A Summoner's Tale
Chapter 6
Graphical User Interface – Part 3

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.

Okay, this will be the last tutorial on graphical user interfaces for a while. What I'm going to cover is building a text box and a file dialog. I will start by adding a text box. For a text box, we need to be able to grab key presses from the user. Unfortunately, this will not work on Android and iOS the same way. For that, we need to hook into the virtual keyboards or create our own. To handle the keyboard input, we need to make some changes to Xin. We need to be able to detect keys that have been pressed this frame. Add the following methods to Xin.

```
public static List<Keys> KeysPressed()
{
    List<Keys> keys = new();

    Keys[] current = keyboardState.GetPressedKeys();
    Keys[] last = lastKeyboardState.GetPressedKeys();

    foreach (Keys key in current)
    {
        if (!last.Contains(key))
        {
            keys.Add(key);
        }
    }

    return keys;
}

public static List<Keys> KeysReleased()
{
    List<Keys> keys = new();

    Keys[] current = keyboardState.GetPressedKeys();
    Keys[] last = lastkeyboardState.GetPressedKeys();

    foreach (Keys key in current)
    {
        if (last.Contains(key))
        {
```

```
                keys.Add(key);
            }
        }

        return keys;
    }
```

The two methods work similarly. The return a List<Keys> for keys that have just been pressed or keys that have just been released. The first step is to create a List<Keys> to hold the keys. Next, I use the GetPressedKeys method of the keyboard to capture all of the pressed keys. I then look over all of the pressed keys in this frame. I check to see if the last keyboard state does or does not contain the key. If it does not contain the key, the key has just been pressed. If it does contain the key, the key has just been released. Finally, the methods return the key collection.

Before I get to the text box, there are a few minor bugs that I want to squash. First, there is some tearing of textures. This is for two reasons. First, I'm using the wrong sampler when sampling the GUI textures. Second, things are being offset by partial pixels. Fixing but of these has the game rendering much nicer. Fortunately, there are only a few places to tweak. To start, open the MessageForm class. Replace the class with the following code.

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

namespace SummonersTale.Forms
{
    public enum CloseReason { OK, Cancel, Yes, No }
    public class MessageForm : Form
    {
        public string Message { get; set; }
        public CloseReason CloseReason { get; set; }

        public MessageForm(Game game, Vector2 position, Point size, string message,
bool auto) : base(game, position, size)
        {
            Size = size;
            Message = message;

            Bounds = new(
                (_graphicsDevice.PreferredBackBufferWidth - size.X) / 2,
                (_graphicsDevice.PreferredBackBufferHeight - size.Y) / 2,
                size.X,
                size.Y);

            Position = position;

            if (auto)
            {
                Position = new(Bounds.X, Bounds.Y);
            }
        }
```

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

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

            Button okay = new(content.Load<Texture2D>("GUI/Button"),
ButtonRole.Accept)
            {
                Text = "OK",
                Color = Color.Black,
            };

            okay.Position = new((Bounds.Width - okay.Width) / 2, Bounds.Height -
okay.Height - 10);
            okay.Offset = Position;

            Message = "Message box!";

            Label label = new()
            {
                Text = Message,
                Position = new(
                    (Bounds.Width -
ControlManager.SpriteFont.MeasureString(Message).X) / 2,
                    (Bounds.Height -
ControlManager.SpriteFont.MeasureString(Message).Y) / 2 - 10),
                Color = Color.Black,
            };

            Controls.Add(label);

            Button cancel = new(content.Load<Texture2D>("GUI/Button"),
ButtonRole.Cancel)
            {
                Text = "Cancel",
                Color = Color.Black
            };

            Controls.Add(okay);

            okay.Click += Okay_Click;
            Background.Position = new(Bounds.X, Bounds.Y);
        }

        private void Okay_Click(object sender, EventArgs e)
        {
            CloseReason = CloseReason.OK;
            manager.PopTopMost();
        }

        public override void Update(GameTime gameTime)
        {
            foreach (Control control in Controls)
            {
```

```
                control.Offset = Position + new Vector2(0, 20);
            }
            if (Xin.WasKeyReleased(Microsoft.Xna.Framework.Input.Keys.Escape))
            {
                manager.PopTopMost();
                CloseReason = CloseReason.Cancel;
            }

            if (Xin.WasKeyReleased(Microsoft.Xna.Framework.Input.Keys.Enter))
            {
                manager.PopTopMost();
                CloseReason = CloseReason.OK;
            }

            base.Update(gameTime);
        }

        public override void Draw(GameTime gameTime)
        {
            Matrix m = Matrix.CreateTranslation(new Vector3(Position, 0));

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

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

            SpriteBatch.End();

            m = Matrix.CreateTranslation(new Vector3(0, 20, 0) + new
Vector3(Position, 0));
            // m = Matrix.Identity;
            SpriteBatch.Begin(SpriteSortMode.FrontToBack, BlendState.AlphaBlend,
SamplerState.AnisotropicWrap, null, null, null, m);

            Controls.Draw(SpriteBatch);

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

The main change is in the Draw method. Instead of PointClamp I use AnisotrophicWrap as the sampler state. Before I get to the other changes, I want to add a helper class that will hold useful methods. Right-click the SummonersTale project, select Add and then Class. Name this new class Helper. Here is the code for the Helper class.

```
using Microsoft.Xna.Framework;

namespace SummonersTale
{
    public static class Helper
    {
        public static Point V2P(Microsoft.Xna.Framework.Vector2 vector)
        {
```

```
            return new Point((int)vector.X, (int)vector.Y);
        }

        public static Vector2 NearestInt(Vector2 v)
        {
            return new Vector2((int)v.X, (int)v.Y);
        }
    }
}
```

Two very similar methods. Indeed, the only difference is the output. One returns a Point, and the other returns a Vector2. Both are returning an object that has the X and Y coordinates of the vector passed rounded to an integer. This resolves some of the tearing that occurs, especially with fonts.

Labels and buttons are two of the objects that experience tearing. We can resolve this using the two new methods to cast coordinates to integers. First, I will fix Labels. Replace the Label class with the following code.

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

namespace SummonersTale.Forms
{
    public class Label : Control
    {
        public Label()
        {
            _visible = true;
            _enabled = true;
            _tabStop = false;
        }

        public override void Draw(SpriteBatch spriteBatch)
        {
            spriteBatch.DrawString(_spriteFont, Text, Helper.NearestInt(Position),
Color);
        }

        public override void HandleInput()
        {
        }

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

As you can see, I now use the NearestInt method to return a vector that has the X and Y coordinates cast as integers. I do the same for the Button class. Replace the Button class with the following code.

```
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input.Touch;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework;
using SummonersTale.Forms;
using System;
using System.Collections.Generic;
using System.Text;
using static System.Net.Mime.MediaTypeNames;

namespace SummonersTale
{
    public enum ButtonRole { Accept, Cancel, Menu }

    public class Button : Control
    {
        #region

        public event EventHandler Click;

        #endregion
        #region Field Region

        private readonly Texture2D _background;
        float _frames;

        public ButtonRole Role { get; set; }
        public int Width { get { return _background.Width; } }
        public int Height { get { return _background.Height; } }

        #endregion

        #region Property Region
        #endregion

        #region Constructor Region

        public Button(Texture2D background, ButtonRole role)
        {
            Role = role;
            _background = background;
        }

        #endregion

        #region Method Region

        public override void Draw(SpriteBatch spriteBatch)
        {
            Rectangle destination = new(
            (int)Position.X,
                (int)Position.Y,
                _background.Width,
                _background.Height);

            spriteBatch.Draw(_background, destination, Color.White);
```

```
            _spriteFont = ControlManager.SpriteFont;

            Vector2 size = _spriteFont.MeasureString(Text);
            Vector2 offset = new((_background.Width - size.X) / 2,
((_background.Height - size.Y) / 2));

            spriteBatch.DrawString(_spriteFont, Text, Helper.NearestInt((Position +
offset)), Color);
        }

        public override void HandleInput()
        {
            MouseState mouse = Mouse.GetState();
            Point position = new(mouse.X, mouse.Y);


            Rectangle destination = new(
                (int)(Position.X + Offset.X),
                (int)(Position.Y + Offset.Y),
                _background.Width,
                _background.Height);

            if ((Role == ButtonRole.Accept && Xin.WasKeyReleased(Keys.Enter)) ||
                (Role == ButtonRole.Accept && Xin.WasKeyReleased(Keys.Space)))
            {
                OnClick();
                return;
            }

            if (Role == ButtonRole.Cancel && Xin.WasKeyReleased(Keys.Escape))
            {
                OnClick();
                return;
            }

            if (Xin.WasMouseReleased(MouseButton.Left) && _frames >= 5)
            {
                if (destination.Contains(position))
                    OnClick();
            }
        }

        private void OnClick()
        {
            Click?.Invoke(this, null);
        }

        public override void Update(GameTime gameTime)
        {
            _frames++;
            HandleInput();
        }

        public void Show()
        {
            _frames = 0;
        }
```

```
        #endregion
    }
}
```

Next, I made some changes to the Form class. I update the draw method to use the items already done in the Draw method. Update the Draw method of the Form class to the following.

```
        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.CreateTranslation(new Vector3(Position, 0));

                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();
            }
        }
```

I also changed the MessageForm to use the same methods as the controls and forms that I shared previously. Update the Draw method of the MessageForm class to the following code.

```
        public override void Draw(GameTime gameTime)
        {
            Matrix m = Matrix.CreateTranslation(new Vector3(Position, 0));

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

            Background.Draw(SpriteBatch);
            TitleBar.Draw(SpriteBatch);
            CloseButton.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();
        }
    }
```

So, now everything is in place for the text box. Right-click the Forms folder in the SummonersTale project, select Add and then Class. Name this new class TextBox. Here is the code for that class.

```
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 _background;
        private readonly Texture2D _caret;
        private double timer;
        private Color _tint;
        private List<string> validChars = new();

        public Textbox(Texture2D background, Texture2D caret)
            : base()
        {
            _text = "";
            _background = background;
            _caret = caret;
            _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;
            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,
                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));
        }
    }
}
```

Like all of our controls, this class inherits from the base control class Control. That means it can be added to a control manager and not have to be worried about calling its methods. I added a few new fields. The first is _background, which is the background. The other fields are _caret which, as the name implies, is the caret for the text box. The timer field controls when the caret blinks. The last, validChars, holds the characters that can be entered into the text box. The constructor takes two parameters. The text box's background and the caret. It sets the text to the empty string to prevent null reference exceptions. It sets the _background and _caret fields to the value passed in. Next, it sets the tint to black.

What I do next is loop over all of the characters the text box will accept. They are the lower case letters, upper case letters, digits, space, dash and underscore. I then add each of these characters to the list of valid characters.

In the Draw method, I get the dimensions of the text in the text box, so I know where to draw the caret. I set the Y coordinate to zero, so it will be set at the height of the text box. Next, I draw the background of the text box by creating a new rectangle using the V2P helper method that I added earlier. When drawing the text, I use the NearestInt method of the Helper class. I pad it five pixels right and down. I just noticed I use the fixed colour black instead of the font colour. I also pad the caret when rendering it.

You could have a whole mess of if statements checking for individual key presses in the HandleInput method. It's a little easier if you check all keys pressed. First, if the control does not have focus, I exit the method. This makes me think. It would be good if the control with the focus was highlighted in some way. I will add that to my to-do list.

The first step is to get the pressed keys in this frame. Then, iterate over the collection of keys. I then used the GetName method of the Enum class. If the key is the backspace key, and there is text in the text box. I shrink the text by one character and exit the method. If it is pressed and there is one character or less, I set the text to the empty string and exit the method.

Next, I handle digits. Digits are the letter D followed by the digit. So, I check to see if the length is do, and the first letter is a D. If it is, value is set to the last digit.

Next, I check to see if the right or left shift keys are down and that the caps lock is off. If all that is true, I set the value to the ToLower value. Now, I check to see if the key pressed is in the list of valid keys. If it is, and the length of the text box is less than the size of the text box, I add the key to the _text field.

That just leave the Update method. What I do is update the timer by three times the amount of time that has passed since the last frame. It's a decent blink rate for the caret. I then calculate the sine of it. I do that because sine waves are cyclical and range between one and minus one. I then multiply the colour black by the round of the absolute value of sine. This has the colour blinking between transparent and opaque.

Let's just drop a text box on the main form to test. Change the LoadContent method of the MainForm class to the following. Also, add the TextBox field.

```
        TextBox _textBox;

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

            _test = new(content.Load<Texture2D>(@"GUI/Button"), ButtonRole.Menu)
            {
                Text = "Click Me",
                Position = new(100, 100),
                Size = new(300, 30),
                Color = Color.Black,
                Visible = true,
                Enabled = true,
                Offset = new(0, TitleBar.Height)
            };
```

```
            _test.Click += Test_Click;
            Controls.Add(_test);

            Color[] buffer = new Color[3 * 20];

            for (int i = 0; i < buffer.Length; i++)
                buffer[i] = Color.Black;

            Texture2D caret = new(GraphicsDevice, 3, 20);
            caret.SetData(buffer);

            _textbox = new(content.Load<Texture2D>(@"GUI/TextBox"), caret)
            {
                HasFocus = true,
                Visible = true,
                Position = new(100, 20),
                Enabled = true,
                Size = new(300, 30),
                Color = Color.Black,
            };
            Controls.Add(_textbox);
        }
```

What's changed? Well, other than the addition of the field, inside the LoadContent method, I create a Color array that is 3 * 20. This is filled with the colour black and will be used to create a Texture2D on the fly. After filling the array, I create a Texture2D that is three pixels made and twenty pixels high. I find that ratio acceptable. If you'd like it thinner or taller, just adjust the values. Once the texture has been created, I use the SetData method to set the pixel. Then, I create the text box using a texture we will add in just a minute and the caret. During creation, I set several of the properties. Enabled, Visible and HasFocus are all set to true. Its position is set to (100, 20) and its size to (300, 30). Finally, the colour is set to black.

You don't have the texture for the text box and possibly the button. I missed the button when I published the last tutorial. I've updated the download to include it. You can download the textures from this link.

If you build and run now, you should be presented with the editor window with a text box and a button. The title bar is a little Windows XP-like. Clicking the button will display the message box. I'm not going to dive into the editor any further in this tutorial. I think I've fed you enough for one day. So, that is going to be it for this tutorial. 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