**Eyes of the Dragon Tutorials**
**Part 3**
**Even More Core Game Components**

I'm writing these tutorials for the MonoGame 3.8 framework using Visual Studio 2019. The tutorials will make more sense if they are read in order. You can find the list of tutorials on the Eyes of the Dragon page of my web blog. I will be making each version of the project available on GitHub here. It will be included on the page that links to the tutorials.

This is part two in my series of Eyes of the Dragon tutorials. What I'm planning on doing in this tutorial is fill out the start menu and add a new screen for game play. I'm going to add in a new control as well, a picture box control. I'm also going to add a new event to the **ControlManager** class that is fired when a control receives focus.

To get started, load up your project from last time. I'm going to start by adding the new event to the **ControlManager**. First, you will want to add in the event. I added in a new **Event** region to the regions of the control manager class. I changed the **NextControl** and **PreviousControl** method to fire the event if it is subscribed to. Add the following region near the **Field** region and change the **PreviousControl** and **NextControl** methods to the following.

```
#region Event Region
public event EventHandler FocusChanged;
#endregion
public void NextControl()
{
    if (Count == 0)
        return;

    int currentControl = selectedControl;
    this[selectedControl].HasFocus = false;

    do
    {
        selectedControl++;

        if (selectedControl == Count)
            selectedControl = 0;

        if (this[selectedControl].TabStop && this[selectedControl].Enabled)
        {
            if (FocusChanged != null)
                FocusChanged(this[selectedControl], null);

            break;
        }
    } while (currentControl != selectedControl);

    this[selectedControl].HasFocus = true;
}

public void PreviousControl()
{
    if (Count == 0)
        return;
```

```
            int currentControl = selectedControl;
            this[selectedControl].HasFocus = false;

            do
            {
                selectedControl--;

                if (selectedControl < 0)
                    selectedControl = Count - 1;

                if (this[selectedControl].TabStop && this[selectedControl].Enabled)
                {
                    if (FocusChanged != null)
                        FocusChanged(this[selectedControl], null);

                    break;
                }
            } while (currentControl != selectedControl);

            this[selectedControl].HasFocus = true;
        }
```

The new event is called **FocusChanged** and it will be fired when the control that has focus changes. It will send the control that received focus as the sender. I changed **NextControl** and **PreviousControl** to fire the event if it is subscribed to. I did that in the if statement that breaks out of the do-while loop if a control has a tab stop and it is enabled.

Before filling out the start screen I'm going to add in a screen for game play to take place. Right click the **GameScreens** folder in the solution explorer, select **Add** and then **Class**. Name this new class **GamePlayScreen**. This is the code for the class.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using MGRpgLibrary;

namespace EyesOfTheDragon.GameScreens
{
    public class GamePlayScreen : BaseGameState
    {
        #region Field Region
        #endregion

        #region Property Region
        #endregion

        #region Constructor Region

        public GamePlayScreen(Game game, GameStateManager manager)
            : base(game, manager)
        {
        }
```

```
        #endregion

        #region XNA Method Region

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

        protected override void LoadContent()
        {
            base.LoadContent();
        }

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

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

        #endregion

        #region Abstract Method Region
        #endregion
    }
}
```

Like the **StartMenuScreen**, this is just a bare bones class that inherits from **BaseGameState**. There are using statements to bring some MonoGame Framework classes into scope for our **MGRpgLibrary**. The next step is to add in a field for the **GamePlayScreen** in the **Game1** class and create an instance in the constructor. Add the following property to the **Game State** region. Also, change the constructor to the following.

```
        public GamePlayScreen GamePlayScreen { get; private set; }

        public Game1()
        {
            _graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
            IsMouseVisible = true;

            Components.Add(new InputHandler(this));

            _gameStateManager = new GameStateManager(this);
            Components.Add(_gameStateManager);

            TitleScreen = new TitleScreen(this, _gameStateManager);
            StartMenuScreen = new StartMenuScreen(this, _gameStateManager);
            GamePlayScreen = new GamePlayScreen(this, _gameStateManager);

            _gameStateManager.ChangeState(TitleScreen);
        }
```

I want to add in a new control, the picture box control. Right click the **Controls** folder in the **MGRpgLibary** project, select **Add** and then **Class**. Name this class **PictureBox**. This is the code for the **PictureBox** class.

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

namespace MGRpgLibrary.Controls
{
    public class PictureBox : Control
    {
        #region Field Region

        Texture2D image;
        Rectangle sourceRect;
        Rectangle destRect;

        #endregion

        #region Property Region

        public Texture2D Image
        {
            get { return image; }
            set { image = value; }
        }
        public Rectangle SourceRectangle
        {
            get { return sourceRect; }
            set { sourceRect = value; }
        }
        public Rectangle DestinationRectangle
        {
            get { return destRect; }
            set { destRect = value; }
        }

        #endregion

        #region Constructors

        public PictureBox(Texture2D image, Rectangle destination)
        {
            Image = image;
            DestinationRectangle = destination;
            SourceRectangle = new Rectangle(0, 0, image.Width, image.Height);
            Color = Color.White;
        }

        public PictureBox(Texture2D image, Rectangle destination, Rectangle source)
        {
            Image = image;
            DestinationRectangle = destination;
            SourceRectangle = source;
```

```
            Color = Color.White;
        }

        #endregion

        #region Abstract Method Region

        public override void Update(GameTime gameTime)
        {
        }

        public override void Draw(SpriteBatch spriteBatch)
        {
            spriteBatch.Draw(image, destRect, sourceRect, Color);
        }

        public override void HandleInput(PlayerIndex playerIndex)
        {
        }
        #endregion

        #region Picture Box Methods

        public void SetPosition(Vector2 newPosition)
        {
            destRect = new Rectangle(
            (int)newPosition.X,
            (int)newPosition.Y,
            sourceRect.Width,
            sourceRect.Height);
        }

        #endregion
    }
}
```

There are a few extra using statements to bring the **XNA Framework**, **XNA Framework Input**, and **XNA Framework Graphics** classes into scope. The class inherits from the base abstract class **Control** so it can be added to the **ControlManager** class.

There are three new fields and properties to expose them. The fields are **image**, **destRect**, and **sourceRect**. The fields that expose them are **Image**, **DestinationRectangle**, and **SourceRectangle** respectively. The first one **image** is of course the image for the picture box and is a **Texture2D**. The next one **destRect** is the destination rectangle where the image is to be drawn. The last, **sourceRect**, is the source rectangle in the image. This will be handy down the road as you will see.

There are two constructors in this class. The first takes a **Texture2D** for the picture box and a **Rectangle** for the destination of the image on the screen. That constructor assigns the **image** field and **destRect** field to the values passed in. It also sets the **sourceRect** field to be the entire image using zero for the **X** and **Y** properties of the rectangle and the **Width** and **Height** properties of the **Texture2D** for the **Width** and **Height** of the rectangle. That means the entire **Texture2D** will be used as the source rectangle. The second constructor takes a **Texture2D**, and two **Rectangle** parameters. The **Texture2D** is the image. The first **Rectangle** is the destination and the second **Rectangle** is the source in the image. It just assigns the fields using the parameters passed in. I also set the **Color** property to white.

The base class **Control** has three abstract methods that must be implemented: **Draw**, **Update**, and **HandleInput**. They don't have to do anything however. The one that I added code to was the **Draw** method. The **Draw** method draws the **Texture2D** using the **image**, **destRect**, **sourceRect**, and **color** fields.

I added in some of the functionality by adding a **SetPosition** method that takes a **Vector2** for the new position of the picture box. I create a new destination rectangle by casing the X and Y properties of the vector passed to integers and use the **Width** and **Height** properties of the source rectangle.

Before I get to the **StartMenuScreen** I want to add in a couple graphics for GUI controls. Again, I'd like to thank Tuckbone from my forum for providing the graphics. You can download the graphics from https://cynthiamcmahon.ca/blog/downloads/guigraphics.zip. After you've downloaded the graphics extract them to a folder. Go back to your game and open the **MonoGame Pipeline Tool.** Right click the **Content** node, select **Add** and then **New Folder**. Name this new folder **GUI**. Right click the **GUI** folder, select **Add** and then **Existing Item**. Navigate to where you extracted the controls and add them all.

Open up the code for the **StartMenuScreen**. I made a lot of changes to the screen as it was basically a place holder before. I just noticed that I had the code for checking if the player presses Escape in the **Draw** method. That was a mistake on my part. This is the code for the **StartMenuScreen**.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Content;
using MGRpgLibrary;
using MGRpgLibrary.Controls;

namespace EyesOfTheDragon.GameScreens
{
    public class StartMenuScreen : BaseGameState
    {
        #region Field region

        PictureBox backgroundImage;
        PictureBox arrowImage;
        LinkLabel startGame;
        LinkLabel loadGame;
        LinkLabel exitGame;
        float maxItemWidth = 0f;

        #endregion

        #region Property Region
        #endregion

        #region Constructor Region

        public StartMenuScreen(Game game, GameStateManager manager)
        : base(game, manager)
        {
```

```
}

#endregion

#region XNA Method Region

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

protected override void LoadContent()
{
    base.LoadContent();

    ContentManager Content = Game.Content;

    backgroundImage = new PictureBox(
        Content.Load<Texture2D>(@"Backgrounds\titlescreen"),
        GameRef.ScreenRectangle);

    ControlManager.Add(backgroundImage);

    Texture2D arrowTexture = Content.Load<Texture2D>(@"GUI\leftarrowUp");
    arrowImage = new PictureBox(
        arrowTexture,
        new Rectangle(
            0,
            0,
            arrowTexture.Width,
            arrowTexture.Height));

    ControlManager.Add(arrowImage);

    startGame = new LinkLabel();
    startGame.Text = "The story begins";
    startGame.Size = startGame.SpriteFont.MeasureString(startGame.Text);
    startGame.Selected += new EventHandler(menuItem_Selected);

    ControlManager.Add(startGame);

    loadGame = new LinkLabel();
    loadGame.Text = "The story continues";
    loadGame.Size = loadGame.SpriteFont.MeasureString(loadGame.Text);
    loadGame.Selected += menuItem_Selected;
    ControlManager.Add(loadGame);

    exitGame = new LinkLabel();
    exitGame.Text = "The story ends";
    exitGame.Size = exitGame.SpriteFont.MeasureString(exitGame.Text);
    exitGame.Selected += menuItem_Selected;

    ControlManager.Add(exitGame);
    ControlManager.NextControl();
    ControlManager.FocusChanged += new EventHandler(ControlManager_FocusChanged);

    Vector2 position = new Vector2(350, 500);

    foreach (Control c in ControlManager)
    {
```

```
            if (c is LinkLabel)
            {
                if (c.Size.X > maxItemWidth)
                    maxItemWidth = c.Size.X;

                c.Position = position;
                position.Y += c.Size.Y + 5f;
            }
        }

        ControlManager_FocusChanged(startGame, null);
    }

    void ControlManager_FocusChanged(object sender, EventArgs e)
    {
        Control control = sender as Control;

        Vector2 position = new Vector2(control.Position.X + maxItemWidth + 10f,
            control.Position.Y);

        arrowImage.SetPosition(position);
    }

    private void menuItem_Selected(object sender, EventArgs e)
    {
        if (sender == startGame)
        {
            StateManager.PushState(GameRef.GamePlayScreen);
        }

        if (sender == loadGame)
        {
            StateManager.PushState(GameRef.GamePlayScreen);
        }

        if (sender == exitGame)
        {
            GameRef.Exit();
        }
    }

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

    public override void Draw(GameTime gameTime)
    {
        GameRef.SpriteBatch.Begin();
        base.Draw(gameTime);
        ControlManager.Draw(GameRef.SpriteBatch);
        GameRef.SpriteBatch.End();
    }

    #endregion

    #region Game State Method Region
    #endregion
}
```

}

You may be wondering why I haven't added a menu component to the game when I have a menu on this screen. You could add a menu component and use one, and there is nothing wrong with that route. I instead decided to use the controls that I created and the control manager. The way the menu is going to work is it is going to display the items using link labels. The currently selected link label will have an arrow to the right of it.

There are some using statements to bring a few of the MonoGame Framework and our **MGRpgLibrary** members into scope. The class inherits from **BaseGameState** so it can be used in our game state manager.

There are six fields in the class. The first two are **PictureBox** controls. The one is for the background image and the other is for the arrow. There are also three **LinkLabel** controls, one for each of the three menu items: one to start a new game, one to load an old game, and one to exit the game. The last field is a float and it is used to control where the arrow appears in the menu. It holds the maximum width of the link labels. The constructor for the class just calls the constructor of the base class with the values passed in.

The **LoadContent** method is where I create the controls on the form, wire their event handlers, and position them on the form. Again, you want to do this after the call to **base.LoadContent** so that the **ControlManager** exists.

The first control I create is the **PictureBox** for the background image. I load in the same image from the title screen and set the destination rectangle of the picture box to our **ScreenRectangle**. I then add it to the **ControlManager**.

I then load in the image for the other picture box. I create the picture box for that control passing in the image I loaded and a rectangle at coordinates (0, 0) with the width and height of the image. Its position at the moment isn't important as it will be changed at the end of the method.

The next step is to create the **startGame LinkLabel**. I set the **Text** property to "The story begins." and set the **Size** property to the size of the string using the **MeasureString** method from the **SpriteFont** property. I then wire the event handler for the **Selected** event but not the default handler. I set it to a handler that will handle the **Selected** event of all menu items. I then add it to the **ControlManager**. I create the **loadGame** and **exitGame** controls the same way. I then call the **NextControl** method of the control manager to have **startGame** as the currently selected control. I then wire the event handler for the **FocusChanged** event of the **ControlManager**.

I create a **Vector2** that holds the position of the first **LinkLabel** on the screen. Then I loop over all of the controls that were added to the control manage. In the loop I check to see if the control is a **LinkLabel**. If it is I compare the **X** property of its size with **maxItemWidth** field. If it is greater I set **maxItemWidth** to that value. I then set the **Position** property of the control to the **Vector2** I created. I then increase the **Y** property of the **Vector2** by the **Y** property of the **Size** property of the control plus five pixels.

At the end of the method I call the **FocusChanged** event handler passing in the **startGame** control and null. What this does is call the code that positions the arrow to the right of the sender. In this case, the arrow will be to the right of the **startGame** item.

The **ControlManager_FocusChanged** method sets the **sender** parameter as a control. I then create a **Vector2**, **position**, using **X** value of the **Position** property of the control and the **maxItemWidth** field plus 10 pixels for the **X** value of the **Vector2**. For the **Y** value of the vector I use the **Y** value of the controls position.

In the **menuItem_Selected** method I check to see which link label was selected. I compare the **sender** parameter with the different link labels. If it is the **startGame** or **loadGame** link label I call the push the **GamePlayScreen** on top of the stack. If it was the **exitGame** link label I exit out of the game. It would probably be a good idea to have a little pop up screen that asks if that is really what you want to do and I will add that in some time.

In the **Update** method I call the **Update** method of the **ControlManager** class. I pass the **gameTime** parameter from the **Update** method and **playerIndexInControl**, a field of **PlayerIndex** that I added to the **BaseGameState** screen last tutorial. For the **Draw** method I call the **Begin** method of the sprite batch from our game before the call to **base.Draw**. This allows any child components to be drawn before we draw our components. I then call the **Draw** method of the **ControlManager**. Finally, the call to **End** of the sprite batch.

I think this is enough for this tutorial. I'd like to try and keep them to a reasonable length so that you don't have too much to digest at once. I encourage you to visit the news page of my site, https://cynthiamcmahon.ca/blog/, for the latest news on my tutorials.

Good luck in your game programming adventures!
Cynthia