ushort)amount);
            break;
        case AttackType.Stamina:
            entity.Stamina.Damage((ushort)amount);
            break;
    }
}
```

There are two foreach loops. The first loops through all of the weaknesses in the **Weakness** property of the entity. If the **WeaknessType** property of the weakness matches the **damageType** field then the **amount** variable is set to the return value of the **Apply** method for the weakness. The foreach loop for resistances works the same way.

I'm going to end this tutorial here. I just wanted to add in resistances and weaknesses to the different types of effects. Things are really starting to take form and hopefully soon I will get to actually clobbering some mobs. 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