Creating a Role Playing Game with XNA Game Studio 3.0

Core Components

In this tutorial I am going to create the basic project that all other tutorials will add to. To get started you will want to create a new project, so go a head and load up Visual C# and create a new XNA 3.0 Windows game project. Name it New2DRPG. If you are interested in downloading the project, you can find the latest version of the project on my web site at this link: New 2D RPG

Once that is done you will want to add a new folder to the project. Right click the project and select **New Folder**. Call it **CoreComponents.** To this folder you will want to add a **Game Component**, call it **GameScreen**. I will be using **Game Components** a lot in the project. I like them because you can use them to separate the updating and drawing into the separate components, although you have to do a little more work initially.

This is the code for the **GameScreen** component. In this series I will always present the code and then explain it. As well, I will organize the code like this: fields, constructors, properties and methods.

```
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;


namespace New2DRPG.CoreComponents
{
    public class GameScreen : Microsoft.Xna.Framework.DrawableGameComponent
    {
        private List<GameComponent> childComponents;

        public GameScreen(Game game)
            : base(game)
        {
            childComponents = new List<GameComponent>();
            Visible = false;
            Enabled = false;
        }

        public List<GameComponent> Components
        {
            get { return childComponents; }
        }

        public override void Initialize()
        {
```

```
            base.Initialize();
        }

        public override void Update(GameTime gameTime)
        {
            foreach (GameComponent child in childComponents)
            {
                if (child.Enabled)
                {
                    child.Update(gameTime);
                }
            }

            base.Update(gameTime);
        }

        public override void  Draw(GameTime gameTime)
        {
            foreach (GameComponent child in childComponents)
            {
                if ((child is DrawableGameComponent) &&
                    ((DrawableGameComponent) child).Visible)
                {
                    ((DrawableGameComponent) child).Draw(gameTime);
                }
            }
            base.Draw(gameTime);
        }
        public virtual void Show()
        {
            Visible = true;
            Enabled = true;
        }

        public virtual void Hide()
        {
            Visible = false;
            Enabled = false;
        }
    }
}
```

The first thing you will notice is that I change the inheritance from **GameComponent** to **DrawableGameComponent**. I did this because this component will be responsible for drawinw itself. The I create a **List** of child components of the screen, such as menus, and property to return the list. The constructor takes one parameter, the **Game** object. It creates the list of child components and sets the **Visible** and **Enabled** properties to false so the component will not draw or update itself until you want it to. You could add the child components to the components of the **GameComponent** but in this case I will update and draw them individually.

The next change is in the **Update** method. I used a loop to go through each child component and if it is enabled I call it's **Update** method.

Since this is a **DrawableGameComponent** you have to add the **Draw** method to the component. Again, I loop through each child component. This time I check to see if the component is a

DrawableGameComponent and if it is visible. If these conditions are true I call the **Draw** method of the component. If you try and draw a component that is not drawable you will get an exception.

Next I wrote two methods, one to **Show** the component by setting it to visible and enabled. The second to **Hide** the component by setting it not to be visible and enabled. Later you will see the advantages of using game components.

To the **CoreComponents** folder you will want to add another **DrawableGameComponent** and call it **MenuComponent**. This is the code for the complete **MenuComponent**.

```
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
{
    public class MenuComponent : Microsoft.Xna.Framework.DrawableGameComponent
    {
        SpriteBatch spriteBatch = null;
        SpriteFont spriteFont;

        Color normalColor = Color.Yellow;
        Color hiliteColor = Color.Red;

        KeyboardState oldState;

        Vector2 position = new Vector2();

        int selectedIndex = 0;
        private StringCollection menuItems = new StringCollection();

        int width, height;

        public MenuComponent(Game game, SpriteFont spriteFont)
            : base(game)
        {
            this.spriteFont = spriteFont;
            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 = 0;
    height = 0;
    foreach (string item in menuItems)
    {
        Vector2 size = spriteFont.MeasureString(item);
        if (size.X > width)
            width = (int)size.X;
        height += spriteFont.LineSpacing;
    }
}

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

```
}

public override void Update(GameTime gameTime)
{
    KeyboardState 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)
{
    KeyboardState newState = Keyboard.GetState();

    return oldState.IsKeyDown(theKey) && newState.IsKeyUp(theKey);
}

public override void Draw(GameTime gameTime)
{
    Vector2 menuPosition = Position;
    Color myColor;

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

        spriteBatch.DrawString(
            spriteFont,
            menuItems[i],
            menuPosition + Vector2.One,
            Color.Black);

        spriteBatch.DrawString(spriteFont,
            menuItems[i],
            menuPosition,
            myColor);

        menuPosition.Y += spriteFont.LineSpacing;
    }
```

```
            base.Draw(gameTime);
        }
    }
}
```

This component is more complicated than the **GameScreen** component. I will walk you through why I did what I did. There are several fields in this component. There is one for a **SpriteBatch** object, one for a **SpriteFont** object, two **Colors** one for the regular text color and one for the selected text color. There is a field to hold the last state of the keyboard and the current state of the keyboard. The position of the menu on the screen. What the currently selected index is and a collection of strings for the menu. Finally, the height and the width of the menu.

The constructor takes two parameters. The first is the **Game** object and the second is the **SpriteFont** that will be used to draw the menu. Inside the constructor I used **Game.Services.GetService** to get the **SpriteBatch** to be used to draw the menu instead of creating a new **SpriteBatch** object or passing one to the class. The reason will become clearer when I get to the **Game1** file. To use the **Game.Services.GetService** method you have to add the service you want to use in the **Game** class.

There are a number of properties. There is a get property for the width and height of the menu. One to get and set the **SelectedIndex** of the menu. In this property I used **MathHelper.Clamp** method to make sure the value sent is valid. There are also get and set properties for the color of the normal menu text and the selected menu text. There is also a property to get and set the position of the menu.

The first method is the **SetMenuItems** method. This method takes an array of **strings**. It clears the **StringCollection** and adds the strings to it then calls the **CalculateBounds** method that will calculate the height and width of the menu items.

The **CalculateBounds** method sets the width and height to zero, loops through the menu items. Gets the size of the string using the **MeasureString** method of the **SpriteFont** class. Then, if the width of the current items is larger than any of the previous methods it sets the width to that width. Then the height of the menu is increased using the **LineSpacing** property.

In the **Update** method, I call a method that I wrote to check if the down key has been pressed. If the down key was pressed, I increment the selected item. If it is equal to the number of items in the menu it is set to zero. Then, I check to see if the up key was pressed. If it was I decrement the counter, if it is -1 set it to **Count – 1**. Then I set the old keyboard state to the current keyboard state.

The **CheckKey** method takes one parameter, the key to be checked. It checks to see if in the last state the key was down and if in the current state the key is up. This tells if the key has been pressed and released. I return the result of that check.

That just leaves the **Draw** method. You will notice that I did not call **Begin** or **End** in the draw method. This is because of the way I have set up the **Draw** method in the **Game1** class. First I create a **Vector2** to hold the position to draw the menu item at and a **Color** for the color to draw the item. I use a **for** loop to go through the menu items. Then, I check to see what color to draw the item in. Next, I draw a shadow of the item and then the item. The reason I draw the shadow first is because when you are drawing in 2D the order in which you draw things is important. If you draw the shadow after the item the shadow will be on top of the item. Drawing the shadow second does give a rather interesting effect. Try it and if you like that better, use it.

There is one more component that I want to make. That is a component to hold a background image. I don't think that it is that hard of a component. Add a new **GameComponent** to the **CoreComponents** folder and call it **BackgroundComponent**. Here is the code for the **BackgroundComponent**.

```
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;


namespace New2DRPG.CoreComponents
{
    public class BackgroundComponent :
            Microsoft.Xna.Framework.DrawableGameComponent
    {
        Texture2D background;
        SpriteBatch spriteBatch = null;
        Rectangle bgRect;

        public BackgroundComponent(Game game, Texture2D texture)
            : base(game)
        {
            this.background = texture;
            spriteBatch =
                  (SpriteBatch)Game.Services.GetService(typeof(SpriteBatch));

            bgRect = new Rectangle(0,
                        0,
                        Game.Window.ClientBounds.Width,
                        Game.Window.ClientBounds.Height);

        }

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

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

        public override void Draw(GameTime gameTime)
        {
            spriteBatch.Draw(background, bgRect, Color.White);
            base.Draw(gameTime);
        }
```

```
    }
}
```

There are three fields in this class, one to hold the image to be drawn, one for the **SpriteBatch** object and one to hold the rectangle where the image will be drawn.

The constructor for this component takes two parameters, the **Game** object and a **Texture2D** of the image. Like in the other components I used the **GetServices** method to get the **SpriteBatch** object. Then I set the rectangle to the size of the rectangle using the **Game.Window.ClientBounds.Width** and **Game.Window.ClientBounds.Height** properties.

The next method that is different is the **Draw** method. All the **Draw** method does is draw the background image. Like I said, it is a very simple component.

Now I'm going to make two classes. Eventually there will be more but this tutorial is already starting to get a little long. The first class is a class for the starting screen that will have a **BackgroundComponent** and a **MenuComponent**. To the project add a new class called **StartScreen**. This is the code for the **StartScreen** class.

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

namespace New2DRPG
{
    class StartScreen : GameScreen
    {
        MenuComponent menu;

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

            string[] items = { "Help", "Quit" };
            menu = new MenuComponent(game, spriteFont);
            menu.SetMenuItems(items);
            Components.Add(menu);
        }

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

        public override void Show()
        {
            menu.Position = new Vector2(
                (Game.Window.ClientBounds.Width - menu.Width) / 2, 330);
            base.Show();
        }
```

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

You will notice that I've added three **using** statements to the class. The **New2DRPG.CoreComponents** was because when I created the **CoreComponents** folder they were in a new namespace. The other two are for the XNA classes that I needed. This class inherits from **GameScreen**.

There is only one field in this class, a **MenuComponent**.

The constructor for this class takes three parameters: a **Game** object, a **SpriteFont** and a **Texture2D**. In the constructor I add a **BackgroundComponent** to the list of components for this class. Then I created a **MenuComponent** and add that to the list of components for the class.

There is only one property in this class. It returns the selected index of the **MenuComponent**.

There are two methods in this class that override the virtual methods of the **GameScreen** class. One to show the component and one to hide the component. In the **Show** method I set the position of the menu on the screen and call the **Show** method of the parent class. In the **Hide** method I just call the **Hide** method of the parent class.

There is one more class that I want to add to the project. That is a class to show a help screen. For now, it will just display a image on the screen. You can return to the menu screen by pressing the space bar, escape key or enter key. Add a new class to the project and call it **HelpScreen**. This is the code for that class.

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

namespace New2DRPG
{
    class HelpScreen : GameScreen
    {
        public HelpScreen(Game game, Texture2D background)
            : base(game)
        {
            Components.Add(new BackgroundComponent(game, background));
            base.Hide();
        }

    }
}
```

This is really just a holder class that hides and shows the help screen. More functionality can be built into it later. There are no fields in this class.

The constructor takes two paramaters. A **Game** object and a **Texture2D** for the image to be displayed. It creates and adds a **BackgroundComponent** and hides itself.

That just leaves writing the **Game** class. Before I give you the code there is something that you need to do. You need to add two images and a **SpriteFont** to the **Content** folder. In the project I have included two images that I got from**:** http://feebleminds-gifs.com, they have an excellent collection of art for non-commercial use. I chose the **fire-dragon** and the **gryphon**. To add images to the **Content** folder just right click the **Content** folder and click **Existing Item**. Choose two images. If you don't use these images you will have to make a couple adjustments to your code. Now you need to add a **SpriteFont** to the **Content** folder. You need to rigth click the **Content** folder and click **New Item**. Select **SpriteFont** and call it **normal.spritefont**.

Now you can code the **Game** class. Again, I will give you the code and then explain 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 New2DRPG.CoreComponents;

namespace New2DRPG
{
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;

        StartScreen startScreen;
        HelpScreen helpScreen;
        GameScreen activeScreen;

        SpriteFont normalFont;

        Texture2D background;

        KeyboardState newState;
        KeyboardState oldState;

        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
        }

        protected override void Initialize()
        {
            base.Initialize();
        }
```

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

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

    background = Content.Load<Texture2D>("gryphon");
    startScreen = new StartScreen(this, normalFont, background);
    Components.Add(startScreen);

    background = Content.Load<Texture2D>("fire-dragon");
    helpScreen = new HelpScreen(this, background);
    Components.Add(helpScreen);

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

    activeScreen = startScreen;
}

protected override void UnloadContent()
{
}

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();
    }

    oldState = newState;

    base.Update(gameTime);
}

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

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

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

        protected override void Draw(GameTime gameTime)
        {
            GraphicsDevice.Clear(Color.CornflowerBlue);

            spriteBatch.Begin(SpriteBlendMode.AlphaBlend);
            base.Draw(gameTime);
            spriteBatch.End();
        }
    }
}
```

Again, I had to add a using statement to refrence the **CoreComponents** namespace. Next I added some variables to the class. There is one for the **HelpScreen**, **StartScreen** and one to hold the **active screen**. I also added variables for the **SpriteFont**, the background image and variables to hold the current and last states of the keyboard.

The **LoadContent** method is where you load in your content and create the **SpriteBatch** object. After creating the **SpriteBatch** object I registered it as a **GameService**. I used the **Content.Load** method to load in the font. Then I loaded in the image for the start screen and created the **StartScreen** object and added it to the list of **Components**. Then I loaded in the image for the **HelpScreen** and created it and added it to the list of **Components**.

Then I called the **Show** method of the **startScreen** and the **Hide** method of the **helpScreen** and set the active screen to the start screen.

I tried to keep the **Update** method clean by separating the logic for the different screeens. At the start of the **Update** method I get the current state of the keyboard. Then I have a set of if statements. If the active screen is the start screen I call the **HandleStartScreenInput** method. If the active screen is the help screen I call the **HandleHelpScreenInput** method.

The **HandleHelpScreenInput** method checks to see if the space, enter or escape key has been pressed calling a method the same as the one in the **MenuComponent**. If any of the keys have been pressed it hides the help screen, sets the active screen to the start screen and shows the start screen.

The **HandleStartScreenInput** checks to see if the space or enter key has been pressed. Then there is a **switch** statement that uses the **SelectedIndex** of the **MenuComponent**. If the index is 0, the help screen index, the active screen is hid, the active screen is set to the help screen and the active screen is shown. If the index is 1, the quit index, the game exits.

The **CheckKey** method is the same as in the **MenuComponent**.

The **Draw** method is a little different. After clearing the screen, I call the **Begin** method of the **SpriteBatch** object with **AlphaBlend** mode. Then **base.Draw** draws any game components. Finally I call the **End** method of the **SpriteBatch** object.

That is all for this tutorial. I will try and have another one ready very soon.