Creating a Role Playing Game with XNA Game Studio 3.0
Part 13
Adding Two New Screens

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

For this tutorial you will need the new graphics that I have created. You can find them in the graphics file for these tutorials mentioned above. If you are not interested in typing the tutorial but just reading it you can just download the latest source code of Eyes of the Dragon.

I mentioned in my blog that I was working on my entry in the latest Game Institute game challenge called Space Raptor. I did two pretty interesting effects and I thought you all might be interested in them. The first is in the introduction screen to the game. The game displays a back ground image and the text for the introduction scrolls from the bottom of the screen to the top and then fades from being fully opaque to fully transparent. When the fade is finished the games goes to the start menu.

The other effect I did was I had a screen where there is a background and the text goes from fully transparent to fully opaque. This screen had a single button menu on it. I liked both effects so I thought I would add them to Eyes of the Dragon. I pretty much just plucked the screens from Space Raptor and added them to Eyes of the Dragon with a few little tweeks.

To get started, you will need the four graphics that I created. They are in the Graphics.zip file. Create a new folder in the **Content** folder called **Backgrounds**. In this folder you will need to add the following images:

- credits.png
- creditsbackground.jpg
- introduction.png
- introbackground.jpg

I will start with the screen for the introduction to the game. Add a new class to the project called **IntroScreen**. As I always do, I will show you the new code and then explain what I've done.

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

namespace New2DRPG
{
    class IntroScreen : GameScreen
```

```
{
    Texture2D introduction;
    TimeSpan myTimeSpan;
    ContentManager Content;
    SpriteBatch spriteBatch;
    Vector2 position;
    bool fadeFinished = false;
    bool topReached = false;
    byte alphaValue = 254;
    Color tintColor = Color.White;

    public IntroScreen(Game game)
        : base(game)
    {
        Texture2D background;

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

        background = Content.Load<Texture2D>(@"Backgrounds\introbackground");
        introduction = Content.Load<Texture2D>(@"Backgrounds\introduction");
        Components.Add(new BackgroundComponent(game, background));
        position = new Vector2(0, Game.Window.ClientBounds.Height);
    }

    public bool IntroFinished
    {
        get { return fadeFinished; }
    }

    public override void Update(GameTime gameTime)
    {
        myTimeSpan += gameTime.ElapsedGameTime;
        if (myTimeSpan > TimeSpan.FromMilliseconds(25) && !topReached)
        {
            position.Y -= 2.0f;
            myTimeSpan -= TimeSpan.FromMilliseconds(25);
            if (position.Y < 0.0f)
            {
                position.Y = 0.0f;
                topReached = true;
            }
        }
        if (myTimeSpan > TimeSpan.FromMilliseconds(15) && topReached)
        {
            if (alphaValue > 0)
                alphaValue--;
            else
                fadeFinished = true;
            tintColor.A = alphaValue;
            myTimeSpan -= TimeSpan.FromMilliseconds(15);
        }
        base.Update(gameTime);
    }

    public override void Draw(GameTime gameTime)
    {
```

```
            base.Draw(gameTime);
            spriteBatch.Draw(introduction, position, tintColor);
        }
    }
}
```

I of course had to add some using statements to the class. I used the **BackgroundComponent** and I inherited the class from **GameScreen** so I needed to add using statement for the **CoreComponents**. I needed the XNA Framework, a **ContentManager** and this is visual component so I needed to add using statements for the framework, content and graphics.

There are a lot of variables in this class. There is a **Texture2D** for the image of the introduction. This variable might be new to some of you. There is a **TimeSpan** object. A TimeSpan object is used to measure how much time has elapsed since a certain point. I will use this variable to know when to move the image up the screen and when to reduce the transparency of the image.

The next three variables will be familiar. There is a **ContentManager** object for loading in textures. There is a **SpriteBatch** object for drawing and a **Vector2** for the position of the introduction image on the screen. The rest of the variables are new, in a way. There are two boolean variables. There is one to tell if the fade effect is finished and one to tell if the introduction image has reached the top of the screen.

Then next two variables may need more than a little explanation. There is a byte variable that I will use to determine the alpha channel for the tint color. The tint color variable will be used for making the introduction image transparent. As you probably know, when you are drawing an object in 2D you can tint the object using a color. Using white does not tint the image but keeps it the same. You can modify the alpha value of the tint color to make the image more or less transparent. To keep track of how transparent the image is I needed a variable for the color.

Now it is time to move on to the constructor. This should all be fairly familiar to you if you have followed along with these tutorials. The constructor only takes one parameter. A **Game** object to pass to the parent class. There is a variable to hold the images that will be loaded. I retrieve the **SpriteBatch** and **ContentManager** objects that I stored in the **Game1** class. I load in the images I created for the background and the introduction. I add a **BackgroundComponent** to the list of components using the background I just loaded and position the image for the introduction at the bottom of the screen.

There is a public get only property in this class, **IntroFinished**. I will use this property in the **Game1** class to determine if the introduction has finished playing.

Much of **Update** method might be new to some of you. The first thing I do is add the elapsed time the game has been running to the **TimeSpan** object using the **ElapsedGameTime** property of the **GameTime** object. Next there is an if statement. In this if statement I check to see if the amount of time that has passed is greater than 25 millisecond and the introduction image hasn't reached the top of the screen.

Inside that if statement I update the position of the image and I deduct the elapsed time in the game from the time span. This basically resets the time span back to it's original value. If the position of the image is past the top of the screen I reset it to the top of the screen and set the variable to say the image has reached the top of the screen.

The second if statement is pretty much the same as the first. It checks to see if a certain amount of time has passed and that the image is at the top of the screen and it is time to start to fade the image. Inside this if statement I check to see if the alpha value for the tint color is greater than 0. If it is greater than zero I dcrement the value. If it is 0 then the fading effect is finished and I set the **fadeFinished** variable to true. Now I set the alpha channel of the tint color to be the new alpha value. Finally I deduct the elapsed game time from the time span.

The only other thing that I did in this class is override the **Draw** method. After drawing the parent and all of the visible game components, ie the background image, I can draw the introduction image on the screen. I will add in the code needed to use this screen into the **Game1** class after I have given you the code for the **CreditScreen** class and exlpained it.

The **CreditScreen** is very much like the **IntroScreen** with three differences. The first is that the text does not scroll up the screen like in the **IntroScreen**. The second is that the text starts transparent and changes to fully opaque. The last difference is that there is a single button at the bottom of the screen that says **RETURN TO MENU**. You will want to add a new class to the project and call it **CreditScreen**. As I always do, I will give you the new code and then explain why I did what I did. This is the code for the **CreditScreen**.

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

namespace New2DRPG
{
    class CreditScreen : GameScreen
    {
        Texture2D credits;
        TimeSpan myTimeSpan;
        ContentManager Content;
        SpriteBatch spriteBatch;
        Vector2 position;
        bool fadeFinished = false;
        byte alphaValue = 0;
        Color tintColor = Color.White;
        ButtonMenu buttonMenu;

        public CreditScreen(Game game)
            : base(game)
        {
            Texture2D tempTexture;
            SpriteFont spriteFont;

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

            tempTexture =
                 Content.Load<Texture2D>(@"Backgrounds\creditsbackground");
```

```
            credits = Content.Load<Texture2D>(@"Backgrounds\credits");
            Components.Add(new BackgroundComponent(game, tempTexture));
            spriteFont = Content.Load<SpriteFont>("normal");
            tempTexture = Content.Load<Texture2D>(@"GUI\buttonbackground");
            string[] items = { "RETURN TO MENU" };

            buttonMenu = new ButtonMenu(game, spriteFont, tempTexture);
            buttonMenu.SetMenuItems(items);
            Components.Add(buttonMenu);

            position = new Vector2(0, 0);
        }

        public override void Update(GameTime gameTime)
        {
            myTimeSpan += gameTime.ElapsedGameTime;
            if (myTimeSpan > TimeSpan.FromMilliseconds(15))
            {
                if (alphaValue < 254)
                    alphaValue++;
                else
                    fadeFinished = true;
                tintColor.A = alphaValue;
                myTimeSpan -= TimeSpan.FromMilliseconds(15);
            }
            base.Update(gameTime);
        }

        public override void Draw(GameTime gameTime)
        {
            base.Draw(gameTime);
            spriteBatch.Draw(credits, position, tintColor);
        }

        public override void Show()
        {
            buttonMenu.Position = new Vector2((Game.Window.ClientBounds.Width -
                                        buttonMenu.Width) / 2, 700);
            alphaValue = 0;
            base.Show();
        }
    }
}
```

As you can see the using statements are all identical because basically they are the same screen with a few minor differences. This is a screen for the game so, like I just said, I inherited the class from **GameScreen**. The variables are pretty much the same as well. There is one for the image and one for a **TimeSpan** object. So I don't have to pass in textures I have a **ContentManager**. Since I will be drawing in this class there is a **SpriteBatch** object. A **Vector2** for the posititon of the image. I guess I really didn't need it but there is a boolean to tell when the fading is completed. There is a byte for the alpha channel and a tint color that will be white that I will start at transparent and make opaque. There is also a **ButtonMenu** object. That does it for variables.

The constuctor for this class also has just one parameter, the **Game** object. There are two local variables in the constructor. A **Texture2D** that will be used to load in the back ground image and the image of the button.

Just like I did with the **IntroScreen**, I get the **ContentManager** and **SpriteBatch** objects that were added to the list of game services. The I loaded in the back ground image for the screen and the actual credits. I add a **BackgroundComponent** to the list of **GameComponents** then I load in the font and the texture of the button. There is only one item in this menu, the option to return to the main menu. I created the button menu, set the menu items and added the menu to the list of components. Finally, the position of the texture is in the upper left hand corner so I set the position of the texture to (0, 0).

There are no properties in this class so I will move onto the **Update** method. Just like in the **Update** method of the **IntroScreen**, I add the elapsed game time to the time span. If the appropriate number of milliseconds have passed, I increment the alpha value of the tint color by one if it is less than 254 since the maximum value of a byte is 255. If the value is 255 I set the boolean variable to true. I then set the alpha channel of the tint color and deduct the appropriate number of millisecond from time span.

The **Draw** method is exactly the same as in the **IntroScreen**. It draw the parent components and then draws the texture for the credits.

I did add in an override of the **Show** method. First to set the position of the button menu as well as reset the alpha channel to 0 so each time the screen is displayed the texture will always start transparent and move to being fully opaque.

There was one minor change to the **StartScreen** class. I updated the menu items to include an option for displaying the **CreditScreen**. Otherwise there would be no way to display it.

These are the updated menu items of the **StartScreen**.

```
string[] items = { "THE STORY BEGINS",
                   "THE STORY CONTINUES",
                   "HELP",
                   "CREDITS",
                   "QUIT" };
```

I'm sure you have figured out already that the order in which the items appear is important. Otherwise when you are handling the input for that screen you would get the wrong results. What I am saying is that the order in which you have the menu items must be the same as the way you handle the index of the menu item.

Now it is time to turn your attention to the **Game1** class. The first thing you will have to do is add in variables for the two screens. These are the variables I created for the two screens.

```
CreditScreen creditScreen;
IntroScreen introScreen;
```

Next you will actually have to create the two screens and add them to the list of components for the game. I did this with all of the other screens in the **LoadContent** method. You will also want to set the first active screen to be the **IntroScreen** instead of the **StartScreen**. This is the updated **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");
createPCScreen = new CreatePCScreen(this, normalFont);
Components.Add(createPCScreen);

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

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

actionScreen = new ActionScreen(this, normalFont, "tileset1");
Components.Add(actionScreen);
actionScreen.Hide();

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

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);

background = Content.Load<Texture2D>(@"GUI\chooseclass");

classPopUpScreen = new PopUpScreen(this,
    normalFont,
    background,
    Content.Load<Texture2D>(@"GUI\buttonbackgroundshort"));
string[] classItems = new string[] { "Fighter", "Wizard", "Thief", "Priest" };
classPopUpScreen.SetMenuItems(classItems);
Components.Add(classPopUpScreen);

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);

    background = Content.Load<Texture2D>(@"GUI\choosename");

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

    creditScreen = new CreditScreen(this);
    Components.Add(creditScreen);
    creditScreen.Hide();

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

    introScreen = new IntroScreen(this);
    Components.Add(introScreen);
    activeScreen = introScreen;
    activeScreen.Show();
}
```

Again, the screens are created and added to the list of components just as all the other screens. Next I had to update the **Update** method to handle the input for the two new screens. I added in two new else ifs to call the methods to handle the input for the screens. I called them **HandleIntroScreenInput** and **HandleCreditScreenInput**. I also editted the if statement where I checked to see if the **Escape** key has been pressed. I don't want to exit the entire game if the escape key is pressed while the **IntroScreen** is being displayed. I just added in an and to check if the active screen is not the introduction screen. This is the updated **Update** method.

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

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

    if (newState.IsKeyDown(Keys.Escape) && activeScreen != introScreen)
        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();
    }
    oldState = newState;

    base.Update(gameTime);
}
```

After that I created the methods to handle the input for the two screens. I will give you the two methods and then explain what they both do. This is the code for the **HandleIntroScreenInput** and **HandleCreditScreenInput** methods.

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

private void HandleIntroScreenInput()
{
    if (CheckKey(Keys.Escape) || CheckKey(Keys.Space) || introScreen.IntroFinished)
    {
        activeScreen.Hide();
        activeScreen = startScreen;
        activeScreen.Show();
    }
}
```

The **HandleCreditScreenInput** method just checks to see if the enter key has been pressed. If it has it

hides the active screen, sets the active screen to the start screen and shows the active screen just like I did in the **HandleHelpScreenInput** method. The **HandleIntroScreenInput** method checks to see if the space or escape keys have been pressed or if the introduction screen has finished processing. Then it just hides the active screen, sets the active screen to the start screen then show the active screen.

There is one other thing that has to be changed. That is the **HandleStartScreenInput** method. It needs to be updated to reflect the changes made to the menu in the start screen. I just added in a new case for the credits screen and changed the case for the quit item to reflect. In the new case all I did was hide the active screen, set the active screen to the credit screen and finally show the active screen. This is the updated **HandleStartScreenInput** 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();
                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;
        }
    }
}
```

Well, that is it for this tutorial. I will try and have another tutorial available on the web site soon, so you can check out the blog that goes with these tutorials, http://xna-rpg.tutorials.com, for the latest news on these tutorials or the news page on my web site for the latest news on these tutorials.