A Summoner's Tale
Chapter D
Where Did I Put That?

Welcome to my Summoner's Tale tutorial series on creating a Pokémon-inspired game with MonoGame. I'm writing these tutorials for the MonoGame 3.8.1 framework using Visual Studio 2022. The code should work on previous versions of MonoGame and Visual Studio. I plan on creating the editors on macOS and Windows. I'm unfamiliar with Linux, so a few projects may not be done for that platform.

The tutorials will make more sense if they are read in order. You can find the list of tutorials on my web blog, A Summoner's Tale page. In addition to the PDFs, I will make the code for each tutorial available on GitHub here: https://github.com/Synammon/summoners-tale. It will also be included on the page that links to the tutorials.

This tutorial will cover updates to forms and adding a file form that can be used to select files. It parallels tutorial 08 in Eyes of the Dragon 4.0. In fact, if you read that tutorial, this one will be a walk in the park. So, let's get started.

To begin, I am going to add two extension methods. The first fills a Texture2D with a specific colour. The second returns a rectangle greater or smaller than the size passed in. So, for example, passing in one will grow the rectangle and passing in minus one will shrink the rectangle. First, update the Extension method class to the following code.

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

namespace SummonersTale
{
    public static class ExtensionMethods
    {
        public static void Fill(this Texture2D texture2D, Color color)
        {
            Color[] data = new Color[texture2D.Width * texture2D.Height];

            for (int i = 0; i < data.Length; i++)
            {
                data[i] = color;
            }

            texture2D.SetData(data);
        }
```

```
        public static Rectangle Grow(this Rectangle r, int size)
        {
            return new(r.X - size,
                       r.Y - size,
                       r.Width + size * 2,
                       r.Height + size * 2);
        }

        public static Rectangle Scale(this Rectangle rect, Vector2 scale)
        {
            return new Rectangle(
                (int)(rect.X * scale.X),
                (int)(rect.Y * scale.Y),
                (int)(rect.Width * scale.X),
                (int)(rect.Height * scale.Y));
        }

    }
}
```

Several times, you've seen the code for filling in a Texture2D, hence the extension method. It is so much cleaner being able to do texture2D. For example, fill (Color.White), than have to keep creating the array of colours, looping over the elements, setting the colour, and setting the data. But wait, I just described the method without intending to.

Grow is a little more interesting. To grow or shrink the X and Y properties, you adjust the X and Y properties by that many pixels. To change the height and width, you must do twice the number of pixels. Why twice? If you have a five-by-five rectangle at (10, 10) and want to grow it 2 pixels, you subtract that from the X and Y properties moving it to (8, 8). If you change the height and width of two pixels, you are (8, 8) with a height and width of (7, 7). To grow so it is two pixels bigger, you need to add another two pixels. Clear? Not really? It works, however. For example, I have a text box texture. I want to draw a border around it. I created a black texture, positioned it up, and left one pixel. That shifted the texture up and left the same. You are growing the width and height by one only extended to the edges of the text box. I had to grow another pixel to get the desired effect.

The text box, list box, picture box, and form classes underwent some major adjustments. This was to fix some rendering issues that were coming through. Mainly, it had to do with rendering on partial pixels. It is just easier to give you the code for the classes than try to feed it to you piece by piece. I will tackle the changes to the controls in the order listed.

Text Boxes

Honestly, I wouldn't say I liked how text box textures scaled. The borders were disproportionate, and the shadow irked me. That was part of the reason why I changed text boxes. Replace the Textbox class with this new version.

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

namespace SummonersTale.Forms
{
    internal class Textbox : Control
    {
        private readonly Texture2D _border;
        private readonly Texture2D _background;
        private readonly Texture2D _caret;

        private double timer;
        private Color _tint;
        private readonly List<string> validChars = new();

        public Textbox(GraphicsDevice graphicsDevice, Vector2 size)
            : base()
        {

            _text = "";

            _background = new Texture2D(graphicsDevice, (int)size.X, (int)size.Y);
            _background.Fill(Color.White);

            _border = new Texture2D(graphicsDevice, (int)size.X, (int)size.Y);
            _border.Fill(Color.Black);

            _caret = new Texture2D(graphicsDevice, 2, (int)size.Y);
            _caret.Fill(Color.Black);

            _tint = Color.Black;
            foreach (char c in
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTWYYXZ0123456789 -_".ToCharArray())
            {
                validChars.Add(c.ToString());
            }
        }

        public override void Draw(SpriteBatch spriteBatch)
        {
            Vector2 dimensions = ControlManager.SpriteFont.MeasureString(_text);
            dimensions.Y = 0;

            Rectangle location = new(
                new((int)Position.X, (int)Position.Y),
                new((int)Size.X, (int)Size.Y));

            spriteBatch.Draw(_border, location.Grow(1), Color.White);
            spriteBatch.Draw(_background, location, Color.White);

            spriteBatch.DrawString(
                ControlManager.SpriteFont,
                Text,
                Helper.NearestInt(Position + Vector2.One * 5),
```

```
                Color.Black,
                0,
                Vector2.Zero,
                1f,
                SpriteEffects.None,
                1f);
            spriteBatch.Draw(
                _caret,
                Position + dimensions + Vector2.One * 5,
                _tint);
        }

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

            List<Keys> keys = Xin.KeysPressed();

            foreach (Keys key in keys)
            {
                string value = Enum.GetName(typeof(Keys), key);

                if (value == "Back" && _text.Length > 0)
                {
                    _text = _text.Substring(0, _text.Length - 1);
                    return;
                }
                else if (value == "Back")
                {
                    Text = "";
                    return;
                }

                if (value.Length == 2 && value.Substring(0,1) == "D")
                {
                    value = value.Substring(1);
                }

                if (!Xin.IsKeyDown(Keys.LeftShift) &&
!Xin.IsKeyDown(Keys.RightShift) && !Xin.KeyboardState.CapsLock)
                {
                    value = value.ToLower();
                }

                if (validChars.Contains(value))
                {
                    if (ControlManager.SpriteFont.MeasureString(_text + value).X <
Size.X)
                        _text += value;
                }
            }
        }

        public override void Update(GameTime gameTime)
        {
```

```
            timer += 3 * gameTime.ElapsedGameTime.TotalSeconds;
            double sine = Math.Sin(timer);
            _tint = Color.Black * (int)Math.Round(Math.Abs(sine));
        }
    }
}
```

Quite a radical change. That much is not in question. What has changed, though? First is the addition of the field for the border. Second, I changed the constructor to take a GraphicsDevice instead of Texture2Ds to be assigned to the fields. It also requires a Vector2 for the size of the text box. It then creates the textures and fills them with the new Fill extension method. Due to the nature of these changes, the text box can be scaled at run time with no ill effects from the scaling.

Everything else remains the same until you get to the Draw method. So what I do is create a rectangle using the Position property for the X and Y coordinates of the rectangle and the Size property for the height and width. Next, I draw the border, increasing its dimensions by one. I then draw the background. After that, the rest of the method flows as before.

List Box

Arguably, the most complex control, at least in what has been implemented so far. Therefore, it is also the longest of the controls. So, I apologize, in advance, for the wall of code that you are about to be presented with. Replace the ListBox class with the following.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using SummonersTale.Forms;
using SummonersTale;
using System;
using System.Collections.Generic;
using SummonersTale.SpriteClasses;

namespace SummonersTale.Forms
{
    public class SelectedIndexEventArgs : EventArgs
    {
        public int Index;
    }

    public class ListBox : Control
    {
        #region Event Region

        public event EventHandler<SelectedIndexEventArgs> SelectionChanged;
        public new event EventHandler<SelectedIndexEventArgs> Selected;
        public event EventHandler Enter;
        public event EventHandler Leave;

        #endregion
```

```
        #region Field Region

        private readonly List<string> _items = new();

        private int _startItem;
        private int _lineCount;

        private readonly Texture2D _background;
        private readonly Texture2D _border;
        private readonly Texture2D _cursor;

        private Color _selectedColor = Color.White;
        private int _selectedItem;
        private readonly Button _upButton, _downButton;
        private double timer;

        private bool _mouseOver;

        #endregion

        #region Property Region

        public bool MouseOver => _mouseOver;

        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; }
        }

        public new bool HasFocus
        {
            get { return _hasFocus; }
            set
            {
                _hasFocus = value;

                if (_hasFocus)
                    OnEnter(null);
                else
                    OnLeave(null);
            }
```

```
        }

        public Rectangle Bounds
        {
            get { return new(Helper.V2P(Position), Helper.V2P(Size)); }
        }
        #endregion

        #region Constructor Region

        public ListBox(GraphicsDevice graphicsDevice, Texture2D downButton,
Texture2D upButton, Vector2 size)
            : base()
        {
            HasFocus = false;
            TabStop = true;
            Size = size;

            _upButton = new(upButton, ButtonRole.Menu) { Text = "" };
            _downButton = new(downButton, ButtonRole.Menu) { Text = "" };

            _upButton.Click += UpButton_Click;
            _downButton.Click += DownButton_Click;

            _background = new(graphicsDevice, (int)Size.X, (int)Size.Y);
            _background.Fill(Color.White);

            _border = new(graphicsDevice, (int)Size.X, (int)Size.Y);
            _border.Fill(Color.Black);

            _cursor = new(graphicsDevice, (int)Size.X - 40,
(int)ControlManager.SpriteFont.LineSpacing);
            _cursor.Fill(Color.Blue);

            _startItem = 0;
            Color = Color.Black;
        }

        private void DownButton_Click(object sender, EventArgs e)
        {
            if (_selectedItem != _items.Count - 1 && timer > 0.5)
            {
                timer = 0;
                _selectedItem++;
                OnSelectionChanged();
            }
        }

        private void UpButton_Click(object sender, EventArgs e)
        {
            if (_selectedItem > 0 && timer > 0.5)
            {
                timer = 0;
                _selectedItem--;
                OnSelectionChanged();
            }
        }
```

```
        #endregion

        #region Abstract Method Region

        public override void Update(GameTime gameTime)
        {
            timer += gameTime.ElapsedGameTime.TotalSeconds;

            _upButton.Update(gameTime);
            _downButton.Update(gameTime);
            HandleInput();
        }

        public override void Draw(SpriteBatch spriteBatch)
        {
            _lineCount = (int)(Size.Y / SpriteFont.LineSpacing);

            Rectangle d = new((int)Position.X, (int)Position.Y, (int)Size.X,
(int)Size.Y);

            spriteBatch.Draw(_border, d.Grow(1), Color.White);

            spriteBatch.Draw(_background, d, Color.White);

            Point position = Xin.MouseAsPoint;
            Rectangle destination = new(0, 0, Bounds.Width - 40,
(int)SpriteFont.LineSpacing);
            _mouseOver = false;

            for (int i = 0; i < _lineCount; i++)
            {
                if (_startItem + i >= _items.Count)
                {
                    break;
                }

                destination.X = (int)Position.X;
                destination.Y = (int)(Position.Y + i * SpriteFont.LineSpacing);

                if ((destination.Contains(position) &&
                    Xin.MouseState.LeftButton == ButtonState.Pressed) ||
                    (destination.Contains(Xin.TouchLocation) &&
                    Xin.TouchPressed()))
                {
                    _mouseOver = true;
                    _selectedItem = _startItem + i;
                    OnSelectionChanged();
                }

                float length = 0;
                int j = 0;
                string text = "";

                while (length <= Size.X - _upButton.Width && j < _items[i].Length)
                {
                    j++;
                    length = SpriteFont.MeasureString(_items[i].Substring(0, j)).X;
                    text = _items[i].Substring(0, j);
```

```
                }

                if (_startItem + i == _selectedItem)
                {
                    Vector2 location = new(Position.X + 5, Position.Y + i *
SpriteFont.LineSpacing + 2);
                    spriteBatch.Draw(
                        _cursor,
                        Helper.NearestInt(location),
                        Color.White);
                    location.X += 5;
                    spriteBatch.DrawString(
                        SpriteFont,
                        text,
                        Helper.NearestInt(location),
                        SelectedColor);
                }
                else
                {
                    spriteBatch.DrawString(
                        SpriteFont,
                        text,
                        Helper.NearestInt(new Vector2(Position.X + 8, Position.Y + i
* SpriteFont.LineSpacing + 2)),
                        Color);
                }
            }

            _upButton.Position = new((int)(Position.X + Size.X - _upButton.Width),
(int)Position.Y);
            _downButton.Position = new((int)(Position.X + Size.X -
_downButton.Width), (int)(Position.Y + Size.Y - _downButton.Height));

            _upButton.Draw(spriteBatch);
            _downButton.Draw(spriteBatch);
        }

        public override void HandleInput()
        {
            if (Xin.WasKeyReleased(Keys.Down) && HasFocus && timer > 0.5)
            {
                timer = 0;
                if (_selectedItem < _items.Count - 1)
                {
                    _selectedItem++;

                    if (_selectedItem >= _startItem + _lineCount)
                    {
                        _startItem = _selectedItem - _lineCount + 1;
                    }

                    OnSelectionChanged();
                }
            }
            else if (Xin.WasKeyReleased(Keys.Up) && HasFocus && timer > 0.5)
            {
                timer = 0;
                if (_selectedItem > 0)
```

```
            {
                _selectedItem--;

                if (_selectedItem < _startItem)
                {
                    _startItem = _selectedItem;
                }

                OnSelectionChanged();
            }
        }

        if (Xin.WasMouseReleased(MouseButton.Left) && _mouseOver && timer > 0.5)
        {
            timer = 0;
            HasFocus = true;
            OnSelected();
        }

        if (Xin.IsMouseDown(MouseButton.Right))
        {
            HasFocus = false;
            OnSelected(null);
        }

        if (Xin.WasKeyReleased(Keys.Enter) && timer > 0.5)
        {
            timer = 0;
            HasFocus = true;
            OnSelected();
        }

        if (Xin.WasKeyReleased(Keys.Escape))
        {
            HasFocus = false;
            OnLeave(null);
        }
    }

    #endregion

    #region Method Region

    public virtual void OnSelected()
    {
        var e = new SelectedIndexEventArgs()
        {
            Index = _selectedItem,
        };

        Selected?.Invoke(this, e);
    }
    protected virtual void OnSelectionChanged()
    {
        var e = new SelectedIndexEventArgs()
        {
            Index = _selectedItem,
        };
```

```
            SelectionChanged?.Invoke(this, e);
        }

        protected virtual void OnEnter(EventArgs e)
        {
            Enter?.Invoke(this, e);
        }

        protected virtual void OnLeave(EventArgs e)
        {
            Leave?.Invoke(this, e);
        }

        #endregion
    }
}
```

Yikes! That just doubled the length of this tutorial. There are a few differences between this one and the one from Eyes of the Dragon 4.0. The first thing is adding a class that inherits for EventArgs, SelectedIndexEventArgs. As the name implies, it is for when the Selected Index of the list box changes. It has a single property Index, which is, of course, what item in the list box has been selected.

I made some changes to the events. First is an update to the SelectionChanged event. It now has a SelectedIndexEventArgs argument associated with it. This is so that whoever subscribed to our event will know what the index was changed to. Next, I created a new version of the Selected event. In essence, these are virtually the same event with different names.

Three fields have changed. First is the _background field, which is the background of the list box. Next, like in the text box, there is an image for the border. Finally, there is the cursor, which is drawn before the selected text. I updated the colour for the selected item to white. I also added a property, MouseOver. It tells if the mouse is over the list box or not.

Like the text box, the control requires a graphics device and a Vector2. It still requires a Texture2D for the up and down buttons. In the constructor, I create the textures and fill them.

I made essentially the same change to the DownButton_Click and UpButton_Click event handlers. The only difference is the direction in which I change the selection. Down is, of course, moving the selection down and up for up. The change is that I now check to see if the timer field is greater than half a second. That is probably too long. Then, if the selection has changed, I call the OnSelectionChanged method to notify any subscribers of that event that the selection has changed.

I call the HandleInput method to capture any input events in the Update method. Then, in the Draw method, I first draw the border for the list box, expanding the destination of the list box by one pixel. Then, instead of drawing the cursor with a width of one hundred pixels, I create a

rectangle that is the width of the list box minus forty pixels for the buttons at the right edge of the list box and some padding on the left. There were some changes to the code that determines if an item is selected or not. First, there is the check to see if the start item plus the current item is the selected item. If it is, I create a Vector2 that is that location. I then draw the list box cursor at that location. I then indent five pixels to the right and draw the item in the list box. If it is not the selected item, I indent five pixels to the left and draw the text. I also updated the position of the scroll-up and scroll-down buttons.

The HandleInput method has changed a lot. I no longer exit the method if the control does not have focus. That is because of the mouse. The mouse can be over the control and trigger events when it does not have focus. So, for the code where the up or down key has been pressed, I add an additional clause checking if the control has focus. If it does, I update the cursor and call the OnSelectionChanged method.

If the mouse is over the control, the timer is greater than half a second, and the mouse button has been released, or the control has focus, the enter key has been released, and the timer is greater than half a second, I reset the timer, call the OnSelected method passing in the selected index. I also set the HasFocus property to true.

So, the OnSelected and OnSelectionChanged methods are virtual and identical. I create a new SelectedIndexEventArgs with the Index property being the selected Index. I then call the event handler. They are virtual because they can be overridden in any child.

Picture Box

Thankfully, this is a rather simple class. Unfortunately, it is still on the long side. For that, I apologize again. Thankfully, it requires little explanation. What has changed in the picture box is that I added a new field for the border. Before rendering the PictureBox, I draw the border. Also, rather than accepting a texture for the border, it accepts a GraphicsDevice so it can create the border itself. Because it is creating the border texture, it calls the new Fill extension method. Replace the PictueBox class with the following version.

```
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework;
using System;
using System.Collections.Generic;
using System.Text;
using System.Diagnostics.Contracts;

namespace SummonersTale.Forms
{
    public enum FillMethod { Clip, Fill, Original, Center }

    public class PictureBox : Control
    {
        #region Field Region
```

```
        private Texture2D _image;
        private readonly Texture2D _border;
        private Rectangle _sourceRect;
        private Rectangle _destRect;
        private FillMethod _fillMethod;
        private int _width;
        private int _height;

        #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; }
        }

        public FillMethod FillMethod
        {
            get { return _fillMethod; }
            set { _fillMethod = value; }
        }

        public int Width
        {
            get { return _width; }
            set { _width = value; }
        }

        public int Height
        {
            get { return _height; }
            set { _height = value; }
        }

        #endregion

        #region Constructors

        public PictureBox(GraphicsDevice GraphicsDevice, Texture2D image, Rectangle
destination)
        {
            Image = image;
```

```
            _border = new Texture2D(GraphicsDevice, image.Width, image.Height);
            _border.Fill(Color.Black);

            DestinationRectangle = destination;

            Width = destination.Width;
            Height = destination.Height;

            if (image != null)
                SourceRectangle = new Rectangle(0, 0, image.Width, image.Height);
            else
                SourceRectangle = new Rectangle(0, 0, 0, 0);

            Color = Color.White;

            _fillMethod = FillMethod.Original;

            if (SourceRectangle.Width > DestinationRectangle.Width)
            {
                _sourceRect.Width = DestinationRectangle.Width;
            }

            if (SourceRectangle.Height > DestinationRectangle.Height)
            {
                _sourceRect.Height = DestinationRectangle.Height;
            }
        }

        public PictureBox(GraphicsDevice GraphicsDevice, Texture2D image, Rectangle
destination, Rectangle source)
            : this(GraphicsDevice, image, destination)
        {
            SourceRectangle = source;
            Color = Color.White;

            _fillMethod = FillMethod.Original;

            if (SourceRectangle.Width > DestinationRectangle.Width)
            {
                _sourceRect.Width = DestinationRectangle.Width;
            }

            if (SourceRectangle.Height > DestinationRectangle.Height)
            {
                _sourceRect.Height = DestinationRectangle.Height;
            }
        }

        #endregion

        #region Abstract Method Region

        public override void Update(GameTime gameTime)
        {
        }

        public override void Draw(SpriteBatch spriteBatch)
        {
```

```
            Rectangle borderDest = DestinationRectangle.Grow(1);

            spriteBatch.Draw(_border, borderDest, Color.White);

            if (_image != null)
            {
                switch (_fillMethod)
                {
                    case FillMethod.Original:
                        _fillMethod = FillMethod.Original;

                        if (SourceRectangle.Width > DestinationRectangle.Width)
                        {
                            _sourceRect.Width = DestinationRectangle.Width;
                        }

                        if (SourceRectangle.Height > DestinationRectangle.Height)
                        {
                            _sourceRect.Height = DestinationRectangle.Height;
                        }

                        spriteBatch.Draw(Image, DestinationRectangle,
SourceRectangle, Color);
                        break;
                    case FillMethod.Clip:
                        if (DestinationRectangle.Width > SourceRectangle.Width)
                        {
                            _destRect.Width = SourceRectangle.Width;
                        }

                        if (_destRect.Height > DestinationRectangle.Height)
                        {
                            _destRect.Height = DestinationRectangle.Height;
                        }

                        spriteBatch.Draw(Image, _destRect, SourceRectangle, Color);
                        break;
                    case FillMethod.Fill:
                        _sourceRect = new(0, 0, Image.Width, Image.Height);
                        spriteBatch.Draw(Image, DestinationRectangle, null, Color);
                        break;
                    case FillMethod.Center:
                        _sourceRect.Width = Image.Width;
                        _sourceRect.Height = Image.Height;
                        _sourceRect.X = 0;
                        _sourceRect.Y = 0;

                        Rectangle dest = new(0, 0, Width, Height);

                        if (Image.Width >= Width)
                        {
                            dest.X = DestinationRectangle.X;
                        }
                        else
                        {
                            dest.X = DestinationRectangle.X + (Width - Image.Width)
/ 2;
                        }
```

```
                        if (Image.Height >= Height)
                        {
                            dest.Y = DestinationRectangle.Y;
                        }
                        else
                        {
                            dest.Y = DestinationRectangle.Y + (Height -
Image.Height) / 2;
                        }
                        spriteBatch.Draw(Image, dest, SourceRectangle, Color);
                        break;
                }
            }
        }

        public override void HandleInput()
        {
        }

        #endregion

        #region Picture Box Methods

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

        #endregion
    }
}
```

Form

To almost the end of the modified code. There is the form, of course. There is also the new game state because it has a text box on it. In the form, there is the LoadContent method and the Draw method. In the LoadContent method, because I'm not passing the same parameters to picture boxes. In the Draw method because I no longer wish to use a transformation matrix when rendering. Change the Draw and LoadContent methods of the Form class to the following.

```
protected override void LoadContent()
{
    base.LoadContent();
    _controls = new(content.Load<SpriteFont>("Fonts/MainFont"), 100);

    TitleBar = new(
        GraphicsDevice,
        content.Load<Texture2D>("GUI/TitleBar"),
        new(
            0,
```

```
            0,
            Size.X,
            20));

    Background = new(
        GraphicsDevice,
        content.Load<Texture2D>("GUI/Form"),
        new(
            0,
            20,
            Bounds.Width,
            Bounds.Height))
    { FillMethod = FillMethod.Fill };

    CloseButton = new(
        content.Load<Texture2D>("GUI/CloseButton"),
        ButtonRole.Cancel)
    { Position = Vector2.Zero, Color = Color.White, Text = "" };

    CloseButton.Click += CloseButton_Click;

    if (FullScreen)
    {
        TitleBar.Height = 0;
        Background.Position = new();
        Background.Height = _graphicsDevice.PreferredBackBufferHeight;
    }
}

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

    if (!Visible) return;

    if (!FullScreen)
    {
        Vector2 size = ControlManager.SpriteFont.MeasureString(Title);
        Vector2 position = new((Bounds.Width - size.X) / 2, 0);
        Label label = new()
        {
            Text = _title,
            Color = Color.White,
            Position = position
        };

        Matrix m = Matrix.Identity;

        SpriteBatch.Begin(SpriteSortMode.BackToFront, BlendState.AlphaBlend,
SamplerState.AnisotropicWrap, null, null, null, m);

        Background.Draw(SpriteBatch);
        TitleBar.Draw(SpriteBatch);

        CloseButton.Draw(SpriteBatch);

        SpriteBatch.End();
```

```
        SpriteBatch.Begin();

        label.Position = Helper.NearestInt(position + Position);
        label.Color = Color.White;
        label.Draw(SpriteBatch);

        SpriteBatch.End();

        //m = Matrix.CreateTranslation(new Vector3(0, 20, 0) + new Vector3(Position,
0));

        SpriteBatch.Begin(SpriteSortMode.FrontToBack, BlendState.AlphaBlend,
SamplerState.AnisotropicWrap, null, null, null, m);

        _controls.Draw(SpriteBatch);

        SpriteBatch.End();
    }
    else
    {
        SpriteBatch.Begin();

        Background.DestinationRectangle = new(
            0,
            0,
            _graphicsDevice.PreferredBackBufferWidth,
            _graphicsDevice.PreferredBackBufferHeight);

        _controls.Draw(SpriteBatch);

        SpriteBatch.End();
    }
}
```

Nothing really complicated going on here. I pass the graphics device as a parameter when creating a picture box. Instead of using a transformation matrix, I use the identity matrix. The identity matrix is a special matrix. First, it is square. Second, when multiplying a matrix by the identity matrix, you get the original matrix. That is why I could substitute it. I substituted it because I may go back to using a transformation matrix at some point.

Yikes! This tutorial is longer than I wanted. I want to address one quick thing before I move on to the next form. That is the NewGameState. I updated the LoadContent method to create the text box using the new constructor. Update that method to the following.

```
protected override void LoadContent()
{
    renderTarget2D = new(GraphicsDevice, TargetWidth, TargetHeight);

    Controls = new(content.Load<SpriteFont>(@"Fonts/MainFont"), 100);
    _genderSelector = new(
        content.Load<Texture2D>(@"GUI\g22987"),
        content.Load<Texture2D>(@"GUI\g21245"))
```

```
{
    Position = new Vector2(207 - 70, 298)
};
_genderSelector.SelectionChanged += GenderSelector_SelectionChanged;
_genderSelector.SetItems(new[] { "Female", "Male" }, 270);

_portraitSelector = new(
    content.Load<Texture2D>(@"GUI\g22987"),
    content.Load<Texture2D>(@"GUI\g21245"))
{
    Position = new Vector2(207 - 70, 458)
};

_portraitSelector.SelectionChanged += PortraitSelector_SelectionChanged;

Color[] colourData = new Color[2 * 25];

for (int i = 0; i < colourData.Length; i++)
{
    colourData[i] = Color.White;
}

_nameTextBox = new(GraphicsDevice, new(100, 25))
{
    Position = new(207, 138),
    HasFocus = true,
    Enabled = true,
    Color = Color.White,
    Text = "Bethany"
};

_femalePortraits.Add(
    "Female 0",
    content.Load<Texture2D>(@"CharacterSprites/femalefighter"));
_femalePortraits.Add(
    "Female 1",
    content.Load<Texture2D>(@"CharacterSprites/femalepriest"));
_femalePortraits.Add(
    "Female 2",
    content.Load<Texture2D>(@"CharacterSprites/femalerogue"));
_femalePortraits.Add(
    "Female 3",
    content.Load<Texture2D>(@"CharacterSprites/femalewizard"));

_malePortraits.Add(
    "Male 0",
    content.Load<Texture2D>(@"CharacterSprites/malefighter"));
_malePortraits.Add(
    "Male 1",
    content.Load<Texture2D>(@"CharacterSprites/malepriest"));
_malePortraits.Add(
    "Male 2",
    content.Load<Texture2D>(@"CharacterSprites/malerogue"));
_malePortraits.Add(
    "Male 3",
    content.Load<Texture2D>(@"CharacterSprites/malewizard"));
```

```
    _portraitSelector.SetItems(_femalePortraits.Keys.ToArray(), 270);
    _sprite = new(_femalePortraits["Female 0"], animations)
    {
        CurrentAnimation = "walkdown",
        IsAnimating = true
    };

    _create = new(
        content.Load<Texture2D>(@"GUI\g9202"),
        ButtonRole.Accept)
    {
        Text = "Create",
        Position = new(180, 640)
    };

    _create.Click += Create_Click;

    _back = new(
        content.Load<Texture2D>(@"GUI\g9202"),
        ButtonRole.Cancel)
    {
        Text = "Back",
        Position = new(350, 640),
        Color = Color.White
    };

    _back.Click += Back_Click;

    Controls.Add(_nameTextBox);
    Controls.Add(_genderSelector);
    Controls.Add(_portraitSelector);
    Controls.Add(_create);
    Controls.Add(_back);
}
```

File Form

That brings us to the last piece for this tutorial. I was going to cover triggering the form, but I am going to hold off on that one. Right-click the Controls folder in the SummonersTale project, select Add and then Class. Name this new class FileForm. The code for that class follows next.

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

namespace SummonersTale.Forms
{
    public enum FileFormRole { Open, Save, Create, Directory }

    public class FileForm : Form
    {
        Textbox selected;
```

```
        Button action;
        ListBox items;
        string dir = Environment.CurrentDirectory;

        public string FileName { get; set; }
        public FileFormRole Role { get; set; } = FileFormRole.Open;

        public FileForm(Game game, Vector2 position, Point size) : base(game,
position, size)
        {
            Position = new(0, 0);

            Size = new(_graphicsDevice.PreferredBackBufferWidth,
_graphicsDevice.PreferredBackBufferHeight);
            Bounds = new(0, 0, _graphicsDevice.PreferredBackBufferWidth,
_graphicsDevice.PreferredBackBufferHeight);
            FullScreen = true;
        }

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

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

            Size = new(_graphicsDevice.PreferredBackBufferWidth,
_graphicsDevice.PreferredBackBufferHeight);
            Bounds = new(0, 0, _graphicsDevice.PreferredBackBufferWidth,
_graphicsDevice.PreferredBackBufferHeight);

            Background.Image = new(GraphicsDevice, Size.X, Size.Y);
            Background.Image.Fill(Color.White);

            Texture2D caret = new(GraphicsDevice, 2, 16);
            caret.Fill(Color.Black);

            Texture2D bg = new(GraphicsDevice, 4, 4);
            bg.Fill(Color.White);

            Texture2D brdr = new(GraphicsDevice, 4, 4);

            brdr.Fill(Color.Black);

            selected = new(GraphicsDevice, new(Size.X - 80, 25))
            {
                Position = new(5, 25),
                Color = Color.White,
                Text = "",
                Visible = true,
                Enabled = true,
                TabStop = true,
                HasFocus = true,
            };

            Controls.Add(selected);
```

```
            action = new(Game.Content.Load<Texture2D>(@"GUI/Button"),
ButtonRole.Accept)
            {
                Position = new(Size.X - 55, 25),
                Color = Color.Black,
                Text = "Select",
                Visible = true,
                Enabled = true,
                TabStop = true,
                Size = new(50, 25),
                Role = ButtonRole.Menu,
            };

            action.Click += Action_Click;

            Controls.Add(action);

            caret = new(GraphicsDevice, Size.X - 40, 16);
            caret.Fill(Color.Blue);

            items = new(GraphicsDevice,
                Game.Content.Load<Texture2D>(@"GUI/DownButton"),
                Game.Content.Load<Texture2D>(@"GUI/UpButton"),
                new(Size.X, Size.Y - 60))
            {
                Position = new(0, 60),
                Size = new(Size.X, Size.Y - 60),
                Color = Color.Black,
                TabStop = true,
                Enabled = true,
                Visible = true,
            };

            items.Selected += Items_Selected;
            Controls.Add(items);

            FillItems();
        }

        private void Action_Click(object sender, EventArgs e)
        {
            string path = string.Format("{0}{1}", dir, selected.Text);

            if (Role == FileFormRole.Open)
            {
                if (!File.Exists(path))
                {
                    // uh-oh!
                }
                else
                {
                    FileName = path;
                    manager.PopState();
                }
            }
            else if (Role == FileFormRole.Create)
            {
```

```
            if (File.Exists(path))
            {
                // uh-oh!
            }
            else
            {
                FileName = path;
                manager.PopState();
            }
        }
        else if (Role == FileFormRole.Save)
        {
            if (File.Exists(path))
            {
                // uh-oh. Overwrite?
            }
            else
            {
                FileName = path;
                manager.PopState();
            }
        }
    }

    private void Selected_Selected(object sender, EventArgs e)
    {
    }

    private void FillItems()
    {
        if (!dir.EndsWith("\\")) dir += "\\";

        string[] folders = Directory.GetDirectories(dir);
        string[] files = Directory.GetFiles(dir);

        items.Items.Clear();
        items.Items.Add("..");
        items.SelectedIndex = 0;

        foreach (var v in folders)
        {
            items.Items.Add(new DirectoryInfo(v).Name);
        }

        foreach (var v in files)
        {
            string path = Path.GetFileName(v);
            items.Items.Add(path);
        }
    }

    private void Items_Selected(object sender, SelectedIndexEventArgs e)
    {
        if (e.Index > 0)
        {
            if (File.Exists(dir + items.Items[e.Index]))
            {
                FileName = items.Items[e.Index];
```

```
                    selected.Text = FileName;
                }
                else
                {
                    dir += items.Items[e.Index];
                    FillItems();
                }
            }
            else if (e.Index == 0)
            {
                dir =
Directory.GetParent(Directory.GetParent(dir).FullName).FullName;
                FillItems();
            }
        }

        public override void Update(GameTime gameTime)
        {
            foreach (Control c in Controls)
            {
                c.Update(gameTime);
            }

            base.Update(gameTime);
        }

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

        public override void Show()
        {
            if (items != null)
            {
                FillItems();
                Title = Role.ToString();
            }

            base.Show();
        }

        public override void Hide()
        {
            base.Hide();
        }
    }
}
```

First, there is an enumeration that is the type of form: Open, Save, Create, and Directory. Open forms will open files, making sure they exist. Save files will save files that exist. Create is for creating new files that do not exist. Directory has not yet been implemented at this time and will be used to select folders.

Since it is a form, there are a few controls listed. The first is a text box that allows for capturing

the selected file name. The second is a button that will, eventually, trigger that a selection has occurred. The last is a list box that will hold the contents of the current folder. Currently, there are no filtering capabilities on the file form. They may be implemented in the future. The dir field holds the current folder being browsed. The FileName property is, as you would imagine, the name of the selected file. Finally, there is a FileFormRole property that will determine what is being navigated.

The constructor takes no additional parameters than the Form class. It sets the position to (0, 0) and size to the size of the window. The bounds are also set to the size of the window. Finally, the constructor sets the FullScreen property to true.

In the LoadContent method, I create the controls and position them. Size and Bounds are re-initialized because there can be a race condition where LoadContent is called before the constructor. I create the background image and set it to white. Next, I create the text box and have it fill most of the window, leaving some room for a button. I then set its properties.

Creating the action button is trivial compared to the other controls. It is created with the Button texture and the Accept button role. I then position it in the upper right corner of the form. The Enabled, Visible and TabStop properties are set to true. I then set the size. After initializing the control, I wire the event handler for the click event and add it to the controls.

I create a new texture for the caret of the list box. I then fill it with blue. I then create the list box passing in the required parameters. I position it just below the text box and button and have it fill the remainder of the screen. The text colour is set to black. I also set the TabStop, Enabled and Visible properties to true. Next, I wire the Selected event handler, which is fired if an item is selected. It is added to the list of controls. The last thing I do is call the FillItems method that, which the name implies, fills the items in the list box.

The Action_Click method is where I handle the click event of the action button. It creates a string that represents the selected path. Even though it is just two strings, it is better to use string,Format instead of string concatenation. It is just more efficient to use that method. Next, I check to see what role the form is. If it is Open, we are checking to see that the selected path does not exist. We should do something if that is the case. Currently, it is commented out. Otherwise, the FileName property is set to the file name, and the form is popped off the stack of states. Similarly, if the role of the form is Create, I check to see if the path does exist. If it does, there is a problem. If it does not, we are good and the file name is set, and the form popped off the stack. While identical to the Create option, the Save option has a role of its own.

There is a method stub, and I am not sure what my thoughts were in regard to this one. I left it in because I'm sure I had a good reason for adding it. Anyway, which weighs 2.4 kilograms, the next method is the FillItems method. The first thing it does is check that the delimiter is the

backslash. I know, I just said you should do this, but I concatenate a backslash to the end of the string. I then use the GetDirectories and GetFiles methods of the Directory class to get the folders and files. I clear the items in the list box. I add .. which will be used to go up a level. I also set the selected index to that item. I loop over the items in the folders variable. I use the Name property of the DirectoryInfo class to get just the name. For files, I do much the same, only using the GetFileName of the path class.

The Items_Selected method takes a SelectedIndexEventArgs, unlike the other event handles we have used this far. It has an Index property that is the index of the item in the list box that was selected. If that index is greater than zero, a file or folder is selected. I check if a file is selected using the Exists method of the File class. If a file is selected, I set the FileName property to that item in the list of items and the Text property of the selected file. Otherwise I add the selected item to the dir field and call the FillItems method. If it was zero, I go up a level using the GetParent method of the Directory class.

The Update method cycles over the controls and calls their Update methods. The Draw and Hide methods are stubs for now. In the Show method I check to see if the items list box is not null. If it is not, I call FillItems to fill at and set the Title property to the role.

Okay, I was going to end the tutorial. However, it is bugging me that you made it this far only not to see the main point of the tutorial in use. So, I'm going to implement the form in the map editor quickly. First, add a field for the form to MainForm.

```
        private FileForm _fileForm;
```

Now, create an instance in the LoadContent method of MainForm.

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

    GraphicsDeviceManager gdm = Game.Services.GetService<GraphicsDeviceManager>();

    _menuButton = new(content.Load<Texture2D>(@"GUI/g21688"), ButtonRole.Menu)
    {
        Text = "",
        Position = new(gdm.PreferredBackBufferWidth - 84, 20)
    };

    _menuButton.Click += MenuButton_Click;

    _renderTarget2D = new(GraphicsDevice, 1280, 1080);
```

```
    _mapDisplay = new(null, 40, 32)
    {
        HasFocus = false,
        Position = new(0, 0)
    };

    TileSheet sheet = new(content.Load<Texture2D>(@"Tiles/Overworld"), "test",
new(40, 36, 16, 16));
    TileSet set = new(sheet);

    TileLayer ground = new(100, 100, 0, 0);
    TileLayer edge = new(100, 100, -1, -1);
    TileLayer building = new(100, 100, -1, -1);
    TileLayer decore = new(100, 100, -1, -1);

    TileMap tileMap = new(set, ground, edge, building, decore, "test");

    _mapDisplay.SetMap(tileMap);

    Color[] data = new Color[1];
    data[0] = Color.Transparent;

    Texture2D b = new(GraphicsDevice, 1, 1);
    b.SetData(data);

    _fileForm = new(Game, Vector2.Zero, Size);
}
```

That leaves us with invoking the instance. Since I will eventually be implementing save and load, in that order, I'm going to bind it to the F1 and F2 keys. Change the Update method of the MainForm to the following.

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

    _mapDisplay.Update(gameTime);
    _menuButton.Update(gameTime);

    if (Xin.WasKeyReleased(Microsoft.Xna.Framework.Input.Keys.F1))
    {
        _fileForm.Role = FileFormRole.Save;
        manager.PushState(_fileForm);
    }

    if (Xin.WasKeyReleased(Microsoft.Xna.Framework.Input.Keys.F2))
    {
        _fileForm.Role = FileFormRole.Open;
        manager.PushState(_fileForm);
    }
}
```

All I do is check if either the F1 or F2 keys have been pressed. If they have, I push the form on top of the stack. If you build and run things mostly work. There are a few rendering issues

where textures are not being rendered. I'm not going to back to fix them at this point. I will fix them in the next tutorial.

So, I'm most definitely not going to dive any further into this tutorial. I think I've fed you more than enough for one day. I covered a lot, and I don't want to overload you. I encourage you to keep visiting my blog for the latest news on my tutorials.

Good luck with your game programming adventures.
Cynthia