Eyes of the Dragon Tutorials 4.0

Part 8

Where Is That?

I'm writing these tutorials for the MonoGame 3.8.1 framework using Visual Studio 2022. 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 4.0 page of my web blog. I will be making the project available on GitHub here. It will be included on the page that links to the tutorials.

This is part eight of a series of tutorials I plan to write on creating a role-playing game with MonoGame. I've worked on similar tutorials using XNA and the past version of MonoGame. In the process of writing more tutorials, I discovered better ways of doing some things and had to go back to fix things. I'm hoping in this series to avoid making those same mistakes. Also, I am going to make the game cross-platform. In my previous tutorials, they were focused on Windows only. I want to open things up for macOS, Linux, Android, iOS, and any platform MonoGame supports.

This tutorial is meant more for Windows, macOS, and Linux. It is about the editor. There is a big piece that needs to be included. We need to be able to select files for loading and saving. So, I will be creating a file dialog. It will be used for navigating the file system and selecting files. What should have been a simple matter turned into more than a bit of a nightmare. Mainly, it revolved around rendering on partial pixels. I did end up solving it, or would I be writing this?

To start this tutorial, I want to include a few extension methods that I found useful. The first fills a Texture2D with a colour. The second expands or shrinks a Rectangle. Replace the ExtensionMethods class in the SharedProject with this new version.

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

namespace SharedProject
{
    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));
        }

    }
}
```

The first method, Fill, is on the Texture2D class. It takes as a parameter the colour to fill the texture with. First, it creates a new array of type Color that has as dimensions the width times the texture's height. Next, it loops over the collection and sets each element to the fill colour. Finally, it calls the SetData method on the texture to fill it. None of this should be anything new to you. You've seen it all before.

The other method will be new to you. It returns a rectangle grown or shrunk by several pixels. For the X and Y property, you subtract the amount from the X and Y properties. Nothing tricky there. The height and width were problematic and drove me around the bend. It turns out more than adding the values was needed. You need to add twice the value. Which, after thinking about it, made perfect sense. Suppose you have a Rectangle (10, 10, 10, 10) and want to grow it by 1. So, the X and Y become (9, 9). If you increase the width and height by 1, the height and width would be (11, 11). So, the area would cover (9, 9, 10, 10). So, you need to add 2 to expand it from (9, 9, 11, 11).

I adjusted the Control class. I added a new property Offset. This property is much, as you would guess, an offset used in rendering and updating. Add this property to the Control class in the Controls folder in the SharedProject project.

```
        public Vector2 Offset { get; set; }
```

The text box, list box, picture box and form classes went through some serious renovations working through the rendering issues. Since there were several changes, I will give you the code for each class, then go over each in turn. They are all required for the file selection form. So I will tackle them in that order. Replace the TextBox class with this version.

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

```
namespace SharedProject.Controls
{
    public class TextBox : Control
    {
        private readonly Texture2D _background;
        private readonly Texture2D _border;
        private readonly Texture2D _caret;
        private double timer;
        private Color _tint;
        private readonly List<string> validChars = new();
        public bool ReadOnly { get; set; }

        public TextBox(Texture2D background, Texture2D caret, Texture2D brdr)
            : base()
        {
            Text = "";
            _background = background;
            _border = brdr;
            _caret = caret;
            _tint = Color.Black;

            ReadOnly = false;

            foreach (char c in "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRST-
WXYZ0123456789 -_".ToCharArray())
            {
                validChars.Add(c.ToString());
            }
        }

        public override void Draw(SpriteBatch spriteBatch)
        {
            Vector2 dimensions = ControlManager.SpriteFont.MeasureString(Text);
            dimensions.Y = 0;
            spriteBatch.Draw(_border,
                             new Rectangle(Helper.V2P(Position),
Helper.V2P(Size)).Grow(1),
                             Color.White);
            spriteBatch.Draw(_background,
                             new Rectangle(
                                 Helper.V2P(Position),
                                 Helper.V2P(Size)),
                             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,
                Helper.NearestInt(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));
        }
    }
}
```

The first change was the addition of a new field, _border. This is the border of the text box. Of course, it is, Cynthia! I know. Each of the controls has a new _border property. Come to think of it, I had a thought, but I'm not going back and changing it now. I will tackle it in another tutorial. The next addition is a property called ReadOnly. I could have gotten away with just the Enabled property, but

there will be instances in the future will having both will be helpful.
I updated the constructor to take an additional parameter, the text box's border. It sets the field to the value passed in. Also, it sets the ReadOnly property to false.

The last changes were to the Draw method. What I did is call the Draw method passing in the _border texture. I grow it by 1 pixel, making it a full pixel greater than the text box. It then draws the background texture for the text box. This gives the illusion that there is a single-pixel texture around the text box. After drawing the background, I draw the text and caret, cast to the nearest integer.

That is all for the text box. The list box had more changes than the text box. Replace the code of the ListBox class with the following. Sorry for the wall of code that you are about to run into.

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

namespace SharedProject.Controls
{
    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 Texture2D _image;
        private 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(Texture2D background, Texture2D downButton, Texture2D upButton,
Texture2D cursor, Texture2D border)
            : base()
        {
            HasFocus = false;
            TabStop = true;

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

            _upButton.Click += UpButton_Click;
```

```
            _downButton.Click += DownButton_Click;

            this._image = background;
            this._border = border;

            this.Size = new Vector2(_image.Width, _image.Height);
            this._cursor = cursor;

            _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(_image, d, Color.White);

            Point position = Xin.MouseAsPoint;
            Rectangle destination = new(0, 0, Bounds.Width - 40, (int)SpriteFont.Line-
Spacing);
            _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 * Sprite-
Font.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(MouseButtons.Left) && _mouseOver && timer > 0.5)
            {
                timer = 0;
                HasFocus = true;
                OnSelected();
            }

            if (Xin.IsMouseDown(MouseButtons.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
    }
}
```

So, much new stuff, but a lot of old stuff. 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 _image 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. Also, I updated the constructor to require another Texture2D for the border.

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 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 must be overridden in any child classes.

So, that is it for the list box. It is a bit of a beast but is very important in the grand scheme of things. So, next up is the PictureBox. 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;
```

```
namespace SharedProject.Controls
{
    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 des-
tination)
        {
            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 des-
tination, 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
    }
}
```

That brings up to the form. There were a few changes to the Form class. They revolve main around the picture boxes and text Here is the new code for the form class.

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

namespace SharedProject.Controls
{
    public abstract class Form : GameState
    {
        private ControlManager _controls;
        protected readonly GraphicsDeviceManager _graphicsDevice;
        private Point _size;
        private Rectangle _bounds;
        private string _title;

        public string Title { get { return _title; } set { _title = value; } }
        public ControlManager Controls { get => _controls; set => _controls = value; }
```

```
public Point Size { get => _size; set => _size = value; }
public bool FullScreen { get; set; }
public PictureBox Background { get; private set; }
public PictureBox TitleBar { get; private set; }
public Button CloseButton { get; private set; }
public Rectangle Bounds { get => _bounds; protected set => _bounds = value; }
public Vector2 Position { get; set; }

public Form(Game game, Vector2 position, Point size) : base(game)
{
    Enabled = true;
    Visible = true;
    FullScreen = false;
    _size = size;

    Position = position;
    Bounds = new(Point.Zero, Size);
    _graphicsDevice = (GraphicsDeviceManager)Game.Services.GetService(
        typeof(GraphicsDeviceManager));

    Initialize();
    LoadContent();
    Title = "";
}

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

protected override void LoadContent()
{
    base.LoadContent();
    _controls = new((Game.Content.Load<SpriteFont>("Fonts/MainFont")));

    TitleBar = new(
        GraphicsDevice,
        Game.Content.Load<Texture2D>("GUI/TitleBar"),
        new(0, 0, Size.X, 20));

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

    Background.Image.Fill(Color.Wheat);

    CloseButton = new(
        Game.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.Width = _graphicsDevice.PreferredBackBufferWidth;
                Background.Height = _graphicsDevice.PreferredBackBufferHeight;
            }
        }

        private void CloseButton_Click(object sender, EventArgs e)
        {
            StateManager.PopState();
        }

        public override void Update(GameTime gameTime)
        {
            if (Enabled)
            {
                CloseButton.Update(gameTime);
                Controls.Update(gameTime);
            }

            base.Update(gameTime);
        }

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

            if (!Visible) return;

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

                SpriteBatch.Begin(SpriteSortMode.BackToFront, BlendState.AlphaBlend, Sam-
plerState.AnisotropicWrap);

                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();
```

```
                SpriteBatch.Begin(SpriteSortMode.FrontToBack, BlendState.AlphaBlend, Sam-
plerState.AnisotropicWrap);

                _controls.Draw(SpriteBatch);

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

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

                SpriteBatch.End();
            }
        }
    }
}
```

In the LoadContent method, when I create the picture box for the title bar and background, I pass in the GraphicsDevice that is now required. I set the background to fill, and I set the background colour to wheat. It is a bit of an off-white, and it kind of works. Feel free to experiment with other colours. If the form is full screen, I set the height to be the height of the window and the width to be the width of the window.

In the Draw method, when drawing the title, I now use the NearestInt method to cast the position to an integer. I do that so much. I'm going to create extension methods that draw using Points instead of Vector2. It would make life simpler, I think. Well, that will be in a future tutorial. When calling Begin, I no longer specify a transformation matrix. That seemed to be causing issues when rendering text.

Hmmmm. This tutorial is much longer than I expected. However, I think that we can power through the file form and add it to the game. On the other hand, it is a lot of new code. So, I'm torn. I think it is better to power through.

So, let's get started with the file dialog. Right-click the MapEditor project in the Solution Explorer, select Add and then New Folder. Name this new folder Forms. Now, right-click that folder, choose Add and then Class. Name this new class FileForm. Here is the code for that form.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using SharedProject;
using SharedProject.Controls;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace MapEditor.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.Pre-
ferredBackBufferHeight);
            Bounds = new(0, 0, _graphicsDevice.PreferredBackBufferWidth, _graphicsDe-
vice.PreferredBackBufferHeight);
            FullScreen = true;
        }

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

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

            Size = new(_graphicsDevice.PreferredBackBufferWidth, _graphicsDevice.Pre-
ferredBackBufferHeight);
            Bounds = new(0, 0, _graphicsDevice.PreferredBackBufferWidth, _graphicsDe-
vice.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(bg, caret, brdr)
            {
                Position = new(5, 25),
                Size = new(Size.X - 80, 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),
        };

        action.Click += Action_Click;

        Controls.Add(action);

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

        items = new(bg,
                    Game.Content.Load<Texture2D>(@"GUI/DownButton"),
                    Game.Content.Load<Texture2D>(@"GUI/UpButton"),
                    caret,
                    brdr)
        {
            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;
                StateManager.PopState();
```

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

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

            base.Show();
        }

        protected 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. I create the caret and filled it black. Next, I create a background for the text box and fill it in white. The next texture I created is for the border of the text box and fill it in black. I then create the textbox and 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 textures. 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.

There is only one more thing that I'm going to pack into this tutorial. That is creating a file form and rendering it in the editor. I was going to include the message box, but I think you're at about your stamina limit. That, and I want to be done and move on to something new. Replace the MainForm class with the following code.

```
using MapEditor.Forms;
using Microsoft.Xna.Framework;
using SharedProject;
using SharedProject.Controls;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MapEditor.GameStates
{
    public interface IMainForm
    {
        Form Target { get; }
    }

    public class MainForm : Form, IMainForm
    {
        private FileForm fileForm;

        public Form Target => this;

        public MainForm(Game game, Vector2 position, Point size)
            : base(game, position, size)
        {
            Game.Services.AddService<IMainForm>(this);
        }

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

        protected override void LoadContent()
        {
            Point size = new(700, 400);
```

```
            fileForm = new(Game, new(0, 0), size)
            {
                Visible = false,
                Enabled = false
            };

            base.LoadContent();
        }

        public override void Update(GameTime gameTime)
        {
            FullScreen = true;

            if (Xin.WasKeyReleased(Microsoft.Xna.Framework.Input.Keys.F1))
            {
                fileForm.Visible = true;
                fileForm.Enabled = true;
                fileForm.Role = FileFormRole.Save;
                StateManager.PushState(fileForm);
                this.Visible = true;
            }

            if (Xin.WasKeyReleased(Microsoft.Xna.Framework.Input.Keys.F2))
            {
                fileForm.Visible = true;
                fileForm.Enabled = true;
                fileForm.Role = FileFormRole.Open;
                StateManager.PushState(fileForm);
                this.Visible = true;
            }

            base.Update(gameTime);
        }

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

        protected override void Show()
        {
            base.Show();
        }
    }
}
```

The change is the addition of a FileForm field. It is initialized in the LoadContent method. In the Update method, I check if the F1 or F2 keys have been released. If either key has been released, I set the Visible and Enabled property of the file form to true. If the F1 key has been pressed, I give the form the Save role and the Open role for the F2 key. The form is then pushed onto the stack, and its Visible property is set to true.

That was quicker than I expected. I am going to make a couple of quick tweaks to the Editor class. Replace that class with the following code.

```
using MapEditor.GameStates;
using Microsoft.Xna.Framework;
```

```
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using SharedProject;

namespace MapEditor
{
    public class Editor : Game
    {
        private readonly GraphicsDeviceManager _graphics;
        private SpriteBatch _spriteBatch;
        private readonly GameStateManager _manager;

        public MainForm MainForm { get; private set; }

        public Editor()
        {
            _graphics = new(this)
            {
                PreferredBackBufferWidth = 1900,
                PreferredBackBufferHeight = 1000
            };

            _graphics.ApplyChanges();

            Services.AddService<GraphicsDeviceManager>(_graphics);

            _manager = new(this);
            Components.Add(_manager);

            Services.AddService<GameStateManager>(_manager);

            Components.Add(new Xin(this));

            Content.RootDirectory = "Content";
            IsMouseVisible = true;
        }

        protected override void Initialize()
        {
            // TODO: Add your initialization logic here

            base.Initialize();
        }

        protected override void LoadContent()
        {
            _spriteBatch = new SpriteBatch(GraphicsDevice);
            Services.AddService<SpriteBatch>(_spriteBatch);

            // TODO: use this.Content to load your game content here
            MainForm = new(this, Vector2.Zero, new(1900, 1000))
            {
                FullScreen= false,
            };

            _manager.PushState(MainForm);
        }

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

```
            if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed ||
Keyboard.GetState().IsKeyDown(Keys.Escape))
                Exit();

            // TODO: Add your update logic here

            base.Update(gameTime);
        }

        protected override void Draw(GameTime gameTime)
        {
            GraphicsDevice.Clear(Color.CornflowerBlue);

            // TODO: Add your drawing code here

            base.Draw(gameTime);
        }
    }
}
```

All that the new code does is initialize the screen to 1900 by 1000 pixels. I did that because at 1920 by 1080, there were parts that were not visible. Setting it to 1900 by 1000 has everything visible.

So, this has been a monster tutorial, and I am going to end it here. There is more that I could pack in, but I will save it for the following tutorials. I don't want you to 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. Also, I'm thinking of reviving my newsletter so you will be informed of new stuff rather than having to keep looking for new things.

Good luck with your game programming adventures!
Cynthia