# Creating a Role Playing Game with XNA Game Studio 3.0
# Part 20
# Cleaning Things Up

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 19 You can download the graphics from this link: Graphics.zip

This is going to be another long tutorial as I had done quite a lot of coding again. I added a new screen to Eyes of the Dragon to display the player character's statistics. I also made changes to the **CreatePCScreen**, **PlayerCharacter**, **FighterCharacter**, **ThiefCharacter**, **PriestCharacter** and **WizardCharacter**. I made a changes to the **CharacterAbilities** class, the **DifficultyPopUpScreen** and the **Game1** class. I also added in a new class for the difficulty levels. I will be getting to the new screen last because

I changed a lot of the graphics for this tutorial. I decide to zip all of the graphics from their folders in the **Content** folder. The first thing you will want to is download the Graphics.zip file and extract the contents from it into your **Content** folder.

I will start with new class that I added in for the difficulty levels of the game. Right click the **PlayerCharacter** folder and add a new class called **Difficulty**. As always when giving you a new class I will give you the new code and then explain why I have done what I did. This is the code for the **Difficulty** class.

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

namespace New2DRPG
{
    public class Difficulty
    {
        public enum Level { Easy = 0, Normal = 1, Hard = 2, Ultimate = 3};

        static float[] experience = { 1.25f, 1.0f, 0.85f, 0.7f };
        static float[] abilities =  { 0.75f, 1.0f, 1.25f, 1.5f };

        static int pointsPerLevel = 4;

        public static float[] Experience
        {
            get { return experience; }
        }

        public static float[] Abilities
        {
            get { return abilities; }
        }
        public static int Length
        {
            get { return Enum.GetNames(typeof(Level)).Length; }
        }

        public static int PointsPerLevel
        {
            get { return pointsPerLevel; }
        }

        public static Level DifficultyLevel(int index)
        {
            if (index >= 0 && index < (int)Level.Ultimate)
                return (Level)index;
            throw new System.IndexOutOfRangeException();
        }
    }
}
```

The first thing I did was change the namespace to **New2DRPG**. You will see that everything in this class is static except the **enum** which can't be static but can already be accessed using the class name. You may find this a little confusing and wonder what I am doing. The reason is that I never need to create an instance of this class to use it in the game. I just need to be able to access the values of the class.

The next thing that you will see is there is an **enum** called **Level**. This **enum** has the different difficulty levels that I will be using in the game: **Easy**, **Normal**, **Hard** and **Ultimate**. Next there are two static arrays of floats **experience** and **abilities**.

The first array **experience** has the modifiers for the player character's experience. These values will be multiplied by the experience the player earns for wins in battle. Since **Normal** is the base mode experience is multiplied by 1 giving the character no advantage or penalty. To make the **Easy** mode character gain experience faster I multiply the experience by 1.25 giving them a 25% bonus to experience earned. To make the **Hard** and **Ultimate** mode characters gain experience more slowly I will multiply their experience by 0.85 and 0.7 giving the **Hard** mode characters a 15% penalty and the **Ultimate** mode characters a 30% penalty.

The second array **abilities** has modifiers for the points the player character will get when they gain a level. This array will work in the reverse order of the previous array to entise the player to choose the harder mode character. There is a static int variable **pointsPerLevel** that give the points the player will get to distribute to their ability scores when they gain a level. I chose 4 as a good number. For an **Easy** mode character the multiplier is 0.75 giving them only 3 points per level. **Normal** mode, being the base mode, has a multiplier of 1. **Hard** and **Ultimate** values are 1.25 and 1.5 giving the **Hard** mode character 5 and the **Ultimate** mode character 6. So you see the **Ultimate** mode character will be far stronger at level 10 than an **Easy** mode character. The **Ultimate** mode character will have gained 60 points and the **Easy** mode character will have only gained 30. So the reward for playing the **Ultimate** mode character is worth the experience penalty.

There are three static get only properties in this class. The first one, **Experience**, is used to retrieve the **experience** array that has the modifiers for experience gained by the player's character. The second one, **Abilities**, does the same for the **abilities** array. The last one **PointsPerLevel** is used to access how many points the player character gets per level.

There is one static method in this class as well. In this method I accept as a parameter an integer index value that is between 0 and the integer value of the highest level. It then returns a **Level** that was casted from the index. If the index is out of range I throw an exception.

I made a change to the constructor of the **DifficultyPopUpScreen** class to use this new class. All I did was use the **Level enum** to get the items for the menu. This is the new constructor.

```
public DifficultyPopUpScreen(Game game)
    : base(game)
{
    LoadContent();

    Components.Add(new BackgroundComponent(game, image, false));

    imagePosition = new Vector2();
    imagePosition.X = (game.Window.ClientBounds.Width - image.Width) / 2;
    imagePosition.Y = (game.Window.ClientBounds.Height - image.Height) / 2;

    string[] items = Enum.GetNames(typeof(Difficulty.Level));

    menu = new ButtonMenu(game, spriteFont, buttonImage);
    menu.SetMenuItems(items);
    Components.Add(menu);
}
```

Then I made a few changes to the **CreatePCScreen** to use this new class as well. The first change was the **difficultyLevels** variable. Instead of setting them implicitly I set them using the **Level enum** from the **Difficulty** class and I also added in a variable of the **Level enum**.

```
string[] difficultyLevels = Enum.GetNames(typeof(Difficulty.Level));
Difficulty.Level difficultyLevel;
```

I changed the constructor to set the **difficultyLevel** variable. I also added in a get only property **DifficultyLevel** to get the **difficultyLevel** variable from the **CreatePCScreen** class. The last change was in the **ChangeDifficulty** method. To that method I set the **difficultyLevel** variable using the index from the menu. These are the new constructor, property and method.

```
public CreatePCScreen(Game game)
    : base(game)
{
    LoadContent();
    difficultyLevel = Difficulty.DifficultyLevel(difficultyIndex);
    Components.Add(new BackgroundComponent(game, background, true));

    buttonMenu = new ButtonMenu(game, spriteFont, buttonImage);
    buttonMenu.SetMenuItems(menuItems);
    Components.Add(buttonMenu);

    screenWidth = Game.Window.ClientBounds.Width;
    screenHeight = Game.Window.ClientBounds.Height;
}

public Difficulty.Level DifficultyLevel
```

```
{
    get { return difficultyLevel; }
}

public void ChangeDifficulty(int difficultyIndex)
{
    this.difficultyIndex = difficultyIndex;
    difficultyLevel = Difficulty.DifficultyLevel(difficultyIndex);
}
```

I also made a quite few changes to the **CharacterAbilities** class. Since I had made so many changes and it is different enough I will give you the code for the new class and then go over the changes. This is the new **CharacterAbilities** class.

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

namespace New2DRPG
{
    class CharacterAbilities
    {
        public enum Abilities
        {
            Strength = 0,
            Stamina = 1,
            Agility = 2,
            Speed = 3,
            Intellect = 4,
            Luck = 5
        };

        const float minAbilityScore = 1f;
        const float maxAbilityScore = 200f;

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

        public CharacterAbilities()
        {
            foreach(Abilities a in Enum.GetValues(typeof(Abilities)))
                SetAbilityScore(a, 10);
        }

        public int this[Abilities index]
        {
            get { return GetAbilityScore(index); }
            set { SetAbilityScore(index, value); }
        }

        public int this[int index]
        {
```

```
    get { return GetAbilityScore((Abilities)index); }
    set { SetAbilityScore((Abilities)index, value); }
}

public int Length
{
    get
    {
        return Enum.GetNames(typeof(Abilities)).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);
            }
        }
        private int GetAbilityScore(Abilities index)
        {
            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 System.IndexOutOfRangeException();
        }
        private void SetAbilityScore(Abilities index, int value)
        {
            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 System.IndexOutOfRangeException();
        }
    }
}
```

The first thing that I did was remove the string array that held the names of the attributes. I can get them instead using the **Enum.GetNames** method. Not a signifcant change but I changed the **maxAbilityScore** constant to 200f.

I added in a default constructor to the class. In the constructor there is just a **foreach** loop. In the **foreach** loop I get the **values** of the **Abilities enum** using the **Enum.GetValues** method. This method works like the **Enum.GetNames** method but instead of returning an array of strings it returns an array of **enum values**. Inside the **foreach** loop I call a method called **SetAbilityScore** passing in the **enum value** and 10, the value that I decided to be what an average person would have.

I also changed the **indexers** in this class and there are two of them. One of them takes an **Abilities enum** as it's index. In the get part I return a value from a method that I created called **GetAbilityScore** that takes an **Abilities enum** as a parameter and returns an integer. In the set part I call the method **SetAbilityScore** that I used above passing in the **Abilities enum** and the **value** sent to the **indexer**.

The second **indexer** takes an integer as it's index. It does the same thing as the first **indexer**. It calls the **GetAbilityScore** method with the index cast as an **Abilities** variable and returns it. The set part works the same as well. It calles the **SetAbilityScore** method after it casts the index to an **Abilities** variable and the **value** sent to the **indexer**.

The **Length** property changes as well. Since I no longer have the array of strings for the ability score names I had to find another way to get the lenght. Fortunately I can use the **Enum.GetNames** method to return an array and use the **Length** property of an array to return the length. The other properties didn't need to change.

All that is left for this class is the **GetAbilityScore** and **SetAbilityScore** methods. I will start with the **GetAbilityScore** method. In this method I simply copied what was in the old indexer that I used to return the proper ability score for the corrosponding **Abilities enum** or throw an exception if there is no value. The **SetAbilityScore** method was created the same way, simply copying the code for the old **indexer** and pasting it into the new method. That is all for the **CharacterAbilities** class.

I also made a few changes to the **PlayerCharacter** class and the classes that inherit form it: **FighterCharacter**, **ThiefCharacter**, **PriestCharacter** and **WizardCharacter**. In the inherited classes I just changed the constructors. I will start with the **PlayerCharacter** class. Much of it didn't change but I will give you the whole class and then go over the changes. This is the new version of the **PlayerCharacter** class.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace New2DRPG
{
    class PlayerCharacter : Microsoft.Xna.Framework.DrawableGameComponent
    {
        public enum CharClass { Fighter = 0, Wizard = 1, Priest = 2, Thief = 3};
        static string[] classNames = Enum.GetNames(typeof(CharClass));

        protected string name;
        protected bool gender;

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

        protected Difficulty.Level difficulty;
        protected int level;
        protected ulong gold;
        protected ulong experience;

        protected CharacterAbilities abilities = new CharacterAbilities();

        protected string className;
        protected Game game;

        protected SpriteBatch spriteBatch;
        protected ContentManager Content;

        protected Texture2D hpspDisplay;
        protected Texture2D blueBar;
        protected Texture2D redBar;

        protected SpriteFont spriteFont;
        protected Vector2 hpspPosition;

        public PlayerCharacter(Game game)
            : base(game)
        {
            this.game = game;

            spriteBatch =
                (SpriteBatch)Game.Services.GetService(typeof(SpriteBatch));
            Content =
                (ContentManager)Game.Services.GetService(typeof(ContentManager));

            difficulty = Difficulty.Level.Normal;
            level = 1;
            experience = 0;
            gold = 0;

            hpspDisplay = Content.Load<Texture2D>(@"GUI\hpspborder");
            blueBar = Content.Load<Texture2D>(@"GUI\bluebar");
            redBar = Content.Load<Texture2D>(@"GUI\redbar");
            spriteFont = Content.Load<SpriteFont>(@"smallFont");
            hpspPosition = new Vector2(10, 10);
        }

        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 int Level
{
    get { return level; }
}

public ulong Gold
{
    get { return gold; }
}

public ulong Experience
{
    get { return experience; }
}

public Difficulty.Level DifficultyLevel
{
    get { return difficulty; }
}

public virtual CharacterAbilities Abilities
{
    get { return abilities; }
}

public void Show()
{
    Visible = true;
}

public void Hide()
{
    Visible = false;
}

public override void Draw(GameTime gameTime)
{
    DrawHitPointsSpellPoints();

    base.Draw(gameTime);
}

private void DrawHitPointsSpellPoints()
{
    Vector2 barPosition = new Vector2();
    Vector2 textPosition = new Vector2();
    Vector2 stringSize;
    string text;
    Color tintColor = Color.White;
    tintColor.A = 128;

    spriteBatch.Draw(hpspDisplay, hpspPosition, tintColor);

    barPosition = hpspPosition + Vector2.One * 5;
    spriteBatch.Draw(redBar, barPosition, tintColor);

    textPosition = barPosition;
    textPosition.Y -= 5;
```

```
                text = hitPoints[0].ToString() + "/" + hitPoints[1].ToString();
                stringSize = spriteFont.MeasureString(text);
                textPosition.X += (200 - stringSize.X) / 2;
                textPosition.Y += (20 - spriteFont.LineSpacing) / 2;
                spriteBatch.DrawString(spriteFont, text, textPosition, tintColor);

                barPosition.Y += 19;
                spriteBatch.Draw(blueBar, barPosition, tintColor);

                textPosition = barPosition;
                textPosition.Y -= 5;
                text = spellPoints[0].ToString() + "/" + spellPoints[1].ToString();
                stringSize = spriteFont.MeasureString(text);
                textPosition.X += (200 - stringSize.X) / 2;
                textPosition.Y += (20 - spriteFont.LineSpacing) / 2;
                spriteBatch.DrawString(spriteFont, text, textPosition, tintColor);
            }
        }
    }
```

The first thing I did was remove the **enum** from this class because I now have the **Difficulty** class to use. I also added in four new protected variables: **difficulty**, **level**, **gold** and **experience**. **difficulty** will hold the difficulty level the player is playing in. **level** is an integer that holds the player character's level. **gold** is an unsigned long that will hold how much gold the player character has. **experience** holds how much experience the player character has.

I changed the constuctor of the class to set these new protected variables. In the constructor for the class I set **level** to 1 and **gold** and **experience** to 0. I also set **difficulty** to **Level.Normal**.

There are four new public get only properties in the class. **Level** is used to access the **level** variable. **Gold** is used to access the **gold** variabled. **Experience** is used to access the **experience** variable. I had to call the next one **DifficultyLevel** because just **Difficulty** would conflict with the **Difficulty** class. **DifficultyLevel** is of course used to access the **difficulty** variable in this class.

I also changed the constructors for all of the player character classes: **FighterCharacter**, **PreistCharacter**, **WizardCharacter** and **ThiefCharacter**. I added in a new parameter to the constructors of the classes for the difficulty level that the player is playing in. I also set the **difficulty** variable to the value passed in to the constructor. Since that is the only changes to these classes I will just give you the new constructors for the classes.

```
public FighterCharacter(string name,
        bool gender,
        Difficulty.Level difficulty,
        Game game)
    : base(game)
{
    this.difficulty = difficulty;
    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;
    }

    public PriestCharacter(string name,
            bool gender,
            Difficulty.Level difficulty,
            Game game)
        : base(game)
    {
        this.difficulty = difficulty;
        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;
    }
```

```
public ThiefCharacter(string name,
        bool gender,
        Difficulty.Level difficulty,
        Game game)
    : base(game)
{
    this.difficulty = difficulty;
    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;
}

public WizardCharacter(string name,
        bool gender,
        Difficulty.Level difficulty,
        Game game)
    : base(game)
{
    this.difficulty = difficulty;
    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;
}
```

Now I will turn my attention to the **Game1** class to finish off the changes to the classes related to the **PlayerCharacter** class. I first had to change the **CreatePlayerCharacter** method. Since the new parameter added to the constructors is the third parameter when I call the constructor of the different classes I use the **DifficultyLevel** property of the **CreatePCScreen** object **createPCScreen**. Below is the updated method **CreatePlayerCharacter**.

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

I also had to change the **HandleStartScreenInput** method because if you select the second option in the start screen menu it creates a default character. I just passed in **Difficulty.Level.Normal** as the third parameter. 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,
                    Difficulty.Level.Normal,
                    this);
                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;
            }
        }
    }
```

I made one small change to the **ButtonMenu**. White does not show up well on the new buttons so I changed the hilite color from **Color.White** to **Color.Yellow**.

```
Color hiliteColor = Color.Yellow;
```

Like I said I also made new screen for the game to display the player character's statistics. Right click the **Screens** folder and add a new class called **ViewCharacterScreen**. I will give you the code for the class and then explain it.

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

namespace New2DRPG
{
    class ViewCharacterScreen : GameScreen
    {
        ButtonMenu menu;
        Texture2D image;
        Texture2D buttonImage;
        SpriteFont spriteFont;
        Vector2 imagePosition;
        PlayerCharacter playerCharacter;

        public ViewCharacterScreen(Game game)
            : base(game)
        {
            LoadContent();

            Components.Add(new BackgroundComponent(game, image, false));

            imagePosition = new Vector2();
            imagePosition.X = (game.Window.ClientBounds.Width - image.Width) / 2;
            imagePosition.Y = (game.Window.ClientBounds.Height - image.Height) / 2;

            string[] items = { "OK" };
            menu = new ButtonMenu(game, spriteFont, buttonImage);
            menu.SetMenuItems(items);
            Components.Add(menu);
        }

        protected override void LoadContent()
        {
            image = Content.Load<Texture2D>(@"GUI\popupbackground");
            buttonImage = Content.Load<Texture2D>(@"GUI\buttonbackgroundshort");
            spriteFont = Content.Load<SpriteFont>(@"normal");
            base.LoadContent();
        }

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

            Vector2 position = new Vector2();
            Vector2 stringSize = new Vector2();
            string text;
            float maxLength = 0.0f;

            text = playerCharacter.Name;
            stringSize = spriteFont.MeasureString(text);

            position.Y = imagePosition.Y + 10;
            position.X = imagePosition.X + (image.Width - stringSize.X) / 2;

            DrawText(text, position);

            position.Y += 10;

            string[] abilityWords =
                  Enum.GetNames(typeof(CharacterAbilities.Abilities));
            CharacterAbilities abilities = playerCharacter.Abilities;
            int[] scores = new int[abilities.Length];

            for (int i = 0; i < scores.Length; i++)
            {
                scores[i] = abilities[i];
            }

            foreach (string s in abilityWords)
            {
                text = s + ": ";
                stringSize = spriteFont.MeasureString(text);
                if (stringSize.X > maxLength)
                    maxLength = stringSize.X;
            }

            for (int i = 0; i < abilities.Length; i++)
            {

                text = abilityWords[i] + ": ";
                position.Y += spriteFont.LineSpacing;
                position.X = imagePosition.X + 15;

                DrawText(text, position);

                position.X += maxLength;

                text = scores[i].ToString();
                DrawText(text, position);
            }


            string[] statNames = {
                    "Class: ",
                    "Level: ",
                    "HP: ",
                    "SP: ",
                    "XP: ",
                    "Gold: " };
            string[] statValues = {
                    playerCharacter.ClassName,
                    playerCharacter.Level.ToString(),
                    playerCharacter.HitPointsCurrent.ToString() + "/" +
                        playerCharacter.HitPointsMax.ToString(),
                    playerCharacter.SpellPointsCurrent.ToString() + "/" +
                        playerCharacter.SpellPointsMax.ToString(),
                    playerCharacter.Experience.ToString(),
                    playerCharacter.Gold.ToString() };

            maxLength = 0.0f;

            foreach (string s in statNames)
            {
                stringSize = spriteFont.MeasureString(s);
                if (stringSize.X > maxLength)
                    maxLength = stringSize.X;
            }

            position.X = imagePosition.X + (image.Width / 2);
            position.Y = imagePosition.Y + 20;

            for (int i = 0; i < statNames.Length; i++)
            {
                position.Y += spriteFont.LineSpacing;
                position.X = imagePosition.X + (image.Width / 2);

                text = statNames[i];
                DrawText(text, position);

                position.X += maxLength;

                text = statValues[i].ToString();
                DrawText(text, position);
            }
        }

        private void DrawText(string text, Vector2 position)
        {
            Vector2 shadow = new Vector2();
```

```
            shadow = position + Vector2.One;

            spriteBatch.DrawString(spriteFont, text, shadow, Color.Black);
            spriteBatch.DrawString(spriteFont, text, position, Color.Yellow);
        }

        public override void Show()
        {
            base.Show();
            menu.Position = new Vector2((image.Width -
                                 menu.Width) / 2 + imagePosition.X,
                                 image.Height - menu.Height - 10 + imagePosition.Y);
        }

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

This class inherits from **GameScreen** so I will need using statements for the XNA Framework, Graphics and Content namespaces. I changed the namespace to just **New2DRPG**. On this screen I have a **ButtonMenu**, an image, the image of the button for the **ButtonMenu**, I need to draw text so there is a **SpriteFont** object, like the other pop up type screens there is a **Vector2** for the position of the image on the screen and since I will be draw the play character's statistics I will need the **PlayerCharacter** object.

The constructor for this class is the same as all of the other screens. It calls the **LoadContent** method to load in the content. It adds a **BackgroundComponent** to the list of components passing false for the fill parameter. It then sets the position of the image to the center of the screen. Finally it adds a one button menu to allow the screen to close.

The **LoadContent** method loads in a graphics that I created that has no text in it like the other pop up screens. It also loads in the smaller of the buttons and the sprite font.

There is an override of the **Draw** method. Since the order of drawing things is important is 2D I draw everything after the call to **base.Draw** so things will be drawn in the right order. There is a lot of positioning of things in this method. I have used a few tricks to make writing things a little easier and not have to position every item individually.

There are a few local variables that I use a lot in this method. There are two **Vector2s position** and **stringSize**. The names will give you a clue to their purpose. **position** is of course where I will be drawing the items. **stringSize** will be used to measure strings to figure out where to draw the items. **text** is just a variable that I use to hold the string I want to draw. The float **maxLength** is used to help align the strings into columns that line up nicely.

The first string that I write out is the player character's name. I set the **text** variable to the player character's name and use the **MeasureString** method of the **SpriteFont** object. Next I set the position where I will be drawing the string. To find the **Y** coordinate I take the **Y** coordinate of the position of the image and add 10 pixels. I added the 10 pixels because of how I made the background image. For the **X** coordinate I take the **X** coordinate of the image and add the width of the image minus the width of the string divided by 2 centering the text in the image. Then I call a method I wrote **DrawText** passing it **text** and **position**. Then I added 10 pixels to the **Y** value of **position** to add a bit of spacing between the name and the rest of the items.

Now I do a little magic so I don't have to write the ability score names and values individually and can do it inside of a loop. I create an array of strings **abilityWord** and fill it using the **Enum.GetNames** method passing it the **Abilities enum** of the **CharacterAbilities** class. I then created a **CharacterAbilities abilities** object and set it to the **CharacterAbilities** object of the **PlayerCharacter**. I then create an array of integers **scores** to hold the ability scores of the character. Next you will see why I created the **indexer** of the **CharacterAbilities** class. In a **for** loop I loop through the number of ability scores and set the **scores** variable to the corrosponding ability score of the player character.

In the **foreach** loop I find out the longest length of the ability score names plus a colon and a space. I do this by setting the **text** variable to the current string plus a colon and a space. I use the **MeasureString** method of the **SpriteFont** object. If the **X** value of the result is greater than the **maxLength** variable, which I set to 0 initially, I set the **maxLength** variable to the value. This will allow me to align all of the ability scores in a column on the right of the ability names.

Now I am ready to draw the ability score names and scores. I use a **for** loop to loop through all of the ability scores. I first set the **text** variable to the ability score name plus a colon and a space. I add the **LineSpacing** property of the **SpriteFont** object to the **Y** value of the **position** variable and set the **X** value to the **X** value of the position of the image plus 15 pixels. I call the **DrawText** method passing the **text** and **position** variables. I then add the **maxLength** variable to the **X** value of the **position** variable. Set the **text** variable to the score variable and call **DrawText** passing **text** and **position**.

There was no real easy way to get the rest of the items that I wanted to draw but I used a similar method to get the items. I created an array of strings **statNames** to hold these strings: "**Class: **", "**Level: **", "**HP: **", "**SP: **", "**XP: **" and "**Gold: **". Then I create another array of strings **statValues** to hold the values for these items. For the first one I use the **ClassName** property of the **PlayerCharacter** object. The second one I was able to use the **Level** property of the **PlayerCharacter** object. For hit points and spell points I use the current value property of the **PlayerCharacter** object plus a slash and then the maximum value property of the **PlayerCharacter** object. For the experience and gold values I use the properties of the **PlayerCharacter** object as well.

I then find the maximum length of the first array like I did previously. Then I set the initial **Y** value of **position** to the **Y** value of the image plus 20 pixels. I know that didn't account for the position of the name in the screen, I will account for that when I do the loop to draw the items.

There is a **for** loop that loops through all of the items. The inside of the loop does the same as the previous loop. It adds the **LineSpacing** property of the **SpriteFont** object to **Y** value of **position**. It then sets the **X** value of **position** to the **X** value of the image plus half the width of the image. It sets the **text** variable to the **statsName** variable and calls the **DrawText** method passing the **text** variable and **position** variable. Then I add **maxLength** to the **X** value of **position** and set **text** to **statVaules** and call **DrawText** passing in **text** and **position**.

In the **DrawText** method there is a local variable called **shadow** that is a **Vector2**. This variable will draw a shadow under the text. **shadow** is set to **position** plus **Vector2.One** (1.0f, 1.0f). The shadow is drawn first with **Color.Black** because it has to be drawn behind the text. Then the text is drawn in **Color.Yellow**.

There is an override of the **Show** method to position the menu. It sets the position like all of the other screens. It centers it horizontally and then places it near the bottom of the image.

There is one other method **SetPlayerCharacter** that take a **PlayerCharacter** object as a parameter. This method just sets the **playerCharacter** object in the class.

All that is left is to update the **Game1**class to handle the new screen. The first thing I did was add in a variable for the new screen called **viewCharacterScreen**.

```
ViewCharacterScreen viewCharacterScreen;
```

In the **LoadContent** method I call a method that I wrote **LoadViewCharacterScreen**. This is the updated **LoadContent** method and the new **LoadViewCharacterScreen** 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();
    LoadViewCharacterScreen();

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

    activeScreen = introScreen;
    activeScreen.Show();
}

private void LoadViewCharacterScreen()
{
    viewCharacterScreen = new ViewCharacterScreen(this);
    Components.Add(viewCharacterScreen);
    viewCharacterScreen.Hide();
}
```

I changed the **HandleActionScreen** method to be able to display the new screen. I have set so that if the player presses **V** it will show the **ViewCharacterScreen**. **activeScreen.Enabled** is set to false so the **Update** method won't be called but the **Draw** method will be called. **activeScreen** is set

**viewCharacterScreen**. I call the **SetPlayerCharacter** of **viewCharacterScreen** to set the **PlayerCharacterObject**. Finally I call the **Show** method of **activeScreen**.

```
private void HandleActionScreeenInput()
{
    if (CheckKey(Keys.Escape))
    {
        this.Exit();
    }
    if (CheckKey(Keys.V))
    {
        activeScreen.Enabled = false;
        activeScreen = viewCharacterScreen;
        viewCharacterScreen.SetPlayerCharacter(playerCharacter);
        activeScreen.Show();
    }
}
```

I had to change the **Update** method. I needed to be able to handle the input for the new screen. I had created a new method **HandleViewCharacterScreenInput**. In the **Update** method I added in an **else if** condition to call the **HandleViewCharacterScreenInput** method if **activeScreen** is the **viewCharacterScreen**. In the **HandleViewCharacterScreenInput** all I do is check if the **Enter** key has been pressed. If the **Enter** key has been pressed I hide the active screen, set the active screen to the action screen and finally show the action screen. This is the **HandleViewCharacterScreenInput** method and the new **Update** method.

```
protected override void Update(GameTime gameTime)
{
    newState = Keyboard.GetState();

    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
        this.Exit();

    if (activeScreen == startScreen)
    {
        HandleStartScreenInput();
    }
    else if (activeScreen == helpScreen)
    {
        HandleHelpScreenInput();
    }
    else if (activeScreen == createPCScreen)
    {
        HandleCreatePCScreenInput();
    }
    else if (activeScreen == quitPopUpScreen)
    {
        HandleQuitPopUpScreenInput();
    }
    else if (activeScreen == genderPopUpScreen)
    {
        HandleGenderPopUpScreenInput();
    }
    else if (activeScreen == classPopUpScreen)
    {
        HandleClassPopUpScreenInput();
    }
    else if (activeScreen == difficultyPopUpScreen)
    {
        HandleDifficultyPopUpScreenInput();
    }
    else if (activeScreen == nameInputScreen)
    {
        HandleNameInputScreenInput();
    }
    else if (activeScreen == introScreen)
    {
        HandleIntroScreenInput();
    }
    else if (activeScreen == creditScreen)
    {
        HandleCreditScreenInput();
    }
    else if (activeScreen == actionScreen)
    {
        HandleActionScreeenInput();
    }
    else if (activeScreen == viewCharacterScreen)
    {
        HandleViewCharacterScreenInput();
    }
    oldState = newState;

    base.Update(gameTime);
}

private void HandleViewCharacterScreenInput()
{
    if (CheckKey(Keys.Enter))
    {
        activeScreen.Hide();
        activeScreen = actionScreen;
        actionScreen.Show();
    }
}
```

Well that is it for this tutorial. I am already working on coding the next part of Eyes of the Dragon so I encourage you to keep either visiting my site http://xna.jtmbooks.com or my blog, http://xna-rpg.blogspot.com for the latest news on these tutorials.