Creating a Role Playing Game with XNA Game Studio 3.0
Part 17
Improving the Player Character System Part 2
Improving the HUD

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 16.You can download the graphics for the these tutorials at this link: Graphics.zip

I guess I have been fussing too much over little things lately. I will try and focus more on functionality for the next little while instead of appearance. Appearance is an important part of a game and it does deserver a lot of attention when you are creating your own game. I did add three new textures to the game: the new version of the character HUD (**characterHUD.png**), a frame for displaying the hit points and spell points(**textureBorder.jpg**) and a white bar(**whitebar.jpg**) that I will tint to display the hit points and spell points. All of the graphics are in the file above. You will need to right-click the **Backgrounds** folder and add the new **characterHUD.png** file to it. The other two files are added to the **GUI** folder.

This is really a two part tutorial. I made a few modifications to the player character classes and I made a few changes to the HUD of the game. I had said on my blog that I was going to add a pop up to the game to display the statistics of the character. I will not get to that in today's tutorial. I will be getting to it soon as well as modifying the screen management system.

To get started you will want to right-click the **PlayerCharacter** folder and add in a new class called **CharacterAbilities**. Instead of having individual variables inside the **PlayerCharacter** class I will be using a class to access the player character's ability scores. I will give you the new class and then explain the way the class works.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Xna.Framework;

namespace New2DRPG
{
    class CharacterAbilities
    {
        string[] abilityNames = {
                 "STRENGTH",
                 "STAMINA",
                 "AGILITY",
                 "SPEED",
                 "INTELLECT",
                 "LUCK" };

        public enum Abilities
        {
```

```
    Strength = 0,
    Stamina = 1,
    Agility = 2,
    Speed = 3,
    Intellect = 4,
    Luck = 5
};

const float minAbilityScore = 1f;
const float maxAbilityScore = 20f;

int strength;
int stamina;
int agility;
int speed;
int intellect;
int luck;

public int this[Abilities index]
{
    get
    {
        if (index == Abilities.Strength)
            return strength;
        if (index == Abilities.Stamina)
            return stamina;
        if (index == Abilities.Agility)
            return agility;
        if (index == Abilities.Speed)
            return speed;
        if (index == Abilities.Intellect)
            return intellect;
        if (index == Abilities.Luck)
            return luck;
        throw new Exception("Index out of bounds exception.");
    }
    set
    {
        if (index == Abilities.Strength)
            Strength = value;
        else if (index == Abilities.Stamina)
            Stamina = value;
        else if (index == Abilities.Agility)
            Agility = value;
        else if (index == Abilities.Speed)
            Speed = value;
        else if (index == Abilities.Intellect)
            Intellect = value;
        else if (index == Abilities.Luck)
            Luck = value;
        else
            throw new Exception("Index out of bounds exception.");
    }
}

public string this[int index]
{
    get
    {
```

```
            if (index > Length)
                throw new Exception("Index out of bounds exception.");
            return abilityNames[index];
        }
    }

    public int Length
    {
        get { return abilityNames.Length; }
    }

    public int Strength
    {
        get { return strength; }
        set { strength = (int)MathHelper.Clamp(value,
                minAbilityScore,
                maxAbilityScore); }
    }

    public int Stamina
    {
        get { return stamina; }
        set
        {
            stamina = (int)MathHelper.Clamp(value,
              minAbilityScore,
              maxAbilityScore);
        }
    }

    public int Agility
    {
        get { return agility; }
        set
        {
            agility = (int)MathHelper.Clamp(value,
              minAbilityScore,
              maxAbilityScore);
        }
    }

    public int Speed
    {
        get { return speed; }
        set
        {
            speed = (int)MathHelper.Clamp(value,
              minAbilityScore,
              maxAbilityScore);
        }
    }

    public int Intellect
    {
        get { return intellect; }
        set
        {
            intellect = (int)MathHelper.Clamp(value,
              minAbilityScore,
```

```
                maxAbilityScore);
            }
        }

        public int Luck
        {
            get { return luck; }
            set
            {
                luck = (int)MathHelper.Clamp(value,
                  minAbilityScore,
                  maxAbilityScore);
            }
        }

    }
}
```

The first thing I did was to add in a **using** statement for the XNA framework. To be honest I didn't have to do this step. The reason I did was I wanted to use the **MathHelper.Clamp** method in the properties that I wanted to use for the class to limit the values that the abilities could be set to. I didn't want values below 1 and I chose 20 as a resonable upper limit of the abilities. The 20 can be easily changed down the road to allow greater values.

There is an array of strings in the class with the name of the ability. One for each of the abilities: **Strength**, **Stamina**, **Agility**, **Speed**, **Intellect** and **Luck**. I created an **enum** for the abilities as well. There are also two constants in the class. One for the minimum attribute value and one for the maximum attribute value. There are six other variables in the class. One for each of the attributes of the character.

Next there are two indexer. Indexers allow you to use an object like it is an array. Indexers look a lot like properties. The big difference is the use of **this** instead of the name of the property and the **index parameter**. Using the first indexer you can get and set the values of the attributes like you would an array using the **Abilities enum** as the index as follows:

```
CharacterAbilities abilities = new CharacterAbilities();
abilities[Abilities.Strength] = value;
value = abilities[Abilities.Strength];
```

I added that indexer in as it may come in handy down the road. The second indexer is a get only indexer. This indexer used an integer as it's index. You can have multiple indexers but they can not use the same parameter type. You could not have two indexers, with different return values, that used integers as their paremeter. Each of the indexers will throw exceptions if the value is outside of the range of values that are expected by them.

There are seven properties in this class. The first property was added to help those using the class from generating an exception when using the indexers. For example, if the person using the class wanted to use a for loop to get all the names of the attributes they would know how many there are and know when to stop.

The other six properties are for accessing the different abilities of the character. **Strength** works for **strength**, **Stamina** works on **stamina** and so on. The get part just returns the ability score. In the set

part I used the **MathHelper.Clamp** method to make sure the ability score was never outside of the allowed values.

Next I had to modify the **PlayerCharacter, FighterCharacter, WizardCharacter, PriestCharacter** and **ThiefCharacter** classes to reflect this new update. I will start with the **PlayerCharacter** class, as it is the parent class to all the other classes. This is the code for the new **PlayerCharacter** class.

```
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 CharacterAbilities abilities = new CharacterAbilities();

        protected string className;

        public string Name
        {
            get { return name; }
        }

        public string ClassName
        {
            get { return 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 CharacterAbilities Abilities
        {
            get { return abilities; }
        }
    }
}
```

All that changed in this class was that I removed the six protected fields for the player character's ability scores and the properties for them and replaced them with a protect field for the **CharacterAbilities** class and a public property to get the object.

In the other four classes all I did was change them to reflect the change made to the **PlayerCharcter** class. I change them from using the individual variables to use the new class. I will give you the code for the **FighterCharcter**, **WizardCharacter**, **ThiefCharacter** and **PriestCharacter** classes.

```
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)
        {
            this.className = "Fighter";
            this.name = name;
            this.gender = gender;

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

```
            abilities.Strength = startingStrength;
            abilities.Stamina = startingStamina;
            abilities.Agility = startingAgility;
            abilities.Speed = startingSpeed;
            abilities.Intellect = startingIntellect;
            abilities.Luck = startingLuck;
        }

    }
}

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 = 12;
        const int startingIntellect = 16;
        const int startingLuck = 10;

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

            abilities.Strength = startingStrength;
            abilities.Stamina = startingStamina;
            abilities.Agility = startingAgility;
            abilities.Speed = startingSpeed;
            abilities.Intellect = startingIntellect;
            abilities.Luck = startingLuck;
        }
    }
}

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)
        {
            this.className = "Priest";
            this.name = name;
            this.gender = gender;
            this.hitPoints[0] = startingHitPoints;
            this.hitPoints[1] = startingHitPoints;
            this.spellPoints[0] = startingSpellPoints;
            this.spellPoints[1] = startingSpellPoints;

            abilities.Strength = startingStrength;
            abilities.Stamina = startingStamina;
            abilities.Agility = startingAgility;
            abilities.Speed = startingSpeed;
            abilities.Intellect = startingIntellect;
            abilities.Luck = startingLuck;
        }
    }
}

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)
        {
            this.className = "Thief";
            this.name = name;
            this.gender = gender;
            this.hitPoints[0] = startingHitPoints;
            this.hitPoints[1] = startingHitPoints;
            this.spellPoints[0] = startingSpellPoints;
```

```
            this.spellPoints[1] = startingSpellPoints;

            abilities.Strength = startingStrength;
            abilities.Stamina = startingStamina;
            abilities.Agility = startingAgility;
            abilities.Speed = startingSpeed;
            abilities.Intellect = startingIntellect;
            abilities.Luck = startingLuck;
        }
    }
}
```

I now will be making a lot of changes to the **ActionScreen** class and a few changes to the **TileEngine** class. The changes to the tile engine are really simple. I changed the view port width and height for the new HUD, which is 1024x96 pixels, as well as the tile width and height to have the tiles fit nicely in the window. I'm not sure why but I always like the tiles to fit nicely on the screen. These are the new values:

```
static int tileWidth = 64;
static int tileHeight = 48;
static int viewPortWidth = 1024;
static int viewPortHeight = 672;
```

That just leaves the **ActionScreen** class. In this class I made a few changes. One to use the new graphic I created for the HUD and another to display the character's current hit points and spell points in a bar with their value centered in the bar. There is one thing that you will need to do before working with this part though. I added a new sprite font to the game. Right-click the **Content** folder and add a new sprite font called **smallFont**. You will need to change the Size and Style tags:

```
    <Size>14</Size>
    <Style>Bold</Style>
```

I will give you the code for the **ActionScreen** and then explain why I have done what I have done. This is the code for the **ActionScreen** class.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using New2DRPG.CoreComponents;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Content;
using New2DRPG.SpriteClasses;

namespace New2DRPG
{
    class ActionScreen : GameScreen
    {
        SpriteFont gameFont;
        SpriteFont interfaceFont;
        Texture2D tilesetTexture;
        Texture2D characterHUDTexture;

        Texture2D borderTexture;
        Texture2D barTexture;
```

```
ContentManager Content;
SpriteBatch spriteBatch;

string tilesetName;
TileEngine tileEngine;
Tileset tileset;
PlayerCharacter playerCharacter = null;
CharacterAbilities abilityNames = new CharacterAbilities();

int viewportWidth;
int viewportHeight;
int screenWidth;
int screenHeight;

public ActionScreen(Game game, SpriteFont gameFont, string tilesetName)
    : base(game)
{
    this.gameFont = gameFont;
    Content =
        (ContentManager)Game.Services.GetService(typeof(ContentManager));

    spriteBatch =
        (SpriteBatch)Game.Services.GetService(typeof(SpriteBatch));
    this.tilesetName = tilesetName;
    LoadContent();

    tileset = new Tileset(tilesetTexture, 128, 128, 4, 4);
    tileEngine = new TileEngine(game, this.tileset, 50, 50);
    Components.Add(tileEngine);
    tileEngine.Show();

    viewportWidth = TileEngine.ViewPortWidth;
    viewportHeight = TileEngine.ViewPortHeight;
    screenWidth = game.Window.ClientBounds.Width;
    screenHeight = game.Window.ClientBounds.Height;
}

protected override void LoadContent()
{
    base.LoadContent();
    tilesetTexture = Content.Load<Texture2D>(@"Tilesets\" + tilesetName);
    characterHUDTexture =
          Content.Load<Texture2D>(@"Backgrounds\characterhud");
    borderTexture = Content.Load<Texture2D>(@"GUI\textureborder");
    barTexture = Content.Load<Texture2D>(@"GUI\whitebar");
    this.interfaceFont = Content.Load<SpriteFont>("smallFont");
}

public override void Update(GameTime gameTime)
{
    base.Update(gameTime);

}

public override void Draw(GameTime gameTime)
{
    base.Draw(gameTime);
```

```
        Vector2 position = new Vector2(0, 0);
        position.Y = viewportHeight;
        spriteBatch.Draw(characterHUDTexture, position, Color.White);
        position = new Vector2(viewportWidth + 10, 5);

        DrawHitPointsSpellPoints();
    }

    private void DrawHitPointsSpellPoints()
    {
        string text;
        Vector2 stringSize;

        Vector2 position = new Vector2(
            8, viewportHeight + 12);

        spriteBatch.Draw(borderTexture,
            position,
            Color.White);
        position.X += 5;
        position.Y += 5;
        spriteBatch.Draw(barTexture,
            position,
            Color.Red);

        text = playerCharacter.HitPointsCurrent.ToString();
        stringSize = interfaceFont.MeasureString(text);
        position.X += (200 - stringSize.X) / 2;
        position.Y -= 5;
        position.Y += (30 - interfaceFont.LineSpacing) / 2;
        spriteBatch.DrawString(interfaceFont, text, position, Color.White);

        position.X = 8;
        position.Y = viewportHeight + borderTexture.Height + 24;

        spriteBatch.Draw(borderTexture,
            position,
            Color.White);
        position.X += 5;
        position.Y += 5;
        spriteBatch.Draw(barTexture,
            position,
            Color.Blue);

        text = playerCharacter.SpellPointsCurrent.ToString();
        stringSize = interfaceFont.MeasureString(text);
        position.X += (200 - stringSize.X) / 2;
        position.Y -= 5;
        position.Y += (30 - interfaceFont.LineSpacing) / 2;
        spriteBatch.DrawString(interfaceFont, text, position, Color.White);
    }

    public override void Show()
    {
        base.Show();
        Enabled = true;
        Visible = true;
    }
```

```
        public override void Hide()
        {
            base.Hide();
            Enabled = false;
            Visible = false;
        }

        public void SetPlayerCharacter(PlayerCharacter playerCharacter)
        {
            this.playerCharacter = playerCharacter;
        }
    }
}
```

I added in a new **SpriteFont** object, **interfaceFont** and two **Texture2D**s for the frame for displaying the hit points and spell points(**borderTexture**) and the white bar(**barTexture**) that I will tint to display the current spell points and hit points. In the **LoadContent** method I loaded in the new font that I added to the project, **smallFont**, as the **interfaceFont**. I also loaded in all of the other textures and the game font. There is also a **PlayerCharacter** object variable in this class. For now I will be passing this variable to the class after creating the character. I have a plan for the future but for now this will do. There is a method in the class called **SetPlayerCharacter** that takes a **PlayerCharacter** object as it's parameter.

There are also for integer variables that I thought might end up being useful. They hold the and and the width of the screen and the viewport. They are set in the constructor. The viewport variable are set using the **TileEngine** and the screen variables are set using the **Window** property.

In the **Draw** method I handled drawing the HUD and I called a method to draw the player character's hit points and spell points **DrawHitPointsSpellPoints**. The new HUD is just a solid bar. I used the view port height to find where to draw the HUD.

In the **DrawHitPointsSpellPoints** method all I really did is a lot of positioning based on the size of the HUD and the size of the textures and draw the textures and the strings. I will just mention how I centered the text in the bars. I created the frames for the bars to represent the hit points and spell point to be 210 pixels wide and 30 pixels high. The bars for the hit points are 200 pixels wide by 20 pixels high. The bars have a 5 pixel border around them. After I draw the frames I add 5 pixels to the X and Y coordinates to center the bars in their frames.

To center the text in the frames I get the size of the text using the **MessureString** method of the **SpriteFont** object. For the X value I just needed to take the width of the bar, 200 pixels and subtract the width of the text and divide that by 2. If you try and do that with the Y value it doesn't work as well. The text gets a little cut off. For the Y value it is better to use the **LineSpacing** property of the **SpriteFont**. I first subtracted 5 from the current Y value and used a similar formula: 30 pixels minus the **LineSpacing** value divided by 2.

I had to make a couple changes in the **Game1** class. The first change I made was in the **HandleStartScreenInput** method. At the moment if the user chooses the second option in the menu, **The Story Continues**, the game jumps right to action screen. What I did was I created the default character when the user chooses the first option, **The Story Begins**, Evander a male fighter character and pass it to the **ActionScreen**. This is the updated method.

```
private void HandleStartScreenInput()
{
    if (CheckKey(Keys.Enter) || CheckKey(Keys.Space))
    {
        switch (startScreen.SelectedIndex)
        {
            case 0:
                activeScreen.Hide();
                activeScreen = createPCScreen;
                activeScreen.Show();
                break;
            case 1:
                activeScreen.Hide();
                playerCharacter = new FighterCharacter("Evander", false);
                actionScreen.SetPlayerCharacter(playerCharacter);
                activeScreen = actionScreen;
                actionScreen.Show();
                break;
            case 2:
                activeScreen.Hide();
                activeScreen = helpScreen;
                activeScreen.Show();
                break;
            case 3:
                activeScreen.Hide();
                activeScreen = creditScreen;
                activeScreen.Show();
                break;
            case 4:
                activeScreen.Enabled = false;
                activeScreen = quitPopUpScreen;
                activeScreen.Show();
                break;
        }
    }
}
```

Another change was in the **HandleCreatePCScreenInput** method. In this method I had put the call to **CreatePlayerCharacter** in the wrong case. It should have been in the last case. This is the updated 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();
                activeScreen = startScreen;
                activeScreen.Show();
                break;
            case 5:
                activeScreen.Hide();
                CreatePlayerCharacter();
                activeScreen = actionScreen;
                activeScreen.Show();
                break;
        }
    }
}
```

The one last change was in the **CreatePlayerCharacter** method. In this method I needed to call the **SetPlayerCharacter** method of the action screen. Again, here is the new method.

```
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);
    }
    actionScreen.SetPlayerCharacter(playerCharacter);
}
```

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.