Creating a Role Playing Game with XNA Game Studio 3.0
Part 16
Improving the Player Character System

To follow along with this tutorial you will have to have read the previous tutorials to understand much of what it going on. You can find a list of tutorials here: XNA 3.0 Role Playing Game Tutorials You will also find the latest version of the project on the web site on that page. If you want to follow along and type in the code from this PDF as you go, you can find the previous project at this link: Eyes of the Dragon - Version 15 You can download the graphics for the these tutorials at this link: Graphics.zip

Again, I did quite a bit of programming on Eyes of the Dragon, the name I have given to the game that I am creating to teach one way on how to create a role playing game with XNA 3.0. I will not be doing anything with XNA in this tutorial. This tutorial has to do with improving the character generator to allow the creation of the player's character and the player character system.

The first thing that I did was create a new folder in the project called **Screens**. I did this because the project is getting a little disorganized. There were a lot of classes in the main folder. Then I moved the **ActionScreen**, **CreatePCScreen**, **CreditScreen**, **HelpScreen**, **InputScreen**, **IntroScreen**, **PopupScreen** and **StartUpScreen** into this folder. You can add them all at once by clicking them while holding down the **ctrl** key and then dragging them to the new folder.

I also added in a new folder called **PlayerCharacter** to the project. In this folder I will be putting all of the classes related to the **PlayerCharacter**. The first class that I will add to this folder will be called **PlayerCharacter**. Right-click the folder and select **Class** and call the class **PlayerCharacter**. I was going to make this class an abstract class that had to be inherited from. Instead, I made it a base class and will use polymorphism instead. That way, when I want to create a player character I can do like I did with the **ActiveScreen**. It is a **GameScreen** but I can make it into a **StartScreen**, **CreatePCScreen** or any other class that inherits from **GameScreen**. I will go into this in a little more detail later.

This class will continue to evolve as the game evolves. It is just a starting point for now. As always, I will give you the code and then explain it.

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

namespace New2DRPG
{
    class PlayerCharacter
    {
        public enum CharClass { Fighter = 0, Wizard = 1, Priest = 2, Thief = 3};

        static string[] classNames = {
                 "Fighter",
                 "Wizard",
                 "Priest",
                 "Thief" };

        protected string name;
```

```
protected bool gender;

protected int[] hitPoints = new int[2];
protected int[] spellPoints = new int[2];

protected int level = 1;

protected int strength = 10;
protected int stamina = 10;
protected int agility = 10;
protected int speed = 10;
protected int intellect = 10;
protected int luck = 10;

protected static string className;

public static string[] ClassNames
{
    get { return classNames; }
}

public int HitPointsMax
{
    get { return hitPoints[1]; }
}

public int HitPointsCurrent
{
    get { return hitPoints[0]; }
}

public int SpellPointsMax
{
    get { return spellPoints[1]; }
}

public int SpellPointsCurrent
{
    get { return spellPoints[0]; }
}

public virtual int Strength
{
    get { return strength; }
}

public virtual int Stamina
{
    get { return stamina; }
}

public virtual int Agility
{
    get { return agility; }
}

public virtual int Speed
{
    get { return speed; }
```

```
        }

        public virtual int Intellect
        {
            get { return intellect; }
        }

        public virtual int Luck
        {
            get { return luck; }
        }

    }
}
```

I made all of the variables in this class **protected** so that they would be **private** which I always like to do if at all possible, keep variables **private** and access them through properties. When you have folder in a project and you create, for example a new class, the folder name with a . is appended to the namespace. I just deleted that for all of the classes that I added to the folder. Just to keep from having to add **using** to these classes and other classes that need to use these classes.

There is a public **enum** in this class. This **enum** will be used to help in the character generator and telling what type of character to create in the game. There is also a **static** array of strings that I will expose through a get only property. You will notice that the order of the items in the **enum** are the same as the strings in the array. This is important. If you don't do this when you choose an item in the menu for the class when I create the character it could be of the wrong class. For example if you had in the **enum** the following items **Fighter, Wizard, Priest, Thief** and in the array of names **Fighter, Priest, Wizard, Thief**. If the player chose **Priest** in the list when the game went to create the character they would get a **Wizard** character. The same would be true in reverse if the player chose **Wizard** they would get a **Priest**.

Next there are some protected variables that will be common to all characters. I will be add more later as needed but I at least wanted to get a start on what is needed. All characters will have a name and a gender, those were two of the easy ones.

There are two other variables that are an array of integers of length 2, **hitPoints** and **spellPoints**. They are for the character's hit points and the character's spell points. I created public properties to get these integers. For hit points I will use **HitPointsCurrent** and **HitPointsMax**. As well, for spell points I will use **SpellPointsCurrent** and **SpellPointsMax**. If you look at the properties the current properties use index 0 and the max properties used index 1. This is just something I've seen a lot of other RPGs use and I have used in other RPGs I've made.

Next there are the ability scores for the character. When I started this tutorial series I had chosen the following ability scores: **Strength, Stamina, Agility, Speed, Intellect** and **Luck**. This is what each of these abilities represents:

- **Strength** measures a character's ability to inflict damage on an opponent and how much weight a character can carry. If a character is carrying too much their Agility and Speed will be effected.
- **Stamina** determines how much damage can take and how resistant they are to poisons and magic.
- **Agility** is how nimble a character is. It helps in striking and evading attacks.

- **Speed** is how fast a character is. In combat a character with high speed will strike faster in combat.
- **Intellect** measures a character's ability to learn and retain information. It also determines how powerful a character's spells are.
- **Luck** helps in landing critical hits (that inflict 1.5x damage) on an opponent.

I made the properties for these variables virtual so they can be overriden in the inherited classes. They are also, at the moment get only properties. Being virtual that can be overriden in the inherited class.

There is one other protected variable in the **PlayerCharacter** class, **className**. This variable will hold the name of the character's class. There is a public get only property to expose the variable **ClassName**.

As the game progresses I will be adding more and more virtual methods to this class that will be overriden in the inherited classes.

I added in four other classes to the **PlayerCharacter** folder: **FighterCharacter**, **WizardCharacter**, **PriestCharacter** and **ThiefCharacter**. Each of them has as a base class the **PlayerCharacter** class. Right now there are a little basic. As the game evolves though they will be expanded. I created them so I can add some details to the HUD of the game in this tutorial as well. Go a head and add four classes to the **PlayerCharacter** folder with the names: **FighterCharacter, WizardCharacter, PriestCharacter** and **ThiefCharacter**.

The classes are all the same except for the values of the constants in the classes. The constants are used to determine the starting hit points, spell points and the ability scores of the player character. This is the reason I decided to use inheritance and polymorphism. In the game the player's character will have as it's type **PlayerCharacter**. If the player chooses to play a fighter character when I create the player's character I will use **FighterCharacter** class to create the **PlayerCharacter** object. Now, if I have a virtual method in **PlayerCharacter** and override it in **FighterCharacter** when I call the method from the **PlayerCharacter** object the method from **FighterCharacter** will be called. This is one of the hardest concepts of object-oriented programming to understand so don't worry of you don't understand it right now. When you see it in practise later and I will go more into depth into this. I will actually write a tutorial on the subject.
I will give you the code for each of the new classes and then explain them. I will start with the **FighterCharacter** class.

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

namespace New2DRPG
{
    class FighterCharacter : PlayerCharacter
    {
        const int startingHitPoints = 20;
        const int startingSpellPoints = 0;

        const int startingStrength = 16;
        const int startingStamina = 14;
        const int startingAgility = 12;
        const int startingSpeed = 10;
```

```
        const int startingIntellect = 10;
        const int startingLuck = 10;

        public FighterCharacter(string name, bool gender)
        {
            FighterCharacter.className = "Fighter";
            this.name = name;
            this.gender = gender;

            this.hitPoints[0] = startingHitPoints;
            this.hitPoints[1] = startingHitPoints;
            this.spellPoints[0] = startingSpellPoints;
            this.spellPoints[1] = startingSpellPoints;

            this.strength = startingStrength;
            this.stamina = startingStamina;
            this.agility = startingAgility;
            this.speed = startingSpeed;
            this.intellect = startingIntellect;
            this.luck = startingLuck;
        }

    }
}
```

The first thing that I did in the class was to change the namespace of the class to just **New2DRPG,** not **New2DRPG.PlayerCharacter**. I did this in all of the other classes so I won't go into this again in the other classes.

There are eight constants in this class at the moment. There are variables for the starting values of hit points, spell points, strength, stamina, agility, speed, intellect and luck. The values are not terribly important at the momnet though. They are just values that I came up with that would fit a fighter character.
Fighters typically have good hit points so I set the hit points high. I am a firm believer that fighters should never be able to cast spells so I set the spell points to 0. I was thinking, that for a normal everyday person would have 10 as a base for their abilities. Fighters should be strong, have good stamina and fair agility wouldn't hurt so I set those values higher than average. Some fighters can be fast but I was thinking that the average starting out fighter wouldn't be all that fast. Many people think that fighters are dumb brutes. I think for a fighter to be successful they should have at least average intelligence. I didn't think a figher would have any special luck bonuses so I set that to average as well.

The constructor for all of the classes take two parameters: **name** and **gender**. **name** is of course the name of the character and **gender** is the gender of the character. In the constructor I just then set the values of the protected variables in the base class. Since this class is for fighter characters I set the name of the character class to **Fighter**. All of the constructors are basically the same so I won't go into them again in this tutorial. The only difference being what I set the name of the character class to.

So, I was thinking I would just give you the code for the other character classes and then give a quick description after I have given you all the code why I chose the values I chose. This is the code for the **WizardCharacter.**

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Text;

namespace New2DRPG
{
    class WizardCharacter : PlayerCharacter
    {
        const int startingHitPoints = 10;
        const int startingSpellPoints = 20;

        const int startingStrength = 10;
        const int startingStamina = 12;
        const int startingAgility = 12;
        const int startingSpeed = 10;
        const int startingIntellect = 16;
        const int startingLuck = 10;

        public WizardCharacter(string name, bool gender)
        {
            WizardCharacter.className = "Wizard";
            this.name = name;
            this.gender = gender;
            this.hitPoints[0] = startingHitPoints;
            this.hitPoints[1] = startingHitPoints;
            this.spellPoints[0] = startingSpellPoints;
            this.spellPoints[1] = startingSpellPoints;

            this.strength = startingStrength;
            this.stamina = startingStamina;
            this.agility = startingAgility;
            this.speed = startingSpeed;
            this.intellect = startingIntellect;
            this.luck = startingLuck;
        }
    }
}
```

When most people think of wizards they generally don't have a lot of hit points. They will have high spell points though because of all the time the spent studying magic. I thought that wizards would have a bit better stamina than an average person and a bit better agility because of all the complex magic that they make. Of course wizards would have high intellects to learn and memorize all of their spells. I never thought of wizards are particularly lucky.

Next I will give you the code for the **PriestCharacter** class.

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

namespace New2DRPG
{
    class PriestCharacter : PlayerCharacter
    {
        const int startingHitPoints = 15;
        const int startingSpellPoints = 15;
```

```
        const int startingStrength = 12;
        const int startingStamina = 12;
        const int startingAgility = 10;
        const int startingSpeed = 10;
        const int startingIntellect = 12;
        const int startingLuck = 12;

        public PriestCharacter(string name, bool gender)
        {
            PriestCharacter.className = "Priest";
            this.name = name;
            this.gender = gender;
            this.hitPoints[0] = startingHitPoints;
            this.hitPoints[1] = startingHitPoints;
            this.spellPoints[0] = startingSpellPoints;
            this.spellPoints[1] = startingSpellPoints;

            this.strength = startingStrength;
            this.stamina = startingStamina;
            this.agility = startingAgility;
            this.speed = startingSpeed;
            this.intellect = startingIntellect;
            this.luck = startingLuck;
        }
    }
}
```

I am taking my idea of a priest from D&D's cleric class. They have good hit points and good spell points. They are fairly strong and have good stamina. I've never really thought of them as particularly agile or fast. They would have good intellects and have the favor of their god or godess. (I do not believe in false gods. I'm saying this from a fantasy point of view.)

That just leaves the thief class. This is the code for the **ThiefCharacter** class.

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

namespace New2DRPG
{
    class ThiefCharacter : PlayerCharacter
    {
        const int startingHitPoints = 15;
        const int startingSpellPoints = 0;

        const int startingStrength = 12;
        const int startingStamina = 10;
        const int startingAgility = 14;
        const int startingSpeed = 12;
        const int startingIntellect = 10;
        const int startingLuck = 12;

        public ThiefCharacter(string name, bool gender)
        {
            ThiefCharacter.className = "Thief";
            this.name = name;
```

```
            this.gender = gender;
            this.hitPoints[0] = startingHitPoints;
            this.hitPoints[1] = startingHitPoints;
            this.spellPoints[0] = startingSpellPoints;
            this.spellPoints[1] = startingSpellPoints;

            this.strength = startingStrength;
            this.stamina = startingStamina;
            this.agility = startingAgility;
            this.speed = startingSpeed;
            this.intellect = startingIntellect;
            this.luck = startingLuck;
        }
    }
}
```

I have decided that a typical thief will have not bad hit points. I don't believe in thieves in being able to cast spells so there are no spell points. Thieves tend to be a little strong. They will almost always have great agility with the ability to easily manipulate fine devices. The are usually quick on their feet. As well when I think of a thief they have tend to have the ability to get out of tricky situations and have a little luck.

I made a few changes to the **CreatePCScreen** class. The first change was instead of setting the array **classNames** manually I created it from the array in the **PlayerCharacter** class.

```
string[] classNames = PlayerCharacter.ClassNames;
```

I added in a new variable to the class **characterClass** of type **CharClass**, the **enum** in the **PlayerCharacter** class. I set it to the first character class, **Fighter**.

```
PlayerCharacter.CharClass characterClass = PlayerCharacter.CharClass.Fighter;
```

I also added in two new get only properties called **CharacterClass** and **CharacterName**. **CharacterClass** is used to access the **characterClass** variable outside of the class. **CharacterName** is used to access the **name** variable in the **CreatePCScreen** class. I changed the name of the **Gender** property to **CharacterGender**. It is still used to access the **gender** variable.

```
public PlayerCharacter.CharClass CharacterClass
{
    get { return characterClass; }
}

public bool CharacterGender
{
    get { return gender; }
}

public string CharacterName
{
    get { return name; }
}
```

There was one other change to the **CreatePCScreen** class. I made a change to the **ChangeClass** method. I will give you the new method and then explain the new method.

```
public void ChangeClass(int className)
{
    this.className = className;
    switch (className)
    {
        case 0:
            characterClass = PlayerCharacter.CharClass.Fighter;
            break;
        case 1:
            characterClass = PlayerCharacter.CharClass.Wizard;
            break;
        case 2:
            characterClass = PlayerCharacter.CharClass.Priest;
            break;
        case 3:
            characterClass = PlayerCharacter.CharClass.Thief;
            break;
    }
}
```

The method still sets the **className** variable. It also sets the **characterClass** variable. I set the variable according to the value of the **className** variable. This is where the order of the class names in the **classNames** array matches the order of the values in the **enum CharClass** in the **PlayerCharacter** class.

Now, I also changed the **LoadContent** method in the **Game1** class. That method was a little long. I did a fair bit of refractoring. I was going to go over the refractoring process but this tutorial is already a little long. Basically what I did was for each of the different screens, I selected the code. Then I clicked **Refractor** and chose the **Extract method** option and then gave the method a name. For example I had selected the following code that was used to create the **StartScreen** object.

```
background = Content.Load<Texture2D>("titlescreen");
startScreen = new StartScreen(this,
    normalFont,
    background,
    Content.Load<Texture2D>(@"GUI\buttonbackground"));
Components.Add(startScreen);
```

Then I gave the method the name **LoadStartScreen** when I chose **Extract method**. Visual C# gave me the following method.

```
private void LoadStartScreen()
{
    background = Content.Load<Texture2D>("titlescreen");
    startScreen = new StartScreen(this,
        normalFont,
        background,
        Content.Load<Texture2D>(@"GUI\buttonbackground"));
    Components.Add(startScreen);
}
```

I will give you the new code for the **LoadContent** method and all of the methods that I created to shorten the **LoadContent** method.

```
protected override void LoadContent()
{
    spriteBatch = new SpriteBatch(GraphicsDevice);
    Services.AddService(typeof(SpriteBatch), spriteBatch);
    Services.AddService(typeof(ContentManager), Content);

    normalFont = Content.Load<SpriteFont>("normal");

    LoadCreatePCScreen();
    LoadStartScreen();
    LoadHelpScreen();
    LoadActionScreen();
    LoadQuitPopUpScreen();
    LoadGenderPopUpScreen();
    LoadClassPopUpScreen();
    LoadDifficultyPopUpScreen();
    LoadNameInputScreen();
    LoadCreditScreen();
    LoadIntroScreen();

    creditScreen.Hide();
    startScreen.Hide();
    helpScreen.Hide();
    createPCScreen.Hide();

    activeScreen = introScreen;
    activeScreen.Show();
}

private void LoadIntroScreen()
{
    introScreen = new IntroScreen(this);
    Components.Add(introScreen);
}

private void LoadCreditScreen()
{
    creditScreen = new CreditScreen(this);
    Components.Add(creditScreen);
}

private void LoadNameInputScreen()
{
    background = Content.Load<Texture2D>(@"GUI\choosename");
    nameInputScreen = new InputScreen(this,
        normalFont,
        background,
        Content.Load<Texture2D>(@"GUI\buttonbackgroundshort"));
    Components.Add(nameInputScreen);
}

private void LoadDifficultyPopUpScreen()
{
    background = Content.Load<Texture2D>(@"GUI\choosedifficulty");
    difficultyPopUpScreen = new PopUpScreen(this,
        normalFont,
        background,
        Content.Load<Texture2D>(@"GUI\buttonbackgroundshort"));
    string[] difficultyItems = new string[] { "Easy", "Normal", "Hard",
```

```
"Ultimate" };
    difficultyPopUpScreen.SetMenuItems(difficultyItems);
    Components.Add(difficultyPopUpScreen);
}

private void LoadClassPopUpScreen()
{
    background = Content.Load<Texture2D>(@"GUI\chooseclass");
    classPopUpScreen = new PopUpScreen(this,
        normalFont,
        background,
        Content.Load<Texture2D>(@"GUI\buttonbackgroundshort"));
    string[] classItems = PlayerCharacter.ClassNames;
    classPopUpScreen.SetMenuItems(classItems);
    Components.Add(classPopUpScreen);
}

private void LoadGenderPopUpScreen()
{
    background = Content.Load<Texture2D>(@"GUI\maleorfemale");
    genderPopUpScreen = new PopUpScreen(this,
        normalFont,
        background,
        Content.Load<Texture2D>(@"GUI\buttonbackgroundshort"));
    string[] genderItems = new string[] { "Female", "Male" };
    genderPopUpScreen.SetMenuItems(genderItems);
    Components.Add(genderPopUpScreen);
}

private void LoadQuitPopUpScreen()
{
    background = Content.Load<Texture2D>(@"GUI\quitpopupbackground");
    quitPopUpScreen = new PopUpScreen(this,
        normalFont,
        background,
        Content.Load<Texture2D>(@"GUI\buttonbackgroundshort"));
    Components.Add(quitPopUpScreen);
    quitPopUpScreen.Hide();
}

private void LoadActionScreen()
{
    actionScreen = new ActionScreen(this, normalFont, "tileset1");
    Components.Add(actionScreen);
    actionScreen.Hide();
}

private void LoadHelpScreen()
{
    background = Content.Load<Texture2D>("helpscreen");
    helpScreen = new HelpScreen(this,
        normalFont,
        background,
        Content.Load<Texture2D>(@"GUI\buttonbackground"));
    Components.Add(helpScreen);
}

private void LoadStartScreen()
{
```

```
    background = Content.Load<Texture2D>("titlescreen");
    startScreen = new StartScreen(this,
        normalFont,
        background,
        Content.Load<Texture2D>(@"GUI\buttonbackground"));
    Components.Add(startScreen);
}

private void LoadCreatePCScreen()
{
    createPCScreen = new CreatePCScreen(this, normalFont);
    Components.Add(createPCScreen);
}
```

If you look at the **LoadClassPopUpScreen** method you will see that I made one change. I used the array of class names from the **PlayerCharacter** class to set the menu items for the **ClassPopUpScreen** class.

I added a new class level variable to the **Game1** class. This variable will hold the player's character. It is of type **PlayerCharacter**. It will be able to be of any of the types that inherit from **PlayerCharacter** though: **FighterCharcter**, **WizardCharacter**, **PriestCharacter** and **ThiefCharacter**.

```
PlayerCharacter playerCharacter;
```

I made a change to the **HandleCreatePCScreenInput** method. I called a new method, **CreatePlayerCharacter**, that I will give you the code for in a moment that will create the player's character. I will give you the code for both methods and then explain the new method.

```
private void HandleCreatePCScreenInput()
{
    if (CheckKey(Keys.Enter) || CheckKey(Keys.Space))
    {
        switch (createPCScreen.SelectedIndex)
        {
            case 0:
                activeScreen.Enabled = false;
                activeScreen = nameInputScreen;
                activeScreen.Show();
                break;
            case 1:
                activeScreen.Enabled = false;
                activeScreen = genderPopUpScreen;
                activeScreen.Show();
                break;
            case 2:
                activeScreen.Enabled = false;
                activeScreen = classPopUpScreen;
                activeScreen.Show();
                break;
            case 3:
                activeScreen.Enabled = false;
                activeScreen = difficultyPopUpScreen;
                activeScreen.Show();
                break;
            case 4:
                activeScreen.Hide();
```

```
                CreatePlayerCharacter();
                activeScreen = startScreen;
                activeScreen.Show();
                break;
            case 5:
                activeScreen.Hide();
                activeScreen = actionScreen;
                activeScreen.Show();
                break;
        }
    }
}

private void CreatePlayerCharacter()
{
    if (createPCScreen.CharacterClass == PlayerCharacter.CharClass.Fighter)
    {
        playerCharacter = new FighterCharacter(
            createPCScreen.CharacterName,
            createPCScreen.CharacterGender);
    }
    if (createPCScreen.CharacterClass == PlayerCharacter.CharClass.Priest)
    {
        playerCharacter = new PriestCharacter(
            createPCScreen.CharacterName,
            createPCScreen.CharacterGender);
    }
    if (createPCScreen.CharacterClass == PlayerCharacter.CharClass.Thief)
    {
        playerCharacter = new ThiefCharacter(
            createPCScreen.CharacterName,
            createPCScreen.CharacterGender);
    }
    if (createPCScreen.CharacterClass == PlayerCharacter.CharClass.Wizard)
    {
        playerCharacter = new WizardCharacter(
            createPCScreen.CharacterName,
            createPCScreen.CharacterGender);
    }
}
```

All that the new method does is go through a series of if-statements. It then creates a character based on the player's selection. If the player has chosen a **Fighter** character. The **playerCharacter** is created using the **FighterCharacter** class and for **Wizard** the **WizardCharacter** class, **Priest** the **PriestCharacter** class and **Thief** the **ThiefCharacter**.

Well, that is it for another tutorial. I will be working on more so I encourage you to keep either visiting this site or my blog, http://xna-rpg.blogspot.com for the latest news on these tutorials.