XNA Screen Manager

XNA Screen Manager
Part One
Creating a Menu

This is part one of a two part tutorial about creating screen management system with XNA. This tutorial is generic enough to work in both XNA 3.0 and XNA 3.1 so you can use either one of them for the tutorial. This tutorial uses some common object-oriented programming principles. If you do not have an understanding of object-oriented programming you may have a hard time understanding parts of this tutorial.

Think of a typical game out today. It is made up of many different screens. Some games have an introduction that plays to introduce you to the game. There is usually a menu that you can use to select options. There is the game itself. A game might also have a high score screen to display the players with the highest score.

As games get larger it gets harder and harder to manage the different states of a game, especially if you are creating a complex game like a role playing game with lots of different screens that the player will interact with. Using a screen management, or scene management, system you can reduce the complexity of using multiple screens in your game. This tutorial will use the concept of XNA Game Components to implement a screen manager for your games. An XNA **Game Component** can be thought of as almost like a game inside your game. It has all of the methods that an XNA Game has, it only has visual elements if you create **Drawable Game Component** though.

If you look at the class definition of an XNA game you will see that it inherits from the **Microsoft.Xna.Framework.Game** class. Inheritance is an important concept in object-oriented programming. Using inheritance, a class that is derived from a base class can use the public, and in C# protected, methods, properties, and fields of the base class.

If you look at the methods of an XNA game they all contain they override keyword. This means that they are virtual methods that you override in your game. Virtual methods allow you to override, or change is a better word, the way the base class works. In an XNA game you are overriding the methods of the **Microsoft.Xna.Framework.Game** class which controls the way the game executes. It is in this overriding of the default behavior of the **Game** class that makes your game work.

What I am going to do in this tutorial is set up a simple menu that will allow you to scroll between various options in the menu. I will implement the switching between screens in the second part of this tutorial.

Go a head and create a new XNA game and you can call it whatever you would like. You are going to need a **SpriteFont** for the menu. **SpriteFonts** are XNA's way of drawing text in your game. Add a **SpriteFont** to your game by right clicking the Content folder, select Add and then select New Item. In the left pain make sure that XNA Game Studio is selected. Now choose **SpriteFont** and call it **menufont.spritefont**.

XNA Screen Manager

Now it is time to make the menu. I will use a **Drawable Game Component** for this. Right click your game project in the solution explorer select Add and then New Item. Make sure that XNA Game Studio is selected on the left side. Select the **GameComponent** entry and call it **MenuComponent**. When you create the component a lot of code will be generated for you. There will be the default constructor for the component, an override of the Initialize method and an override of the Update method. I will be making a few changes to this. I will let you read the code and then explain the way that it works.

```
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 ScreenManager
{
    public class MenuComponent : Microsoft.Xna.Framework.DrawableGameComponent
    {
        string[] menuItems;
        int selectedIndex;

        Color normal = Color.White;
        Color hilite = Color.Yellow;

        KeyboardState keyboardState;
        KeyboardState oldKeyboardState;

        SpriteBatch spriteBatch;
        SpriteFont spriteFont;

        Vector2 position;
        float width = 0f;
        float height = 0f;

        public int SelectedIndex
        {
            get { return selectedIndex; }
            set
            {
                selectedIndex = value;
                if (selectedIndex < 0)
                    selectedIndex = 0;
                if (selectedIndex >= menuItems.Length)
                    selectedIndex = menuItems.Length - 1;
            }
```

```
}

public MenuComponent(Game game,
    SpriteBatch spriteBatch,
    SpriteFont spriteFont,
    string[] menuItems)
    : base(game)
{
    this.spriteBatch = spriteBatch;
    this.spriteFont = spriteFont;
    this.menuItems = menuItems;
    MeasureMenu();
}

private void MeasureMenu()
{
    height = 0;
    width = 0;
    foreach (string item in menuItems)
    {
        Vector2 size = spriteFont.MeasureString(item);
        if (size.X > width)
            width = size.X;
        height += spriteFont.LineSpacing + 5;
    }

    position = new Vector2(
        (Game.Window.ClientBounds.Width - width) / 2,
        (Game.Window.ClientBounds.Height - height) / 2);
}

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

private bool CheckKey(Keys theKey)
{
    return keyboardState.IsKeyUp(theKey) &&
        oldKeyboardState.IsKeyDown(theKey);
}

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

    if (CheckKey(Keys.Down))
    {
        selectedIndex++;
        if (selectedIndex == menuItems.Length)
            selectedIndex = 0;
    }
    if (CheckKey(Keys.Up))
    {
```

```
                selectedIndex--;
                if (selectedIndex < 0)
                    selectedIndex = menuItems.Length - 1;
            }
            base.Update(gameTime);

            oldKeyboardState = keyboardState;
        }

        public override void Draw(GameTime gameTime)
        {
            base.Draw(gameTime);
            Vector2 location = position;
            Color tint;

            for (int i = 0; i < menuItems.Length; i++)
            {
                if (i == selectedIndex)
                    tint = hilite;
                else
                    tint = normal;
                spriteBatch.DrawString(
                    spriteFont,
                    menuItems[i],
                    location,
                    tint);
                location.Y += spriteFont.LineSpacing + 5;
            }
        }
    }
}
```

When you create a **Game Component** it is inherited from **GameComponent**. I changed this to inherit from **DrawableGameComponent** because I will be drawing the menu items in this component. I added eleven fields to the class. The first two, **menuItems** and **selectedIndex**, will hold the items for the menu and which item is currently selected. **normal** and **hilite** will be used in drawing the menu items. **normal** will be the color to draw the normal menu items and **hilite** will be the color to draw the currently selected menu item.

The **KeyboardState** fields **keyboardState** and **oldKeyboardState** will hold the current state of the keyboard and the state of the keyboard in the previous frame of the game. What I mean by frame is that an XNA game tries to call the **Update** and **Draw** methods as close to 60 times per second by default. Each call to the **Update** and **Draw** methods are considered a frame in the game. These fields will be used to detect single key presses of the **Up** and **Down** arrow keys.

The **spriteBatch** and **spriteFont** fields will be used for drawing the text of the menu. The **SpriteBatch** class has a method called **DrawString** that draws text on the window. Like I mentioned earlier you need a **SpriteFont** object to draw text with XNA. The last three fields, **position**, **width**, and **height** will be used to position the menu on the screen.

XNA Screen Manager

I truly do not like making fields inside a class public unless it is absolutely necessary. I will eventually need a way to get and set the selected menu item. I used a property for this called **SelectedIndex**. The get part of the property just returns the value of the **selectedIndex** field. The set property preforms a little validation on the value to set the **selectedIndex** field. It first sets **selectedIndex** to value, the value passed to the property. It then checks to see if **selectedIndex** is less than zero. If it is it then sets **selectedIndex** to zero. It then checks to make sure that **selectedIndex** is not greater than or equal to the length of the array. If it is it sets **selectedIndex** to be the length of the array minus 1.

The reason for these checks is that the menu items are stored in an array of strings. If you try to access an array element using a negative index you will get an array out of bounds exception. Similarly if you try and access an element greater than or equal to length of the array you will get the same exception.

The constructor for the class has four parameters: **game**, **spriteBatch**, **spriteFont**, and **menuItems**. The reason for the first one is that the class that **GameComponents** and **DrawableGameComponents** inherit from require a **Game** object. That is why there is a **: base(game)** after the right bracket. The constructor sets the **spriteBatch**, **spriteFont**, and **menuItems** fields. It then calls the **MeasureMenu** method that I wrote to measure the height and width of the menu. It will also center the menu on the screen.

The **MeasureMenu** method first sets the height and width fields to be zero. The reason is that they will be used to center the menu in the window. Each time this method is called you should reset the values so they will be set to the appropriate values.

In a foreach loop I then loop through the items in the menu. I get the size of the string using the **MeasureString** method of the **SpriteFont** class. This method returns a **Vector2** with the **X** property being the width of the string and the **Y** property being the height of the string. If the **X** property of the size is greater than the width I set width to be the **X** property of the size. This is because to center the menu properly I need to know the width of the menu. To calculate the height of the menu I use the **LineSpacing** property of the **SpriteFont** class. If you think of a word processor. When you are typing and you go to the next line of text there is a little spacing between the lines of text. The **LineSpacing** property measures the distance from the start of the text to the start of the text on the next line. I also add an extra 5 pixels to space it out a little more.

Centering objects is something that you will come across frequently in programming, especially game programming. To center an object horizontally you take the width of what you want to center the object in, subtract the width of the object and divide that by two. So for the **X** property of the **Vector2** that will hold the position of the menu I use the width of the window and the width of the largest menu item. Similarly for the height you take the height of what you are centering in, subtract the height of the object, and divide that by two. So for the **Y** property of the **Vector2** of the position I take the height of the window, subtract the height of the menu, and divide that by two.

After the **Initialize** method, that I didn't do anything in because there was no need, is a method called **CheckKey** that accepts a **Key** parameter and returns true if the **Key** parameter has been pressed in the last frame and released in the current frame. Many people do this the opposite way, checking to see if it

is down in the current frame and released in the last frame. This will lead to problems when working with menus. The reason is when you move from one menu to another the keyboard states could be in the same state causing the currently selected item in the next menu to be selected. You check to see if a key is currently up using the **IsKeyUp** method of the **KeyboardState** class passing in the key you are interested. Similarly to check if a key is down you use the **IsKeyDown** method passing in the key. In the first one the method returns true if the key is currently up, false otherwise. **IsKeyDown** returns true if the key is currently down and false if it is up.

The **Update** method is where I actually handle the moving of the selected menu item up and down using the Up and Down arrow keys. I first set the current state of the keyboard, **keyboardState**, to the current state of the keyboard using **Keyboard**.**GetState**. That reads the current state of the keyboard. I then I call the **CheckKey** method passing in the Down key. If the Down key was pressed I increase the **selectedIndex** field by 1. I check to make sure that it is not the length of the menu items. If it is I go to the first item in the causing the selected item to wrap from bottom to top. I then check to see if the Up key was pressed once. If it was I decrease the **selectedIndex** field by 1. If it is less than zero I set **selectedIndex** to be the number of items minus 1, the last item, causing the item to wrap from the top of the menu to the bottom. Before exiting the **Update** method I set the **oldKeyboardState** field to be **keyboardState** field so that in the next frame of the game I will have the current state of the keyboard and the last state of the keyboard.

The Draw method is where I draw the menu items on the screen. You maybe wondering why I am drawing things after the call to **base.Draw(gameTime)**. The reason is that since I am using **Game Components** drawing after that call will cause this component to be drawn after other components. The reason is that the order of rendering objects in 2D is important. The last drawn objects will appear over top of previously drawn objects. If you draw the menu before the call to **base.Draw(gameTime)** it will be drawn over by other objects. You want the menu to appear on the top. You also maybe wondering why there is no call to the **Begin** and **End** methods of the **SpriteBatch** class. The reason is when you add a **Game Component** to the list of **Game Components** in the game, XNA will automatically call their **Draw** and **Update** methods when the **base.Update** and **base.Draw** methods are called in your game, if the components are enabled and visible. You will see further when I get to the **Game1** class.

There are two local variables in this method: **location** and **tint**. **location** will hold where to draw the next line of the menu and is set initially to the position field that I calculated earlier. The **tint** variable will be used to determine what color to draw the text. There is a for loop that loops through all of the menu items. I used a for loop instead of a foreach loop so I could check if the index, i, is equal to **selectedIndex**. If i is equal to **selectedIndex** I set **tint** to **hilite** so the item will be hi lighted other wise I set **tint** to **normal** because it is regular text.

I used the **DrawString** method of the **SpriteBatch** class to draw the actual string. The overload that I used requires four parameters: the **SpriteFont** to draw the text with, the text to draw, the location to draw it, and the color to draw it.

To have the lines of the menu appear on different lines I add the **LineSpacing** property of the

SpriteFont and 5 pixels to the **Y** property of the **location** variable. Just the same as I did in the MeasureMenu method. If you don't do it the same as you did there it won't be centered in the window.

Now I can implement a simple menu that you can scroll through the items of the menu in. At the moment it will not do anything else. Adding that in will make the tutorial longer than it already is. The first thing you will need to do is add a field to the **Game1** class to hold the **MenuComponent**. Add the following field to your class just below the **SpriteBatch** field.

```
MenuComponent menuComponent;
```

Now you need to actually create a menu and add it to the list of components of the game. You will do that in the **LoadContent** method. What I did was create a string array with three entries in it: Start Game, High Scores, and End Game. After the **spriteBatch** field was created, because I need to pass it to the component, I created the component using the constructor passing in: this, **spriteBatch**, the **SpriteFont**, and the **menuItems** variable. The word this refers to the current **Game** object. **spriteBatch** is the current **SpriteBatch** object in use. I used the **Content**.**Load** method to load in the **SpriteFont** that was added to the game. I finally add the new **MenuComponent** to the list of components of the game. This is the code for the **LoadContent** method.

```
protected override void LoadContent()
{
    string[] menuItems = { "Start Game", "High Scores", "End Game" };
    spriteBatch = new SpriteBatch(GraphicsDevice);

    menuComponent = new MenuComponent(this,
        spriteBatch,
        Content.Load<SpriteFont>("menufont"),
        menuItems);
    Components.Add(menuComponent);
}
```

I didn't have to add anything into the **Initialize**, **UnloadContent** or **Update** methods. Because the component was added to the list of components of the game its **Update** method will automatically be called when **base.Update** is called.

In the Draw method I call the Begin method of the **SpriteBatch** class after the call to the **Clear** method which clears the window. Then there is the call to **base.Draw** so that all child components will be drawn and finally call the **End** method of the **SpriteBatch** class to stop drawing. This the code for the **Draw** method.

```
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);
    spriteBatch.Begin(SpriteBlendMode.AlphaBlend);
    base.Draw(gameTime);
    spriteBatch.End();
}
```