Eyes of the Dragon Tutorials
Part 6
Character Generator

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 adding in a new game screen to create a character. First I will be adding in a new control, a left and right selector. When the control is selected pressing the left or right arrow key will cycle through a collection of strings. Right click the **Controls** folder in the **MGRpgLibrary** project in the solution explore. Select **Add**, and then **Class**. Name this new class **LeftRightSelector**. The code for this class follows.

```
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 LeftRightSelector : Control
    {
        #region Event Region

        public event EventHandler SelectionChanged;

        #endregion

        #region Field Region

        List<string> items = new List<string>();
        Texture2D leftTexture;
        Texture2D rightTexture;
        Texture2D stopTexture;
        Color selectedColor = Color.Red;
        int maxItemWidth;
        int selectedItem;

        #endregion

        #region Property Region

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

        public int SelectedIndex
        {
            get { return selectedItem; }
```

```
        set { selectedItem = (int)MathHelper.Clamp(value, 0f, items.Count); }
    }

    public string SelectedItem
    {
        get { return Items[selectedItem]; }
    }

    public List<string> Items
    {
        get { return items; }
    }

    #endregion

    #region Constructor Region

    public LeftRightSelector(Texture2D leftArrow, Texture2D rightArrow, Texture2D stop)
    {
        leftTexture = leftArrow;
        rightTexture = rightArrow;
        stopTexture = stop;
        TabStop = true;
        Color = Color.White;
    }

    #endregion

    #region Method Region
    public void SetItems(string[] items, int maxWidth)
    {
        this.items.Clear();
        foreach (string s in items)
            this.items.Add(s);
        maxItemWidth = maxWidth;
    }
    protected void OnSelectionChanged()
    {
        if (SelectionChanged != null)
        {
            SelectionChanged(this, null);
        }
    }

    #endregion

    #region Abstract Method Region

    public override void Update(GameTime gameTime)
    {
    }

    public override void Draw(SpriteBatch spriteBatch)
    {
        Vector2 drawTo = position;

        if (selectedItem != 0)
            spriteBatch.Draw(leftTexture, drawTo, Color.White);
        else
            spriteBatch.Draw(stopTexture, drawTo, Color.White);
```

```
            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;

            if (selectedItem != items.Count - 1)
                spriteBatch.Draw(rightTexture, drawTo, Color.White);
            else
                spriteBatch.Draw(stopTexture, drawTo, Color.White);
        }

        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))
            {
                selectedItem--;
                if (selectedItem < 0)
                    selectedItem = 0;
                OnSelectionChanged();
            }

            if (InputHandler.ButtonReleased(Buttons.LeftThumbstickRight, playerIndex) ||
                InputHandler.ButtonReleased(Buttons.DPadRight, playerIndex) ||
                InputHandler.KeyReleased(Keys.Right))
            {
                selectedItem++;
                if (selectedItem >= items.Count)
                    selectedItem = items.Count - 1;
                OnSelectionChanged();
            }
        }

        #endregion
    }
}
```

There are a few using statements to bring some of the MonoGame framework classes into scope. There is an event associated with this control, **SelectionChanged**, that if subscribed to will fire if the selection in the selector is changed.

There are a number of fields in this class. There is a **List<string>** called **items** that holds the items for the selector. There are three **Texture2D** fields. They hold a stop bar, a left arrow, and a right arrow. I again want to thank Tuckbone for taking the time to make the graphics. There is a **Color** field,

selectedColor, that holds the color to draw the control if it is currently selected. There are also two integer fields, **maxItemWidth** and **selectedIndex**. The first is the width of the longest item in the selector. It is needed because I will be centering items in the selector. The last is the index of the currently selected item in the selector.

There are a few properties in the class to expose the fields. The **Items** property exposed the **items** field. It is a read only, get, property but you can manipulate the **items** field with it. You just can't assign to it directly. The get part of the **SelectedIndex** property returns the **selectedIndex** field. The set part sets the **selectedIndex** field but uses the **Clamp** method of **MathHelper** to make sure the field stays with in the range of items. The **SelectedItem** property just returns the item that is currently selected. The **SelectedColor** property exposes the **selectedColor** field.

The constructor for **LeftRightSelector** takes three parameters. They are an image for displaying the selector can move left, an image for displaying the selector can move right and a stop bar displaying the selection can not move in that direction. The constructor sets the fields with the values passed in, sets the **TabStop** property to true, and the colour of the control to white.

There are two methods that are specific to this control. The first is the **SetItems** method. It takes an array of strings for the items and an integer for the maximum width. It loops through the array of strings passed in and adds them to the **items** collection. It also assigns the **maxItemWidth** field with the value passed in. The **OnSelection** method will be called if the **SelectionChanged** if the selection is changed. It checks to see if the **SelectionChanged** event is subscribed to. If it is it fires the **SelectionChanged** event using itself for the sender and null for the event arguments.

The **Update** method for this class does nothing. It is there because it is part of the base abstract class. It would be used if you were doing some sort of animation with the control or other things that required updating. The **HandleInput** method will handle the input for the control. It checks to make sure there are items in the **items** collection to make sure that there is something to work with. If there aren't it exits the method. There is next an if statement that checks to see if the left thumb stick has been released in the left direction, the direction pad has released in the left direction or the left key has been released. If they have I decrease the selected item by one. If the selected item is less than zero it is set to zero. It then checks for the releases to the right. I increment the selected item by one. If the selected item is greater than or equal to the number of items I set the selected item to be the number of items minus one.

The **Draw** method draws the control. The first step is to set a local variable to the position the control is to be drawn at. The reason is that there are several parts to the control. There are the left and right arrows, the stop bar, and the text. The first step is to see if the left arrow needs to be drawn. It is only drawn if the selected item is not the first item, the item at index zero. If there is one I draw the left arrow. If the left arrow does not need to be drawn I draw the stop bar. I then increment the X position of drawing point by the width of the left arrow plus 5 pixels for padding. I decided to center the text horizontally in the control. First you need to find the width of the text using the **MeasureString** method of the **SpriteFont** class. I don't need the height of the string so I just used the **X** for the string.

The **offset** value is used to determine where to draw the text relative to the **drawTo** local variable. I

then add the **offset** value to the **X** part of the **drawTo** variable. You want the control drawn in the **selectedColor** value if it has focus so there is an if statement to check if the control has focus before drawing the text. If it selected, it is drawn using the **selectedColor** and if not the **Color** value. To figure out where to draw the right arrow you remove the offset, add the **maxItemWidth** field and the padding of five pixels. The last step is to see if the right arrow should be drawn by checking if the selected item is the last item. It it doesn't need to be drawn then you draw the stop bar.

Now it is time to add in the new screen to the game. Right click the **GameScreens** folder, select **Add** and then **Class**. Name this new class **CharacterGeneratorScreen**. The code follows next.

```
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;
using MGRpgLibrary.Controls;

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

        LeftRightSelector nameSelector;
        LeftRightSelector genderSelector;
        LeftRightSelector classSelector;
        PictureBox backgroundImage;

        readonly string[] genderItems = { "Male", "Female", "Non-Binary" };
        readonly string[] classItems = { "Fighter", "Wizard", "Rogue", "Priest" };
        readonly string[] maleName = { "Balthazar", "Logan", "Alfred", "Johnson" };
        readonly string[] femaleName = { "Lucinda", "Cynthia", "Ezmarelda", "Millicent" };
        readonly string[] nbName = { "Jaime", "Kelly", "Jordan", "Pat" };

        #endregion

        #region Property Region
        #endregion

        #region Constructor Region

        public CharacterGeneratorScreen(Game game, GameStateManager stateManager)
            : base(game, stateManager)
        {
        }

        #endregion

        #region MonoGame Method Region
        public override void Initialize()
        {
            base.Initialize();
        }
```

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

    CreateControls();
}

public override void Update(GameTime gameTime)
{
    ControlManager.Update(gameTime, PlayerIndex.One);

    base.Update(gameTime);
}

public override void Draw(GameTime gameTime)
{
    GameRef.SpriteBatch.Begin();

    base.Draw(gameTime);

    ControlManager.Draw(GameRef.SpriteBatch);

    GameRef.SpriteBatch.End();
}

#endregion

#region Method Region
private void CreateControls()
{
    Texture2D leftTexture = Game.Content.Load<Texture2D>(@"GUI\leftarrowUp");
    Texture2D rightTexture = Game.Content.Load<Texture2D>(@"GUI\rightarrowUp");
    Texture2D stopTexture = Game.Content.Load<Texture2D>(@"GUI\StopBar");

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

    ControlManager.Add(backgroundImage);

    Label label1 = new Label
    {
        Text = "Who will search for the Eyes of the Dragon?"
    };
    label1.Size = label1.SpriteFont.MeasureString(label1.Text);
    label1.Position = new Vector2(
        (GameRef.Window.ClientBounds.Width - label1.Size.X) /2, 150);

    ControlManager.Add(label1);

    nameSelector = new LeftRightSelector(leftTexture, rightTexture, stopTexture);
    nameSelector.SetItems(maleName, 125);
    nameSelector.Position = new Vector2(label1.Position.X, 200);

    genderSelector = new LeftRightSelector(leftTexture, rightTexture, stopTexture);
    genderSelector.SetItems(genderItems, 125);
    genderSelector.Position = new Vector2(label1.Position.X, 250);
    genderSelector.SelectionChanged += GenderSelector_SelectionChanged;

    ControlManager.Add(genderSelector);
```

```
            classSelector = new LeftRightSelector(leftTexture, rightTexture, stopTexture);
            classSelector.SetItems(classItems, 125);
            classSelector.Position = new Vector2(label1.Position.X, 300);

            ControlManager.Add(classSelector);

            LinkLabel linkLabel1 = new LinkLabel
            {
                Text = "Accept this character.",
                Position = new Vector2(label1.Position.X, 350)
            };
            linkLabel1.Selected += new EventHandler(LinkLabel1_Selected);

            ControlManager.Add(linkLabel1);
            ControlManager.NextControl();
        }

        private void GenderSelector_SelectionChanged(object sender, EventArgs e)
        {
            if (genderSelector.SelectedIndex == 0)
                nameSelector.SetItems(maleName, 125);
            else if (genderSelector.SelectedIndex == 1)
                nameSelector.SetItems(femaleName, 125);
            else
                nameSelector.SetItems(nbName, 125);
        }

        void LinkLabel1_Selected(object sender, EventArgs e)
        {
            InputHandler.Flush();

            StateManager.PopState();
            StateManager.PushState(GameRef.GamePlayScreen);
        }

        #endregion
    }
}
```

There are using statements to bring some MonoGame Framework classes into scope and our **MGRpgLibrary** project. The class inherits from **BaseGameState** so it can be used in the state manager and to add some inherited fields and methods.

There are three **LeftRightSelector** fields, a **PictureBox**, and five string arrays. The string arrays hold the three genders, Male, Female and Non-Binary (trying to be gender inclusive). Another holds the classes in the game. I'm going with the rather basic classes for the game, Rogue, Fighter, Priest, and Wizard. I'm hoping to design the class system so it will be easy to add in different classes. For now they will do. The other three string arrays hold the male, female and non-binary names. The **LeftRightSelectors** are to select the name, gender and class of the character. The **PictureBox** is for the background image. Eventually the name selector will be replaced with a text box but for now it works.

The constructor at the moment does nothing. In the **LoadContent** method, after the call to to the base class, I call a method **CreateControls** that creates the controls on screen. The **Update** method calls the

Update method of the **ControlManager** passing in the **gameTime** parameter of the **Update** method and **PlayerIndex.One** for the first game pad. If you're coding for the Xbox 360 is important for the player to be able to select what controller they want to use. I will address this in a future tutorial. The **Draw** method calls calls the **Begin** method of the **SpriteBatch** object from the **Game1** class using the **GameRef** field. The **base.Draw** to draw any components on the screen, the **Draw** method of the **ControlManager**, and then **End** on the **SpriteBatch** object.

The **CreateControls** method is where I create the controls on the screen. The first step is to load in the images for the left arrow, right arrow, and stop bar. Order is important when drawing in 2D. If you don't get the order right objects will be drawn on top of others and won't be visible. That is why I create the picture box for the background first and add it to the controls first. Later on down the road I'm going to update the control manager so you can control how controls will be drawn. After creating the picture box and adding it to the control manager I create a label with the text: **Who will search for the Eyes of the Dragon?**. I find out its size using the **MeasureString** method of the sprite font then center it horizontally on the screen. The Y value for its position was completely arbitrary. The label is then added to the control manager. I then create the **LeftRightSelectors** for the character's name, gender and class. I wire the event handler for the **SelectionChanged** event of the gender selector so that I can change the name based on the selected gender.

The images for the left arrow, right arrow, and stop bar are passed to the constructors. For the name selector the **maleName** items are passed in for the items. For the gender selector the **genderItems** array is passed in to the **SetItems** method. Similarly, the **classItems** array is passed to the **SetItems** method of the class selector. For the X coordinate of their position they are lined up with the X coordinate of the label. I separated the controls 50 pixels a part vertically. I then create a **LinkLabel** that the player will select when they are happy with their choices. I set the text to: **Accept this character**. I line it up horizontally with the other controls and space it 50 pixels from the last selector. I also wire an event handler for the **Selected** event. The control is then added to the control manager and I call the **NextControl** method to move the selection to the first control that is a tab stop.

In the **GenderSelector_SelectionChanged** event is where I handle the player choosing a different gender. If the **SelectedIndex** is zero I call the **SetItems** method passing in the **maleName** field. If it is one I call the **SetItems** method passing in the **femaleName** field. If all other cases fail I pass in the **nbName** field to the **SetItems** method.

In the **LinkLabel1_Selected** method is where I handle that the player is happy with their character. I call the **Flush** method of the **InputHandler** to eliminate cascading. I then pop the character generator off the stack of game states and push the game play screen onto the stack of game states.

The next step is to add this screen to the game. The first step is to add a field to the **Game1** class for the character generator screen and create it in the constructor. Add the following property to the **Game State** region of the **Game1** class. Also, change the constructor to the following as well.

```
public CharacterGeneratorScreen CharacterGeneratorScreen { 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);
        CharacterGeneratorScreen = new CharacterGeneratorScreen(this, _gameStateManager);

        _gameStateManager.ChangeState(TitleScreen);
    }
```

The last step to implement the character generator takes place in the **StartMenuScreen**. In the event handler for the menu items being selected you want to push the character generator onto the stack instead of the game play screen. Change the **menuItem_Selected** method to the following.

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

Things are starting to come together but there is still a lot of work to be done. 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