Creating a Role Playing Game with XNA Game Studio 3.0
Part 9
Adding a Pop Up Menu

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. 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 8

In today's tutorial I'm going to be adding a pop up menu to the project. Quite often you need to get a yes or no answer from the player. Pop up menus are a good way to do that. What I will do is make a new class that will inherit from **GameScreen**. Go a head and add a new class to the project called **PopUpScreen**. As usual, I will give you the code 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 New2DRPG.CoreComponents;

namespace New2DRPG
{
    class PopUpScreen : GameScreen
    {
        ButtonMenu menu;
        Texture2D image;
        SpriteBatch spriteBatch;
        Rectangle imageRectangle;

        public PopUpScreen(Game game, SpriteFont spriteFont, Texture2D image,
            Texture2D buttonImage)
            : base(game)
        {
            this.image = image;
            spriteBatch =
                (SpriteBatch)game.Services.GetService(typeof(SpriteBatch));
            imageRectangle = new Rectangle();
            imageRectangle.X = (game.Window.ClientBounds.Width - image.Width) / 2;
            imageRectangle.Y =
                (game.Window.ClientBounds.Height - image.Height) / 2;
            imageRectangle.Width = image.Width;
            imageRectangle.Height = image.Height;

            string[] items = { "YES", "NO" };
            menu = new ButtonMenu(game, spriteFont, buttonImage);
            menu.SetMenuItems(items);
            Components.Add(menu);
        }

        public int SelectedIndex
```

```
        {
            get { return menu.SelectedIndex; }
        }

        public override void Draw(GameTime gameTime)
        {
            spriteBatch.Draw(image, imageRectangle, Color.White);
            base.Draw(gameTime);
        }

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

Since this is a visual game component I needed to add using statements for the XNA framework and the XNA graphics classes. I also had to add a using statement for the **CoreComponents** so I can inherit this screen from **GameScreen**.

This screen has only one game component, a **ButtonMenu** component. I didn't add the background image like I did with other screens because I want this background to float on top of the other screens and have the other screens visible but not updating. Otherwise this screen is just like the **StartScreen** component. There are four variables in this class: the **ButtonMenu** component, the image for the menu, a **SpriteBatch** object and a **Rectangle** object for where the image will be drawn on the screen.

The constructor takes four parameters: the **Game** ojbect, a **SpriteFont**, an image for the pop up screen and an image for the button. The constructor then sets the image for the component and gets the **SpriteBatch** object that was added to the list of services. Next the constructor finds where to draw the pop up menu on the screen. It takes half the heigh of the screen minus half the height of the image and does the same for the width. There are two items in this menu, **YES** or **NO**. The constructor then creates the menu, sets the menu items and adds it to the list of components.

There is one public property for this component. The property is used to get the selected item in the menu.

I added an override of the **Draw** method because I want to draw the image for the pop up screen. All the **Draw** method does is draw the image of the pop up screen using the **Rectangle** that was created in the constructor.
The only other thing I did in this component was override the **Show** method. Just like I did in the **StartScreen**, I set the location of the menu.

Instead of giving you all the changes to the **Game1** class, I will just give you what has changed and what is new. The first new thing is I added a variable for the pop up screen.

```
        PopUpScreen quitPopUpScreen;
```

I of course created the new screen in the **LoadContent** method. This is the **LoadContent** method:

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

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

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

    activeScreen = startScreen;
}
```

Before I could do that though, I had to add two new images to the **Content** folder. I added them in the **GUI** sub-folder. You can find them in the **Graphics** file. I also change the other image for the button to match the new, smaller button image. So, open the **GUI** sub-folder in the **Content** folder and add these images: **buttonbackground.jpg**, **buttonbackgroundshort.jpg** and **quitpopupbackground.jpg**. The pop up screen was created and added to the list of components just like the other screeens.

Of course there had to be a change made to the **Update** method to call a method to handle the input for the screen. There is no real mystery, I think, so I will just give you 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 (newState.IsKeyDown(Keys.Escape))
            this.Exit();

        if (activeScreen == startScreen)
        {
            HandleStartScreenInput();
        }
        else if (activeScreen == helpScreen)
        {
            HandleHelpScreenInput();
        }
        else if (activeScreen == createPCScreen)
        {
            HandleCreatePCScreenInput();
        }
        else if (activeScreen == quitPopUpScreen)
        {
            HandleQuitPopUpScreenInput();
        }
        oldState = newState;

        base.Update(gameTime);
    }
```

As you can see, the only change in the **Update** method was to handle the input for the new screen if it is the active screen.

To test the screen, I made a change to the **HandleStartScreenInput** method. Instead of exiting the game if the quit option is selected, I stop the start screen from updating by setting the **Enabled** property to false. I set the active screen to the pop up screen and show the pop up screen. This is the new **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.Enabled = false;
                activeScreen = quitPopUpScreen;
                activeScreen.Show();
                break;
        }
    }
}
```

There is one thing left to do to use this screen. I needed to create a method that would handle the input for the screen. I called it **HandleQuitPopUpScreenInput**. It is like the other methods to handle input for the other screens. This the code for the **HandleQuitPopUpScreenInput** method:

```
private void HandleQuitPopUpScreenInput()
{
    if (CheckKey(Keys.Enter) || CheckKey(Keys.Space))
    {
        switch (quitPopUpScreen.SelectedIndex)
        {
            case 0:
                this.Exit();
                break;
            case 1:
                activeScreen.Hide();
                activeScreen = startScreen;
                activeScreen.Show();
                break;
        }
    }
}
```

As you can see, all it does is check the selected index of the menu. If it is zero, the yes index, the game exits. If it is one, then it hides the screen, sets the start screen as the active screen and shows the start screen.

Well, that is all for this tutorial. I will get to writing another one soon. Keep on coming back to the blog or the web site and I will try and have new stuff on both of them.