

Creating a Role Playing Game with XNA Game Studio 3.0

Part 8

Nicer Screens and Menus

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 7](#)

So far the screens are functional and don't look all that bad. In today's tutorial I'm going to make them look a little better. The first thing I'm going to do is add two new back grounds to the content folder and get rid of the dragon and the gryphon images. You can find these images in this [graphics](#) zip file. Again, I made them with help from Genetica Viewer 3 by [Spiral Graphics](#). Download the two new screens, I'm going to work on the character screen another day, and add them to your **Content** folder. They are called **titlescreen.png** and **helpscreen.png**. There is one other new item in this file. It is an image for making the menus look nicer. It is called **buttonbackground.png**. While you are in the **Content** folder create a new folder called **GUI** and add this image to that folder.

Before I go any farther, there are a couple minor changes I made to the project. The first is I made a few modification to the constructor for **Game1**. I just made the screen a little bigger, set the title of the window and set the game to fullscreen mode. You have to be careful when you are working in fullscreen mode. If your program crashes, you might not be able to get to the debugger and end it. I suggest until you are sure your game works as you expect, keep to windowed mode. Well, here is the new constructor for this class.

```
public Game1 ()
{
    graphics = new GraphicsDeviceManager(this);
    graphics.PreferredBackBufferWidth = 1024;
    graphics.PreferredBackBufferHeight = 768;
    graphics.IsFullScreen = true;
    this.Window.Title = "Eyes of the Dragon";
    Content.RootDirectory = "Content";
}
```

I also made a small change to the tile engine. I changed the height and the width of the tiles. You don't really have to do this but I set them to these values:

```
static int tileWidth = 64;
static int tileHeight = 48;
```

Since most screens are 4:3 ratios I used that ratio for the tiles. I also use this because they will fit well in a 1024 by 768 pixel screen.

With all of that out of the way, let's get to making things a little nicer. The first thing I'm going to do is make a new menu component. This component will use the image that I made for buttons. In a later tutorial I will add in mouse support to the game and be able to click the buttons. To get started, you will want to add a new **GameComponent** to the **CoreComponents** folder called: **ButtonMenu**. As always

I will give you the new code and explain why I did what I did.

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

namespace New2DRPG.CoreComponents
{
    /// <summary>
    /// This is a game component that implements IUpdateable.
    /// </summary>
    public class ButtonMenu : Microsoft.Xna.Framework.DrawableGameComponent
    {
        SpriteFont spriteFont;
        SpriteBatch spriteBatch;
        Texture2D buttonImage;

        Color normalColor = Color.Black;
        Color hiliteColor = Color.White;

        KeyboardState oldState, newState;

        Vector2 position = new Vector2();

        int selectedIndex = 0;

        private StringCollection menuItems = new StringCollection();

        int width, height;

        public ButtonMenu(Game game, SpriteFont spriteFont, Texture2D buttonImage)
            : base(game)
        {
            this.spriteFont = spriteFont;
            this.buttonImage = buttonImage;

            spriteBatch =
                (SpriteBatch)Game.Services.GetService(typeof(SpriteBatch));
        }

        public int Width
        {
            get { return width; }
        }

        public int Height
        {
            get { return height; }
        }
    }
}
```

```

}

public int SelectedIndex
{
    get { return selectedIndex; }
    set
    {
        selectedIndex = (int)MathHelper.Clamp(
            value,
            0,
            menuItems.Count - 1);
    }
}

public Color NormalColor
{
    get { return normalColor; }
    set { normalColor = value; }
}

public Color HiliteColor
{
    get { return hiliteColor; }
    set { hiliteColor = value; }
}

public Vector2 Position
{
    get { return position; }
    set { position = value; }
}

public void SetMenuItems(string[] items)
{
    menuItems.Clear();
    menuItems.AddRange(items);
    CalculateBounds();
}

private void CalculateBounds()
{
    width = buttonImage.Width;
    height = 0;
    foreach (string item in menuItems)
    {
        Vector2 size = spriteFont.MeasureString(item);
        height += 5;
        height += buttonImage.Height;
    }
}

public override void Initialize()
{
    base.Initialize();
}

public override void Update(GameTime gameTime)
{

```

```

newState = Keyboard.GetState();

if (CheckKey(Keys.Down))
{
    selectedIndex++;
    if (selectedIndex == menuItems.Count)
        selectedIndex = 0;
}

if (CheckKey(Keys.Up))
{
    selectedIndex--;
    if (selectedIndex == -1)
    {
        selectedIndex = menuItems.Count - 1;
    }
}

oldState = newState;
base.Update(gameTime);
}

public bool CheckKey(Keys theKey)
{
    return oldState.IsKeyDown(theKey) && newState.IsKeyUp(theKey);
}

public override void Draw(GameTime gameTime)
{
    Vector2 textPosition = Position;
    Rectangle buttonRectangle = new Rectangle(
        (int)Position.X,
        (int)Position.Y,
        buttonImage.Width,
        buttonImage.Height);

    Color myColor;

    for (int i = 0; i < menuItems.Count; i++)
    {
        if (i == SelectedIndex)
            myColor = HiliteColor;
        else
            myColor = NormalColor;

        spriteBatch.Draw(buttonImage,
            buttonRectangle,
            Color.White);

        textPosition = new Vector2(
            buttonRectangle.X + (buttonImage.Width / 2),
            buttonRectangle.Y + (buttonImage.Height / 2));

        Vector2 textSize = spriteFont.MeasureString(menuItems[i]);
        textPosition.X -= textSize.X / 2;
        textPosition.Y -= spriteFont.LineSpacing / 2;

        spriteBatch.DrawString(spriteFont,
            menuItems[i],

```

```

        textPosition,
        myColor);
    buttonRectangle.Y += buttonImage.Height;
    buttonRectangle.Y += 5;
}

base.Draw(gameTime);
}
}
}

```

The code for this component is almost the same as for the other menu component that I made. One of the first things you will have to do is change the component from deriving from **GameComponent** to **DrawableGameComponent** as this is a visual component. I used a **StringCollection** for the menu items so I had to add a using statement for **System.Collections.Specialized** name space.

There are several variables for this component. There is a **SpriteBatch** object for drawing, a **SpriteFont** object for drawing text, a **Texture2D** for the image of the button, two colors, one for the normal text and one when the button is selected, variables to hold the states of the keyboard, a **Vector2** for the position of the menu, an integer for the selected item and the height and width of the menu and a **StringCollection** for the menu items.

The constructor is almost the same as that for the **MenuComponent**. The only difference is that this one takes a **Texture2D** for the image of the button. The constructor just sets the **SpriteFont** and **Texture2D** then retrieves the **SpriteBatch** object using the game services.

There are a number of public properties for this class. There are get properties to get the height and the width of the menu. I made them get only because you don't want others to change the width and height of the menu. That is done when they set the collection of strings for the menu. I made the property for the selected index both get and set. I did use the **MathHelper.Clamp** method to make sure that the index could not be set outside the range of menu items. There are get and set properties for the color of the menu text and the high-lighted menu text. I also made the property for getting and setting the position of the menu get and set. I probably should have done some validation in the set part as you don't want to create a menu off the screen but for now it is good enough.

Like in the other menu component there is a method to set the menu items, **SetMenuItems**. It takes an array of strings as a parameter. It clears the old menu items, sets the new menu items and then calls the method to calculate the width and the height of the menu, **CalculateBounds**.

The **CalculateBounds** method sets the width of the menu to the width of the image of the button. To calculate the height, I just loop through the number of items, take the height of the image and add a spacing value. I used 5 pixels for spacing the buttons on the menu.

The next change is in the **Update** method. This method is exactly the same as the **Update** method of the other menu component. It gets the current state of the keyboard. It checks to see if the down key is pressed and released by calling the **CheckKey** method. If it is, it increments the selected menu item. If the item is past the number of menu items it is set back to the first item. It does a similar thing if the up key has been pressed and released. It decrements the selected index and if it is -1 it sets it to the last item. It then stores the current state of the keyboard in the last state of the keyboard. Finally it calls the **Update** method of its parent class.

There is a short method called **CheckKey**. It checks to see if the key passed to it has been pressed and released within the last update call. If it has it returns true, false otherwise.

That leaves the **Draw** method. The **Draw** method is also similar to the **Draw** method of the other menu component. It loops through each of the menu items and draws them. I think it may require a little explanation though. The first thing it does is create a **Vector2** to hold where the text for the button will be drawn and a **Rectangle** for where the button will be drawn on the screen. It also creates a variable to hold the color that the text is to be drawn in. The **Draw** method then loops through each of the menu items.

Inside this loop it determines what color to draw the text in. Since the order in which you draw things in 2D is important the image for the button has to be drawn first. It is drawn using the destination rectangle calculated for the button. The next bit I did a little bit of magic. I find the center of the button on the screen by taking its X and Y coordinates and adding half its width to the X coordinate and the half the height to the Y coordinate. It then measures the string using the **MeasureString** of the font. It then subtracts half the width of the text to center it horizontally on the button. To center it properly vertically you don't just subtract half the height. You need to subtract half the line spacing of the font. It then draws the text. After drawing the text it calculates the position of the next button by taking the height of the button and the spacing value, 5 pixels.

There were a few changes to the **StartScreen**. I will give you the new code then go over the changes that were made to it.

```

using System.Text;
using New2DRPG.CoreComponents;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;

namespace New2DRPG
{
    class StartScreen : GameScreen
    {
        ButtonMenu buttonMenu;

        public StartScreen(Game game, SpriteFont spriteFont, Texture2D background,
Texture2D buttonImage)
            : base(game)
        {
            Components.Add(new BackgroundComponent(game, background));

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

            buttonMenu = new ButtonMenu(
                game,
                spriteFont,
                buttonImage);

            buttonMenu.SetMenuItems(items);
            Components.Add(buttonMenu);
        }

        public int SelectedIndex
        {
            get { return buttonMenu.SelectedIndex; }
        }

        public override void Show()
        {
            buttonMenu.Position = new Vector2((Game.Window.ClientBounds.Width -
                buttonMenu.Width) / 2, 450);

            base.Show();
        }

        public override void Hide()
        {
            base.Hide();
        }
    }
}

```

The first change was to remove the old menu and add a variable for the new menu. I had to change the constructor to pass the image of the button to the class. As before I add the background component to the list of components. I created the menu items for the menu. I then created the menu, set the menu items and added it to the list of components.

The only other change was in the **Show** method. All I did was find a nice location for the Y coordinate for the menu.

I made a few changes to the **HelpScreen** class as well. I added in a one line menu to return back to the main menu. This is the code for the new **HelpScreen**.

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

namespace New2DRPG
{
    class HelpScreen : GameScreen
    {
        ButtonMenu buttonMenu;

        public HelpScreen(Game game, SpriteFont spriteFont, Texture2D background,
Texture2D buttonImage)
            : base(game)
        {
            Components.Add(new BackgroundComponent(game, background));

            string[] items = { "RETURN TO MENU" };

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

            base.Hide();
        }

        public int SelectedIndex
        {
            get { return buttonMenu.SelectedIndex; }
        }

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

            base.Show();
        }

        public override void Hide()
        {
            base.Hide();
        }
    }
}
```

Of course the first difference is I added a menu component to the class. So, in the constructor I had to pass in the sprite font and the image for the button. In the constructor I added the background component to the list of components and created the one line menu. For such a simple menu I didn't need to add in selected index property but I did in case I wanted to extend it later. I also set the position

of the menu in the **Show** method, like I did in the start screen.

There were a few changes in the **Game1** class. Instead of giving the entire **Game1** class, I will just give you the changes to the class and then explain them. Most of the changes were in 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();

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

    activeScreen = startScreen;
}
```

will post the entire class but I'm going to tell you the differences before hand. Most of the changes were in the **LoadContent** method. The first change was where I loaded in the background for the **StartScreen**. I changed it to **titlescreen**, the new background image. I then changed where I actually created the **StartScreen**. Instead of creating a variable to hold the button image, I just used the **Content.Load** method inside the call to the constructor. As well, I loaded in the new image for the

HelpScreen. Then I changed the way I created the **HelpScreen**, passing in the font and the button image.

The only other change was in the **HandleHelpScreenInput** method. This is the code for the new **HandleHelpScreenInput**. The only change to this method was that I added in the switch statement on the selected index of the **HelpScreen**.

```
private void HandleHelpScreenInput ()
{
    if (CheckKey(Keys.Enter) || CheckKey(Keys.Space))
    {
        switch (helpScreen.SelectedIndex)
        {
            case 0:
                activeScreen.Hide();
                activeScreen = startScreen;
                activeScreen.Show();
                break;
        }
    }
}
```

Well, that is it for another tutorial. The next one probably would be about XNA exactly. It will focus on one of the most important elements of a role playing game, the player character. When you are creating a role playing game, a lot of thought has to go into the character system. This will be a class based system with four classes, Warrior/Fighter, Thief/Rogue, Wizard/Mage and Priest(ess)/Cleric. So, check back again soon and I hope to have another tutorial available on my web site.