Creating a Role Playing Game with XNA Game Studio 3.0
Part 45
Xbox 360 Controller Support

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 on that page. 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: http://www.jtmbooks.com/rpgtutorials/New2DRPG44.zip You can download the graphics from this link: Graphics.zip

In this tutorial I will be adding in support for the Xbox 360 controller. If you don't have an Xbox 360 controller don't worry about it I will still be adding in support for the keyboard. This way if you want to develop a role playing game for the Xbox 360 you will not have figure out how to use the controller on your own.

To get started you will want to open the **ButtonMenu** class. If you recall that class handles moving the selected item using the keyboard. I'm going to add in support for changing the selected item with the Xbox 360 controller, game pad means the same thing as Xbox 360 controller. I will be using the direction pad to change the selected item. The first thing you are going to want to do is add two fields to the class near the **KeyboardState** fields. Add these two fields to the class.

```
GamePadState oldPadState;
GamePadState newPadState;
```

Just like with the keyboard I will be detecting presses and releases so I need the old state of the game pad and the new state of the game pad. I will be checking to see if the last state was down and the current state is up. The next thing you will want to do is change the update method to get the state of the game pad, check to see if there is anything of interest, and process anything of interest. There are a few new concepts here so I will explain them after you have read the code.

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

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


    if (CheckKey(Keys.Up) || CheckButton(Buttons.DPadUp))
    {
        selectedIndex--;
        if (selectedIndex == -1)
        {
```

```
            selectedIndex = menuItems.Count - 1;
        }
    }

    oldState = newState;
    oldPadState = newPadState;

    base.Update(gameTime);
}
```

The first thing to do is get the current state of the game pad. There is only ever one active keyboard but on the Xbox, and in Windows, there can be up to four controllers at once. That is why when I get the state of the game pad using **GamePad.GetState** I had to pass in the enum value **PlayerIndex.One**. I won't be adding in support for two controllers to the game so I will always be using **PlayerIndex.One** for the game pad. Just like I wrote the little helper method **CheckKey**, I also wrote a little helper method **CheckButton** that takes as a parameter an enum value of **Buttons.** If you look at the controller there are quite a few buttons on it. The direction pad is considered to have four buttons: DPadUp, DPadLeft, DPadDown, and DPadRight. There are also the four colored buttons: A, B, X, and Y. There is the Guide, Back and Start buttons and others. For this class we are only interested in the up and down buttons on the direction pad. By checking if the Up key on the keyboard or the Up button on the game pad are pressed instead of checking them separately keeps the selected item from moving twice if both the Up key on the Up button on the game pad are pressed at the same time. It would not be the easiest thing in the world to do but it could be done. At the end of the method I set **oldPadState** to be **newPadState**.

I mentioned that the **CheckButton** method is implemented the same was as the **CheckKey** method. The only differences is instead of using the **Keys** enum, I use the **Buttons** enum. This is the code for the **CheckButton** method.

```
private bool CheckButton(Buttons theButton)
{
    return oldPadState.IsButtonDown(theButton) &&
           newPadState.IsButtonUp(theButton);
}
```

Those are all of the changes that need to be made to the **ButtonMenu** class. The next thing to do is have a way for the player to select the menu items using the game pad. The first thing is to add in fields to hold the state of the game pad and the state of the game pad in the last frame. Add the following fields to the **Game1** class.

```
GamePadState newPadState;
GamePadState oldPadState;
```

The next thing to do is get the state of the game pad and handle the input from the game pad. In the start of the **Update** method you will want to get the state of the game pad and at the end set it to the last state. You should also remove the code from the **Update** method that will exit the game if you press the back button on the game pad .You will need a copy of the **CheckButton** method as well. This is the code for the new **Update** method and the **CheckButton** method.

```
protected override void Update(GameTime gameTime)
{
    newState = Keyboard.GetState();
```

```
newPadState = GamePad.GetState(PlayerIndex.One);

if (!dialog.Enabled)
    inDialog = false;

if (activeScreen == startScreen)
{
    HandleStartScreenInput();
}
else if (activeScreen == helpScreen)
{
    HandleHelpScreenInput();
}
else if (activeScreen == createPCScreen)
{
    HandleCreatePCScreenInput();
}
else if (activeScreen == quitPopUpScreen)
{
    HandleQuitPopUpScreenInput();
}
else if (activeScreen == genderPopUpScreen)
{
    HandleGenderPopUpScreenInput();
}
else if (activeScreen == classPopUpScreen)
{
    HandleClassPopUpScreenInput();
}
else if (activeScreen == difficultyPopUpScreen)
{
    HandleDifficultyPopUpScreenInput();
}
else if (activeScreen == nameInputScreen)
{
    HandleNameInputScreenInput();
}
else if (activeScreen == introScreen)
{
    HandleIntroScreenInput();
}
else if (activeScreen == creditScreen)
{
    HandleCreditScreenInput();
}
else if (activeScreen == actionScreen)
{
    HandleActionScreeenInput();
    HandlePlayerInput(gameTime);
}
else if (activeScreen == viewCharacterScreen)
{
    HandleViewCharacterScreenInput();
}
else if (activeScreen == quitActionScreen)
{
    HandleQuitActionScreenInput();
}
else if (activeScreen == combatScreen)
```

```
    {
        HandleCombatScreenInput();
    }

    oldState = newState;
    oldPadState = newPadState;

    base.Update(gameTime);
}

private bool CheckButton(Buttons theButton)
{
    return newPadState.IsButtonUp(theButton) &&
        oldPadState.IsButtonDown(theButton);
}
```

This next thing to do is to have all of the input handling methods use the game pad as well. The first thing I will do is have the handlers that accept the Enter or Space key now accept either the Enter or the B button on the game pad. The **HandleActionScreenInput** method will check to see if the Back button on the game pad has been pressed. To bring up the statistics for the player character I will use the Y button on the controller. There a few methods that I will not be updating at the moment, I will get to them shortly, they are: **HandleCombatScreenInput**, **HandlePlayerInput**, and **HandlePlayerMovement**.

```
private void HandleQuitActionScreenInput()
{
    if (CheckKey(Keys.Enter) || CheckButton(Buttons.B))
    {
        switch (quitActionScreen.SelectedIndex)
        {
            case 0:
                activeScreen.Hide();
                activeScreen = startScreen;
                activeScreen.Show();
                break;
            case 1:
                activeScreen.Hide();
                activeScreen = actionScreen;
                activeScreen.Show();
                break;
        }
    }
}

private void HandleViewCharacterScreenInput()
{
    if (CheckKey(Keys.Enter) || CheckButton(Buttons.B))
    {
        activeScreen.Hide();
        activeScreen = actionScreen;
        actionScreen.Show();
    }
}

private void HandleActionScreeenInput()
{
    if (CheckKey(Keys.Escape) || CheckButton(Buttons.Back))
```

```
    {
        playerSprite.IsAnimating = false;
        activeScreen.Enabled = false;
        activeScreen = quitActionScreen;
        activeScreen.Show();
    }
    else if (CheckKey(Keys.V) || CheckButton(Buttons.Y))
    {
        playerSprite.IsAnimating = false;
        activeScreen.Enabled = false;
        activeScreen = viewCharacterScreen;
        viewCharacterScreen.SetPlayerCharacter(playerCharacter);
        activeScreen.Show();
    }
}

private void HandleCreditScreenInput()
{
    if (CheckKey(Keys.Enter) || CheckButton(Buttons.B))
    {
        activeScreen.Hide();
        activeScreen = startScreen;
        activeScreen.Show();
    }
}

private void HandleIntroScreenInput()
{
    if (CheckKey(Keys.Space) || CheckButton(Buttons.B) ||
introScreen.IntroFinished)
    {
        activeScreen.Hide();
        activeScreen = startScreen;
        activeScreen.Show();
    }
}

private void HandleNameInputScreenInput()
{
    if (CheckKey(Keys.Enter) || CheckButton(Buttons.B))
    {
        createPCScreen.ChangeName(nameInputScreen.Text);
        activeScreen.Hide();
        activeScreen = createPCScreen;
        activeScreen.Show();
    }
}

private void HandleDifficultyPopUpScreenInput()
{
    if (CheckKey(Keys.Enter) || CheckButton(Buttons.B))
    {
        createPCScreen.ChangeDifficulty(difficultyPopUpScreen.SelectedIndex);
        activeScreen.Hide();
        activeScreen = createPCScreen;
        activeScreen.Show();
    }
}
```

```
private void HandleClassPopUpScreenInput()
{
    if (CheckKey(Keys.Enter) || CheckButton(Buttons.B))
    {
        createPCScreen.ChangeClass(classPopUpScreen.SelectedIndex);
        activeScreen.Hide();
        activeScreen = createPCScreen;
        activeScreen.Show();
    }
}

private void HandleGenderPopUpScreenInput()
{
    if (CheckKey(Keys.Enter) || CheckButton(Buttons.B))
    {
        switch (genderPopUpScreen.SelectedIndex)
        {
            case 0:
                createPCScreen.ChangeGender(true);
                activeScreen.Hide();
                activeScreen = createPCScreen;
                activeScreen.Show();
                break;
            case 1:
                createPCScreen.ChangeGender(false);
                activeScreen.Hide();
                activeScreen = createPCScreen;
                activeScreen.Show();
                break;
        }
    }
}

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

private void HandleHelpScreenInput()
{
    if (CheckKey(Keys.Enter) || CheckButton(Buttons.B))
    {
        switch (helpScreen.SelectedIndex)
        {
            case 0:
                activeScreen.Hide();
```

```
                activeScreen = startScreen;
                activeScreen.Show();
                break;
        }
    }
}

private void HandleStartScreenInput()
{
    if (CheckKey(Keys.Enter) || CheckButton(Buttons.B))
    {
        switch (startScreen.SelectedIndex)
        {
            case 0:
                activeScreen.Hide();
                activeScreen = createPCScreen;
                activeScreen.Show();
                break;
            case 1:
                activeScreen.Hide();
                playerCharacter = new FighterCharacter(
                    "Evander",
                    false,
                    Level.Normal,
                    this);
                playerSprite = new AnimatedSprite(this,
                    playerTextures[0, 0],
                    ClonePlayerAnimations());
                playerSprite.CurrentAnimation = AnimationKey.Down;
                player = new PlayerComponent(this, playerSprite, playerCharacter);
                activeScreen = actionScreen;
                player.Show();
                actionScreen.Show();
                break;
            case 2:
                activeScreen.Hide();
                activeScreen = helpScreen;
                activeScreen.Show();
                break;
            case 3:
                activeScreen.Hide();
                activeScreen = creditScreen;
                activeScreen.Show();
                break;
            case 4:
                activeScreen.Enabled = false;
                activeScreen = quitPopUpScreen;
                activeScreen.Show();
                break;
        }
    }
}

private void HandleCreatePCScreenInput()
{
    if (CheckKey(Keys.Enter) || CheckButton(Buttons.B))
    {
        switch (createPCScreen.SelectedIndex)
        {
```

```
                case 0:
                    activeScreen.Enabled = false;
                    activeScreen = nameInputScreen;
                    activeScreen.Show();
                    break;
                case 1:
                    activeScreen.Enabled = false;
                    activeScreen = genderPopUpScreen;
                    activeScreen.Show();
                    break;
                case 2:
                    activeScreen.Enabled = false;
                    activeScreen = classPopUpScreen;
                    activeScreen.Show();
                    break;
                case 3:
                    activeScreen.Enabled = false;
                    activeScreen = difficultyPopUpScreen;
                    activeScreen.Show();
                    break;
                case 4:
                    activeScreen.Hide();
                    activeScreen = startScreen;
                    activeScreen.Show();
                    break;
                case 5:
                    activeScreen.Hide();
                    CreatePlayerCharacter();
                    activeScreen = actionScreen;
                    activeScreen.Show();
                    break;
            }
        }
    }
```

I know that was a lot of code to throw at you, and a lot of it stayed the same as before, but I could not think of a better way to do it. Before I go much further I want to add support for the controller to the **DialogCompontent** class. You will again have to add fields to hold the current state of the game pad and the last state of the game pad. Add the following fields near the **KeyboardState** fields.

```
GamePadState oldPadState;
GamePadState newPadState;
```

Now you will need to make a few changes to the **Update** method and add in a copy of the **CheckButton** method. In the **Update** method you will do like you did in the **ButtonMenu**. You will also want to get the current state of the controller and at the end of the method set it to the old state of the controller. You will check to see if the direction pad on the game pad has been moved up or down. To be consistent with the selection of items, I changed selecting an item to either the Enter key pressed or the B button on the game pad. This is the new code.

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

```
    if (dialog != null && npc != null)
    {
        if (CheckKey(Keys.Up) || CheckButton(Buttons.DPadUp))
        {
            currentHandler--;
            if (currentHandler < 0)
                currentHandler = dialog.HandlerCount - 1;
        }
        if (CheckKey(Keys.Down) || CheckButton(Buttons.DPadDown))
        {
            currentHandler++;
            if (currentHandler == dialog.HandlerCount)
                currentHandler = 0;
        }
        if (CheckKey(Keys.Enter) || CheckButton(Buttons.B))
        {
            dialog.InvokeHandler(npc, currentHandler);
            currentHandler = 0;
        }
    }
    base.Update(gameTime);

    oldPadState = newPadState;
   oldState = newState;
}

private bool CheckButton(Buttons theButton)
{
    return oldPadState.IsButtonDown(theButton) &&
            newPadState.IsButtonUp(theButton);
}
```

What I will do is modify the **HandleCombatScreenInput** method. In this method I will just have it so that if the player presses the Back button on the game pad the game will go back to the action screen, like with the escape key. This is the new method.

```
private void HandleCombatScreenInput()
{
    if (CheckKey(Keys.Escape) || CheckButton(Buttons.Back))
    {
        activeScreen.Hide();
        combatScreen.End();
        activeScreen = actionScreen;
        activeScreen.Show();
    }
}
```

What I will do now is add in being able to start conversations with NPCs in the game. That will be done in the **HandlePlayerInput** method. All I will do is when I check to see if the space key has been pressed check to see if the A button on the game pad has be pressed. This is the code for the **HandlePlayerInput** method.

```
private void HandlePlayerInput(GameTime gameTime)
{
    player.Update(gameTime);
    if (!inDialog)
    {
        foreach (NPC npc in npcs)
```

```
            npc.Update(gameTime);

        if (CheckAttackRadius(gameTime))
            return;

        if (CheckKey(Keys.Space) || CheckButton(Buttons.A))
            CheckSpeakingRadius();
    }
    if (!inDialog)
        HandlePlayerMovement();
}
```

The method for handling the movement of the sprite will be a little complex. Because a lot of the code for handling the menus stayed the same I decided to add in that as well. So, this tutorial is probably a little longer than I expected it to be. There is going to be a bit of new code here and a bit of explanation when it gets to moving the player character with the game pad. The first thing that I did was refactor the **HandlePlayerMovement** method so that the code that checks for movement using the keyboard is in a method of its own called **HandleKeyboardMovement** that returns a Vector2 called motion. I then wrote a method called **HandleGamepadMovement** that will handle movement using the game pad. This is the code for those three methods.

```
private void HandlePlayerMovement()
{
    Vector2 motion = new Vector2();
    playerSprite.IsAnimating = true;

    motion = HandleKeyboardMovement();

    if (motion == Vector2.Zero)
        motion = HandleGamepadMovement();

    if (motion != Vector2.Zero)
    {
        motion.Normalize();
        motion *= playerSprite.Speed;

        foreach (NPC npc in npcs)
        {
            float distance = Vector2.Distance(
                npc.Origin,
                playerSprite.Origin + motion);

            if (distance <= npc.CollisionRadius)
                return;
        }

        if (!actionScreen.CheckUnWalkableTile(playerSprite.Bounds, motion))
            playerSprite.Position += motion;

    }
    else
        playerSprite.IsAnimating = false;

    playerSprite.LockToMap();
    camera.LockToSprite(playerSprite);
    camera.LockCamera();
}
```

```
private Vector2 HandleKeyboardMovement()
{
    Vector2 motion = new Vector2();

    if (newState.IsKeyDown(Keys.Up) || newState.IsKeyDown(Keys.NumPad8))
    {
        motion.Y--;
        playerSprite.CurrentAnimation = AnimationKey.Up;
    }
    if (newState.IsKeyDown(Keys.Down) || newState.IsKeyDown(Keys.NumPad2))
    {
        motion.Y++;
        playerSprite.CurrentAnimation = AnimationKey.Down;
    }
    if (newState.IsKeyDown(Keys.Right) || newState.IsKeyDown(Keys.NumPad6))
    {
        motion.X++;
        playerSprite.CurrentAnimation = AnimationKey.Right;
    }
    if (newState.IsKeyDown(Keys.Left) || newState.IsKeyDown(Keys.NumPad4))
    {
        motion.X--;
        playerSprite.CurrentAnimation = AnimationKey.Left;
    }
    if (newState.IsKeyDown(Keys.NumPad9))
    {
        motion.X++;
        motion.Y--;
        playerSprite.CurrentAnimation = AnimationKey.Right;
    }
    if (newState.IsKeyDown(Keys.NumPad3))
    {
        motion.X++;
        motion.Y++;
        playerSprite.CurrentAnimation = AnimationKey.Right;
    }
    if (newState.IsKeyDown(Keys.NumPad1))
    {
        motion.X--;
        motion.Y++;
        playerSprite.CurrentAnimation = AnimationKey.Left;
    }
    if (newState.IsKeyDown(Keys.NumPad7))
    {
        motion.X--;
        motion.Y--;
        playerSprite.CurrentAnimation = AnimationKey.Left;
    }

    return motion;
}

private Vector2 HandleGamepadMovement()
{
    Vector2 motion = new Vector2();
    Vector2 direction = newPadState.ThumbSticks.Left;

    if (direction.Y < -0.5f)
```

```
    {
        motion.Y = 1;
        playerSprite.CurrentAnimation = AnimationKey.Down;
    }
    if (direction.Y > 0.5f)
    {
        motion.Y = -1;
        playerSprite.CurrentAnimation = AnimationKey.Up;
    }
    if (direction.X < -0.5f)
    {
        motion.X = -1;
        playerSprite.CurrentAnimation = AnimationKey.Left;
    }
    if (direction.X > 0.5f)
    {
        motion.X = 1;
        playerSprite.CurrentAnimation = AnimationKey.Right;
    }

    return motion;
}
```

Most of the code should look familiar to you. What will not be familiar is some of the code that I used in the **HandleGamepadMovement** method. In the **HandlePlayerMovement** method I set the **motion** vector to be the return result of the **HandleKeyboardMovement** method. If that is the zero vector, ie there was no movement with the keyboard, I set the **motion** vector to the return result of the **HandleGamepadMovement** method. There shouldn't be anything in the **HandleKeyboardMovement** method that you haven't seen before.

The first thing I did in that method is create a **Vector2** called **motion** that will hold the direction the player is moving the player character like with the keyboard. The next part will definitely be new to you if you have never programmed with the Xbox 360 controller before. The game pad has two thumb sticks, a left one and a right one, I will be using the left one to control the player character so I get the **Vector2** that holds the current state of the left thumb stick.

There are four if statements next that compare the X and Y values of the thumb stick to -0.5f and 0.5f. This might be a little puzzling to you. Why am I using 0.5f and -0.5f for this? The reason is that the thumb sticks are analog controls, not digital controls. Digital controls, like buttons on the game pad and keys on a keyboard, have two states: on or off. Values for the thumb sticks will range between 1.0f and -1.0f for their maximum and minimum values. This is because analog controls can measure a range of values. I'm comparing against -0.5f and 0.5f so that just a little push in any direction will not move the player's sprite. There has to be at least greater than half way for the sprite to move.

The first comparison is **direction.Y** with -0.5f. If that is true I set **motion.Y** to 1. You might be asking why the heck I'm doing that. The direction of thumb stick the Y is negative so you should be moving up! Well, the reason is the Y-axis of the screen and the thumb stick are reversed. Pushing the thumb stick down makes the Y value negative but to move the sprite down the screen you increment Y value of its position. That is why I also set the current animation to the down animation of the sprite. The reason I checked for up and down motion first is because when I was working with keyboard input I decided that the left and right animations would override the up and down animations. It just looked better in my opinion. I then compared **direction.Y** with 0.5f, set motion to -1 and the current animation to the up animation. The X-axis of the controller works the same way as the X-axis of the screen so I

did not have to reverse the directions. I'm pretty sure you can figure out what is going on there.

Well, that is it for this tutorial. This is something that I've wanted to do for a long time now and I'm happy that I finally did. I am planning on adding in more and more functionality that you would find in a complete role playing game. I encourage you to keep either visiting my site http://xna.jtmbooks.com or my blog, http://xna-rpg.blogspot.com for the latest news on my tutorials.