XNA Screen Manager

XNA Screen Manager
Part Three
Adding Xbox 360 Controller Support

This is a third part of the XNA Screen Manager tutorials that I have written. In this tutorial I will add in support for the Xbox 360 controller with the menus. 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.

To get started you will want to open your project from the previous tutorials I wrote on creating an XNA screen manager. If you haven't completed those tutorial go and read them first and come back when you are done. You can find the first one at this links to the tutorials on the XNA Tutorials page of my web site.

Now that you have the project loaded you are ready to start. One of the good things you can do with XNA is create games for the Xbox 360. Also, if you have an Xbox 360 that you can plug into your computer you can use it with your Windows games. You can find both wired and wireless controllers that you can attach to your computer.

To get started I will add in support to the MenuComponent. Open the MenuComponent in the Code View window. What I will add is have the menu scroll if the player is pressing the Up or Down keys on the keyboard or using the direction pad on the controller. You are going to need two new fields in the MenuComponent class. One to hold the current state of the game pad and another to hold the last state of the game pad. The reason you need two fields is you want to detect if the pad has been pressed and released. If you just check if its current state is down the selected item will move rapidly up or down the menu depending what direction you are pressing. This is because XNA tries to call the Update method 60 times a second, which is about 1 every 16 milliseconds, much too fast for you to see what is happening. Add the following two fields to the class.

```
GamePadState gamePadState;
GamePadState oldGamePadState;
```

Now, there will be quite a few changes to the Update method. The one thing you are going to want to do is get the state of the game pad for player one. You will want to check to see if the direction pad has been pressed and released in the up or down direction. At the end of the method you will want to set the old state of the game pad to the current state of the game pad. I will explain the code after I have shown it to you.

```
public override void Update(GameTime gameTime)
{
    keyboardState = Keyboard.GetState();
    gamePadState = GamePad.GetState(PlayerIndex.One);

    if (CheckKey(Keys.Down) || CheckButton(Buttons.DPadDown))
    {
        selectedIndex++;
```

```
            if (selectedIndex == menuItems.Length)
                selectedIndex = 0;
        }
        if (CheckKey(Keys.Up) || CheckButton(Buttons.DPadUp))
        {
            selectedIndex--;
            if (selectedIndex < 0)
                selectedIndex = menuItems.Length - 1;
        }

        base.Update(gameTime);

        oldKeyboardState = keyboardState;
        oldGamePadState = gamePadState;
    }
```

Unlike keyboards, you can have more than one Xbox controller attached. To differentiate between them you pass in a PlayerIndex parameter to the GamePad.GetState method. The first controller is assigned to PlayerIndex.One. To simplify scrolling the menu you can use an or, ||, in the if statement that checks for if the player has pressed the up or down keep checking if the player has pressed up or down using the direction pad. Is an or in the if statement means that if the player presses the down key, down or the Xbox controller's direction pad, or both it will move the selected item once. The same is true for the up key and pressing up on the game pad.

Like the keyboard has an enum for the different keys on the keyboard called Keys, the game pad has an enum for the different buttons called Buttons. There is an enum for each of the buttons on the game pad. There are four buttons you can check on the direction pad: DPadDown, DPadUp, DPadLeft, and DPadRight. At the end of the Update method I set the oldGamePadState field to be the current gamePadState fields just like I did with the Keyboard state field.

Like I wrote a helper method to check for key presses and releases I wrote a helper method to check for button presses and releases for the game pad. I called the method CheckButton. It will return true if a button had been pressed and released and false otherwise. It is pretty much the same as the CheckKey method. This is the code.

```
private bool CheckButton(Buttons button)
{
    return gamePadState.IsButtonUp(button) &&
        oldGamePadState.IsButtonDown(button);
}
```

So, now you can move the selected items in the menu up or down using either the keyboard or the game pad. What I will add now is being able to select an item using the A button on the game pad. Open the Game1 class in the editor. Like you did with the MenuComponent class, you will want to add in fields for the current and last state of the game pad. Add the following two fields to your class.

```
GamePadState gamePadState;
GamePadState oldGamePadState;
```

XNA Screen Manager

I changed the Update method and I also add a copy of the CheckButton method to the Game1 class. This is the code for both the Update method and the CheckButton methods.

```
protected override void Update(GameTime gameTime)
{
    keyboardState = Keyboard.GetState();
    gamePadState = GamePad.GetState(PlayerIndex.One);

    if (CheckButton(Buttons.Back))
        this.Exit();

    if (activeScreen == startScreen)
    {
        if (CheckKey(Keys.Enter) || CheckButton(Buttons.A))
        {
            if (startScreen.SelectedIndex == 0)
            {
                activeScreen.Hide();
                activeScreen = actionScreen;
                activeScreen.Show();
            }
            if (startScreen.SelectedIndex == 1)
            {
                this.Exit();
            }
        }
    }
    base.Update(gameTime);

    oldKeyboardState = keyboardState;
    oldGamePadState = gamePadState;
}

private bool CheckButton(Buttons button)
{
    return gamePadState.IsButtonUp(button) &&
        oldGamePadState.IsButtonDown(button);
}
```

So, what is going on here? Well, just like in the Update method of the MenuComponent, I get the current state of the game pad. I also replaced the code that allows the game to exit. Since we are now using the game pad and have the nice CheckButton method, I use the CheckButton method passing in the enum for the back button. Like I did in the MenuComponent I check to see if the enter key was pressed or if the A button was pressed on the game pad in the if statement where I was checking to see if the enter key was pressed. At the end of the method I set oldGamePadState to be gamePadState.

Well, that is it for this tutorial. I'm considering writing a forth tutorial that will cover adding in a pop up screen so when the player wants to exit the game they will be asked if that is really what they want to do.