Eyes of the Dragon Tutorials
Part 59
User Experience Updates

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.

In this tutorial I will be making some user experience tweaks. Mainly, I will be adding in mouse support. So, let's get started.

The plan for this tutorial is to add more mouse support to the game. There is one big problem with the control manager. You can only move between controls using the keyboard or game pad. I'm going to update the ControlManager to allow movement between controls as the mouse moves in and out of them. First, I need to add a virtual method to the Control class that measures the control as a Rectangle. Right now we have the size property as a Vector2 but we really need it as a Rectangle. Add the following method to the Control class.

```
public virtual Rectangle GetBounds()
{
    return new Rectangle((int)position.X, (int)position.Y, (int)size.X, (int)size.Y);
}
```

It is a simple method that we can override in any child classes that are more than just text. It creates a Rectangle using the position and size fields and returns it.

Now, I am going to update the ControlManager to allow for movement between controls based on the position of the mouse. It is what players would expect to happen and that is all part of a good user experience. That the controls of a game are intuitive, and that they make sense. Update the Draw method of the ControlManager to the following.

```
public void Draw(SpriteBatch spriteBatch)
{
    Vector2 mouse = InputHandler.MouseAsVector2;

    foreach (Control c in this)
    {
        if (c.Visible)
        {
            c.Draw(spriteBatch);

            if (c.GetBounds().Contains(mouse))
            {
                foreach (Control control in this)
                {
                    control.HasFocus = false;
                }

                c.HasFocus = true;
```

```
            }
        }
    }
}
```

I know, logic should not be applied to the Draw method. It made sense, though, because everything is based on the position of the mouse. So, I decided to handle it in the Draw method. Also, I was already cycling over the controls. First thing I do is grab the position of the mouse as a Vector2. If the control is visible I call the GetBounds method on the control then check if that contains the mouse. If it does, I loop over all of the controls in the control manager and set their HasFocus property to false. I then set the HasFocus property of that control to true.

I will now add mouse support to the left/right selector, because that has really been bothering me for some time. Add these two fields to the field region and replace the Draw and HandleInput methods with the following versions. Also, there is an override of the GetBounds method.

```
bool overLeft;
bool overRight;

public override void Draw(SpriteBatch spriteBatch)
{
    Vector2 drawTo = position;
    Rectangle dest = new Rectangle((int)position.X, (int)position.Y, 0, 0);
    Vector2 mouse = InputHandler.MouseAsVector2;

    if (selectedItem != 0)
    {
        spriteBatch.Draw(leftTexture, drawTo, Color.White);
        dest.Width = leftTexture.Width;
        dest.Height = leftTexture.Height;
    }
    else
    {
        spriteBatch.Draw(stopTexture, drawTo, Color.White);
        dest.Width = stopTexture.Width;
        dest.Height = stopTexture.Height;
    }

    overLeft = dest.Contains(mouse);

    drawTo.X += leftTexture.Width + 5f;

    float itemWidth = spriteFont.MeasureString(items[selectedItem]).X;
    float offset = (maxItemWidth - itemWidth) / 2;

    drawTo.X += offset;

    if (hasFocus)
        spriteBatch.DrawString(spriteFont, items[selectedItem], drawTo, selectedColor);
    else
        spriteBatch.DrawString(spriteFont, items[selectedItem], drawTo, Color);

    drawTo.X += -1 * offset + maxItemWidth + 5f;

    dest.X = (int)drawTo.X;

    if (selectedItem != items.Count - 1)
```

```
    {
        spriteBatch.Draw(rightTexture, drawTo, Color.White);
        dest.Width = rightTexture.Width;
        dest.Height = rightTexture.Height;
    }
    else
    {
        spriteBatch.Draw(stopTexture, drawTo, Color.White);
        dest.Width = stopTexture.Width;
        dest.Height = stopTexture.Height;
    }

    overRight = dest.Contains(mouse);
}

public override void HandleInput(PlayerIndex playerIndex)
{
    if (items.Count == 0)
        return;

    if (InputHandler.ButtonReleased(Buttons.LeftThumbstickLeft, playerIndex) ||
        InputHandler.ButtonReleased(Buttons.DPadLeft, playerIndex) ||
        InputHandler.KeyReleased(Keys.Left) ||
        (InputHandler.CheckMouseReleased(MouseButton.Left) && overLeft))
    {
        selectedItem--;

        if (selectedItem < 0)
            selectedItem = 0;

        OnSelectionChanged();
    }

    if (InputHandler.ButtonReleased(Buttons.LeftThumbstickRight, playerIndex) ||
        InputHandler.ButtonReleased(Buttons.DPadRight, playerIndex) ||
        InputHandler.KeyReleased(Keys.Right) ||
        (InputHandler.CheckMouseReleased(MouseButton.Left) && overRight))
    {
        selectedItem++;

        if (selectedItem >= items.Count)
            selectedItem = items.Count - 1;

        OnSelectionChanged();
    }
}

public override Rectangle GetBounds()
{
    return new Rectangle(
        (int)Position.X,
        (int)Position.Y,
        maxItemWidth + 10 + leftTexture.Width + rightTexture.Width,
        rightTexture.Height);
}
```

The fields are overLeft and overRight and their intent can be discerned from their names. The first tells if the mouse cursor is over the left side of the selector, and the second tells if the mouse cursor is over the right side of the selector.

I added two local variables to the Draw method: dest and mouse. Again, their purpose should be clear from the names. The first, dest is the destination rectangle that describes the portion of the control that is being queried to see if it contains that mouse. The second is mouse and it holds the position of the mouse as a Vector2. The destination has its X and Y coordinates set to the position the control is being drawn to.

When checking to see if the currently selected item is not the first I set the width and height of the destination to the width and height of the left pointing texture. Otherwise, I set it to width and height of the stop texture. Technically I didn't need to do that step because we don't care if the mouse is over the stop texture. Though, I know a lot of games where they would play a buzz sound if you can't go back. I then set overLeft to be if the destination contains the mouse.

The case for the right texture is similar. If we are not at the end of the list I set the width and height to the width and height of the right texture. Otherwise, it is set to the stop texture. I then capture the overRight field the same way as the overLeft.

In the HandleInput method I updated the condition where I check to see the player has selected to move the selection left or right. For moving the selection left, I added an or clause that checks to see if the left mouse button has been released, and that overLeft is true. I do something similar for the right side of the control.

The next thing that I'm going to change is the control manager. The reason being, is that if a left/right selector is the currently select control and you click on it, things work as expected. If it is not, then nothing happens. It makes more sense to have a control that can do something gain focus when the mouse enters it. Update the Draw method of the control manager to the following.

The override of the GetBounds method is a little complicated. The X and Y coordinates are the position of the control. The width is what is complex. It is the width of the left selector, the maximum width of the items contained, the width of the right selector plus the 10 pixel padding between the selectors and the text. I chose an arbitrary texture for the height of the rectangle.

The LinkLabel needs to be updated as well. It is a relatively short class so I will give you the code for the entire 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 LinkLabel : Control
    {
        #region Fields and Properties

        Color selectedColor = Color.Red;
```

```
        public Color SelectedColor
        {
            get { return selectedColor; }
            set { selectedColor = value; }
        }

        #endregion

        #region Constructor Region

        public LinkLabel()
        {
            TabStop = true;
            HasFocus = false;
            Position = Vector2.Zero;
        }

        #endregion

        #region Abstract Methods

        public override void Update(GameTime gameTime)
        {
            size = SpriteFont.MeasureString(Text);
        }

        public override void Draw(SpriteBatch spriteBatch)
        {
            if (hasFocus)
                spriteBatch.DrawString(SpriteFont, Text, Position, selectedColor);
            else
                spriteBatch.DrawString(SpriteFont, Text, Position, Color);
        }

        public override void HandleInput(PlayerIndex playerIndex)
        {
            if (!HasFocus)
                return;

            if (InputHandler.KeyReleased(Keys.Enter) ||
                InputHandler.ButtonReleased(Buttons.A, playerIndex))
                base.OnSelected(null);

            if (InputHandler.CheckMouseReleased(MouseButton.Left))
            {

                if (GetBounds().Contains(InputHandler.MouseAsPoint))
                    base.OnSelected(null);
            }
        }

        #endregion
    }
}
```

Relatively, straightforward changes. In the Update method I set the size of the control using the MeasureString method of the font. I could have equally well done this in the Draw method. In the end, it just matters that we update the size every frame because the text could, in theory, change.

In the HandleInput method I now use GetBounds to test if the LinkLabel contains the mouse as a point.

If you build and run the game now the control manager responds to the mouse entering and leaving controls. I didn't in this tutorial, but there could be a case for mouse enter and mouse leaves events on controls. In the event that we need them in the future, I will add them.

There is something that has been bothering for a long time, and that is the conversation states. In particular the shop state. You should be able to navigate using the mouse, and exit the state if the escape key is pressed. I updated the GameScene class in the MGRpgLibrary class to allow for the select of options using the mouse. Add the following property and replace the Draw method of the GameScene class with the following code.

```
public bool IsOver { get; private set; }

public virtual void Draw(GameTime gameTime, SpriteBatch spriteBatch, Texture2D portrait = null)
{
    Vector2 selectedPosition = new Vector2();
    Rectangle portraitRect = new Rectangle(25, 25, 425, 425);
    Color myColor;

    if (selected == null)
        selected = game.Content.Load<Texture2D>(@"GUI\rightarrowUp");

    if (textPosition == Vector2.Zero)
        SetText(text);

    if (texture == null)
        texture = game.Content.Load<Texture2D>(@"GameScenes\" + textureName);

    spriteBatch.Draw(texture, Vector2.Zero, Color.White);

    if (portrait != null)
        spriteBatch.Draw(portrait, portraitRect, Color.White);

    spriteBatch.DrawString(font,
        text,
        textPosition,
        Color.White);

    Vector2 position = menuPosition;
    Vector2 mouse = InputHandler.MouseAsVector2;
    Rectangle dest = new Rectangle(0, 0, 1280, font.LineSpacing);
    IsOver = false;

    for (int i = 0; i < options.Count; i++)
    {
        dest.Y = (int)position.Y;

        if (dest.Contains(mouse))
        {
            selectedIndex = i;
            IsOver = true;
        }

        if (i == SelectedIndex)
```

```
        {
            myColor = HighLightColor;
            selectedPosition.X = position.X - 35;
            selectedPosition.Y = position.Y;
            spriteBatch.Draw(selected, selectedPosition, Color.White);
        }
        else
            myColor = NormalColor;

        spriteBatch.DrawString(font,
            options[i].OptionText,
            position,
            myColor);

        position.Y += font.LineSpacing + 5;
    }
}
```

Just a simple auto-property with a private set that lets us know that the mouse is over an option. The changes to the Draw method are similar to the ControlManager. I create a local Rectangle variable that will hold a rectangle that describes the current line of the scene option. I also capture the coordinates of the mouse as a Vector2. Then, when looping over the items I set the Y property of the rectangle to be the position of the line. If that rectangle contains the mouse I set the selectedIndex field to be that item. I also set IsOver to true.

The next thing I'm going to change is the ShopState. There are two things that I want to do. The first is exit the state if the player is talking to the merchant and the escape key is pressed. Second, I want to activate options if the mouse is over a scene option and the player presses the left mouse button. Replace the Update method of the ShopState with the following code.

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

    scene.Update(gameTime, playerIndexInControl);

    switch (State)
    {
        case ShopStateType.Buy:
            if (isFirst)
            {
                isFirst = false;
                break;
            }

            if (InputHandler.KeyReleased(Keys.Down) ||
                InputHandler.KeyReleased(Keys.S))
            {
                selected++;
                if (selected >= merchant.Backpack.Items.Count)
                {
                    selected = 0;
                }
            }

            if (InputHandler.KeyReleased(Keys.Up) ||
```

```
            InputHandler.KeyReleased(Keys.W))
        {
            selected--;
            if (selected < 0)
            {
                selected = merchant.Backpack.Items.Count - 1;
            }
        }

        if (InputHandler.KeyReleased(Keys.Space) ||
            InputHandler.KeyReleased(Keys.Enter) ||
            (InputHandler.CheckMouseReleased(MouseButton.Left) && mouseOver))
        {
            if (selected != -1 &&
                GamePlayScreen.Player.Gold >=
                merchant.Backpack.PeekItem(
                    merchant.Backpack.Items[selected].Item.Name).Price)
            {
                if (selected >= merchant.Backpack.Items.Count)
                {
                    return;
                }

                GamePlayScreen.Player.Gold -=
                    merchant.Backpack.PeekItem(
                        merchant.Backpack.Items[selected].Item.Name).Price;
                GamePlayScreen.Player.Backpack.AddItem(
                    merchant.Backpack.GetItem(
                        merchant.Backpack.Items[selected].Item.Name));
            }
        }
        break;
    case ShopStateType.Sell:
        if (isFirst)
        {
            isFirst = false;
            break;
        }

        if (InputHandler.KeyReleased(Keys.Down) ||
            InputHandler.KeyReleased(Keys.S))
        {
            selected++;

            if (selected >= GamePlayScreen.Player.Backpack.Items.Count)
            {
                selected = 0;
            }
        }

        if (InputHandler.KeyReleased(Keys.Up) ||
            InputHandler.KeyReleased(Keys.W))
        {
            selected--;
            if (selected < 0)
            {
                selected = GamePlayScreen.Player.Backpack.Items.Count - 1;
            }
        }
```

```
            if ((InputHandler.KeyReleased(Keys.Space) ||
                InputHandler.KeyReleased(Keys.Enter) ||
                (InputHandler.CheckMouseReleased(MouseButton.Left) && mouseOver)))
            {
                if (selected >= 0)
                {
                    GameItem item = GamePlayScreen.Player.Backpack.GetItem(
                        GamePlayScreen.Player.Backpack.Items[selected].Item.Name);
                    GamePlayScreen.Player.Gold += item.Item.Price * 3 / 4;
                }
            }
            break;
        case ShopStateType.Talk:
            if (InputHandler.KeyReleased(Keys.Space) ||
                InputHandler.KeyReleased(Keys.Enter) ||
                (InputHandler.CheckMouseReleased(MouseButton.Left) && scene.IsOver))
            {
                if (scene.SelectedIndex == 0)
                {
                    isFirst = true;
                    State = ShopStateType.Buy;
                    //selected = -1;
                    return;
                }

                if (scene.SelectedIndex == 1)
                {
                    isFirst = true;
                    State = ShopStateType.Sell;
                    //selected = -1;
                    return;
                }

                if (scene.SelectedIndex == 2 && State == ShopStateType.Talk)
                {
                    StateManager.PopState();
                }
            }
            break;
    }

    if (InputHandler.CheckMouseReleased(MouseButton.Right) ||
InputHandler.KeyReleased(Keys.Escape))
    {
        switch (State)
        {
            case ShopStateType.Buy:
            case ShopStateType.Sell:
                State = ShopStateType.Talk;
                break;
            case ShopStateType.Talk:
                StateManager.PopState();
                break;
        }
    }
}
```

What I did was when I check to see if the space bar or enter key have been pressed all check to see if IsOver on the scene is true and the left mouse button has been released. Also, when I check to see if

the right mouse button has been released I check to see if the escape key has been released. If either is true there is a new case that pops the state off the stack.

The last thing I am going to change today is the conversation state. This is pretty much the same as the shop state, since they are siblings. Replace the Update method of the ConversationScreen with the following code.

```
public override void Update(GameTime gameTime)
{
    conversation.Update(gameTime);

    if (InputHandler.KeyReleased(Keys.Enter) || InputHandler.KeyReleased(Keys.Space) ||
        InputHandler.ButtonReleased(Buttons.A, PlayerIndex.One) ||
        (InputHandler.CheckMouseReleased(MouseButton.Left) &&
conversation.CurrentScene.IsOver))
    {
        InputHandler.Flush();
        SceneAction action = conversation.CurrentScene.OptionAction;

        switch (action.Action)
        {
            case ActionType.Talk:
                conversation.ChangeScene(conversation.CurrentScene.OptionScene);
                break;
            case ActionType.Quest:
                conversation.ChangeScene(conversation.CurrentScene.OptionScene);
                break;
            case ActionType.Change:
                conversation =
                conversations.GetConversation(conversation.CurrentScene.OptionScene);
                conversation.StartConversation();
                break;
            case ActionType.End:
                StateManager.PopState();
                break;
        }
    }

    base.Update(gameTime);
}
```

The exact same concept as the ShopState. When I check to see if the enter key or space bar have been pressed I also check to see if the left mouse button has been pressed and the the mouse if over a scene item for the current scene property of the conversation.

That is it for this tutorial. I accomplished what I had planned on, and I don't want to get into something new at this point. I accomplished most of what I intended, and I don't want to venture further in this tutorial. So, please continue to visit my blog, https://cynthiamcmahon.ca/blog/, for the latest news on my tutorials and other goodness.

Good luck with your Game Programming Adventures!

Cynthia