A Summoner's Tale
Chapter 4
Graphical User Interface – Part 2

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 is going to continue with the graphical user interface. But first, I want to make a couple of changes to the projects. Expand the Dependencies node in the Solution Explorer for the ShadowEdit project. Expand the Projects node under there. Now, right-click the SummonersTaleGame node and select Delete. This will prevent some warnings in the editor. Now, change the code for the BaseGameState to the following.

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

namespace SummonersTale.StateManagement
{
    public class BaseGameState : GameState
    {
        #region Field Region

        protected readonly static Random random = new();
        protected readonly SpriteBatch spriteBatch;

        #endregion

        #region Proptery Region

        public SpriteBatch SpriteBatch
        {
            get { return spriteBatch; }
        }

        #endregion

        #region Constructor Region

        public BaseGameState(Game game)
            : base(game)
        {
```

```
            spriteBatch =
(SpriteBatch)Game.Services.GetService(typeof(SpriteBatch));
        }

        #endregion
    }
}
```

All I did was remove the unneeded using statement for the SummonersTaleGame namespace. Now, I will add another control vital for the map editor, a list box. The list box will hold the tile sets, the levels, and the map layers. They are composite controls made up of two buttons against a picture box. The buttons will scroll the selection up and down. Again, I didn't make the most attractive pieces, but they will be functional.

So, to start, we need to add the content. To do that, you need the content. You can download it using this link. Once you have downloaded and extracted the content, open the Content folder in the SummonersTale project. Double-click the Content.mgcb to open the content builder. Select the root node and click the Add New Folder button. Name the new folder GUI. Select the GUI folder and click the Add Existing File button. Add all of the images from the file that you downloaded and add them to the project. Save the project and close the content builder.

Now, right-click the Forms folder in the SummonersTale project, select Add and then Class. Name this new class, ListBox. Here is the code for that class.

```
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;

namespace SummonersTale.Forms
{
    public class ListBox : Control
    {
        #region Event Region

        public event EventHandler SelectionChanged;
        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 _image;
        private readonly Texture2D _cursor;

        private Color _selectedColor = Color.Red;
```

```
        private int _selectedItem;
        private readonly Button _upButton, _downButton;
        private double timer;

        #endregion

        #region Property Region

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

        public int SelectedIndex
        {
            get { return _selectedItem; }
            set { _selectedItem = (int)MathHelper.Clamp(value, 0f, _items.Count); }
        }

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

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

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

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

        #endregion

        #region Constructor Region

        public ListBox(Texture2D background, Texture2D downButton, Texture2D
upButton, Texture2D cursor)
            : 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.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 > 0.1)
            {
                timer = 0;
                _selectedItem++;
                OnSelectionChanged(null);
            }
        }

        private void UpButton_Click(object sender, EventArgs e)
        {
            if (_selectedItem > 0 && timer > 0.1)
            {
                timer = 0;
                _selectedItem--;
                OnSelectionChanged(null);
            }
        }

        #endregion

        #region Abstract Method Region

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

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

        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(_image, d, Color.White);
            Point position = Xin.MouseAsPoint;
            Rectangle destination = new(0, 0, 100, (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)
                {
                    _mouseOver = true;
                    _selectedItem = _startItem + i;
                    OnSelectionChanged(null);
                }

                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)
                {
                    spriteBatch.DrawString(
                        SpriteFont,
                        text,
                        new Vector2(Position.X + 3, Position.Y + i *
SpriteFont.LineSpacing + 2),
                        SelectedColor);
                }
                else
                {
                    spriteBatch.DrawString(
                        SpriteFont,
                        text,
                        new Vector2(Position.X + 3, Position.Y + i *
SpriteFont.LineSpacing + 2),
                        Color);
                }
            }

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

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

        public override void HandleInput()
        {
            //if (_upButton.ContainsMouse(Xin.MouseAsPoint))
            //{
            //    _upButton.HandleInput();
```

```
        //}

        //if (_downButton.ContainsMouse(Xin.MouseAsPoint))
        //{
        //    _downButton.HandleInput();
        //}
        if (!HasFocus)
        {
            return;
        }

        if (Xin.WasKeyReleased(Keys.Down))
        {
            if (_selectedItem < _items.Count - 1)
            {
                _selectedItem++;

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

                OnSelectionChanged(null);
            }
        }
        else if (Xin.WasKeyReleased(Keys.Up))
        {
            if (_selectedItem > 0)
            {
                _selectedItem--;

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

                OnSelectionChanged(null);
            }
        }
        if (Xin.WasMouseReleased(MouseButton.Left) && _mouseOver)
        {
            HasFocus = true;
            OnSelectionChanged(null);
        }
        if (Xin.WasKeyReleased(Keys.Enter))
        {
            HasFocus = false;
            OnSelected(null);
        }

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

    #endregion
```

```
        #region Method Region

        protected virtual void OnSelectionChanged(EventArgs e)
        {
            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
    }
}
```

There are three events for this class: SelectionChanged, Enter and Leave. The SelectionChanged event will be triggered when the selected item changes. Enter will be triggered when the control gets focus and Leave will be triggered when the focus leaves the control. There is a readonly List<string> which is the text of the items. You could use a List<object> and cast the items to a string when rendering them. For our purposes, a List<string> is good enough. There are two fields: _startItem and _lineCount which are used in rendering the list items. More on them later. There is a field, _selectedColor, that controls how the selected item is drawn. There is a field, _selectedItem, that holds which item in the list box is currently selected. The _upButton and _downButton fields are buttons that move the selected item up and down. That last field, timer, counts the amount of time that has passed for the control. It is used because I found that MonoGame can sometimes stutter when it comes to checking if an item was down in one frame and up in the previous frame. So, I use a timer to measure the amount of time that has passed since the event has occurred.

There are properties to expose the _selectedColor, _selectedItem, and value of the selected item. There is also an override of the HasFocus property. The interesting properties are SelectedIndex and HasFocus. The set part of SelectedIndex uses the Clamp method of MathHelper class to ensure that the value is in the range of the bounds of the array. The HasFocus property sets the _hasFocus property and then calls the OnEnter method if the control has received focus and OnLeave if the control has lost focus.

The constructor takes four parameters, the background of the list box, the texture for the down button, the texture for the up button and the texture for the selected item. The _hasFocus field is set to false and the _tabStop property is set to true. I then create the buttons passing in the texture for the button and setting the role to Menu. I also set the Text property to the empty string. Otherwise, you will get a null value exception. I then wire the Click event for the buttons. I set the _image field to the background passed in and the initial size to be the size of the image. The _cursor field is set the parameter passed in. Finally, I set the _startItem field to zero and the colour to draw the text to black.

In the event handler for the down button, I check to see if the _selectedItem property is not set to the last item in the list of items. Also, I check to see that more than a tenth of a second has passed. This

prevents the event from being fired multiple times. If these conditions are true, I set the timer to zero, increment the selected item and call the OnSelectChanged method, which I will discuss shortly. The event hander for the up butting works in much the same way. The difference is that it checks to see if the _selectedItem is zero. If that is true and the timer is greater than a tenth of a second, the selected item is decremented. It then calls the OnSelectionChanged event handler.

In the Update method, I increment the timer field with the amount of time that has passed since the last frame of the game. Remember that in total, one second corresponds to the value one. So, one-tenth of a second is 0.1, and one full second is represented by one. It then calls the Update method of the buttons.

In the Draw method, I set the _lineCount field to the number of lines that can fit in the size of the list box. That is calculated by taking the height of the control and dividing that by the LineSpacing property of the font and casing that to an integer. I then create a rectangle that will be where the control will be drawn. It is calculated by taking the Position property of the control and the Size property of the control. It then draws the background image. Next, I used a new property, MouseAsPoint of the Xin class, that returns the coordinates of the mouse as a point. I will add this property, and another, once I have finished with the list box. I then create a destination rectangle that is positioned at (0, 0) and is 100 pixels wide and the line spacing property of the font high. Next, it sets the _mouseOver property of the list box to false.

After that, there is a for loop that loops for the number of line items there are. I then check to see if the item at the top of the list box plus i is greater than or equal to the number of items. If that is true, I break out of the loop. The X coordinate of the destination is set the X coordinate of the position. The Y coordinate of the destination is set to the Y coordinate of the position plus i times the line spacing property of the font. If the destination contains the position of the mouse and the left mouse button is down, _mouseOver is set to true, and the _selectedItem is set to the _startItem plus the value of i. The OnSelectionChanged method is called.

Next, there are some local variables. The first, length, measures the length of a substring of the current item being drawn. It is used to prevent the line from overflowing the list box. The j variable is the number of characters that are currently being checked. Finally, the text variable is the substring of the current item. There is a while loop that will loop for the number of characters that can safely be drawn. That is calculated by comparing the length of the string to the size of the list box minus the width of the buttons. The first step is to increment the number of characters to be drawn. After that, I use the MeasureString method passing in a substring j characters long. I also set the text variable to that substring.

About the _startItem field. It is set to the first item drawn in the list box. While all the items in the list box can safely be drawn and not overflow the list box's height, it will remain zero. Once there are more items in the list box than items in the list, it will change. When you scroll down, it is incremented. When you scroll up, it is decremented.

Next, there is an if statement that checks to see if the _startItem plus i is the selected item. If it is, that item is drawn using the SelectedColor property. Otherwise, it is drawn in the colour of the control. I added a smidge of padding to render the item. The X coordinate of the up button is set to the X coordinate of the list box plus the width of the list box minus the width of the button. The Y coordinate of the button is the Y coordinate of the list box. The X coordinate of the down button is the same as the X coordinate of the up button. The Y coordinate of the down button is the Y coordinate of the list box

plus the height of the list box minus the height of the button. After setting the position of the buttons, I call their Draw methods.

The HandleInput method checks to see if the control has focus. If it does not, the method is exited. In addition to using the buttons, the list box can be scrolled using the Up and Down keys. So, there is a check to see if the Down key is currently down. If it is, and the selected item is not the last item in the list, I increment the selected item field. Also, there is a check to see I the selected item is greater than or equal to the start item field plus the line count. If it is, I scroll down one item. I then call the OnSelectionChanged method. I do something similar if the up key has been released. If the selected item is greater than zero, I decrement the selected item. If the selected item is less than the start item I decrement that as well. I then call the OnSelectionChanged method.

After drawing everything, there are three if statements. The first checks to see if the left mouse button was released and if the mouse is over the list box. If they are both true, the list box gains focus and the OnSelectChanged method is called. If the Enter key is released, the OnSelected method is called. Finally, if the escape key has been released, the list box loses focus, and the OnLeave method is called.

Finally there are three methods: OnSelectionChanged, OnEnter and OnLeave. The first raises the SelectionChanged event if it is subscribed to. The OnEnter method fires the Enter event if it is subscribed to. Finally, OnLeave fires the Leave event if it is subscribed to.

That is it for the list box. It is a key component to the building of the editors. Before I get to the next part, I want to add a couple of properties to Xin. They return the current position of the mouse as a Point and the last position of the mouse as a point. Add these properties to Xin.

```
public static Point MouseAsPoint
{
    get { return new Point(MouseState.X, MouseState.Y); }
}

public static Point LastMouseAsPoint
{
    get { return new Point(LastMouseState.X, LastMouseState.Y); }
}
```

Okay, I need to make a quick adjustment to the Control class. I need to add a property. This property will be use shortly. Replace to Control class with the following version.

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

namespace SummonersTale.Forms
{
    public abstract class Control
    {
        #region Field Region

        protected string _name;
        protected string _text;
        protected Vector2 _size;
```

```
        protected Vector2 _position;
        protected object _value;
        protected bool _hasFocus;
        protected bool _enabled;
        protected bool _visible;
        protected bool _tabStop;
        protected SpriteFont _spriteFont;
        protected Color _color;
        protected string _type;
        protected bool _mouseOver;
        protected Vector2 _offset;

        #endregion

        #region Event Region

        public event EventHandler Selected;

        #endregion

        #region Property Region

        public string Name
        {
            get { return _name; }
            set { _name = value; }
        }

        public string Text
        {
            get { return _text; }
            set { _text = value; }
        }

        public Vector2 Size
        {
            get { return _size; }
            set { _size = value; }
        }

        public Vector2 Position
        {
            get { return _position; }
            set
            {
                _position = value;
                _position.Y = (int)_position.Y;
            }
        }

        public object Value
        {
            get { return _value; }
            set { this._value = value; }
        }

        public virtual bool HasFocus
        {
```

```
        get { return _hasFocus; }
        set { _hasFocus = value; }
    }

    public bool Enabled
    {
        get { return _enabled; }
        set { _enabled = value; }
    }

    public bool Visible
    {
        get { return _visible; }
        set { _visible = value; }
    }

    public bool TabStop
    {
        get { return _tabStop; }
        set { _tabStop = value; }
    }

    public SpriteFont SpriteFont
    {
        get { return _spriteFont; }
        set { _spriteFont = value; }
    }

    public Color Color
    {
        get { return _color; }
        set { _color = value; }
    }

    public string Type
    {
        get { return _type; }
        set { _type = value; }
    }

    public Vector2 Offset { get { return _offset; } set { _offset = value; } }

    #endregion

    #region Constructor Region

    public Control()
    {
        Color = Color.White;
        Enabled = true;
        Visible = true;
        SpriteFont = ControlManager.SpriteFont;
        _mouseOver = false;
        Offset = Vector2.Zero;
    }

    #endregion
```

```
        #region Abstract Methods

        public abstract void Update(GameTime gameTime);
        public abstract void Draw(SpriteBatch spriteBatch);
        public abstract void HandleInput();

        #endregion

        #region Virtual Methods

        protected virtual void OnSelected(EventArgs e)
        {
            Selected?.Invoke(this, e);
        }

        #endregion
    }
}
```

So, I added a new field _offset that measures the offset of the control from the baseline, (0, 0). I also added a property to expose the value. The last think that I did was initialize it to the zero vector.

We have some controls and a control manager. What we really need is a form. A form will be a container that has a control manager. It will have a title bar, a background window and a close button. It will inherit from the BaseGameState method so it can be used with the state manager. I won't be designing a GUI builder like Windows Forms. Controls will have to be placed manually. Right-click the Forms folder in the SummonersTale project, select Add and then Class. Name this new class Form. This is the code for the Form class.

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using SummonersTale.StateManagement;
using System;
using System.Collections.Generic;
using System.Reflection.Metadata.Ecma335;
using System.Text;

namespace SummonersTale.Forms
{
    public abstract class Form : BaseGameState
    {
        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(content.Load<SpriteFont>("Fonts/MainFont"), 100);

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

            Background = new(
                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;
            }
        }

        private void CloseButton_Click(object sender, EventArgs e)
        {
            manager.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 = 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.PointClamp, null, null, null, m);

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

                CloseButton.Draw(SpriteBatch);

                SpriteBatch.End();

                SpriteBatch.Begin();

                label.Position = position + Position;
                label.Draw(SpriteBatch);

                SpriteBatch.End();
```

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

                SpriteBatch.Begin(SpriteSortMode.FrontToBack, BlendState.AlphaBlend,
SamplerState.PointClamp, 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();
            }
        }
    }
}
```

This class inherits from BaseGameState, so it can be used with the state manager. There is a private, might want to make it protected at some point, a field for a control manager. There is a readonly GraphicsDeviceManager field. We need this to get the width and height of the window. There is a Point _size that is the size of the window and a Rectangle _bounds that is the bounds of the window. There is a bit of overlap there. I guess I could have gotten away with just the rectangle. A form also has a title, so there is a field for that. There are properties to expose the title, control manager, Size and if the form should be full-screen or windowed. There are three controls next two picture boxes and a button. The picture boxes hold the title bar and the background. The button is to close the form. There is also a property for the bounds of the form and a Vector2 for the position of the form. There is an overlap, but for reason. I will get to it when I go over rending the form.

The form takes as parameters: a Game object, a Vector2 for its position, and a Point that defines its size. A form is initialized to Enabled and Visible by default. It is windowed, not fullscreen., by default. The constructor then sets the _size field to the size passed in. The Position property is set as the position argument that is passed as well. The Bounds property is set to a rectangle with X and Y coordinates of 0 and the width and height equal to the size argument. I need a GraphicsDeviceManager, so in the game and editor, I will register the GraphicsDeviceManager as a service, and I retrieve it in the base form class. I explicitly call the Initialize and LoadContent methods rather than wait for them to be called.

I really didn't need the Initialize method, but I included it in case it might be required in the future. In the LoadContent method, I call the LoadContent method of the base class. I initialize the control manager passing in the main font that was added in a previous tutorial. I create the title bar next. Its width is set to the width of the form, and its height is set to twenty pixels. The background is created next. It's width is set to the width of the Bounds, and its height is set to the height of the bounds. I set its fill method to fill. Next, I create the close button. I position it in the upper left-hand corner like Apple. Yes, I am an

Apple fan girl now. I have an iPhone, iPad, Apple Watch and MacBook. I don't see myself giving up my PC any time soon for a MacStudio. I set the colour to white so there won't be any tinting and the text to the empty string. I also wire the click event handler for the close button. If this is a full-screen window, I set the height of the title bar to zero and the height of the background to the full height of the window.

The event handler for the close button pops the state off of the stack. In the Update method, if the form is Enabled, the Update method of the close button and the control manager is called.

Now, the reason why there is a position and bounds for the form. Forms will seldom have their upper left-hand corner in the upper left-hand corner of the screen. It could be a problem handling that, especially if there are a lot of controls. Enter the solution: translation matrices. What we do is place everything in the upper left-hand corner of the window and then use a translation matrix in rendering to have MonoGame have MonoGame move everything for us. This is also why I added the Offset property to controls. In order to have the click in the right spot, you need to offset the control.

First, the Draw method of the base class is called. Then, if the form is not visible, I exit the method. If the window is not full-screen, I use the sprite font of the control manager to measure the Title property of the form. I then center the title horizontally. I create a label on the fly and set its position to the value just computed, the colour to white, and the text to the _title field. Next, I create the translation matrix passing in the Position property of the window.

I then call Begin on the sprite batch object to begin rendering. I have the sort order back to front, the blend state to alpha blend, the sampler state to PointClamp, null for the next three parameters and finally our matrix. I call the Draw method of the form elements, not the controls on the form. Separately, I call the Draw method of the title after setting its position. I create a new matrix next, adding twenty pixels for the width of the title bar. I then call the Draw method of the control manager.

If the window is full-screen, I just call Begin on the sprite batch instead of worrying about offsets because everything is relative to the upper left-hand corner. Since there is no need to draw anything but the window background, I don't. I set the destination rectangle of the background to fill the entire screen. I then call the Draw method of the control manager.

I need to make a change to the button class to handle offsets correctly. What I need to do is add the offset of the window to the position of the control when checking if the mouse is in the button. Update the HandleInput method of the Button class to the following.

```
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();
    }
}
```

Okay, I was going to end this tutorial here. Then, I thought it would be nice to see forms in action! So, bare with me a little longer. Right-click the Forms folder in the SummonersTale project, select Add and then Class. Name this new class MainForm. Here is the code for that class.

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

namespace SummonersTale.Forms
{
    public class MainForm : Form
    {
        private Button _test;
        public MainForm(Game game, Vector2 position, Point size) : base(game,
position, size)
        {
            Title = "";
        }

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

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

            _test = new(content.Load<Texture2D>(@"GUI/Button"), ButtonRole.Menu)
            {
                Text = "Click Me",
                Position = new Vector2(100, 100),
                Size = new Vector2(300, 30),
                Color = Color.Black,
                Visible = true,
                Enabled = true,
                Offset = new Vector2(0, TitleBar.Height)
```

```
            };

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

        private void Test_Click(object sender, EventArgs e)
        {
            MessageForm frm = new(Game, new(500, 500), new(300, 100), "Message
box!", true);
            manager.PushTopMost(frm);
            Visible = true;
            Enabled = false;
        }

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

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

Short and sweet, just like a form should be. It just adds a single Button object. The Title property is initialized to the empty string. The Initialize method is currently empty. In the LoadContent method, I create a new button using the button texture that we added at the start of the tutorial. I set the role of the button to ButtonRole.Menu. I initialize the Text property to "Click Me," the Position to (100, 100), the size to (300, 30), the Color property to black, enabled and visible to true, and the offset to (0, 20). I wire the Click event handler and add it to the control manager.

In the Click event handler, I create another form, a message box, passing in the Game property of the form, (500, 500) for the position, (300, 100) for the size, text "Message Box!", and true for the last parameter which is if the form should automatically be positioned. The form is then pushed on as topmost. The Visible property of the main form is set to true, and Enable property is set to false. The other methods call the base method.

Okay, almost done. There are just two things left to do. The first is to implement the message box that I referenced just above. The second is to implement all of the changes in the editor. So, right-click the Forms folder in the SummonersTale project, select Add and then Class. Name this new class MessageForm. Here is the code for that class.

```
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.PointClamp, 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.PointClamp, null, null, null, m);

            Controls.Draw(SpriteBatch);

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

A little bit longer than the other form. A lot of it you have seen previously, so I will gloss over the small changes. This class is going to be extended in the future, so I included an enumeration that will be the way in which the form was closed, using an OK button, Cancel button, Yes button or No button. The class inherits from Form. It will have a message, so there is a Message property. There is also a CloseReason property.

The constructor takes as extra parameters a message and an auto parameter. If the auto parameter is true, the form will be centred on the screen. The Size property is initialized to the size argument. The Message property is set to the message passed in. The Bounds are calculated by centering the form horizontally and vertically, setting the width and height to the size passed in. The Position property is set to the position passed in. If the auto argument was true, the Position property is set to the X and Y coordinated of the Bounds. The Initialize method was included for completeness.

In the LoadContent method, I create a button object with the role Button.Accept. I set its Text property to OK and Color property to black because the buttons are white. I center the button horizontally in the bounds of the form, and it is just above the bottom of the form. I set its Offset property to the position of the form. I set the Message property of the form to "Message box!". I then create a Label, centring it in the form but up ten pixels. I then add the label to the control manager. I create a Cancel button, but I don't do anything with it, yet. I added the okay button to the list of controls and wired its click event handler. I set the background position to the bounds of the window. In the event handler for the Okay button's Click event, I set the CloseReason to OK and call PopTopMost to pop it off the stack. In theory, you should be fine using PopState. It is safer to use the opposite of what you used to push the state on the stack.

In the Update method, I loop over all of the controls in the control manager and set their Offset property to the position of the form. This will become important when I add being able to drag forms. I then checked to see if the Escape key was released. If it was, I pop the state off the stack and set the close reason to cancel. I do something similar for the Enter key.

In the Draw method, I create a translation matrix using the position of the form. Like before, I call the overload of Begin that takes seven arguments. I draw the background, title bar, and close button and call the End method. I create another translation matrix to move things down another twenty pixels. I call the Begin method again with the same arguments. I then call the Draw method of the control manager. Finally, I call the End method to stop rendering.

We're almost at the end zone. There are two things left that I want to do. The first is to make the editor that active project. So, right-click the SummonersTaleEditor in the Solution Explorer, and select Set As Startup. The other is to update the editor to render a main form. Replace the code of the Editor class with the following.

```
using Microsoft.Xna.Framework;
```

```
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using SummonersTale;
using SummonersTale.Forms;
using SummonersTale.StateManagement;

namespace SummonersTaleEditor
{
    public class Editor : Game
    {
        private readonly GraphicsDeviceManager _graphics;
        private SpriteBatch _spriteBatch;
        private readonly GameStateManager manager;
        private MainForm _mainForm;

        public Editor()
        {
            _graphics = new GraphicsDeviceManager(this)
            {
                PreferredBackBufferWidth = 1920,
                PreferredBackBufferHeight = 1080,
                IsFullScreen = false,
                SynchronizeWithVerticalRetrace = false
            };

            _graphics.ApplyChanges();

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

            Services.AddService(typeof(GraphicsDeviceManager), _graphics);

            manager = new GameStateManager(this);
            Components.Add(manager);
        }

        protected override void Initialize()
        {
            // TODO: Add your initialization logic here
            Components.Add(new FramesPerSecond(this));
            Components.Add(new Xin(this));

            base.Initialize();
        }

        protected override void LoadContent()
        {
            _spriteBatch = new SpriteBatch(GraphicsDevice);

            // TODO: use this.Content to load your game content here

            Services.AddService(typeof(SpriteBatch), _spriteBatch);

            _mainForm = new(this, Vector2.Zero, new(1920, 1080))
            {
                FullScreen = false,
                Title = "A Summoner's Tale Editor"
            };
```

```
            manager.ChangeState(_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);
        }
    }
}
```

The first thing is I added a GameStateManager field to the class. I also added a MainForm to the class. In the constructor, I set the width and height of the window to 1920 by 1080. I think it is safe to assume that my readers have at least an HD monitor. If that is bigger than you have, enter your values here. I'm keeping the editor windowed, for now, but I set it to have no fixed frame rate. I then apply the changes. I add the graphics device manager to the services. I then create the GameStateManager and add it to the list of game components. In the Initialize method, I add a new instance of the frames per second component and the Xin component.

In the LoadContent method, I add the SpriteBatch object to the list of game services. I then create the instance of the MainForm having the width and height of the editor. I set its FullScreen property to false and the title to "A Summoner's Tale Editor."

If you build and run now, you should be presented with the editor window with a single 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